The Interlocking Bus Network For Fault-Tolerant Processor Arrays

Markus G. Siegle[1] Douglas S. Reeves[2] Krzysztof Kozminski[3]

Abstract

A method of fault-tolerance in mesh-connected processor arrays is presented. This method is based on a new type of interconnection network called the Interlocking Bus Network. The array can be reconfigured in the presence of faulty processors, using an algorithm for bipartite graph matching. The survivability of this method and its hardware/delay overhead are presented and compared to other schemes. The new technique is very general, leading to a number of important extensions. Application of the method to the BLITZEN parallel computer is also discussed.
Keywords: Fault-tolerance, reliability, reconfiguration, yield, processor array, interconnection network, bus, graph matching, BLITZEN.

1 Introduction

In modern high-performance computer systems there is usually some sort of parallelism involved. Massively parallel computers are being proposed and built, promising to provide outstanding computing power for many classes of applications. These computers are constructed from very simple processors (processing elements), each with its own local memory. The processors normally operate in SIMD mode (single instruction stream, multiple data stream), in which all processors execute the identical instruction stream under the control of a master processor (host). Examples of such computers are Illiac-IV[1], the Connection Machine[2], the MPP[3], and BLITZEN[4].

The processors are connected by some sort of interconnection network over which they exchange data. Most massively parallel computers use a two-dimensional mesh as the basic interconnection network (processor array). The two-dimensional mesh is attractive because it is easy to design and manufacture (e.g., it is planar, which matches current technology well) and the dataflow is simple to control. In addition, it has excellent performance characteristics.

Computers with thousands of processors and tens or hundreds of thousands of interconnections are likely to have severe fabrication yield and reliability problems. To make these computers practical, they must include techniques for tolerating faults. One way of achieving fault-tolerance is to provide hardware redundancy, in which case "spare" elements are provided which can replace faulty elements. This process is called reconfiguration. We have developed a new type of

[1]Institut für Informatik VII, Universität Erlangen, Martensstr.3, 8520 Erlangen,Germany.
e-mail siegle@faui77.informatik.uni-erlangen.de
[2]Department of Computer Science, North Carolina State University, Raleigh, NC 27965.
[3]MCNC Center for Microelectronics, Research Triangle Park, NC 27709.

interconnection network which supports, among others, the two-dimensional mesh interconnection pattern, and which is highly fault-tolerant. This network is called the **Interlocking Bus Network (IBN)**. The most important features of this network are:

1. The level of fault-tolerance provided, considering the amount of additional hardware and delay introduced, is superior to previous methods;

2. A well-known graph algorithm (bipartite graph-matching) provides provably optimal reconfiguration with modest complexity; and

3. IBN can be easily generalized to provide arbitrarily high fault-tolerance, at the cost of additional hardware.

In the next two sections, we present the network and the reconfiguration algorithm. This is followed by a discussion of the overhead incurred and a description of some important extensions and applications.

2 The Interlocking Bus Network

Our investigation of this problem was motivated by the BLITZEN project[4]. BLITZEN is a prototype SIMD computer, with 128 processors (and their associated memory) integrated onto a single chip of approximately 1.1 million transistors. Each processor is quite simple, and has a word width of 1 bit, i.e. it is bit-serial. In the full-scale version, 128 chips are configured to provide a 128×128 array of processors. Attaining satisfactory yield and reliability is a formidable challenge; this is due to the aggressiveness of the technology and the low production volume. A first-generation version of 128 processors per chip has been successfully tested, and applications are being ported to it. Plans for the second-generation mandate some form of fault-tolerance.

In BLITZEN, processors are connected by the "X-grid".[4] With this interconnection network, each processor can communicate with eight neighbours (in the N, NE, E, SE, S, SW, W, NW directions), with only 4 wires per processor. During data transfers, each processor outputs data on one of its four wires, accepts data on one of its other wires, and sets the remaining two wires to the high-impedance state. For our further discussion it is important to notice that each "X" in the grid can be viewed as a bus to which four processors are attached. At any one time, only one processor may be transmitting on the bus, while all others are merely receiving (i.e., the output is tri-stated). Logical and physical views of this interconnection network are shown in Figure 1a and Figure 1b. In this basic configuration there is no ability to tolerate faults, neither in the processors nor in the interconnection network.

It is well known that the bus is a basic communication medium which is highly tolerant of faults in processors; the loss of one processor on a bus does not prevent communication between any other processors also on that bus. However, a single bus is not a suitable method of interconnection for thousands of processors, because of excessive contention problems. A possible combination of high fault-tolerance with low contention would be to provide many small buses,

[4] This interconnection network is also being used in commercial computers supplied by the MasPar Corporation.

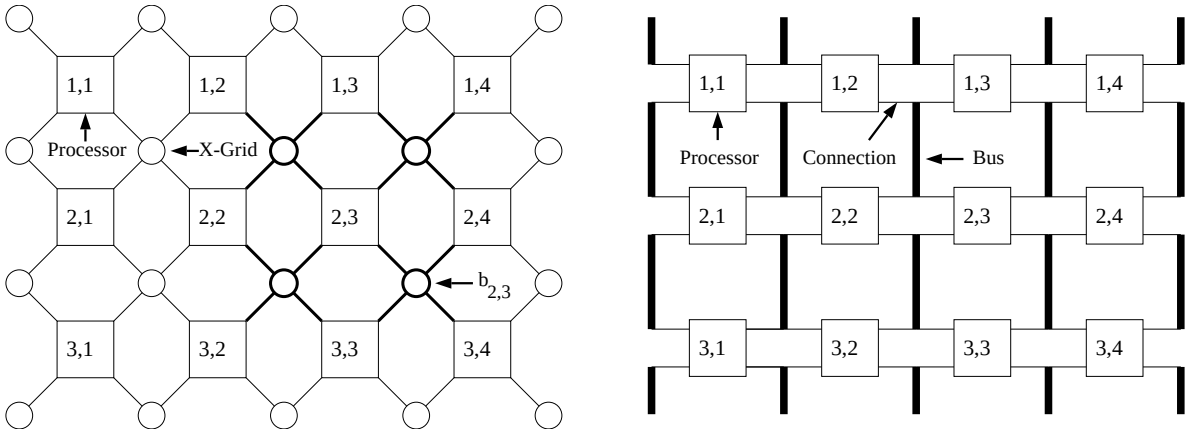

Figure 1: The BLITZEN architecture (processors and X-grid), a) logical view, b) physical construction (processors and connections not drawn to scale).

each with a small number of processors attached. To implement this idea, a way must be found of interconnecting these buses, so that the network will not be partitioned into non-communicating parts. As mentioned above, in BLITZEN the buses are connected *by the processors*, each of which is attached to multiple buses. In IBN, each processor may be configured to be connected to certain buses in its neighbourhood. Each processor contains switching logic to determine dynamically to which buses it is connected.

An example of our architecture follows. Let us first describe the basic non-fault-tolerant architecture. We will use the symbol $b_{i,j}$ to represent the X-grid bus which connects together processors $[i,j]$, $[i,j+1]$, $[i+1,j]$, and $[i+1,j+1]$. We will denote by $B[i,j]$ the set of four buses that are immediately adjacent to processor $[i,j]$. For example, $B[2,3]$ is shown highlighted in Figure 1a. In this basic architecture, $B[2,3] = (b_{1,2}, b_{1,3}, b_{2,2}, b_{2,3})$. Alternatively, we define $N_{i,j}$ as the neighbourhood of bus $b_{i,j}$, that is, the set of processors which it connects. In this same example, $N_{2,3} = ([2,3],[2,4],[3,3],[3,4])$.

In order to provide fault-tolerance, the interconnection network is expanded by additional connections which can be opened or closed by switches. Now each corner of a processor may connect to one out of *three* buses. This is illustrated in Figure 2. Note that in Figure 2 only the connections originating at one of the four corners of a processor are shown. There are three similar connections at each of the other corners. The extra connections allow a processor $[i,j]$ to be configured into the array in one of four ways:

1. Connected normally, to $B[i,j]$;
2. Connected North, to $B[i-1,j]$;
3. Connected West, to $B[i,j-1]$; or
4. Unconnected (i.e., configured out of the array).

To the array of processors, we will also add one extra row and one extra column of "spare" processors on the South and East borders. The additional bus connections and spare processors

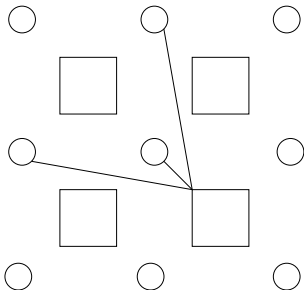

Figure 2: Possible connections between a corner of a processor and three buses

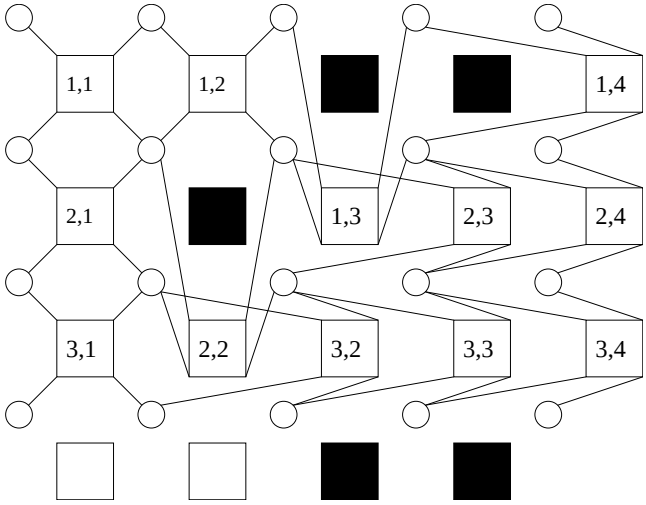

Figure 3: Example of reconfiguration, using one extra row and one extra column of processors. The indices represent logical processors.

allow the network to be reconfigured in the presence of faulty processors. The reconfiguration can be done by setting switches in the interconnection network. An example of reconfiguration using this architecture is shown in Figure 3. In this figure, a 3×4 array of processors is enhanced by one row and one column of spares. In the figure, faulty processors are shown shaded. In the example, logical processor $(1,3)$ is mapped onto physical processor $[2,3]$ because physical processor $[1,3]$ is faulty. This in turn causes logical processor $(2,3)$ to be moved right one position onto physical processor $[2,4]$, etc.

In this expanded network, each bus $b_{i,j}$ has connections to *eight* processors (four of which will be open and four of which will be closed). This means the neighbourhood of bus $b_{i,j}$ has changed. For example:
$N_{2,2} = ([2,2],[2,3],[3,2],[3,3],[4,2],[4,3],[2,4],[3,4])$
and
$N_{2,3} = ([2,3],[2,4],[3,3],[3,4],[4,3],[4,4],[2,5],[3,5])$.

In general, the defining features of the class of Interlocking Bus Networks are (i) processors are interconnected by buses and (ii) the buses must have partially overlapping neighbourhoods. In the above case,
$N_{2,2} \cap N_{2,3} = ([2,3],[3,3],[4,3],[2,4],[3,4])$
If a circle is drawn for each bus, with center at the midpoint of the bus and circumference just including those processors in the neighbourhood of the bus, the resulting diagram looks like a group of interlocking circles. Hence the name "Interlocking Bus Network". In addition to the present example, many other architectures (with different adjacency domains, interconnections, and location of spares) are also included in the IBN class according to these criteria.

A number of methods for reconfiguring processor arrays have been proposed in the past. In the fault-tolerant array taxonomy of Chean and Fortes[5], the IBN architecture would be classified as a method that is Globally Redundant, Processor Switched, and Locally Switched. Other reconfigurable architectures in this class are Direct Reconfiguration and Complex Fault-Stealing[6], FUSS[7], and CHIP[8]. Advantages shared by members of this class are:

1. With global redundancy, spares are allocated globally. A particular spare is not assigned to one element, but may potentially replace one out of a large set of elements. This gives a high degree of flexibility.

2. Using processor switching, a single fault does not cause a whole set of elements to be discarded which means improved fault-tolerance.

3. Local switching means strictly local communications; as a result, communication delays are tightly bounded.

We compare our method with the other members of this same class in more detail below. The hardware and delay overhead are estimated in section 4.

3 The Interlocking Bus Network and Graph Matching

We now explain how the array is successfully reconfigured when faults occur. The task in the example given in Section 2 is to produce a mapping of an array of $i \times j$ logical processors onto an array of $(i+1) \times (j+1)$ physical processors, augmented with the additional connections. Following the terminology of Negrini et al.[6], we define the Inverse Adjacency Domain of a logical processor (i,j) $(IAD(i,j))$ as the set of physical processors it can be mapped onto. In the example architecture, the Inverse Adjacency Domain of logical processor (i,j) is the set $([i,j],[i+1,j],[i,j+1])$ of physical processors. A logical processor can be mapped onto its "twin" (physical processors with same indices) or the eastern or southern neighbour of its twin. On the other hand, the Adjacency Domain of a physical processor $[i,j]$ (denoted $AD[i,j]$) is the set of logical processors which can be mapped onto this physical processor.

Let us define a *feasible mapping* as any mapping in which every logical processor is assigned to a non-faulty physical processor in its Inverse Adjacency Domain. Furthermore, *any* feasible mapping of logical to physical processors that assigns at most one logical processor to any physical

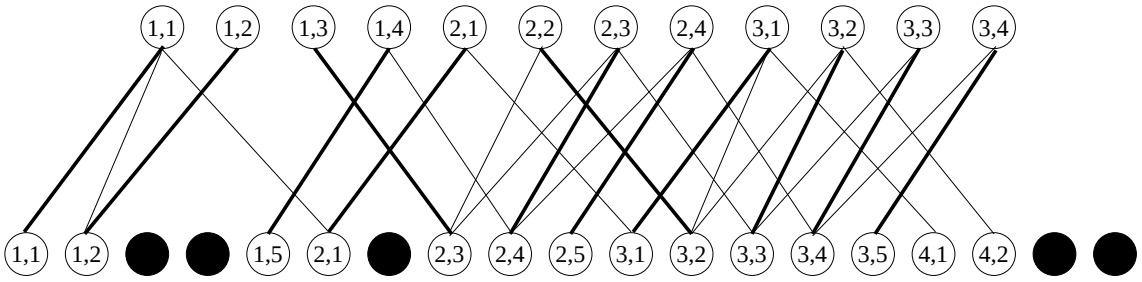

Figure 4: Bipartite graph representing the mapping problem of Figure 3. A logical array of size 3×4 must be mapped onto a physical array of size 4×5. An optimal matching is shown by the darkened lines.

processor *will produce a workable reconfiguration.* The key property of the IBN can now be mentioned. The only effect of an assignment of a logical processor to a physical processor is the removal of this physical processor from all other Inverse Adjacency Domains to which it belongs. The interlocking bus connections guarantee that the network can be reconfigured to implement such a mapping. This property is the major novel feature of IBN and the major finding of this paper.

Because of this property, we can formulate the reconfiguration task as an instance of a graph matching problem. This approach has previously been used for the Interstitial Redundancy Array[9]. A bipartite graph consists of two sets of vertices, X and Y, and a set of edges, E. The task in bipartite graph matching is to find a subset of edges, $E_0 \subseteq E$, such that no two edges in E_0 share an end point, and such that every vertex in X is connected to exactly one vertex in Y by an edge in E_0. For our purposes, there is one vertex in X for each logical processor in the array; similarly, there is one vertex in Y for each physical processor in the array. Each edge connects a vertex x_i in X with a vertex y_j in Y. An edge from x_i to y_j represents the fact that the the physical processor represented by y_j is in the Inverse Adjacency Domain of the logical processor represented by x_i. The bipartite graph which corresponds to Figure 3 is shown in Figure 4.

There are well-known graph algorithms for solving this problem *optimally*; that is, if there is any way to reconfigure the array in the presence of a specific pattern of faults, such a way will be found by the algorithm. Note that some fault patterns (e.g., a large cluster of neighbouring faulty processors) may make it impossible to reconfigure, despite the fact that the number of faults does not exceed the number of spare processors. Algorithms for graph matching have low complexity. For non-weighted matching, the complexity of solution is $O(|E| * |X|^{1/2})$ [10]. In the example architecture, $|E| \leq 3|X|$, which results in a bound of $O(|X|^{3/2})$ in this particular case. The reconfiguration algorithm for all networks that fall into the IBN class is always the same, regardless of the extra connections or number/location of spares. We return to this point in section 5.

As mentioned above, there are other proposals for array reconfiguration which are Globally Redundant, Locally-, and Processor-Switched methods. In order to quantitatively compare IBN

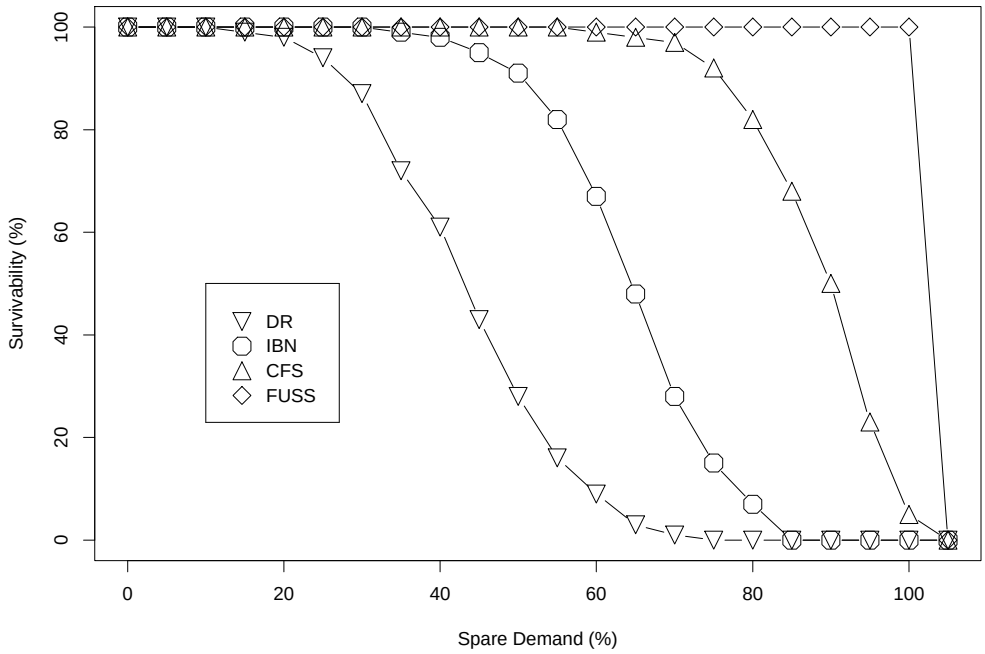

Figure 5: Comparison of survivability of a 20×20 processor array, for four reconfiguration methods, including the proposed method (IBN).

with these other methods, we have simulated the reconfiguration of a 20×20 array. In this simulation, faults were assumed to be independent and equally likely at every location in the array (including the spares). After 100,000 trials, the results were tabulated as Survivability vs. Spare Demand; survivability is defined as (# successful reconfigurations / # trials) and spare demand is defined as (# faulty processors / # spares).

The results are shown in Figure 5. Data for Direct Reconfiguration (DR), Complex Fault Stealing (CFS), and Full Utilization of Suitable Spares (FUSS) come from the article by Chean and Fortes[5]. We note the following from this graph:

- Survivability of IBN is considerably higher than for Direct Reconfiguration.
- Survivability is lower than for Complex Fault Stealing and FUSS.

One has to take into account several other issues which are not addressed in this graph, most importantly the additional (redundant) hardware required and the extra delay incurred during communication. These are now discussed.

4 Hardware Design and Cost

The hardware overhead of IBN consists of increased switch complexity and extra wiring. In the non-fault-tolerant X-grid, there is a bi-directional pad at each corner of a processor for connecting to the adjacent bus. This is illustrated in Figure 6a. In the fault-tolerant version, a multiplexer

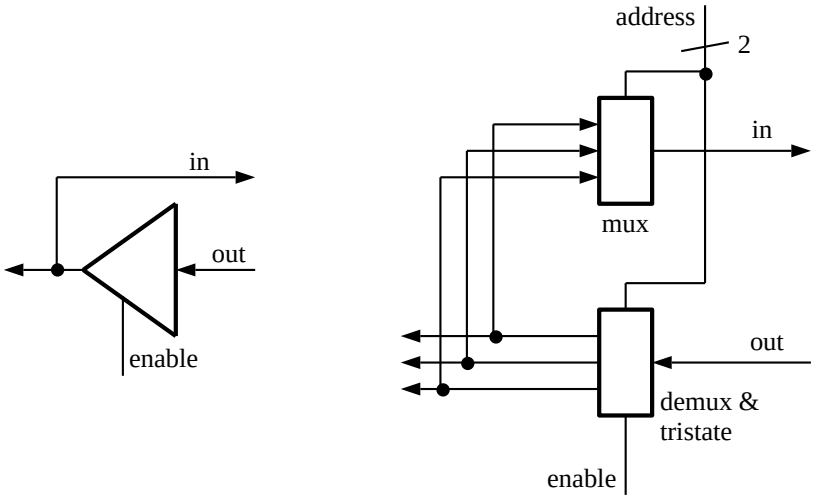

Figure 6: (a) I/O buffer at each corner of each processor for the non-fault-tolerant BLITZEN. (b) Architecture of a switch at each corner of each processor for the fault-tolerant version of BLITZEN (IBN).

must be used to determine which bus to read from, and a demultiplexer determines which bus is being written to (when enable is high). The switch design for our example architecture is illustrated in Figure 6b. While the switches are obviously more complex in IBN, they are straightforward to implement.

The logic for setting the "direction" bits is part of the reconfiguration mechanism. In the simplest implementation the graph matching algorithm is run on a host computer. Input for the reconfiguration algorithm are the number and location of faulty processors. Reconfiguration information is computed on a per processor basis and this information is then downloaded into the processor array. One could also think of computing the reconfiguration information on-line in the processor array using efficient hardware mechanisms. Each processor $[i, j]$ requires $\log_2(|AD[i, j]|+1)$ bits of storage to hold the direction information. For the example architecture this is 2 bits (corresponding to the 4 ways each processor can be configured). The area overhead for the extra switching logic and configuration registers in the reconfiguration example of this paper is less than 3% of the normal (non-fault-tolerant) processor area.

As for wiring overhead, the number of wires per processor is equal to $4|AD[i, j]|$. In the example architecture this is 12 wires, as illustrated in Figure 7. Maximum wire length is determined solely by the location of the processors in the Inverse Adjacency Domain. Again, for the example architecture, the extra wiring adds less than 1% to the total array area. The only significant increase in area is due to the additional spare processors along the array boundaries.

Overall, hardware complexity for this example of the IBN architecture is comparable to the Direct Reconfiguration scheme of [6] (called "Simple Fault-Stealing, Fixed Choice" in that source), and considerably less than the Complex Fault Stealing method. As a result, we claim the network attains good fault-tolerance with very reasonable hardware requirements. Survivability

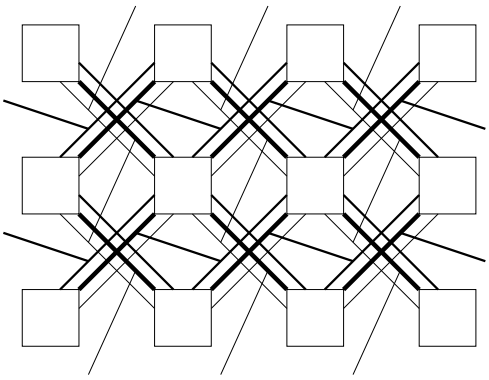

Figure 7: Wiring required for the fault-tolerant network, with Inverse Adjacency Domain $[i, j]$, $[i+1, j]$, and $[i, j+1]$.

very similar to Complex Fault Stealing is achieved if the Inverse Adjacency Domain of each logical processor (i, j) is expanded to include physical processor $[i+1, j+1]$.

In previous work on array reconfiguration, area overhead is almost always acknowledged, while delay "overhead" (penalty) is often given much less attention. In the authors' experience, delay is a much more important factor than area. For IBN there is extra delay due to the multiplexing/demultiplexing of the switches, and delay due to the extra wire length and capacitance. In typical implementations, neither of these is very severe. By contrast, methods such as CFS, FUSS, and CHIP rely on a switched-bus mechanism for reconfigurability. With a switched-bus, communication paths between "adjacent" processors may be quite long. In addition, the delay incurred by passing data through multiple switches in series can seriously degrade performance.

The IBN network is a reasonable balance of high fault-tolerance with modest hardware overhead. It does not suffer the serious delay penalty of some of the other methods. What is more, the basic technique is highly flexible; some extensions are now shown.

5 Applications and Extensions

As was stated, the reconfiguration of an IBN requires only that the bipartite graph be provided; the algorithm (graph matching) remains the same. This graph captures all information about array size (number of vertices in X), adjacency domains (edges), location of spares (by suitable labeling of the vertices in Y), and location of faults (by removal of edges and vertices for faulty processors). As a result, we can immediately adapt the method to the following situations:

- Vary the number and location of spares, for example, by providing *two* extra columns and rows of spares.
- Vary the adjacency domains. For example, the adjacency domain may be customized on a per processor basis. This might reflect some specialization of processors or some

concentration of hardware redundancy based on predicted local failure probabilities.

- Vary the interconnection topology. The technique is not restricted to rectangular two-dimensional meshes. For example, it can be used with hexagonal meshes. Even more promising, it can be used with *three-dimensional* networks. Emerging technologies for connecting die to substrates and printed circuit boards to printed circuit boards are likely to be three-dimensional in nature. Most reconfiguration methods previously proposed do not extend obviously to three dimensions.

- When reconfiguration fails (because there is no possibility of reconfiguration with the given pattern of faults), a *maximal matching* is guaranteed. In this case, "maximal" means that no reconfiguration algorithm could match more logical processors to available physical processors. Thus the method provides a measure of *graceful degradation* with increasing number of faults.

We now discuss the application of the proposed method to the BLITZEN computer, our original motivation. As mentioned, the basic component of BLITZEN is a one-million transistor chip which contains an 8×16 array of processing elements, each with 1-Kbit of memory. A serious challenge with this level of integration is attaining satisfactory chip yield. For the example IBN architecture (Adjacency Domain of $[i,j],[i,j+1],[i+1,j]$), we can determine the yield of chips for different fault probabilities.

Let p_k represent the probability of k faults occurring in an array of size s. If each processor is assumed independently faulty (non-faulty) with probability f (1-f), then p_k can be approximated by the binomial distribution: $p_k \approx b(k;s,f)$. The expected yield Y under these assumptions can be computed as $Y = \sum_{k=0}^{s} p_k \times succ_k$, where $succ_k$ is the probability that an array with k faults can be successfully reconfigured, and is tabulated directly from the above simulation results. The expected yield without fault-tolerance is simply p_0.

For the example architecture of this paper, we have calculated expected yield Y vs. probability of failure f. For the non-fault-tolerant array, the size of the array (s) is equal to 128 processors (8×16). For the fault-tolerant array, the size is 152 processors ($128 + 8 + 16$), reflecting the increased area (and thus increased likelihood of defects) of the fault-tolerant version. The results are shown in Figure 8 and are quite favourable. Yield may be dramatically increased using the IBN architecture. It should be stated that in practice faults are not uniformly and independently distributed. More research is required to precisely measure the impact of fault-tolerance on yield.

A special requirement of the current BLITZEN chip is that all processors in the same row are connected to a single, special I/O bus. To avoid difficulties of rerouting the I/O bus, it would be much better in this situation only to allow substitutions of processors in the same row. As an example, the Inverse Adjacency Domain for logical processor (i,j) can be restricted to $[i,j-1],[i,j],[i,j+1]$, with a spare column on either side of the array, and no spare rows. Again, the architecture and algorithm adapts easily to this restriction.

Occasionally, a chip may develop faults so numerous or so serious that on-chip fault-tolerance is to no avail. In such a case, there should be another fault-tolerance scheme for replacing faulty chips. The method presented in this paper can be extended to work in a hierarchical fashion. For each level of the hierarchy there will be a bipartite graph representing the mapping of logical

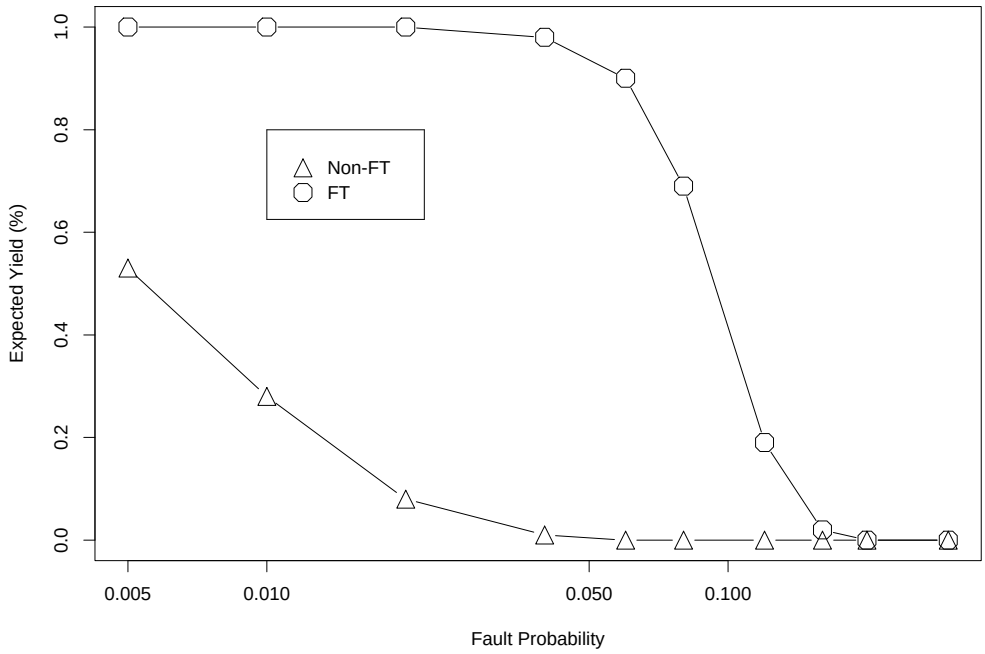

Figure 8: Comparison of yield for non-fault-tolerant and fault-tolerant versions of the BLITZEN chip.

to physical "blocks". At level 0 of the hierarchy, a block is simply a single processor; at level 1, a block might be an $m \times n$ region of processors; and so forth. Each level of the hierarchy will require additional connections between logically adjacent blocks. If chip interconnections on the printed circuit board or substrate must be kept simple, the method of this paper is probably *not* the best choice for reconfiguration on that level. A technique with lower survivability and lower complexity, such as row or column substitution[3], might be a better choice for reconfiguration on that level. This concludes our discussion of applications of IBN.

6 Conclusions

We have developed a fault-tolerant architecture for massively parallel computers, based on the Interlocking Bus Network. This method balances high survivability with modest hardware overhead and very little additional delay. A highlight of this architecture is that reconfiguration can be optimally computed using bipartite graph matching. As a result, the method is very powerful, and can be generalized in a number of useful ways. An open question is whether there are efficient hardware mechanisms for performing the graph matching "on-line" in the array itself.

We have discussed its application to the BLITZEN computer, for which we hope to develop a fault-tolerant version. Hierarchical fault-tolerance can be used to deal with fault clustering, which thwarts the (local) reconfiguration scheme. A problem which remains to be fully addressed is how to cope with faults in the interconnect and in the switches. We are currently investigating this important issue.

References

[1] R. Michael Hord. *The Illiac-IV: The First Supercomputer*. Computer Science Press, 1982.

[2] D. Hillis. *The Connection Machine*. MIT Press, 1985.

[3] K.E. Batcher. Design of a Massively Parallel Processor. *IEEE Trans. Comp.*, C-29:836–840, sep 1980.

[4] D.W. Blevins, E.W. Davis, R.A. Heaton, and J.H. Reif. BLITZEN: A Highly Integrated Massively Parallel Machine. *Journal of Parallel and Distributed Computing*, pages 150–160, aug 1990.

[5] M. Chean and J.A.B. Fortes. A Taxonomy of Reconfiguration Techniques for Fault-Tolerant Arrays. *IEEE Computer*, pages 55–69, jan 1990.

[6] R. Negrini, M. G. Sami, and R. Stefanelli. *Fault Tolerance Through Reconfiguration in VLSI and WSI Arrays*. MIT Press, 1989.

[7] M. Chean and J. A. B. Fortes. Fuss: A reconfiguration scheme for fault-tolerant processor arrays. In *Intl. Wkshp. on Hardware Fault Tolerance in Multiprocessors*, pages 30–32, June 1989.

[8] K. S. Hedlund and L. Snyder. Wafer-scale integration of configurable, highly-parallel architectures and processors. In *Proc. 1982 Intl. Conf. on Parallel Processing*, pages 262–264. IEEE Computer Society Press, 1982.

[9] A. Singh. Interstitial Redundancy: An Area Efficient Fault Tolerance Scheme for Large Area VLSI Processor Arrays. *IEEE Trans. Comp.*, C-37(11):1398–1410, nov 1988.

[10] K. Mehlhorn. *Data Structures and Algorithms 2: Graph Algorithms and NP-Completeness*. Springer-Verlag, 1984.