Efficiency of Flexible Rerouting Scheme for Maximizing Logical Arrays

Guiyuan Jiang[1], Jigang Wu[2], and Jizhou Sun[1]

[1] School of Computer Science & Technology, Tianjin University, Tianjin 300072, China
[2] School of Computer Science & Software Engineering, Tianjin Polytechnic University, Tianjin 300387, China
`{jguiyuan,asjgwu}@gmail.com, jzsun@tju.edu.cn`

Abstract. In a multiprocessor array, some processing elements (PEs) fail to function normally due to hardware defects or soft faults caused by overheating, overload or occupancy by other running applications. Fault-tolerant reconfiguration considered in this paper is to reorganize fault-free PEs from a processor array with faults to a logical array of regular mesh topology by changing the interconnections among PEs. This paper presents the efficiency of the flexible rerouting scheme to maximize the usage of the fault-free PEs, by developing an efficient reconfiguration algorithm without backtracking. The proposed algorithm constructs each logical columns from left to right on candidate PE sets. It updates the candidate sets by excluding the PEs which cannot be used, once a logical column is formed. Also, it is proved that the proposed heuristic algorithm is able to generate the maximum-size logical array in linear time. Experimental results show that 123 logical columns can be constructed on 256×256 host arrays with fault density of 30%, resulting in an improvement of 51% in comparison to the previous algorithm by which only 82 logical columns can be produced. Furthermore, our algorithm is able to generate target arrays with harvest over 56% on host arrays with fault density of 50%, while the previous work cited in this paper fails to construct any target array in this case.

Keywords: Interconnection network, reconfiguration, rerouting scheme, fault tolerant, algorithm.

1 Introduction

The quest for high-performance and low-power leads to design multi-core architectures where an increasing number of processing elements (PEs) are integrated on a single chip in a tightly coupled fashion. In order to fully exploit the processing capabilities offered by the integration of an increasing number of PEs, on-chip communication networks play a critical role in developing high-performance embedded computing system. The option of reconfiguring the interconnect topology using optical circuit switches, which are of low-cost, low-power and high-bandwidth, opens up new possibilities for improving not only the computing performance but also the reliability of multiprocessor systems.

To fully exploit the processing capabilities, many works have investigate application-aware topology reconfiguration techniques which try to customize the multiprocessor array to a topology that matches the traffic pattern of the application in order to reduce the power consumption and message latency etc. Optimizing network topology and mapping cores to network are two important application-specific reconfiguration methods. Topology determines the connectivity of the Network-on-Chip (NoC) nodes, while the core mapping tries to place the processing cores communicating more frequently near each other. The problem of topology selection and core mapping on NoC nodes have been explored to optimize an interconnection network based on a set of communication constraints by many researchers [1][2]. Reconfiguring the topology on NoC to reduce power consumption has been studied in [3]. And the work in [4] extends the paper [3] by proposing a more flexible and efficient structure and by providing extensive evaluations to thoroughly investigate the power, latency, and area of the reconfigurable NoC.

Since these fully customized topologies are designed and optimized for some specific applications, they give the best performance and power results only for that applications. The fault-tolerant reconfiguration considered in this paper is to re-organize the fault-free PEs in the multiprocessor array to a regular topology (standard structured) which ensuring well-controlled electrical parameters. Generally, two distinct types of fault-tolerant approaches, i.e. the redundancy approach and the degradation approach, are mostly investigated for processor array reconfiguration. The redundancy approach tolerates faulty PEs by replacing them with spare PEs [5][6], and if these PEs cannot replace all the faulty ones, the chip has to be discarded. In degradation approach, [7] studied the problem of 2D mesh reconfiguration under three different routing constraints. They have shown that most problems that arise under these constraints are NP-complete and they also proposed some heuristic reconfiguration algorithms for these problems. An algorithm, namely GCR, was proposed in [8] to find a maximal logical array (MLA) that contains a set of the selected rows. The techniques performing row-exclusion and compensation were proposed and combined with GCR into a heuristic algorithm to generate an approximate MLA [9]. Recently, the power dissipation of a logical array was reduced in [10] by reducing the number of long-interconnects utilizing a dynamic programming approach. In [11], a genetic algorithm (GA) for the reconfiguration of degradable mesh arrays is presented to evolve rerouting strategies for constructing logical rows/columns. In [12], novel techniques are presented to accelerate reconfiguration by employ flexible upper and lower bounds for the maximum logical array (MLA). The degradable reconfiguration approach for three-dimensional mesh-connected processor arrays is investigated in [13][14].

In a logical array, two neighboring PEs are interconnected through a group of physical links and these physical links form a interconnect path (interconnect for short) between the two PEs. Note that this architecture implements reconfiguration using simple switches, thus overlap of physical links is not allowed between any two interconnects, i.e., no physical link is shared by any two interconnects of a feasible logical array. In order to generate logical arrays without overlap, most previous works develop reconfiguration algorithms under rerouting schemes with limitation on the routing distance d (see section 2.1 for formal definition), which means that if the distance between two PEs exceeds d, then the two PEs are not allowed to connect to form a logical array. This has brought convenience for designing reconfiguration algorithms, but it leads to large number of unused fault-free PEs, which is a serious waste of computing resources. This motivates us to increase the utilization of fault-free PEs by adopting flexible rerouting scheme. In this paper, we first investigate the performance loss of previous works in terms of utilization of fault-free PEs, then we adopt a flexible rerouting scheme in reconfiguration resulting in an efficient non-backtracking algorithm, to produce maximum logical arrays under the rerouting scheme.

2 Preliminaries

2.1 Fault-Tolerant Architecture and Rerouting Schemes

Let H denotes the physical (host) array on which some of the elements are defective due to hardware/circuit defects or soft faults which means temporary unavailability caused by overheating, overload or occupancy by other applications. Assume the fault density of the physical array is ρ, then there are $N = (1-\rho)\cdot m\cdot n$ fault-free PEs in a $m \times n$ physical array. The rows and columns in physical array are called physical rows and columns respectively. A logical subarray which contains no faulty PEs constructed by changing states of reconfiguration switches is called a logical array (target array), denoted as T. The rows and columns in logical target array are called logical rows and columns respectively. Throughout this paper, $e_{i,j}(e'_{i,j})$ indicates the PE located at the position of (i, j) of the host (logical) array, where i is its row index and j is its column index. $row(u)(col(u))$ denotes the physical row (column) index of the PE u. u=v denotes that u is identical to v.

Fig. 1 shows the fault-tolerant architecture for a 4×4 host array. The fault-tolerant reconfiguration is achieved by inserting several reconfiguration switches in the network allowing the network to flexibly change the connections among PEs. Each square box in the host array represents a processing element (PE), whereas each circle represents a reconfiguration switch. In all figures in this paper, the gray shaded boxes represent faulty PEs while unshaded ones represent the fault-free PEs. There are 4 states for each switch.

In order to generate logical arrays without overlaps among interconnects, two software control schemes, i.e. bypass and rerouting schemes, are utilized to guide reconfiguration algorithms. In particular, row bypass and column rerouting schemes are mostly employed in previous works. As shown in Fig.1, if PE $e_{i,j}$ is faulty, then PE $e_{i,j-1}$ can communicate with PE $e_{i,j+1}$ and data will bypass $e_{i,j}$ through an internal bypass link. This scheme is called *row bypass*. In *column rerouting* scheme, PE $e(i, j)$ could connect to $e(i + 1, j')$ by changing states of related switches where $|j - j'| \leq d$ and d is called the compensation

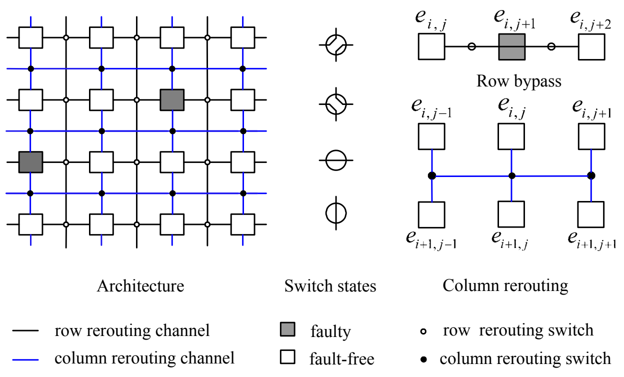

Fig. 1. Switch functions and rerouting manners on a 4 × 4 mesh linked by four-port switches

distance [8][9]. If d is limited to 1, then PE $e_{i,j}$ can connect to one of the three PEs, i.e. $e_{i+1,j-1}$, $e_{i+1,j}$ and $e_{i+1,j+1}$, to form a logical column. The three PEs are called neighbors of $e_{i,j}$, and set of the three PEs is denoted as $\mathcal{A}dj(e)$. *column bypass* and *row rerouting* schemes can be similarly defined. Generally, in order to reduce the complexity of the switching mechanisms and keep low cost of physical implementation, it is necessary to keep d small.

2.2 Construct Target Arrays on Selected Rows

Problem $\mathcal{R}$. *Given an $m \times n$ mesh-connected processor array in which some of its elements are faulty, find a maximum target array that contains the selected rows under the row bypass and column rerouting scheme.*

Let R_1, R_2, ..., R_m be the rows of the given host array. Assume R_{r_1}, R_{r_2}, ..., R_{r_s} are the s selected rows to construct a target array, where $1 \leq r_s \leq m$. Logical columns can be constructed under column rerouting scheme on selected rows, and interconnecting these logical columns under row bypass scheme forms a target array. A logical array T is said to contain $R_{r_1}, R_{r_2}, \cdots, R_{r_s}$ if each logical column in T contains exactly one fault-free PE from each of the selected rows. Note that, if a physical row is not selected for inclusion into target array, all PEs in the row will be bypassed.

Suppose C_p and $C_q (C_p \neq C_q)$ are two logical columns constructed on the s selected rows, thus each of them is consists of s PEs such that one and only one PE lies on each selected row. Let $C_p(i)$ denotes the PE lying in the i-th selected row of column C_p and $C_q(i)$ denotes the PE lying in the i-th selected row of column C_q, for $(1 \leq i \leq s)$.

1. *We say that $C_p < C_q$ if PE $C_p(i)$ in C_p lies to the left-side of PE $C_q(i)$ in C_q, for $1 \leq i \leq s$.*
2. *We say that $C_p \leq C_q$ if PE $C_p(i)$ in C_p lies to the left-side of, or is identical to, the PE $C_q(i)$ in C_q, for $1 \leq i \leq s$.*

3. *We say that C_p and C_q are* ***independent*** *of each other if $C_p < C_q$ or $C_q < C_p$.*
4. *We say that C_p and C_q are* ***overlap*** *with each other if neither $C_p < C_q$ nor $C_q < C_p$ is satisfied.*

We define that a logical column, say column C_l, as the *left-most column* related to column set A if $C_l \in A$ and $C_l \preceq C_p$ for any $C_p \in A$.

The problem $\mathcal{R}$ is proved to be optimally solved by an algorithm named *Greedy Column Rerouting* (GCR)[9]. In fact, the target array produced by GCR is maximum-size under the constraint that the compensation distance d of columns rerouting scheme is limited to 1, instead of the maximum-size target array for the problem $\mathcal{R}$. All operations in GCR are carried out on the adjacent sets of each fault-free PE u in row R_{r_i}, defined as $\mathcal{A}dj(u)$=$\{v : v \in R_{r_{i+1}}$, v is unused fault-free PE and $|col(u) - col(v)| \leq 1$ $\}$. $Adj(u)$ consists of PEs that PE u can directly connect to in the next selected row. The PEs in $Adj(u)$ are ordered in increasing columns index. The PE with the minimum column index in $Adj(u)$ is called the leftmost connectable PE for u.

2.3 Flexible Rerouting Schemes

As we have discussed before, two types of software rerouting schemes, i.e. bypass and rerouting schemes, are employed to guide reconfiguration algorithms. In previous work, the compensation distance d of column rerouting scheme is limited to 1 to keep low cost of hardware implementation, thus 4-port switches are effective for reconfiguration. Rerouting schemes provides convenience for designing algorithms and guarantees that target arrays can be easily implemented using 4-port switches. However, there are considerable *un-connectable* PEs which are fault-free but cannot be used in forming a logical target array. Fig.2 shows two logical arrays constructed from a 5×5 host array. Fig.2(a) shows an logical array of two logical columns constructed by algorithm GCR under column rerouting scheme with d=1. The target array formed under flexible rerouting scheme contains 3 logical columns as shown in Fig.2(b). There are 7 un-connectable PEs in the first target array, but only 2 un-connectable PEs is generated by adopting flexible rerouting scheme as shown in the second array.

Experimental results show that, under fixed column rerouting scheme with $d = 1$, the proportions of un-connectable PEs are up to 17.1%, 34.75% and 52.45% on 64×64 host array with fault densities are 10%, 20% and 30%, respectively. In order to to improve the *harvest* by reducing the number of un-connectable PEs, we adopt the *flexible column rerouting scheme* which is formed as follows. Assume that row R_{r_i} and $R_{r_{i+1}}$ are the i-th and $(i + 1)$-th selected rows, and set E_i and E_{i+1} contain fault-free PEs in R_{r_i} and $R_{r_{i+1}}$ respectively, then a PE in E_i can connect directly to any PE in E_{i+1} by reconfigure relative switch status.

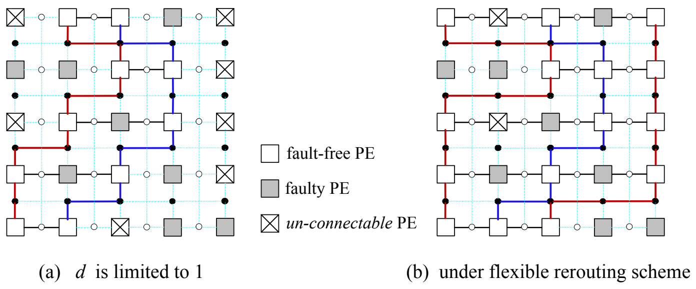

Fig. 2. (a) only 2 logical columns are formed under column rerouting scheme with compensation distance d=1, (b) 3 logical columns can be constructed under flexible rerouting scheme

3 Maximum-Size Target Array under Flexible Rerouting Scheme

In this section, we solve problem $\mathcal{R}$ under flexible rerouting scheme to further increase target array size. In another word, given an $m \times n$ mesh-connected processor array H in which some of its elements are faulty, let $R_{r_1}, R_{r_2}, ..., R_{r_s}$ be the selected rows, find a target array which consists of the maximum number of independent logical columns under the row bypass and flexible column rerouting scheme such that each logical column contains the selected rows. We first present a heuristic algorithm, denoted as FLX, which is capable of solving problem $\mathcal{R}$ in linear time. Then we prove that the target array constructed by FLX is maximum-size target array, and the target arrays constructed in this paper is much larger than that in previous works.

We model the problem $\mathcal{R}$ as follows. Suppose that set S contains all logical columns that can be construct on the selected rows of host array H. Formally, S=$\{C|C$ is a logical column on selected rows and $|C \cap E_i|$=1 for 1$\leq i \leq s\}$. Any column C in S contains exactly one PE from each selected rows. In problem $\mathcal{R}$, we wish to find a maximum-size subset, say A, of independent logical columns.

We shall solve this problem in several steps, in each step we make choice of selecting one column for inclusion into A, and we are left with the subproblem S': finding a maximum-size subset on subproblem S'. We observe that we need only to consider the greedy choice, the left-most logical column, in making each choice. Theorem 1 verifies that the greedy way of making choice leads to optimal solution, the verification is based on Theorem 2 which indicates that the greedy choice is always part of some optimal solution. Let $S_k = \{C_i|C_i \in S$ and $C_k < C_i\}$ be the set of logical columns that lie to the right-side of column C_k. If we make the choice of selecting the left-most column C_1, then S_1 remains as the only subproblem to solve. Since C_1 is in some optimal solution, we can construct an optimal solution to the original problem consists of column C_1 and all logical

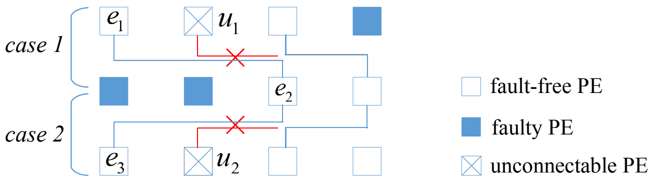

Fig. 3. C_1=<e_1, e_2, e_3> is the left-most column, u_1 and u_2 become un-connectable after column C_1 is constructed

columns in an optimal solution to the subproblem S_1. In another word, $\{C_1\} \cup A'$ is an optimal solution to original problem S if A' is an optimal solution to subproblem S_1.

After constructing the leftmost column, some PEs becomes unconnectable due to overlap of interconnects. Rerouting on the unconnectable PEs leads to backtracking, such as in previous algorithm GCR. In order to improve the efficiency, we adopt the following technique to avoid backtracking. We identify unconnectable PEs so that columns containing unconnectable PEs will not be considered in the next stage selection. To exclude these un-connectable PEs out of sets E_i for $1 \leq i \leq s$, two type of situations are examined as follows, according to the newly constructed column C_k. Let $C_k(i)$ and $C_k(i+1)$ denote two PEs which lie in the ith selected row and the $(i+1)$th selected row of column C_k.

- case 1: $col(C_k(i+1)) > col(C_k(i))$, then PE u is invalid if $row(u) = r_i$, $col(u) > col(C_k(i))$ and $col(u) < col(C_k(i+1))$.
- case 2: $col(C_k(i)) > col(C_k(i+1))$, then PE u is invalid if $row(u) = r_{i+1}$, $col(u) > col(C_k(i+1))$ and $col(u) < col(C_k(i))$.

An example is shown in Fig.3. u_1 is un-connectable according to case 1 and u_2 is un-connectable according to case 2. Thus, after constructing each left-most column, un-connectable PEs must be excluded from candidate PEs set.

We implement the algorithm FLX in a iterative way instead of a recursive manner. The algorithm takes host array H with size of $m \times n$ in which some of its elements are faulty, and the indexes $r_1, r_2, ..., r_s$ that defines the selected rows as input. It returns a maximum-size subset of independent logical columns which form a logical target array T. The algorithm works as follows. The variable k indexes the most resent constructed left-most column,corresponding to the C_k in algorithm FLX. Then the **for** loop initialize E_i as the set of fault-free PEs in row R_{r_i} for $1 \leq i \leq s$. PEs in E_i are sorted initially by increasing column index. In **while** loop, the algorithm construct left-most column C_k one by one and adds C_k into target array T. In each iteration, the algorithm first construct a left-most column C_k, then update candidate set $E_i(1 \leq i \leq s)$ by excluding un-connectable PEs according to C_k. To see which PEs are un-connectable, two type of cases are examined as described above.

Now we analyze the complexity of the proposed greedy algorithm. Initialize candidate set $E_i(1 \leq i \leq s)$ runs in $O(m \cdot n)$. In the **while** loop, each PE is

Algorithm 1: *FLX*$(H, r_1, r_2, ..., r_s\ )$ /* Flexible Column Rerouting Algorithm */

```
Input: host array $H$ with size of $m \times n$; indexes of selected rows: $r_1$,$r_2$,...,$r_s$;
Output: maximum-size target array $T$.
begin
for $i \leftarrow 1$ to $s$ do
    $E_i \leftarrow$ set of fault-free elements in $R_{r_i}$;
$k \leftarrow 1$;        /* index of current logical column */
while $(E_1 \neq \phi)$ do
    /* construct $C_k$ from candidate sets $\cup_{i=1}^{s} E_i$ */
    for $i \leftarrow 1$ to $s$ do
        if $(E_i \neq \phi)$ then
            $e \leftarrow$ the PE with minimum column index in $E_i$;
            $C_k(i) \leftarrow e$;
            mark $e$ as occupied;
        else
            stop the while loop;
    /* exclude unconnectable PEs out of $E_i (1 \leq i \leq s{-}1)$ according to case 1 */
    for $i \leftarrow 1$ to $s - 1$ do
        if $col(C_k(i+1)) - col(C_k(i)) > 1$ then
            for $j \leftarrow col(C_k(i)){+}1$ to $col(C_k(i+1)){-}1$ do
                if $e_{i,j} \in E_i$ then
                    mark $e_{i,j}$ as un-connectable;
    /* exclude unconnectable PEs out of $E_i (2 \leq i \leq s)$ according to case 2 */
    for $i \leftarrow 2$ to $s$ do
        if $col(C_k(i-1)) - col(C_k(i)) > 1$ then
            for $j \leftarrow col(C_k(i)) + 1$ to $col(C_k(i-1)) - 1$ do
                if $e_{i,j} \in E_i$ then
                    mark $e_{i,j}$ as un-connectable;
    $T \leftarrow T \cup \{C_k\}$;
    $k \leftarrow k + 1$;
return $T$;
end
```

examined only once, either to include into a left-most column or excluded out of candidate set E_i, thus the while loop runs in $O(m \cdot n)$. Therefore, the algorithm FLX construct a maximum-size target array under flexible rerouting schemes in $O(m \cdot n)$ time.

Theorem 1. *Algorithm FLX produces maximum-size target array for problem $\mathcal{R}$.*

Proof. Let C_1 be the left-most logical column in S, thus for any $C_p \in S$ we have $C_1 \leq C_p$. Then, based on theorem 2, there always exist an optimal solution, say A, whose left-most column is C_1. Once the greedy choice C_1 is made, the problem S is reduced to find an optimal solution for subproblem $S_1 = \{C_x | C_x \in S$ and $C_1 < C_x\}$ without have to consider any logical columns that overlap with C_1 in S_1. This is because for any $C_p \in S$, we have $C_1 \leq C_p$, and C_1 is the left-most column. Thus, all logical columns that are independent with logical column C_1 must not overlap with C_1, i.e. they certainly belong to S_1. Therefore, we can first construct the left-most column C_1, and then try to find an optimal solution, say A_1, for sub-problem S_1, thus one optimal solution for the original problem S could be $\{C_1\} \cup A_1$. In this way, we could find a maximum-size target array by making greedy choice at each step. In another word, algorithm FLX produces maximum-size target array for problem $\mathcal{R}$.

Theorem 2. *Consider any nonempty subproblem S_k, and let C_m be the left-most logical column on selected rows in S_k. Then C_m is included in some maximum-size subset of independent logical columns of S_k.*

Proof. Let A_k be a maximum-size subset of independent logical columns in S_k, and let C_j be left-most logical column in A_k. If $C_j = C_m$, we are done, since we have shown that C_m is in some maximum-size subset of independent logical columns of S_k. If $C_j \neq C_m$, let the set $A'_k = A_k - \{C_j\} \cup \{C_m\}$ be A_k but substituting C_m for C_j. The logical columns in A'_k are independent, which follows because the columns in A_k are independent, C_j is the left-most column in A_k, and $C_m \leq C_j$. Since $|A'_k| = |A_k|$, we conclude that A'_k is a maximum-size subset of independent logical columns in S_k, and it includes C_m. Therefore, C_m is included in some maximum-size subset of independent logical columns of S_k.

4 Experimental Results and Analysis

In this section, we investigate the efficiency of flexible rerouting scheme by evaluating performance of algorithm FLX in comparison with algorithm GCR. To evaluation metrics, i.e., *harvest* and *degradation*, as formulated in [8-10,13-14], were calculated. The *harvest* indicates how effectively the non-faulty elements are utilized in constructing a target array from a host array with fault elements, whereas the *degradation* measures the degree of potential performance loss due to a smaller target array than the original host array.

$$harvest = \frac{\text{Size of Target Array}}{\text{Total number of Nonfaulty PEs in Host Array}} \times 100\%$$

$$degradation = \frac{\text{Size of Host Array - Size of Target Array}}{\text{Size of Host Array}} \times 100\%$$

For the simplicity, we denote target arrays constructed by algorithms GCR and FLX as *GCRA* and *FLXA*, respectively. The improvement in *harvest* of the FLX over algorithm GCR, is calculated by

$$impr = \frac{harvest \text{ of } FLXA \text{ - } harvest \text{ of } GCRA}{harvest \text{ of } FLXA} \times 100\%$$

Table 1. The performance comparison of the algorithms GCR and FLX for random faults of uniform distribution, averaged over 20 random instances for each case

Host Array		Target array						
Size	Fault density	Array size		*harvest* (%)		*degradation* (%)		*impr* (%)
		GCR	FLX	GCR	FLX	GCR	FLX	
64×64	0.05	64×54	64×55	89.23	89.64	15.23	14.84	0.46
	0.10	64×48	64×49	82.91	84.56	25.39	23.91	2.00
	0.20	64×33	64×38	65.25	73.85	47.81	40.94	13.25
	0.30	64×21	64×30	47.55	66.08	66.72	53.75	39.46
128×128	0.05	128×110	128×111	90.71	91.16	13.83	13.40	0.50
	0.10	128×96	128×98	83.29	85.20	25.04	23.32	2.30
	0.20	128×68	128×77	66.55	75.64	46.76	39.49	13.74
	0.30	128×41	128×61	46.21	67.75	67.66	52.58	46.94
256×256	0.05	256×222	256×224	91.41	92.06	13.16	12.54	0.72
	0.10	256×193	256×198	83.57	86.07	24.79	22.54	2.99
	0.20	256×137	256×157	66.82	76.83	46.54	38.54	14.99
	0.30	256×82	256×123	45.65	68.75	68.05	51.88	50.75

In order to make a fair comparison, we keep the same assumptions as in [8-14]. Faults are only associated with PEs and communication infrastructure is assumed to be fault free. This assumption can be justified since the switches and links use much less hardware resources when compared to the processors and are thus less vulnerable to defects. Both random fault model and clustered fault model were considered[7-10,12]. In a random fault model, the faults of a host array were generated by a uniform random generator. The fault density ρ of the host array is set from 5% to 30% for the experiments. On the other hand, without loss of generality, we model the clustered faults as a subarray that has 80% random faults in the host array. In order to focus on the clustered fault distribution, the fault density out of the subarray is set to 5%, which is far less than that in the subarray. The location of the subarray is generated randomly in the host array. Both algorithms were implemented in C++ language running on a computer with 2.1GHz CPU and 2GB RAM and are compared with each other on the same input instances. For each case, all physical rows are selected for inclusion into target array for experiments.

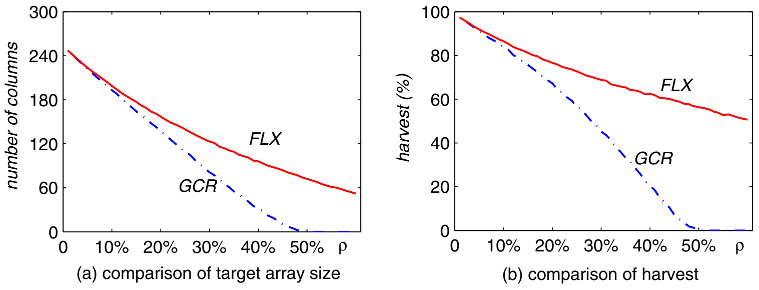

Fig. 4. Comparison of FLX and GCR in terms of target array size and harvest, on host arrays of 256 × 256 with random faults, averaged over 20 instances for each case

Table 1 shows the performance comparison of algorithms FLX and GCR. For the case of constructing target array on 64×64 host array with fault density 10%, the *harvest* of target array produced by GCR is 82.91%, while it is 84.56% by algorithm FLX, thus resulting in the improvement of 2% in *harvest*. When the fault density increases to 30%, FLX improves GCR by 39.46% on *harvest*. It is because, with the increasing fault densities, more PEs become un-connectable in algorithm GCR while they can be used in FLX to form logical columns. It is also worth pointing out that, if the fault density exceeds a certain point, i.e. 50% for host arrays with size of 256 × 256, algorithm GCR fails to form any logical columns as shown in Fig. 4(a). However, algorithm FLX is still capable of constructing target arrays with no less than 70 columns which corresponding to a harvest of 56%, as shown in Fig. 4(b). On the other hand, on host arrays with large fault densities, the improvement in harvest is more significant on large target arrays than on the relatively small ones. For examples, with fault density of 30%, the improvements of FLX over GCR are 39.46%, 46.94% and 50.75% for host arrays with sizes of 64 × 64,128 × 128 and 256 × 256, respectively. It is because, the increasing size increases the difficulty of forming a logical column thus leads to decrease in the number of logical columns and results in increase in the number of un-connectable PEs. This provides more chance for algorithm FLX to improve the target arrays using flexible column rerouting scheme.

Table 2 shows performance gain of algorithms FLX over GCR for the case of the clustered faults on 512×512 host arrays with fault density 5%. Our investigations reveal that, the harvest improvement increased with the increasing number of clustered fault areas. For example, on host array with 24×24 clustered fault areas, the improvement of harvest is 5.15% for the case of 8 clustered fault areas, while it increased to 13.22% and 19.77% on host arrays with 24 and 32 clustered fault areas, respectively. This is because, as the number of clustered fault areas increases, fewer logical columns can be constructed by algorithm GCR, thus leads to more un-connectable PEs which provide more chance for optimization by FLX. In addition, the improvement for harvest decreases is more significant on relatively large clustered fault area. This is because logical columns constructed by GCR cannot go through clustered fault areas, thus large scale clustered fault

Table 2. The performance comparison of the algorithms GCR and FLX for clustered faults of uniform distribution on 512×512 host array, averaged over 20 random instances for each case

Host Array		Target array						
Cluster Distribution		Array size		*harvest* (%)		*degradation* (%)		*impr* (%)
size	number	GCR	FLX	GCR	FLX	GCR	FLX	
16×16	8	512×419	512×431	86.12	88.58	18.18	15.85	2.87
	16	512×400	512×419	82.25	86.09	21.87	18.21	4.72
	24	512×381	512×409	78.42	84.06	25.50	20.15	7.23
	32	512×369	512×401	75.81	82.41	27.98	21.71	8.73
24×24	8	512×395	512×416	81.29	85.44	22.77	18.83	5.15
	16	512×363	512×396	74.69	81.49	29.04	22.59	9.17
	24	512×330	512×374	67.86	76.82	35.54	27.02	13.32
	32	512×294	512×351	60.40	72.13	42.62	31.47	19.77
32×32	8	512×364	512×392	74.83	80.59	28.92	23.44	7.84
	16	512×318	512×364	65.40	74.79	37.87	28.95	14.78
	24	512×251	512×322	51.54	66.27	51.04	37.04	28.80
	32	512×226	512×315	46.48	64.69	55.84	38.54	40.37

areas decreases the area of PEs that can be used for constructing target arrays by algorithm GCR. On the other hand, large clustered fault areas increase the fault density of the host array, thus provide more chance for optimization by algorithm FLX.

5 Conclusions

Many previous works discuss the reconfiguration problems under column rerouting scheme with fixed compensation distance, resulting in great lose of harvest. This paper investigate the efficiency of flexible rerouting scheme by developing an efficient algorithm to reconfigure an multiprocessor array with faults. The proposed algorithm FLX is able to reduce the number of un-connectable PEs caused by the interconnection overlaps. It is capable of producing maximum-size target arrays, resulting in a significant improvement in comparison to the algorithm GCR. Experimental results show that the size of target array can be significantly increased. The algorithm FLX is scalable with the array size, as it is more efficient on large host arrays than on relatively small ones. In addition, its harvest tends to decrease slowly with the increasing fault density, that is superior to algorithm GCR whose harvest value decreases rapidly.

Acknowledgments. This work was supported by the National Science Foundation of China under Grant No. 61070136 and No. 61173032, and the Doctoral Fund of Ministry of Education of China under Grant No. 20100032110041.

References

1. Murali, S., De Micheli, G.: SUNMAP: A tool for automatic topology selection and generation for NoCs. In: Proc. of the 41st Design Automation Conference (DAC 2004), pp. 914–919. ACM Press, San Diego (2004)
2. Bertozzi, D., Jalabert, A., Murali, S., Tamahankar, R., Stergiou, S., Benini, L., De Micheli, G.: NoC synthesis flow for customized domain specific multiprocessor systems-on-chip. IEEE Transactions on Parallel and Distributed System 16(2), 113–129 (2005)
3. Modarressi, M., Sarbazi-Azad, H.: Power-aware mapping for reconfigurable NoC architectures. In: Proc. of the 25th International Conference on Computer Design, pp. 417–422. IEEE Press, California (2007)
4. Modarressi, M., Tavakkol, A., Sarbazi-Azad, H.: Application-Aware Topology Reconfiguration for On-Chip Networks. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 19(11), 2010–2022 (2011)
5. Chen, Y.Y., Upadhyaya, S.J., Cheng, C.H.: A comprehensive reconfiguration scheme for fault-tolerant VLSI/WSI array processors. IEEE Transactions Computers 46(12), 1363–1371 (1997)
6. Horita, T., Takanami, I.: Fault-tolerant processor arrays based on the 1.5-track switches with flexible spare distributions. IEEE Transactions on Computers 49(6), 542–552 (2000)
7. Kuo, S.Y., Chen, I.Y.: Efficient reconfiguration algorithms for degradable VLSI/WSI arrays. IEEE Transactions Computer-Aided Design 11(10), 1289–1300 (1992)
8. Low, C.P., Leong, H.W.: On the reconfiguration of degradable VLSI/WSI arrays. IEEE Transactions Computer-Aided Design of Integrated Circuits and Systems 16(10), 1213–1221 (1997)
9. Low, C.P.: An efficient reconfiguration algorithm for degradable VLSI/WSI arrays. IEEE Transactions on Computers 49(6), 553–559 (2000)
10. Wu, J., Srikanthan, T.: Reconfiguration Algorithms for Power Efficient VLSI Subarrays with 4-port Switches. IEEE Transactions on Computers 55(3), 243–253 (2006)
11. Fukushi, M., Fukushima, Y., Horiguchi, S.: A genetic approach for the reconfiguration of degradable processor arrays. In: Proc. of 20th IEEE International Symposium on Defect Fault Tolerance VLSI System, pp. 63–71. IEEE Press, Ohio (2005)
12. Wu, J., Srikanthan, T., Han, X.: Preprocessing and Partial Rerouting Techniques for Accelerating Reconfiguration of Degradable VLSI Arrays. IEEE Transactions on Very Large Scale Intergration (VLSI) Systems 18(2), 315–319 (2010)
13. Jiang, G., Wu, J., Sun, J.: Non-Backtracking Reconfiguration Algorithm for Three-dimensional VLSI Arrays. In: Proc. of 2012 IEEE 18th International Conference on Parallel and Distributed Systems, pp. 362–367. IEEE Press, Singapore (2012)
14. Jiang, G., Wu, J., Sun, J.: Efficient Reconfiguration Algorithm for Three-dimensional VLSI Arrays. In: Proc. of 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum, pp. 254–258. IEEE Press, Shanghai (2012)