ELSEVIER

Microprocessors and Microsystems 20 (1997) 423–428

MICROPROCESSORS AND MICROSYSTEMS

An on-line fault diagnosis scheme for linear processor arrays

Ding-Ming Kwai, Behrooz Parhami*

Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106-9560, USA

Received September 1995; revised November 1996; accepted December 1996

Abstract

A data-driven method for error detection and fault diagnosis in processor arrays is proposed under the assumption that data streams can only be inserted and observed through the boundary processors. The method consists of attaching tags to data streams, thereby allowing the data items to carry their own control and error information. Our goal is to detect the malfunction of a specific processor at a specific time step. A tag, which initially contains control information to activate a testing process, is changed to indicate the occurrence of an error by a checking processor detecting an inconsistency. To pinpoint a faulty processor, the front-end computer must go through the reverse process of identifying the processor that detected and signalled the inconsistency. Using two data streams, we can control every processor in the array and locate the faulty one. The resulting processor array is regular in structure, and the number of bits used to encode the control and error information is independent of the size of the array, thus leading to efficiency and scalability. © 1997 Elsevier Science B.V. © 1997 Elsevier Science B.V.

Keywords: Data-driven control; Error detection; Fault diagnosis; Fault tolerance; Processor array; Scalability; Systolic computation

1. Introduction

Fault tolerance in processor arrays requires provision of redundancy to detect, locate, and recover from errors. As classified in Ref. [1], redundancy is generally embodied in space, time, information, or algorithm. The simplicity of hardware implementation leads to various methods for on-line testing of processor arrays based on *duplication with comparison* [2]. The space redundancy technique is stated as follows. Consider a computation C_{21} done on processor P_2, whose result is needed by computations C_{12} and C_{32} at neighbouring processors P_1 and P_3, respectively (see the space–time diagrams in Fig. 1). For error detection, the original computations are adapted in such a way that duplicate computations can be entered in adjacent (shaded) slots. This redundancy technique can be easily generalized by adding a second redundant version for each computation so as to provide fault masking by independent voting [3,4]. The duplicate and original computations are interspersed, thus constituting different *states* of each processor.

Owing to their compact layout and limited input/output, processor arrays are inherently of limited observability and controllability. The approach proposed in Ref. [5] solved the problem by emitting a pipelined *trigger* signal to initiate the comparison of intermediate results from a neighbouring processor, but it assumed that comparison outcomes at interior processors can be directly observed. Such an approach is undesirable for scalability reasons. A more appropriate assumption is that the comparison outcomes can only be observed via the outermost processors at the boundaries of an array. Under such an assumption, if a data stream carrying error information is propagated through the array, faults can be detected by monitoring the error information.

However, not all implementations possess the convenient property that fault effects are localized, as shown in Ref. [6]. If error signals are propagated downstream as tags attached to a single data stream, pinpointing the faulty processor would be impossible since the error signal could have been generated by any processor along that data stream.

It has been suggested [7] that a linear array can be split into several segments and the error information monitored at segment boundaries to diagnose the fault to within one segment. In the extreme case of monitoring the error information at each processor interface, complete diagnostic capability is attained. This approach, however, requires additional access to the individual segments and is therefore impractical or at least inefficient.

Algorithm-based fault tolerance [8,9] can be used to locate a faulty processor with insignificant space and time costs, but its application is restricted to specific algorithms and may fail in certain implementations [10]. The band matrix–vector multiplication implemented on a bidirectional linear array, shown in the next section, is one such

* Corresponding author. Tel: +1 805 893 3211; fax: +1 805 893 3262; e-mail: parhami@ece.ucsb.edu.

$p \backslash t$	1	2
P_1	C_{11}	C_{12}
P_2	C_{21}	C_{22}
P_3	C_{31}	C_{32}

$p \backslash t$	1	2	3	4
P_1	C_{11}	▨	C_{12}	▨
P_2	C_{11}	C_{21}	C_{12}	C_{22}
P_3	C_{31}	C_{21}	C_{32}	C_{22}
P_4	C_{31}		C_{32}	

$p \backslash t$	1	2	3	4	5	6
P_1	C_{11}	▨		C_{12}	▨	
P_2	C_{11}		C_{21}	C_{12}		C_{22}
P_3	C_{11}	C_{31}	C_{21}	C_{12}	C_{32}	[illegible]
P_4		C_{31}	C_{21}		C_{32}	C_{22}
P_5	▨	C_{31}		▨	C_{32}	

Fig. 1. Space–time diagram of interleaved duplicate computations (t: time, p: processor).

example. Detection of an error and location of a faulty processor after the entire computation is finished may not be desirable, especially as the processor arrays expand in size. A common divide-and-conquer solution to improve diagnostic efficiency is to add extra observation points. Unfortunately, the algorithm-based approach does not benefit from the addition of observation points.

In this paper, we apply the idea of data-driven control of processor arrays to error detection and fault diagnosis. Our goal is to detect the malfunction of a specific processor at a specific time step. A tag, which initially contains control information, is attached to each data item and pipelined along with it. The tag is changed by a checking processor, which assumes this role because of the instruction on the tag, to indicate the occurrence of an error. To pinpoint a faulty processor, the front-end computer must go through the reverse process of identifying the checking processor that detected the inconsistency and signalled the error. In such a case, two data streams are used to carry the control and error information. The reason for this is obvious. The space–time diagram of any computation on a linear array can be represented on a two-dimensional plane; two one-dimensional data streams then suffice to identify every processor in the array. Similarly, computation on a mesh-connected array can be represented on a three-dimensional space–time diagram. The data stream, with data items moving in parallel, can be viewed as a two-dimensional plane. Again, two data streams suffice to control every processor and to locate the faulty one in such an array.

We will introduce our method by applying it to a particular linear array which follows the fault model of previous work where errors are assumed to result from stuck-at faults or short transients affecting a single processor. The error manifests itself as a discrepancy between one output of the faulty processor and the redundant output in an adjacent processor. The rest of the paper is organized as follows. In Section 2, we introduce a bidirectional linear array for matrix–vector multiplication. Section 3 deals with the error detection method and its associated synthesis procedure. In Section 4, we extend the method to provide fault diagnosis capability. Section 5 contains our conclusions.

2. Matrix–vector multiplication

We now consider a particular architecture for band matrix–vector multiplication which allows each output to be calculated in each of two adjacent processors. The matrix–vector multiplication $\boldsymbol{y} = \boldsymbol{A} \cdot \boldsymbol{x}$ consists of a sequence of inner product step (IPS) operations. It can be written as a set of uniform recurrence equations as follows:

$$y_i^{(j)} \leftarrow y_i^{(j-1)} + a_{ij} x_j^{(i-1)}$$

$$x_j^{(i)} \leftarrow x_j^{(i-1)}$$

where $A = (a_{ij})$ is an $m \times n$ matrix, $\boldsymbol{x} = [x_1\ x_2 \ldots x_n]^{\mathrm{T}}$ is an n-vector, and $y = [y_1\ y_2 \ldots y_m]^{\mathrm{T}}$ is an m-vector. Fig. 2 shows the dependence graph in a two-dimensional Cartesian coordinate for the band matrix–vector multiplication when the band width (the number of diagonals) is $w = 3$. The nodes in the graph represent the inner product steps and the arcs denote the movement of data items.

The algorithm can be implemented using an array of IPS cells, each with three data streams. A complete discussion of different types of IPS cells can be found in Ref. [11]. The resulting design forms a regular bidirectional linear array (BLA) that uses two data streams flowing in opposite directions. One of the data streams is the output data stream $\boldsymbol{y}$. This implementation requires additional input/output time

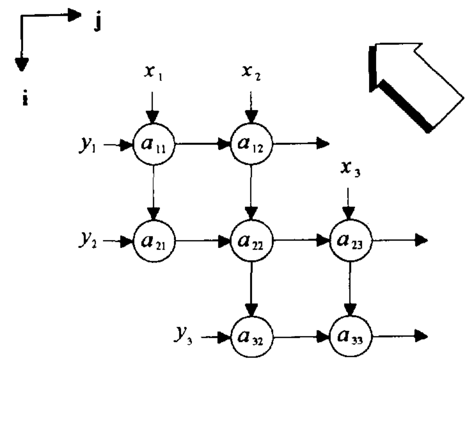


Fig. 2. Dependence graph for matrix–vector multiplication. The projection direction and scheduling (diagonal dotted lines) are also shown.

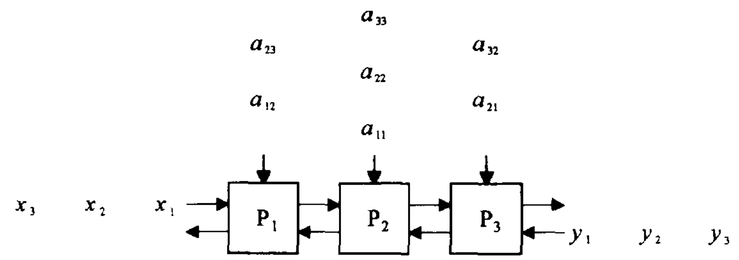


Fig. 3. Regular BLA for matrix–vector multiplication. Each block is an IPS cell.

$p \backslash t$	0	1	2	3	4	5	6
P_1	x_1	–	$y_1 \leftarrow y_1 + a_{12} \times x_2$	–	$y_2 \leftarrow y_2 + a_{23} \times x_3$	–	$y_3 \leftarrow y_3$
P_2		$y_1 \leftarrow y_1 + a_{11} \times x_1$	–	$y_2 \leftarrow y_2 + a_{22} \times x_2$	–	$y_3 \leftarrow y_3 + a_{33} \times x_3$	–
P_3	$y_1 \leftarrow y_1$	–	$y_2 \leftarrow y_2 + a_{21} \times x_1$	–	$y_3 \leftarrow y_3 + a_{32} \times x_2$	–	x_3

Fig. 4. Space–time diagram of the BLA executing the band matrix–vector multiplication. The symbol " – " denotes that the input or output is unspecified or irrelevant.

for some data items to move to where the first computation takes place or to exit from where the last computation finishes. The transformation from a set of uniform recurrence equations to a processor array is a linear mapping onto space and time domains. The method is described, for example, in Refs. [10] and [12]. Fig. 3 shows the BLA with its associated data flow [13].

The space–time diagram of the BLA is shown in Fig. 4. Time steps 0 and 6 have been added to correspond to the input of x_1 and the output of y_3. From Fig. 4, we note that at every time step, either odd-numbered processors or even-numbered processors are idle. Thus, if the idle processors are allowed to do the same operations as the active ones and the results are compared at the next time step, no loss of speed or efficiency occurs. During the computation, an error is detected if a discrepancy exists between these two results. To allow this, the processors must be equipped with mechanisms to detect and report the error to the front-end computer.

3. Error detection

If we use processor P_1 to check the computation result for P_2, and P_2 to check P_3, we will need another processor to do the checking for P_1. A redundant processor (denoted as P_0) is added to the left end of the BLA as shown in Fig. 5. After completing an inner product step, the checking processor not only sends the result to its left neighbour but also stores the result in its local storage. At the next time step, it compares the received data value with its stored value while at the same time performing the inner product step for newly received data items. A partial order exists between these two node operations. The first requires the processor to store the computation result and operand, and the second consists of a comparison of the stored value with the received data values.

These two node operations correspond to two different types of nodes in the dependence graph. The dependence graph is derived by duplicating the original dependence graph of Fig. 2 and interleaving the two along dotted equitemporal lines, as shown in Fig. 6. The dark node computes the inner product step, passes it on to the right as usual, and also sends the result diagonally to a clear node. The clear node compares the horizontally and diagonally incoming data values while at the same time computing the inner product using the received data values. The diagonal transfer in Fig. 6 corresponds to a saving of local values from one time step to the next in the BLA. Hence, the clear and dark nodes are associated with "compute-and-store" and "compare-and-compute" processor functions, respectively.

Since each data stream $\boldsymbol{x}$ or $\boldsymbol{y}$ flows in a regular fashion without meeting more than one type of node operation, it is sufficient to use only one of the data streams (either $\boldsymbol{x}$ or $\boldsymbol{y}$) to carry instructions to each processor for the two different node operations. We can assign a tag bit "0" to specify that the processor should do the "compare-and-compute" operation and "1" to mandate that the processor do the "compute-and-store" operation. The method can be generalized to synthesize pipelined control signals if multiple and

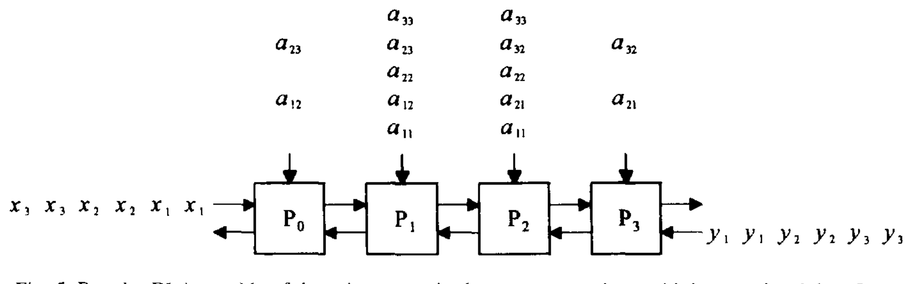

a_{33} a_{33}
a_{23} a_{23} a_{32} a_{32}
a_{22} a_{22}
a_{12} a_{12} a_{21} a_{21}
a_{11} a_{11}
x_3 x_3 x_2 x_2 x_1 x_1
P_0
P_1
P_2
P_3
y_1 y_1 y_2 y_2 y_3 y_3

Fig. 5. Regular BLA capable of detecting errors in the processors, along with its associated data flow.

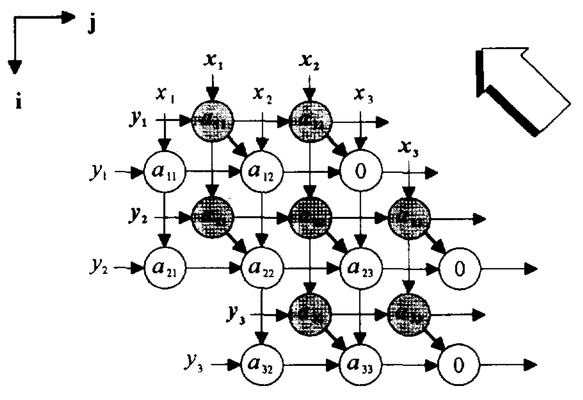


Fig. 6. Dependence graph for matrix–vector multiplication with space redundancy. The two types of nodes are interleaved along the equitemporal lines.

more complex functions are implemented on the processor array [14–16].

With the data-driven control scheme, a tag containing control information is attached to each data item. When a processor receives a data item, it first examines the tag and then decides which operation it should perform. The redundant operations for error detection are to compare or to store the intermediate results. Since neither the comparator nor the latch is more complex than an adder, the hardware costs to the regular BLA are clearly seen to be modest. The space–time diagram in Fig. 7 shows the execution of matrix–vector multiplication on the BLA of Fig. 5. At each time step, two processors perform the same inner product step. One processor in each such pair also compares the intermediate result received from a neighbouring processor to its stored value from the previous time step. Here we have chosen to provide tags on data stream y to control the operations to be performed by the processors.

If we allow the processors to modify the tags once they detect incompatible results, the tags can also indicate whether any error has occurred during the computation. The comparison operation on a data item is performed only when the processor receives a tag bit 0 (or a "compare-and-compute" instruction). We can simply invert the bit from 0 to 1 to indicate the occurrence of an error. The tag bit 1 also mandates the processors downstream to stop checking the output data item. Because the error propagates along the data flow path and each successive comparison would certainly yield incompatible results, it is not necessary to continue the comparison operation for this data item.

The tag values T_1, ..., T_m are then compared to the preassigned values $T_1^{(0)}$, ..., $T_m^{(0)}$ (m is the number of outputs), and the error vector $e = (e_1 \ldots, e_m)$ is computed where for $1 \le i \le m$:

$$e = \begin{cases} 0 & T_i = T_i^{(0)} \\ 1 & T_i \neq T_i^{(0)} \end{cases}$$

The error signal e_i at position index i indicates that at least one of the two y_i outputs is incorrect. Hence, an error is detectable if and only if the m-bit error vector is not equal to (0, ..., 0). The modified tags propagate with the data items to the boundary processor. It can be seen that error detection is not postponed until the whole computation is completed. Rather, an error is signalled with the appearance of the first 1 in the error vector.

For example, let us suppose that a transient error occurs in processor P_2 at time step 4 (see Fig. 7). Processor P_1, after comparing the data value from P_2 with its own stored data value (calculated locally at time step 4), finds the discrepancy. It then changes the tag on y_2 to 1 at time step 5. We can deduce from the tag attached to the data stream $\boldsymbol{y}$ which is changed from $\boldsymbol{T}_y^{(0)} = (000)$ to $\boldsymbol{T}_y = (010)$ a nonzero error vector $\boldsymbol{e}_y = (010)$, indicating an error in y_2. Unfortunately, we are unable to determine which processor changed the tag on y_2. Since the tagged data item y_2 flows to P_3 at time step 3, P_2 at time step 4, P_1 at time step 5, and P_0 at time step 6, any of the four processors could have changed the tag.

Note that if all of the tags attached to the data items are set

$p\backslash t$	0	1	2	3	4	5	6	7	8
P_0	x_1 ( )	x_1 ( )	x_2 ( )	$y_1 \leftarrow y_1+$ [illegible] $L \leftarrow y_1$ (1)	$L\ ?{=}\ y_1$ $y_1 \leftarrow y_1$ x_3 (0)	$y_2 \leftarrow y_2+$ [illegible] $L \leftarrow y_2$ (1)	$L\ ?{=}\ y_2$ $y_2 \leftarrow y_2$ (0)	$y_3 \leftarrow y_3$ $L \leftarrow y_3$ (1)	$L\ ?{=}\ y_3$ $y_3 \leftarrow y_3$ (0)
P_1	– ( )	x_1 ( )	$y_1 \leftarrow y_1+$ [illegible] $L \leftarrow y_1$ (1)	$L\ ?{=}\ y_1$ $y_1 \leftarrow y_1 +$ $a_{12} \times x_2$ (0)	$y_2 \leftarrow y_2+$ [illegible] $L \leftarrow y_2$ (1)	$L\ ?{=}\ y_2$ $y_2 \leftarrow y_2 +$ $a_{23} \times x_3$ (0)	$y_3 \leftarrow y_3+$ [illegible] $L \leftarrow y_3$ (1)	$L\ ?{=}\ y_3$ $y_3 \leftarrow y_3$ (0)	
P_2	– ( )	$y_1 \leftarrow y_1$ $L \leftarrow y_1$ (1)	$L\ ?{=}\ y_1$ $y_1 \leftarrow y_1 +$ $a_{11} \times x_1$ (0)	$y_2 \leftarrow y_2+$ [illegible] $\times x_1$ $L \leftarrow y_2$ (1)	$L\ ?{=}\ y_2$ $y_2 \leftarrow y_2 +$ $a_{22} \times x_2$ (0)	$y_3 \leftarrow y_3+$ [illegible] $L \leftarrow y_3$ (1)	$L\ ?{=}\ y_3$ $y_3 \leftarrow y_3 +$ $a_{33} \times x_3$ (0)	x_3	
P_3	$y_1 \leftarrow y_1$ $L \leftarrow y_1$ (1)	$L\ ?{=}\ y_1$ $y_1 \leftarrow y_1$ (0)	$y_2 \leftarrow y_2$ $L \leftarrow y_2$ (1)	$L\ ?{=}\ y_2$ $y_2 \leftarrow y_2 +$ $a_{21} \times x_1$ (0)	$y_3 \leftarrow y_3$ [illegible] $L \leftarrow y_3$ (1)	$L\ ?{=}\ y_3$ $y_3 \leftarrow y_3 +$ $a_{32} \times x_2$ (0)	x_2	x_3	x_3

Fig. 7. Space–time diagram of the BLA capable of detecting errors. The tag is shown in parentheses. L is a data latch in each processor. The comparison operation is denoted by "? = ".

at the same value and the error signals are ignored, then the BLA can do two different matrix–vector multiplications concurrently. The data items for these computations are interspersed and the throughput is twice that of the original one. However, selective insertion of a duplicate computation leads to periodic application of concurrent error detection, which offers a trade off between promptness of error detection and performance [17,18]. Thus, the above fault detection scheme is both efficient (uses excess node capacity to perform the duplicate computations) and flexible (allows for trading the error detection capability to increase computational throughput).

4. Fault diagnosis

Our approach to diagnosing the faults to the level of individual processors is based on attaching ("error") tags to the data stream $\boldsymbol{x}$, in addition to those attached to the data stream $\boldsymbol{y}$. Tags on the data stream $\boldsymbol{x}$ have no effect on the functions performed by processors; they are just carriers of error messages. The functionality of each processor is still determined by the tags on the data stream $\boldsymbol{y}$. To indicate that the data items are initially error free, all tags on the data stream $\boldsymbol{x}$ are set to 0.

If one of two processors, P_p or P_{p+1}, is unable to perform the computation properly or both processors are faulty but produce different results, the tag bits will be inverted by P_p from 0 to 1. The modified tags propagate with the data items to the boundary processors at both ends of the array. Therefore, the modified tag appearing at the end of the array means that at least one of the two output data items is incorrect. Modified tag bits emerging from both directions are used to find out which processor first reported the error. The computation results can then be checked by the front-end computer to decide which processor in the disagreeing pair is faulty.

Let us resume the example started in Section 3. In addition to the change in tags on data stream $\boldsymbol{y}$, we also observe that the tags on data stream $\boldsymbol{x}$ have been changed from $\boldsymbol{T}_x^{(0)} = (000)$ to $\boldsymbol{T}_x = (001)$. The error vector associated with $\boldsymbol{y}$ is $e_y = (010)$ and the error vector for $\boldsymbol{x}$ is $\boldsymbol{e}_x = (001)$. We denote the position of the leading 1 in $\boldsymbol{e}_y$ as $\boldsymbol{y}$ and the position of the leading 1 in $\boldsymbol{e}_x$ as $\boldsymbol{x}$. From x and y, we can derive the location

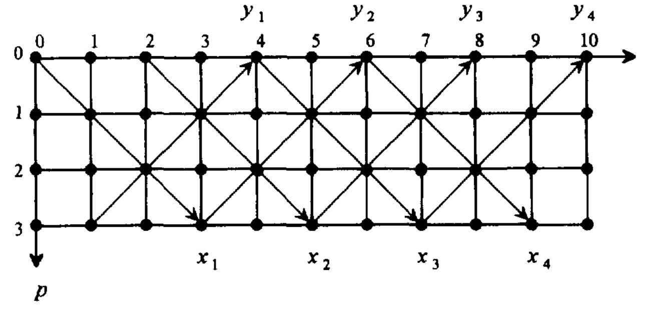


Fig. 8. The space–time diagram represented in two-dimensional Cartesian coordinates.

of the processor P_p first reporting the error and the time step t when the disagreement was detected.

The space–time diagram can be represented in two-dimensional Cartesian coordinates, as shown in Fig. 8. The data streams $\boldsymbol{x}$ and $\boldsymbol{y}$ are represented by two families of lines running in parallel. The position indices x and y become the displacements of the lines. Clearly, given x and y, t and p can be obtained ($1 \le x \le n$, $1 \le y \le m$):

$$t = x + y$$

and

$$p = y - x + p_0$$

where the constant p_0 is the index of the processor in which the first computation takes place ($1 \le p_0 \le N$, N is the total number of processors in the augmented linear array). In our example, $y = 2$, $x = 3$, $p_0 = 2$, and $N = 4$ yield $p = 1$ and $t = 5$. Hence, the error tags point to processor P_1 at time step 5, and lead us to the conclusion that processors P_1 and P_2 have generated incompatible results at time step 4.

If the error persists at the output of the faulty processor in consecutive time steps, then there will be more than a single 1 in each error vector. The following 1 provides another set of equations to resolve the ambiguity. Since the error is also detected at the next time step, the set of equations must include

$$t' = x' + y' = t + 1$$

Depending on the following 1 appearing in the error vector of data stream $\boldsymbol{x}$ or in the error vector of data stream $\boldsymbol{y}$, we can derive

$$p' = y - (x + 1) + p_0 = p - 1 \text{ if } x' = x + 1 \text{ and } y' = y + 1$$

or

$$p' = (y + 1) - x + p_0 = p + 1 \text{ if } x' = x \text{ and } y' = y + 1$$

which indicates that the faulty processor is also within the pair of processors, P_{p-1} and P_p (or P_{p+1} and P_{p+2}). In such cases, the faulty processor P_p (or P_{p+1}) is uniquely identified and the correct results can be selected at the output.

As mentioned earlier, the error detection process does not require that the entire computation be completed, but rather takes effect when the first 1 appears in the error vector. To detect an error we are only concerned with the change of tags; thus, we can add observation points for tags before they reach the boundary processors. This is illustrated in Fig. 9. The diagnostic efficiency is improved by the addition of a small number of observation points. Fault detection latency is then reduced, thereby decreasing the likelihood of a second fault disabling the system. The cost for the extra output pins is low, since a small number of tag bits are often used to carry the control and error information.

5. Conclusion

We have presented a method for introducing fault diagnosis into systolic computations. Error detection is based on dupli-

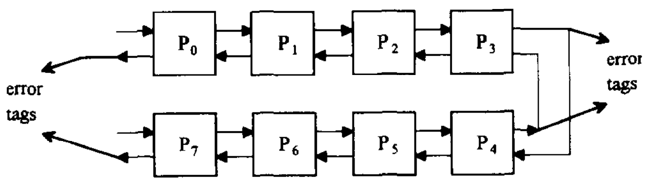


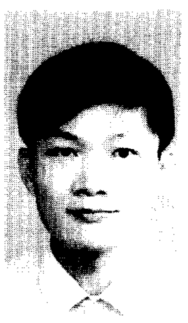
Fig. 9. Adding observation points to reduce the fault detection latency.

cation with comparison. Error reporting is done by modifying function tags attached to the data items or by introducing "error" tags where no such function tags are used. Once a processor finds an error, it changes the function or error tag value as a signal. In either case, the error indication is propagated through the processor array and is examined by the front-end computer when it emerges from a boundary processor. If the tags have remained intact, then the results are assumed correct and nothing is done; otherwise, the error syndromes obtained from the tags are used to pinpoint the fault to within a pair of processors and a particular time step.

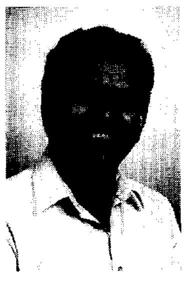
The examples we used show that for simple and regular operations of a systolic computation, such as matrix-vector multiplication and matrix multiplication, a small number of bits are sufficient to encode the control and error information (two bits in our case). The number of bits used is independent of the size of the processor array. Since the array is composed of identical processing elements and we access it only through the boundary processors to control and observe the interior processors, this data-driven approach provides scalability. Once a processor array is built, it can be expanded easily as the problem size increases.

References

[1] B.W. Johnson, Design and Analysis of Fault Tolerant Digital Systems, Addison-Wesley, Reading, MA, 1989.
[2] M. Peercy and P. Banedee, Fault-tolerant VLSI systems, Proc. IEEE, 81 (1993) 745–758.
[3] A. Majumdar, C.S. Raghavendra and M.A. Breuer, Fault tolerence in linear systolic arrays using time redundancy, IEEE Trans. Comput., 39 (1990) 269–276.
[4] M.O. Esonu, A.J. Al-Khalili, S. Hariri and D. Al-Khalili, Fault-tolerant design methodology for systolic array architectures, IEE Proc. Comput. Digital Techn., 141 (1994) 17–28.
[5] Y.-H. Choi, S.-H. Han and M. Malek, Fault diagnosis of reconfigurable systolic arrays, in Proc. IEEE Int. Conf. Computer Design, October 1984, Port Chester, Silver Spring, MD, USA: IEEE Comput. Soc. Press, pp. 451–455.
[6] R.J. Cosentino, Concurrent error correction in systolic architectures, IEEE Trans. Comput.-Aided Design, 7 (1988) 117–125.
[7] L. Li, Systolic computation with fault diagnosis, Parallel Comput., 14 (1990) 235–243.
[8] K.-H. Fluang and J.A. Abraham, Algorithm-based fault tolerance for matrix operations, IEEE Trans. Comput. C, 33 (1984) 297–311.
[9] J.-Y. Jou and J.A. Abraham, Fault-tolerant matrix arithmetic and signal processing on highly concurrent computing structures, Proc. IEEE, 72 (1986) 732–741.
[10] S.Y. Kung, VLSI Array Processors, Prentice-Hall, Englewood Cliffs, NJ, 1988.
[11] M. Gusev and D.J. Evans, YLSI processor array IPS cells, Parallel Comput., 18 (1992) 997–1007.
[12] S.K. Rao and T. Kailath, Regular iterative algorithms and their implementation on processor array, Proc. IEEE, 76 (1988) 259–269.
[13] H.T. Kung and C.E. Leiserson, Algorithms for VLSI processor arrays, in C. Mead and L. Conway (Eds.), Introduction to VLSI Systems, Addison-Wesley, Reading, MA, 1980, pp. 271–292.
[14] J. Xue and C. Lengauer, The synthesis of control signals for one-dimensional systolic arrays, Integration, VLSI J., 14 (1992) 1–32.
[15] D.-M. Kwai and B. Parhami, A data-driven control scheme for linear processor arrays, Parallel Comput., in press.
[16] D.M. Kwai and B. Parhami, Fault-tolerant processor arrays using space and time redundancy, in Proc. IEEE 2nd Int. Conf. Algorithm and Architectures for Parallel Processing, June 1996, Singapore, New York, NY, USA: IEEE, pp. 303–310.
[17] Y.-M. Wang, P.-Y. Chung and W.K. Fuchs, Scheduling for periodic concurrent error detection in processor arrays, J. Parallel Distrib. Comput., 23 (1994) 306–313.
[18] P.P. Chen, A.N. Mourad and W.K. Fuchs, Probability of correctness of processor-array outputs using periodic concurrent error detection, IEEE Trans. Reliab., 45 (1996) 285–296.



Ding-Ming Kwai received the B.S. degree from National Cheng Kung University, Tainan, Taiwan, in 1987, and the M.S. degree from the Institute of Electronics, National Chiao Tung University, Hsinchu, Taiwan, in 1989. He was with the Chung Cheng Institute of Technology, Taoyuan, Taiwan, as a reserve officer from 1989 to 1991, and with the Hualon Microelectronics Corporation, Hsinchu, Taiwan, as a design engineer from 1991 to 1993. He is currently pursuing the Ph.D. degree in Computer Engineering at the University of California, Santa Barbara, USA. His research interests include parallel processing, VLSI architecture, and fault-tolerant computing.



Behrooz Parhami received the Ph.D. degree from the University of California, Los Angeles, USA, in 1973. During his 14-year affiliation with Sharif University of Technology, Tehran, Iran, he carried out research in several areas of computer architecture, and was also instrumental in national projects in technology transfer, educational planning, curriculum development, and standardization. He was the principal founder of the Informatics Society of Iran and served as its first President and Editor-in-Chief for five years, while at the same time guiding the IEEE Iran Section through a turbulent decade. Since 1988, he has been Professor of Computer Engineering at the University of California, Santa Barbara, with research interests in computer arithmetic, parallel processing, and fault-tolerant computing. His current projects in these areas emphasize architectures and algorithms for scalable massively parallel systems, and their VLSI implementations. Dr Parhami is a Fellow of both the IEEE and the British Computer Society.