Reconfiguration Algorithms for Degradable VLSI Arrays with Switch Faults

Yuanbo Zhu, Wu Jigang
School of Computer Science and Software Engineering
Tianjin Polytechnic University
Tianjin, China 300387
Email: {asybzhu, asjgwu}@gmail.com

Siew-Kei Lam, Thambipillai Srikanthan
Centre for High Performance Embedded Systems
Nanyang Technological University
Singapore, 639798
Email: {assklam, astsrikan }@ntu.edu.sg

***Abstract*—The problem of reconfiguring two-dimensional VLSI arrays with faults is to find a maximum logical array without faults. The existing algorithms only consider faults associated with processing elements, and all switches and links are assumed to be fault-free. But switch faults may often occur in the network-on-chips with high density. In this paper, two novel approaches are proposed to tackle the reconfiguration problem of degradable VLSI arrays with switch faults. The first approach, called CRRR, extends the well-known existing algorithm with simple pre-processing and row bypass scheme. The second one, called RCCR, employs a novel row and column rerouting scheme to maximize the size of the logical array. Simulation results show that the proposed two approaches can effectively generate the logical arrays on the given host array with switch faults, and RCCR performs more favorably with the increasing number of the switch faults. To the best of our knowledge, this paper is the first work to tackle the reconfiguration problem of degradable VLSI arrays in the presence of switch faults.**

***Index Terms*—Fault-tolerance, Degradable VLSI array, Reconfiguration algorithms, NP-complete.**

I. Introduction

Mesh-connected processor arrays have been extensively investigated and widely employed for high-speed implementation of many signal and image processing due to its simplicity and performance benefits. These high-performance processor arrays normally consist of components such as processing elements (PEs), switches, links, *etc*. With the advances of very large scale integration (VLSI) technologies, mesh-connected processor arrays can now be built on a single chip to form many-core systems. The regular and modular characteristics of these many-core systems can be exploited to shorten verification time, hence alleviating time-to-market pressures.

As the density of VLSI arrays increases, the probability of occurrence of defects in the arrays during fabrication also increases. In addition, post deployment defects can also occur due to harsh environments. Thus, it is nearly impossible to guarantee that all components are fault-free throughout their product lifetime. This has led to the significance of fault tolerant techniques to maintain reliability of VLSI arrays in order to increase their product lifetime. In particular, the regular and modular structure of mesh-connected architecture lends itself well for efficient fault-tolerant realizations.

Fault-tolerance has been studied from several perspectives on a number of parallel architectures. For example, distributed memory architectures often use spare hardware or identify a healthy subarray from a host array with faults [1]. Some of these ideas have been adopted for fault tolerant VLSI arrays. In particular, efficient reconfiguration strategies for degradable VLSI arrays [2], such as *redundancy approach* and *degradation approach* have been extensively studied in the past two decades.

In redundancy approach, additional components called spare PEs are incorporated during chip fabrication. These spare PEs can be reconfigured to replace faulty PEs. Various techniques based on the redundancy approach have been described in [3], [4], [5], [6]. The limitation of this approach stems from the fact that there must be sufficient spare elements that can replace all the faulty components. If the system contains excessive faulty elements such that the spare elements cannot replace all the faulty ones, the system is no longer reconfigurable and has to be discarded.

The degradation approach does not rely on spare PEs to replace faulty ones, and all PEs in the system are treated uniformly. Instead, the degradation approach aims to utilize as many fault-free PEs as possible to reconstruct a fault-free subarray for the target system [7]. Kuo and Chen [8] studied the reconfiguration problems for reconfigurable mesh array under the three switching and routing constraints, namely, 1) *row and column bypass*, 2) *row bypass and column rerouting*, and 3) *row and column rerouting*. They have shown that most reconfiguration problems under these constraints are NP-complete. Moreover, it becomes very difficult if rerouting in both row and column directions are considered simultaneously. The first constraint, i.e., row and column bypass, is well studied in the reconfiguration of memory arrays [9]. Low *et al*. proposed a optimal algorithm of linear time in [10] called *Greedy Column Rerouting* (GCR), to find a maximal sized target array that contains the selected rows under the row bypass and column rerouting scheme for a given $m \times n$ mesh-connected host array. This optimal algorithm was employed in [11] and generated an approximate target array by performing row exclusion and compensation, resulting in an efficient reconfiguration algorithm under the row and column rerouting constraint. Jigang *et al.* have simplified the row-selection scheme for the rows to be excluded in [12] and proposed partial rerouting scheme in [13] to accelerate the reconfigu-

ration of the target array. In addition, [14] proposed an upper bound of target array size to further reduce the execution time of GCR. Fukushi *et al.* utilized heuristic approach in [15] and genetic approach in [16] to construct the maximum target array (MTA) for the small host arrays. A more efficient algorithm based on an integrated row and column rerouting strategy was reported in [17] to further increase the harvest. A preprocessing and partial rerouting technique was proposed in [18] to reduce the reconfiguration time. Following GCR, an algorithm that focuses on optimizing power efficiency was proposed in [19] under the second constraint. This algorithm, denoted as LDP, employs a heuristic strategy and dynamic programming to reduce power dissipation by minimizing the interconnection length in logical columns of the MTA without harvest penalty. LDP was later simplified by reducing the number of the operations during reconfiguration [20]. A reconfiguration algorithm that combines GCR and LDP was proposed in [21] for constructing low temperature subarray. Different tracks and switches have also been proposed to increase the harvest on reconfigurable mesh-connected arrays [22], [23], [24], [25].

In the literature, faults in VLSI arrays are usually associated only with processing elements (PEs), *i.e.* all switches and links in an array are assumed to be fault-free. However in practice, switches and link faults can also occur during fabrication or after deployment due to harsh conditions. When switch fault is taken into consideration, the reconfiguration problem becomes more complex. In this paper, two novel techniques are proposed to tackle the reconfiguration problem of VLSI arrays with switch faults. To the best of our knowledge, our work is the first to take into account switch faults in reconfiguration of VLSI array. In particular, this research is the first contribution towards constructing a healthy logical array on a given $m \times n$ mesh-connected processors array which include PE faults and switch faults.

The rest of the paper is organized as follows: In section 2, we provide notations that are used throughout the paper. In section 3, we present our algorithms for reconfiguring VLSI arrays with PE and switch faults. In section 4, we present the experimental results to show the benefits of our algorithms. Finally, we conclude our work in section 5.

II. Preliminaries and Related Works

A VLSI array is fault-tolerant if it continues to work (possibly in degraded form) when at least one of its components fails [1]. Let's denote the original processor array obtained after manufacturing as host array H, which contains some defective components.

In this paper, we use the same architecture and assumptions in [10-21], with the exception of those related to switches and interconnects. Figure 1 illustrates the architecture of a 4×4 host array with different switch states and rerouting schemes. Neighboring PEs are connected to each other by a four-port switch. There are two kinds of links that exist in the host array: row rerouting channel (thin line) and column rerouting channel (bold line). A switch is called a row rerouting switch if it is located on row rerouting channel, while a column rerouting switch is located on column rerouting channel. In Fig. 1, circles that appear filled or empty, can be used as indicators of column switches or row switches. An $m \times n$ host array indicates a host array with m rows and n columns, whose size is defined as $m \cdot n$. Each PE in H can be represented by $e(i, j)$, where i is its row index and j is its column index. There are $m \times (n-1)$ Row Switches (*RS*) and $(m-1) \times n$ Column Switches (*CS*) in an $m \times n$ host array. Each row (column) switch in H can be represented by $rs(i, j)$ ($cs(i, j)$), where i is its row index and j is its column index.

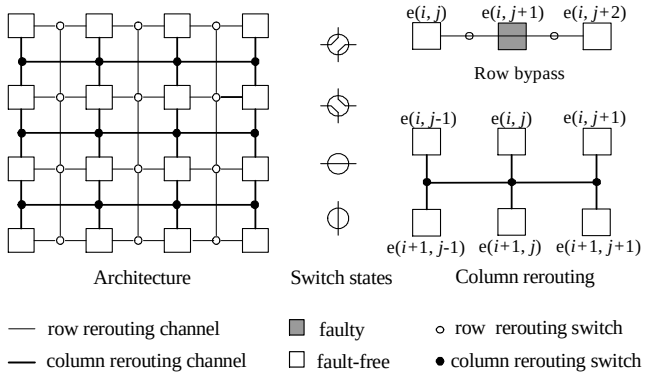

Fig. 1: Architecture and routing schemes of a 4×4 array linked by 4-port switches.

A PE is defective if it is incapable of processing data or it is unable to read (write) information from (to) its neighbors [1]. When a row (column) switch fault occur, the row (column) rerouting channel is broken and does not allow data through it. Let ρ and σ be the fault densities of PEs and switches (including row switches and column switches) in the host array respectively, where $0 < \rho < 1$ and $0 < \sigma < 1$. A degradable subarray of the host array, which contains only fault-free PE after reconfiguration, is called a target array or logical array. The rows (columns) in host array are called physical rows (columns). The rows (columns) in logical array are called logical rows (columns). In a logical array, PEs belonging to a logical column (row) may come from different physical columns (rows). An $r \times s$ target array is constructed from an $m \times n$ defective host array with r logical rows and s logical columns ($r \leq m, s \leq n$), whose size is defined as $r \cdot s$.

Typical switching and routing schemes include the *row (column) bypass* scheme and the *row (column) rerouting* scheme. The *Row Bypass Scheme* assumes that a faulty PE allows communication between its neighbors. For example, in Fig. 1, if $e(i, j+1)$ is faulty, then $e(i, j)$ can communicate with $e(i, j+2)$ directly and data will bypass $e(i, j+1)$ through row switches. Alternatively, the *Column Rerouting Scheme* assumes that $e(i, j)$ can connect directly to $e(i+1, j')$ with external column switches, where $|j' - j| \leq d$, and d is called *compensation distance*, which is limited to 1 [10-21]. The *Column Bypass Scheme* and *Row Rerouting Scheme* are similarly defined. In practice, it is important to keep the

compensation distance small in order to reduce the overhead of the switching mechanisms. As such, in this paper we also limit d to one.

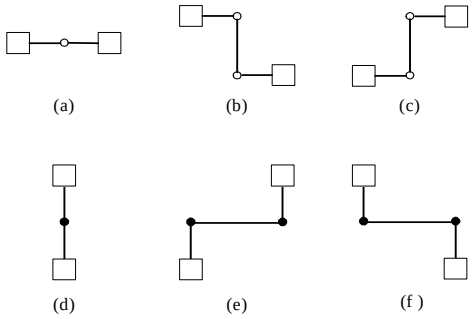

Fig. 2: Possible link-ways in target array.

As shown in Fig. 2, there are 6 possible types of interconnects for a target array. They can be categorized into two classes based on the kind of switch used. Note that (a), (b) and (c) are only used for row rerouting, while (d), (e) and (f) are only used for column rerouting. From Fig. 1 and Fig. 2, it can be seen that if a row switch $rs(i,j)$ is faulty, where $1 \leq i \leq m$ and $1 \leq j < n$, then $e(i,j)$ and $e(i,j+1)$ cannot be utilized for row rerouting. This is due to the fact that $e(i,j)$ cannot be connected to any PEs on its right via $rs(i,j)$, while no PEs can be connected to the left of $e(i,j+1)$. Similarly, if a column switch $cs(i,j)$ is faulty, both $e(i,j)$ and $e(i+1,j)$ cannot be used for column routing, where $1 \leq i < m$, $1 \leq j \leq n$.

In this paper, faults composed of PE faults and switch faults. As proved in [8], constructing a $r \cdot s$ sized subarray from an $m \times n$ host array with only PEs faults is NP-Hard. When considering switch faults, this reconfiguration problem becomes more complex. This is because the switches faults affect the connectivity of circuits and cannot be bypassed. Therefore, our goal is to find solutions for the following problem, which is NP-hard and more complex than the existing solutions that do not consider switch faults.

Problem $\mathcal{P}$: *Given an $m \times n$ host array H with faults (including PE faults and switch faults), positive integers r and c, find an $m' \times n'$ fault-free logical array T under the constraint of row and column rerouting such that $m' \geq r$ and $n' \geq c$.*

III. Proposed Algorithms

As problem $\mathcal{P}$ is NP-hard, we focus on generating approximate solutions in polynomial time. For PE faults only, both GCR [12] and LDP [14] can reconstruct a maximum target array (MTA) with selected rows under row bypass and column routing scheme in linear time. The proposed algorithms in this paper also employ GCR as a sub-algorithm. The following briefly describes the GCR algorithm. For a given $m \times n$ host array *H*, let $R_1, R_2, \cdots, R_m$ be the rows. To simplify the description of GCR and without loss of generality, we also assume that the *MTA* produced by *GCR* contains the same selected rows. Let $col(u)$ and $col(v)$ be the physical column index of PE u and v respectively. All operations of GCR are carried out on adjacent sets of a fault-free PE u in row i, where $1 \leq i < m$. Based on the limitation of *compensation distance*, the adjacent set $\mathcal{A}dj(u)$ of PE u is defined as follows:

$\mathcal{A}dj(u) = \{v : v \in R_{i+1}, v$ is fault-free and $|col(u) - col(v)| \leq 1\}$.

GCR constructs the target array in a left-to-right manner. It begins by selecting the leftmost fault-free element, say u of the first row R_1 for inclusion into a logical column. Next, the leftmost element in $Adj(u)$, say v, is connected to u. This process is repeated until a logical column is fully constructed. In each iteration, *GCR* produces the current leftmost logical column. In the next iteration, GCR attempts to construct a new logical column by selecting the leftmost PE in R_1 that has not been examined and the entire process described above is repeated. GCR finally terminates when all PEs in R_1 have been examined. Upon completion, there are k logical columns in the logical array L, denoted as $L_1, L_2, \cdots, L_k$. Further details about GCR can be found in [11].

In this section, we propose two novel algorithms to resolve problem $\mathcal{P}$, namely 1) Column Routing and Row Releasing (CRRR), 2) Row Connecting and Column Routing (RCCR). *Row Bypass* and *Column rerouting* scheme are utilized in the first approach, while the second approach employs the *Row and Column rerouting* scheme.

A. Column Rerouting and Row Releasing

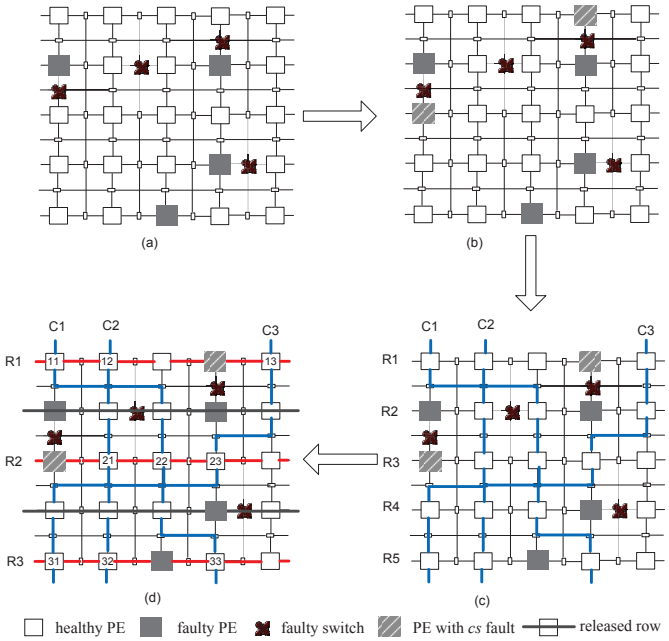

Fig. 3: Column Routing and Row Releasing .

In CRRR, we first employ a *column switch preprocessing* procedure to remove redundant PEs due to faulty column switches on the host array H. For each faulty column switch, the healthy PEs which are directly connected to it will be assigned as faulty PEs. The host array H after *column switch preprocessing*, is denoted as H^C. Next, we employ GCR

to reconstruct a maximum target array (MTA) from H^C. In this process, the connectivity of each logical column in MTA can be guaranteed, as PEs that are adjacent to faulty column switches cannot be utilized to form any logical column (since they are assumed to be faulty). However, GCR does not take into consideration faulty row switches. Therefore, the MTA still contains faulty row switches. In order to overcome this problem, the rows in the MTA that contain row switch faults are removed (released) and the healthy PEs on released rows allow data to be bypassed. As a result of this, a target array without PE faults and switch faults can be generated.

Figure 3 describes the various steps of CRRR for a given host array. Fig. 3(a) shows the original 5×5 host array with PE faults and switch faults. Each box signifies a PE, and black boxes represent faulty PEs. Circle with a cross refers to a faulty switch. CRRR first identifies the healthy PEs with column switch faults. Fig. 3(b) shows the host array H^C after *column switch preprocessing*, whereby 2 redundant PEs (boxes with diagonal strips) are identified. GCR is then employed to construct a MTA from H^C. Fig. 3(c) reveals a 5×3 target array generated by GCR that still consists of row switch faults. By releasing the rows with switch faults, a 3×3 healthy subarray is obtained in Fig. 3(d). Detailed description of *CRRR* is shown in Algorithm 1.

Algorithm 1 CRRR

Input:
Mesh-connected $m \times n$ sized Host array H with faults;
$m \times (n-1)$ sized Row Switch array RS with faults;
$(m-1) \times n$ sized Column Switch array CS with faults.
Output:
$r \times s$ sized Target subarray $T \leftarrow \{T_1, T_2, \ldots T_s\}$.
Steps

for all faulty column switch $cs(i,j) \in CS$ **do**
  /* identify redundant PEs due to RS faults on H*/
  if ($e(i,j)$ is fault-free) **then**
    $e(i,j) \leftarrow$ faulty;
  end if
  if ($e(i+1,j)$ is fault-free) **then**
    $e(i+1,j) \leftarrow$ faulty;
  end if
end for
k $\leftarrow 0$; /* k is the number of logical columns */
r $\leftarrow m$; /* r is the number of logical rows */
/*s is final number of logical columns of T */
$s \leftarrow$ GREEDY_COLUMN_ROUTING(H, m, n, T, k);
for **i** $\leftarrow 1$ to m **do**
  if exist faulty row switch in R_i **then**
    release all PEs in row i of T;
    r $\leftarrow r - 1$; /* decrease the number of logical rows by 1.*/
  end if
end for
return T; ▷ Return the final $r \times s$ logical array

The algorithm CRRR consists of three procedures, namely *column switch preprocessing*, *GCR* and *row releasing*. The preprocessing process takes $O(1)$ running time as only two neighboring PEs of the faulty switch need to be updated. The time complexity of GCR is $O(N)$ [11], where N represents the number of PEs in host array. According to [17], the process of *row releasing* also takes $O(1)$ running time. Hence, the time required by CRRR is bounded by $O(N)$.

B. Row Connecting and Column Rerouting (RCCR)

In this subsection, we propose a heuristic algorithm for reconfiguration problem $\mathcal{P}$ under row and column rerouting constraints. In contrast to the CRRR technique where healthy PEs in rows with switch faults are discarded, the proposed technique in this subsection enables row rerouting. Row and column rerouting provides more flexibility in the reconfiguration as healthy PEs in rows with switch faults can be utilized to construct logical rows, which may lead to notable increase in the size of target array. We named the proposed technique for row and column rerouting as *Row Connecting and Column Routing* (RCCR).

In RCCR, we first identify the logical rows. For a given host array H, only row switch faults are taken into consideration initially and PE faults are ignored. The adjacent PEs of each row switch $rs(i,j)$ fault in H, namely $e(i,j)$ and $e(i,j+1)$, are set to faulty in the first step.

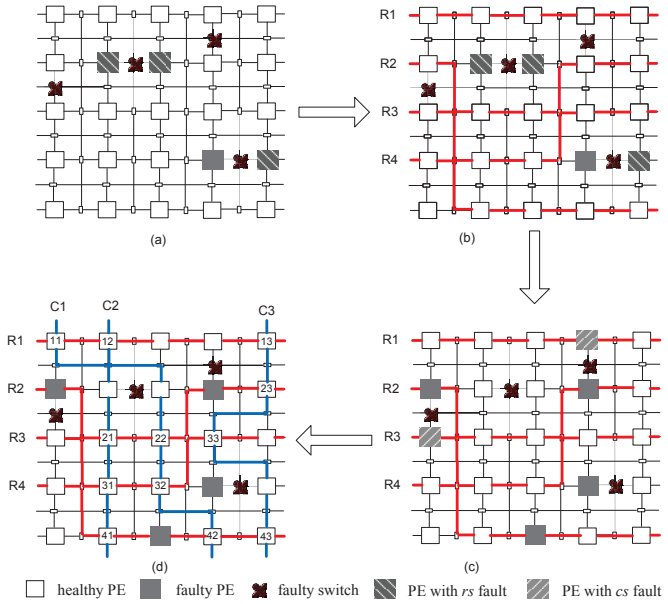

Fig. 4: Row Connecting and Column Routing.

Figure 4 shows the steps for RCCR that is based on the example host array in Fig. 3(a). Fig. 4(a) illustrates the array faults after *row switch preprocessing*, denoted as H^R, where faulty PEs consist of only those that are adjacent to row switch faults. Boxes with backslashes represent the healthy PEs that are adjacent to row switch faults, which will not be considered in row rerouting. An algorithm namely *Greedy Row Rerouting* (GRR) was developed to construct maximum logical rows in

H^R. GRR works in the same manner as GCR except that it construct logical rows in a top-to-down manner. Let $row(u)$ and $row(v)$ denote the physical row index of PE u and v, respectively. The right adjacent sets $\mathcal{A}dj^R(u)$ of each fault-free PE u in column C_j, $1 \leq j < n$, can be defined as follows:

$\mathcal{A}dj^R(u) = \{v : v \in C_{j+1}$, v is fault-free, and $|row(u) - row(v)| \leq 1\}$.

In H^R, GRR begins by selecting the uppermost fault-free PE u of the first column to be included in the logical row R_1. Next, the uppermost successor PE v in $Adj^R(u)$ will be selected for inclusion into the logical row R_1. This process to construct R_1 is repeated until a PE in the last column is chosen. In each step, GRR attempts to connect current PE u to its uppermost successor in $Adj^R(u)$ that has not been examined. GRR terminates when all PEs in the first column have been examined. Upon termination, r logical rows are generated, denoted as $R_1, R_2, \cdots, R_r$. Fig. 4(b) shows 4 logical rows constructed from H^R using GRR. As GRR only considers faulty PEs that are adjacent to row switch faults, some faulty PEs in the original host array may exist in the logical rows. For example in Fig. 4(c), a faulty PE (i.e. $e(2,1)$) is located within logical row R_2. Let's denote the logical rows that are constructed by GRR on H^R as *connected rows*.

After generating *connected rows*, we mark the *connected rows* on original array H and perform *column switch preprocessing* as described in III-A. Fig. 4(c) shows the host array H^C after this preprocessing. Next, column rerouting is performed. In order to construct logical columns, an attempt is made to connect the *connected rows*. This process is more complex than GCR as a fault-free PE (excluding those adjacent to column switch faults) in connected row R_i, $1 \leq i < r$, may not be able to connect directly to a fault-free PE in next *connected row* R_{i+1}.

Let's define the successor $succ(u)$ of a fault-free PE $u \in R_i$, $1 \leq i < r$, taking into account the *compensation distance*, as:

$succ(u) = \{v : v$ is fault-free, $v \in R_{i+1}$, and $|col(v) - col(u)| < (row(v) - row(u))\}$.

We devised a procedure called *Column Rerouting on Connected Rows*, denoted as CRCR, to optimally solve this problem in linear time by employing a greedy algorithm. CRCR constructs the logical column from top-to-down in the left-to-right manner. The fault-free PEs from the *connected rows* in H^C are routed to form logical columns. CRCR begins by selecting the leftmost PE u of the first *connected row* R_l, which is located within the first row or is able to connect to a PE in the first row (the second condition is necessary for the target array to obtain external inputs), for inclusion into a logical column. Next, the leftmost successor v in $succ(u)$ will be included in the logical column C_1. In each step, CRCR attempts to connect the current element v to leftmost fault-free PE w in $succ(v)$ that has not been examined previously. If CRCR fails in doing so, no logical column that contains v can be formed. When this happens, CRCR backtracks to the previous PE u, which was connected to v, and attempts to connect u to its leftmost successor (excluding v) in $succ(u)$ that has not been examined and removes v from C_1. This process is repeated until either **i)** a PE in the last connected row R_r is connected to the previous one and it is able to connect to a PE in the last physical row (the second condition is necessary so that the target array can produce an external output), or **ii)** CRCR backtracks to u in R_1. Termination under condition i) results in the construction of a logical column that passes through each of the connected rows. In termination under condition ii), no logical column that begins with u can be formed. In the next iteration, CRCR attempts to construct a new logical column by selecting the leftmost fault-free PE in R_l that has not been examined and the entire process described above is repeated. CRCR terminates when all PEs in R_l have been examined. Finally, s logical columns are generated, denoted as $C_1, C_2, \cdots, C_s$. Fig. 4(d) shows the resulting 4×3 target array. Detailed description of the algorithm RCCR is shown in *Algorithm 2*.

RCCR is composed of *row switch preprocessing*, *GRR*, *column switch preprocessing* and *CRCR*. Step 1 and 3 can be achieved concurrently and are bounded by $O(1)$ as pointed out in section III-A. GRR works in the same way as GCR, which is bounded by $O(N)$. The procedure CRCR is applied on the selected rows, and the unused healthy PEs are automatically bypassed during column rerouting. In CRCR, the leftmost unexamined PE in $succ(u)$ will be selected as a healthy PE u in selected rows. Noting that $Adj(u)$ in GCR can be found in $O(N)$, we conclude that the time complexity of CRCR is also bounded by $O(N)$. From the above analysis, it can be concluded that the time complexity of RCCR is $O(2N)$.

C. Optimality of proposed algorithm

Theorem 1. Row Connecting and Column Rerouting (RCCR) algorithm produces target array with maximum number of logical rows for Problem $\mathcal{P}$.

Proof. We developed a procedure Greedy Row Rerouting (GRR) to construct the connected rows on the host array H^R after *row switch preprocessing*. In H^R, we ignore all the PE faults and only deal with the faults caused by faulty row switches. The number of PEs that can be used in row connecting is maximum. As the selection of the uppermost element for inclusion into a logical row at each step of the algorithm is based on a greedy approach, this maximizes the opportunity for the remaining fault-free elements to form logical rows. Similarly, if rerouting is performed in column direction first, we can generate a target array with maximum logical columns. We can therefore prove the above theorem based on the same reasoning described in [11] for proving the optimality of GCR algorithm,

IV. Experimental Results

The algorithms CRRR and RCCR were implemented in C and executed on a 1.6 GHz Intel Core 2 Duo CPU with 2 GB RAM. In these experiments, we have used the same assumptions in [10-21], i.e., the faults of random host arrays were generated by a uniform random generator. Both algorithms are tested and compared with each other on the same input instances. In our simulations, random PE faults and switch

Algorithm 2 RCCR

Input:
Mesh-connected $m \times n$ sized Host array H with faults;
$m \times (n-1)$ sized Row Switch array RS with faults;
$(m-1) \times n$ sized Column Switch array CS with faults.
Output:
r connected rows $R \leftarrow (R_1, R_2, \ldots R_r)^T$
s logical columns $C \leftarrow (C_1, C_2, \ldots C_s)$
$r \times s$ sized Target sub-array T.

procedure CRCR(H^C, m, n, R, C, r, s)
  for i $\leftarrow 1$ to r **do**
    $E_i \leftarrow$ set of fault-free PE in R_i of H^C; ▷ Assume the PEs in E_i are arranged in increasing column numbers.
  end for
  for i $\leftarrow 1$ to $r-1$ **do**
    for each PE $u \in E_i$ **do**
      $succ(u) \leftarrow \{v : v \in E_{i+1}, rol(v) > row(u), |col(u) - col(v)| \leq (rol(v) - row(u))\}$; ▷ Assume the PEs in $succ(u)$ are arranged in increasing column numbers.
    end for
  end for
  all PEs in $E \leftarrow$ unmarked;
  while there are unmarked PEs in E_1 **do**
    $cur \leftarrow$ leftmost unmarked PE in E_1 and be able to connect to first row;
    mark cur;
    repeat
      if (there are unmarked PEs in $succ(cur)$) **then**
        $w \leftarrow$ leftmost unmarked PE in $succ(cur)$;
        $prev(w) \leftarrow cur$; ▷ w is connected to cur.
        $cur \leftarrow w$;
        mark w;
      else if cur not in E_1 **then**
        $cur \leftarrow priv(cur)$;
      end if
    until ($cur \in E_r$ and cur can connect to last row) or ($cur \in E_1$)
    if ($cur \in E_r$ and cur can connect to last row) **then**
      $s \leftarrow s + 1$; ▷ increase number of logical columns by 1.
      repeat
        add cur to C_s;
        $cur \leftarrow priv(cur)$;
      until ($cur \in E_1$)
    end if
  end while
end procedure

Begin

/* Performing *column switch preprocessing* on H ignored PEs faults.*/
for all faulty row switch $rs(i,j) \in RS$ **do**
  if ($e(i,j)$ is fault-free) **then**
    $e(i,j) \leftarrow$ faulty;
  end if
  if ($e(i,j+1)$ is fault-free) **then**
    $e(i,j+1) \leftarrow$ faulty;
  end if
end for
$H^R \leftarrow$ the H after *row switch preprocessing*;
/* Generate connected rows R from H^R by GRR.*/
GREEDY_ROW_ROUTING(H^R, m, n, R, r);
perform *column switch preprocessing* on H.
$H^C \leftarrow$ the H after *column switch preprocessing*.
mark the *connected rows* on H^C.
s $\leftarrow 0$; /* s is number of logical columns of C */
/* Column Rerouting on the connected rows from H^C*/
CRCR(H^C, m, n, R, C, r, s);
End

faults with uniform distribution are placed on the host array H. Data collected for host arrays with different fault densities are averaged over 20 random instances, with the decimal points rounded off to the nearest integer in all cases.

Let *harvest* indicates the size of the final target array generated by the proposed algorithms. This can be used to compare the performance of the proposed algorithms. The improvement (*imp*) of RCCR over CRRR in terms of *harvest* is evaluated by the following formula:

$$imp = (\frac{harvest\_of\_RCCR}{harvest\_of\_CRRR} - 1) \times 100\%$$

In our empirical study, we have used realistic fault densities of both PEs and switches that ranges from 0.0% to 1%. Without loss of generality, our analysis is based on 128×128 host arrays.

Fig. 5 shows the performance of CRRR and RCCR on 128×128 host array with different PE faults and switch faults. It is evident that for both CRRR and RCCR, the switch faults has a higher impact on the harvest when compared to PE faults. For a 128×128 host array with 0.0% PE faults, CRRR produces 14228, 8358 and 4086 healthy PEs in the target array when the switch faults are 0.1%, 0.5% and 1.0% respectively. The variation in the harvest are marginal when the host array has 0.0% switch faults. In particular, for 128×128 host array with 0.0% switch faults, CRRR produces harvests of 16140, 15936 and 15699 when the PE fault density is 0.1%, 0.5% and 1.0% respectively. The same can be observed for RCCR. It can also be observed that the harvests are gradually degraded with the increased in PE faults for both CRRR and RCCR. However, compared to switch faults, the decrease in harvest caused by PE faults is rather slow. This is evident in Fig. 5 where the degradation of harvest due to increasing switch faults for a fixed PE fault density is significant. This indicates that switch faults have greater impact in the reconfiguration on host array as compared to PE faults. This is due to the fact that the switch faults affect the connectively of host array and cause the adjacent healthy PEs to be redundant and hence, they cannot be included in the target array. On the other hand,

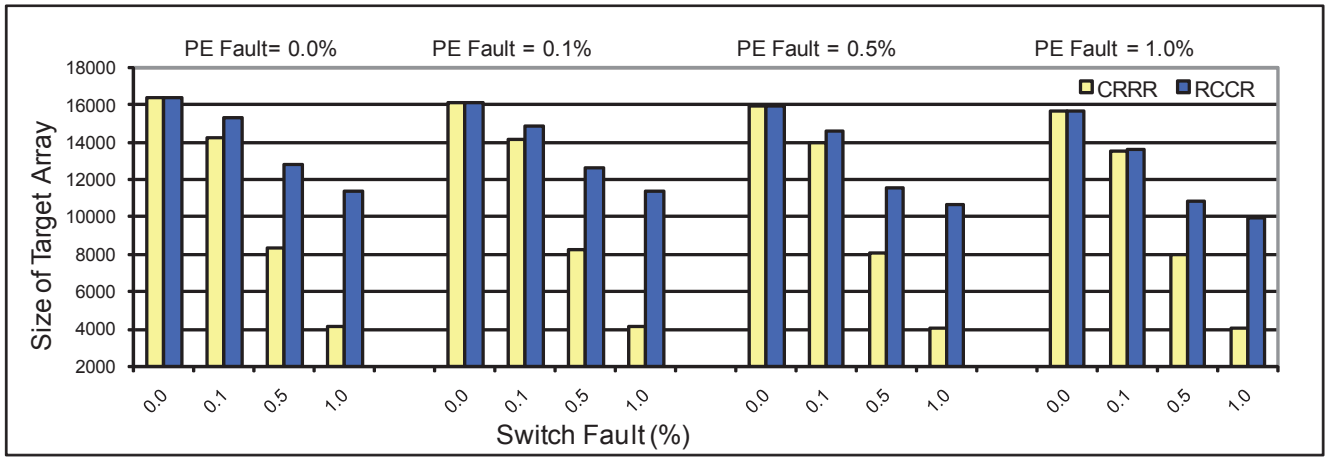

Fig. 5: Comparison in *harvest* for CRRR and RCCR, on 128×128 host arrays with different PE and switch faults.

PE faults can be bypassed.

Fig. 6 shows that the harvest improvement of RCCR over CRRR gradually decreases with increasing PE fault density. It is evident that larger number of switch defects in host array leads to higher number of unusable PEs in rerouting. Hence, the harvest will decrease with increasing switch fault rate. For instance, *imp* is about 52.6% in the case of 0.1% PE faults and 0.5% switch faults. With the same switch fault density (i.e. 0.5%), *imp* decrease to about 43.8% and 36.3% when PE faults are 0.5% and 1.0% respectively. This is due to the fact that the number of fault-free PEs in released rows becomes less with increasing PE faults.

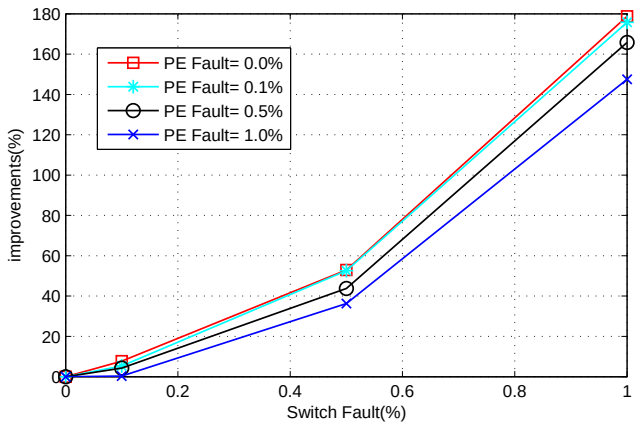

Fig. 6: Improvements of RCCR over CRRR in terms of *harvest*, on 128×128 host arrays with different PE and switch faults.

Figure 6 clearly shows that RCCR outperforms CRRR. Moreover, the improvements over CRRR increase with increasing density of switch faults. This is due to the fact that the harvest of CRRR is significantly reduced with the increase of switch faults density. In Fig. 5, for 128×128 host array with 0.5 percent PE faults, both RCCR and CRRR can achieve the same harvest (*i.e.*, 15936) when no switch failure occur. For the switch faults densities of 0.1%, 0.5% and 1%, the harvests of CRRR are 14012, 8012 and 4013, respectively. However for RCCR, the decrease in harvest are not as significant (i.e. 14616, 11525 and 10666). As shown in Fig. 6, when the switch fault density is below 0.1 percent, RCCR generates only slightly more harvest than CRRR, and the improvements over CRRR is no more than 5%. However, for switch fault density above 0.1 percent, the improvement (*imp*) in harvest becomes increasingly significant with the increase in switch fault density. For switch fault density of 0.5 percent, the *imp* over CRRR is about 43.8%. When the switch fault density increases to 1 percent, the harvest of RCCR is about 165.8% more than that of CRRR. Hence, it is evident that the RCCR performs more favorably than CRRR, especially when the switch faults are increased. This reason for this is as follows: CRRR constructs the target array in the column direction only, and the rows with switch faults are simply removed from the target array. On the other hand, RCCR constructs the target array on the *connected rows*, where healthy PEs in rows that contains switch faults can be utilized to maximize the size of the target array.

Table 1 reveals the runtime of the reconfiguration algorithms CRRR and RCCR in milliseconds (ms). The runtime consists of all operations that are required to perform the reconfiguration. Experimental results indicates that the running time of RCCR is nearly twice of that required by CRRR. This is consistent with the complexity analysis. While CRRR can compute faster than RCCR, the execution time for RCCR is still acceptable. For a 128×128 host array with 1% PE faults and 0.5% switch faults, RCCR only requires about 42.2 milliseconds. In addition, RCCR can produce greater harvest than CRRR. In particular, RCCR is able to produce a target array with maximum number of logical rows in linear time for the reconfiguration problem P. For the case of host array with 1% switch faults and 0.1% PE faults, *imp* could reach 175.8 percent (as shown in Fig. 6). Finally, the improvement

TABLE I: Running time comparisons between CRRR and RCCR on 128×128 host arrays with different PE and switch faults.

Host Array	Algorithm	Running Time (ms)			
PE Fault	CRRR	Switch Fault (%)			
(%)	RCCR	0.0	0.1	0.5	1.0
0.0	CRRR	21.4	22.8	20.3	19.6
	RCCR	43.8	41.5	42.8	44.5
0.1	CRRR	21.2	20.3	22.0	20.3
	RCCR	41.5	41.4	42.9	41.3
0.5	CRRR	20.3	21.8	20.9	21.8
	RCCR	43.8	39.1	41.4	40.7
1.0	CRRR	21.3	19.5	22	18.8
	RCCR	42.2	42.2	42.2	42.2

becomes more significant when the number of switch faults increase.

V. Conclusion

In this paper, we have introduced the problem of reconfiguring two dimensional degradable VLSI arrays in the presence of PE faults and switch faults. Two algorithms, CRRR and RCCR were proposed to tackle the reconfiguration problem of VLSI arrays with PE and switch faults. We demonstrated that RCCR performs more favorably than CRRR, especially for the case of the host array with large number of switch faults. We have shown that RCCR is able to produce an optimal solution in linear time. Experimental results indicate that switch faults have greater impact on the size of the target array as compared to PE faults. This is due to the fact that the switch faults affect the connectivity of host array by causing the adjacent healthy PEs to be redundant and thus, they cannot be included in the target array. These results clearly demonstrate the importance of considering switch faults for efficient reconfiguration of VLSI arrays.

Acknowledgment

This work was supported by the National Science Foundation of China under Grant No. 60970016 and No. 61173032.

References

[1] A. Estrella-Balderrama, J. A. Fernandez-Zepeda, A. G. Bourgeois, Fault Tolerance and Scalability of the Reconfigurable Mesh, In Proc. of the 18th International Parallel and Distributed Processing Symposium (IPDPS), 2004, pp: 172-177.

[2] R. Negrini, M. G. Sami and R. Stefanelli, *Fault tolerance through reconfiguration in VLSI and WSI arrays*. The MIT Press, 1989.

[3] S. Y. Kuo and W. K. Fuchs, Efficient spare allocation for reconfigurable arrays, *IEEE Design and Test*, Feb. 1987, vol. 4, no, 7, pp. 24-31.

[4] C. W. H Lam, H. F. Li and R. Jakakumar, A study of two approaches for reconfiguring fault-tolerant systoric array, *IEEE Trans. on Computers*, June 1989, vol. 38, no. 6, pp. 833-844.

[5] Y. Y. Chen, S. J. Upadhyaya and C. H. Cheng, A comprehensive reconfiguration scheme for fault-tolerant VLSI/WSI array processors, *IEEE Trans. on Computers*, vol. 46, no. 12, Dec. 1997, pp. 1363-1371.

[6] I. Takanami, Built-in self-reconfiguring systems for fault tolerant mesh-connected processor arrays by direct spare replacement, In Proc. of IEEE International Workshop on Defect and Fault Tolerance in VLSI Systems, 2001, pp. 134-142.

[7] L. LaForge, Extremely Fault Tolerant Arrays. In: Proceedings of. Int'l Conf. Wafer Scale Integrations, 1989, pp. 365-378.

[8] S. Y. Kuo and I. Y. Chen, Efficient reconfiguration algorithms for degradable VLSI/WSI arrays, *IEEE Trans. on Computer-Aided Design*, Oct. 1992, vol. 11, no, 10, pp.1289-1300.

[9] M. D. Smith and P. Mazumder, Generation of Minimal Vertex Covers for Row/Column Allocation in Self-Repairable Arrays, *IEEE Trans. on Computers*, vol. 45, no. 1, pp. 109C115, Jan. 1996.

[10] C. P. Low and H. W. Leong, On the reconfiguration of degradable VLSI/WSI arrays, *IEEE Trans. on Computer-Aided Design of integrated circuits and systems*, Oct. 1997, vol. 16, no. 10, pp. 1213-1221.

[11] C. P. Low, An efficient reconfiguration algorithm for degradable VLSI/WSI arrays, *IEEE Trans. on Computers*, June 2000, vol. 49, no. 6, pp. 553-559.

[12] W. Jigang, T Srikanthan. An improved reconfiguration algorithm for degradable VLSI/WSI arrays, *Journal of Systems Architecture*, 2003, 49(1-2):23-31.

[13] W. Jigang, T Srikanthan. Accelerating reconfiguration of degradable VLSI arrays. *IEE Proceedings, Circuits, Devices and Systems*, 2006, 153(4): 383-389

[14] W. Jigang and T. Srikanthan, Fast Reconfiguring Mesh-Connected VLSI Arrays, Proc. IEEE Intl Symp. Circuits and Systems (ISCAS'04), vol. II, May 2004 , pp. 949-952.

[15] M. Fukushi and S. Horiguchi, Reconfiguration Algorithm for Degradable Processor Arrays Based on Row and Column Rerouting, Proc. 19th IEEE Intl Symp. Defect and Fault Tolerance in VLSI Systems (DFT'04), 2004, pp. 496-504.

[16] M. Fukushi, Y. Fukushima, S. Horiguchi, A genetic approach for the reconfiguration of degradable processor arrays. In: Proceedings of 20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 2005, 63-71

[17] W. Jigang, T. Srikanthan, H. Xiaogang, Preprocessing and Partial Rerouting Techniques for Accelerating Reconfiguration of Degradable VLSI Arrays. *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, 2010, 3(2): 315-319

[18] W. Jigang, T. Srikanthan , W. Xiaodong, Integrated Row and Column Rerouting for Reconfiguration of VLSI Arrays with Four-Port Switches, *IEEE Trans. on Computers*, 2007, 56(10): 1387-1400.

[19] W. Jigang and T. Srikanthan, Reconfiguration Algorithms for Power Efficient VLSI subarrays with 4-port Switches, *IEEE Trans. on Computers*, MAR 2006, vol. 55, no. 3, pp. 243-253.

[20] W. Jing , W. Jigang, Z. Yuanrui, J. Guiyuan. Accelerating Reconfiguration for Degradable Mesh-connected Processor Arrays. In: Proceedings of The 3rd International Symposium on Parallel Architectures, Algorithms and Programming (PAAP), 2010, 55-58.

[21] W. Jigang, N. Zhipeng, Z. Yuanbo, T. Srikanthan, S. Qingnin. Reconfiguration algorithm for low temperature sub-array on VLSI/WSI arrays with faults. In: Proceedings of The 18th International Symposium on Physical and Failure Analysis of Integrated Circuits (IPFA), Korea, 2011, 203-206

[22] N. R. Mahapatra and S. Dutt. Hardware-efficient and highly reconfigurable 4- and 2-track fault-tolerant designs for mesh-connected arrays, *Journal of Parallel and Distributed Computing*, 2001, 61(10): 1391-1411,.

[23] M. Fukushi, S. Horiguchi. A self-reconfigurable hardware architecture for mesh arrays using single/double vertical track switches, IEEE Transactions on Instrumentation and Measurement, 2004, 53(2): 357-367.

[24] I. Takanami. Self-reconfiguring of 1.5-track-switch mesh arrays with spares on one row and one column by simple built-in circuit, *IEICE Trans. on Information and Systems*, 2004, E87-D(10): 2318-2328.

[25] Wu Jigang and T. Srikanthan, Schroder Heiko. Efficient Reconfigurable Techniques for VLSI Arrays with 6-port Switches, *IEEE Trans. on VLSI Systems*, 2005, 13(8):976-979.