On Constructing the Minimum Orthogonal Convex Polygon for the Fault-Tolerant Routing in 2-D Faulty Meshes [1]

Jie Wu
Department of Computer Science and Engineering
Florida Atlantic University
Boca Raton, FL 33431

Zhen Jiang
Department of Computer Science
Information Assurance Center
West Chester University
West Chester, PA 19383

[1]This was supported in part by NSF grants CCR 0329741, CNS 0422762, CNS 0434533, ANI 0073736, and EIA 0130806. Contacting address: jie@cse.fau.edu

Abstract

The rectangular faulty block model is the most commonly used fault model for designing fault-tolerant and deadlock-free routing algorithms in mesh-connected multicomputers. The convexity of a rectangle facilitates simple and efficient ways to route messages around fault regions using relatively few or no virtual channels to avoid deadlock. However, such a faulty block may include many non-faulty nodes which are disabled, i.e., they are not involved in the routing process. Therefore, it is important to define a fault region that is convex and, at the same time, to include a minimum number of non-faulty nodes. In this paper, we propose an optimal solution that can quickly construct a set of minimum faulty polygons, called *orthogonal convex polygons*, from a given set of faulty blocks in a 2-D mesh (or 2-D torus). The formation of orthogonal convex polygons is implemented using either a centralized or distributed solution. Both solutions are based on the formation of faulty components each of which consists of adjacent faulty nodes only, followed by the addition of a minimum number of non-faulty nodes to make each component a convex polygon. This provides a solution to an open problem raised in [16]. Extensive simulation has been done to determine the number of non-faulty nodes included in the polygon, and the result obtained is compared with the best existing known result. Results show that the proposed approach can not only find a set of minimum faulty polygons but also does so quickly in terms of the number of rounds in the distributed solution.

Index Terms: 2-dimensional meshes (tori), fault tolerance, fault model, orthogonal convex polygons, routing.

ACRONYMS

2-D meshes	2-dimensional meshes
EW (WE)	East-to-West (West-to-East)
NS (SN)	North-to-South (South-to-North)
FB	Faulty Block
FP	Faulty Polygon
MFP	Minimum Faulty Polygon
CMFP	Centralized Solution of Minimum Faulty Polygon
DMFP	Distributed Solution of Minimum Faulty Polygon

NOTATION

u, v	Nodes u and v
ID	Node Identifier with an Encoded Address (x, y)
(E, S, W, N)	Boundary Entry with East, South, West, and North Boundary Node Information
$V[1..n]$	Boundary Array
$min_x(C)$	Minimum Coordinate of Component C along X Dimension
$max_x(C)$	Maximum Coordinate of Component C along X Dimension
$min_y(C)$	Minimum Coordinate of Component C along Y Dimension
$max_y(C)$	Maximum Coordinate of Component C along Y Dimension

1 Introduction

Mesh-connected multicomputers, especially those with low-degree, are one of the simplest and least expensive structures for building a system using hundreds and even thousands of processors. Low-degree mesh-connected multicomputers include meshes and tori (which are meshes with wraparound connections). Dally [5] and Agarwal [1] recommended the use of low-degree networks, such as 2-dimensional (2-D) and 3-dimensional (3-D) meshes (tori), over high-degree networks, such as hypercubes, for better performance and cost-effectiveness. In a mesh-connected multicomputer, processors (also called nodes) exchange data and coordinate their efforts by sending and receiving messages through the underlying mesh network. Thus, the performance of such a system depends heavily on the end-to-end cost of

communication mechanisms. Routing is the process of transmitting data from one node to another node in a given system. As the number of nodes in a mesh-connected multicomputer increases, the chance of failure also increases. At the same time, applications that run on such a system are often critical and may have real-time constraints. Therefore, the ability to tolerate failure is becoming increasingly important, especially for routing [6]. In this paper, we focus on 2-D meshes (tori) and, in the subsequent discussion, we use meshes to represent both meshes and tori.

In designing a fault-tolerant routing algorithm, one of the most important issues is to select an appropriate fault model. A good fault model should accurately reflect fault situations in a real system. It should be defined in such a way that fault information can be easily established and maintained. The model should not be too complex and, at the same time, not overly simplified so that certain objectives, such as optimization and deadlock-free requirements, are still obtainable. Most of the literature on fault-tolerant routing uses disjoint rectangular blocks [2, 3, 4, 7, 11, 20, 21] to model node faults (link faults can be treated as node faults) and to facilitate routing in 2-D meshes. First, a node labelling scheme that identifies nodes (faulty and non-faulty) that cause routing difficulties is defined and such nodes are called unsafe nodes. Connected unsafe nodes form a faulty rectangular region. Such a region is also called a rectangular faulty block, or simply a faulty block. A faulty block can be easily established and maintained through message exchanges among neighboring nodes.

The convexity of each faulty block facilitates a simple fault-tolerant and deadlock-free routing using either relatively few virtual channels [4, 11, 13] or no virtual channel [18]. The convex feature of a faulty block is a necessary condition for progressive routing, where the routing process never backtracks. The absence of backtracking in turn is a necessary condition for minimal routing (also called shortest path routing [9, 19]), where the destination is reached through a minimal path from the source. The fault-tolerant minimal routing algorithm for 2-D meshes has been developed in [17] using the faulty block model. There are several studies ([3, 12, 14]) on fault-tolerant routing that can handle non-rectangular fault regions, such as H-shape, L-shape, T-shape, U-shape, and +-shape fault regions.

Despite all the desirable features of the faulty block model, a major problem is that a faulty block may include many non-faulty nodes treated as faulty nodes (with the unsafe label). A non-faulty node in the faulty block will not be used in any routing process. This leads to the loss of the communication ability of such a node. In the worst case, a majority of nodes in a faulty block are non-faulty. Although some efforts have been made either to enhance the faulty block definition to include fewer non-faulty nodes in a faulty block [10] or to activate some boundary non-faulty nodes in a faulty block as in [2, 11], the above problem still exists.

In this paper, we try to solve the above problem by providing an approach to find a set of disjoint convex polygons that cover all the faulty nodes with a minimum number of non-faulty nodes included

in these polygons. In this way, the fault situation in the networks can be desribed accurately in this new fault model without including any redundant non-faulty node. It is the fundamental block of providing a fault-tolerant routing so that a robust network communication can be obtained at the most degree without extra loss of communication ability of nodes in a situation when some faults occur in the networks.

A convex region (polygon) is defined as a region (polygon) P for which a line segment connecting any two points in P lies entirely within P. If we change the "line segment" in the standard convex region definition to "horizontal or vertical line segment", the resultant region is called an *orthogonal convex region (polygon)* ([8, 16]). Clearly, a faulty block is a special orthogonal convex region. In 2-D meshes, the boundary lines of a region are either horizontal or vertical, therefore, each region is a polygon. In the subsequent discussion, we use the terms polygon and region interchangeably.

The challenge here is not only to conduct a theoretical study on the feasibility of finding such a smallest orthogonal convex polygon but also to search for a practical and efficient distributed implementation in a system where each processor knows only the status of its neighbors. In [16], a simple and efficient distributed algorithm is presented that determines a set of small orthogonal convex polygons to cover all the faults in a given faulty block. This approach consists of two phases. First, a set of disjoint faulty blocks are constructed from a given set of faulty nodes. Each faulty block may contain many non-faulty nodes. Then, each faulty block is partitioned into one or many disjoint orthogonal convex polygons by removing some non-faulty nodes from the block. It is shown in [16] that a resultant region generated after removing non-faulty nodes from the given faulty block is the minimum orthogonal convex polygon (simply called a *faulty polygon*) that covers all the faults in the region. Note that there may be several polygons generated from a given faulty block. In addition, it is shown that the number of non-faulty nodes covered in these faulty polygons (from a given faulty block) is no more than that of the minimum orthogonal convex polygon that includes all the faulty nodes in the original faulty block. However, for certain cases, a faulty polygon can be further partitioned, and more non-faulty nodes in the region can be removed. Therefore, such a faulty polygon is called *sub-minimum faulty polygon*. This brings the following open problem: For a given faulty block, find a set of disjoint orthogonal convex polygons that covers all the faults in the faulty block and contains a minimum number of non-faulty nodes. Clearly, each faulty polygon is minimum (called *minimum faulty polygon*) in the sense that it cannot be replaced by a set of faulty polygons that include fewer non-faulty nodes.

In this paper, we first provide a centralized solution for the open problem and, then, an efficient distributed solution is provided. This approach again consists of two phases. In the first phase, a set of components are formed, where each *component* consists of adjacent faulty nodes with no non-faulty node. In the second phase, a minimum number of non-faulty nodes are added to make each component

a convex polygon. Extensive simulation has been done to determine the number of non-faulty nodes included in the polygon, and the result obtained is compared with the best existing known result.

This paper is organized as follows: In Section 2 we provide some preliminaries including the definition of an orthogonal convex polygon, its application in fault-tolerant and deadlock-free routing in 2-D meshes, and a distributed formation of a sub-minimum faulty polygon. In Section 3, we propose an optimal solution for constructing minimum faulty polygons. An efficient distributed solution is also provided. In Section 4, simulation results on the average number of rounds needed to construct minimum faulty polygons and the number of non-faulty nodes in the polygons are given. Finally, in Section 5, we provide our conclusions and ideas for future work.

2 Preliminaries

2.1 Orthogonal convex polygons

We consider only node faults and assume that faulty nodes just cease to work. Also, each non-faulty node knows the status of its neighbors only; that is, there is no *priori* global information of fault distribution. A 2-dimensional (2-D) $n \times n$ mesh with n^2 nodes has an interior node degree of 4 and a network diameter of $2(n-1)$. Each node u has an address (u_x, u_y), where $u_x, u_y \in \{0, 1, ..., n-1\}$. Two nodes u: (u_x, u_y) and v: (v_x, v_y) are connected if their addresses differ in one and only one dimension, say dimension x. Moreover, $|u_x - v_x| = 1$. Similarly, if they differ in dimension y, then $|u_y - v_y| = 1$.

In the following, we give the definition of an orthogonal convex region.

Definition 1 [16]: *A fault region is* orthogonal convex *if and only if the following condition holds: For any horizontal or vertical line, if two nodes on the line are inside the region, all the nodes on the line that are between these two nodes are also inside the region.*

The difference between a standard convex region and an orthogonal convex region is that the line in the latter is restricted to only horizontal and vertical, whereas the line in the former can be along any direction in a standard convex region. In 2-D meshes, each orthogonal convex region is also an orthogonal convex polygon. Clearly, T-shape, L-shape, and +-shape (the shadowed region in Figure 1) fault regions are orthogonal convex polygons, whereas U-shape and H-shape fault regions are non-orthogonal convex polygons. For example, orthogonal convex region $\{(2,4),(3,4),(4,3)\}$ is an L-shape polygon (the shadowed region in Figure 2).

Note that an orthogonal convex polygon is the same as a solid fault in the solid fault model [3]. It is shown in [3] that only four virtual channels are needed around a convex region to support fault-tolerant and deadlock-free routing. However, finding an algorithm for computing a smallest solid fault was not included in [3]. Wang [15] independently proposed a routing-sensitive convex region where

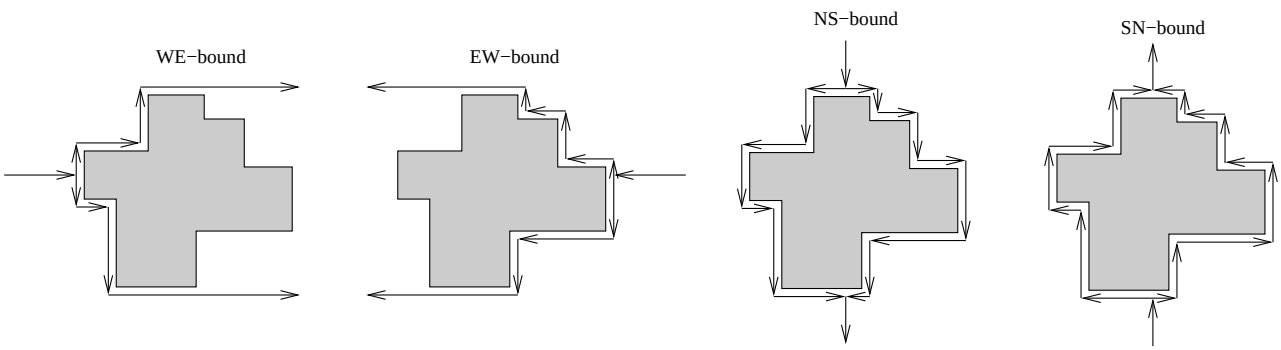

Figure 1: Routing of messages around disabled regions.

each convex region has two versions (shapes), one for west-east routing and the other for east-west routing.

2.2 Fault-tolerant and deadlock-free routing

We first show the application of orthogonal convex polygon in achieving fault-tolerant and deadlock-free routing in 2-D meshes. It is assumed that faults in a given 2-D mesh are contained in a set of disjoint faulty polygons. Chalasani and Boppana's *extended e-cube routing* [3] is used to illustrate the routing process. The *e-cube routing* (also called x-y routing) sends a message in a row (x-direction) until the message reaches a node that is in the same column as its destination and, then, sends the message in the column (y-direction). Consider a routing process from node (1,3) to node (6,4), the message is first routed along the row to node (6,3) and, then, the message follows the column to reach destination (6,4). The extended e-cube routing follows the base e-cube routing until it hits a faulty polygon, and then, sends the message around the region (and the message is in an "abnormal" mode), clockwise or counterclockwise based on a set of rules (to be discussed shortly), until it becomes "normal" again, i.e., the region no longer has effect on the routing process.

Messages are classified into one of four types: EW (East-to-West), WE, NS, or SN. A message is initially labelled as EW-bound or WE-bound, depending on its direction of travel along the row. Once a message completes its row hops, it becomes a NS-bound or SN-bound message. For the routing example from node (1,3) to node (6,4), the message is WE-bound initially. The message becomes SN-bound once it reaches node (6,3). Once a message hits a faulty polygon, it has to route around the region. Figure 1 shows ways to route around a faulty polygon for different types of messages. The orientation (clockwise or counterclockwise) is a don't care for an NS- or SN-bound message. The orientation for a WE-bound (EW-bound) message is the following: clockwise (counterclockwise) if the

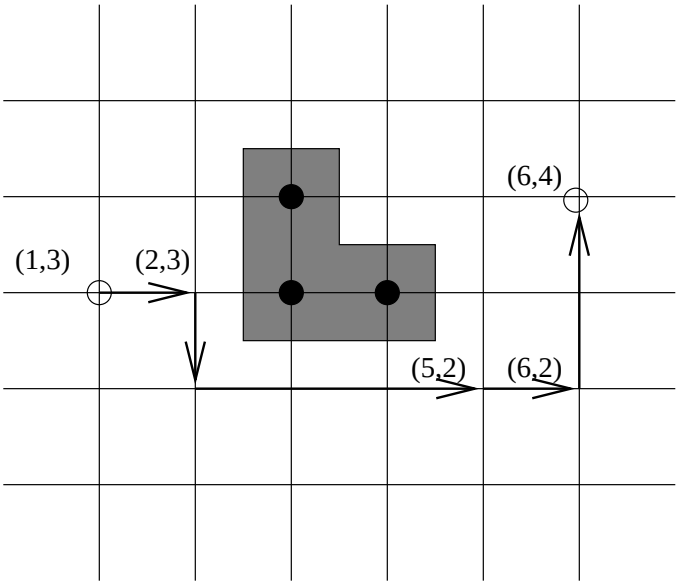

Figure 2: A routing example from source (1,3) to destination (6,4).

message is in a row above its row of travel; counterclockwise (clockwise) if the message is in a row below its row of travel, and don't care (don't care) if the message is in the same row of travel.

Consider again the routing process from (1,3) to (6,4). The message is routed along the row towards east and it is WE-bound. Suppose there is a faulty polygon {(2,4), (3,4), (4,3)} (see Figure 2). Once it reaches the boundary of the faulty polygon at node (2,3), the message is routed along the faulty polygon. Since node (2,3) is in a row that is below the row of travel (or the row of the destination), the message is routed around the faulty polygon counterclockwise as shown in Figure 2. Once the message reaches node (5,2) it becomes "normal" again. The rest of the routing process follows the base e-cube routing along the row to node (6,2), and then, along the column to reach node (6,4).

To ensure freedom from deadlock and livelock, four virtual channels, vc_0, vc_1, vc_2, vc_3, are used for channels around faulty polygons. Note that more virtual channels are needed if the region is concave. The use of these virtual channels is as follows: EW-bound messages use vc_0 for all hops around faulty polygons, WE-bound messages use vc_1, NS-bound messages uses vc_2, and SN-bound messages use vc_3. Other details related to the extended e-cube routing can be found in [3].

2.3 Distributed formation of sub-minimum faulty polygon

Wu's sub-minimum faulty polygon construction [16] is based on two labelling schemes:

- **Labelling scheme 1**: All faulty nodes are *unsafe*, and all non-faulty nodes are *safe* initially. A non-faulty node is changed to *unsafe* if it has a faulty or unsafe neighbor in both dimensions; otherwise, it remains *safe*.

- **Labelling scheme 2**: All faulty nodes are marked *disabled*. All safe nodes are marked *enabled*. An unsafe node is initially marked *disabled*, but it is changed to *enabled* if it has two or more enabled neighbors.

Based on the above labelling schemes, a faulty node must be unsafe and disabled. For a non-faulty node, there are three possible cases: (1) safe and enabled, (2) unsafe and enabled, and (3) unsafe and disabled. The labelling scheme 1 corresponds to a *growing phase* that includes non-faulty nodes in the block and, as a result, rectangular faulty blocks are generated. The labelling scheme 2 corresponds to a *shrinking phase* that removes non-faulty nodes from rectangular faulty blocks.

Figure 3 (a) shows rectangular faulty blocks generated from ten faulty nodes (black nodes). Gray nodes are unsafe nodes. Other safe non-faulty nodes are not shown. Figure 3 (b) shows two orthogonal convex polygons generated from Figure 3 (a). Gray nodes are unsafe and disabled nodes (i.e., non-faulty nodes contained in the polygon), and white nodes are unsafe but enabled nodes (i.e., non-faulty nodes that are originally in rectangular faulty block, but are removed from the orthogonal convex polygon). Clearly, the right polygon in Figure 3 (b) is not minimum and it can be further partitioned into three polygons as shown in Figure 3 (c).

Wu [16] shows that under the condition that each polygon cannot be further partitioned (such as the left polygon in Figure 3 (b)), the polygon is minimum. That is, the polygon contains the minimum number of non-faulty nodes unless the original faulty block can be further partitioned.

3 Minimum Orthogonal Convex Polygon

We start with a centralized solution for determining minimum faulty polygons, followed by a distributed solution. Both solutions consist of two phases. First, faulty nodes are grouped into a set of components, where faulty nodes in each component are adjacent. Then, a minimum number of non-faulty nodes are included to make each component a polygon.

3.1 Centralized solutions

In the centralized solution, first we define the notion of *node adjacency*.

Definition 2: *For a given node* (x, y)*, the adjacent nodes of* (x, y) *are* $(x-1, y-1)$*,* $(x-1, y)$*,* $(x-1, y+1)$*,* $(x, y-1)$*,* $(x, y+1)$*,* $(x+1, y-1)$*,* $(x+1, y)$*, and* $(x+1, y+1)$*.*

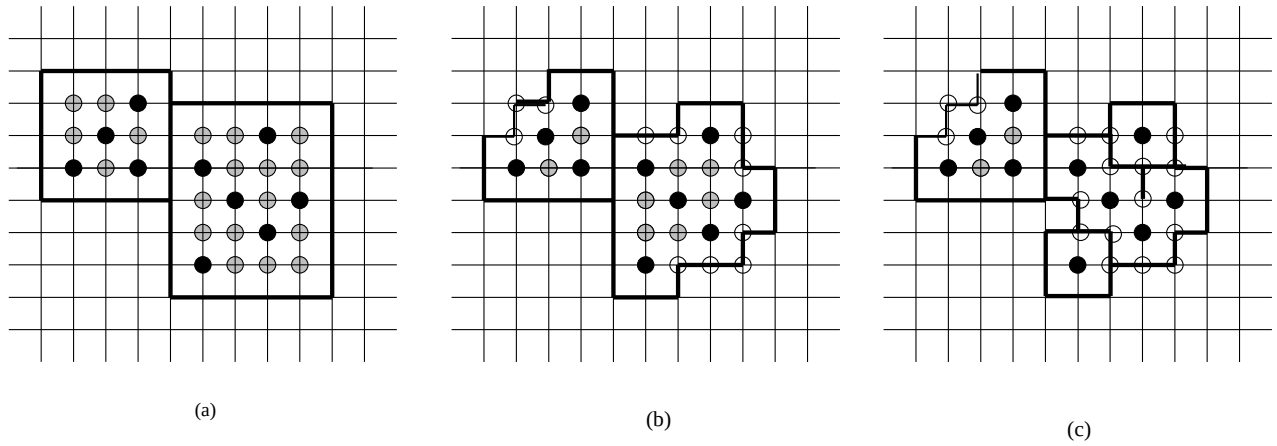

Figure 3: (a) Rectangular faulty blocks, (b) sub-minimum faulty polygon, and (c) minimum faulty polygon.

In phase 1, a *merge process* is used that groups faulty nodes into components, where each component consists of adjacent faulty nodes only. Clearly, a sub-minimum faulty polygon may include multiple components. In addition, for each component, four coordinates $\min_x$, $\min_y$, $\max_x$, and $\max_y$, representing the minimum and maximum coordinates of any nodes in this component along X and Y dimensions, are maintained in merge process.

In phase 2, there are two possible solutions. One is based on applying labelling schemes 1 and 2 on each component. The other is based on the definition of orthogonal convex polygon to "fill in" each concave region of a component with non-faulty nodes.

In the first solution of phase 2, for each component, a *virtual faulty block* is constructed by applying the labelling scheme 1. In the virtual faulty block, four corners are (min_x, min_y), (min_x, max_y), (max_x, min_y), and (max_x, max_y). Such a rectangle can be simply represented by two opposite corners $[(min_x, min_y), (max_x, max_y)]$. Applying labelling scheme 2, minimum faulty polygons are derived, one for each virtual faulty block.

Figure 4 shows an example of two components and the corresponding virtual faulty blocks. In Figure 4 (a), white nodes are unsafe but enabled nodes in the sub-minimum faulty polygon. In Figure 4 (b) and (c), white nodes are unsafe but enabled nodes in the virtual faulty block, but removed from the minimum faulty polygon. Gray nodes are unsafe and disabled nodes. Virtual faulty blocks (Figures 4 (b) and (c)) are encircled by rectangles with dashed lines and minimum faulty polygons are represented by rectangles with thick lines. The final diagram (Figure 4 (d)) is constructed by "piling" all the diagrams on top of each other, with the following superseding rule for node status:

Minimum Orthogonal Convex Polygons (Centralized Solution based on Labelling Schemes 1 and 2)

1. Construct components using the merge process. Each component C maintains minimum and maximum coordinates along X and Y dimensions: $min_x(C)$, $min_y(C)$, $max_x(C)$, and $max_y(C)$.
2. Apply labelling scheme 1 to each component C to generate a virtual faulty block. Basically, all nodes inside the boundary of $X : [min_x(C), max_y(C)]$ and $Y : [min_y(C), max_y(C)]$ other than those of C are treated as non-faulty nodes, and then become unsafe after the labelling process.
3. Apply labelling scheme 2 to enable some non-faulty but unsafe nodes in the virtual faulty block.
4. Use the superseding rule to resolve conflicting node status for final status of each non-faulty node.

- **Superseding rule**: black nodes overwrite gray and white nodes, and gray nodes overwrite white nodes.

Note that using the sub-minimum faulty polygon construction, two components in Figure 4 will be placed in one faulty block after the labelling scheme 1. Also, the white node on the north-west corner will be enabled. The resultant polygon (shown in Figure 4 (a)) still includes two components plus one gray node and three white nodes in Figure 4 (d) and it is not minimum.

Theorem: *Each polygon P derived based on the proposed process is a minimum faulty polygon.*

Proof: We need only to prove that there exists no set of disjoint convex polygons that covers all the faulty nodes in polygon P, and that in addition, the number of non-faulty nodes included in these polygons is less than that of P.

Suppose P is constructed out of k components $C_1, C_2, ..., C_k$, and the corresponding polygons are $P_1, P_2, ..., P_k$. Based on the definition of component, each P_i $(1 \leq i \leq k)$ cannot be partitioned. Let $P_1^{'}, P_2^{'}, ..., P_m^{'}$ be any set of disjoint faulty polygons that covers all faults in P. Then for each P_i, there is a $P_j^{'}$ $(1 \leq j \leq m)$ that covers all faults in P_i. Since P_i is derived from labelling schemes 1 and 2 on C_i and it cannot be further partitioned (see in Theorem 2 in [16]), $P_j^{'}$ includes also all non-faulty nodes in P_i. Therefore, P: P_1, P_2, ... P_i contains no more non-faulty nodes than those of $P_1^{'}$, $P_2^{'}$, ... $P_m^{'}$. ■

The first solution is based on emulating labelling scheme 1 (growing each component into a virtual

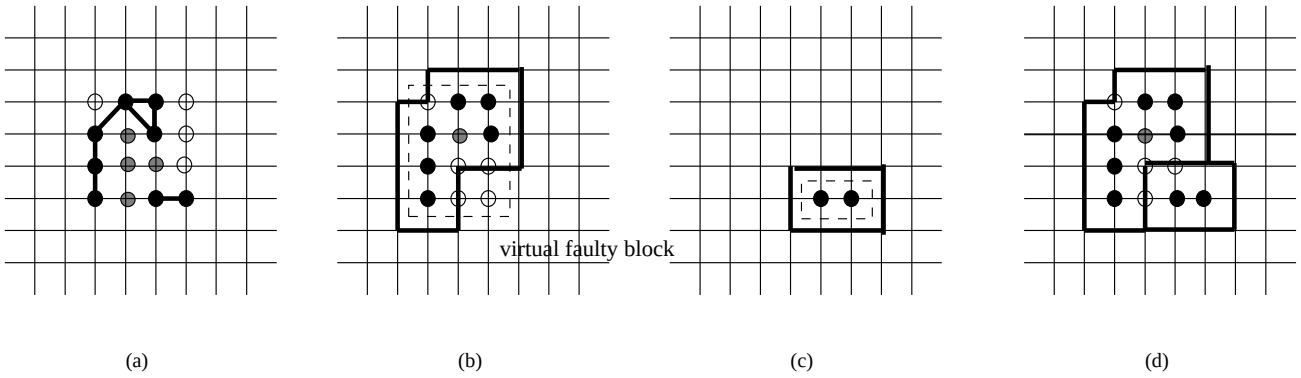

Figure 4: Centralized solution: (a) components, (b) and (c) faulty polygons, and (d) "piling" all faulty polygons.

faulty block) and labelling scheme 2 (shrinking each virtual faulty block into a minimum faulty polygon). In the second solution, non-faulty nodes are directly added to the concave region of a component to convert it to a convex polygon. The correctness of this approach is clear since it is equivalent to the labelling schemes 1 and 2 applied on each component. We first define the notion of *concave row section* and *concave column section*.

Definition 3: *Given a component, for a horizontal (vertical line) where two end nodes on the line are inside the component, each section of the line that is outside the component is called a concave row (column) section.*

To find minimum faulty polygons, we only need to disable all nodes on the concave row (column) section. To identify all concave row (column) sections, we need to "scan" each component twice, horizontal (from min_x to max_x) and vertical (from min_y to max_y). Each non-faulty node is enabled by default, the final status of each non-faulty node is decided based on the same superseding rule discussed in the last subsection.

3.2 Distributed solutions

The distributed solution is more involved. The first challenge is the construction of a component. This is done by *boundary nodes* of a component, i.e., nodes outside any faulty components but adjacent to this component. If a boundary node is at the north-side of a component, it is called a *north boundary* node with respect to the component. Other adjacent nodes towards south, east, and west are defined in a similar way. Note that each node may have multiple boundary status (e.g., a node can be a south and north boundary node if it is at the south and north of the component). Boundary nodes

Minimum Orthogonal Convex Polygons (Centralized Solution based on Concave Row and Column Sections)

1. Construct components using the merge process. Each component C maintains minimum and maximum coordinates along X and Y dimensions: $min_x(C)$, $min_y(C)$, $max_x(C)$, and $max_y(C)$.
2. Identify all concave row and column sections by scanning each component twice, one horizontally from min_x to max_x and one vertically from min_y to max_y.
3. Assign disable status to all nodes in concave row and column sections.
4. Use the superseding rule to resolve conflicting node status for final status of each node.

form a "ring" surrounding the faulty component. A boundary node is the *south-west outer corner* (see the triangle node in Figure 5 (a)) if it has a west boundnary neighbor and a south boundary neighbor. A boundary node is the *south-west inner corner* (see the triagnle node in Figure 5 (c)) if it is an east and a north boundary node at the same time. Note that south-west outer corner belongs to the boundary ring, but it is not an east, south, west, or north boundary node. Once the boundary ring is constructed, it is assumed that each node knows the next boundary node (along the clockwise direction).

We assume that both the west-most south-west outer corner and the west-most south-west inner corner (if any) initiate the ring construction process and these corners are also called the *initiators*. The initiation message goes around the faulty component in a clockwise direction (see Figure 5, where an initiator is represented by a triangle). The situation for multiple initiations will be discussed later.

The next challenge is the distributed method of determining the status of each non-faulty node. Since a concave row/column section may overlap with another faulty component (as will be seen later with an example), the centralized approach based on labelling schemes 1 and 2 cannot be directly applied. A disabled node in a concave row/column section of one faulty component can be a faulty node in another faulty component. Therefore, such a node cannot participate in neighbor information exchange as required in labelling schemes 1 and 2. Instead of involving all nodes in the concave row and column section, we use only boundary nodes that are end nodes of concave row/column sections, since a boundary node cannot be a faulty node in any faulty component. Each concave row/column section is defined during the boundary ring construction. The following information needs to be piggybacked with the initiation message:

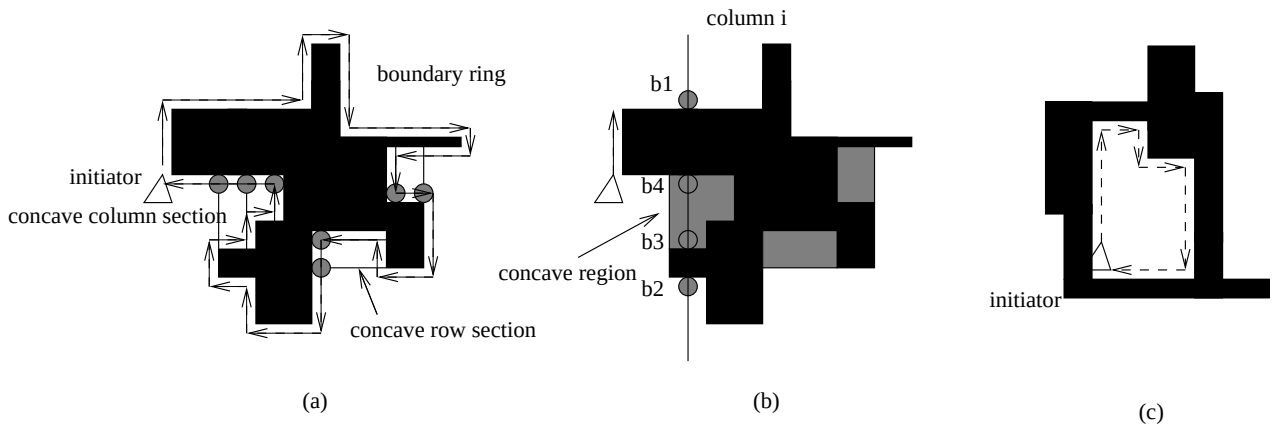

Figure 5: Distributed solution for (a) and (b) open concave region, and for (c) closed concave region.

- Initiator ID (x, y)
- Boundary 2-D array $V[1..n](E, S, W, N)$

Initiator ID is used as the message id, and it is also used to determine when the message should terminate (i.e., when a node receives a message with its own id). Boundary array ($n \times 4$ 2-D array) is used to determine concave row and column sections, where E, S, W, N are used to store east, south, west, and north boundary node information. This array includes *row number* for the most recently visited north and south boundary node at each column and *column number* for the most recently visited east and west boundary node at each row. Basically, in a given $n \times n$ 2-D mesh, one entry is used for south (north) boundary nodes on each row (from row 1 to row n) and one entry is used for east (north) boundary nodes on each column (from column 1 to column n). Initially, all entries in the boundary array are initialized to "-" (undefined). It should be stressed that during the ring formation, multiple boundary nodes of the same type (east, south, west, or north) may appear. There is only one entry for each type per row (column). During the ring formation of the example in Figure 5 (b), there are four boundary nodes, b_1 (north), b_2 (south), b_3 (north), and b_4 (south), in column i. $V[i](S)$ stores the row number of b_1 and later the row number of b_3. Similarly, $V[i](N)$ stores the row number of b_2 which is later substituted by the row number of b_4.

In the following algorithm, one end node of each concave row (column) section is selected to be in charge of notification of disable status to all nodes in the corresponding section (see gray nodes in Figure 6). Such an end node is also called *notification end node* which is responsible for notification and maintains the positions of two end nodes of the corresponding concave section.

In the example of Figure 5 (b), the initiation message visits node b_1 first and stores its row

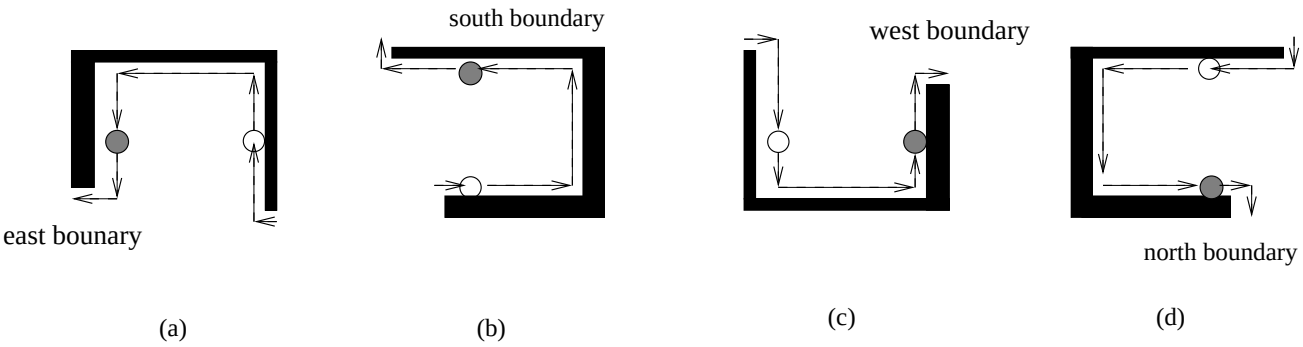

Figure 6: Four different cases of concave sections: (a) and (c) concave row section, and (b) and (d) concave column section.

Concave Row and Column Section (Distributed Solution)

1. Upon receiving the initiation message, the current node does the following if it is an east, south, west, or north boundary node:

   - Update the corresponding entry $V[1..n](E, S, W, N)$ based on the type of boundary node and the current node position. In case of multiple boundary status, multiple entries are updated.
   - The current node is a notification end node for concave row or column section
     - if the current node is an east (west) boundary and the west (east) boundary of the same row has a record with no smaller (no larger) a column number (see Figures 6 (a) and (c)), or
     - if the current node is a south (north) boundary and the north (south) boundary of the same column has a record with no smaller (no larger) a row number (see Figures 6 (b) and (d)).
   - If the current node is a notification end node, it records the positions (stored in V) of two end nodes of the corresponding concave row (column) section.

2. Pass on the initiation message to the next node in the boundary ring with the updated boundary array.

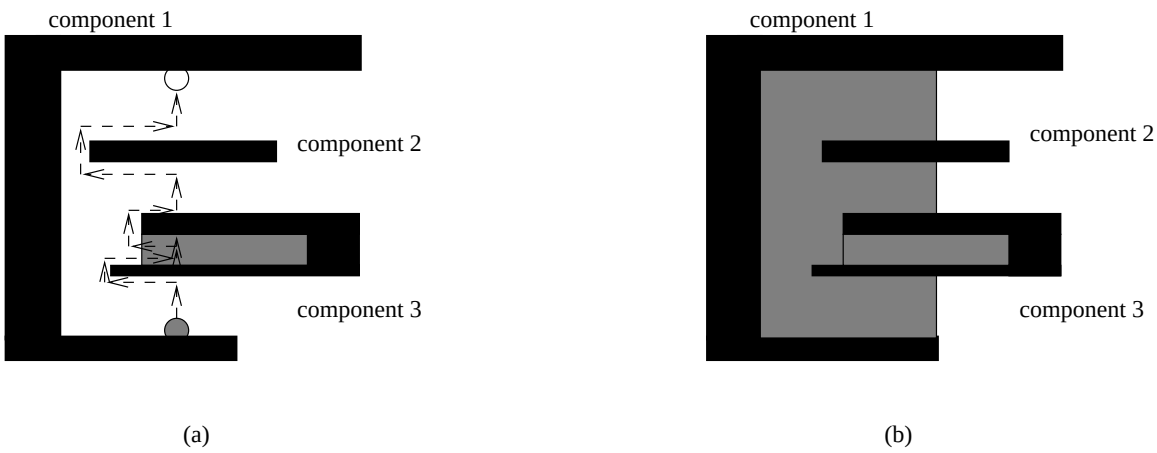

Figure 7: Notification in a concave column section with blocking polygons.

information in $V[i](N)$ (which is initiated to "-") . When b_2 is visited, its row information is stored in $V[i](S)$ (which is also initiated to "-"). Since the row number of b_2 is smaller than the row number of b_1 (i.e., $V[i](S) < V[i](N)$), no action is needed. When b_3 is visited, its row information is stored in $V[i](N)$ (i.e., the row number of b_1 is replaced). Again, no action is needed since $V[i](S) < V[i](N)$. Finally, when b_4 is visited, its row information is stored in $V[i](S)$. Since the row number of b_4 is no smaller than the row number of b_3 (i.e., $V[i](S) \geq V[i](N)$), b_4 is the notification end node for a concave section in column i. The row positions of two end nodes are recorded in $V[i](S)$ and $V[i](N)$.

Various optimizations are possible on reducing memory space by combining $V[i](S)$ with $V[i](N)$ and $V[i](E)$ with $V[i](W)$, and removing redundant portions in different concave sections by also holding the second most recently visited boundary node information. Details are more involved and are skipped here.

One more challenge is that a concave region (consists of concave row and/or column) may actually contain faults (it is fault-free only with respect to the current component) and the corresponding faulty polygons are called *blocking polygons*. In the absence of blocking polygons, forwarding status along the concave row (column) section is straightforward along a row (column); otherwise, the message that carries node status must route around blocking polygons. Figure 7 (a) shows an example where a concave column section in a concave region (component 1) has to bypass two blocking polygons, components 2 and 3. Note that, when routing around blocking polygons, node status for portions (of the section) that also belong to concave regions of blocking polygons (such as that of component 3 in Figure 7 (a)) is determined multiple times (i.e., colored twice in Figure 7 (a), one by component 1 and one by component 3). The end result of concave region is shown in Figure 7 (b).

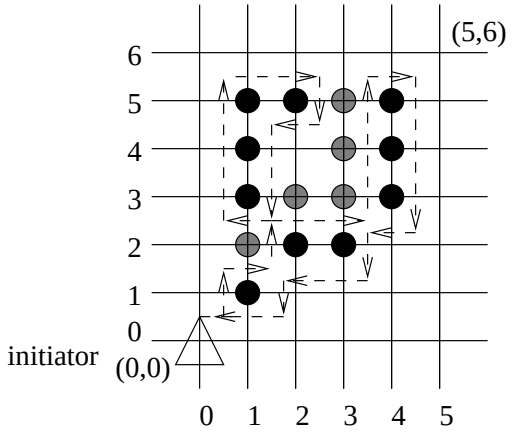

Figure 8: An example.

In a component, there are many south-west corners (inner or outer). It is possible that many or all of them can initiate the process. Therefore, the procedure starts whenever a new south-west corner is formed and we have the following overwriting rule in the priority order of (a) and (b): When a node receives more than one initiation message, (a) the one with a smaller x value in initiator ID overwrites the rest and, then, (b) the one with a smaller y value in initiator ID overwrites the rest. A south-west corner is "awakened" by an incoming initiation message if it has not started its process and its ID overwrites that of the incoming message. In this way, the west-most south-west (inner or outer) corner will dominate even if it is not the initiator originally.

Figure 8 shows an example of one component with ten faulty nodes (black nodes). Gray nodes in the figure correspond to notification end nodes of concave row/column sections. The remaining disabled nodes are not shown. Based on the distributed solution, node (1, 2) is a south boundary node responsible for concave column section [1, 2..2] (a column section with one row). In fact, node (1, 2) has three statuses: north, west, and south boundary node; this node updates three entries in the boundary array with the same timestamp. Node (2, 3) is a north boundary node for column section [2, 3..4]. Node (2,3) is also an east boundary node. Nodes (3, 3), (3, 4), and (3, 5) are all west boundary nodes for concave row sections [2..3, 3], [2..3, 4], and [3..3, 5], respectively.

4 Simulation

A simulation has been conducted in a 100×100 mesh to test the effectiveness of the new faulty polygon model. To simplify the simulation, we assume that the number of faults is no more than 800. After the

occurrence of faults, faulty blocks, sub-minimum faulty polygons, and minimum faulty polygons (in both centralized and distributed solutions) are constructed through rounds of information exchanges and updates between neighbors. We consider two fault distributions, assuming all faults are sequentially added to the network: First, we randomly generate the positions of these faults in the random fault distribution model. Second, to test cases with large faulty components, a clustered fault distribution model is adopted here. In this model, all points have the same failure rate initially. After a fault (x, y) is inserted, the failure rate of its adjacent neighbors (eight in all) is doubled. Therefore, there are two different failure rates, one for nodes that are not adjacent neighbors of existing faults and the other for nodes that are adjacent neighbors of existing faults. Under the clustered fault distribution model, faults tend to form clusters.

Figure 9 shows the average number of non-faulty nodes contained in faulty blocks (FB), sub-minimum faulty polygons (FP), and minimum faulty polygons (MFP) in the whole network. The results under the random fault distribution model are shown in Figure 9 (a) and those under the clustered fault distribution model are shown in Figure 9 (b). The faulty polygon (FP or MFP) covers all the faults in the FB model but contains fewer non-faulty nodes than does FB. Under the FP model, 50% of non-faulty nodes contained in FB can be enabled. Under the MFP model, 90% of non-faulty nodes contained in FB can be enabled.

Figure 10 shows the average size of FB, FP, and MFP by the number of faulty and non-faulty nodes they contain. The results under the random fault distribution model are shown in Figure 10 (a) and those under the clustered fault distribution model are shown in Figure 10 (b). The average size of both FP and MFP is smaller than that of FB. The average size of MFP is the least of the three (FB, FP, and MFP). Because of the scattered distribution of faults under the random fault distribution model, most faulty blocks (under FB or FP model) contain no more than four nodes and most virtual faulty blocks (under MFP model) contain only one faulty node. Under the clustered fault distribution model, the size of each faulty block becomes large and the average size can be six times that of the random faulty distribution model. However, the average size of MFP does not increase much, even when the number of faults reaches 800.

Figure 11 shows the average number of rounds of information exchanges between neighbors for status determination in the networks under FB, FP, and MFP (the centralized solution CMFP and the distributed solution DMFP) model. More specifically, under the FB model, numbers of rounds are needed for applying the labelling scheme 1. Under the FP model, extra rounds are needed for applying labelling scheme 2. Under the CMFP model, we "emulate" labelling schemes 1 and 2 on faulty components, where a virtual faulty block is generated from each component. Rounds of information exchanges are needed when labelling schemes 1 and 2 are applied on each faulty component. Under the DMFP model, we use the boundary ring construction followed by notification from one end node

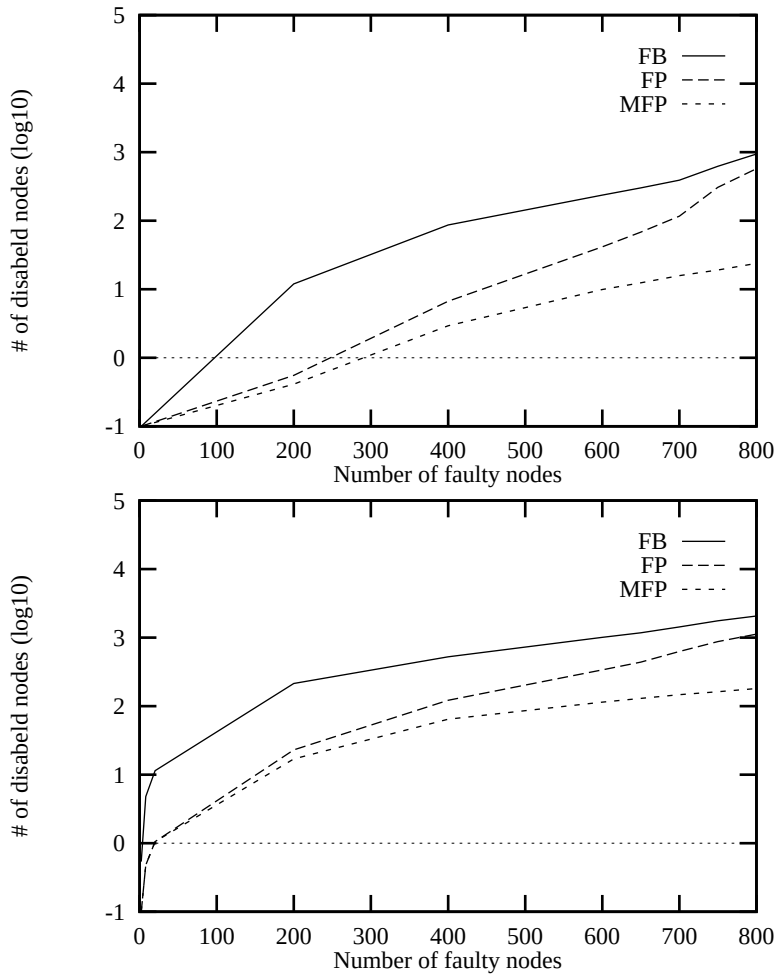

Figure 9: Average number of non-faulty but disabled nodes contained in faulty blocks (FB), sub-minimum faulty polygons (FP) and minimum faulty polygons (MFP) in the whole network: (a) the random fault distribution model and (b) under the clustered fault distribution model.

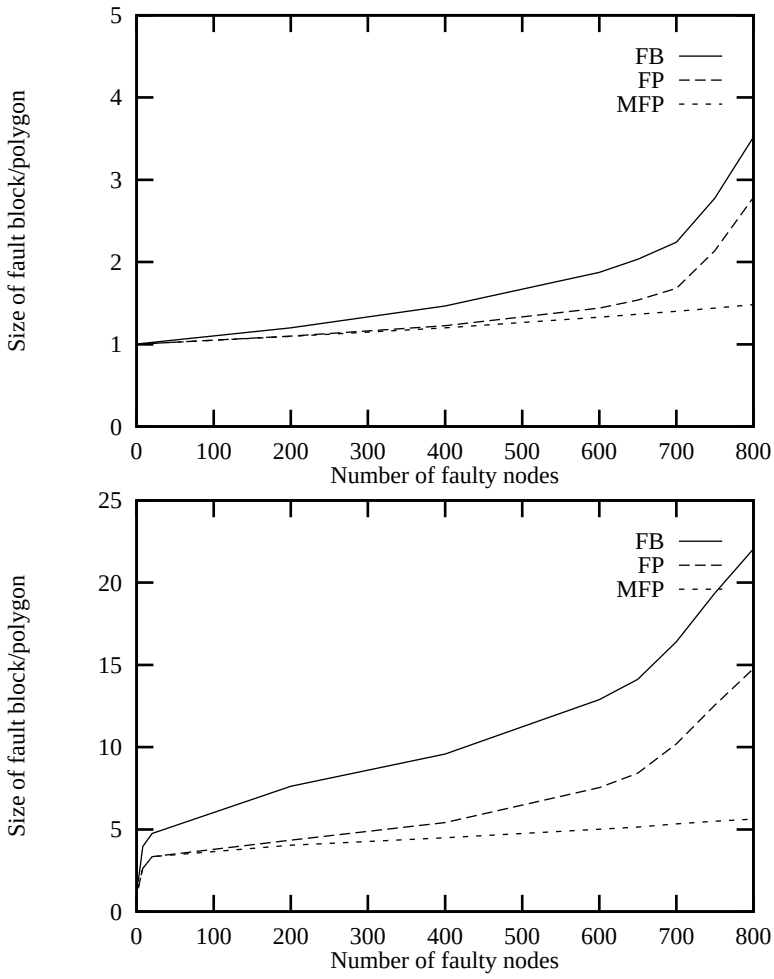

Figure 10: Average size of FB, FP, and MFP: (a) the random fault distribution model and (b) under the clustered fault distribution model.

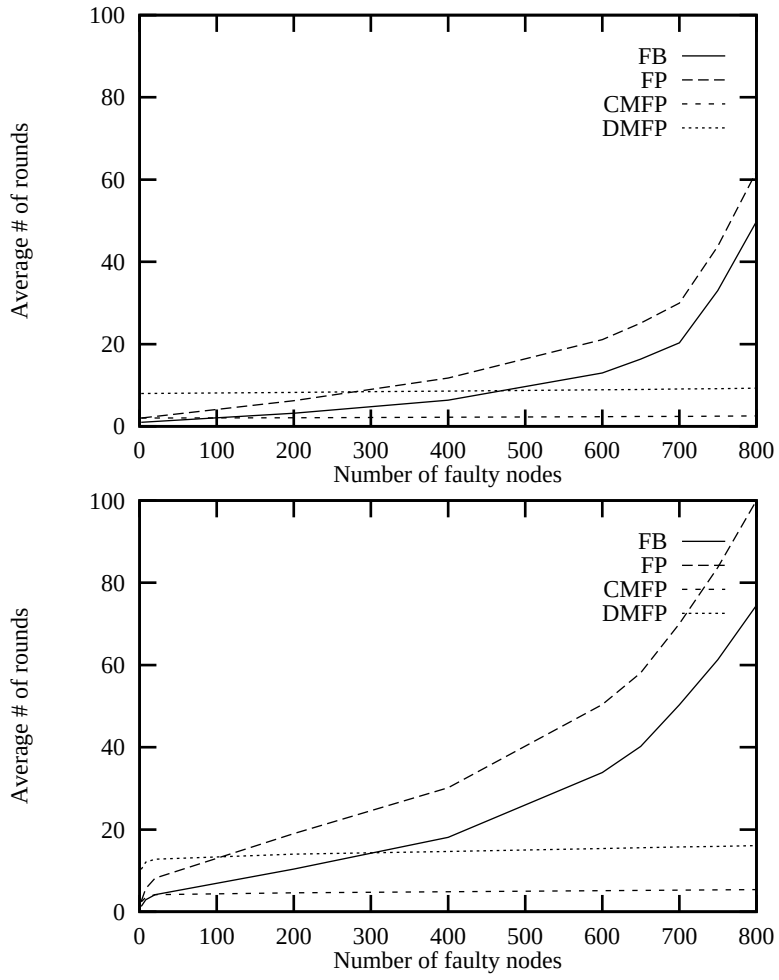

Figure 11: Average number of rounds for status determination under FB, FP, and MFP (the centralized solution CMFP and the distributed solution DMFP): (a) under the random fault distribution model and (b) under the clustered fault distribution model.

in each concave row/column section. Rounds of information propagation are needed in boundary ring construction and the notification process. That is, the number of rounds for status determination of FB or FP depends on the maximum size of faulty blocks and that of MFP (CMFP or DMFP) depends on the maximum size of faulty components. Because in the presence of a large number of faulty nodes, the average size of faulty blocks is significantly larger than that of components, the number of rounds needed needed for MFP (DMFP or CMFP) is much less than that of FB or FP. In MFP, the construction of DMFP needs more rounds than that of CMFP, because the boundary ring construction needs to circle around each faulty component and to notify nodes in the concave region. However, the number of rounds in DMFP is still much less than that of FB or FP. This result confirms the effectiveness of our minimum faulty polygon (MFP) model since the process is more parallel in nature and the status of nodes in MFP can be determined more quickly.

5 Conclusion

In this paper, we provided a solution to an open problem posed in [16]. That is, given a set of faulty nodes, find a set of disjoint orthogonal polygons to cover these faulty nodes such that the number of non-faulty nodes in these orthogonal polygons is minimized. We proposed a centralized solution and its distributed counterpart. Simulation has been conducted to compare the proposed fault model with previous ones, including the rectangular faulty block and sub-optimal orthogonal convex polygon, in terms of the average size of the block and the average number of non-faulty nodes included in the block. Results have shown that the proposed approach can not only find a set of minimum faulty polygons but also does so quickly in terms of the number of rounds in the distributed solution. Our future work will focus on extending the proposed method to higher dimension meshes (tori).

References

[1] A. Agarwal. Limits on interconnection network performance. *IEEE Transactions on Parallel and Distributed Systems.* 2, (4), October 1991, 398-412.

[2] Y. M. Boura and C. R. Das. Fault-tolerant routing in mesh networks. *Proc. of 1995 International Conference on Parallel Processing.* August 1995, I 106- I 109.

[3] S. Chalasani and R. V. Boppana. Communication in multicomputers with nonconvex faults. *IEEE Transactions on Computers.* 46, (5), May 1997, 616-622.

[4] A. A. Chien and J. H. Kim. Planar-adaptive routing: Low-cost adaptive networks for multiprocessors. *Journal of ACM.* 42, (1), January 1995, 91-123.

[5] W. J. Dally. Performance analysis of k-ary n-cube interconnection networks. *IEEE Transactions on Computers*. 39, (6), June 1990, 775-785.

[6] J. Duato, S. Yalamanchili, and L. Ni. *Interconnection Networks: An Engineering Approach*. IEEE Computer Society. 1997.

[7] R. Libeskind-Hadas and E. Brandt. Origin-based fault-tolerant routing in the mesh. *Proc. of the 1st International Symposium on High Performance Computer Architecture*. January 1995, 102-111.

[8] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 1985.

[9] L. Sheng and J. Wu. Maximum-shortest-path (msp) is not optimal for a general n/spl times/n torus. *IEEE Transactions on Reliability*. Vol. 52, No. 1, 2003, 22-25.

[10] J. D. Shih. Adaptive fault-tolerant wormhole routing algorithms for hypercube and mesh interconnection networks. *Proc. of the 11th International Parallel Processing Symposium*. April 1997, 333-340.

[11] C. C. Su and K. G. Shin. Adpative fault-tolerant deadlock-free routing in meshes and hypercubes. *IEEE Transactions on Computers*. 45, (6), June 1996, 672-683.

[12] Y. J. Suh, B. V. Dao, J. Duato, and S. Yalamanchili. Software based fault-tolerant oblivious routing in pipelined networks. *Proc. of the 1995 International Conference on Parallel Processing*. August 1995, I 101 - I 105.

[13] P. H. Sui and S. D. Wang. An improved algorithm for fault-tolerant wormhole routing in meshes. *IEEE Transactions on Computers*. 46, (9), September 1997, 1040-1042.

[14] Y. C. Tseng, M. H. Yang, and T. Y. Juang. An Euler-path-based multicasting model for wormhole-routed networks with multi-destination capability. *Proc. of the 1998 International Conference on Parallel Processing*. August 1998, 366-373.

[15] D. Wang. Minimal-connected-component (MCC) – a refined fault block model for fault-tolerant minimal routing in mesh. *Proc. of IASTED Int'l Conf. Parallel and Distributed Computing and Systems*. Nov. 1999, 95-100.

[16] J. Wu. A distributed formulation of smallest faulty orthogonal convex polygons in 2-d meshes. *Proc. of International Parallel and Distributed Processing Symposium (IPDPS)*. 2001, (CD-ROM).

[17] J. Wu. Fault-tolerant adaptive and minimal routing in mesh-connected multicomputers using extended safety levels. *IEEE Trans. on Parallel and Distributed Systems*. 11, (2), Feb. 2000, 149-159.

[18] J. Wu. A fault-tolerant and deadlock-free routing protocol in 2-d meshes based on odd-even turn model. *IEEE Transactions on Computers*. 52, (9), Sept. 2003, 1154-1169.

[19] J. Wu. Maximum-shortest-path (msp): An optimal routing policy for mesh-connected multicomputers. *IEEE Transactions on Reliability*. Vol. 48, No. 3, 1999, 247-255.

[20] D. Xiang. Fault-tolerant routing in hpercube multicomputers using local safety information. *IEEE Transactions on Parallel and Distributed Systems*. Vol. 12, No. 9, 2001, 942-951.

[21] D. Xiang, A. Chen, and J. Wu. Reliable broadcasting in wormhole-routed hypercube-connected networks using local safety information. *IEEE Transactions on Reliability*. Vol. 52, No. 2, 2003, 245-256.