Some New Algorithms for Reconfiguring VLSI/WSI Arrays

Theodora Varvarigou Vwani P. Roychowdhury Thomas Kailath *

Abstract

This paper deals with the issue of reconfiguring processor arrays in the presence of faulty processors and fixed hardware resources. The models discussed in this paper consist of a set of identical Processing Elements (PEs) embedded in a flexible interconnection structure that is configured in the form of a *rectangular grid.* Furthermore in order to incorporate fault tolerance, a given array has a pre-determined distribution of spare PEs. The general issue in reconfiguration is to replace faulty non-spare PEs by healthy spare ones, subject to given hardware constraints. In this paper, we present some new algorithms for reconfiguring $N \times (N+1)$ arrays (where the spare PEs are configured in the form of a spare row) into $N \times N$ arrays when N of the PEs are faulty. The algorithms developed here are simple and perform better than other reconfiguration algorithms presented in the literature.

1 Introduction

This paper deals with the issue of developing efficient algorithms for reconfiguring processor arrays in the presence of faulty processors. Such studies are relevant in the case of Wafer Scale Integration (WSI) technology where for example, a large number of processors, configured in the form of a grid, can be put on a single wafer. Due to yield problems, some of the processors are invariably going to be faulty. In such a case, instead of treating the whole wafer as defective, one can work around the faulty processors and reconfigure the rest in the form of a grid. Thus, reconfiguration methodologies can be viewed as possible tools to increase the effective yield of the processing technology.

The motivation behind this paper is to consider reconfiguration strategies when the hardware resources are fixed. The general model to be discussed here (see *e.g.* , [4, 5, 6, 7, 2, 3]) consists of a set of identical processors embedded in a flexible interconnection structure that is configured in the form of a rectangular grid. Each grid line in the mesh has a fixed number of data paths that can be routed along it (*i.e.* , the model has fixed channel width); switches can be placed at every grid point and at every location where a processor is connected to the grid. Furthermore, often the processors are divided into a set of non-spare PEs (say an $m \times n$ array) and a set of spare PEs that are distributed in a pre-determined fashion. The general question asked in such models is: if some of the non-spare PEs are faulty, then can the array be reconfigured to replace the faulty PEs with some of the spare PEs? Obviously, the power of the reconfigurable architecture is determined by the available hardware resources such as the channel width, the complexity of the switches and their distribution in the array. Although one would like to put as much hardware as possible, often it is expensive to do so.

*Authors are with the Information Systems Laboratory, Stanford University, CA, 94305, and their work was supported in part by the Department of the Navy, Office of Naval Research under Contract N00014-86-K-0726, the SDIO/IST, managed by the Army Research Office under Contract DAAL03-87-K-0033, and the U. S. Army Research Office under Contract DAAL03-86-K-0045.

Recently, several researchers [6, 4, 1, 8] have given heuristic algorithms for reconfiguration for models that differ only slightly from the general model discussed above. The particular model that we discuss in this paper (see also [6]) consists of an $N \times (N+1)$ array, where N spare PEs are configured in the form of a spare column, and are located along one boundary of the original $N \times N$ array. The goal of the reconfiguration algorithm is to obtain an $N \times N$ array of healthy processors given that any N of the PEs are faulty; moreover, the reconfigured array must satisfy certain neighborhood constraints. Let the ***physical array*** be the array given to us, and let the ***logical array*** be the reconfigured array. Then the neighborhood constraint requires that in the logical array the neighbors of a healthy processor be restricted to lie in a fixed neighborhood around the healthy processor; the neighborhood is shown in Fig. 1.

The same model is considered in [6]; however, the reconfiguration algorithms presented there are very fragile. The algorithms presented in the above mentioned paper fail to reconfigure even when the faulty PE distributions are very simple and following are only two simple instances (one can generate several other instances) where they fail: (1) If the bottom row or the top row does not have any faulty processors, (2) If there are two columns each with a stack of faulty processors, even of size two, as shown in the Fig. 7.

The algorithms that we develop can reconfigure em all instances of arrays that the algorithms reported in [6] can. Moreover, we can also reconfigure arrays with several other faulty distributions, ***without increasing the neighborhood constraint.*** In particular, we give reconfiguration algorithms for the following cases (1) If every column has one faulty PE (2) If one column has two faulty PEs and the rest of the columns have one faulty PE each or no faulty PEs, (3) If there are two stacks of faulty PEs, each of length $N/2$, and (4) If the faulty PEs can be partitioned into blocks such that each block can be considered as one of the above special cases.

The reconfiguration algorithms are based on one simple algorithm that shows how to reconfigure an $N \times (N+1)$ array into an $(N+1) \times N$ array and vice versa. Moreover, our algorithms generate very regular patterns, and can be implemented in a distributed fashion instead of being processed by one host processor.

We should also mention here that in [1], a model similar to ours is considered; however, they do not put the strict neighborhood restriction. We can show that the algorithms in [1] require larger neighborhood for arrays that can be reconfigured with smaller neighborhoods with our algorithm; the details will be provided in the full paper.

2 Reconfiguration Algorithms

The reconfiguration algorithms presented in this paper are based on an algorithm that reconfigures an $N \times (N+1)$ array into an $(N+1) \times N$ array and vice versa. Based on this algorithm we are going to describe ways to reconfigure (subject to the neighborhood constraint):

1. $N \times (N+1)$ arrays that have only one faulty processor in each column into $N \times N$ ones.
2. We shall just give an outline of how to reconfigure:
   - $N \times (N+1)$ that have only one faulty processor in each column except from one that has two of them.
   - $N \times (N+1)$ arrays that have two stacks of faulty processors of size N/2 each.
   - Arrays that can be segmented into arrays of any of the kinds above.

Before proceeding to the analysis of the reconfiguration procedure in any of the above cases let us define the following.

Definition 1 *For each entry (i,j) of the given array , we define as neighborhood (or legal neighborhood) of (i, j) the set of processors :* $N_{i,j} = \{(k,l) | \ (k,l) = (i,j) + (m,n) \ , \ (m,n) \in T\}$ *where* T *is* *a set of Tuples.*

In this section we are going to consider that $N_{i,j}$ is the one shown in Fig.1 . In other words : $N_{i,j} = \{(k,l) | (i-k)^2 + (j-l)^2 \leq 5\}$.

Definition 2 *Physical array (column,row) is the given $N \times (N+1)$ array (column, row).*

2.1 Reconfiguration of $N \times (N+1)$ arrays into $(N+1) \times N$ ones

We are going to describe a procedure to reconfigure a physical $N \times (N+1)$ array into a $(N+1) \times N$ logical array maintaining the neighborhood constrains described above :
Reconfiguration Algorithm 1.
Let's consider entry (i,j) of the physical array , where $1 \leq i \leq N$ and $1 \leq j \leq N+1$. Then we define the (i',j') entry of the logical array to be:

$$(i', j') = \begin{cases} (i, j) & \text{if } i' = j' \\ (i-1, j) & \text{if } i' - j' \geq 2 \\ (i, j+1) & \text{if } j'x - i' \geq 1 \\ (i-1, j+1) & \text{if } i' = j' + 1 \end{cases} \quad \square$$

The construction is illustrated in Fig.2 .

The procedure for mapping an $(N+1) \times N$ physical array into an $N \times (N+1)$ logical one is exactly similar: one just uses the above procedure by interchanging the rows with the columns and vice versa.

Theorem 1 *The construction of a $(N+1) \times N$ $(N \times (N+1))$logical array out of a $N \times (N+1)((N+1) \times N)$ physical array maintains the neighborhood constrains mentioned above.*

Proof: Proof comes out directly from Fig. 1. It is easy to check that for each entry $(i'j')$, entries $(i'+1,j')$, $(i'-1,j')$, $(i',j'-1)$, $(i',j'+1)$ are within the legal neighborhood. $\square$

2.2 Reconfiguration of $N \times (N+1)$ arrays with one faulty processor in each physical column

Definition 3 *Logical array (column row) is the $N \times (N+1)$ array(row, column) which we derive after the first reconfiguration procedure described in the previous section.*

Definition 4 *S_l is the subset of the processors (i,j) of the physical array for which $i - j \geq 0$ is true,(see Fig.2).*

Definition 5 *S_u is the subset of the processors (i,j) of the physical array for which $i - j < 0$ is true (see Fig.2).*

Definition 6 *A i-fault physical(logical) column (row) $i = 0, \ldots, 3$ is a physical(logical) column(row) that has i faulty processors.*

Definition 7 *For each 2-fault logical column i the corresponding 0-fault logical column is defined as the 0-fault logical column j, j>i for which the following is true: all logical columns k, $i < k < j$ are 1-fault logical columns.*

Lemma 1 *The S_l processors of the i^{th} logical column are part of the i^{th} physical column whereas the S_u ones are part of the $(i+1)^{th}$ physical column.*

Proof: Immediately from the Reconfiguration Algorithm 1. $\square$

Lemma 2 *Each logical column has at most two faulty processors. If a logical column has two faulties then one of them is in S_l and one in S_u.*

Proof: The first part follows immediately from the first reconfiguration algorithm by noting that each logical column i occupies parts of only two physical columns, the part of the physical column i which is in S_l and the part of the physical column (i+1) which is in S_u(see previous lemma). So it is obvious that it cannot have both faulty processors in $S_l(S_u)$ because there is only one faulty processor in physical column i (i+1). □

Corollary 1 *The number of 2-fault logical columns equals the number of the 0-fault logical columns.*

Lemma 3 *Suppose that the only faulty processor of logical column i is in S_l . Then one of the following is true :*
- *The (i+1) logical column has only one faulty processor which is in S_l as well.*
- *The (i+1) logical column has two faulties (one in S_l and one in S_l).*

Proof: Since the only faulty processor of the logical column i is in S_l , in the i^{th} physical column, the faulty processor of the $(i+1)^{th}$ physical column should be in S_l. So the logical column (i+1) has one faulty processor in S_l . So if logical column (i+1) is 1-fault logical column 1 is true , otherwise (by lemma2) 2 is true. □

Lemma 4 *Suppose that logical column i has one faulty processor in S_u . Then one of the following is true:*
- *The logical column (i+1) has only one faulty processor in the S_u part of physical column (i+2).*
- *The logical column (i+1) is 0-fault logical column.*

Proof: Since there is a fault in S_u in physical column (i+1) there is no faulty processor in S_l in the same physical column. So , if physical column (i+2) has a faulty processor in S_u then 1 is true otherwise 2 is true. □

Lemma 5 *Suppose that logical column i is fault free. Then one of the following is true:*
- *The (i+1) logical column has only one faulty processor in S_l*
- *The (i+1) logical column is a 2-fault logical column.*

Proof: Since i is a 0-fault logical column , the S_u part of (i+1) physical column is fault free. So there is a fault in S_l part of (i+1) logical column . So, is (i+2) physical column has no fault in S_u 2 is true , otherwise 1 is true. □

Lemma 6 *Each 2-fault logical column i has a corresponding 0-fault logical column j . Moreover all the 1-fault logical columns k with $i < k < j$ have their faulty processors in S_u*

Proof: Since i is a 2-fault logical column then (by lemma 1)it has a faulty processor in S_u. Then by lemma 4 the (i+1) logical column is: a. Fault free , in which case we are done. b. Has one faulty in S_u in which case theorem 5 applies again. So, corollary 1 and lemma 4 guarantee that there is a fault free logical column corresponding to each 2 fault logical column i , and more than that, lemma 5 guarantees that the faulty processors in columns (i+1)..(j-1) will be in S_u. □

We now present the reconfiguration procedure:

Reconfiguration Algorithm 2:

1. Construct an $(N + 1) \times N$ logical array out of the $N \times (N + 1)$ physical (given) one.

2. Identify the 0 1 2-fault logical columns.

3. For the logical columns j' with one faulty processor in the entry $(k'j')$ do the following renaming :

$$(i', j') \longrightarrow \begin{cases} (i', j') & \text{if } i' < k' \\ (i' - 1, j') & \text{if } i' > k' \end{cases}$$

4. For the 2-fault logical columns j' with faults in entrees : $(k'j')$ and $(l'j')$, $(l > k)$ do the following renaming :

$$(i', j') \longrightarrow \begin{cases} (i', j') & \text{if } i' < k' \\ (i' - 1, j') & \text{if } k' < i' < l' \\ (i' - 2, j' & \text{if } l' < i' \end{cases}$$

5. For the 2-fault logical columns j' , if l' is the corresponding 0-fault column, do the following renaming:

$$(N, k') \longrightarrow \begin{cases} (N, k' + 1) & \text{if } j' < k' < l' \\ (N + 1, l') & \text{if } k' = l' - 1 \end{cases}$$

Theorem 2 *The algorithm presented above reconfigures the* $N \times (N + 1)$ *physical array into a* $N \times N$*logical array maintaining the neighborhood constrains.*

Proof:

• Neighborhood constrains are maintained along the rows of the logical array, (see Fig.4).

• Neighbourhood constrains are maintained along the logical columns, (see Fig.4).

Note that by lemma 6 , there is no fault in S_l between the 2-fault column and the fault free column and there is no more than one borrowing process going on, along physical row N (see Fig.4). It is also easy to check that each column of the resulting logical arrays has N processors and that there are N columns overall. □

Fig.3 and Fig.4, give an example of the faulty pattern and the reconfigured array.

2.3 Algorithms for Reconfiguring Arrays with Other distributions of Faulty Processors

Due to space limitations, we shall just outline the algorithms here; the underlying ideas are same as before.

1. The next faulty distribution we can deal with is the one that has one faulty processor in each column except for one that has two of them. The reconfiguration algorithm is quite similar to the second reconfiguration algorithm of the previous section.The main difference is that now there is a 2-fault or 3-fault logical column with no corresponding 0-fault logical column .The reconfiguration of such arrays cannot be achieved by a borrowing process along the N^{th} physical column as we did in the second reconfiguration algorithm. So there is a borrowing process needed along the rows of S_u. One can easily prove that there is only one such borrowing process required and that the latter does not violate the neighborhood constrains. An example of such a reconfiguration is shown if Fig.5 and Fig.6. The two borrowing processes in the bottom of the fig are regular borrowing processes similar to those in the second reconfiguration algorithm
   The borrowing process on the top reconfigures the logical column that has no corresponding 0-fault logical column.

2. Another interesting faulty distribution that can be reconfigured using similar reconfiguration methods is the one which has two stacks of faulty processors of size N/2 each . The reconfiguration procedure is illustrated in Fig.7 and Fig.8 . The basic step is the segmentation of the faulty pattern into subarrays A $(N/2 \times (N/2 - 1))$ and B $(N/2 \times (N/2 + 1))$. A can be

reconfigured into an $((N/2-1) \times N/2)$ one by the first reconfiguration algorithm whereas B can be reconfigured into an $(N/2 \times N/2)$ one by the second reconfiguration algorithm. It can be easily proved that we can always segment this faulty pattern into subarrays A and B and that the overall reconfiguration does not violate the neighborhood constrains. Note that the faulty processors in subarray B need not be lined up in a stuck. As shown in Fig.9 and Fig.10 the array is reconfigurable if subarray B can be reconfigured with any of the above reconfiguration algorithms .

3. The ideas in above comments can be used for the reconfiguration of any array that can be segmented into subarrays that can be reconfigured through the reconfiguration algorithms described above. Fig.11 and Fig.12 show the way of handling the problem of multiple heaps. The idea is to segment the whole faulty pattern into 2-heap subarrays each of which can be reconfigured by the previous algorithms.

3 Concluding Remarks

In this paper we have considered a model and provided reconfiguration algorithms that perform better than other algorithms proposed in the literature for the same model. The question of designing more efficient algorithms that would work for any array and any neighborhood constraint remains unresolved.

References

[1] Mengly Chean and J. A. B. Fortes. A FUSS Approach to Hardware Reconfiguration for Fault-Tolerant Processor Arrays. Preprint, Purdue University, West Lafayette, IN 47907, also presented at HFTM, June 1989, University of Illinois, Urbana, IL, USA., June 1989.

[2] J. W. Greene and A. El Gamal. Configuration of VLSI arrays in the presence of defects. *JACM*, 31, No. 4:694–717, October 1984.

[3] T. Leighton and C. E. Leiserson. Wafer-Scale Integration of Systolic arrays. *IEEE Trans. on Computers*, C-34, No. 5:448–461, May 1985.

[4] F. Lombardi, M. G. Sami, and R. Stefanelli. Reconfiguration of VLSI arrays by covering. to appear in IEEE Trans. on CAD, 1989.

[5] W. R. Moore. A review of Fault-tolerant Techniques for the enhancement of Integrated Circuit yield. *Proc. of the IEEE*, pages 684–698, May 1986.

[6] M. Sami and R. Stefanelli. Reconfigurable architectures for VLSI processing arrays. *Proc. of the IEEE*, pages 712–722, May 1986.

[7] S. K. Tewksbury. *Wafer-Level Integrated Systems: Implementation Issues.* Kluwer Academic Publishers, 1989.

[8] H. Y. Youn and A. D. Singh. Bounded Channel Width Restructuring Algorithm for VLSI/WSI Arrays With Channel Faults. Submitted to IEEE Trans. On Computers, also presented at HFTM, June 1989, University of Illinois, Urbana, IL, USA., June 1989.

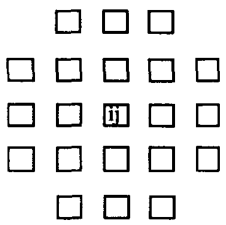

Fig.1: Neighbourhood.

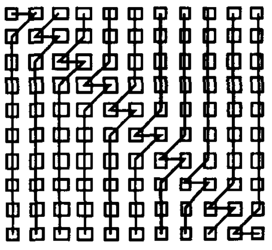

Fig.2: Reconfiguration Algorithm 1.

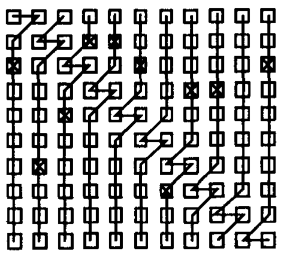

Fig.3: Reconfiguration Algorithm 2: faulty pattern.

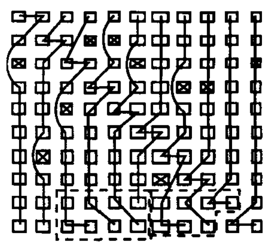

Fig.4: Reconfiguration of faulty pattern in Fig.3.

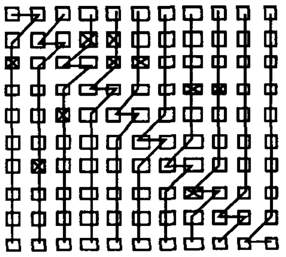

Fig.5: A faulty pattern corresponding to sec1.3.1.

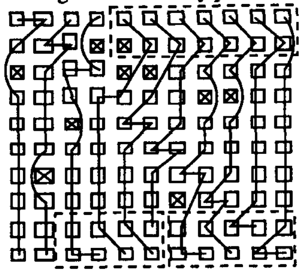

Fig.6: Reconfiguration of the faulty pattern in Fig.5.

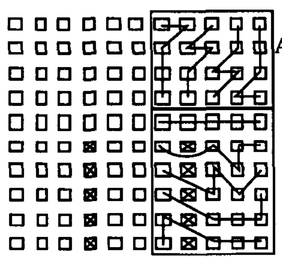

Fig.7: A faulty pattern corresponding to sec 1.3.2.

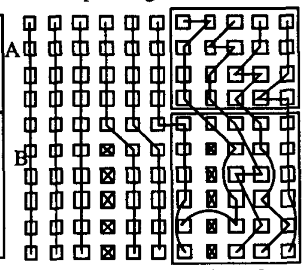

Fig.8: Reconfiguration of the faulty pattern in Fig.7.

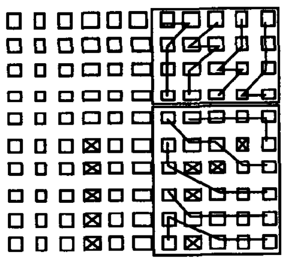

Fig.9: The 2-stack case ,no faulty ones lined up.

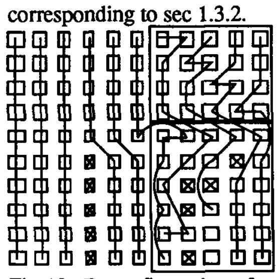

Fig.10: Reconfiguration of faulty pattern in Fig.9.

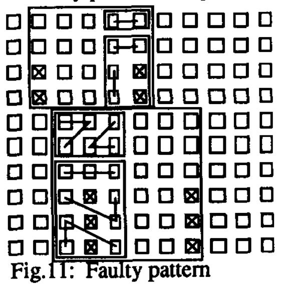

Fig.11: Faulty pattern with multiple stacks.

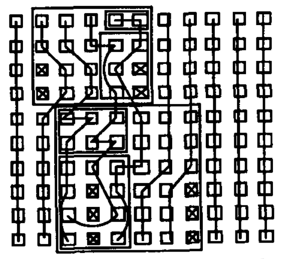

Fig.12: Reconfiguration of faulty pattern in Fig.11.