Restructuring of Square Processor Arrays by Built-in Self-Repair Circuit

Pinaki Mazumder, *Member, IEEE,* and Yih-Shyr Jih, *Member, IEEE*

***Abstract*—This paper introduces two types of neural networks that can be integrated into array-type VLSI/WSI chips to restructure themselves automatically so that, in the presence of multiple random faults, the array circuits can correctly compute their tasks. In the first implementation, the neural network is interconnected and programmed such that it can readily execute a *maximum matching* algorithm and thereby it can substitute the faulty components by appropriate fault-free spare elements. The second implementation rearranges the surviving fault-free processors to restore the logical structure, and it can adjust its interconnection complexity based on the quality of solution (i.e., performance) desired. These two approaches have been compared with the traditional reconfiguration algorithms, and by simulation it is shown that the proposed neural net techniques provide superior quality performance (i.e., higher survivability rates). It is also shown that the intrinsic fault-tolerant nature of neural networks maintains a degradable reconfiguration control even in the presence of faulty neural network components. The speed of neural networks provides an added advantage for on-line reconfiguration, where the chip can be repaired quickly by itself, thus reducing the system down-time (unavailability).**

I. Introduction

A LARGE number of scientific and signal processing problems are frequently computed in an orderly and pipelined fashion by replicated hardware similar to iterative logic or processor arrays, where each processing element is connected to a fixed number of physically adjacent processing elements in such a way that the overall interconnection structure describes a regular geometry. One vulnerable feature of such array networks is that, even if a single processing element fails to perform its intended function correctly due to some physical defects, the entire computation may become erroneous. To restore the correct computation capabilities of the array, it must be restructured appropriately so that the defective processing elements are eliminated from the computation paths, and the working processing elements maintain correct logical connectivities between themselves. Various reconfiguration schemes have been described in the literature to diagnose and restructure a faulty physical array into the fault-free target logical structure [2], [3], [11], [22], [29]. Some of these techniques employ very powerful reconfiguration schemes that can repair a faulty array with almost certainty, even in the presence of clusters of multiple faults. Nevertheless, the key limitation of these powerful reconnection algorithms is that they are written in software programs to run on an external host computer, and they cannot be implemented efficiently within a processor array chip by using special-purpose circuits. This paper shows how self-reconfiguration (self-repair) can be done by exploiting the intelligence and combinatorial optimization capability of a built-in electronic neural network. This automatic self-repair helps reduce system down-time and makes reconfiguration possible when the processor array is embedded within a VLSI chip such that the fault pattern cannot be monitored externally through the boundary pins of the chip.

A first attempt for developing self-reconfiguration schemes was due to Negrini *et al.* [23], and Sami and Stefanelli [26]. The design approach of their repair control scheme begins with a heuristic algorithm that involves only some simple local reconnection operations. The redundant interconnection circuits are then derived accordingly to support the chosen control scheme. Such control scheme may fail to successfully exploit the full potentialities of the interconnection network for finding a reconfiguration path which requires the knowledge of the global structure of the array network. Consequently, a partially damaged processor array, which is certainly *repairable*, may frequently be considered unrepairable by a control scheme similar to theirs.

Conversely, in the proposed neural network reconfiguration scheme, the neural net is distributed over all processing elements to gather the complete defect pattern, and, through massive associative interactions between its neurons, it derives globally optimized repair solutions. Through simulation studies, it is shown here that the probability for a successful repair on an array with random defects significantly improves, if the neural net is distributed over the entire processor array as described herein.

Another advantage of the use of a built-in neural network is that it is intrinsically fault-tolerant and robust in the sense that it only gracefully degrades in performing its intended function, even in the presence of multiple faulty neural network circuit elements. The present authors made extensive simulations in their earlier work on

Manuscript received November 7, 1990; revised December 28, 1992. This paper was recommended by Associate Editor J. Allen. This work was partially supported by the National Science Foundation under Grant MIPS 9013092.

The authors are with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109.

IEEE Log Number 9207792.

memory self-repair by a built-in neural network, and they showed that the network's capabilities in finding a repairable solution degrades very slowly if multiple synaptic resistors fail or biasing circuits slightly change [20]. The network degrades rather rapidly with the growing number of faulty neurons whose outputs are stuck permanently to either a firing or nonfiring state. In contrast to the neural restructuring technique, a self-reconfiguration scheme similar to [23] has a potential disadvantage in this respect, because its simple logic design scheme lacks intelligence and will require high overhead if it is designed for a comparable fault-tolerance capability.

Finally, the overhead of a neural network in electronic implementation is small. A neuron processor can be as simple as a pair of inverters, and a synaptic link can be emulated by a transistor. The basic device switching time is only a few nanoseconds, and because the neurons in the entire network interact to alter their states concurrently, the reconfiguration task can be performed very quickly.

The rest of this paper discusses the neural-net-controlled self-repair approach, and is organized as follows. In Section II, a method suitable for neural computing, called *Direct Substitution*, is proposed. To study the feasibility of the proposed built-in self-repair (BISR) scheme, the neural network has been implemented on an 8×8 image processing processor array. The overhead complexities involved in the redundant interconnections and the neural nets are also analyzed. Comparisons are made with some conventional approaches to demonstrate the effectiveness of the neural network self-repair method. In Section III, another neural network self-repair design, called *Window Substitution*, is presented as a possible improvement over the Direct Substitution method. Finally, the neural net reconfiguration schemes are evaluated further with faulty neural net components, and the overall yield is estimated.

II. Direct Substitution Method

In random access memory repair, columns or rows containing defective memory elements are replaced by spare columns and rows [19], [27], [32]. For some array processors, such as the Massive Parallel Processor by Goodyear Aerospace [1], faulty processor columns are also replaced in entirety by spare columns. In contrast, this work concentrates on individual substitutions that bypass faulty cells and maximize the utilization of fault-free processor elements.

Some repair algorithms embed certain logical structures, such as linear arrays, rectangular arrays, and trees, into the original array having defective components [4], [8], [12], [25]. In this work, we limit ourselves to the repair of faulty arrays, where the defective components are carefully bypassed to restore the rectangular target array. The embedding of other logical structures can be done on the restored rectangular logical arrays, if needed.

Koren has designed a strategy such that reconnections are done by passing messages through multistage paths between adjacent processors [12]. In the proposed design, a dedicated physical connection between a pair of neighboring elements is always maintained to ensure fast interprocessor communications.

This study concentrates on the reconfiguration of two-dimensional (2-D) square meshes with *one row* and *one column* of redundant elements. Usually the masking complexities of interconnects are much simpler than those of processing elements and also the interconnects fail largely due to pinhole defects which can be controlled to a large extent in the modern processing techniques. On the contrary, the processing elements employ complex logic circuits where the materials and processing defects can easily manifest themselves into timing, parametric, or functional defects. Interconnect defects can be accommodated into our fault model, but to retain the simplicity of our design approach, we assume that only the processors can be defective in our model.

A. Interconnections Design

Let the size of a 2-D array be $m \times n$. The element at the intersection of row i and column j is denoted by (i, j), $1 \leq i \leq m$ and $1 \leq j \leq n$, with the spare elements located in row 0 and column 0. By direct substitution, we mean that a spare element can replace any single faulty element in its row or column. For example, a faulty element (i, j) can be bypassed horizontally by identifying each of its input-outputs with the corresponding one of $(i, 0)$. Similarly, a faulty element (i, j) can be bypassed vertically by identifying each of its input-outputs with the corresponding one of $(0, j)$. The physical locations of processors and the interconnections are illustrated in Fig. 1 by a 2×2 processor array.

Let h and w be the height and width of a processor element, respectively. We assume that the distance between two adjacent processors is negligible compared with the dimensions of the processors. The total length of the links is $4nh + 4mw$, which increases linearly with the dimensions of the processor array and the processor element, while the total number of 3×3 switch boxes is $(n + 1)(m + 1)$. One disadvantage of this redundant connection scheme is, for the worst case, that the signal line length between two adjacent elements is proportional to the number of elements in a row or column. An obvious way to reduce the maximum signal line length by half is to place the spares in the center row and column.

Now let us represent a damaged array in a graph model. Let U be the set of defects in the array and V the set of fault-free spare elements. We have $U = \{(i, j): 1 \leq i \leq M, 1 \leq j \leq N; (i, j) \text{ being defective}\}$, and $V = \{(i, j): 0 \leq i \leq M, 0 \leq j \leq N \textit{ such that } i \cdot j = 0; (i, j) \text{ being fault-free}\}$. The replacement relation between the defects and the spare elements can be defined by a set E, $E = \{((i, j), (m, n)): (i, j) \in U, (m, n) \in V, \text{ and either } i = m \text{ or } j = n\}$. The graph $G = (U \cup V, E)$ is clearly bipartite, since by definition of U and V there are no edges in G connecting vertices within U or V. Fig. 2 shows an ex-

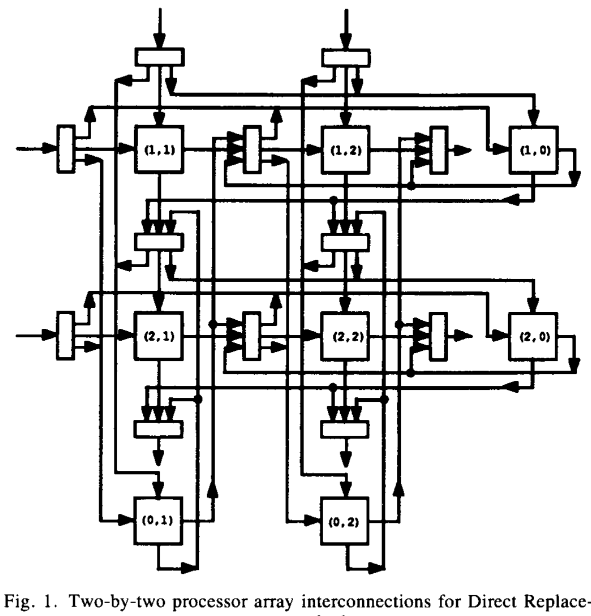


Fig. 1. Two-by-two processor array interconnections for Direct Replacement method.

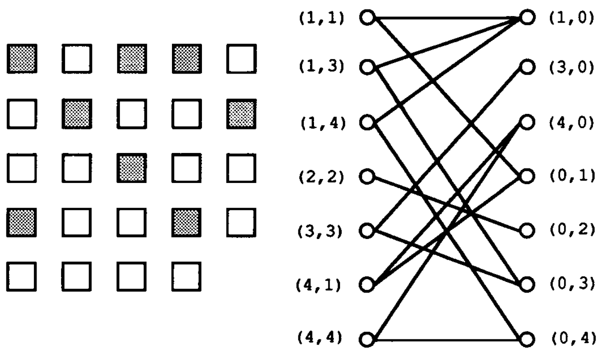


Fig. 2. Example defect pattern and its corresponding graph representation [*Note:* The indices for the spare row and column are (0, *) and (*, 0), respectively.].

ample of a defect pattern and its corresponding graph representation.

As is pointed out in [16], the spare allocation part of the reconfiguration can be considered as the well-known *maximum matching* problem in graph theory [24]. Given the vertex set partition $\{U, V\}$, and the edge set E, let δ be the maximum number of mutually disjoint edges (replacements) in E. We are interested in finding out if $\delta = |U|$ that is, whether every defective element can be uniquely replaced by a working spare element.

B. Neural Net Reconnection Control Unit

As first described by Hopfield [9], the neural net is modeled here as a set of *neurons* (computation elements) interconnected by *synapses* (weighted links). The state of neuron i is represented by a binary value s_i, with $s_i = 0$ representing a nonfiring condition an $s_i = 1$ and impulse-firing condition. The unit influence fired between neuron i and neuron j is multiplied by a synaptic weight factor w_{ji} $(= w_{ij})$. Besides the influences it receives from other neurons connected to it, a neuron, say i, also obtains a constant bias b_i regardless of its present state. Now, let s_i' denote the ith neuron's next state value. From the preceding description one can represent the neural net *state transition function* as follows:

$$s_e' = \begin{cases} 0, & \text{if } \sum_j w_{ij} s_j + b_i < 0, \\ 1, & \text{if } \sum_j w_{ij} s_j + b_i > 0, \\ s_i, & \text{otherwise.} \end{cases}$$

We assume that neurons which receive a larger influence in magnitude will have a faster state transition (i.e., it will have higher probability of early transition). Neuron processors are expected to operate in an asynchronous and random fashion.

The total energy E^{NN}, commonly known as the Lyapunov function [9], is given by

$$E^{NN} = -\tfrac{1}{2} \sum_i \sum_j s_i w_{ij} s_j - \sum_i b_i s_i.$$

Note that if $w_{ij} = 0$, for all i's, the change of state by neuron i will result in an absolute decrease of $\Sigma_j\, w_{ij} s_j + f(s_i) b_i$ in energy, where $f(s_i) = 1$ if $s_i = 0$, and -1 otherwise. Together with the fact that the total energy is bounded, the network is guaranteed to reach a local minimal energy fixed point [10].

For the processor array repair problem, since it is necessary to determine the direction (vertical or horizontal) for each faulty element, we allocate two neurons for each processing element in the array. The two neurons associated with processor (i, j) will be represented by $(i, j, 1)$ and $(i, j, 2)$.

Let the state of a neuron associated with processor (i, j) be $s_{i,j,k}$. A faulty processing element is replaced vertically (horizontally) if and only if $s_{i,j,1} = 1$ $(s_{i,j,2} = 1)$. Since a faulty element is replaced by one spare element only, the case $s_{i,j,1} = s_{i,j,2} = 1$ is considered acceptable, but redundant. As for the fault-free processing elements, the associated neuron pairs are disabled after fault diagnosis, that is, $s_{i,j,1} = s_{i,j,2} = 0$ if (i, j) is operational.

The second step is to formulate an *acceptability* function $\mathcal{A}$ of the neural network states. Our first objective here is to discourage the cases where there are more than one vertical fault in a column or more than one horizontal fault in a row. Therefore, we have

$$\mathcal{A}_1 = \begin{cases} -A/2 \sum_i \left(\sum_j s_{i,j,2} - 1 \right)^2 \\ -A/2 \sum_j \left(\sum_i s_{i,j,1} - 1 \right)^2, \quad A > 0, \end{cases}$$

as a term in $\mathcal{A}$. Here A is a constant and its value is decided empirically. According to the definition, $\mathcal{A}_1$ is maximized when there is exactly one fault marked vertically in each column and one fault marked horizontally in each row. Incorrect assignment of spares are thus discouraged. But, some side effects are also incurred. For example, if (i, j) is the only defective element in row i and column j, it is encouraged to be marked vertically and horizontally at the same time, which is redundant but acceptable as mentioned earlier.

For the processor array repair problem, a partially recovered array is no better than the unrepaired faulty array. Only, a complete matching of the faulty processing elements to the spare elements is of interest for the repair purpose. To get a complete solution, we create a term

$$\mathcal{A}_2 = -B/2 \sum_i \sum_j (s_{i,j,1} + s_{i,j,2})^2 \mathfrak{F}(i, j),$$
$$2A > B > 0,$$

where $\mathfrak{F}(i, j) = 1$ if (i, j) is defective, and zero otherwise. Here B is a constant which is typically larger than A, but not by a larger factor because if B is very much larger than A the effect of the first factor $\mathcal{A}_1$ will be masked by $\mathcal{A}_2$. This way, every fault is encouraged to be replaced by a spare in the same row or a spare in the same column. The redundancy of one defect replaced by two spares is also avoided.

The third step of the construction is to determine the weights of the synapses and neuron biases of the neural network. This can be done by making the negative acceptability function, $-\mathcal{A}$, equivalent to the neural network energy function. The symmetric synaptic interconnection weight matrix $[w_{xy}\text{'s}]$ and the biases b_z's are derived as follows:

$$w_{xy} = \begin{cases} -A, & \text{if } x = s_{i,k,1},\ y = s_{j,k,1}, \text{ or} \\ & \quad x = s_{k,i,2},\ y = s_{k,j,2}, \quad \forall i, j, k. \\ -B, & \text{if } x = s_{i,j,1},\ y = s_{i,j,2} \text{ or} \\ & \quad x = s_{i,j,2},\ y = s_{s,j,1}, \quad \forall i, j. \\ 0, & \text{otherwise.} \end{cases}$$

$b_z = A + B/2$ for all z associated with a defect.

Thus, by making the neural network energy function E^{NN} equivalent to $-\mathcal{A}$, the neural network convergence toward local minimal energy levels is transformed into searches of maximal matchings. Note that the synaptic weights are independent of the actual defect patterns. Only the biases to the neuron processors are affected by the actual array defect pattern. In general, a neural network of this kind requires $2mn$ neurons and $mn(n - 1)/2 + nm(m - 1)/2 + 3mn$ synapses. Let l_s be the total wire length for synaptic connections, then

$$l_s = mh \left(\sum_{k=1}^{n} \sum_{i=0}^{k-1} i \right) + nw \left(\sum_{k=1}^{m} \sum_{i=0}^{k-1} i \right),$$

which is of complexity $O(nm(hn^2 + wm^2))$.

The example neural network synaptic interconnections defined for a 4×4 processor array is shown in Fig. 3,

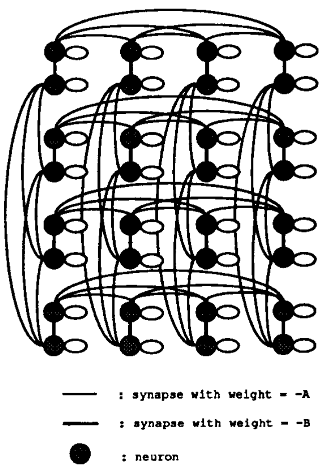


Fig. 3. Example neural net synaptic interconnections.

where the biases are defined according to the fault pattern given in Fig. 2.

Now, let us examine the effect of an unrepaired defective element u in a maximal but incomplete matching of defects. According to the definitions of w_{xy}'s and b_z's, the two neurons associated with the element u will each receive a positive bias of $A + B/2$, and a negative feedback of $-A$. The total influence to each of the two neurons is $B/2$, $B > 0$. Once one of the two neurons is turned on, the other neuron will be suppressed from turning on due to an extra $-B$ amount of influence, provided by the competing neuron after it locks in the firing state.

If the neuron, which represents a vertical (horizontal) repair, is turned on, a conflict between two repairs will occur in the column (row). The two conflicting firing neurons will in turn each fire $-A$ amount of feedback to itself and the other neuron. At the same time, each neuron will also receive $-2A$ amount of feedback and $(A + B/2)$ amount of bias. By the assumption $A > B/2$, a total of $-(A - B/2)$ amount of influence will be received by a firing neuron to turn itself off. After one neuron is turned off, the other one will be freed of a $-A$ amount of feedback, resulting in a total influence of $B/2$, which will ensure that it will remain in the firing state.

Thus, we have shown that incomplete matchings cannot be made stable states (local minima) of the neural network, and there will only be at most one neuron firing in each row and column. Although the backtracking decision from incomplete maximal matchings toward a complete matching are made in a random fashion, the neural network thus constructed will eventually find a complete defect-spare matching, if the defect pattern is repairable. Conversely, the neural network will not be able to stop naturally by arriving at a stable state if the given array defect pattern is not repairable in the first place. In this case, a time-out mechanism must be installed to stop the self-repair operation.

We have simulated processor arrays of sizes 8×8 and 16×16 with one redundant row and one column for performance evaluation. For each class of defect patterns with k defects, up to 5000 random patterns have been generated for reconfiguration. The neural-net-controlled reconfigurations were simulated in software to determine its effectiveness in terms of the percentage of array defect patterns that can be repaired. The proposed neural-net-controlled scheme, Direct Substitution (NNDS), is compared against the Direct Reconfiguration (DR), Fixed Stealing (FS), and Variable Stealing (VS) algorithms. The resulting curves are shown in Fig. 4.

Roughly, all four repair schemes can easily repair nearly all defect patterns with the number of defects within one-fourth of the maximum. As the number of defects increases, NNDS continuously maintains the best repair percentages. The differences in repair percentage of NNDS against the other three schemes are narrowed only when the number of defects exceeds about three-fourths of the maximum.

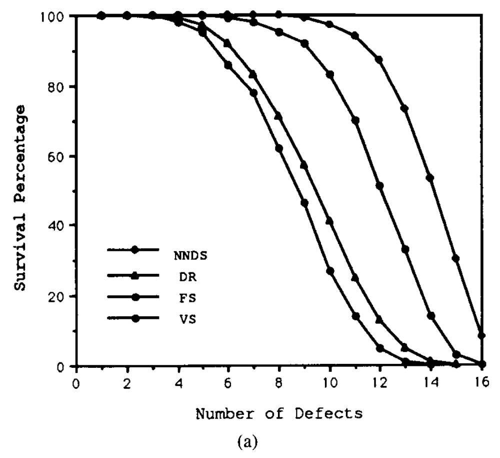
C. Electronic Implementation

A feasible implementation of an electronic neural network for processor array reconfiguration is demonstrated here. In this design, a neuron is realized as a pair of inverters in series connection which convert analog input voltages to binary output voltages. The firing neuron state is represented by the high-voltage output, and the nonfiring state by the low-voltage output. The synaptic influences between neurons as well as the biases to neurons are represented by electrical currents. Thus, a positive influence propagated to a neuron can be realized by injecting current to the corresponding inverter input. Similarly, a negative influence propagated to a neuron can be realized by sinking current from the corresponding inverter input. The synaptic amplification is simulated by the branch resistance regulating the amount of current that passes through it. The extra neural net logic in transistor level for each processor element of an 8×8 array is shown in Fig. 5. The overhead per processor, in terms of transistor count for processor arrays of any size, can be easily determined as follows:

- $64 \cdot w$ transistors for four switching circuits, where w is the width of the input-output buses. Each switching circuit consists of six transmission gates and two inverters.
- $12 + 2(n + m)$ transistors for neurons and synapses, since the array size is $n \times m$.
- Four transistors for neuron enable/disable logic.

An 8×8 processor array for image processing is implemented for an example. Each processor holds one pixel of the image to be processed. The only function of a processor is to take inputs from itself and the four adjacent neighbors to perform a majority vote. The result replaces the pixel previously stored in the processor. The processor element, redundant interconnections, and neural network have been implemented in MOSIS 2-μm CMOS digital process technology. Because of space constraints, the actual layout is not shown here. It may be noted that, for an 8×8 array, altogether 112 extra transistors per processor are required for implementing the neural net reconfiguration control circuit. The total chip area including the neural network was 8128 μm $\times$ 12 686 μm. The most critical part of the neural net design is the calculation of the transistor sizes for the exact amounts of currents through pull-up and pull-down transistors. The neural network behavior has been verified by performing SPICE simulations. An example repair on a 16×16 array with nine defects is shown in Fig. 6. The defective processors are represented by the shaded squares. An arbitrary incomplete assignment of defective processors to the boundary spare processors was made to define the initial state of the neural net.

Let $Hi(Vi)$ be the horizontal (vertical) neuron of processor i. The defective processors 2 and 3 were not assigned to any spares initially. Therefore, all the neurons associated with these two processors, $H2$, $V2$, $H3$, and $V3$ started firing right away. But, as soon as their voltage levels were high enough, $H2$ and $H3$ started to discourage each other, since they were seeking the same spare.



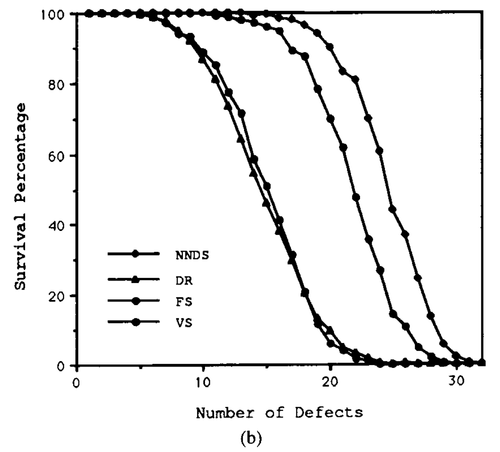


Fig. 4. Performance evaluations between conventional and neural network reconfiguration methods: (a) 8×8 processor arrays; (b) 16×16 processor arrays.

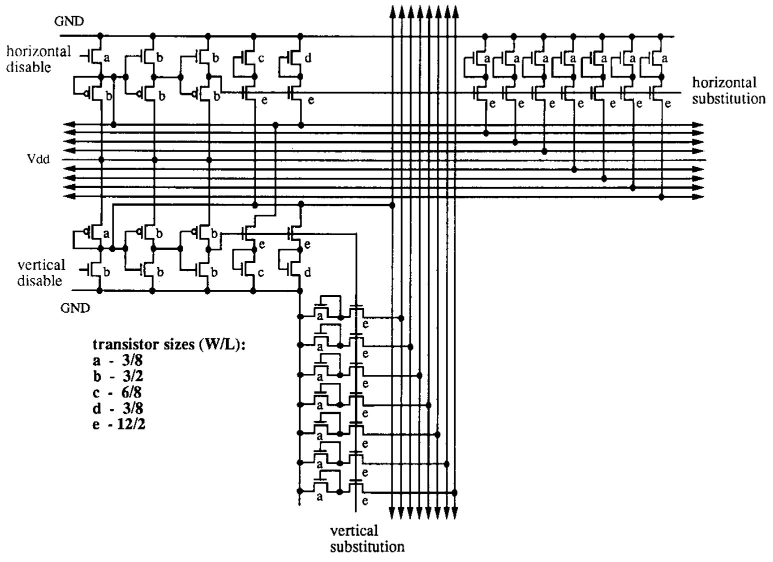


Fig. 5. Neural subnet for a basic cell of an 8 × 8 BISR processor array.

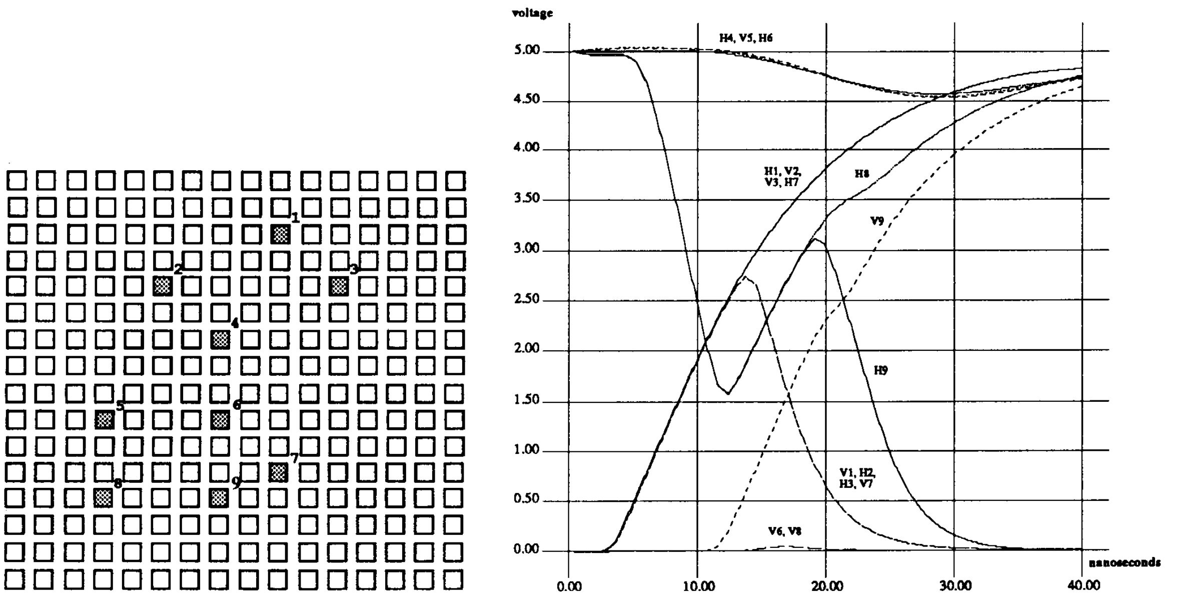


Fig. 6. Example 2-D processor array reconfiguration circuit simulation.

Moreover, $H2$ ($H3$) were also receiving negative influence from $V2$ ($V3$) to turn off. As a result, both processors 2 and 3 were substituted vertically. A similar scenario happened to processors 1 and 7, except they were eventually substituted horizontally.

For the remaining substitutions, notice that initially both $H8$ and $H9$ were firing, that is, these two processors were assigned horizontally to the same spare. After the reconfiguration started, $H8$ and $H9$ tried to turn each other off. Gradually, both neurons came down to a low-voltage output state so as to remove the conflicting assignments. Then, to repair processors 8 and 9, the same two neurons started to turn on the firing state again. But, this time another competitor, $V9$, also started to turn on to the firing state. $V8$ was discouraged by $V5$, which had been in the firing state since the beginning. In the competition, $H8$ and $V9$ were mutually independent, but both opposed $H9$ to turn on to the firing state. Finally, $H9$ started to turn off, and the network reached a stable state for a successful assignment of spares.

III. Window Substitution Method

Direct Substitution, which has been proven to be a powerful self-repair method, is based on the direct replacement of a defective processor by a spare processor that is located on the same row or on the same column as the defective processor. In this section, an approach, called *Window Substitution*, is invented as an improvement over Direct Substitution. The idea behind this design is to associate each element in the physical array with some subset of positions in the logical array so that the element can *move* to any one of the positions in the subset. The reconfiguration serves to arrange the movements of the surviving fault-free elements to fill all the positions in the logical array. While a subset can be any combination of the logical positions, we will narrow the attention to square subarrays, or windows, as possible subsets of positions.

Definition 1: A window of size $p \times q$, denoted by $L_{i,j}$, associated with the processing element (i, j) is given by

$$L_{i,j} = \{(i - a, j - b) \mid 0 \le a \le \min(p - 1, i),\ 0 \le b \le \min(q - 1, j)\}.$$

Definition 2: Let the positions inside a window be numbered sequentially in row-major fashion starting from zero. A *move* by element (i, j) is denoted by (i, j, k) where k is the position number inside the window. In other words, the logical position that corresponds to the move (i, j, k) is $(i + \lfloor k/q \rfloor - p + 1, j - q + 1 + k \bmod q)$.

If we assume that the set of positions and the set of fault-free elements are two sets of vertices, and each pair of a fault-free element and an associated position as an edge, then the proposed processor array interconnection can be modeled as a bipartite graph G. The reconfiguration problem will be to find a maximum matching between the set of positions. Fig. 7 shows an example 4×4 processor array and its bipartite graph model.

A. Redundant Interconnections

Observation 1: The set of possible south neighbors, $N^s_{i,j}$, of element (i, j) is given by

$$N^s_{i,j} = \{(i - a, j - b) \mid (i - a, j - b) \ne (i, j),\ -p \le a \le p - 2,\ -q + 1 \le b \le q - 1\}.$$

The set of possible east neighbors, $N^e_{i,j}$, of element (i, j) is given by

$$N^e_{i,j} = \{(i - a, j - b) \mid (i - a, j - b) \ne (i, j),\ -p + 1 \le a \le p - 1,\ -q \le b \le q - 2\}.$$

Observation 2: The cardinality of the preceding sets is given by

$$\max_{\forall i,j} |N^e_{i,j}| = \max_{\forall i,j} |N^s_{i,j}| = 4p \cdot q - 2p - 2q.$$

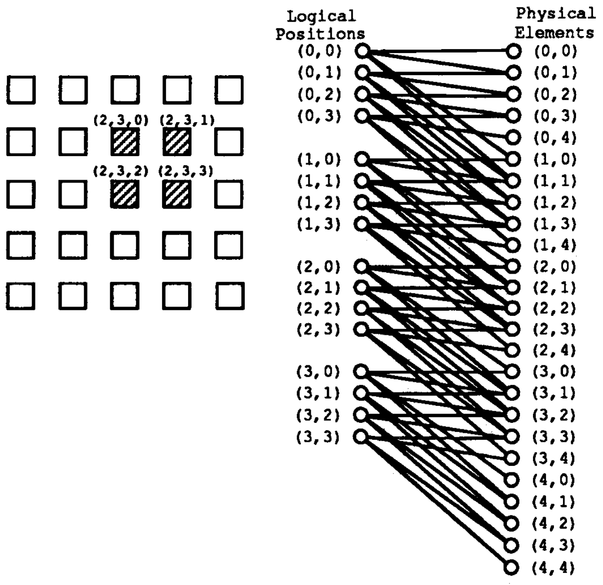


Fig. 7. Four-by-four processor array graph model for Window Substitution method.

Given the neighboring information in Observations 1 and 2, it is sufficient for the design of the redundant interconnections, that is, for each element, a $1 : (4pq - 2p - 2q)$ demultiplexer can be used to direct its horizontal output to one of its possible east neighbors, and another $1 : (4pq - 2p - 2q)$ demultiplexer to direct its vertical output to one of its possible south neighbors. But, there are two problems involved in this approach. First, because of the mutual movement between an element and its potential neighbors, each demultiplexer should be provided with information by up to $4pq - 2p - 2q + 1$ elements about their moves within the windows. Second, the movement information should be further translated to control signals for the demultiplexer.

Observation 3: The set of possible east neighboring moves, $M^e_{i,j,k}$, of (i, j, k) is given by

$$\begin{aligned} M^e_{i,j,k} = \{&(i + \lfloor k/q \rfloor - \lfloor m/q \rfloor,\\ &j + k \bmod q - m \bmod q + 1, m) \mid\\ &(i, j) \ne (i + \lfloor k/q \rfloor - \lfloor m/q \rfloor,\\ &j + k \bmod q - m \bmod q + 1),\\ &\quad k + 1 \ne m \quad \text{if} \quad (k + 1) \bmod q \ne 0,\\ &\quad 0 \le m \le p \cdot q - 1\}. \end{aligned}$$

The set of possible south neighboring moves, $M^s_{i,j,k}$ of (i, j, k), is given by

$$\begin{aligned} M^s_{i,j,k} = \{&(i + \lfloor k/q \rfloor - \lfloor m/q + 1 \rfloor,\\ &j + k \bmod q - m \bmod q, m) \mid\\ &(i, j) \ne (i + \lfloor k/q \rfloor - \lfloor m/q \rfloor,\\ &j + k \bmod q - m \bmod q + 1), \end{aligned}$$

$$k + 1 \neq m \quad \text{if} \quad (k + 1) \bmod q \neq 0,$$
$$0 \leq m \leq p \cdot q - 1\}.$$

To avoid the two problems mentioned earlier, we propose an interconnection design in which an element is now surrounded by two $pq : 1$ multiplexers on the input sides, and two $1 : pq$ demultiplexers on the output sides as shown in Fig. 8(a). If an element takes a move to position k in its window, each of the multiplexers and demultiplexers will direct its horizontal or vertical input-output through line k. Among the elements, a horizontal demultiplexer output line that corresponds to, say, move (i, j, k), is identified with those horizontal multiplexer input lines that correspond to the moves in $M^e_{i,j,k}$ given in Observation 3. Similarly, a vertical demultiplexer output line that corresponds to move (i, j, k) is identified with those vertical multiplexer input lines that correspond to the moves in $M^e_{i,j,k}$. Overall, the interconnection design requires $4nm$ multiplexers or demultiplexers, and the total length of the links is

$$[m + 2(m - 1)](n - 1)w + 2(m - 1)w$$
$$+ [n + 2(n - 1)](m - 1)h + 2(n - 1)h,$$

which is of complexity $O(nm(w + h))$. An example connection of a 3×3 processor array is shown in Fig. 8(b).

In this connection structure, a net may consist of multiple output lines and multiple input lines. But there will be exactly one pair of input and output lines connecting two elements, since each net is dedicated to serve as a communication link between a unique pair of logical positions. This way, the connections are totally controlled by the individual moves of the elements. The task of finding an element's east or south neighbor is made transparent to the element.

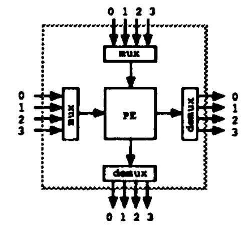
Observation 4: The maximum city block distance between two directly connected physical elements after reconfiguration is $p + q - 1$.

The communication time overhead between two neighboring elements in the logical array involves a $2 \lceil \log_2 pq \rceil$ logic gate level delay, caused by the demultiplexing and multiplexing of the input-output lines, and a propagation delay of $O(p + q)$ caused by the distance in between.

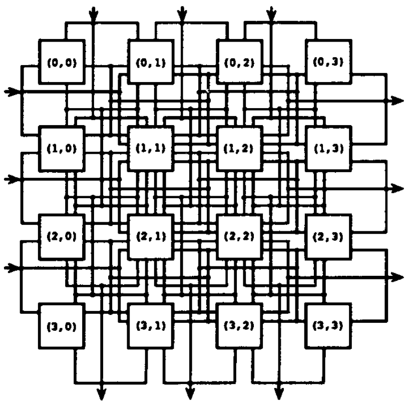
B. Neural Net Reconnection Controller

To represent the possible moves of element (i, j), a group of pq neurons are allocated with their states denoted as $s_{i,j,k}$'s, where $0 \leq k \leq pq$. We assume that all neurons in a group will be disabled if the corresponding element is faulty, that is, we have $s_{i,j,k} = 0$ for all k if element (i, j) is faulty.

$$s_{i,j,k} = \begin{cases} 1, & \text{if element } (i,j) \text{ moves into position} \\ & \quad k \text{ inside its window.} \\ 0, & \text{otherwise.} \end{cases}$$



(a)



(b)

Fig. 8. Three-by-three processor array interconnection for Window Substitution method.

Then, a cost term $\mathcal{C}_1$ is defined to discourage an element from making multiple moves, that is, there is more than one firing neuron within a neuron group.

$$\mathcal{C}_1 = \frac{A}{2} \sum_i \sum_j \sum_k \sum_{k' \neq k} (s_{i,j,k} \cdot s_{i,j,k'}), \qquad A > 0.$$

$\mathcal{C}_1$ is minimized if every element makes either no move at all or a unique move inside the window. An element is not utilized in the reconfiguration if no move is made by the element.

Observation 5: Let $P_{i,j}$ be the set of moves that can take position (i, j).

$$P_{i,j} = \{(i + pi - 1 - \lfloor m/q \rfloor, j + q - 1$$
$$- m \bmod p, m) \mid 0 \leq m \leq pq - 1\}.$$

To encourage exactly one element to move into a position—i.e., there is only one element in $P_{i,j}$ equal to 1—for all $P_{i,j}$'s, we have a cost term $\mathcal{C}_2$ defined.

$$\mathcal{C}_2 = \frac{B}{2} \sum_i \sum_j \left[\left(\sum_{\forall s_{i,j,k} \in P_{i,j}} s_{i,j,k} \right) - 1 \right]^2, \qquad B > A > 0.$$

$\mathcal{C}_2$ is minimized if every logical problem is filled with a unique physical element after reconfiguration.

After transforming the total cost function $\mathcal{C} = \mathcal{C}_1 + \mathcal{C}_2$ to the neural net energy function E^{NN}, the resulting synaptic strengths and biases are determined as follows:

$$w_{xy} = \begin{cases} -A, & \text{if } x \neq y \text{ and } x, y \in L_{i,j} \text{ for some } i, j \\ -B, & \text{if } x, y \in P_{i,j} \text{ for some } i, j \\ 0, & \text{otherwise.} \end{cases}$$

and $b_z = B$ for all z's.

The neural network design requires $pqmn$ neurons, and the total number of synapses is bounded by $mnpq(pq - 1) + pq$. Assuming that the synaptic wirings for neurons within a processor element are negligible, the total length of the synaptic wirings is bounded by

$$mn\left[qh\left(\sum_{k=1}^{p}\sum_{i=1}^{k-1} i\right) + pw\left(\sum_{k=1}^{q}\sum_{i=1}^{k-1} i\right)\right],$$

which is of complexity $O(mnpq(hp^2 + wq^2))$. But, by keeping p and q small, such as two or three, the total numbers of neurons and synapses, and length of synaptic wirings will have a reduced complexity of $O(mn)$.

Now, let us examine how the neural net will act when some positions of the logical array are not filled with a physical element, while a maximal matching has already been reached. Let (s, t) be some position whose associated surviving elements have all been used to fill other positions. We will ignore the possibility that all elements associated with position (s, t) are defective, since it would be impossible to find a complete matching in the first place. Due to the cost term $\mathcal{C}_2$, every neuron in $P_{i,j}$ will be encouraged to fire by a bias of B. However, due to the cost term $\mathcal{C}_1$, the same neuron will receive $-A$ amount of influence to stay in the nonfiring state. By the assumption $B > A$, all neurons in $P_{i,j}$ will try to turn on to the firing state to satisfy the more important need to fill position (i, j). After one of the neurons is turned on, say neuron (i, j, k), the B amount of bias received by the neurons in $P_{i,j}$, including (i, j, k) itself, will be canceled out by the $-B$ amount of influence received from neuron (i, j, k). However, element (i, j) would have made more than one unique move, by having two neurons from its neuron group firing it at the same time. Due to the cost term $\mathcal{C}_1$, the two conflicting neurons will fire $-A$ amount of influence to force each other to turn off. If neuron (i, j, k) succeeds, then the neural net is effectively starting a new search for another maximal matching. This process is repeated until a complete maximal matching is found.

To evaluate the Window Substitution method, three different window sizes, 2×2 (NNWS22), 2×3 (NNWS23), and 3×3 (NNWS33), are described and their performances are compared with the Direct Substitution method (NNDS). According to Fig. 9(a), NNWS, with a window size as large as 2×3, is sufficient to achieve the highest survival rates in all cases. In fact, even with the smallest window, NNWS22, the Window Substitution method is found to be significantly more effective than conventional software algorithms, such as Fixed Stealing and Direct Reconfiguration. On average cases in

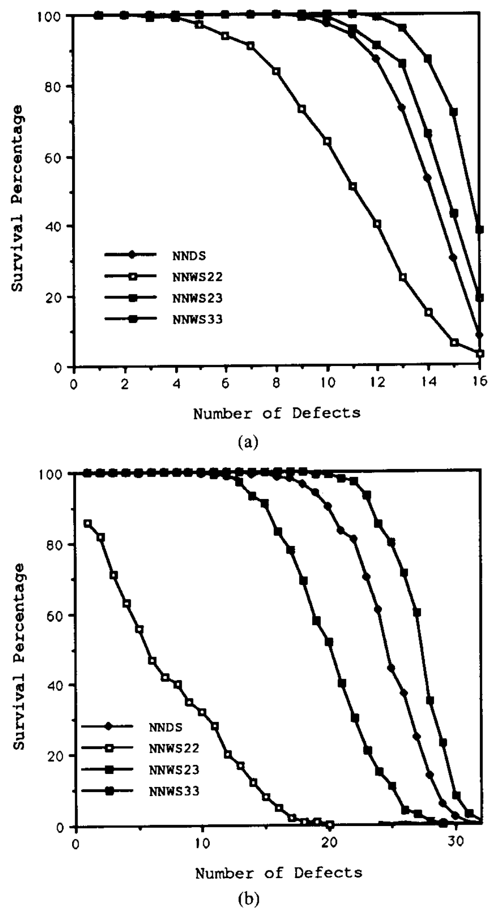


Fig. 9. Performance evaluations among neural network reconfiguration methods: (a) 8×8 processor arrays; (b) 16×16 processor arrays.

the simulation, NNWS22 approaches the performance of Variable Stealing as the number of defects increases, and actually surpasses the VS's performance when there are 14 or more defects occurring in a defect pattern.

According to Fig. 9(b), NNWS33 achieves the highest survival rates, followed by NNDS. However, achievable survival rates by NNWS23 and NNWS22 appear to be severely affected by the increased array size. Let $r_{m,n}^k$ be the survival rate achieved in $m \times n$ arrays with k defects. To demonstrate the effect of array size to the achievable survival rates, $|r_{8,8}^k - r_{16,16}^{2k}|$ versus k is shown in Fig. 10 for each reconfiguration method.

From Fig. 10, it can be easily identified that the Windows Substitution method is the most effective technique by the increase in array size. NNWS22, NNWS23, and NNWS33 show the three largest peak differences in survival rates. The bell-shaped curves can be explained as follows. When there are only a small number of defects in the array, all methods can handle the reconfiguration easily regardless of the array size. As the capacity of a method is gradually saturated by more defects, the array

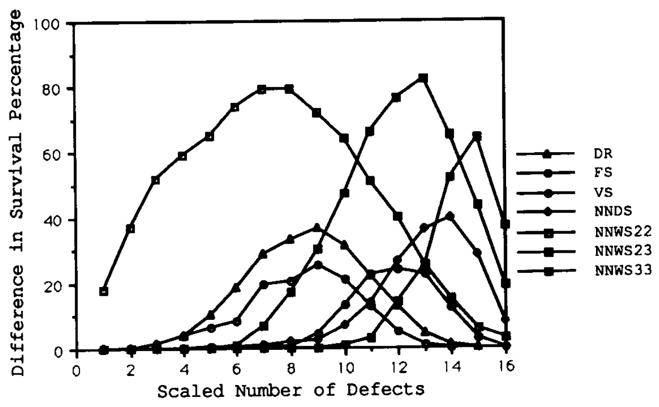


Fig. 10. Example effect of array size on reconfiguration methods.

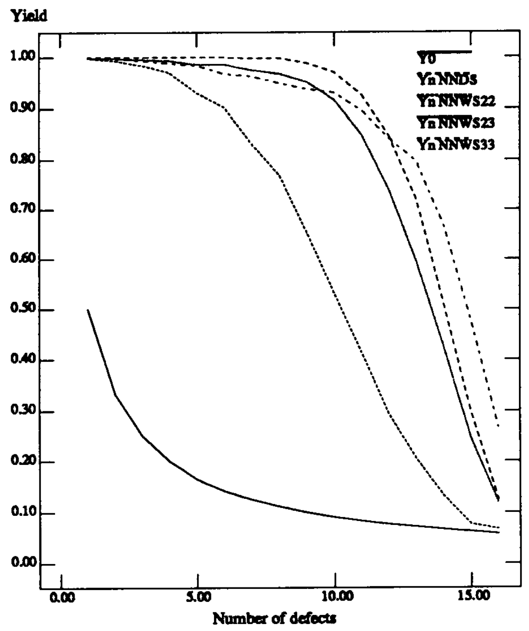


Fig. 11. Yield improvement for four neural net BISR methods on 8 × 8 processor arrays.

size becomes quite critical. Finally, the task of reconfiguration becomes more and more difficult as the number of defects increases to the maximum value that can be repaired, because the differences are reduced by the low survival rates in both arrays. In this aspect, NNWS33 and NNDS show the largest capacities in handling defects by having their peak differences skewed to the maximum defects.

Finally, to show the robustness of neural network control units, random faults are injected into the simulation model of an 8 × 8 processor array such that the processing elements as well as the associated neural networks can have faulty components. The results of simulation have been applied to Stapper's [30], [31] well-known yield model to estimate the improved yield values, Y_n's, of the various types of neural networks discussed in this paper. These estimated yields have been shown in Fig. 11 along with the yield, Y_0, of a processor array having no built-in self-repair circuit.

IV. Summary

As gargantuan ULSI and GSI chips with several billion transistors are becoming a reality, massively large array processors are envisioned to be ideal candidates for the exploitation of large silicon surfaces. Even with very sophisticated processing technologies with as low as 0.0001% defect densities (i.e., one defective transistor per million), several thousand transistors are likely to fail due to some unavoidable manufacturing defects. To achieve reasonable chip yield, built-in self-repair (BISR) schemes are required to eliminate automatically these faulty components. This paper demonstrates a powerful BISR circuit which employs a distributed neural network to collect the global fault pattern and to derive an efficient solution for reconfiguring the array. The output of the neural network is applied to the control inputs of multiplexers to program them such that the defective components are eliminated from the paths of computation. Two neural repair algorithms are proposed in this paper. The Direct Substitution technique performs a *maximum matching* algorithm that provides an efficient one-to-one mapping between the defective and the spare elements, while the more powerful Window Substitution reconfiguration scheme rearranges logical labels of the fault-free processors such that the connectivities of the target array network are maintained over them. The significant difference between these two approaches is that while the first scheme labels the spare elements separately such that each processor in the array has designated spare processors, the second scheme attempts to construct a logical array of the required size over a physical array of good and defective processors. Both schemes are efficient in the sense that they can achieve near-maximum utilization of the redundant interconnections. The simulation results show that even the less sophisticated Direct Substitution technique has superior performance to the software heuristic reconfiguration algorithms; the performance of the more sophisticated Window Substitution scheme can be improved further by adjusting its window size. Both these techniques require relatively low circuit overhead and they can ameliorate the chip yield by a significant factor. At the same time, the BISR circuits can also improve the life span of a chip by automatically restructuring the chip when components fail during its normal operation. In space and other harsh environments, where chips may fail more frequently due to cosmic radiations and ambient noise, the BISR circuits will improve the survivability and availability of the system.

References

[1] K. E. Batcher, "Design of a massively parallel processor," *IEEE Trans. Comput.*, vol. C-29, pp. 836-840, Sept. 1980.

[2] M. Chean and J. A. B. Fortes, "The Full-Use-of-Suitable-Spares (FUSS) approach to hardware reconfiguration of processor arrays," *IEEE Trans. Comput.*, vol. C-39, no. 4, pp. 564-571, Apr. 1990.

[3] Y. H. Choi, S. H. Han, and M. Malek, "Fault diagnosis of reconfigurable systolic arrays," in *Proc. Int. Conf. Computer Design*, 1984, pp. 451-455.

[4] R. K. Chung, F. T. Leighton, and A. L. Rosenberg, "Diogenes: A methodology for designing fault-tolerant VLSI processor arrays," in *Proc. Int. Symp. Fault-Tolerant Comput.*, 1983, pp. 26-32.

[5] F. Distante, *et al.*, "Array partitioning: A method for reconfigurability and reconfiguration problems," in *Proc. Int. Conf. Computer Design*, 1988, pp. 564-567.

[6] S. Even, *Graph Algorithms*. Rockville, MD: Computer Science Press, 1979.

[7] J. W. Greene and A. E. Gamel, "Configuration of VLSI arrays in the presence of defects," *J. ACM*, vol. 31, no. 4, pp. 694-717, Oct. 1984.

[8] K. K. Hedlund and L. Snyder, "Systolic architectures: A wafer scale approach," in *Proc. Int. Conf. Computer Design*, 1982, pp. 604-610.

[9] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," in *Proc. Natl. Acad. Sci. USA*, Apr. 1982, pp. 2554-2558.

[10] J. J. Hopfield and D. W. Tank, "Neural computation of decisions in optimization problems," *Biol. Cybern.*, pp. 141-152, 1985.

[11] S. H. Hosseini, "On fault-tolerant structure, distributed fault-diagnosis, reconfiguration, and recovery of the array processors," in *Proc. Int. Conf. Computer Design*, 1988, pp. 554-559.

[12] I. Koren, "A reconfigurable and fault-tolerant VLSI multiprocessor array," in *Proc. Int. Symp. Computer Architecture*, 1981, pp. 425-442.

[13] I. Koren and M. A. Breuer, "On area and yield considerations for fault-tolerant VLSI processor arrays," *IEEE Trans. Comput.*, vol. C-33, no. 1, pp. 21-27, Jan. 1984.

[14] H. T. Kung, "Why systolic architectures?," *IEEE Computer*, pp. 37-46, Jan. 1982.

[15] S. Y. Kuo and W. K. Fuchs, "Efficient spare allocation in reconfigurable arrays," *IEEE Design and Test of Computers*, vol. 4, no. 1, pp. 24-31, 1987.

[16] S. Y. Kuo and W. K. Fuchs, "Spare allocation and reconfiguration in large area VLSI," in *Proc. Design Automation Conf.*, 1988, pp. 609-612.

[17] T. Leighton and C. E. Leiserson, "Wafer-scale integration of systolic arrays," *IEEE Trans. Comput.*, vol. C-34, no. 5, pp. 448-461, May 1985.

[18] T. E. Mangir and A. Avizienis, "Fault-tolerant design for VLSI: effect of interconnect requirements on yield improvement of VLSI designs," *IEEE Trans. Comput.*, vol. C-31, no. 7, pp. 609-615, July 1982.

[19] T. Mano *et al.*, "A fault tolerant 256k RAM fabricated with molybdenum-polysilicon technology," *IEEE J. Solid-State Circuits*, vol. SC-15, pp. 865-872, Oct. 1980.

[20] P. Mazumder and J. S. Yih, "Built-in self-repair techniques for yield enhancement of embedded memories," in *Proc. Int. Test Conf.*, Sept. 1990, pp. 833-841.

[21] W. S. McCulloch and W. H. Pitts, "A logical calculus of ideas immanent in nervous activity," *Bull. Mathematical Biology*, pp. 115-133, 1943.

[22] W. R. Moore, "A review of fault-tolerant techniques for the enhancement of integrated circuit yield," *Proc. IEEE*, vol. 74, no. 5, pp. 684-698, May 1986.

[23] R. Negrini, M. Sami, and R. Stefanell, "Fault tolerance techniques for array structures used in supercomputing," *IEEE Computer*, vol. 18, no. 2, pp. 78-87, Feb. 1986.

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

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

[26] M. Sami and R. Stefanelli, "Reconfigurable architectures for VLSI processing arrays," *Proc. IEEE*, vol. 74, no. 5, pp. 712-722, May 1986.

[27] R. T. Smith *et al.*, "Laser programmable redundancy and yield improvement in a 64k DRAM," *IEEE J. Solid-State Circuits*, vol. SC-16, pp. 506-513, Oct. 1981.

[28] L. Snyder, "Introduction to the configurable, highly parallel computer," *IEEE Computer*, vol. 15, no. 1, pp. 47-56, Jan. 1982.

[29] A. K. Somani and V. K. Agarwal, "System level diagnosis in systolic systems," in *Proc. Int. Conf. Computer Design*, 1984, pp. 445-450.

[30] C. H. Stapper, A. N. McLaren, and M. Dreckmann, "Yield model for productivity optimization of VLSI memory chips with redundancy and partially good product," *IBM J. Res. Develop.*, vol. 24, pp. 398-409, May 1980.

[31] C. H. Stapper, "On yield, fault distributions, and clustering of particles," *IBM J. Res. Develop.*, vol. 30, no. 3, pp. 326-338, May 1986.

[32] Y. Ueoka, C. Minagawa, M. Oka, and A. Ishimoto, "A defect-tolerant design for full-wafer memory LSI," *IEEE J. Solid-State Circuits*, vol. SC-19, pp. 319-324, June 1984.

Pinaki Mazumder (S'84-M'87) for a photograph and a biography, please see page 135 of the Jan. 1993 issue of this Transactions.

Yih-Shyr Jih (S'85-M'91) for a photograph and a biography please see page, 136 of the Jan. 1993 issue of this Transactions.