Fault-Tolerant Array Processors Using Single-Track Switches

SUN-YUAN KUNG, FELLOW, IEEE, SHIANN-NING JEAN, MEMBER, IEEE, AND CHIH-WEI CHANG, MEMBER, IEEE

Abstract—**An array processor is a collection of many similar processing elements (PE's), which can be executed in both parallel and pipeline processing. For the implementation of arrays of large number of processors, fault tolerance has always been a very critical design issue. Very often, spare PE's and switching lattices are incorporated in the array to improve the (fabrication-time) yield and the (run-time) reliability. In this paper, an array grid model based on single-track switches is proposed. A *reconfigurability theorem* is developed to provide the theoretical footing for new reconfiguration algorithms for the fabrication-time and run-time processing. For fabrication-time yield enhancement, the problem of finding a feasible reconfiguration using *global* control can be reformulated as a *maximum independent set* problem. An existing algorithm in graph theory is adopted to solve this problem. The simulations conducted indicate that the algorithm is computationally very efficient; therefore, it may also be applicable to certain run-time fault tolerance. In real-time fault tolerance, the propagation time of data/control signals between the host computer incurred in the global control is often prohibitively long; therefore, only distributed processing is feasible. Based on the same reconfigurability theorem, a distributive reconfiguration algorithm is developed for (asynchronous) array processors. The algorithm has several important features: 1) it is *distributively* executed by the PE's; 2) no global information is required by the individual PE's; 3) the time overhead for reconfiguration is independent of the array size; 4) *transient faults* are handled by retries or by *deactivating/reactivating* the temporarily failed PE. Based on simulations, the performance of the algorithms are evaluated.**

Index Terms—**Array processor, compensation path, fabrication-time fault tolerance, reconfiguration, run-time fault tolerance, transient faults, yield enhancement.**

I. Reconfigurability Theory

VLSI/WSI processor arrays have regular and modular structures which are very suitable for most signal and image processing algorithms. The parallel/pipelined processing capabilities can provide very high computational throughputs in real-time applications. However, in fabrication time, manufacturing defects on wafers are inevitable in today's IC technology, so the yield of a WSI system of a large number of processing elements (PE's) is usually unacceptably poor. Therefore, some fabrication-time fault-tolerance techniques may be employed at the fabrication stage to enhance the *yield*. On the other hand, in run time, it is almost impossible to guarantee such an array to have all the PE's running correctly over the time span of a mission. To combat this problem, run-time fault-tolerance techniques must be incorporated into the system at the array design stage. The objective now becomes *reliability improvement*. Since the fault characteristics and processing times between fabrication-time and run-time fault-tolerant design are very different [1], different schemes should be adopted.

All algorithms discussed in the paper are based on the *mesh-interconnection network*. Two popular mesh architectures are systolic and wavefront arrays [12], [14]. They feature the important properties of modularity, regularity, local interconnection, and a high degree of pipelining. The data movements in the systolic arrays are controlled by globally clocked "beats." On the contrary, the wavefront array does not require global synchronization; instead, each PE has its own local clock (self-timed) and exchanges data with neighboring PE's by asynchronous handshaking. This opens up new flexibilities in fault-tolerant designs [16].

In Section I, a single-track switch model is described. A *reconfigurability* theorem is established to provide the theoretical footing for the new reconfiguration algorithms proposed in the subsequent sections. In Section II, the problem of finding a feasible reconfiguration using *global control* is formulated as a *maximum independent set* problem. An existing algorithm in graph theory is adopted to solve this problem. The simulations conducted indicate that the algorithm is computationally very efficient; therefore, it may also be very suitable for certain run-time fault tolerance. Furthermore, our approach can be extended to effectively deal with failures of switches/wires/connections. In Section III, we consider the *reconfiguration* of a wavefront array in real-time environment. Based on the reconfigurability theorem, a reconfiguration algorithm is proposed. The algorithm can be *distributively* executed by all PE's without the broadcasting of global information and the time overhead of the algorithm is independent of the array size.

A. Array Grid Model

To overcome the defects (or operational faults) in an array processor, a common approach involves incorporating spare PE's and flexible interconnection structure into the array [6], [29], [23], [5], [13], [18], [25], [22], [15], [19], [17], [26],

Manuscript received July 11, 1988; revised November 8, 1988. This work was supported in part by the National Science Foundation under Grant MIP-87-14689, and by the Innovative Science and Technology Office of the Strategic Defense Initiative Organization, administered through the Office of Naval Research under Contracts N00014-85-K-4069 and N00014-85-K-0599.

S.-Y. Kung is with the Department of Electrical Engineering, Princeton University, Princeton, NJ 08544.

S.-N. Jean is with the Department of Computer Science and Engineering, Wright State University, Dayton, OH.

C.-W. Chang is with Pyramid Technology, Mountain View, CA.

IEEE Log Number 8826381.

[20], [28]. To develop a theoretical analysis of the reconfiguration algorithm, we introduce two kinds of arrays: *physical array* and *logical array*. The *physical array* is the array of PE's which may be contaminated by manufacturing defects (or by operational faults). The *logical array* on the other hand represents the desired array structure, as specified by the intended application. The objective of the fault-tolerance design is to maximize the probability that the desired logical array may be mapped to the healthy PE's in the physical array.

The physical array used in our discussion consists of PE's, double-row-column spares, and switches as shown in Fig. 1, with the following assumptions.

1) Faulty PE's can be converted into connecting elements. This assumption is adopted in many designs [10], [13], [25].

2) The switching elements and the interconnection wirings are fault-free.

These assumptions are justifiable by claiming that the switches/wires use much less hardware as compared to the PE's so they are less vulnerable to faults. In Section II-C, our algorithm will be extended to cover the situations of switch/wire/connection failures.

Our scheme is different from previous works in that only *single-track* switches are used. Here single-track means *only one communication path along each horizontal/vertical channel*. (The communication path includes both the data and control signals between two adjacent PE's.) Note that the connecting function of PE's actually provides one more track *within PE's*, which helps enhance the reconfiguration capability. The switching functions provided by a single-track switch are shown in Fig. 1(b).

Previous works on this problem tend to use multitrack switches (or multiplexers) (e.g., six-track in [24] and [25]) and a centralized global reconfiguration scheme to improve the yield and/or array reliability. The simplicity of the single-track switch not only saves area but also makes the assumption of fault-free switches and interconnection wiring more realistic. Note that the major area saved is not for the control part of the switches but for the communication channels. For example, the one-track switch chip implemented with 3.5 μm NMOS technology in [8]. In the total area of 600 μm $\times$ 822 μm, over half of it is used for the 8-bit data "track."

B. Reconfiguration Theory

Throughout this paper, the logical array is indexed from [1, 1] to $[N, M]$ (including the four corner PE's) and the physical array is from [0, 0] to $[N + 1, M + 1]$ (excluding the four corner PE's), cf., Fig. 1. Reconfiguration in array processors comprises two tasks 1) *placement* and 2) *routing*.

1) *Placement:* A *placement* is defined by a one-to-one (but not onto) function $P(\cdot, \cdot)$ which maps all the logical indexes to (healthy) PE's on the physical array. A mapping $P(i, j) = (x, y)$ means that the physical PE (x, y) is labeled with a logic index (i, j). Here the vertical axis uses x (i) index and the horizontal axis uses y (j) index.

2) *Routing:* On the physical array, a path linking $P(i, j)$ and $P(i + 1, j)$ is called a *vertical link* and a path linking $P(i, j)$ and $P(i, j + 1)$ is a *horizontal link*. A valid *routing* means the establishment of *all* the horizontal links and vertical links

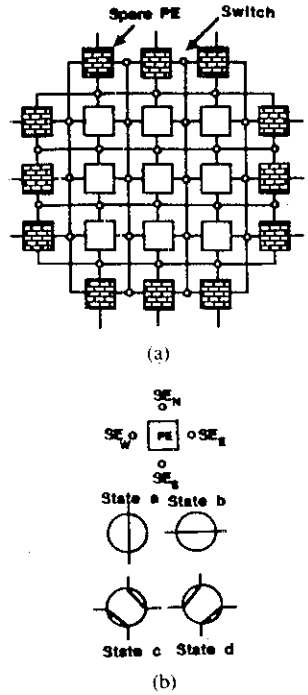


Fig. 1. (a) The physical array with double-row-column spare PE's, and (b) the switch functions.

(for $i = 1, \cdots, M - 1, j = 1, \cdots, N - 1$) on the physical array.

Definition of Compensation Path: If no fault occurs, then the logical indexes for all the physical PE's are the same as their physical indexes, i.e., $(i, j) = (x, y)$. If a nonspare PE, (x, y), is detected to be faulty, then it may be replaced by a healthy PE at say (x', y'), which will in turn be replaced again by another PE at say (x'', y''), and so on. Eventually this replacement process will terminate when a spare PE is used. This process defines the *compensation path* as the ordered sequence of physical PE's, (x, y), (x', y'), (x'', y''), $\cdots$, involved in the chain of reaction. Obviously, there should be as many compensation paths as faulty PE's. These compensation paths have to be mutually exclusive, since a PE cannot be used to replace more than one other PE. It can be shown that the placement mapping function may be *uniquely* defined based on the compensation paths as the following:

- For the PE's on a compensation path: Starting with a (faulty) PE(x, y), $P(x, y) = (x', y')$, $P(x', y') = (x'', y'')$, $\cdots$ etc. Note that the *faulty* PE(x, y) is not mapped onto by any logical index, so it is also an *unassigned* PE.
- If a nonspare PE, say at (x, y), is not on any compensation path, then the mapping will be $P(x, y) = (x, y)$.

It will become clear later that, regarding reconfiguration, only *straight* compensation paths are of interest. For each nonspare faulty PE with physical location (x, y), there are four possible *straight* compensation paths. The South, North, East, and West compensation paths are denoted as $[x^+, y]$, $[x^-, y]$, $[x, y^+]$, and $[x, y^-]$, respectively. For example, the *east* compensation path, $[x, y^+]$, connects PE's (x, y), $(x, y + 1)$, $\cdots$, and $(x, M + 1)$ (see Fig. 2), and the *west* compensation path $[x, y^-]$ connects PE's (x, y), $(x, y - 1)$, $\cdots$, and $(x, 0)$.

Definition of "Near-Miss": Two horizontal compensation

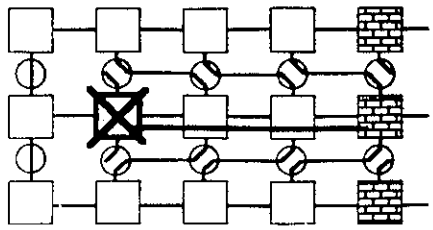


Fig. 2. Single faulty PE with an east compensation path.

paths, $[x_1, y_1^+]$ and $[x_2, y_2^-]$, are declared "near-miss" if $|x_1 - x_2| = 1$ and $y_1 < y_2$ (see Fig. 3). Similar definition goes for two opposite vertical compensation paths. All the other situations are not considered "near-miss."

Reconfigurability Theorem: Given an $(N + 2) \times (M + 2)$ physical array, it is reconfigurable into an $N \times M$ logical array using one-track routing if 1) there exists a set of continuous[1] and straight compensation paths covering all the faulty PE's and 2) there is neither intersection nor "near-miss" among the compensation paths.

The proof of this theorem is discussed in the Appendix. The theorem allows us to select the compensation paths and the corresponding placement so that routing on the grid model is possible. The theory is proved in a constructive way. This means that, once the compensation paths are determined, the corresponding routing scheme can be constructed.

Routing Scheme: The routing is determined by the switching states in the *switching elements* (SE), which are solely determined by the compensation paths. There are two kinds of switches. The first kind, denoted as SW1, is located between two vertical neighboring PE's (i.e., SE_N and SE_S). The other kind, denoted as SW2, is located between two horizontal neighboring PE's (i.e., SE_E and SE_W). Let us look into the switch control design for SW2 type switches. Such a switch control is determined by the so-called *routing states* of its two neighboring PE's. This state, as defined in Table I, will be called the *vertical routing state* (VRS). The state of the SW2 switch is a simple function of VRS_W and VRS_E as shown in Table II. Similarly, SW1 type switches are controlled by the horizontal routing state (HRS) of its two neighboring PE's.

Corollary: With the routing as shown in Fig. 15, the maximum length of horizontal (or vertical) links $\leq 3W_{PE} + 3W_T$, where W_{PE} and W_T are the width of PE and track, respectively. (Assume that links start (and end) at the center of PE's. If links start and end at the boundary of PE's, then $2W_{PE} + 3W_T$ should be used instead.)

This corollary guarantees that the restructured inter-PE interconnection of our scheme will not significantly degrade the array performance.

II. Fabrication-Time Reconfiguration for Yield Enhancement

A. Centralized Reconfiguration Algorithm via Contradiction Graph

Given a defect distribution, it is most important to find a placement which satisfies the condition of the reconfigurability theorem. (The routing scheme can then follow trivially by a table lookup.) To specify the placement, it is sufficient to determine the directions of the compensation paths for all the nonspare faulty PE's. This problem can in turn be reformulated as a well-known problem of finding one *maximum independent* set in graph theory. Then a systematic enumerative method due to Bron and Kerbosch [2] may be employed to solve the problem.

Let us denote the number of nonspare faulty PE's as F. Since there are four alternative compensation paths for each nonspare faulty PE, the total number of possible choices is 4^F. For example, if $F = 10$, there are more than one million possible choices. However, only a small fraction of them will be valid, i.e., not contradicting the conditions described below.

A compensation path is proclaimed as *noncandidate* (and will not be adopted) if either 1) there is another faulty PE on it or 2) the spare PE is previously utilized [cf., $2S$ in Fig. 4(a)]. All the other compensation paths are *candidate compensation paths*. Among them, the selection has to follow certain rules which are represented by a *contradiction graph*.

Definition of Contradiction Graph: Given a fault pattern, a *contradiction graph* is an undirected graph $G(V, E)$ where V denotes the set of all vertices, each representing a candidate compensation path, and E denotes the set of (contradiction) edges. An edge exists between u and v, where $u, v \in V$, if and

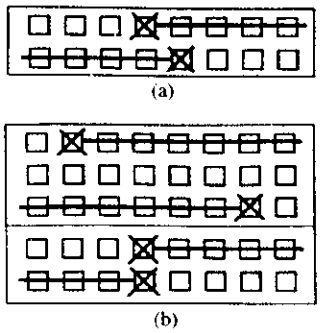


Fig. 3. Two kinds of compensation paths with opposite directions: (a) "near-miss" (b) not considered "near-miss."

TABLE I

0	not on any vertical compensation path
1	not faulty and on a south compensation path
2	faulty and on a south compensation path
3	faulty and on a north compensation path
4	not faulty and on a north compensation path

TABLE II

$VRS_W \backslash VRS_E$	0	1	2	3	4
0	b	c	c	d	d
1	d	b	d	x	x
2	d	c	x	a	x
3	c	x	a	x	d
4	c	x	x	c	b

[1] More precisely, let $(x, y) = P(i, j)$, then $|x - i| + |y - j| \leq 1$.

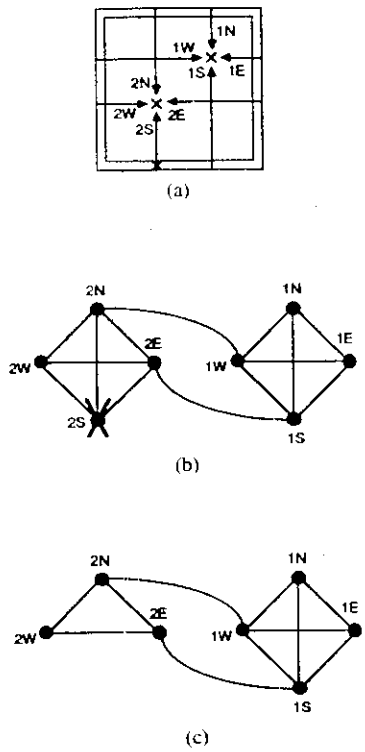


Fig. 4. (a) A fault pattern with three faults, two for nonspare PE's and one for spare PE. (b) The corresponding contradiction graph with a "crossed" vertex $2S$. (c) The reduced equivalent contradiction graph.

only if u and v cannot coexist. Note that u and v cannot coexist if 1) both u and v are for the same faulty PE (since the compensation path for each nonspare faulty PE is unique); or 2) the coexistence of both u and v violates the "nonintersecting" and "nonnear-miss" conditions.

Here we use a simple example to illustrate the contradiction graph. A fault pattern with three faults, two for nonspare PE's and one for spare PE, is shown in Fig. 4(a). The corresponding contradiction graph has two "kites" as shown in Fig. 4(b). In general, the contradiction graph is an F-kite graph, where F is the number of nonspare faulty PE's. Since some compensation paths will be proclaimed as noncandidate in the first place, their corresponding vertices may be "crossed" out. In Fig. 4(b), $2S$ is "crossed" out because there is another faulty PE on the compensation path. The graph is thus reducible to Fig. 4(c).

Graph Terminologies [4]: Given an undirected graph G, an *independent (vertex) set* is a set of vertices of G so that no two vertices of the set are adjacent. An independent set is *maximal* when there is no other independent set that contains it. The set is called *maximal independent set*. If Q is the family of maximal independent sets, then the number $\max_{S \in Q} |S|$ is called the *independence number* of the graph and the set S^* from which it is derived is called the *maximum independent set*.

Corollary: Given a fault pattern in the $(N + 2) \times (M + 2)$ one-track physical array, there exists a valid routing scheme (to support an embedded $N \times M$ logical array) if the independence number of the contradiction graph is F.

An effective method to find *all the maximal independent sets* is proposed by Bron and Kerbosch in [2], which is essentially a tree search method. Based on the above corollary, the problem of finding a placement which satisfies the theorem is equivalent to finding a *maximum* independent set of the contradiction graph with independence number equal to F. Therefore, we modify the testing condition of Bron and Kerbosch's algorithm according to our special need.

The algorithm consists of *forward step* and *backtracking step*. In a forward step, say at the stage k, an independent vertex set S_k is augmented by another proper vertex to produce a new independent set S_{k+1}. Under a smoothest condition, the forward step continues until the set becomes a *maximal* independent set. If the size of the set is not F, then the backtracking step has to be adopted. For this, two intermediate sets, Q_k^- and Q_k^+, are introduced for the algorithm.

Placement Procedure: The procedure is detailed in Fig. 5.

B. Simulations

To estimate the yield of the proposed model and algorithm, Monte Carlo simulations are performed for 12×12, 17×17, and 22×22 physical arrays. Here all the PE's are assumed to have independent failures.

It is computationally advantageous to further reduce the number of vertices in the contradiction graph. The following "screening" procedure is very useful for this purpose. Note that a vertex must be adopted if 1) all the other three vertices in the same kite are already crossed out; or 2) it is not connected to any vertices in another kite. Once a vertex is adopted, all its adjacent vertices should also be crossed-out. (Otherwise, contradiction could occur.) Such a "screening" process [9] may be recursively executed to cross out a large number of vertices in the contradiction graph. Fig. 6(a) shows the average CPU time for the reconfiguration of one fault pattern in a SUN/4-280 workstation (with screening). In all the simulations, the reconfiguration for each fault pattern consumes in average less than 100 ms. The array yields are summarized in Fig. 6(b) and (c). Note that the algorithm provides very good enhancement for smaller array sizes. The results also suggest that the partitioning scheme (cf., Section II-D) may be desirable for larger arrays.

Fault Patterns with Clustering: Defects on a wafer (or a chip) are sometimes clustered [30]. If clusters spread over multiple PE's, then the array yield may degrade. This case has been studied by our simulation. In which, we follow the idea in [31] to generate clustered PE fault patterns, where the degree of clustering is controlled by two parameters, σ_1 and σ_2. The simulation program first generates a fault pattern based on independent PE failures (with σ_1 as the probability of PE failure). Starting with this pattern, the program converts a nonfaulty PE at (x, y) to a faulty PE with a probability of $\sigma_1 + (\nu(x, y) \times \sigma_2)$, where $\nu(x, y)$ is the number of immediate faulty neighbor PE's of (x, y). This process will be repeated until a given number of faulty PE's is generated. Array yields for such kinds of clustered PE faults are simulated and the results are given in Fig. 6(d).

```
procedure MAX-INDEPENDENT(V, Γ)
    /* V: vertex set; Γ(v): the set of vertices adjacent to the vertex v */
    Initialization: Set S_0 = Q_0^- = φ, Q_0^+ = V, k = 0
    Forward:
        CHOOSE-ONE-VERTEX-AND-ADD-TO-S
        Set Q_{k+1}^- = Q_k^- - Γ(v_k) and Q_{k+1}^+ = Q_k^+ - Γ(v_k) - {v_k}
        Set k = k + 1
    Test:
        If (|S_k| + |Q_k^+| < F) or (∃v ∈ Q_k^-, s.t. Γ(v) ∩ Q_k^+ = φ), go to Backtracking
        else if (Q_k^+ ≠ φ) go to Forward
        else if (Q_k^- ≠ φ) or (|S_k| ≠ F) go to Backtracking
        else Print out the solution, S_k, and Stop
    Backtracking:
        If (k = 0) and (Q_0^+ = φ) Stop
        /* All maximal independent sets have been tested */
        Set k = k - 1 and Remove v_k from S_{k+1} to produce S_k
        Retrieve Q_k^- and Q_k^+, Remove v_k from Q_k^+ and add it to Q_k^-
        If (k = 0) and (Q_0^+ = φ) Stop
        /* All maximal independent sets have been tested */
        else go to Test
end MAX-INDEPENDENT;

procedure CHOOSE-ONE-VERTEX-AND-ADD-TO-S
    If (Q_k^- = φ) Choose v_k randomly from Q_k^+
    else
        For all v ∈ Q_k^-, find a v* which minimizes |Γ(v) ∩ Q_k^+|
        Choose v_k randomly from the set Γ(v*) ∩ Q_k^+
        /* Such a choice tends to avoid unnecessary tree search */
    Add v_k to S_k to form a new set S_{k+1}
end CHOOSE-ONE-VERTEX-AND-ADD-TO-S;
```

Fig. 5. Maximum independent set procedure.

C. Reconfigurability Under Failures of Switches, Connections, and Wires

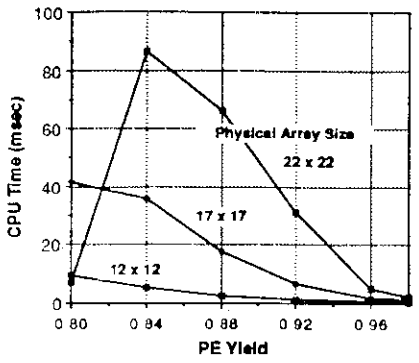
The reconfiguration algorithm can be modified to handle failures of switches, connections, and wires. Our approach hinges upon a modified usage of the contradiction graph according to the switch states given in Table II.

Switch Failure: There are two kinds of switch failures: one is total failure and the other is stuck-at-failure. Let us first focus on the effects of a total switch failure, i.e., all of the four wires connected to switches can no longer be used. Such a switch failure has two effects. 1) The two immediate neighbor PE's can no longer be utilized, thus they have to be declared faulty. 2) Additional restriction on the types of compensation paths which may be selected to cover the two PE's must be imposed. Suppose that the failed switch is of SW2 type, then (according to Table II) both the compensation paths must be vertical and must be in the same direction. This restriction may be incorporated into the contradiction graph.

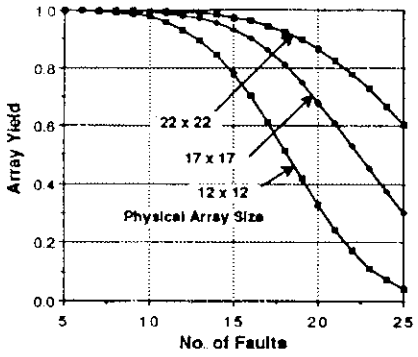
A stuck-at switch failure means that the switch is fixed and all the other switch states are prohibited. (For example, a stuck-at-*b* switch cannot be in state *a*, *c*, or *d*.) The damage of stuck-at switch failures is often more limited. It is not always necessary to declare the two neighboring PE's as faulty. For this, we propose a procedure based on proper modification of the contradiction graph [7]. For example, suppose that there is a (stuck-at-*b*) switch failure in addition to two nonspare faulty PE's as shown in Fig. 7(a). Since the two nonspare faulty PE's are in neighboring columns, the contradiction graph as shown in Fig. 4(b) should be modified with an additional edge ($1S$, $2N$) since $1S$ and $2N$ are "near-miss." With this as an initial graph, one needs to add three edges as shown by dashed edges in Fig. 7(b) for the additional switch failure. From this contradiction graph, we may get a solution, e.g., $\{1N, 2N\}$, for the array reconfiguration. Note that if the switch is stuck-at-*c*, then $\{1E, 2N\}$ can be used.

Simulations: Monte Carlo simulations are performed to estimate the array yield for fault patterns with both PE and (total) switch faults. Let r denote the ratio of the PE and the switch failure rates. In other words, r = (1 − Yield(PE))/(1 − Yield(switch)). The simulations are performed for r = 50, 100, and 1000, respectively, with the logical array size being 10 × 10. The results are shown in Fig. 8. The simulations reveal how the array yield is degraded by switch failures as compared to perfect switches. They also show that the array yield is very much improved by using the proposed scheme to handle switch failures. In fact, the simulations indicate that a better half of those system failures caused by switch failures can be recovered by the proposed scheme.

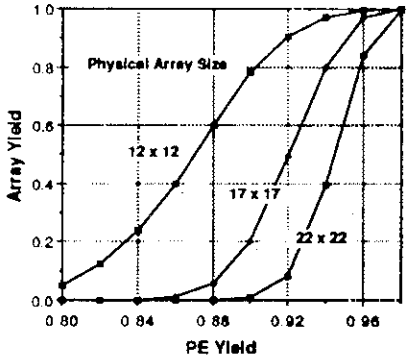
Connection/Wire Failure: Connection failure means that a



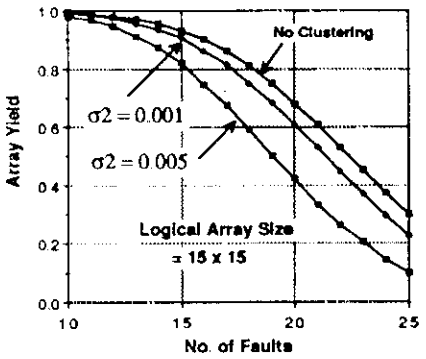
(a)



(b)

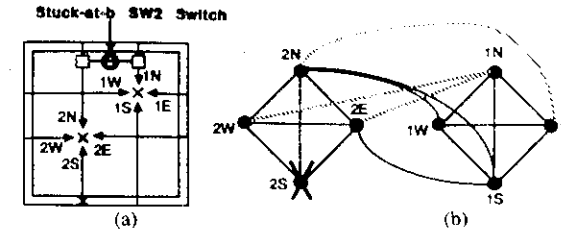


(c)



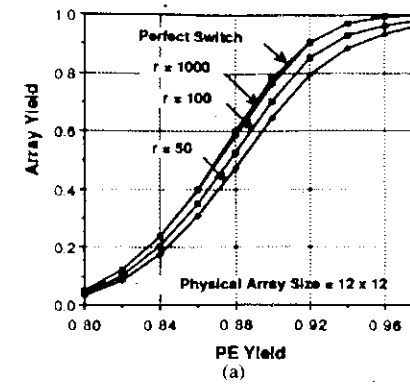
(d)

Fig. 6 (a) The CPU time versus the PE yield. (b) The array yield versus the number of faults. (c) The array yield versus the PE yield. (d) The array yield with clustered faults ($\sigma_1 = 0.001$).

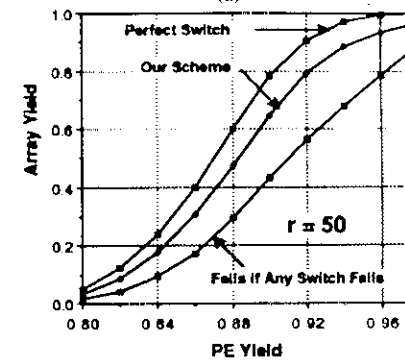


(a) (b)

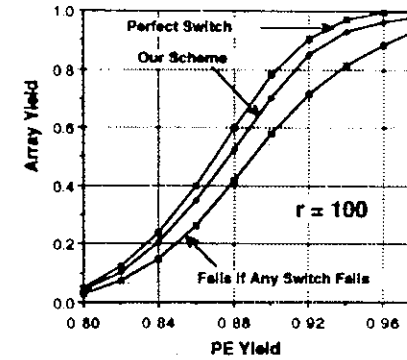
Fig. 7. (a) A fault pattern with a stuck-at-*b* switch fault. (b) The corresponding contradiction graph.



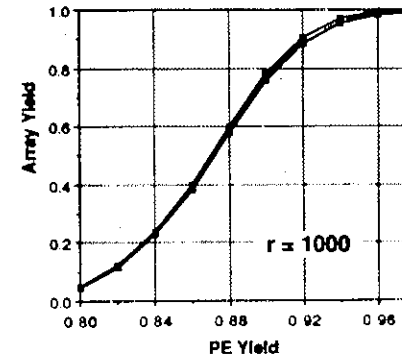
(a)



(b)



(c)



(d)

Fig. 8. Simulation results for PE and (total) switch faults.

faulty PE fails to be converted into a desired connecting element. Again, there are two types: 1) total failure and 2) stuck-at failure. The total failure of a connection (within a faulty PE) has a very severe consequence. It implies the system cannot be reconfigured. This suggests that it pays to "enhance" the hardware for the connection part to minimize such failure. The second type of connection failure is when the faulty PE becomes either stuck-at-vertical or stuck-at-horizontal connecting element. To handle this case, we can again resort to the contradiction graph. For example, if there is one faulty PE and the connection is stuck-at-horizontal, then the vertical compensation paths for the PE should be crossed out.

To tackle the failure of wires, a similar approach (i.e., modifying the contradiction graph according to Table II) may be adopted. If a wire between two switches fails, then one way of arriving at a coarse estimate of the yield is to model the failure in terms of the (stuck-at) switch failures. Another kind of failure is when, say, a *horizontal* wire between a PE and a switch fails. This situation can be shown to be equivalent to the situation that the PE fails and at the same time the PE becomes a stuck-at-vertical connecting element. Therefore, this case can be simulated as the combination of PE fault and stuck-at connection failure.

D. Spare PE Distributions and Partitioning Scheme

Since the spare PE's are used to handle defects, the number of spare PE's used and their distribution can greatly influence the yield. The kinds of spare PE distributions include 1) double-row–column, 2) single-row–column, 3) double-column, and 4) single-column. Tradeoffs between the array reliability and the hardware/software complexity may be made.

Combination of Single-Track Switch and Partitioning Scheme: With single-track switches, the reconfiguration capability is limited and the yield enhancement may not be satisfactory for large arrays. Our approach is to partition a large array (e.g., 32 × 32) into many smaller subarrays (e.g., 8 × 8), each equipped with spare PE's (see Fig. 9). Note that there is only a single spare column between two neighboring subarrays and a spare PE may be used by either (but not both) of the two subarrays. Therefore, the reconfiguration processes for two neighboring subarrays become coupled. This means that some extra edges have to be added to the contradiction graph in the algorithm.

Simulations for Partitioning Scheme: To estimate the yield when extra spares are incorporated with the partitioning scheme, Monte Carlo simulations are adopted. Here all the PE's are assumed to have independent failures. The simulation results are summarized in Fig. 10. The simulations indicate that very good yield enhancement may be achieved with a moderate increase of *redundancy overhead*, which is defined as the number of spares over the number of *logical* PE's.

E. Compile-Time Fault Tolerance

Two kinds of run-time environments need to be distinguished: *compile time* versus *real time*. It is considered a *compile-time* environment if the detection and location of the operational faults and the reconfiguration can be done off-line.

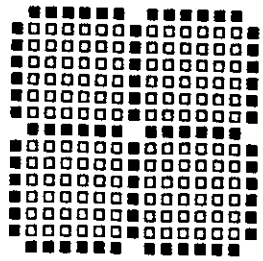
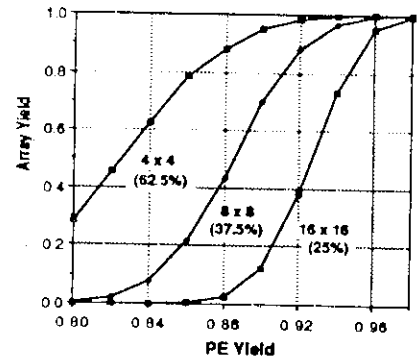
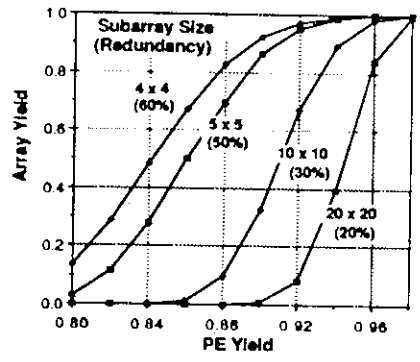


Fig. 9 A partitioned array with spares indicated by black boxes



(a)



(b)

Fig. 10. The array yield with partitioning; the logical array sizes are (a) 16 × 16, and (b) 20 × 20.

Otherwise, it becomes a *real-time* environment, which will be addressed in the next section. The simulations conducted indicate that the algorithm originally proposed for the fabrication-time fault tolerance is computationally very efficient. For example, for all the cases simulated and shown in Fig. 6, the average CPU time for the reconfiguration of one fault pattern takes no more than 100 ms in a SUN/4-280 workstation. Therefore, the same reconfiguration algorithm may also be very suitable for many compile-time reconfiguration applications. To adapt our algorithm to the compile-time environment, we note that the array may have some compensation paths preselected as the result of reconfiguration taken place in the fabrication time. This information can be reflected in the initial contradiction graph (for the compile time) which comprises all the selected compensation paths during the fabrication time. The new "kites" will be added to the initial contradiction graph in run time, and the same algorithm for maximum independent set may be applied.

III. Real-Time Reconfiguration for Reliability Improvement

In this section, a different reconfiguration algorithm based on compensation paths is developed for real-time fault tolerance. For fabrication-time or compile-time fault tolerance, there are no apparent reasons to favor either a systolic array or a wavefront array. However, for the real-time fault tolerance, the wavefront array is superior to the systolic array, since it can cope with on-line reconfigurations due to its asynchronous processing capability.

Reconfiguration in Real Time: One important factor considered in real-time reconfiguration is to minimize the reconfiguration time. To come up with a satisfactory design, we have to resort to wavefront arrays. Its self-timed data-driven property may be exploited to allow the reconfiguration algorithm to be *distributively* executed by all PE's based on local information only. Therefore, the time overhead of the reconfiguration algorithm can be made independent of the array size.

Transient Versus Permanent Faults: The coverage of transient faults is another important issue in real-time fault tolerance. It has been reported that the occurrence of transient faults, mainly due to temporary environmental changes, is about 10 times more frequent than that of permanent faults [27], [21]. To distinguish between transient faults and permanent faults, a typical technique is to perform a fixed number of retry and declare a fault as permanent if it persists beyond an average duration (of transient faults).

A. Distributed Reconfiguration Algorithm

Grid Model: For run-time fault-tolerance design, the *physical array* is considered to be an array which has been successfully tested and delivered by the manufacturer for real mission operations. Operational faults during the mission time are expected, although the operational faults have a much lower probability of occurrence as compared to production defects [11]. The *logical array* again represents the desired array structure corresponding to the application. When operational faults in the physical array occur, the run-time reconfiguration techniques will be applied so that the logical array can be successfully mapped to the *working* PE's in the physical array.

The basic assumptions on the array grid model in Section II are still valid in this section. Here we make an additional assumption that the (built-in) fault detection, the reconfiguration states, and the communication among PE's, are all fault-free.

To achieve real-time reconfiguration, distributive processing is necessary. Therefore, it is important that each individual PE keeps updating the information concerning the reconfigurations incurred in its neighborhood. For distributive routing, the *routing state* discussed in Section II can still be used. For distributive placement, we need to define a *placement state* within each PE to specify the available options of compensation paths to choose from.

Placement State and its Updating: To prevent compensation paths from intersecting and "near-miss," every PE keeps a 4-bit *placement state*. Each bit indicates the availability of one compensation path direction.

In our reconfiguration scheme, both the deactivating and reactivating are considered. The *deactivating* program will be initiated when a PE is declared faulty, while the *reactivating* program will be executed when the failed PE is recovered. The updating of the placement state depends on which program is currently executed.

1) *Deactivating:* Each compensation path for a faulty PE may "block" some future compensation paths. The location and the direction of the newly generated compensation path are propagated to all the other PE's. Based on these data, all PE's update their own placement state. The updating is based on the region it locates with respect to the newly generated compensation path.

2) *Reactivating:* When a faulty PE is recovered, its corresponding compensation path will be cancelled. The cancellation of a compensation path will influence the placement states of the other PE's since some originally "blocked" compensation paths become permissible. To indicate whether a compensation path is allowed, there are four counters used in the scheme for each PE, one for each possible compensation path direction. For example, the south counter records the number of compensation paths which prevent the south compensation path for the PE. These include the horizontal compensation paths which pass directly below the PE and the north compensation paths which "near-miss" with the south compensation path for the PE. The counter is increased (decreased) by the creation (cancellation) of such a special compensation path. When a counter is decreased to zero, the corresponding compensation path becomes allowed and the placement state of the PE should be upgraded.

Distributed Reconfiguration Algorithm: Assume that the *fault detection* is concurrently performed in every PE by means of some *on-line self-testing* circuits, e.g., via duplication of arithmetic and logic units with matching circuits. To avoid unnecessary reconfigurations, a PE failure will not immediately activate the reconfiguration operation. Instead, the *retry* is repeatedly performed by the faulty PE until either the fault disappears or the number of retry exceeds a bound.

1) If the fault disappears, it is viewed as a transient fault and the PE is never considered as faulty. Because of the data-driven property of the wavefront array, the activities of all the neighbor PE's are suspended during the retry period. Once the transient fault recovers, the PE and all the neighbor PE's can continue as normal.

2) If the fault persists beyond a certain number of retries, the PE is declared faulty and the *deactivation* and *reactivation* processes are initiated.

- *Deactivation of a Faulty PE:* When a fault is declared, the faulty PE chooses a compensation path based on the current placement state. If no compensation path can be adopted, the array is declared failed. Otherwise, the faulty PE issues an interrupt to its neighbor PE's and sends out the information about the physical location of the faulty PE and the type of the compensation path. The information is

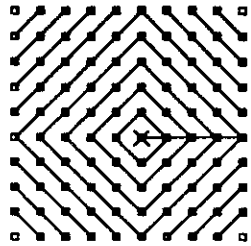


Fig. 11. Wavefronts for the propagation of the fault position and the type of compensation path.

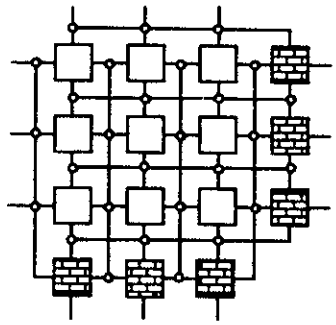


Fig. 12. The physical array with one row and one column of spare PE's.

propagated out not only along the compensation path, but also across almost the entire array similar to an electromagnetic wavefront propagation (see Fig. 11).

a) The PE's not on the compensation path will 1) update their placement states according to the fault position and the type of compensation path and 2) function like a wave media, i.e., interrupt the neighbor PE's and pass information to them.

b) For PE's on the compensation path, in addition to the above tasks, each has the following activities: 1) checking the placement state to see if there are other compensation paths passing through the PE. If yes, the array is declared failed. Otherwise, 2) and 3) are executed. 2) updating its VRS and HRS states (cf., Table I) and thus changes the state of its neighboring switches (cf., Table II), and 3) reassigning the job originally belonging to it to the next PE on the compensation path.

• *Reactivation of a Recovered PE:* Once a fault is declared, the faulty PE becomes a connecting element and enters a dormant state. In the dormant state, the PE tests itself to check its status repeatedly. If the fault is transient and the transient fault is removed, the dormant PE initiates an interrupt to its neighboring PE, takes back the job originally assigned to it, changes its own states (including placement states, HRS, and VRS), and then continues its unfinished work.

B. Example: Single-Row–Column Spare Distribution

For illustration, we apply the above general scheme to the simpler special case of the single-row–column spare distribution (see Fig. 12).

Reconfigurability Theorem for Single-Row–Column Spares: Given an $(N + 1) \times (M + 1)$ one-track physical array, it is reconfigurable into an $N \times M$ logical array if there exists a set of continuous and straight compensation paths covering all of the faulty PE's and they are nonintersecting.

Since this is a special case of the reconfigurability theorem in Section II, the proof is self-evident.

Placement State: Every PE keeps a 2-bit *placement state*, which provides sufficient information to prevent the compensation paths from intersecting.

1) HV: Both horizontal and vertical compensation paths are allowed to pass through this PE. When a PE in this state fails, the shorter compensation path will be chosen to decrease the possibility of future intersection.

2) $\overline{\text{HV}}$: Only horizontal compensation path is allowed to pass through this PE.

3) $\overline{\text{HV}}$: Only vertical compensation path is allowed to pass through this PE.

4) $\overline{\text{HV}}$: No more compensation path is allowed to pass through this PE.

All PE's have the same initial placement state, HV. The updating is based on the region it locates with respect to the newly generated compensation path [see Fig. 13(a)]. The resulting state *transition diagram* is shown in Fig. 13(b) where the dashed lines indicate the state transitions due to fault recoveries. If any PE in the state $\overline{\text{HV}}$ is declared faulty, the system fails.

Routing State: The routing state for single-row–column spares is a special case of what was discussed in Section I-B. Only three VRS states, i.e., 0, 1, and, 2 (see Table I) are required since there is no north compensation path.

C. System Reliability Evaluation

To get some ideas about a real-time fault-tolerance system, let us consider a system with the following parameters.

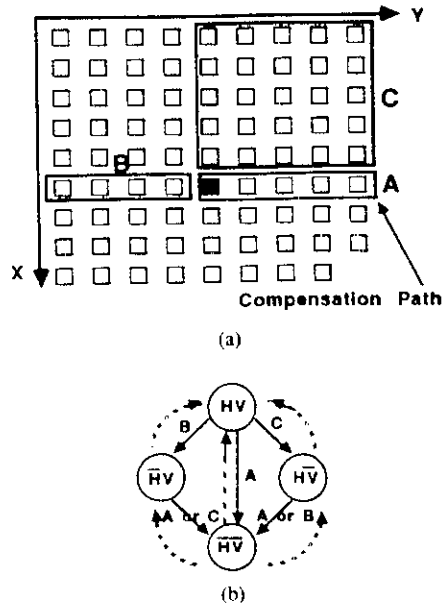


Fig. 13. (a) Placement state updating regions. Similar regions can be defined for a vertical compensation path. (b) State transition diagram.

• The PE computation clock rate is 10 MHz and the system communication clock rate is 100 kHz.

• The failure rate of permanent faults λ is 10^{-6} per second. If faults are assumed to be independent of each other, the PE reliability $r(t)$ $(= e^{-\lambda t})$ is 0.99 in a mission time (t) of 10^4 s. Note that the PE reliability is the probability that a PE works during the mission time given that the PE works initially.

• The failure rate of transient faults is 10 times that of permanent faults. If the transient faults are treated the same as permanent faults in the reconfiguration scheme, the corresponding PE reliability is 0.896 in a mission time of 10^4 s.

• The average duration of transient faults is 10 system communication clocks. That is the recovery rate for transient faults is 10^4 per second.

The following two observations are useful in simplifying our simulation for the system reliability.

• For transient faults the recovery rate (10^4 per second) is much higher than the failure rate (10^{-5} per second). Thus, the system reliability should be approximately the same if we disregard the influence of transient faults which are recovered later. Since in our scheme the recovery of transient faults is by retrying the faulty task, a leakage may be defined as the percentage of transient faults which cannot be recovered by a finite number of retries (and thus is handled as a permanent fault). As an example, if the leakage is 0.5, the equivalent PE reliability is

$$r = e^{-(10^{-6}+0.5\times 10^{-5})10^4} = 0.942.$$

• The failure rate of permanent faults is 10^{-6} per second, or equivalently, 10^{-11} per system communication clock period. Thus, for an array with a moderate size, e.g., 20×20, the probability that a permanent fault occurs during the propagation of the compensation path for another permanent fault is very low.

Monte Carlo Simulation: Assume that all the PE's have independent failures and the same PE reliability r, then the array reliability for a physical array with size $N \times M$ is

$$\sum_{i=0}^{K} \binom{NM}{i} C_i r^{NM-i}(1-r)^i \tag{1}$$

where C_i is the probability that a fault pattern with i faulty PE's does not cause array failure and K is the number of spare PE's. Based on a Monte Carlo simulation, we estimate C_i and compute the array reliability.

The simulation is performed in the following way.

• Since the effect of the *reactivating* process can be incorporated as leakages, only the *deactivating* process is simulated.

• The position of faulty PE's is generated one by one via a random number generator.

• For each fault, the system forms a compensation path. If it is impossible to form a compensation path, the system is failed.

• $C_i = S_i/K$, where S_i is the number of *successful* fault patterns with i faulty PE's and K is the total number of fault patterns. Here a successful fault pattern means the fault pattern does not cause system failure.

The simulation results are summarized in Table III.

Partitioning: With single-track switches, the reconfiguration capability is limited and the array reliability becomes unsatisfactory for large arrays. This problem may be significantly alleviated by adopting the partitioning scheme mentioned in Section II-D. Table IV summarizes the system reliability for different subarray sizes by assuming a 32×32 logical array.

Comparison to Synchronous Schemes: When either a transient or a permanent fault is detected, a systolic/synchronous array must use global interrupt to stop *all* the PE's in order to perform the reconfiguration [3], [15], [25]. Since

TABLE III

Logical Array Size	Without Fault-Tolerance	Deactivation Only	Retry (0.5 Leakage)	Retry (No Leakage)
Equivalent r	0.896	0.896	0.942	0.99
4×4	1.726×10^{-1}	8.289×10^{-1}	9.457×10^{-1}	9.984×10^{-1}
8×8	8.866×10^{-4}	2.670×10^{-1}	6.556×10^{-1}	9.886×10^{-1}
12×12	1.356×10^{-7}	1.901×10^{-2}	2.720×10^{-1}	9.608×10^{-1}
16×16	6.178×10^{-13}	1.734×10^{-4}	5.165×10^{-2}	9.208×10^{-1}

TABLE IV

Logical Subarray Size	Spare PEs Required	System Reliability
2×2	1024	0.9572
4×4	512	0.9695
8×8	256	0.9033
16×16	128	0.7774
32×32	64	0.4994

multiple faults may occur at the same time, a time consuming fault-detection/reconfiguration process is required for host-driven methods [24], [25]. Note that the information transfer between the array and the host is very time-consuming even with the help of global links.

In contrast to the systolic/synchronous array, the (asynchronous) wavefront array does not require global interruption. Each PE can start its computation right after the local reconfiguration is complete without having to wait for the reconfiguration of the whole array to complete. Therefore, the time overhead of our reconfiguration algorithm is independent of the array size. Another advantage of our design is the ability due to its distributed processing to handle multiple faults which occur at the same time. Finally, by avoiding the global link, the scheme offers a reduced hardware cost and increased system reliability. However, the control process for the COPE scheme is more complicated. Furthermore, a larger local memory is required since each PE must be able to store two sets of data/processes, one for its own job and the other for the job of its neighbors.

IV. Conclusion

In our model, the faulty PE's can be converted into connecting elements. This in reality means one additional track *within* the processor elements. (Note that the switch control of a track *within* a PE is easier to handle than those outside a PE.) In this sense, our model is not a "pure" 1-track, instead it may be called a $1\frac{1}{2}$-track model. An important question is, "If a fault pattern is reconfigurable on a $1\frac{1}{2}$-track model, is it always reconfigurable on a pure 2-track array?" In other words, "Can our reconfiguration algorithm for $1\frac{1}{2}$-track arrays be directly applied to "pure" 2-track arrays?" The answers are YES. For example, Fig. 14(a) shows a fault pattern reconfigurable on a $1\frac{1}{2}$-track model. The same pattern is also reconfigurable on a "pure" 2-track model as shown in Fig. 14(b). The basic idea of our proof is the following. (For a detailed proof see [7].) As shown in Fig. 14(b), each vertical (horizontal) channel has two tracks. The left track (resp., the upper track) is reserved exclusively for bypassing a faulty PE while the right track (resp., the bottom track) is used (although not exclusively) for the routing corresponding to inter-PE interconnection. In [26], a pure 2-track WSI array was built in hardware, but no very systematic procedure is available in software. Coincidently, our compensation path reconfiguration algorithm offers a good software solution for the WSI array [26].

In summary, an array grid model based on an enhanced single-track switch is proposed and several reconfiguration algorithms based on the reconfigurability theorem are developed. The proposed reconfiguration algorithm can systematically search all the possible placements and guarantee to find a feasible solution whenever the array is reconfigurable. The algorithm, based on a contradiction graph, can be applied to a normal array or a partitioned array. The algorithm is meant for centralized processing using a host computer. It may also be adopted for compile-time environments. Simulation results indicate very good yield may be delivered.

In real-time fault-tolerance designs, reconfiguration time overhead and handling of transient faults are two important factors. Wavefront arrays offer the asynchronous data-driven computing feature which facilitates fault-tolerant designs for real-time environment. This leads to a different reconfiguration scheme, *distributively* executable by individual PE's. Therefore, the reconfiguration time overhead is independent of the array size. Two methods, retry and *reactivation*, are developed to handle transient faults.

Appendix

Proof of the Reconfigurability Theorem

Reconfigurability Theorem: Given an $(N + 2) \times (M + 2)$ physical array, it is reconfigurable into an $N \times M$ logical array using one-track routing if 1) there exists a set of continuous and straight compensation paths covering all the faulty PE's and 2) there is neither intersection nor "near-miss" among the compensation paths.

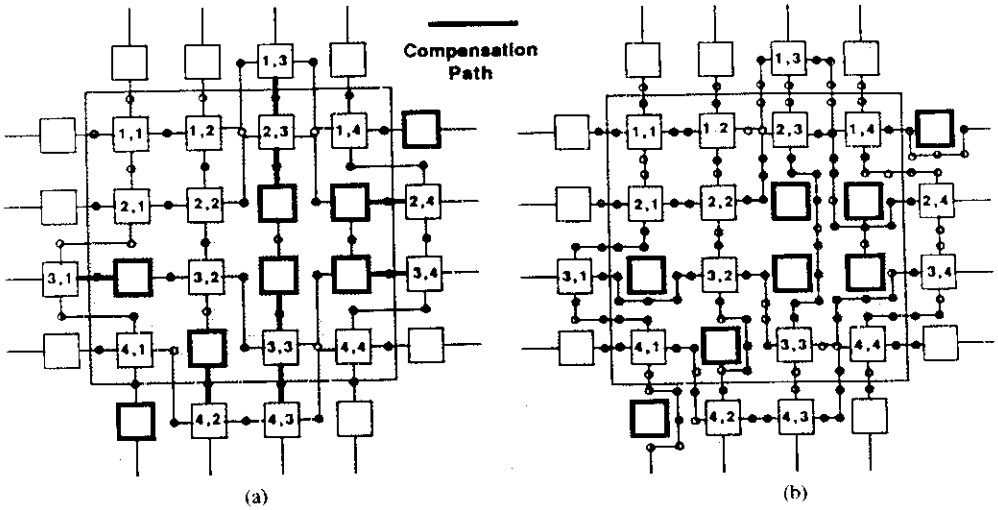


Fig. 14. The routing of a fault pattern (a) in our model, (b) in a two-track array.

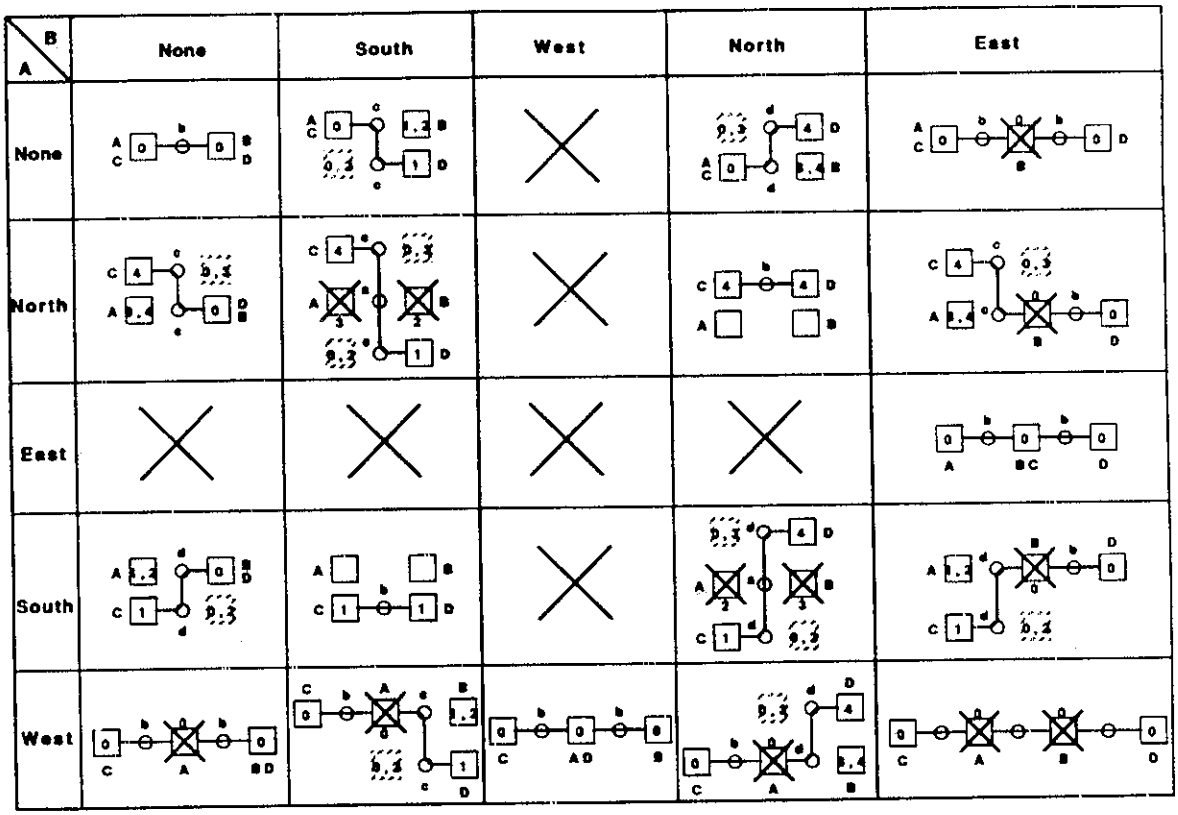


Fig. 15. All the possible horizontal links between C and D according to the compensation path status of A and B. The integers within each PE are the possible VRS of that PE. Lowercase characters a, b, c, and d indicate the switch states.

Proof: A *segment* is defined as the physical communication links between two interconnected switches. There are horizontal segments and vertical segments. We would like to stress that, in our routing scheme, all the vertical segments will be utilized only by the horizontal links and the horizontal segments only by the vertical links.

With the set of compensation paths, a routing scheme can be defined as shown in Tables I and II. We shall prove that, with the routing scheme, all the horizontal links are correctly constructed. Similar proof can be applied to all the vertical links.

Consider the horizontal link between (logical) indexes (i, j) and $(i, j + 1)$. Let A denote the physical PE (i, j) and B the physical PE $(i, j + 1)$. Note that (logical) indexes (i, j) and $(i, j + 1)$ are originally placed at A and B, respectively. Suppose that after the fault pattern occurs, C and D are the new physical PE's where the (logical) indexes (i, j) and $(i, j + 1)$ are placed according to the compensation paths (if any) passing through A and B. The physical processor A (or B) can be either on one of the four (north, south, west, or east) compensation paths or not on any compensation path, i.e., the compensation path status of each PE is unique. There are totally 25 cases about linking C and D. For all the cases, the routing scheme can be applied and the results are summarized in Fig. 15. Note that all the horizontal links between C and D have been correctly constructed.

References

[1] J. A. Abraham and W. K. Fuchs, "Fault and error models for VLSI," *Proc. IEEE*, pp. 639–654, May 1986.

[2] C. Bron and J. Kerbosch, "Algorithm 457—finding all cliques of an undirected graph," *Comm. ACM*, vol. 16, pp. 575–577, 1973.

[3] C. W. Chang, "Real-time fault-tolerant computing for VLSI processor arrays," Ph.D. dissertation, Dep. Elec. Eng., Univ. of Southern California, 1988.

[4] N. Christofides, *Graph Theory: An Algorithmic Approach*. New York: Academic, 1975.

[5] J. W. Greene and A. E. Gamal, "Configuration of VLSI arrays in the presence of defects," *J. ACM*, vol. 31, pp. 694–717, Oct. 1984.

[6] K. S. Hedlund and L. Snyder, "Wafer scale integration of configurable highly parallel (CHIP) processors," in *Proc. Conf. Parallel Processing*, 1982, pp. 262–264.

[7] S. N. Jean, "Mapping/matching algorithms to reconfigurable mesh arrays," Ph.D. dissertation, Univ. of Southern California, Los Angeles, 1988.

[8] J. R. Hwang, "The reconfiguration and switch design of array processors," Master's Thesis, The Institute of Electronics, Chiao Tung University, Taiwan, 1987.

[9] S. N. Jean and S. Y. Kung, "Necessary and sufficient conditions for reconfigurability in single-track switch WSI arrays," in *Proc. Int. Conf. Wafer Scale Integration*, Jan. 1989.

[10] I. Koren, "A reconfigurable and fault-tolerant multiprocessor array," in *Proc. 8th Annu. Symp. Comput. Architecture*, 1981, pp. 425–442.

[11] I. Koren and D. K. Pradhan, "Modeling the effect of redundancy on yield and performance of VLSI systems," *IEEE Trans. Comput.*, pp. 344–355, Mar. 1987.

[12] H. T. Kung and C. E. Leiserson, "Systolic arrays (for VLSI)," in *Proc. Sparse Matrix Symp.*, SIAM, 1978, pp. 256–282.

[13] H. T. Kung and M. S. Lam, "Wafer-scale integration and two-level pipelined implementations of systolic arrays," *J. Parallel Distribut. Comput.*, pp. 32–63, 1984.

[14] S. Y. Kung, K. S. Arun, R. J. Gal-Ezer, and D. V. Bhaskar Rao, "Wavefront array processor: Language, architecture, and applications," *IEEE Trans. Comput.*, pp. 1054–1066, Nov. 1982.

[15] S. Y. Kung, C. W. Chang, and C. W. Jen, "Real-time reconfiguration for fault-tolerant VLSI array processors," in *Proc. Real-Time Syst. Symp.*, Dec. 1986, pp. 46–54.

[16] S. Y. Kung, S. C. Lo, S. N. Jean, and J. N. Hwang, "Wavefront array processors: From concept to implementation," *IEEE Computer*, pp. 18–33, July 1987.

[17] S. Y. Kung, S. N. Jean, and C. W. Chang, "Fabrication-time and run-time fault-tolerant array processors using single-track switches," in *Proc. Int. Workshop Defect Fault Tolerance VLSI Syst.*, 7.2, Oct. 1988.

[18] T. Leighton and C. E. Leiserson, "Wafer-scale integration of systolic arrays," *IEEE Trans. Comput.*, pp. 448–461, May 1985.

[19] F. Lombardi, R. Negrini, M. G. Sami, and R. Stefanelli, "Reconfiguration of VLSI arrays: A covering approach," in *Proc. FTCS*, 1987, pp. 251–256.

[20] L. Jervis, F. Lombardi, and D. Sciuto, "Orthogonal mapping: A reconfiguration strategy for fault tolerant VLSI/WSI 2-D arrays," in *Proc. Int. Workshop Defect Fault Tolerance VLSI Syst.*, 7.4, Oct. 1988.

[21] S. R. McConnel, D. P. Siewiorek, and M. M. Tsao, "The measurement and analysis of transient errors in digital computer systems," in *Proc. IEEE Fault-Tolerant Comput. Symp.*, 1979, pp. 67–70.

[22] W. R. Moore, "A review of fault-tolerant techniques for the enhancement of integrated circuit yield," *Proc. IEEE*, pp. 684–698, May 1986.

[23] A. L. Rosenberg, "The Diogenes approach to testable fault-tolerant arrays of processors," *IEEE Trans. Comput.*, pp. 902–910, Oct. 1983.

[24] M. Sami and R. Stefanelli, "Fault-tolerance and functional reconfiguration in VLSI arrays," in *Proc. Int. Conf. Circuits Syst.*, 1986, pp. 643–648.

[25] ——, "Reconfigurable architectures for VLSI processing arrays," *Proc. IEEE*, pp. 712–722, May 1986.

[26] G. Saucier, J-L. Patry, E-F Kouka, T. Midwinter, P. Ivey, M. Huch, and M. Glesner, "Defect tolerance in a wafer scale array for image processing," in *Proc. Int. Workshop Defect Fault Tolerance in VLSI Syst.*, 8.2, Oct. 1988.

[27] D. P. Siewiorek and R. S. Swarz, *The Theory and Practice of Reliable System Design*. Bedford, MA: Digital, 1982.

[28] A. D. Singh, "Interstitial redundancy: An area efficient fault tolerance scheme for large area VLSI processor arrays," *IEEE Trans. Comput.*, vol. 37, pp. 1398–1410, Nov. 1988.

[29] L. Snyder, "Introduction to the configurable, highly parallel computer," *IEEE Computer*, vol. 15, pp. 47–56, Jan. 1982.

[30] C. H. Stapper, F. M. Armstrong, and K. Saji, "Integrated circuit yield statistics," *Proc. IEEE*, vol. 71, pp. 453–470, Apr. 1983.

[31] C. H. Stapper, "Block alignment: A method for increasing the yield of memory chips that are partially good," in *Proc. Int. Workshop on Defect Fault Tolerance VLSI Syst.*, 6.3, Oct. 1988.

Sun-Yuan Kung (S'74–M'78–SM'84–F'88) received the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA, in 1977.

In 1974, he was associated with the Amdahl Corporation, Sunnyvale, CA as an Associate Engineer in LSI design and simulation. In 1977, he joined the faculty of Electrical Engineering-Systems, University of Southern California, Los Angeles, where he became a Processor in 1986. In 1984, he was a Visiting Professor at Stanford University and a Visiting Professor at the Delft University of Technology, The Netherlands. Since 1987, he has been with the faculty of Department of Electrical Engineering, Princeton University, Princeton, NJ, as a Professor. His research interests include linear systems approximation, modern spectrum analysis, digital signal processing, and VLSI array processors.

From 1984 to 1985, he was the Associate Editor for the VLSI area in the IEEE TRANSACTIONS ON ACOUSTICS, SPEECH, AND SIGNAL PROCESSING and currently serves on the IEEE ASSP Technical Committee on VLSI. He was the Keynote Speaker for the First International Workshop on Systolic Arrays, Oxford, July 1986. He served as the General Chairman of the IEEE Workshops on VLSI Signal Processing from 1982 in 1986 and was Program

Chairman of the International Conference on Systolic Arrays, 1988. He is the author of a textbook *VLSI Array Processors* (Englewood Cliffs, NJ: Prentice-Hall, 1988).

Shiann-Ning Jean (S'82–M'89) received the B.S. and M.S. degrees from the National Taiwan University, Taiwan, in 1981 and 1983, respectively, and the Ph.D. degree from the University of Southern California, Los Angeles, in 1988, all in electrical engineering.

In 1988, he was a visiting student at Princeton University, Princeton, NJ. Currently, he is an Assistant Professor in the Computer Science and Engineering Department of Wright State University, Dayton, OH. His research interests include parallel computer architecture, fault-tolerant computing, and digital signal processing.

Chih-Wei Chang (S'83–M'88) received the B.S.E.E. degree from Chung-Yuan University, Taiwan, R.O.C., in 1977, the M.S.E.E. degree from Tatung Institute of Technology, Taiwan, R.O.C., in 1979, and the M.S. degree in computer engineering and the Ph.D. degree in electrical engineering from the University of Southern California, Los Angeles, in 1984 and 1988, respectively.

He is currently a Senior Staff Engineer at Pyramid Technology, Mountain View, CA, where he is working on RISC-based multiprocessor design. From 1985 to 1988 he was a project engineer at Quotron Systems/Citicorp, Los Angeles, CA, where he worked on tightly-coupled multiprocessors. During 1981–1982, he was employed by Sinotek, Taiwan, as a project leader, designing computerized toll systems. His research interests include computer architecture, parallel processing, fault-tolerant computing, and VLSI/CAD.