Efficient Bi-Level Reconfiguration Algorithms for Fault Tolerant Arrays

R. Libeskind-Hadas*, N. Shrivastava**, R. G. Melhem**, and C. L. Liu*

* Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801

** Department of Computer Science
University of Pittsburgh
Pittsburgh, PA 15260

Abstract

In this paper we consider the problem of reconfiguring processor arrays subject to computational loads that alternate between two modes. A *strict* mode is characterized by a heavy computational load and severe constraints on response time while a *relaxed mode* is characterized by a relatively light computational load and relaxed constraints on response time. In the strict mode, reconfiguration is performed by a distributed local algorithm in order to achieve fast recovery from faults. In the relaxed mode, a global reconfiguration algorithm is used to restore the system to a state that maximizes the probability that future faults occuring in subsequent strict modes will be repairable.

Several new results are given for this problem. Efficient reconfiguration algorithms are described for a number of general classes of architectures. These general algorithms obviate the need for architecture-specific algorithms for architectures in these classes. We show that it is unlikely that similar algorithms can be obtained for related classes of architectures since the reconfiguration problem for these classes is NP-complete. Finally, a general approximation algorithm is described that can be used for any architecture. Experimental results are given, suggesting that this algorithm is very effective.

1 Introduction

Advances in VLSI and WSI technologies allow increasingly larger processor arrays to be fabricated on a single chip or wafer. As the number of processors in an array increases, the problem of reconfiguring the array to replace faults occurring at run-time becomes increasingly important. One common way of providing fault tolerance in processor arrays is to augment the array with a set of spare processors that can replace primary processors that become faulty. This approach has been proposed for a number of architectures [9, 10, 12] and a variety of reconfiguration algorithms for these reconfigurable systems have been studied.

It has been observed that in many applications, systems are subject to computational loads that alternate between a *strict* mode in which the computational load is heavy

* The first and fourth authors were partially supported by the National Science Foundation under grant MIP 89-06932.

** The second and third authors were partially supported by the National Science Foundation under grant MIP 89-11303.

and severe constraints are imposed on response time, and a *relaxed* mode in which the computational load is light and the constraints on response time are relaxed substantially [5, 6]. In order to achieve fast response time, it is desirable that reconfiguration in the strict mode be performed in a distributed fashion so as not to incur the overhead of communication with a central host processor. Moreover, in order to minimize interruption in service in the strict mode, it is important that the replacement of a faulty processor causes only a minimal number of changes to the existing system interconnections. Therefore, during strict mode reconfiguration, a processor uses only very local knowledge about its immediate neighbors. These objectives are met by using a distributed local algorithm in the strict mode in which a faulty processor finds a replacement by selecting an available spare processor to which it is directly connected [6].

Although local reconfiguration allows fast replacement of faulty processors, repeated application of the local reconfiguration algorithm may quickly degrade the reliability of the system since spare elements are often not used in the most effective way. Consequently, in the relaxed mode we wish to use a global reconfiguration algorithm to restore the system to a more reliable state. Specifically, the goal of reconfiguration in the relaxed mode is to assign faulty elements to spare elements maximizing the probability that any processor becoming faulty in the next strict mode will be repairable by the local reconfiguration algorithm. Such an assignment is called an *optimal assignment*. This approach of using both local and global reconfiguration algorithms, depending on the state of the system, is called *bi-level reconfiguration*. It was shown in [5, 6] that bi-level reconfiguration can substantially improve the expected lifetime of a system and this approach was illustrated for the case of an augmented hypercube architecture. Subsequently, Chen *et al.* [2] gave a linear time reconfiguration algorithm for this particular architecture.

In this paper we investigate the problem of finding optimal assignments for several broad classes of architectures. We give efficient algorithms for finding optimal assignments in several important classes of architectures, including a number of well-known architectures that have been proposed in the literature. We show that the problem of finding optimal assignments in several other classes of architectures is NP-complete. Moreover, we give an efficient approximation algorithm that can be used for any architecture. Experimental results indicate that this algorithm performs extremely well.

2 Bi-level Reconfiguration

In this section we describe efficient algorithms for finding optimal assignments in several classes of architectures. These classes contain a number of existing and proposed processor array designs. Consequently, these general algorithms obviate the need for architecture-specific algorithms for architectures in these classes. In addition, we show that the optimal assignment problem is NP-complete for several related classes. We begin by introducing some notation and definitions that will simplify our discussion.

A processor array can be represented by a bipartite graph $G = (X \cup Y, E)$ where X is the set of vertices representing primary processors, Y is the set of vertices representing non-faulty spare processors, and E is the set of edges representing connections between primary processors and non-faulty spare processors. Thus, there is an edge $\{x, y\} \in E$ iff the primary processor corresponding to $x \in X$ is connected to the spare processor corresponding to $y \in Y$. We assume that a faulty processor can only be replaced by a spare processor if the two processors are connected, that is, there is a physical link between them. The vertices in X are denoted *primary vertices* and the vertices in Y are denoted *spare vertices.* We let $F \subseteq X$ represent the set of faulty primary processors and let $N = X - F$ represent the set of non-faulty primary processors. A matching, M, of F into Y represents an assignment of faulty primary processors to non-faulty spare processors. A vertex $x \in N$ is said to be *repairable* with respect to M if it is adjacent to an unmatched vertex in Y. Thus, a repairable vertex corresponds to a non-faulty primary processor that can be replaced using the local reconfiguration algorithm. A vertex $x \in N$ that is adjacent only to matched vertices in Y is said to be *unrepairable* since the corresponding primary processor cannot be replaced by the local reconfiguration algorithm. Our objective is to find a matching for F that maximizes the number of repairable vertices and thus maximizes the probability that a fault occurring in the next strict phase will be replaced by the local reconfiguration algorithm. Such a matching corresponds to an optimal assignment and is therefore called an *optimal matching.* A special case of an optimal matching is a matching in which every vertex in N is repairable. Such a matching is called a *safe matching.*

Theorem 1 *The problem of finding an optimal matching in a bipartite graph is NP-complete.*

The proof of this theorem is by a reduction from the MAX 2-SAT problem [3]. The details of the proof can be found in [4]. Although Theorem 1 tells us that the optimal matching problem is NP-complete in general, some architectures have properties that allow us to find efficient algorithms for optimal matching.

In many processor arrays the degree of each processor, that is, the number of processors to which a single processor can be directly connected, is bounded by a small constant. We show that for certain bounds on processor degrees, optimal matchings can be found by efficient algorithms. Let $D(X)$ represent the maximum degree among all vertices in X, that is, $D(X)$ is the maximum number of spare processors that are connected to a primary processor. Similarly, let $D(Y)$ represent the maximum degree among all vertices in Y, that is, the maximum number of primary processors that are connected to a spare processor.

2.1 Algorithm 1

Theorem 1 can be strengthened to show that the problem of finding an optimal matching in a bipartite graph is NP-complete even when $D(X) \leq 2$. However, in this subsection we describe an $O(n^2)$ algorithm for finding safe matchings when $D(X) \leq 2$, where n is the total number of processors in the array. Clearly, for some patterns of faults there exists an optimal matching but no safe matching. In such cases, our algorithm will fail to find a matching. However, experimental results suggest that when the number of faulty processors is not too large, most optimal matchings are in fact safe matchings and thus our algorithm can be used in these cases. For example, in a simulation on randomly generated arrays with 32 primary processors, 16 spare processors, and $D(X) = 2$, more than 86% of the optimal assignments found[1] were safe assignments when the fraction of faulty processors was not higher than 10%. We note that a number of well-known architectures have the property that $D(X) \leq 2$. For example, the fault tolerant binary tree architecture proposed by Raghavendra *et al.* [8], the interstitial redundancy scheme used in the Hughes 3-D Computer [12], and the 2-D augmented mesh proposed by Melhem [5]

[1]Optimal assignments were obtained by exhaustive search.

all have this property. The reconfiguration algorithm is given in Figure 1. The algorithm first transforms the problem into an instance of the 2-Satisfiability Problem (2-SAT) and then solves 2-SAT using a linear-time algorithm [7]. The proof of correctness and analysis of time complexity can be found in [4]. Finally, we have obtained the following result, suggesting that this algorithm cannot be extended for larger values of $D(X)$.

Theorem 2 *The problem of finding a safe matching in a bipartite graph is NP-complete when* $D(X) \leq 3$.

2.2 Algorithm 2

In this subsection we describe a linear time algorithm for finding optimal matchings when $D(Y) \leq 2$. This result implies that optimal assignments can be found efficiently in a number of well-known architectures, such as the fault tolerant hypercube proposed by Banerjee [1] and a number of interstitial array schemes [10]. The correctness of our algorithm and the analysis of time complexity are based on several technical lemmas which are omitted and can be found in [4]. Here we give a brief overview of the algorithm. The algorithm first determines the connected components in the graph[2]. The connected components share no edges, and can therefore be considered independently. If a connected component contains at least as many spare vertices as primary vertices (both faulty and non-faulty) then we can show that there exists a complete matching of the primary vertices into the spare vertices. Thus, each faulty vertex in the component is matched to a spare vertex and each non-faulty vertex is assigned to a unique unmatched spare vertex. Consequently, the matching found in this component is a safe matching and thus is part of an optimal matching for the graph. Moreover, it is not difficult to show that because $D(Y) \leq 2$, this matching can be obtained in linear time by constructing a spanning tree[3] of the component and repeatedly matching primary vertices of degree 1 to spare vertices. If the connected component contains fewer spare vertices than primary vertices then we can show that the component must be a tree. An arbitrary primary vertex in the tree is assigned to be the root. The tree is then scanned in a bottom-up fashion. When a faulty primary vertex is encountered it can be replaced by either its parent spare vertex or one of its children spare vertices. The rules for determining which of these spares to select are given in the full description of the algorithm in Figure 2. Finally, we have the following result, suggesting that similar general algorithms are unlikely for larger values of $D(Y)$.

Theorem 3 *The problem of finding an optimal matching in a bipartite graph is NP-complete when* $D(Y) \leq 3$.

2.3 Algorithm 3

Although the optimal matching problem is NP-complete for $D(X) \leq 2$ and $D(Y) \leq 3$, we now describe an approximation algorithm that can be applied to any architecture and is guaranteed to be at least $\frac{1}{D(X)}$-optimal. In other words, if the number of repairable vertices in an optimal solution is k then the approximation algorithm finds a solution in which

[2]A *connected component* is a maximal set of vertices such that there is a path in the graph between any two vertices in the set. The connected components of a graph can be found in linear time using breadth-first search.

[3]A *spanning tree* of a graph is a connected subgraph containing all the vertices of the graph and the minimum number of edges. A spanning tree can be found in linear time by using breadth-first search.

the number of repairable vertices is at least $\frac{1}{D(X)} \cdot k$. Experimental results, summarized in the next section, suggest that in practice this algorithm performs substantially better than indicated by this theoretical lower bound.

The approximation algorithm transforms a given instance of the optimal matching problem into a weighted matching problem as follows. From the given bipartite graph $G = (X \cup Y, E)$ and set $F \subseteq X$, we construct a weighted bipartite graph $G' = (F \cup Y, E')$ where $E' \subseteq E$ is the set of edges in E that are incident to vertices in F. For each edge $e = \{x, y\} \in E$ with $x \in F$, let w_e denote the number of vertices in N that are adjacent to y. Edge $e = \{x, y\} \in E'$ is assigned weight w_e. An example of this construction is illustrated in Figure 3.

Theorem 4 *Let M be a minimum weight complete matching in G' and let k be the number of repairable vertices in an optimal matching for G. If matching M is used in G, at least $\frac{1}{D(X)} \cdot k$ vertices are repairable.*

The proof of this result can be found in [4]. The complexity of the approximation algorithm is that of finding a minimum cost maximum matching which, in this case, is $O(nm \log n)$ [11].

2.4 Related Results

In addition to the algorithms and NP-completeness results described above, we have obtained several other related results. We have found a linear time algorithm for the case that the bipartite graph is acyclic. This implies that optimal assignments can be found in a number of tree architectures such as those proposed by Singh [9, 10] and others. This algorithm can also be applied to any architecture to find an optimal solution when the fault pattern in the array induces an acyclic graph. Preliminary experimental results indicate that in many architectures this property is almost always true when the number of faulty elements is not excessively large.

We have also shown that the problem of finding optimal matchings is NP-complete for a number of specific architectures. For example, Chen *et al.* [2] conjectured that the polynomial time algorithm they proposed for the 2-D augmented mesh architecture could be generalized to higher-dimensional augmented meshes. We have shown that the problem of finding optimal matchings in the 3-D augmented mesh architecture is NP-complete, implying that it is very unlikely that the conjectured polynomial time algorithms exist.

3 Experimental Results

In this section we summarize experimental results obtained for the approximation algorithm described in the previous section. The approximation algorithm was implemented in C++ on a Sun Sparcstation and on an Encore Multimax computer.

In the first set of experiments, processor arrays with 32 primary processors and 16 spare processors were generated with connections between primary processors and spare processors selected randomly. In the first experiment we set $D(X) = 2$ and in the second experiment we set $D(X) = 4$. In each randomly generated array, faulty elements were introduced at random with fault frequency ranging from 0% to 30% in increments of 5%. (At fault frequency 35% there were already fewer spare processors than faulty primary processors, and thus no assignments could be found.) For each fault frequency value, 100 random arrays were generated. For each array, the approximation algorithm was used to find an assignment. In addition, the optimal assignment was found for each array using an exhaustive search algorithm and an assignment was found using standard unweighted matching. For $D(X) = 2$, the approximation algorithm found solutions that were at least 0.97 times the size of the optimal solution (on average), depending on the fault frequency. In contrast, the standard matching algorithm found solutions of size only 0.85 times optimal in the worst case. For $D(X) = 4$, the approximation algorithm found solutions that were at least 0.94 times the optimal size, whereas the standard matching algorithm found solutions as low as 0.75 times optimal. The results are summarized in Tables 1 and 2.

Additional experiments were performed for larger arrays and for several specific architectures. Figures 4 and 5 give results for randomly generated arrays with 100 primary processors and 50 spare processors for $D(X) = 2$ and $D(X) = 4$, respectively. The performance of the approximation algorithm is compared only to that of the standard matching algorithm, since these arrays were too large to be solved optimally by the exhaustive search algorithm. These results indicate that our approximation algorithm can find matchings that, in many cases, contain 6% to 14% more repairable processors than are obtained by standard matching. Experimental results for several specific architectures indicate that the approximation algorithm finds solutions that are frequently within 0.95 times the optimal size.

4 Conclusions

In this paper we have presented several efficient algorithms for bi-level reconfiguration of processor arrays. Our algorithms are quite general and can be applied to a large number of architectures. Although the optimal assignment reconfiguration problem is NP-complete in general, we have presented an approximation algorithm that can be used for all processor arrays. Experimental results indicate that this algorithm is very effective.

There are a number of interesting directions for future research. For example, we wish to find efficient reconfiguration algorithms for additional classes of architectures. We also plan to consider other types of local reconfiguration algorithms and their effect on system lifetime.

References

[1] P. Banerjee. Strategies for reconfiguring hypercubes under faults. In *Proceedings of the 20th International Symposium on Fault-Tolerant Computing*, pages 210–215, June 1990.

[2] C. Chen, A. Feng, T. Kikuno, and K. Torii. Reconfiguration algorithm for fault-tolerant arrays with minimum number of dangerous processors. In *Proceedings of the 21st International Symposium on Fault-Tolerant Computing*, pages 452–459, 1991.

[3] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, New York, 1979.

[4] R. Libeskind-Hadas. *Reconfiguration Problems for VLSI Systems.* PhD thesis, University of Illinois at Urbana-Champaign, In preparation.

[5] R. G. Melhem. Bi-level reconfigurations of fault tolerant arrays in bi-modal computational environments. In *Proceedings of the 19th International Symposium on Fault-Tolerant Computing*, pages 488–495, 1989.

[6] R. G. Melhem. Bi-level reconfigurations of fault tolerant arrays. *IEEE Transactions on Computers*, 41(2):231–239, Feb. 1992.

[7] K. Melhorn. *Data Structures and Algorithms 2: Graph Algorithms and NP-Completeness.* Springer-Verlag, Berlin, 1984.

[8] C. S. Raghavendra, A. Avizienis, and M. Ercegovac. Fault tolerance in binary tree architectures. *IEEE Transactions on Computers*, C-33(6):568–572, June 1984.

[9] A. D. Singh. A reconfigurable modular fault tolerant binary tree architecture. *Proceedings of the 17th International Symposium on Fault-Tolerant Computing*, pages 298–304, 1987.

[10] A. D. Singh. Interstitial redundancy: An area efficient fault tolerance scheme for large area VLSI processor arrays. *IEEE Transactions on Computers*, C-37(11):1398–1410, Nov. 1988.

[11] R. E. Tarjan. *Data Structures and Network Algorithms.* Society for Industrial and Applied Mathematics, Philadelphia, 1983.

[12] M. W. Yung, M. J. Little, R. D. Etchells, and J. G. Nash. Redundancy for yield enhancement in the 3-D computer. In *Proc. 1989 IEEE Int. Conf. on Wafer Scale Integration*, pages 73–82, 1989.

```
Input: Bipartite graph $G = (X \cup Y, E)$ and set $F$ where $D(X) \leq 2$.
Output: Safe matching for $F$ in $G$.

begin
  Construct an instance of 2-SAT as follows:
    1. For every edge $\{x_i, y_j\} \in E$ where $x_i \in F$, introduce boolean variable $u_{i,j}$.
    2. For every edge $\{x_i, y_j\} \in E$ where $x_i \in N$, introduce boolean variable $v_{i,j}$.
    3. For each vertex $x_i \in F$ adjacent to vertices $y_j$ and $y_k$,
       introduce clauses $(u_{i,j} \vee u_{i,k})$ and $(\neg u_{i,j} \vee \neg u_{i,k})$.
    4. For each pair of vertices $x_i, x_h \in F$ adjacent to a common vertex $y_j$,
       introduce clause $(\neg u_{i,j} \vee \neg u_{h,j})$.
    5. For each vertex $x_i \in N$ adjacent to vertices $y_j$ and $y_k$,
       introduce clause $(v_{i,j} \vee v_{i,k})$.
    6. For each vertex $x_i \in N$ and $x_h \in F$ adjacent to a common vertex $y_j$,
       introduce clause $(\neg v_{i,j} \vee \neg u_{h,j})$.
  Determine a satisfying assignment for the 2-SAT instance, if one exists,
    using a linear-time algorithm [7].
  if 2-SAT instance unsatisfiable halt
    for each boolean variable $u_{i,j}$ with value true do
    Match $x_i$ to $y_j$;
  endfor
end
```

Figure 1: Safe matching algorithm for $D(X) \leq 2$.

```
Input: Bipartite graph $G = (X \cup Y, E)$ and set $F$ where $D(Y) \leq 2$.
Output: Optimal matching for $F$ in $G$.

begin
  Use breadth-first search to find connected components of $G$;
  for each connected component $C$ do
    if $C$ contains at least as many spare vertices as primary vertices do
      Use breadth-first search to find a spanning tree $T$ in $C$;
      while $T$ contains vertices do
        Find a vertex $v \in T$ of degree 1;
        Match $v$ to its neighbor $w$;
        Remove $v$, $w$, and all incident edges from $T$;
      endwhile
    endif
    else (* $C$ is a tree *)
      Assign any primary vertex of $C$ to be the root;
      while $C$ contains a faulty vertex do
        Find a faulty vertex $v$ with no faulty descendants;
        if $v$ is a leaf do
          Match $v$ to parent $p(v)$; Remove subtree rooted at $p(v)$ from $C$;
        endif
        else if $v$ contains a child vertex $c(v)$ such that by
        matching $v$ to $c(v)$ no descendant of $v$ becomes unrepairable do
          Match $v$ to child $c(v)$; Remove the subtree rooted at $v$ from $C$;
        endif
        else if parent of $v$, $p(v)$, is non-faulty do
          Match $v$ to $p(v)$; Remove the subtree rooted at $p(v)$ from $C$;
        endif
        else if the tree obtained from $C$ by removing the subtree rooted at $v$
        does not contain more faulty primary vertices than spare vertices do
          Match $v$ to $p(v)$; Remove the subtree rooted at $p(v)$ from $C$;
        endif
        else
          Match $v$ to any child vertex $c(v)$; Remove subtree rooted at $v$ from $C$;
        endelse
      endwhile
    endelse
  endfor
end
```

Figure 2: Optimal matching algorithm for $D(Y) \leq 2$.

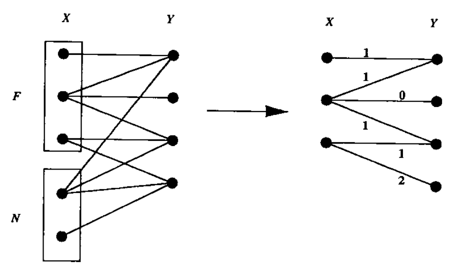

Figure 3: An example of the construction used in the $\frac{1}{D(X)}$-optimal approximation algorithm.

% faulty	*Optimal*	*Approximation Algorithm*		*Matching Algorithm*	
	# repairable	# repairable	$\frac{\text{repairable}}{\text{optimal}}$	# repairable	$\frac{\text{repairable}}{\text{optimal}}$
0	32.00	32.00	1.00	32.00	1.00
5	31.00	31.00	1.00	31.00	1.00
10	28.84	28.45	0.99	28.15	0.98
15	27.32	26.62	0.97	25.77	0.94
20	22.31	21.72	0.97	20.07	0.90
25	15.97	15.53	0.97	13.64	0.85
30	9.33	9.25	0.99	8.42	0.90

Table 1: Experimental results for randomly generated arrays with 32 primary processors, 16 spare processors, and $D(X) = 2$.

% faulty	*Optimal*	*Approximation Algorithm*		*Matching Algorithm*	
	# repairable	# repairable	$\frac{\text{repairable}}{\text{optimal}}$	# repairable	$\frac{\text{repairable}}{\text{optimal}}$
0	32.00	32.00	1.00	32.00	1.00
5	31.00	31.00	1.00	31.00	1.00
10	29.00	29.00	1.00	29.00	1.00
15	28.00	28.00	1.00	27.97	0.99
20	26.00	25.73	0.99	25.28	0.97
25	23.92	22.49	0.94	20.30	0.84
30	18.79	17.81	0.95	14.00	0.75

Table 2: Experimental results for randomly generated arrays with 32 primary processors, 16 spare processors, and $D(X) = 4$.

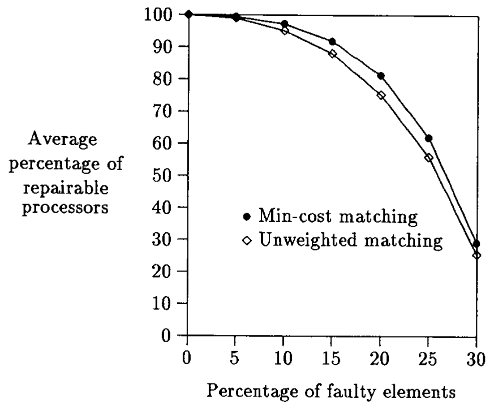

Figure 4: Randomly generated arrays with 100 primary processors, 50 spare processors, and $D(X) = 2$.

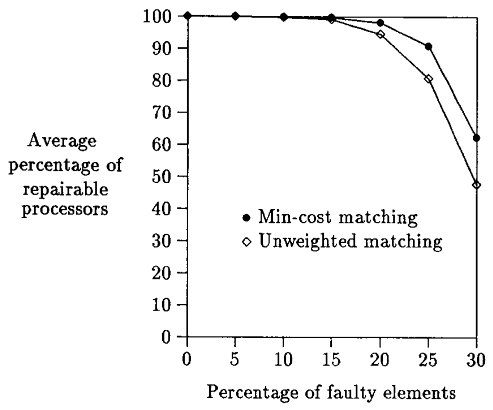

Figure 5: Randomly generated arrays with 100 primary processors, 50 spare processors, and $D(X) = 2$.