Fault-Tolerant Processor Arrays Using Space and Time Redundancy

DING-MING KWAI and BEHROOZ PARHAMI

Department of Electrical and Computer Engineering
University of California, Santa Barbara, USA

Abstract *— Spare processors in a processor array are usually idle in normal operation. They are used only after a fault is detected through periodic or on-line diagnosis and the processor array is reconfigured to include them. In this paper, we propose a design methodology in which the spare processors are used to aid with a data-driven error detection scheme. Our method consists of attaching tags to data streams, thereby allowing the data items to carry their own control information. A checking processor changes the tags when it detects a disagreement among replicated computation results. The faulty processor can then be located by error information derived from two distinct data streams. We incorporate the techniques using space and time redundancy into a fault-tolerant processor array that can provide different levels of fault tolerance according to the availability of fault-free processors. The scheme is also flexible in that it can trade error detection capability for added computational throughput.*

I. Introduction

Fault tolerance is usually achieved through redundancy applied in time or in space domain [18]. Without proper encoding of data, use of time redundancy can only detect or correct transient faults. A concurrent error detection method based on recomputing with shifted operands has been proposed to facilitate the detection of permanent faults in a computational module [16, 17]. This method can also be applied to a processor array if a proper control sequence is provided [5, 13, 20]. Another method using duplicate computations in neighboring processors eliminates the coding and decoding steps at the expense of one redundant processor [1, 2, 12, 14]. Computations have to be appro- priately rescheduled for error detection.

As shown in Fig. 1, these two methods differ only in that one inserts and interleaves the duplicated computation in time and the other in space. The original computations C_{ij} are rescheduled in the redundant schemes in such a way that the precedence relations are kept and the control flow can be pipelined. Both schemes have regular control flow, so we can incorporate them in the same processor array. They can be more efficient for on-line testing if there are idle processors at each time step, or equivalently, idle time steps in each processor.

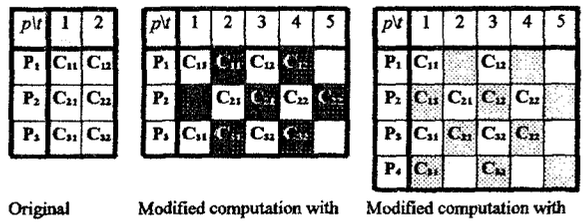

Fig. 1. Two error detection schemes using redundancy in time and in space. The column for $t = 5$ in space redundancy signifies the final comparison needed for C_{22}.

To provide the control sequencing and error indication, a data stream carrying control and error information, in the form of tags attached to data items, can be propagated through the array. Since any processor along the data flow path can be faulty, error information from a single data stream can only indicate that an error has occurred and would be inadequate for locating the faulty processor [12]. In this paper, we propose to use two data streams to carry the control and error information in order to be able to locate the faulty processor. The diagnosis scheme is used to provide fault tolerance in the array. The host initiates the reconfiguration procedure to bypass the faulty processor if it appears repetitively in the fault list.

We will introduce our method by applying it to a particular linear array. In section II, we introduce a bi-directional linear array for matrix-vector multiplication. Section III deals with the error detection methods with time redundancy and space redundancy and their associated synthesis procedures. In section IV, we extend the methods to provide fault diagnosis capability. In section V, we combine the time redundancy and space redundancy schemes into the linear array. In section VI, the extension to 2D array is given. Section VII contains our conclusions.

0-7803-3529-5/96/$5.00

II. Preliminaries

Consider the problem of multiplying an $n \times m$ matrix $A = (a_{ij})$ with an m-vector $x = [x_1\ x_2\ \ldots\ x_m]^T$ to yield an n-vector $y = [y_1\ y_2\ \ldots\ y_n]^T$. It can be written as a set of uniform recurrence equations [7]

$$y_i^{(j)} \leftarrow y_i^{(j-1)} + a_{ij} \times x_j^{(j-1)} \qquad y_i^{(0)} = 0$$

$$x_j^{(i)} \leftarrow x_j^{(i-1)} \qquad x_j^{(0)} = x_j$$

which are referred to as inner product step operations. Suppose A is a band matrix with band width (the number of diagonals) w. Fig. 2 shows the dependence graph in a two- dimensional Cartesian coordinate for the band matrix- vector multiplication when $w = 3$. The nodes in the graph represent the inner product steps and the arcs denote the movement of data items.

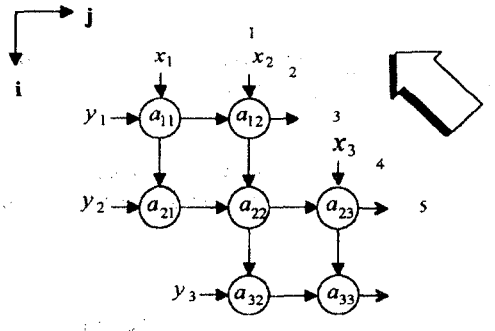

Fig. 2. Dependence graph for matrix-vector multiplication. The projection direction and scheduling (diagonal dotted lines) are also shown.

The algorithm can be implemented on a linear array of processors, each with three data streams. Data streams x and y flow in opposite directions at a velocity of one processor per time step in which consecutive data items x_j (or y_i) are interspersed with irrelevant data items. Fig. 3 shows the array with its associated data flow. The mapping from a dependence graph to a processor array is an *affine* trans- formation onto space and time domains [8, 19]. We have selected the projection direction such that the required array size is the band width w of the matrix.

We assume that the array is reconfigurable with spare processors. One possible structure is shown in Fig. 4 in which the faulty processor P_4 has been bypassed following reconfiguration. Here we are concerned only with how faulty processors can be detected and located, and not with issues related to reconfiguration [4, 13]. We assume that a suitable reconfiguration method is available [3, 6]. Spare processors are used to provide fault tolerance by space redundancy until all spares have been exhausted following reconfiguration of the array. We then can use time redundancy for error detection and fault diagnosis.

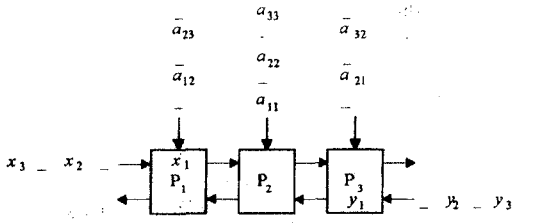

Fig. 3. Bidirectional linear array for matrix-vector multiplication. Each block is a processor.

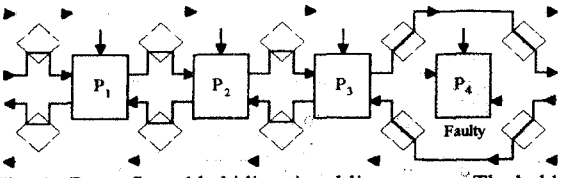

Fig. 4. Reconfigurable bidirectional linear array. The bold lines show the normal data flow. The dotted lines provide external access to the faulty processor(s).

III. Error Detection

A. Time Redundancy

This scheme works as follows. After completing an inner product step, each processor not only sends the result to its left neighbor but also *stores* the result in an internal register. At the next time step, it recomputes the inner product step (perhaps with shifted operands if a wider class of faults must be tolerated) and *compares* the result with its stored value. A mismatch indicates an error in either of the two time steps.

We actually create two events for each computation and a partial order between these two events. The first event (that requires the processor to forward and also *store* the computation result) is followed by the second event (that requires the processor to *compare* the stored value with its recomputed result). These two events correspond to two different types of nodes in the dependence graph. The dependence graph for time redundancy is derived by duplicating the original dependence graph of Fig. 2 and interleaving it in the projection direction. These duplicate nodes are shaded to denote that different instructions are executed. They are connected by arcs in the projection direction to represent the partial order between the two events in the processors (see Fig. 5).

Because duplication is done in the projection direction, the number of processors needed does not

$$e_i = \begin{cases} 1, & if \quad T_i \neq T_i^{(0)} \\ 0, & if \quad T_i = T_i^{(0)} \end{cases}$$

The error signal e_i at position index i indicates that at least one of the two y_i outputs is incorrect.

Note that if all of the tags attached to the data items are set to the same value (either 0 or 1) and the error signals are ignored, the linear array can do two matrix-vector multiplications concurrently. The data items for these two computations are interleaved and the throughput is twice that of the original one. 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 for added computational throughput).

IV. FAULT DIAGNOSIS

Our approach to diagnosing the faults to the level of individual processors is based on attaching error tags to the data stream x in addition to those attached to the data stream y. Error tags on the data stream x have no effect on the control of processors. They are just carriers of error messages. The functionality of each processor is still determined by the tags on the data stream y. To indicate that the data items are initially error free, all tags on the data stream x are set to 0.

A. Time redundancy

For the time redundancy scheme, if processor P_p can not perform the computation properly at one of two time steps, $t-1$ or t, or it is faulty but produces different results at these time steps, the tag bits will be changed by P_p to 1 at time step t. Modified tag bits emerging from both directions are used to find out which processor is faulty.

We assume that faults will manifest themselves with altered values of the output at the level of individual processors and that there is at most one faulty processor in the array. This may not be impractical for on-line diagnosis, if a certain percentage of defective processors have been rejected. Since the communication links are multiple bits, the probability of error masking in all bits is very small. We shall assume that any error at a processor propagate to all successive ones. In such cases, only the leading 1s in error vectors are important. We denote the position of the leading 1 in the error vector of data stream y as y and that of data stream x as x ($1 \le x \le m$ and $1 \le y \le n$). From x and y, we can derive the index p of the processor P_p first reporting an error and the time step t when the disagreement was detected.

The space-time diagram of the linear array can be drawn in a two-dimensional Cartesian coordinate. Fig. 7 shows the space-time diagram with time redundancy. The data streams x and y are represented by two families of lines running in parallel. The position indices x and y are related to the displacements of the lines. Pairs of values that are computed at time steps $t-1$ and t and then compared at time step t have been identified by enclosing the corresponding points of the space-time diagram in dotted boxes.

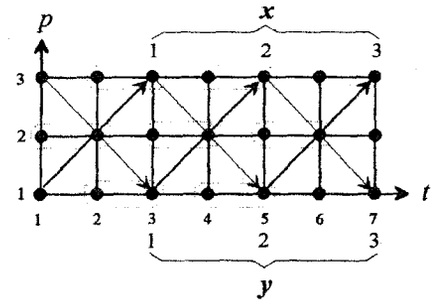

Fig. 7. Space-time diagram of the bidirectional linear array with time redundancy.

Clearly, given x and y, the time step t ($2 \le t \le n+m$) and the processor number p ($1 \le p \le w$) can be obtained:

$$p = y - x + p_0$$

$$t = x + y$$

The constant p_0 is the index of the processor where the first computation takes place ($1 \le p_0 \le w$). To verify the above equations, let us consider the case $x = y = 1$. They point to the processor where the first computation takes place ($p = p_0 = 2$) and the time step $t = 2$ when the first comparison is performed. Here we assume that the computation task starts at $t = 1$. As an example, for the computation depicted in Fig. 7 (where $p_0 = \lceil w/2 \rceil = 2$), $x = 2$ and $y = 1$ identify processor P_1 as faulty ($p = 1$), with the erroneous result (produced at $t = 2$ or $t = 3$) detected at $t = 3$. Clearly not all combinations of x and y values make sense. In a non- redundant bidirectional linear array with w processors, we must have $1 - p_0 \le y - x \le w - p_0$.

B. Space redundancy

Similarly, for the space redundancy scheme, if one of two processors, P_p or P_{p+1}, is unable to perform the computation properly at time step $t-1$, the tag bits will be changed by P_p to 1 at time step t. The modified tag bits emerging from both directions can be used to find out which processor reported the error, but it is insufficient to pinpoint the faulty processor. Fig. 8 shows the space-time diagram of the linear array with space redundancy. Pairs of values that are computed at processors P_p and P_{p+1} and then compared at processor P_p have been identified by enclosing the corresponding points of the space-time

change, although each processor will be slightly more complex. Comparison of Fig. 5 with Fig. 2 shows that the schedule is lengthened by only one time step due to the fact that most re- computations are done by nodes that would otherwise be idle.

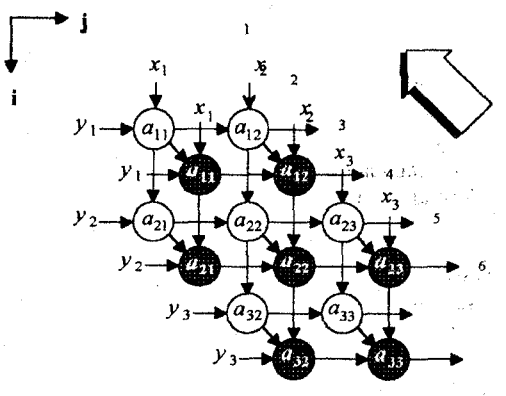

Fig. 5. Dependence graph for matrix-vector multiplication with time redundancy. The two types of nodes are interleaved in the projection direction.

B. Space redundancy

It is assumed that every processor in the non-redundant array checks its computation result with its neighbor such that a fault is detected when these two processors produce inconsistent results. If we use processor P_2 to check with P_1, P_3 to check with P_2, and so forth, we will need another processor to do the checking with P_w. The redundant processor (denoted as P_{w+1}) is appended to the right end of the linear array.

In error-detecting mode of operation, two processors compute the inner product step simultaneously, but the checking processor *stores* the result in an internal register while the checked processor sends its result to the checking processor. At the next time step, the checking processor *compares* the received data value with its stored value. A mismatch indicates an error in either of the two processors.

Here also, there exists a partial order between these two events. The first event requires the processor to store the computation result (which is the same as that in time redundancy) and the second event consists of a comparison of the stored value with the received data value. The latter is different from that in time redundancy, so we denote it as a different type of node in the dependence graph by using lighter shading. The dependence graph is derived by duplicating the original dependence graph of Fig. 2 and interleaving the two along equi-temporal lines as shown in Fig. 6.

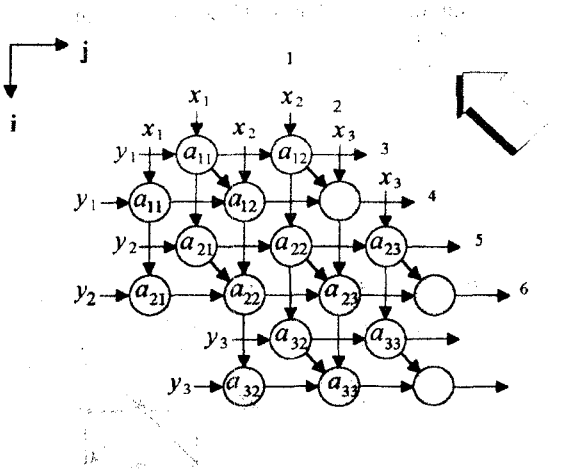

Fig. 6. Dependence graph for matrix-vector multiplication with space redundancy. The two types of nodes are interleaved along the equi-temporal lines.

C. Error indication

One may observe that the above two dependence graphs (Figs. 5 and 6) have similar characteristics: Each data stream x or y flows in a regular fashion without meeting more than one type of node. It is therefore sufficient to use only one of the data streams (either x or y) to carry instructions to each processor in either scheme. The instruction is contained in a tag attached to each data item. To encode the two types of nodes, one tag bit is enough. We assign a tag bit "0" to specify that the processor should do an additional comparison operation (the shaded nodes in Figs. 5 and 6) and "1" to specify that the processor should do an additional storing operation (unshaded nodes in Figs. 5 and 6). Every time a processor receives a data item, it first examines the tag and then decides which operation it should perform. In more general data-driven control schemes, both data streams can carry tags for multiple functions implemented on the processor array [11].

We allow the processors to modify the tags to indicate errors once they detect inconsistent results. For example, assume that we attach function tags to the data items in data stream y. The comparison operation is performed when the processor receives a tag bit 0, and thus, we are concerned only with the tags having the value 0. The processor will invert the tag bit from 0 to 1 if an error is detected; further comparison for this data item is thus prohibited. Since the error propagates along the path, each successive comparison would certainly yield inconsistent results. The modified tags propagate with the data items to the outermost processor. The corresponding tags T_1, T_2, ... , T_n are compared to the pre-assigned tags $T_1^{(0)}$, $T_2^{(0)}$, ... , $T_n^{(0)}$, and the error vector $e = (e_1, e_2, \ldots, e_n)$ is computed where

diagram in dotted boxes. Here again, t and p can be obtained ($1 \le p \le w$ and $2 \le t \le n + m$), given x and y ($1 \le x \le m$ and $1 \le y \le n$):

$$p = y - x + p_0$$

$$t = x + y$$

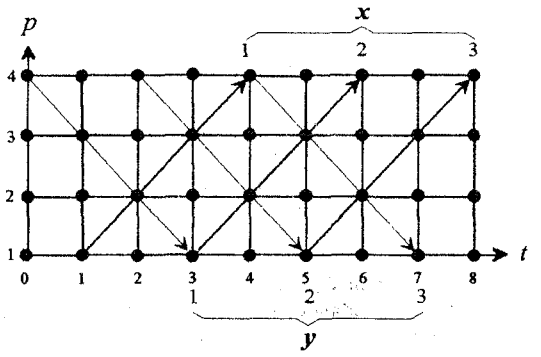

Fig. 8. Space-time diagram of the bidirectional linear array with space redundancy.

It is noteworthy that for both time and space redundancy, the representations are the same. This is due to the fact that Figs. 5 and 6 can be overlapped, with Fig. 6 having an extra column on the left edge (corresponding to $t = 0$ in Fig. 8). With time redundancy, we can pinpoint the faulty processor P_p, given x and y, but we can only locate the faulty processor to within a pair of processors, P_p and P_{p+1}, in the case of space redundancy. However, 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 x or in the error vector of data stream y, we can derive:

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

or

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

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.

V. Combined Time and Space Redundancy

Given the similarities in terms of control and fault/error handling between time and space redundancy schemes, a single array can be designed to operate in either mode under external control. To distinguish these two schemes, we augment the tags in data stream x with information about the mode of operation: tag bit "0" to specify the time redundancy scheme and "1" to specify the space redundancy scheme. Since these two schemes are independent, the mode tag attached to data stream x should not be changed when an error occurs.

The resulting tag assignment for our example is shown in Fig. 9. The error tag that is initially set to 0 is shown in parentheses. When an error occurs, the state of the data stream x will shift horizontally to the columns with error tag set to 1 (enclosed in the dotted box at the right). The mode remains unchanged. Simultaneously, the state of the data stream y will shift vertically to the row with function tag set to 1 (enclosed in the dotted box at the bottom) which changes the function to prohibit further comparison and indicates the occurrence of an error.

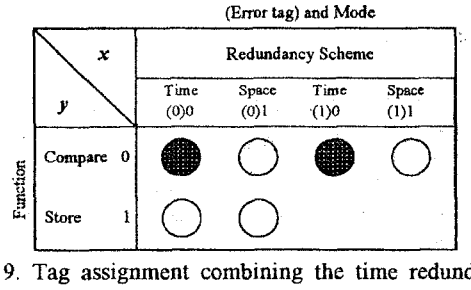

Fig. 9. Tag assignment combining the time redundancy and space redundancy schemes.

The above tag assignment does not fully utilize the entire code space. If we make time redundancy complementary to space redundancy, attaching error tags to data stream x is no longer necessary. The mode tag in data stream x that indicates whether time or space redundancy is used for error detection, can indicate the occurrence of an error by inverting its value. Because of the inversion, all of the subsequent tags will also be changed. Since we are only concerned with the leading 1s in the error vectors, this will not cause any problem in fault diagnosis. However, the error recovery capability of space redundancy, discussed in section IV-B, will be lost. Fig. 10 shows the resulting tag assignment.

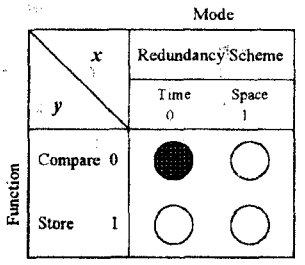

Fig. 10. Reduced tag assignment for combined time/space redundancy.

VI. Extension to 2D Arrays

The error detection and fault diagnosis method can be extended to a 2D mesh-connected array executing matrix multiplication. Similar to the matrix-vector multiplication, the matrix multiplication $Y = A \cdot X$ can be written as a set of uniform recurrence equations as follows:

$$y_{ik}^{(j)} \leftarrow y_{ik}^{(j-1)} + a_{ij} \times x_{jk}^{(i-1)}$$
$$x_{jk}^{(i)} \leftarrow x_{jk}^{(i-1)}$$

where $A = (a_{ij})$, $X = (x_{jk})$, and $Y = (y_{ik})$ are $n \times m$, $m \times l$, and $n \times l$ matrices ($1 \le i \le n$, $1 \le j \le m$, $1 \le k \le l$), respectively.

The dependence graph for matrix multiplication consists of l layers (in the k dimension), each for a matrix-vector multiplication, as that shown in Fig. 2. For error detection, duplicate computations are inserted and interleaved in the k dimension such that the dark node on the even-numbered layer sends the intermediate computation result vertically to a clear node and the clear node on the odd-numbered layer compares horizontally and vertically incoming data values (see Fig. 11). The vertical transfer in the dependence graph corresponds to saving of local values from one time step to the next in the 2D mesh. The even-numbered layers are shifted to the right (along the j direction), leading to a pair of neighboring processors performing the same inner product step. The same tag assignment (1 for "compute-and-store" and 0 for "compare- and-compute") can also apply, since data streams X and Y flow in a regular fashion without meeting more than one type of node operation.

Fig. 12 shows the 2D mesh-connected array with its data flow, where each processor contains two coefficients retrieved in alternate time steps. In this example, the tags attached to data stream Y can also be used to select the appropriate coefficient. It is easy to see that error signals derived from the modified tags attached to data streams X and Y are able to locate the column and row in which the checking processor first reported the error.

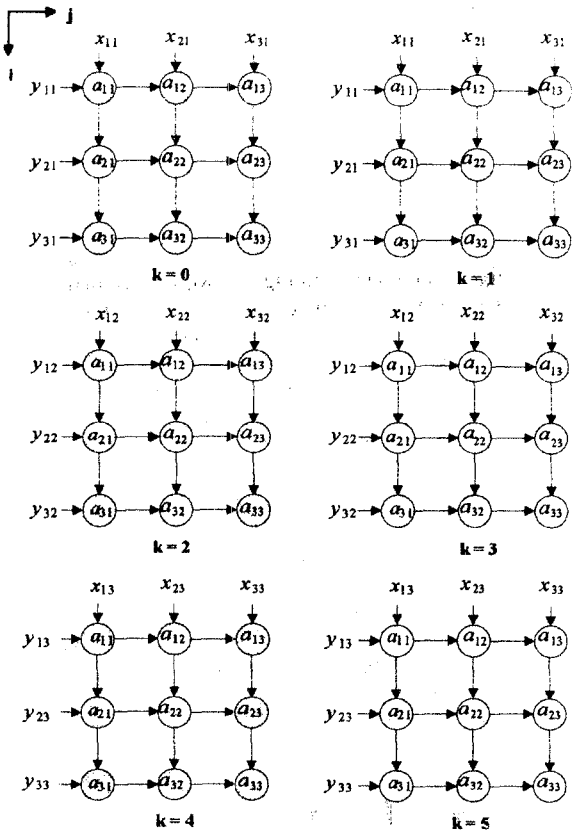

Fig. 11. Dependence graph for matrix multiplication ($n = m = l = 3$) with space redundancy. The two types of nodes are interleaved along the k dimension.

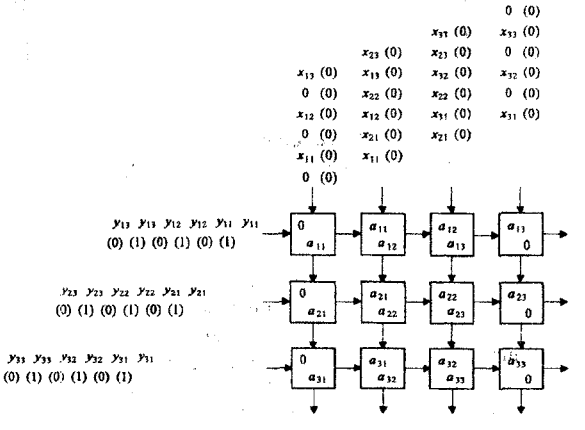

Fig. 12. 2D mesh for matrix multiplication with its associated data flow. Each processor contains two coefficients used in alternate time steps.

We next present another example. In the LU decomposition, a given matrix C is decomposed into $C = A \cdot B$, where $A = (a_{ik})$ is a lower triangular matrix and $B = (b_{kj})$ is an upper matrix [7]. The recursions involved are ($1 \le k \le n, k \le i \le n, k \le j \le n$)

$$b_{kj} \leftarrow c_{kj}^{(k-1)}$$

$$a_{ik} \leftarrow c_{ik}^{(k-1)} / c_{kk}^{(k)}$$

$$c_{ij}^{(k)} \leftarrow c_{ij}^{(k-1)} - a_{ik} b_{kj} \qquad c_{ij}^{(0)} = c_{ij}$$

The dependence graph for the LU decomposition algorithm is shown in Fig. 13 [8]. In each plane, the points that serve as the source of the row and column are marked as dark nodes. These nodes perform a different operation from the clear nodes. For simplicity, we will not discuss how to assign functional tags to distinguish these operations, but only focus on the error detection scheme. The interested reader may refer to [11] for details.

The algorithm can also be implemented on a 2D mesh- connected array, each with three data streams. Fig. 14 shows the array with its associated data flow. Data streams A and B flow to the right and to the bottom respectively, at a velocity of one processor per time step, where consecutive data items a_{ik} and b_{kj} are interspersed with irrelevant data items purposely. The time and space redundancy schemes are readily derived by duplicating the original dependence graph and interleaving it in the projection direction.

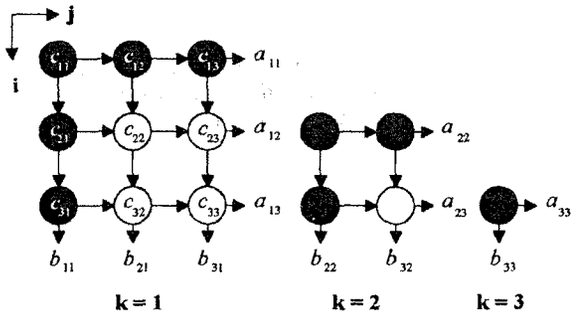

Fig. 13. Dependence graph for LU decomposition. The additional dependence lines in the k dimension are not drawn.

Similar to the 2D mesh-connected array we have shown above, these two redundancy schemes are independent. This can be seen by noting that after we insert the duplicate computation, nodes on odd-numbered planes store the intermediate results and nodes on even-numbered planes compare the results. The same tag assignment as in Figs. 9 and 10 can be used by simply changing x to A and y to B.

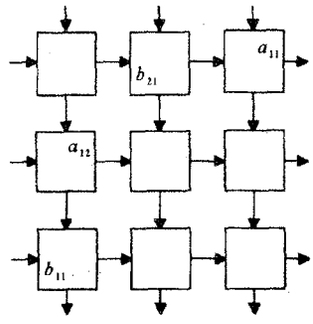

Fig. 14. 2D mesh for LU decomposition.

In order to locate the processor reporting an error and the time step when the disagreement was detected, the leading 1s in error vectors must be found for all outputs. The diagnostic process is similar to constructing an error-locating matrix in coding theory [9, 10].

VII. Conclusions

We have presented a method for introducing fault diagnosis into a processor array. Error detection is based on duplicate computation and comparison. Reporting of error is done by modifying function tags attached to the data items or by introducing "error" tags. Once a processor finds an error, it changes the function or error tag value to signal the error. In either case, the error indication is propagated through the processor array and is examined by the host 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 faulty processor and a particular time step when the error occurred.

We have shown that the space redundancy scheme can recover from an error in some cases while the time redundancy scheme allows the array to survive when the number of processors is reduced to that of a non-redundant array. Combining these two schemes thus provides different levels of fault tolerance in the same array and leads to higher resilience as well as better survival characteristics.

One way to utilize our proposed combined redundancy scheme is to use space redundancy to provide fault tolerance until all spare processors have been exhausted and then fall back to time redundancy for error detection and fault diagnosis. Whereas it would be possible to continue tolerating permanent faults with time redundancy if each processor is provided with the ability to compute and interpret the results of computations with shifted operands, the added cost of such processors is probably too

high for the level of protection provided. This added cost is probably better spent on extra spare processors.

The examples that we used show that for simple and regular operations of systolic computation, a small number of bits are sufficient to encode the control and error information (two bits in our case) and that by adding a single extra bit, both time and space redundancy schemes can be incorporated in the same processor array with no change in processing element (PE) complexity. The number of bits used is independent of the size of the processor array. Since the array is composed of identical PEs and we access it only through the boundary PEs 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 by simply using more of the same PEs and controlling them in exactly the same way.

REFERENCES

[1] Y.-H. Choi, S.-H. Han, and M. Malek, "Fault diagnosis of reconfigurable systolic arrays," *Proc. IEEE Int'l Conf. Computer Design*, Oct. 1984, pp. 451-455.

[2] R. J. Cosentino, "Concurrent error correction in systolic architectures," *IEEE Trans. Computer-Aided Design*, vol. 7, no. 1, pp. 117-125, Jan. 1988.

[3] N. J. Davis *et al.*, "Reconfiguring fault-tolerant two dimensional array architectures," *IEEE Micro*, vol. 14, no. 2, pp. 60-69, Apr. 1994.

[4] S. Dutt and J. P. Hayes, "Some practical issues in the design of fault-tolerant multiprocessors," *IEEE Trans. Computers*, vol. 41, no. 5, pp. 588-598, May 1992.

[5] M. O. Esonu *et al.*, "Design techniques for fault-tolerant systolic arrays," *J. VLSI Signal Processing*, vol. 11, no. 1/2. pp. 151-168, Oct./Nov. 1995.

[6] J. Franzen, "A design method for on-line reconfigurable array processors," *J. VLSI Signal Processing*, vol. 5, no. 1, pp. 21-35, Jan. 1993.

[7] H. T. Kung and C. E. Leiserson, "Algorithms for VLSI processor arrays," in *Introduction to VLSI System*, C. Mead and L. Conway, pp. 271-292, Addison-Wesley: Reading, MA, 1980.

[8] S. Y. Kung, *VLSI Array Processors*, Prentice-Hall: Englewood Cliffs, NJ, 1988.

[9] M. G. Karpovsky, L. B. Levitin, and F. S. Vainstein, "Diagnosis by signature analysis of test responses," *IEEE Trans. Computers*, vol. 43, no. 2, pp. 141-153, Feb. 1994.

[10] M. G. Karpovsky, T. D. Roziner and C. Moraga, "Fault detection in multiprocessor systems and array processors," *IEEE Trans. Computers*, vol. 44, no. 3, pp. 383-393, Mar. 1995.

[11] D.-M. Kwai and B. Parhami, "A data-driven control scheme for linear processor arrays," Submitted for publication.

[12] L. Li, "Systolic computation with fault diagnosis," *Parallel Computing*, vol. 14, pp. 235-243, 1990.

[13] A. Majumdar, C. S. Raghavendra, and M. A. Breuer, "Fault tolerance in linear systolic arrays using time redundancy," *IEEE Trans. Computers*, vol. 39, 269-276, 1990.

[14] H. Mori, J. Tamaki, and M. Uehara, "Stream oriented fault tolerant array," *Proc. 7th IEEE Int'l Conf. Wafer Scale Integration*, Jan. 1995, pp. 172-181.

[15] B Parhami, "Optimal placement of spare modules in a cascaded chain," *IEEE Trans. Reliability*, vol. R-26, pp. 280-282, Oct. 1977.

[16] J. H. Patel and L. Y. Fung, "Concurrent error detection in ALUs by recomputing with shifted operands," *IEEE Trans. Computers*, vol. C-31, no. 7, pp. 589-595, July 1982.

[17] ——, "Concurrent error detection in multiply and divide arrays," *IEEE Trans. Computers*, vol. C-32, no. 4, pp. 417-422, Apr. 1983.

[18] M. Peercy and P. Banerjee, "Fault-tolerant VLSI systems," *Proc. IEEE*, vol. 81, no. 5, pp. 745-758, May 1993.

[19] S. K. Rao and T. Kailath, "Regular iterative algorithms and their implementation on processor arrays," *Proc. IEEE*, vol. 76, no. 3, pp. 259-269, Mar. 1988.

[20] Y.-M. Wang, P-Y. Chung and W. K. Fuchs, "Scheduling for periodic concurrent error detection in processor arrays," *J. Parallel Distributed Computing*, vol. 23, no. 3, pp. 306-313, Dec. 1994.