

Redundancy design for a fault tolerant systolic array

J.-J. Wang
C.-W. Jen

Indexing term: Array processing

Abstract: A systematic design methodology for redundant systolic arrays is proposed. Redundancies consisting of space-shift, time-shift and space-time-shift schemes are applied successfully to detect or mask permanent faults, transient faults or both. Various redundancy designs for different utilisation efficiencies of processor elements can be obtained at the design stage by a dependent graph and its associated algebraic transformation. A customised optimal redundant systolic array design can be achieved for various performance requirements, including throughput rate, latency, average computation time, hardware cost and capabilities of fault detection and fault masking.

1 Introduction

A systolic array [1] is a computing network that is composed of many processor elements (PEs) and local interconnections between them. It maximises computation concurrences, including multiple processing and pipeline processing; it is therefore suitable for the computation-intensive problems existing frequently in image and digital-signal processing. However, a major difficulty with such high degrees of integration is that a single flaw in a PE will lead to an erroneous result and render the entire array useless. Therefore, fault-tolerance techniques must be incorporated.

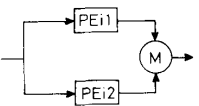
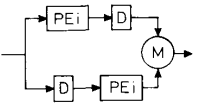
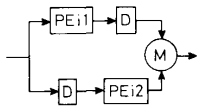
Various fault-tolerance techniques on systolic arrays, such as static redundancy and dynamic reconfiguration, have been proposed [2-13]. For conventional static redundancy, three techniques can be applied: fault detection (duplication and RESO [3]), fault masking (TMR) and algorithm-based fault tolerance [4, 5]. RESO is useful for detecting transient faults in PE, but strictly it causes time redundancy. Algorithm-based fault tolerance has the advantages of lower hardware and time overheads, but the drawbacks of algorithm-specific design and arithmetic errors, including truncations and overflows.

In general, replicated computation in a systolic array can repeat the PE processing and then make a matching or voting test, which is a recognised, effective means of concurrent error detection or masking in real-time fault tolerance applications [9, 10]. Duplication, TMR or time redundancy at PE level all belong to this class. The design objective is to maximise the fault coverages while

minimising the corresponding hardware and time overheads.

Transient faults and permanent faults both occur in real-time environments. For detecting purpose, a permanent fault can be resolved by comparing the computing results from different PEs assuming that only single fault exists in systolic array. A transient fault can be detected by comparing the results from same PE at a different time. Of course, comparison of results from different PEs executed at different times will detect both permanent and transient faults. Therefore, the replicated computation techniques working on array structure can be summarised into three operating modes: space-shift, time-shift and space-time-shift, as shown in Table 1. The enhanced RESO technique [11, 12] is one specific design example. In this paper, a systematic and more generalised redundant design approach for systolic array is provided.

Table 1: Three operating modes of the replicated computation in space-time domain

operating modes	structure	fault detected
space-shift		permanent fault
time-shift		transient fault
space-time-shift		permanent fault and transient fault

The systolic array can be characterised by the following attributes [14]: (1) synchronicity, (2) regularity and modularity, (3) spatial and temporal locality and (4) pipelinability. Regular, modular design and spatial locality will improve space-shift, while synchrony and temporal locality will benefit time-shift. Pipeline design gives the system high throughput rate, but the PEs often lie idle in some nonfull pipelining cases. Nonfull pipeline often occurs because of the synchronous systolisation or limited input/output bandwidth [15, 16]. These idle PEs can be seen as the pseudohardware to be used for redundancies. Hence the effective redundancy design in systolic arrays is a kind of space-time management problem. And, as we know, the synthesis procedure of a systolic array can be seen as a space-time transformation. Therefore it will be the most effective to consider redundancies in design stage for a systolic array.

Paper 7138E (C2, C3), first received 7th October 1988 and in revised form 25th September 1989

The authors are with the Institute of Electronics, National Chiao Tung University, 75 Po-Ai St., Hsinchu, Taiwan, Republic of China

2 Design methodology for a systolic array

In this Section, a systematic design method for a systolic array is introduced. It is a two-step procedure. The first step is to construct a dependence graph from the algorithm expression. The dependence graph approach is an effective way to maximise the parallelisms in temporal and spatial domains [14]. The second step is a space-time transformation from the dependence graph to a systolic array.

2.1 Dependence graph

The dependence graph (in short DG) is the 'unrolling' of an algorithm, which exposes the inherent data dependencies so that the concurrences can be easily extracted. The DG can be easily constructed from an indexed and localised single assignment form of an algorithm expression. The formal definition can be described as

Definition 1: A dependence graph is a directed graph composed of nodes and directed arcs. Nodes locate at some index points in n -dimensional index space, and each node associates with a function whose operands reside in incoming arcs while the computing results reside in outgoing arcs. Therefore, the directed arcs represent the data flow dependencies. The DG can be expressed as an algebraic structure

$$DG = (J^n, D), \text{ where}$$

J^n is the set of triples (j, c_j, f_j) , where j is an index point from a finite integer set Z^n , c_j is the function associated with the index point and f_j is the set of data associated with the index point j .

D is the set of triples $(d, e(d), f_e)$, where d is the data-dependent vector in n -dimensional index space and $e(d)$ the data-dependent edges or arcs in the graph along the direction specified by d . f_e is the set of data associated with $e(d)$.

2.2 Space-time transformation

Given a DG, a systolic array can be derived by a linear transformation. This transformation matrix, which maps the DG into a systolic array can be expressed as

$$T = \begin{bmatrix} W \\ S \end{bmatrix} \quad (1)$$

where the $1 \times n$ vector W , termed the time-schedule function, maps the index space into time sequence, and the $(n-1) \times n$ submatrix S , termed the space-transformation function, maps n -dimensional index space into an $(n-1)$ -dimensional systolic array. This transformation T , comprising time schedule and node assignment to array space, is sometimes called the space-time transformation.

To determine W and S , we first have to choose arbitrarily a projection direction P_d on the DG. The space-transformation function S can be obtained by the chosen P_d [5]. For correct timing scheduling, W has to obey the following conditions:

- (1) $W \cdot d_i \geq 1$ for any dependent vector d_i on the DG.
- (2) $W \cdot P_d \neq 0$.

If W and S are determined through this space-time transformation T , a systolic array can be derived. The systolic array can be abstracted by a model defined as follows:

Definition 2: A systolic array model can be expressed as a structure $SA = (I^{n-1}, L)$ where: I^{n-1} is the set of triples (i, C_i, F_i) , where i is the index point in $(n-1)$ dimension

space where PE is located. C_i and F_i are the function and the set of data streams associated with the index point i .

L is the set of triples $(l, De(l), F_l)$, where l is the physical directed links connected between processors. $De(l)$ is the delay elements on the physical directed link l . F_l is the set of data streams associated with the physical directed link l .

The PE space and physical directed link of a systolic array are therefore easily obtained by:

(1) **Node transformation:** an index point $j \in J^n$ (index in DG) is mapped by

$$T \cdot j = \begin{bmatrix} W \\ S \end{bmatrix} \cdot j = \begin{bmatrix} t(j) \\ i(j) \end{bmatrix}$$

which means that the index point j is executed in index point $i(j)$ of the corresponding I^{n-1} PE space in systolic array at time $t(j)$.

(2) **Link transformation:** a data-dependent vector d in DG is mapped to a physical directed link

$$T \cdot d = \begin{bmatrix} W \\ S \end{bmatrix} \cdot d = \begin{bmatrix} De(l) \\ l \end{bmatrix}$$

The l is a physical directed link in systolic array and $De(l)$ is the delay associated with the link l . Actually, the number of extra delay units inserted between PEs is $(De(l) - 1)$.

One important performance parameter for the designed systolic array is the pipeline period α , which is the time interval in clock units between two successive input data. It also indicates the time interval between two successive activities of a processor. By our transformation, $\alpha = W \cdot P_d$. $\alpha = 1$ means that the processor is busy in every clock and $\alpha = 2$ means the processor is activated in every other clock, i.e. busy and idle alternately.

Taking an example of band matrix-vector multiplication, DG is shown in Fig. 1a while two array designs

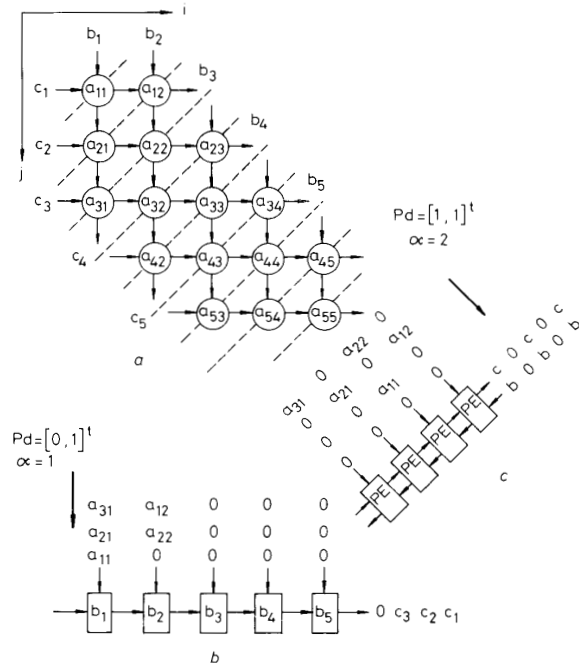


Fig. 1
a Dependence graph for band matrix-vector multiplication
b Systolic array with pipelining period $\alpha = 1$
c $\alpha = 2$

which are projected along two projection directions [0, 1] and [1, 1] are shown in Figs. 1b and c, respectively. In these designs, time schedule function W is selected as [1, 1]. For [0, 1] projection, $S = [1, 0]$, then $\alpha = 1$ is obtained. For [1, 1] diagonal projection, $S = [1, -1]$, α is 2 and so the data throughput rate and the utilisation efficiency of processor would be halved.

2.3 Pipeline period scaling and delay transfer

As described in Section 2.2, various transformations may be found by choosing different projection directions, which will result in different designs. Meanwhile, some rules may be applied on the algebraic transformation matrix to get new transformations, which will be useful for fault-tolerant systolic array design.

2.3.1 Pipeline period scaling: If a new transformation T_k obtained by the time schedule function is multiplied by a positive integer k , i.e. $W_k = kW$, this is termed pipeline period scaling. For this new transformation, not only α but also delay elements on the links have to be scaled up by a factor of k . Incidentally, the PE is idle more frequently and more delay elements are needed. Therefore, a time schedule function W_k is called a scaled time-schedule function if it can be described by $W_k = k \cdot W$ where k is a positive integer.

2.3.2 Delay transfer: Given any cutset that partitions a systolic array into two parts, we can group the edges of the cutset into two sets: inbound and outbound edges. As we know, on systolisation [13], the advancing k time units on all the outbound edges would pause k time units on the inbound edges, or vice versa. This procedure of delay transfer can be described mathematically by

$$T_c \cdot D = \left[\begin{array}{c} W + \sum h_m \cdot s_m \\ \dots \\ S \end{array} \right] \cdot D \quad \text{for } m = 1, 2, \dots, n-1$$

where s_m is any row vector of S and h_m is any integer.

The new transformation T_c is obtained by transferring $\sum h_m \cdot s_m \cdot d_i$ delay time units to each directed link l_i . Two examples follow. Fig. 2a is a systolic array design in which the corresponding transformation is described by $T \cdot D$. The first element of each column in $T \cdot D$ represents the delay element $De(l)$ in each directed link l . The other elements represent the existing direction of l . In

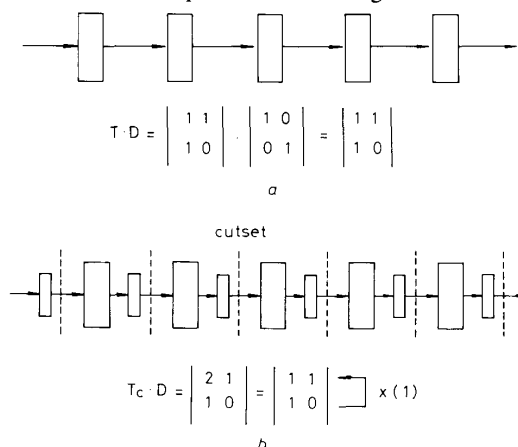


Fig. 2
a Systolic array for band matrix-vector multiplication
b Delay transfer for systolic array

Fig. 2a, there are $(De(l) - 1)$ delay elements shown in physical link because one delay element is included in PE. If we select $h_1 = 1$ and add $h_1 \cdot s_1$ to W , then one extra delay element is transferred to l_1 which constitute the cutset shown in the Fig. 2b. In Fig. 3a, it shows one assumed systolic array whose directed links construct a loop. By selecting different T_c s corresponding to different cutsets, different delay transfers between directed links are resulted, which are shown in Figs. 3b, c and d respectively.

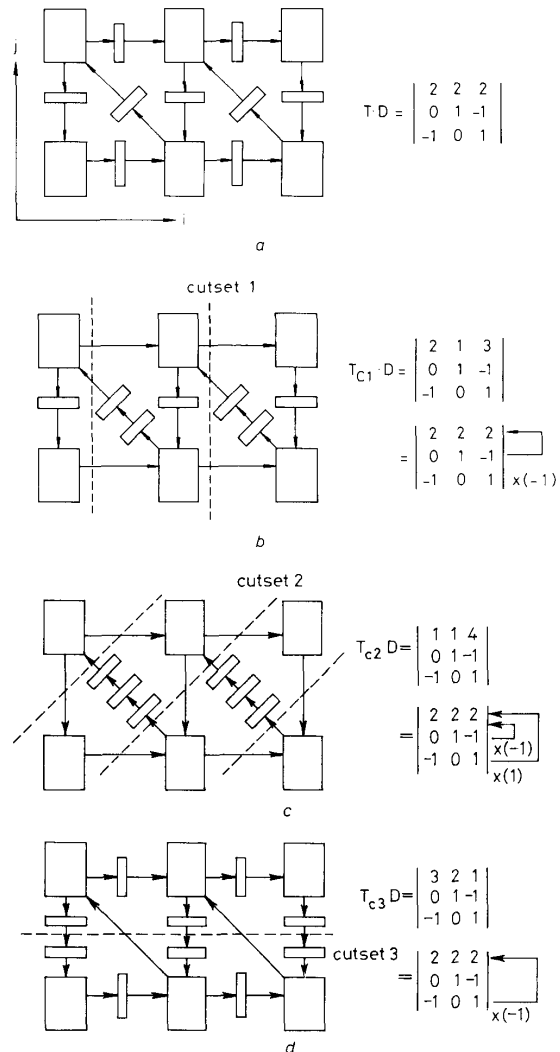


Fig. 3
a Systolic array whose directed links construct a loop
b, c, d Delay transfers for systolic array

Note that the pipelining period α does not change after delay transfer. This means that the data throughput rate remains the same but the computation latency may be changed.

3 Redundancy design

In Section 2, we proposed a systematic way to design an algorithm-specific systolic array. In practice, by this method, various architectures for an algorithm can be

explored by varying W and S , resulting in different pipeline periods, α . The pipeline period α , means that each PE activates for one clock cycle and idles for the next $\alpha - 1$ clock cycles. A systolic array is nonfull pipelining and hence is not efficient if $\alpha > 1$. Methods of increasing the utilisation of systolic arrays have been proposed in some papers. One is to execute simultaneously two or more problem instances at a systolic array so that the effective throughput rate can increase [15, 16]. Another is to perform replicated computations in systolic array to form a fault-tolerant systolic array (FTSA) [8, 14].

3.1 Design methodology for systolic array with redundant scheme

A systolic array with pipeline period α can perform an original DG (ODG) and $\alpha - 1$ (or less) redundant DGs (RDGs) concurrently. These RDGs can be executed by idle PEs at idle clock cycles. We use a general transformation T'_n to describe the relation between RDG $_n$ and ODG. The node transformation and link transformation associated with T'_n are shown as follows:

node transformation

$$T'_n \cdot j = \begin{bmatrix} W \\ S \end{bmatrix} \cdot j + k1_n \cdot \begin{bmatrix} W \\ S \end{bmatrix} \cdot d_i + k2_n \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad (2)$$

where $k1_n$ and $k2_n$ are integers and d_i is any dependent vector, but $S \cdot d_i \neq 0$.

link transformation

$$T'_n \cdot d = \begin{bmatrix} W \\ S \end{bmatrix} \cdot d \quad (3)$$

In eqn. 2, the first term on the right-hand side is the original node transformation. The second term means that RDG $_n$ is shifted $k1_n$ units from ODG along the direction d_i . Now, when two DGs are put together, there may be many nodes residing at same index point. Therefore, the result is still not correct, because one PE may need to execute two computations at a clock cycle. So we have to use the third term to split them by delaying $k2_n$ time units. Using the new transformation, a redundant systolic array (RSA) can be split from an original systolic array (OSA) by space-shift, time-shift or space-time-shift, and then both SAs can be merged and implemented into one physical systolic array. The details of these three redundant schemes are described below.

3.1.1 Space-shift scheme: If the replicated computations of different DGs are performed simultaneously by different PEs, this is a space-shift scheme. When $\alpha = 1$, there are no idle PEs or idle time cycles, so different DGs have to be executed by different systolic arrays like conventional DMR or TMR.

For $\alpha > 1$, the replicated computations associated with m different DGs ($m \leq \alpha$) can be executed simultaneously by different PEs if there are transformation T'_n for each DG $_n$, $n = 1, \dots, m$ where T'_n should have the form of eqn. 2 and satisfy the following three conditions:

- (i) values of $k1_n$ are different from each other
- (ii) the time shift $t_n = k2_n + (k1_n \cdot W \cdot d_i) = 0$, for each n
- (iii) values of f_n are different from each other, if $f_n = k2_n \text{ mod } \alpha$.

The t_n is the time shift between transformations T'_n and the original T . If t_n is zero, there is no time shift. When two DGs are put together and two f_n s have the same value, there may be many nodes at the same place. Condition (iii) is used to prevent one PE from executing two

computations at one cycle. After the transformation, every systolic array is shifted by $(k1_n \cdot S \cdot d_i)$ PE positions from original systolic array. They can be merged into one systolic array and executed simultaneously. By this scheme, the permanent fault can be detected or masked.

3.1.2 Time-shift scheme: If the replicated computations associated with different DGs are computed by the same PE at different times, this is a time-shift scheme. For m different DGs ($m \leq \alpha$), the replicated computations in different DGs can be computed by same PE at different time if there exist a transformation T'_n for each DG $_n$, $n = 1, \dots, m$ where T'_n should have the form of eqn. 2 and satisfy the following two conditions:

- (i) $k1_n = 0$
- (ii) values of f_n are different from each other, if $f_n = k2_n \text{ mod } \alpha$.

The starting computation time corresponding to each node in DG $_n$ has been delayed by $k2_n$ time units. The transient fault can be detected or masked when this scheme is used. No hardware overhead is needed. The other advantage is that it needs no extra communication, because results can be compared or voted in the same PE.

3.1.3 Space-time-shift scheme: If the replicated computations corresponding to different DGs are computed by different PEs at different times, this is a space-time-shift scheme. For the replicated computations corresponding to m different DGs, they can be executed by different PEs at different time if there is a transformation T'_n for each DG $_n$, $n = 1, \dots, m$, where T'_n should have the form of eqn. 2 and satisfy the following three conditions:

- (i) values of $k1_n$ are different from each other
- (ii) $t_n = k2_n + (k1_n \cdot W \cdot d_i)$ and t_n is different from each other
- (iii) values of f_n are different from each other, if $f_n = k2_n \text{ mod } \alpha$.

Once this scheme is used to design a FTSA, both permanent and transient faults can be detected or masked. If we have to keep spatial locality and temporal locality of systolic array characteristics, the variable $k1_n$ and $k2_n$ must be as small as possible. Therefore, communications between PEs will be simple.

We illustrate these three schemes by taking the same example of band matrix-vector multiplication. If we select $W = [1, 1]$, $Pd = [1, 1]^T$ and $S = [1, -1]$ for ODG, then the pipeline period α of systolic array is 2. If we select $k1 = 1$, $k2 = -1$ and $d_i = [1, 0]^T$ for RDG, the space-shift scheme is obtained and shown in Fig. 4a. At time 1, PE $_3$ and PE $_4$ perform the same computation. It is similar to the DMR scheme, except that we use only one systolic array instead of two. The hardware overhead is only one PE. When $k1 = 0$ and $k2 = 1$ are selected for RDG, it shows the time-shift scheme in Fig. 4b. Computation 1 is executed repeatedly by PE $_3$ at time 1 and time 2; this is similar to a time-redundancy scheme, but the time overhead is only one clock cycle. In another case, the space-time-shift scheme is obtained if we select $k1 = 1$ and $k2 = 1$ for RDG. This is shown in Fig. 4c. Computation 1 is performed by PE $_3$ at time 1 and repeated by PE $_4$ at time 3. This scheme has also been called TRIFT (time-redundancy with interleaving for fault-tolerance) [14].

One assumption made in some fault-tolerant architectures is that there is no need for roll-back to minimise the error latency. Using this assumption, the computed

results must be checked before they are sent to the next PE. This assumption constrains the applications of time-shift and space-time-shift. A result generated in time-shift or space-time-shift schemes can be checked and corrected before being passed to the next PE, if the time shift $t_n = (|k2_n + k1_n \cdot W \cdot d_i|)$ is not greater than $(|W \cdot d_c| - 1)$, which is the number of delay elements in the computational link d_c . By using delay transfer or pipeline period scaling, one can increase delay elements in the computational link so that roll-back is avoided.

Finally, the major concern of redundancy design is to minimise the hardware or time overhead. The time overhead, according to the new transformation, is minimised

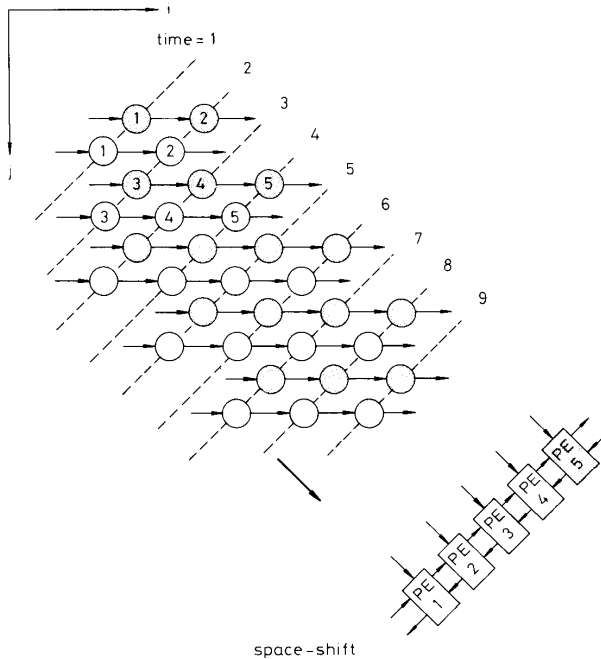


Fig. 4a Space-shift scheme for systolic array with pipelining period $\alpha = 2$

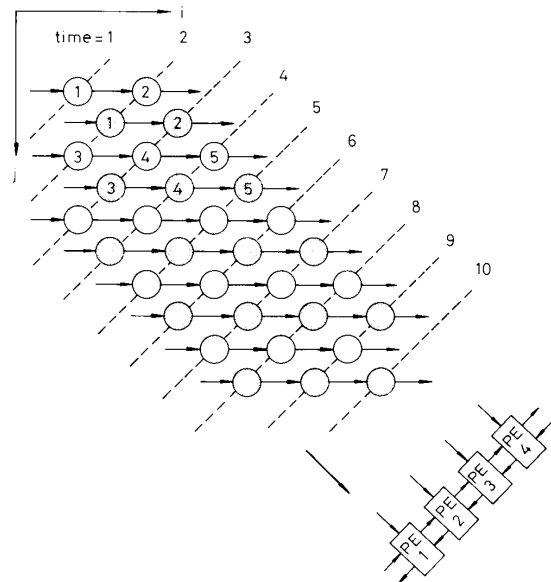


Fig. 4b Time-shift scheme for systolic array with pipelining period $\alpha = 2$

when $k1_n$ s and $k2_n$ s are selected that $\max \{t_{nm}\}$ is minimal for $n, m = 1, \dots, \alpha$, where $t_{nm} = |k2_n + k1_n \cdot W \cdot d_i - k2_m - k1_m \cdot W \cdot d_i|$ is the difference of time shift between transformations T'_n and T'_m . The hardware overhead according to the new transformation is minimal if $d_i, k1_n$ and $k2_n$ are selected such that the number of overlap PEs between SAs is maximum.

In the following Sections, concurrent error-detection and error-masking techniques will be discussed for fault-tolerant systolic arrays with different pipeline periods. For each case, three redundancy schemes (space-shift, time-shift and space-time-shift) will be applied to obtain fault-tolerant systolic arrays with different performances.

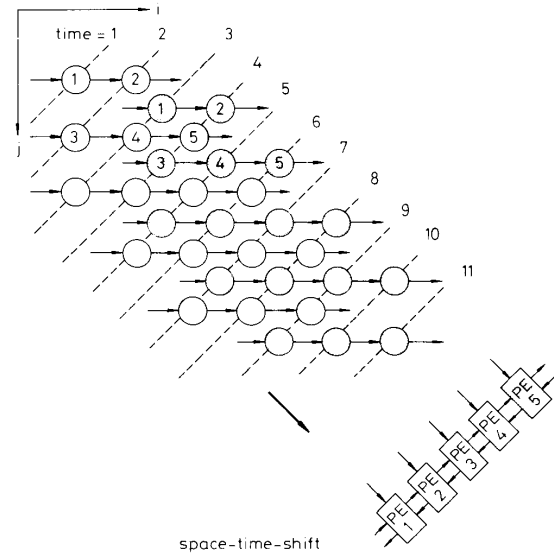


Fig. 4c Space-time-shift scheme for systolic array with pipelining period $\alpha = 2$

3.2 Fault-tolerant systolic array design with concurrent error detection

To detect an error, it is necessary to duplicate computations (one ODG and one RDG) and compare the results. The two replicated computations may be executed by two PEs simultaneously, one PE at different times or by two PEs at different times. So, three redundancy schemes can be used to design FTSA with concurrent error detection.

3.2.1 $\alpha = 1$ case: A space-shift scheme is like a conventional DMR scheme, in which a duplicated PE is tightly coupled to every PE. A time-shift scheme cannot be used, because each PE is active in every clock cycle for $\alpha = 1$. This leaves a space-time-shift scheme.

Using a DMR scheme and then shifting the RSA by $k2$ time units from OSA, a space-time-shift FTSA can be obtained. There is a roll-back problem to be solved when an error occurs. By the time that a PE of the RSA performs the replicated computation 1 and finds an error, computation 2 has already been executed in the OSA. The result of computation 1 in the OSA may be erroneous, and therefore it needs to roll-back and recompute. The problem can be solved if a delay transfer rule is applied such that there are at least $k2$ delay elements in the computational link. Therefore, results have been checked and corrected before being passed to next PE. An example of band matrix-vector multiplication is

shown in Fig. 5, which uses the design shown in Fig. 2b. One extra delay element is added to computational link l_1 so that roll-back can be avoided. After delay transfer, the latency increases but data throughput rate does not change. In fact, this scheme is the best choice when data throughput rate is the most important factor to be concerned. If $\alpha = 1$, whichever scheme is selected, the hardware overhead is 100% over original SA but performance will be unaffected.

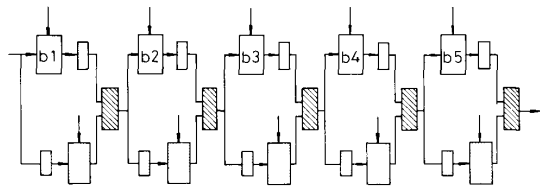
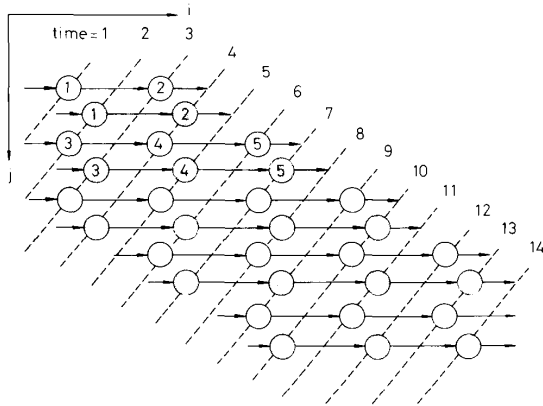


Fig. 5 Space-time-shift scheme for systolic array with pipelining period $\alpha = 1$

▨ matcher □ delay element

3.2.2 $\alpha = 2$ case: For a transformation T which results the pipeline period $\alpha \geq 2$, the time schedule function W has two alternatives. One can be described for $k \cdot W'$ (scaled W) and the other cannot be scaled. In the first case, all PEs simultaneously activate for one clock cycle and idle for the next, alternately. In the latter case, PE, activate alternate clock cycles exactly out of phase with their neighbours.

A space-shift scheme can only be used in the unscaled- W case. A space-shift FTSA is usually obtained by finding a suitable T'_n . For some algorithms, for example (Fig. 4a), only one PE overhead is needed and the resulting computing latency will not be different from that in the $\alpha = 1$ case when this scheme is applied.

A time-shift scheme is like a conventional time-redundancy scheme if the original W is a scaled time schedule function. In the band matrix-vector multiplication example, when $W = [2, 2] = 2 \cdot [1, 1]$, $S = [1, 0]$ are selected and $k_1 = 0$, $k_2 = 1$ are chosen, a time-shift FTSA is accomplished. Note that it does not need to roll back and recompute when an error occurs. In the unscaled W case, a time-shift FTSA can still be obtained. An example is shown in Fig. 4b. A drawback of this scheme is that a roll-back is necessary to correct faults and the error latency will increase. The latency is, however, smaller than the space-shift scheme in general cases.

A space-time-shift scheme can be applied to both scaled and unscaled time schedule functions. An example for unscaled W is shown in Fig. 4c. This scheme needs only one PE and k_2 time units overheads, but it can detect both transient and permanent faults.

3.2.3 $\alpha \geq 3$ case: Any systolic array with $\alpha \geq 3$ can be designed as an error-detectable FTSA with any of the three types of schemes. An error-masking FTSA which offers better fault-tolerance capability can also be obtained with only a small hardware or time overhead, which will be described later. The time and hardware overheads for different error detection schemes are summarised in Table 2.

Table 2: Time and hardware overheads for different error detection schemes

	error detection	time overhead	hardware overhead
$\alpha = 1$	space-shift	0	N_{tot}
	space-time-shift (no delay transfer)	t_{st}	N_{tot}
	space-time-shift (delay transfer)	t_d	N_{tot}
$\alpha = 2$	space-shift	0	$\frac{N_{tot}}{N_d}$
	time-shift	K_2	0
	space-time-shift	t_{st}	$\frac{N_{tot}}{N_d}$
$\alpha \geq 3$	space-shift	0	$\frac{N_{tot}}{N_d}$
	time-shift	K_2	0
	space-time-shift	t_{st}	$\frac{N_{tot}}{N_d}$

1. N_{tot} is the total number of processor elements.

2. N_d is the number of PEs along one direction.

3. t_d is the number of additional delay time units along the computational direction d_c after delay transfer.

4. $K_2 = |k_2|$.

5. $t_{st} = |k_2 + k_1 \cdot W \cdot d_c|$.

3.3 Fault-tolerant systolic array design with concurrent error masking

In the error masking approach, which is known as N -tuple modular redundancy, N copies (N odd) of a module and a majority voter are used to mask the error from failed module. At least three modules are necessary in a voting system which is typically called a triple modular redundancy (TMR). It seems that we need at least 200 percent hardware overhead for fault tolerance. In practice, it needs to put triplicate computations to the voter and then gets a correct result. The triplicate computations (ODG and 2 RDGs) may be computed in different PEs and/or at different time. Using space-shift, time-shift or space-time-shift, we may obtain a better FTSA whose performance is acceptable. For example, an error masking systolic array which corresponds to a space-shift scheme with $\alpha = 2$ systolic array has been proposed, and this hardware overhead is about 50 percent [8]. When space-shift scheme is used to a systolic array with $\alpha = 3$ it needs only a very small amount of hardware overhead for a 1-dimensional array, i.e. $O(1/N_{tot})$, where N_{tot} is total number of PEs. Surely, the time overhead increases. But in some applications, time overhead may not be over 100 percent. The kind of array structures and redundancy schemes that are chosen depend on the user's requirements for hardware and time cost. In the following, three redundancy schemes will be investigated to design FTSA with error masking for different α .

3.3.1 $\alpha = 1$ case: A space-shift scheme is like the conventional TMR scheme in which triplicated PEs are tightly coupled.

A time-shift scheme cannot be used because each PE is active in any clock cycle.

A space-time-shift FTSA is obtained by shifting RSA1 by k_2 time units and RSA2 by k_2 time units from OSA. If the results of OSA and RSA1 are not correct before being used by next PEs, the rollback problem occurs. This problem can be solved if delay transfer rule is applied such that at least $K_2 = \text{Max} \{|k_2|, |k_2|\}$ delay elements existing in computational link. No matter whatever scheme is selected, the hardware overhead is 200%.

3.3.2 $\alpha = 2$ case: In this case, the utilisation of the PE is half so that RSA1 and OSA can merge into one SA and execute RDG1 and ODG correctly. Another SA is necessary to execute RDG2.

A space-shift scheme shifts RDG1 with respect to ODG and then transforms them into a FTSA, thereafter, an extra redundant PE is tightly coupled to FTSA's PE whose sum of indexes is odd (or even). This result is similar to that in Reference 8 of which the hardware overhead is 50 percent only. Meanwhile, there may be no time overhead in some applications, for example, band matrix-vector multiplication.

There is no time-shift scheme because an extra systolic array is always necessary for $\alpha = 2$ case.

The space-time-shift used as in subsection 3.2.2 and then adding extra SA, obtains a pseudo space-time-shift scheme. It is called pseudo because some replicated computations are executed simultaneously and some are executed at different time.

3.3.3 $\alpha = 3$ case: For a given non-full pipelining systolic array with $\alpha = 3$, for example, banded matrix-matrix multiplication [1], each PE activates one clock cycle and idles the next 2 clock cycles alternatively. Using these idle resources to execute two replicated computations pipelining can be filled, the efficiency of SA increased, and therefore the time and hardware overhead will become small.

In a space-shift scheme, a space-shift FTSA is obtained by suitably shifting RDG1 and RDG2 to ODG. The band matrix multiplications is taken as an example. Fig.

6 shows the systolic array with space-shift scheme while Fig. 7 show the operations in three consecutive cycles. In the original design [1], the gray PEs are active and the other PEs are idle in each cycle. In our design, one gray PE and two redundant PEs execute the same computations to perform a TMR scheme. In this example, it needs only $2N$ extra PEs instead of $2N^2$ PEs to perform TMR scheme. The detail architecture of this array is shown in Fig. 8. The voter takes the three results to vote and broadcast the result to three multiplexers. Note that each multiplexer takes the data from three different voters at three different cycles under the control signal, cycle, respectively.

The time-shift scheme is only applied in the case which has a computational link including $\text{Max} \{|k_2|, |k_2|\}$ or more delay elements. Otherwise, the erroneous result may be passed to next PE before being corrected.

This space-time-shift scheme can also be applied by selecting suitable transformation T_n . As a summary, the time and hardware overheads for different error masking schemes are listed in Table 3.

Table 3: Time and hardware overheads for different error masking schemes

error masking	time overhead	hardware overhead
$\alpha = 1$ space-shift	0	$2N_{tot}$
space-time-shift (no delay transfer)	t_{st}	$2N_{tot}$
space-time-shift (delay transfer)	t_d	$2N_{tot}$
$\alpha = 2$ space-shift	0	$\frac{N_{tot}}{N_d} + N_{tot}$
time-shift	X	X
space-time-shift	t_{st}	N_{tot}
$\alpha \geq 3$ space-shift	0	$\frac{2N_{tot}}{N_d}$
time-shift	K_2	0
space-time-shift	t_{st}	$\frac{N_{tot}}{N_d}$

- N_{tot} is the total number of processor elements.
- N_d is the number of PEs along one direction.
- t_d is the number of additional delay time units along the computational direction d_c after delay transfer.
- $K_2 = \text{Max} \{|k_2|, |k_2|, |k_2 - k_2|\}$.
- $t_{st} = \text{Max} \{t_1, t_2, t_{12}\}$, where $t_1 = |k_2 + k_1 \cdot W \cdot d_1|$, $t_2 = |k_2 + k_1 \cdot W \cdot d_2|$ and $t_{12} = |k_2 + k_1 \cdot W \cdot d_1 - k_2 - k_1 \cdot W \cdot d_2|$.

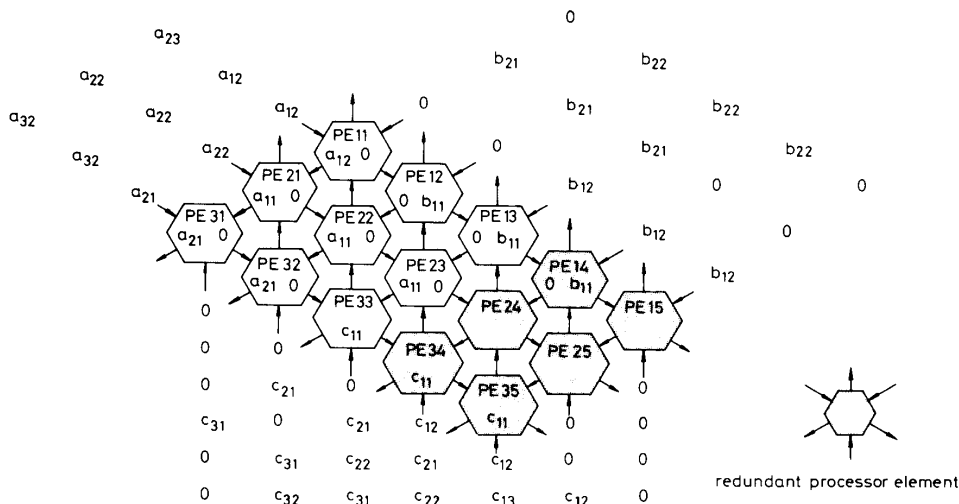


Fig. 6 Systolic array for band matrix multiplications with space-shift scheme

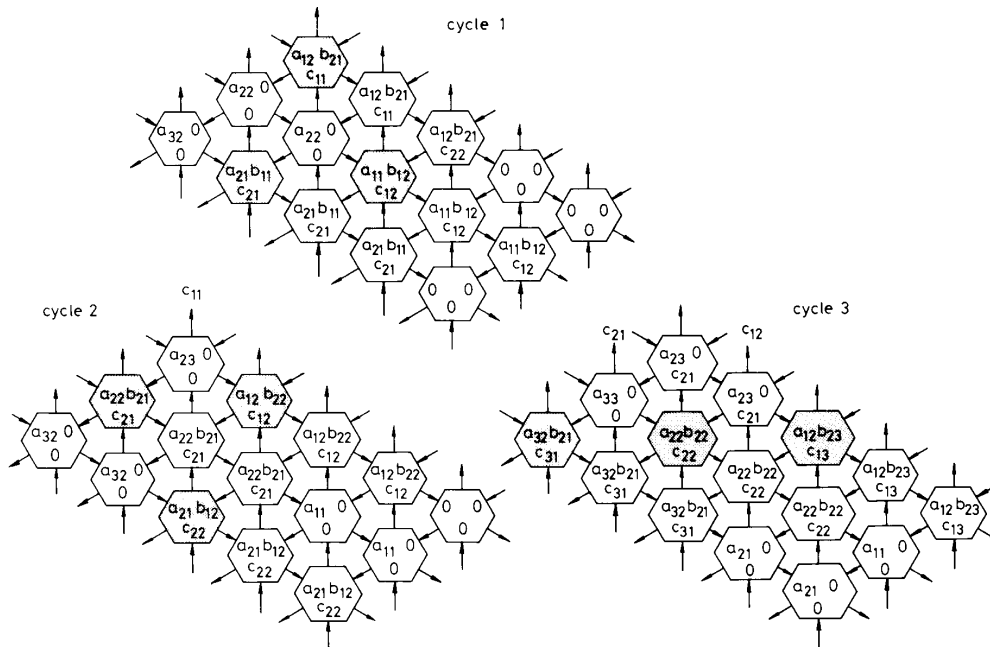


Fig. 7 Three operation cycles for band matrix multiplications

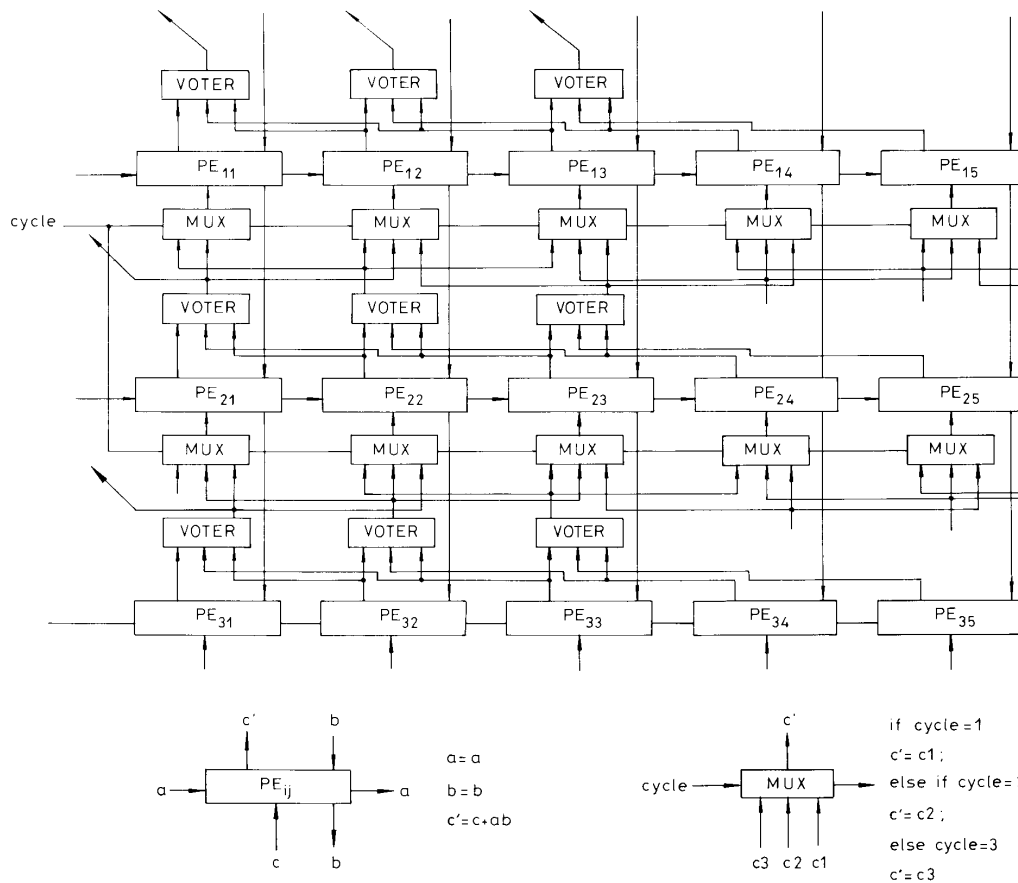


Fig. 8 Detailed architecture of systolic array for band matrix multiplications

4 Conclusions

A systematic design methodology for a redundant systolic array has been proposed. Redundancy schemes which consist of *i*) a space-shift, *ii*) a time-shift and *iii*) a space-time-shift schemes can be applied to fault tolerant systolic array design in order to detect (or mask) permanent fault, transient fault or both. By this design method, various redundancy designs for different utilisation efficiency, α , of PE in systolic array can result. According to the performance requirements including throughput rate, latency, block pipeline period, capability of fault detection (or masking) and hardware cost, a customised optimal redundant systolic array design can be achieved.

5 Acknowledgment

This work was supported by the National Science Council, Taiwan ROC, under grant NSC 77-0404-E009-04.

6 References

- 1 KUNG, H.T.: 'Why systolic architectures?', *Computer*, 1982, **15**, 1, pp. 37-46
- 2 FORTES, J.A.B., and RAGHAVENDRA, C.S.: 'Gracefully Degradable Processor Arrays', *IEEE Trans.*, 1985, **C-34**, 11, pp. 1033-1044
- 3 KOREN, I.: 'A Reconfigurable and Fault-Tolerant VLSI Multiprocessor Array'. Proc. 8th Int. Symp. Computer Architecture, 1981, pp. 425-442
- 4 HUANG, K.H., and ABRAHAM, J.A.: 'Algorithm-Based Fault Tolerance for Matrix Operations', *IEEE Trans.*, 1984, **C-33**, 6, pp. 518-528
- 5 LIU, C.M., and JEN, C.W.: 'On the Design of Algorithm-Based Fault-Tolerant VLSI Array Processor', *IEE Proc. E*, 1989, **136**, (6), pp. 539-547
- 6 SAMI, M.G., and STEFANELLI, R.: 'Reconfigurable Architecture for VLSI Processing Array'. National Computer Conf., 1983, pp. 565-577
- 7 ROSENBERG, A.L.: 'The Diogenes Approach to Testable Fault-Tolerant Arrays of Processors', *IEEE Trans.*, 1983, **C-32**, 10, pp. 902-910
- 8 KIM, J.H., and REDDY, S.M.: 'A Fault-Tolerant Systolic Array Using TMR method'. Proc. IEEE Internat. Conference Computer Design: VLSI in Computers, 1985, pp. 769-773
- 9 PATEL, J.H., and FUNG, L.Y.: 'Concurrent Error Detection in ALUs by Recomputing with Shifted Operands'. *IEEE Trans.*, 1982, **C-31**, 7, pp. 589-595
- 10 JEN, C.W., KUNG, S.Y., and CHANG, C.W.: 'Fault-Tolerant Design for VLSI Array Processors'. Proc. of Real-Time System Symp., 1987, pp. 46-64
- 11 CHAN, S.W., LEUNG, S.S., and WEY, C.L.: 'Systematic Design Strategy for Concurrent Error Diagnosable Iterative Logic Arrays', *Proc. IEE Pt. E*, **135**, 2, 1988, pp. 87-94
- 12 CHAN, S.W., and WEY, C.L.: 'The Design of Concurrent Error Diagnosable Systolic Arrays for Band Matrix Multiplications'. *Proc. IEEE Trans. on CAD*, 1988, pp. 21-37
- 13 KUNG, H.T., and LAM, M.S.: 'Wafer-Scale Integration and Two-Level Pipelined Implementations of Systolic Arrays', *J. Parallel and Distributed Computing*, 1984, pp. 32-63
- 14 KUNG, S.Y.: 'VLSI Array Processors', Prentice-Hall, 1988
- 15 NAVARRO, J.J., LLABERIA, J.M., and VALERO, M.: 'Solving Matrix Problems with no Size Restriction on a Systolic Array Processor'. Proc. of Internat. Conf. on Parallel Processing, 1986, pp. 676-683
- 16 NAVARRO, J.J., LLABERIA, J.M., and VALERO, M.: 'Computing Size-Independent Matrix Problems on Systolic Array Processors'. Proc. 13th Internat. Symp. Computer Architecture, 1986, pp. 271-278