Maximum Reconfiguration of 2–D Mesh Systems with Faults

Nian-Feng Tzeng and Guanghua Lin

Center for Advanced Computer Studies
University of Southwestern Louisiana
Lafayette, Louisiana 70504

E-mail: tzeng@cacs.usl.edu

Abstract — *It is desirable to reconfigure a faulty system in a way that the system size after reconfiguration is maximized while reconfiguration hardware overhead is kept low. This paper proposes a maximum reconfiguration scheme for 2–D mesh systems by retaining as many fault-free nodes in reconfigured subsystems as possible. The basic idea is to reconfigure a faulty mesh system into a maximum convex subsystem, using the fault-free upper or lower boundary nodes to compensate for the internal (i.e., non-boundary) faulty nodes. To this end, our reconfiguration problem is transformed into the well-known min-cost flow problem, for which an efficient polynomial algorithm is introduced. Our reconfiguration requires a channel width of two only and is shown by simulation to exhibit much better utilization of fault-free nodes than other schemes.*

1. Introduction

Mesh-based machines have received increasing attention. For a large mesh system, the probability of fault occurrence can be high, making it necessary to consider the reconfiguration of a faulty system. As system performance tends to be proportional to the system size, a reconfigured system involving more nodes is likely to retain performance better. This study focuses on reconfiguring 2–D meshes in an attempt to keep as many fault-free nodes as possible.

Reconfiguration in a 2–D mesh structure has been investigated extensively [1–8]. In particular, reconfiguration schemes in [1] and [7] follow the maximal matching algorithm to map logical cells to fault-free cells. If complete matching is not possible, an additional process (namely, *assignment borrowing* in [1] or *rule-based assignment* in [7]) is then evoked to assign an available fault-free processor to an unmatched logical cell. These two schemes need three tracks along every channel for reconfiguration. Mesh array processors with two-level redundancy are introduced in [2], where a processor array is partitioned into identical blocks with a fixed number of spares assigned to each block as local redundancy and spare blocks added as global redundancy. This design with block size $m \times m$ requires a channel width of m tracks between two adjacent blocks for reconfiguration. The fault-tolerant mesh architecture in [5] adds spare nodes (i.e., processors) and links in a way that the identified subsystem (following graph renaming) after faults occur is guaranteed to contain the desired mesh. For this architecture to tolerate k faults, the node degree in 2–D meshes is $k+4$, making it feasible only for a small k in practice. It also requires a considerably large channel width for reconfiguration to yield short connection wires [5]. Reconfiguration in [6] is achieved by replacing a failed processor with a spare through a compensation path. Spares are those processors in the boundary rows/columns of the mesh, and they are not used for regular operations until they are reconfigured into the system after failures occur. This scheme also requires three tracks along every channel for reconfiguration. The reconfiguration procedure considered in [8] involves local and global reconfiguration, where local reconfiguration employs a typical fault-stealing strategy [4] to replace a failed node with a spare (on the perimeter of the 2–D mesh) and global reconfiguration is invoked to identify a healthy region of specific shape through pattern growth after local reconfiguration fails.

All prior reconfiguration techniques require that the target subsystems be rectangular or square. This tends to result in many fault-free nodes being discarded in order to ensure that a reconfigured subsystem is rectangular or square. Consequently, these techniques often yield unnecessarily low utilization of fault-free nodes in the system (where utilization is the ratio of the number of nodes in the reconfigured subsystem to the number of fault-free nodes in the physical system).

In this paper, we introduce a mesh reconfiguration scheme that can retain *as many* fault-free nodes as possible after reconfiguration, promising *maximum reconfiguration* which exhibits near-optimal utilization of fault-free nodes. This way of maximum reconfiguration is highly desirable as a mesh system so reconfigured will retain its computational power better than a smaller subsystem derived according to prior reconfiguration approaches

This work was supported in part by the NSF under Grant CCR-9300075 and by the State of Louisiana under Contract LEQSF(1994–96)-RD-A-39.

0190-3918/96 $5.00 © 1996 IEEE

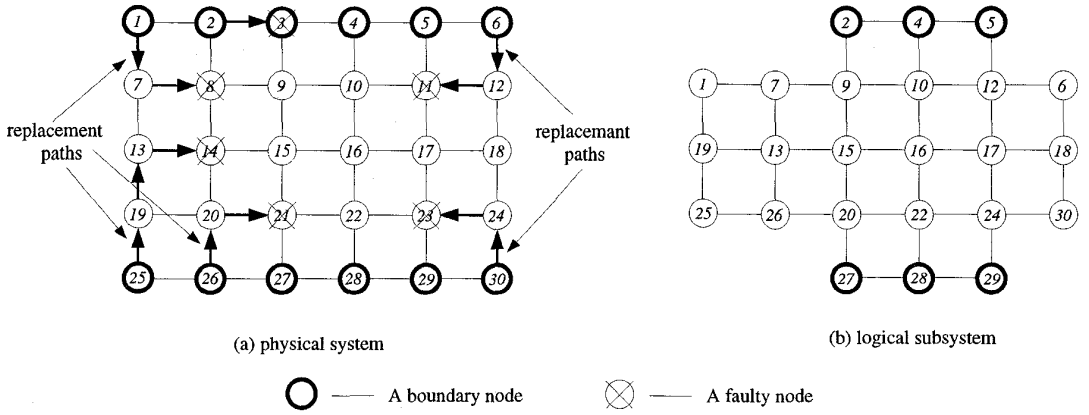


Fig. 1. An example of the reconfiguration in a 5× 6 mesh system.

(which often discard healthy nodes unnecessarily). Our reconfigured 2–D mesh system is no longer rectangular or square, but is ensured to have a *convex*† shape. A convex 2–D network presents a simple and deadlock-free routing algorithm which always delivers messages over shortest paths [9], an essential property of distributed systems. If a faulty system is not reconfigured and all healthy nodes are retained, it is not possible to derive a simple and deadlock-free routing method, possibly causing congested points to arise. As a result, reconfiguring the mesh with faults is more beneficial in practice than maintaining all the healthy nodes without reconfiguration.

The basic idea of our scheme is to reconfigure a faulty mesh system into a convex subsystem by using the fault-free upper or lower *boundary* nodes to compensate for the *internal* (i.e., non-boundary) faulty nodes via a set of non-intersecting paths, called *replacement paths*. Note that a boundary node plays the same role as an internal node, but only internal faulty nodes need to be replaced. If a boundary node becomes faulty, it is either discarded (without replacement) or converted into a connecting element. It will be shown that our scheme requires only two tracks along every channel (or equivalently, a channel width of two) for reconfiguration. While we choose the upper or lower boundary nodes to compensate internal faulty nodes in subsequent discussion, it is obvious that our introduced scheme is equally applicable to mesh systems with internal faulty nodes compensated by the left or right boundary nodes.

An example reconfiguration for a 5×6 mesh system is given in Fig. 1, where the boundary nodes are the bold ones, the faulty nodes are marked by "×", and the replacement paths are those bold, directed paths, each connecting an internal faulty node to a distinct fault-free boundary node. To obtain the reconfigured subsystem, each node on a replacement path is *logically* shifted along the path one position toward the respective (internal) faulty node. In other words, each node on a path is functionally replaced by the node immediately behind it. In Fig. 1, for example, node 8 (a faulty node) is functionally replaced by node 7, which in turn is replaced by node 1. As a result, three nodes 27, 28, and 29 remain logically in the lower boundary row. These nodes are consecutive and thus can be included in the subsystem being constructed. As for the upper boundary row, nodes 2, 4 and 5 are also regarded as *logically consecutive* since node 2 connects node 4 through faulty node 3 (recall that a faulty node can be converted into a connecting element). Therefore, these three nodes can be a part of the subsystem as well. The resulting convex subsystem is given in Fig. 1(b).

In this example, the replacement paths are so determined that each internal faulty node is connected to either a fault-free corner node (where nodes 1, 6, 25, and 30 in Fig. 1(a) are the four corner nodes), or a fault-free boundary node that is as close to a corner node as possible when no corner node is available. This way of determining replacement paths guarantees that the reconfiguration scheme produces a convex subsystem and achieves maximum utilization of fault-free nodes (to be explained in Section 3). We shall adopt the method described above for determining the replacement paths in subsequent discussion.

The rest of this paper is organized as follows. Section 2 shows how the reconfiguration problem is reduced to a *min-cost* flow problem. An efficient algorithm for the min-cost flow problem is presented in Section 3. Section 4 discusses the implementation issues of our reconfiguration. Section 5 gives the simulation results.

† Given the set $R = \{R_{i,j}\}$ of routing relations of non-faulty network N, a set of survived nodes, $S \subseteq N$, is **convex** under R, if for every pair of nodes, $u, v \in S$, there exists *at least one legal route* leading from u to v that is generated by R [9].

2. Problem Definitions

This work is based on the following assumptions: (1) operational faults happen randomly to nodes in the mesh system and the faults are permanent; (2) the system has been tested on-line by built-in testing circuitry or off-line, with diagnostic information available for reconfiguration; (3) reconfiguration is carried out off-line by an external controller; (4) a faulty node can be converted into a connecting element, as commonly found in earlier work [6]; and (5) an $m \times n$ mesh, where m is the number of rows and n is the number of columns, contains no more than $2n$ faults. The problem of our interest is to reconfigure a faulty mesh system into a *maximum* convex subsystem (which retains as many fault-free nodes as possible). This problem can be formally stated as follow.

***Problem 1**: Given an $m \times n$ mesh system with a set of faulty nodes, F (where $|F| \leq 2n$), determine a set of non-intersecting replacement paths such that the reconfigured subsystem is convex and with the maximum size.*

Our reconfiguration thus achieves near-optimal utilization of fault-free nodes. To solve this problem, we shall first reduce it to the following min-cost flow problem, where the value of an s-t flow (called a flow value) is the summation of the costs of all paths from node s to node t.

***Problem 2**: Let $N = (V, E, b, k)$ be a flow network with an underlying directed graph $G = (V, E)$, a capacity $k(u) \in R^+$ on each node $u \in V$, a cost $c(u, v) \in R^+$ and a capacity $b(u, v) \in R^+$ on each edge $(u, v) \in E$, and a flow value $|F|$. Let s and t be two distinguished nodes of V, namely, the source node and the sink node. The min-cost flow problem is to find a feasible s-t flow of value $|F|$ that has a minimum cost, subject to both node capacity $k(u)$ and edge capacity $b(u, v)$ constraints.*

The desired flow network $N = (V, E, b, k)$ can be constructed from a given faulty mesh according to the subsequent rules, which are followed by explanation as to why edge costs are so assigned.

1) V consists of the set of nodes in the mesh, a source node s, and a sink node t.

2) E consists of the following three types of edges:

   I) For every pair of adjacent nodes u and v in the mesh, two edges (u, v), $(v, u) \in E$;

   II) For each *fault-free* boundary node $u \in V$, $(u, t) \in E$;

   III) For each *internal* faulty node $u \in F$, $(s, u) \in E$.

3) $k(u)$ is unity, for each node $u \in V - \{s, t\}$; $k(s)$, $k(t)$ are assumed to be infinity.

4) $b(u, v)$ is unity, for each edge $(u, v) \in E$.

5) The costs on edges in E are assigned as follows:

   I) The cost on each edge connecting two boundary nodes is of two units;

   II) If u is the i^{th} $(1 \leq i \leq n)$ boundary node, where the left-most one is referred to as the 1^{st} boundary node, then the cost on edge (u, t) is of $\min(i, n-i) * L$ units, where L is the upper bound on the cost of a path connecting a faulty node to a boundary node (note that L can be obtained simply by taking summation of costs on edges resided on any two adjacent sides of the flow network (see Fig. 2));

   III) The cost on each of the remaining edges is unity.

The reconfiguration problem in [6] was transformed into a max-flow problem with node and edge capacity constraints. Our min-cost flow problem, however, is slightly more complex than the max-flow problem considered in [6] because the latter involves no cost constraint.

Fig. 2 shows the flow network constructed from Fig. 1(a). For simplicity, only the non-unity costs are given in Fig. 2. Note that a faulty boundary node in the network is connected neither to s nor to t; it is treated just like an internal fault-free node in the flow network. The cost assignments of 5.I and 5.III are to make sure that a flow path from s to t goes through as few boundary nodes as possible; in other words, they are so assigned to ensure that every internal faulty node in the network is connected to a distinct boundary node, which is as close to a corner node as possible. In Fig. 2, for example, path $s \rightarrow 14 \rightarrow 13 \rightarrow 19 \rightarrow 25 \rightarrow t$ is chosen, instead of $s \rightarrow 14 \rightarrow 20 \rightarrow 26 \rightarrow 25 \rightarrow t$. The cost assignment of 5.II indicates that the cost of an s-t flow going through a boundary node i is always smaller than that of an s-t flow going through a different boundary node j, if i is closer to a corner node than j. Based on the flow network constructed above, we have the following lemma, whose proof can be found in [12].

***Lemma 1**: A feasible s-t flow of value $|F|$ in the flow network constructed above defines uniquely a set of $|F|$ non-intersecting replacement paths in the mesh.*

In the next section, we first present an efficient algorithm for Problem 2, and then show how a convex subsystem can be obtained from a min-cost flow in the network constructed above.

3. Efficient Algorithm

Problem 2 (which has both node and edge capacity constraints) is first reduced to a simpler version of min-cost flow problem in which only edges have capacities. This is done by defining a new flow network $N' = (V', E', b')$ from $N = (V, E, b, k)$ as follows: For each $u \in V - \{s, t\}$, we make two corresponding nodes $u', u'' \in V'$; if edge $(u, v) \in E$, then $(u'', v') \in E'$ *and* $(u', u'') \in E'$, for each $u \in V - \{s, t\}$. The edge

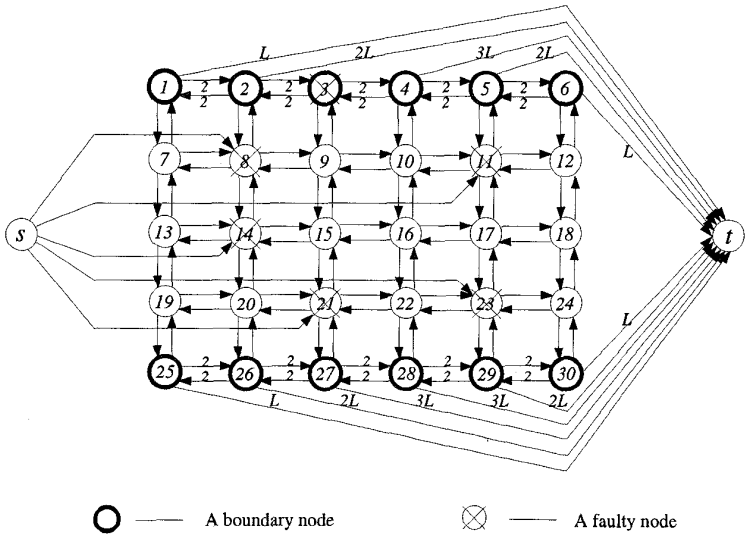


Fig. 2. The flow network N constructed from the faulty mesh shown in Fig. 1(a). (Only non-unity costs are given for simplicity.)

capacity defined on N' is

$$b'\left(u'', v'\right) = b(u, v), \qquad (u, v) \in E,$$
$$b'\left(u', u''\right) = k(u), \qquad u \in V - \{s, t\}.$$

The cost assignment on N' is

$$c'\left(u'', v'\right) = c(u, v), \qquad (u, v) \in E,$$
$$c'\left(u', u''\right) = 0, \qquad u \in V - \{s, t\}.$$

Hence, creating the new network $N' = \left(V', E', b'\right)$ can be done in time $O\left(|V'|\right)$. Fig. 3 shows how network N' is constructed from the given network N. In effect, each node $u \in V - \{s, t\}$ has been split into two parts, a "left" part u' and a "right" part u'', so that all edges entering u now arrive at its left part, whereas all edges leaving u now depart from its right part. The capacity $k(u)$ is then imposed as an edge capacity on the new edge $\left(u', u''\right)$, which has a zero cost.

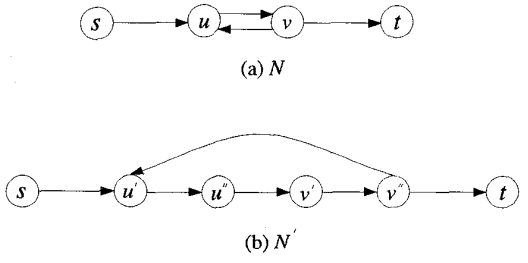


Fig. 3. Flow network N and corresponding new flow network N'.

The new network N' has a special structure: Each node has either indegree 1 or 0, or outdegree 1 or 0. Since each edge in N' has a unit capacity, only one edge incident to a node can be saturated. This indicates that every unit of the s-t flow in network N' defines a unique path connecting a node in F and a fault-free boundary node, and no two such paths intersect. We therefore have the following lemma.

***Lemma 2**: A feasible s-t flow of value $|F|$ in the new flow network N' uniquely defines a set of $|F|$ non-intersecting replacement paths in the mesh.*

Now, the problem is translated to finding a min-cost flow of value $|F|$ in network N'. There are several algorithms for this reduced version of min-cost flow problems [10]. The algorithm, called *buildup*, is slightly modified (due to the special characteristics of the network under consideration) and adopted for our use. The basic idea of *buildup* is to build up a min-cost flow step by step. At each step, an auxiliary network $AN'(f)$ with respect to the current flow f in N' (f is initialized with zero) is constructed as follows: (1) It has the same node set as N'; (2) For each edge (u, v) in N' with nonzero flow (i.e., a unit flow) and cost c, put its reverse edge (v, u) in $AN'(f)$ with cost $-c$; whereas for each edge in N' with a zero flow, put the same edge in $AN'(f)$ (see Fig. 4). Then, we find a shortest s-t path P in the current $AN'(f)$ (note that the auxiliary network $AN'(f)$ is not a flow network) and augment f in N' by adding one unit of flow along the path corresponding to P. This process continues until f reaches the value $|F|$, or until there is no more shortest path in $AN'(f)$.

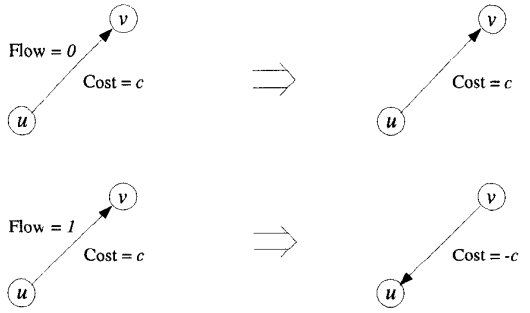


Fig. 4. Two edges in N' and corresponding edges in $AN'(f)$.

The algorithm for Problem 2 is given below as Algorithm $\Re$. In the course of execution, some edges in $AN'(f)$ may have negative costs (however, there never exists any negative-cost cycle in $AN'(f)$, see the following lemma for a proof). Thus, the shortest-path procedure used for finding P in Algorithm $\Re$ must be capable of dealing with negative costs. The Bellman-Ford shortest-path procedure [11] has such capability and is employed (with a small modification) for the implementation of Algorithm $\Re$ below.

Algorithm $\Re$: Finding a min-cost flow of value $|F|$ in the flow network N

Input: flow network $N = (V, E, b, k)$ and a value $|F|$;
Output: min-cost flow of value $|F|$;

construct a new flow network $N' = (V', E', b')$ from $N = (V, E, b, k)$ as described in the beginning of this section;
initialize f with zero;
repeat
  construct the auxiliary network $AN'(f)$ with respect to the current flow f;
  find a shortest *s-t* path, P, in $AN'(f)$;
  if such a path does not exist **then**
    terminate;
  else
    augment f in N' by adding 1 unit of flow along the path corresponding to P;
  end if
until f reaches value $|F|$

Lemma 3: *In Algorithm $\Re$, there is no negative-cost cycle in the auxiliary network $AN'(f)$ for any instance of min-cost flow f.*

Proof: Assume that there is no negative-cost cycle in the current auxiliary network $AN'(f)$. Assume also that a negative-cost cycle, C, appears in the new auxiliary network resulting from augmenting the flow f in N' along the path corresponding to a shortest *s-t* path P in $AN'(f)$. Then, C contains at least one edge that is the reverse of an edge (u, v) on P, namely, edge (v, u). If we replace edge (u, v) on P by C–(v, u), then, the cost on the resulting path is $(c(P)-c(u,v)+c(C)-c(v,u))$ $= c(P)+c(C)$, which is less than $c(P)$ (i.e., the cost of P). Thus, P was not a shortest *s-t* path in $AN'(f)$. This contradiction proves the lemma. ■

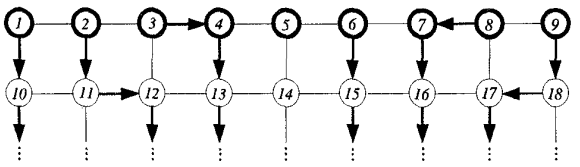


Fig. 5. Residual boundary nodes split into separate groups.

Time Complexity of Algorithm $\Re$: Since the number of edges on any *s-t* path in N' cannot exceed $\left(|V'|/|\hat{f}|\right)$ due to the special structure of N' described above, where $|\hat{f}|$ is the value of the maximum flow [10], the modified Bellman-Ford algorithm terminates in $\left(|V'|/|\hat{f}|+1\right)$ passes. Thus, it takes time $O\left(\left(|V'|/|F|\right)\cdot|E'|\right)$ to find a shortest *s-t* path in $AN'(f)$. The number of repetitions in Algorithm $\Re$ cannot be more than $|F|$, since each time we augment the flow by one unit. Therefore, the worst-case time complexity of Algorithm $\Re$ for an $m\times n$ mesh is

$$\begin{aligned} T_w &= O\left(|V'|\right) + O\left(|F|\cdot\left(|V'|/|F|\right)\cdot|E'|\right) \\ &= O((2mn)\cdot(2m(n-1)+2n(m-1)+mn+2n+|F|)) \\ &= O(M^2), \end{aligned}$$

where M $(=mn)$ is the number of nodes in the mesh.

If Algorithm $\Re$ terminates with a min-cost flow of value $|F|$, then, we know there exists a set of $|F|$ non-intersecting replacement paths in the mesh (from Lemma 2). The replacement paths can be easily identified by choosing the corresponding edges in the mesh (see Lemma 1) such that each path is from a faulty node to its corresponding boundary node. To obtain the reconfigured subsystem, each fault-free node on a replacement path is logically shifted along the path to its respective logical position (see Fig. 1). As a result, for an $m\times n$ mesh, a total of $(2n-|F|)$ fault-free boundary nodes remains logically in the upper and lower boundary rows. We refer to these boundary nodes as *residual* boundary nodes. If the residual boundary nodes in each boundary row are consecutive, then all the $(2n-|F|)$ fault-free boundary nodes contribute to the subsystem obtained. Otherwise, there is at least one boundary row whose residual boundary nodes are split into groups, each containing consecutive residual boundary nodes. In this case, the largest group in the corresponding boundary row is selected to be included in the subsystem. An example of such a case is shown in Fig. 5, where residual boundary nodes 3, 5, and 8 are split

into two groups (as node 6 is shifted to substitute node 15), and the group containing nodes 3 and 5 is chosen for inclusion in the reconfigured subsystem.

It is obvious that the subsystem so reconfigured is convex. The simulation results presented in Section 5 demonstrate that this reconfiguration scheme indeed gives maximum utilization of fault-free nodes in the system. This proposed reconfiguration is distinctive in that it identifies a maximum convex subsystem in a 2–D meshes with faults, whereas all previous reconfiguration techniques search for submeshes which are rectangular or square only. In addition, it can be proved that two tracks between every two neighboring rows and two tracks between every two neighboring columns are sufficient to realize our proposed reconfiguration, as illustrated in the next section, while earlier techniques require more tracks for reconfiguration. Unlike the fault-tolerant mesh architecture described in [5], our proposed design has a fixed node degree irrespective of the number of faults tolerable.

4. Implementation Issues

When there is no fault in the system, each node is connected to its immediate neighbors through direct links. To allow for reconfiguration after faults occur, two horizontal tracks between every two neighboring rows, and two vertical tracks between every two neighboring columns, are introduced. Each node has four *I/O* ports, namely *North* (*N*), *East* (*E*), *South* (*S*) and *West* (*W*). The actual interconnections for the convex subsystem shown in Fig. 1(b) is depicted in Fig. 6 (a), where each cross point is a switch. Fig. 6(b) shows all the possible states a switch may be at (so two control bits for each switch are sufficient). Note that two neighboring port positions at a node may be swapped if the node is on a *bent* replacement path. For instance, in Fig. 6, the positions of *E* and *S* ports in node 1 are swapped since node 1 is on a bent replacement path and the edge connecting node 1 and node 7 is a horizontal one in the logical system (see Fig. 1). Also, if a node is on a replacement path, it has two directed edges (i.e., an incoming edge and an outgoing edge) on the path, each with one of the following four directions: From/To the left, right, top, bottom.

Before giving a basic property of the replacement paths resulting from the proposed reconfiguration scheme, we introduce the next definition.

Definition 1: Two replacement paths are said to *overlap in opposite directions by γ nodes* if and only if (1) one path has exactly γ particular nodes (referred to as *p-nodes*), and the other has no more than γ *p*-nodes, such that for each *p*-node on a path, there is a *p*-node physically adjacent to it on the other path; (2) for each *p*-node on a path and its adjacent *p*-node on the other path, either their

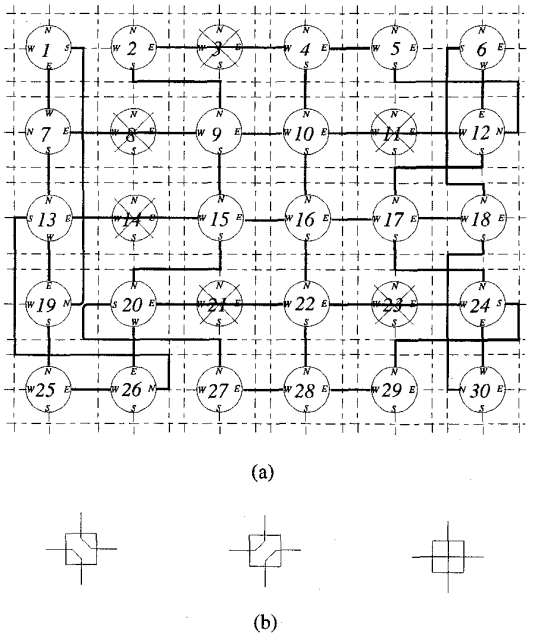


Fig. 6. (a) The actual interconnections for the subsystem shown in Fig. 1(b). (b) The states of a switch.

incoming edges or their outgoing edges are of opposite directions.

Lemma 4 below states the basic property of replacement paths, and its proof is provided in [12].

***Lemma 4**: In the proposed reconfiguration scheme, no two replacement paths overlap in opposite directions by more than two nodes.*

We arrive at the following theorem in regard to our reconfiguration, and a proof of the theorem is given in [12]. This result has been confirmed by our extensive simulation study.

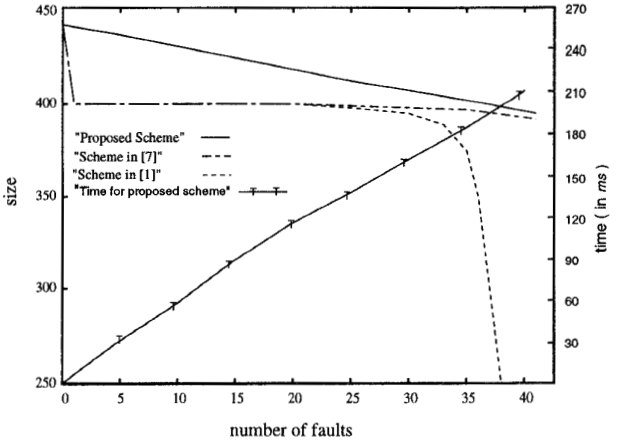
***Theorem 1**: The channel width of two is sufficient to realize all the interconnections resulting from our reconfiguration.*

It is observed from the above discussion that only a few cases use both tracks in a channel to realize interconnections and one track suffices for most cases (see Fig. 6 as an example). Thus, our reconfiguration is very likely to be successful even when one track fails (but still allows a path to cross it safely) along a channel. Since the probability of track failures is much lower than that of node failures, it appears unnecessary to deal with track failures explicitly. However, if one insists on successful reconfiguration in the presence of a single track failure at all times, an additional track has to be introduced in each channel.

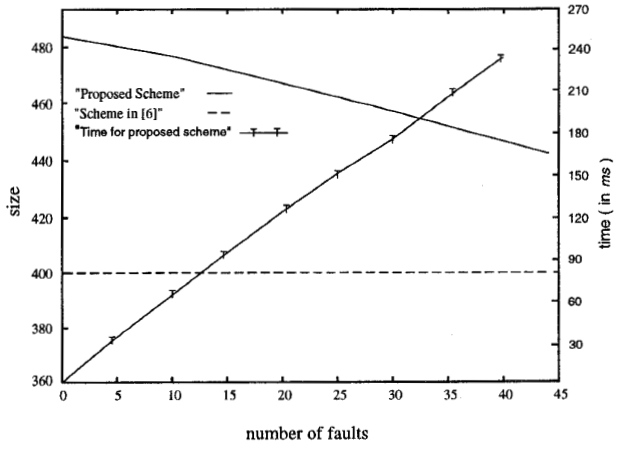
5. Simulation Results

Extensive simulations were carried out on the SUN SPARCstation 4 for various sized mesh systems. Faults are assumed to occur at nodes randomly. For a given fault number $|F|$ in a mesh system, 500 fault patterns were produced and the algorithm under consideration was activated to reconfigure the faulty system, with the derived subsystem size averaged over the 500 repetitions. For comparison, we present in Fig. 7 the results obtained for 21×21 and 22×22 mesh systems, which were considered in [7] and in [6], respectively. Those curves corresponding to previous techniques were taken directly from [7] and [6]. As shown in Fig. 7(a), when there is a single fault in the 21×21 mesh, our reconfiguration is able to retain all the fault-free nodes, i.e., 440 nodes, in contrast to only 400 nodes obtainable by either the schemes in [1] or the scheme in [7]. As the number of faulty nodes $|F|$ increases, the size of reconfigured systems reduces gradually under our scheme. For the scheme in [1] (without *assignment borrowing*), the size of reconfigured systems keeps almost unchanged as $|F|$ ranges from 1 up to 30, indicating that it is highly possible to reconfigure the faulty system into a target subsystem of 20×20 if $|F|$ is within that range. A further increase in $|F|$ causes the size of reconfigured systems to drop dramatically. This is because when $|F|$ is large, it becomes increasingly impossible to satisfy assignments of all logical cells simultaneously. The scheme in [7] achieves a similar result as the scheme in [1] except that as $|F|$ grows beyond 30, the former is still able to keep the size of reconfigured systems close to 400. This is because the scheme in [7] employs its second phase to assign available fault-free nodes to unmatched logical cells, using a set of quite complicated row and column assignment rules. In all cases, our scheme exhibits superior utilization of fault-free nodes over both the scheme in [1] and the scheme in [7] because it intends to retain as many fault-free nodes as possible. Fig. 7(b) gives the comparison between our scheme and the scheme in [6], where the number of nodes used in a 22×22 mesh without faults is only 400, since boundary rows/columns are all designated as spares. As can be observed, our scheme demonstrates much better utilization of fault-free nodes. Our proposed scheme clearly enjoys a far superior capability in retaining computational power as faults occur, despite it requires one track less than the schemes in [1], [6], and [7].

The average time spent in determining the reconfigured mesh using our proposed algorithm is also collected and illustrated in Fig. 7. It is the measured result on the SUN SPARCstation 4. For a given system, the reconfiguration time is found to increase almost linearly with respect to $|F|$, the number of faults. Our maximum reconfiguration is rather fast, as can be seen from the depicted results. In the 21×21 mesh with $|F|$=10, for example, it took less than 60 *ms* to reconfigure using our proposed



(a) 21 X 21 mesh system



(b) 22 X 22 mesh system

Fig. 7. Simulation results obtained for various sized mesh systems.

algorithm, and the reconfiguration time grows to roughly 210 *ms* for $|F|$=40. With a given $|F|$, it is observed from Fig. 7(a) and Fig. 7(b) that the time for reconfiguring the 22×22 mesh is slightly higher than that for the 21×21 mesh, as a larger system takes longer to reconfigure.

Interestingly, the aspect ratio of a 2-D mesh affects our reconfiguration results. To illustrate this point, consider a mesh composed of 900 nodes. Table I gives the simulation results under different aspect ratios. As can be observed, non-perfect utilization appears when the aspect ratio drops to 20:45 or below. In addition, for $|F|$>10, the utilization value is smaller in the last column than in the second last column. As an example, when $|F|$=50, the value is 0.9999 in the second last column compared with 0.9921 in the last column. This is because given a fixed number of faults, the chance for a replacement path to go through more than one boundary node increases as the aspect ratio reduces (note that ratio 25:36 is larger than

Table I. Utilization resulting from different aspect ratios in an $m \times n$ mesh *with* $mn = 900$

$\|F\|$	*Aspect ratio m:n*			
	30:30	*25:36*	*20:45*	*15:60*
10	*1.0000*	*1.0000*	*1.0000*	*1.0000*
20	*1.0000*	*1.0000*	*1.0000*	*1.0000*
30	*1.0000*	*1.0000*	*1.0000*	*1.0000*
40	*1.0000*	*1.0000*	*1.0000*	*0.9986*
45	*1.0000*	*1.0000*	*1.0000*	*0.9956*
50	*1.0000*	*1.0000*	*0.9999*	*0.9921*
55	*1.0000*	*1.0000*	*0.9998*	*0.9893*
60	*1.0000*	*1.0000*	*0.9996*	*0.9872*

ratio 20:45). As a consequence, residual boundary nodes in a boundary row are more likely to be split into groups when the aspect ratio decreases, causing some fault-free boundary nodes to be discarded (see Fig. 5).

6. Concluding Remarks

This paper has introduced a novel reconfiguration scheme for mesh systems, realized by reconfiguring a faulty mesh system into a maximum convex subsystem so as to retain as many fault-free nodes as possible. The reconfiguration problem is solved by transforming it into the well-known min-cost flow problem, for which an efficient polynomial algorithm is derived. It is shown that a channel width of two is sufficient for implementing the proposed reconfiguration, keeping hardware overhead low. Our simulation results indicate that reconfiguring a faulty mesh in this way indeed gives rise to a considerably larger subsystem than what can be obtained by the earlier schemes. The proposed algorithm achieves maximum reconfiguration with a fairly short execution time.

While our discussion assumes the use of the upper and lower boundary nodes for compensating internal faulty nodes, it is apparent that the proposed scheme can readily be applied to mesh systems with compensation from the left or right boundary nodes. The proposed algorithm is aimed at 2–D meshes, but the concept of searching for non-intersecting replacement paths may be extended to higher dimensional meshes. In 3-D meshes, for example, one may first determine the replacement paths on each constituent 2–D plane individually. If any plane fails to have adequate replacement paths, it is then needed to run the replacement paths across neighboring 2–D planes. We assume in this work that only nodes may fail. However, switch/link failures can be handled by disallowing some replacement paths or by translating the failures into nodes failures. It should be noted that the probability of switch/link failures is much smaller than the probability of node failures. It is also assumed that the number of faulty nodes in an $m \times n$ system is less than or equal to $2n$. The reconfiguration scheme presented here may be extended to the problem where the number of faulty nodes is arbitrary. Finding solutions to these extended problems seems to be very challenging, and is currently under investigation.

Acknowledgment — The authors wish to thank Mr. Lilong Yu at *CACS*, University of Southwestern Louisiana, for his help in implementing Algorithm $\Re$.

References

[1] H. Y. Youn and A. D. Singh, "A Highly Efficient Design for Reconfiguring the Processor Array in VLSI," *Proc. 17th Int'l Conference on Parallel Processing*, Aug. 1988, pp. 375-382.

[2] M. Wang, M. Cutler, and S. Y. Su, "Reconfiguration of VLSI/WSI Mesh Array Processors with Two-Level Redundancy," *IEEE Trans. Computers*, vol. 38, pp. 547–554, Apr. 1989.

[3] F. Lombardi *et al.*, "Reconfiguration of VLSI Arrays by Covering," *IEEE Trans. CAD Integ. Circ.*, vol. 8, pp. 952–965, Sept. 1989.

[4] M. Chean and J. A. B. Fortes, "A Taxonomy of Reconfiguration Techniques for Fault-Tolerant Processor Arrays," *IEEE Computer Magazine*, vol. 23, pp. 55–69, Jan. 1990.

[5] J. Bruck, R. Cypher, and C.-T. Ho, "Efficient Fault-Tolerant Mesh and Hypercube Architectures," *Proc. 22nd Int'l Symposium on Fault-Tolerant Computing*, July 1992, pp. 162–169.

[6] T. A. Varvarigou *et al.*, "Reconfiguring Processor Arrays Using Multiple-Track Models: The 3-Track-1-Spare-Approach," *IEEE Trans. Computers*, vol. 42, pp. 1281-1293, Nov. 1993.

[7] J. H. Kim and P. K. Rhee, "The Rule-Based Approach to Reconfiguration of 2–D Processor Arrays," *IEEE Trans. Computers*, vol. 42, pp. 1403–1408, Nov. 1993.

[8] N. J. Davis IV *et al.*, "Reconfiguring Fault-Tolerant Two-Dimensional Array Architectures," *IEEE Micro*, pp. 60–69, Apr. 1994.

[9] J. Y. Ngai, "A Framework for Adaptive Routing in Multicomputer Networks," *Caltech-CS-TR-89–09*, Computer Science Dept., California Institute of Technology, 1989.

[10] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. Englewood Cliffs, NJ: Prentice-Hall, 1982.

[11] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts: McGraw-Hill Company, 1990.

[12] N.-F. Tzeng and G. Lin, "Maximum Reconfiguration of 2–D Mesh Systems with Faults," *TR-96–8–1*, Center for Advanced Computer Studies, University of Southwestern Louisiana, 1996.