Design of algorithm-based fault-tolerant VLSI array processor

C.-M. Liu
C.-W. Jen

Indexing terms: Very large scale integration, Array processing, Algorithms

Abstract: In the paper a systematic design methodology which maps a matrix arithmetic algorithm to a fault-tolerant array processor with different topologies and dimensions is presented. The design issues to be addressed in the method are: (*a*) how to derive a VLSI array with different topologies and dimensions from the algorithm; (*b*) how to distribute the data processing to the PEs so that a faulty PE will result in limited erroneous data on which the checking scheme is valid. Two examples, matrix multiplication and Givens reduction, are used to illustrate this design method.

1 Introduction

Fast matrix arithmetic is highly demanded in signal processing, image processing and scientific applications. The evolution in VLSI technology allows this matrix arithmetic to be implemented on VLSI chip using multiple, regularly connected processing elements (PEs), which has been named a systolic array [1]. It exploits the great potential concurrency of pipelining and multiprocessing so that it is suitable to conquer the intensive computation problems. A major difficulty with such a high degree of integration is that a single flaw on a chip will often result a PE to be erroneous and then render an entire computing system useless. It is, therefore, desirable to have a system which not only achieves high performance but which also can tolerate physical failures in the system, so that the correct result is still produced.

Recently, several researchers [2–14] have presented their results on error diagnosis, correction and reconfiguration in fault-tolerant array processors. For concurrent error detection and correction, algorithm-based fault tolerance has been shown to be efficient in matrix arithmetic [4–7]. The scheme is to encode the input data at the system level in the form of error detection or correction code (e.g. Weighted Checksum Code, WCC). Erroneous data can be detected according to the data redundancy in the coding scheme. The finite coding distance dominates the number of error to be tolerated. To design a fault-tolerant array with the scheme, the major design consideration is to avoid the error diffusion which will cause too many erroneous data to preserve the finite-distance property even by a faulty PE. How to design an array to avoid this kind of error diffusion is one of the issues to be addressed.

In the last few years, several systematic design methods have been proposed for the systolic array [15–19]. All these methods are restricted to mapping an n-dimensional algorithm representation into an $(n-1)$-dimensional array processor. How to break the restriction of dimension and blend with the centre of the WCC is another issue of this paper.

Paper 6906E (C2), first received 15th April 1988 and in revised form 28th February 1989

The authors are at the Institute of Electronics, Chiao Tung University, 75 Po-Ai St., Hsinchu, Taiwan

2 Design methodology for VLSI array

A dependence graph is the unrolling of an algorithm exposing the inherent data dependencies and so the concurrency can be easily extracted. The DG can be constructed from a localised, indexed, single assignment form called Uniform Recurrence Equations (UREs) [18, 20]; in the literature there have been different systematic design methods [15–19]. However, their common restriction of mapping an n-dimensional DG to an $(n-1)$-dimensional systolic array has limited the design flexibility, especially when the hardware cost is an considering factor. The methodology presented in this Section breaks the restriction of dimensions, and it adopts a mathematical form to provide its generality. The extension of this method to the design of the fault-tolerant array processor will be presented in Section 3.

Formally, a dependence graph is a finite directed graph consisting of nodes and directed arcs. Nodes locate at some index points in n-dimensional index space and each node corresponds to a computation whose operands reside in incoming arcs whereas results reside in outgoing arcs, so the directed arcs represent the data flow dependencies.

The DG can be expressed as a structure of $(\boldsymbol{J}^n, \boldsymbol{D})$ where $\boldsymbol{J}^n$ is the set of triples (j, c, f), where j is an index from a finite integer set $\boldsymbol{Z}^n$; c and f are the computation and the set of data stream associated with the index. $\boldsymbol{D}$ is the set of couples $(\boldsymbol{d}, f)$, where $\boldsymbol{d}$ is the dependent vector which is a displacement associated with data stream f from one node to another.

Given a localised, indexed and uniform DG [18, 20], a VLSI array processor with topologies, dimensions and timing schedule can be derived from a linear transformation. The linear transformation matrix which maps the DG to the array processor can be expressed as

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

where W and S are defined as

$W: J^n \rightarrow J^1$ timing schedule function

$S: J^n \rightarrow J^{n-k}$ space transformation matrix

S maps a DG with dimension n to the PE space with dimension $n-k$, and W maps the DG to time sequence with dimension one. If the value of k is greater than one, local memory is required in each PE. The size of the memory may be determined by the number of jobs to be assigned. This Section deals with the derivation of the linear transformation, and will bring together the derivation with the concepts of the WCC in next Section.

The transformation matrix can be determined from the following five aspects:

2.1 Projection directions

Given an n-dimensional DG, the dimension of the array processor is first decided. If the dimension of the array processor is $(n-k)$, there will be k projection directions to be selected in the consideration of the geometry mapping. Basically, the projection directions of a DG on the index space may be arbitrarily chosen. However, to achieve fault tolerance some restrictions for the projection are necessary, which will be discussed later in Section 3.

Given the projection directions $Pd_1, Pd_2, \ldots, Pd_k$, these projection directions are orthogonalised so that they are represented with mutual independent and orthogonal vectors. This process will provide the ease of developing the space transformation and timing schedule. The orthogonalised projection vectors $v_1, v_2, \ldots, v_k$ can be constructed according to the Gram–Schmidt orthogonalisation [21].

$$v_1 = Pd_1 \tag{2}$$

$$v_2 = Pd_2 - \frac{v_1^t Pd_2}{v_1^t v_1} v_1 \tag{3}$$

$$\vdots$$

$$v_k = Pd_k - \frac{v_1^t Pd_k}{v_1^t v_1} v_1 - \cdots - \frac{v_{k-1}^t Pd_k}{v_{k-1}^t v_{k-1}} v_{k-1} \tag{4}$$

Furthermore, these projection vectors should be multiplied by some real number to meet the requirements: (i) all elements of the vector are integer, and (ii) GCD $(v_{i1} v_{i2}, \ldots, v_{in}) = 1$, for $v_i = [v_{i1} v_{i2}, \ldots, v_{in}]^t$.

2.2 Space transformation matrix

The space transformation matrix is a linear transformation to distribute nodes in the DG to the processor elements PEs. Once the projection vectors are known, the space transformation S_t is first derived by the following Theorem, and then the final form S should be extracted from the row space of S_t. This procedure will be illustrated by examples in Section 4.

Theorem 1: Given the projection directions, the space transformation can be constructed as S_t, where

$$S_t = \left[I - \frac{v_1 v_1^t}{v_1^t v_1} - \frac{v_2 v_2^t}{v_2^t v_2} - \cdots - \frac{v_k v_k^t}{v_k^t v_k} \right] \tag{5}$$

where v_i is the projection vector as defined above.

Proof: Given a projection direction Pd, a vector P in space can be projected to P' from the Gram–Schmidt orthogonalisation:

$$P' = P - \frac{Pd^t P}{Pd^t Pd} Pd = P - Pd \frac{Pd^t P}{Pd^t Pd} = S' P$$

where

$$S' = \left[I - \frac{PdPd^t}{Pd^t Pd} \right]$$

If there are multiple projection directions, S_t can then be constructed as

$$S_t = \left[I - \frac{v_1 v_1^t}{v_1^t v_1} \right] \left[I - \frac{v_2 v_2^t}{v_2^t v_2} \right] \cdots \left[I - \frac{v_k v_k^t}{v_k^t v_k} \right]$$

Since projection directions are mutually orthogonal

$$S_t = \left[I - \frac{v_1 v_1^t}{v_1^t v_1} - \frac{v_2 v_2^t}{v_2^t v_2} - \cdots - \frac{v_k v_k^t}{v_k^t v_k} \right]$$

Since the rank of the matrix S_t is $(n-k)$, the row space can be represented by $(n-k)$ independent vectors. The selection of representative $(n-k)$ vectors for the row space directly relates to the geometry of the array processor, but will not change which set of nodes to be executed with the same PE.

2.3 Timing schedule W

The timing schedule W is a linear schedule to assign the computation time of the nodes in the DG to the array processor with maximum parallelism and concurrency. A good linear schedule is then to maximise the global efficiency of the PE in the array processor. This efficiency is inversely proportional to the computation time of an algorithm, so our optimal timing schedule is derived from this consideration. Furthermore, a representation of nodes to be distributed to a PE should be given in deriving the timing schedule function. If the orientation of these finite node spaces can be globally and closely represented by IS, a good timing schedule function can then be derived.

$$IS = a_1 v_1 + a_2 v_2 + \cdots + a_k v_k \tag{6}$$

where $|a_i| \leqslant M_i$ is the maximum node number along vector v_i in all these finite spaces. Here 'globally, closely represented' is stated because the timing schedule function is a linear scheduling. When the shape of node space to be distributed to a PE is not rectangular shape or is sparse to fill the rectangular bulk as represented by IS, they will introduce some null time and the efficiency of the PEs will be decreased. This is the drawback of a linear schedule. This phenomenon can be seen and avoided if a process called 'data compaction' is used, which will be described in Section 4.2. Here, the linear timing schedule can be obtained by the following Theorem.

Theorem 2: Given a DG, the orientation of these finite node spaces can be globally and closely represented by

$$IS = a_1 v_1 + a_2 v_2 + \cdots + a_k v_k$$

where $|a_i| \leqslant M_i$ is the maximum node number along vector v_i in all these finite spaces. The timing schedule function $W = [w_1 w_2, \ldots, w_n]$ should obey the following two constraints

(a) $Wd \geqslant 1$ where $d \in D$ (7)

(b) $|Wv_j| \geqslant (\sum_i |M_i Wv_i|) + 1$ for a selected j (8)

where $i \equiv \{(1, 2, \ldots, k) - (j)\}$ and $j \in \{1, 2, \ldots, k\}$ and the optimal solution is found to minimise the following cost function:

$$\textit{Computation time} = \max\,[\boldsymbol{W}(j_2 - j_1)] + 1 \quad \text{for every index } j_1, j_2 \in \boldsymbol{J}^n \tag{9}$$

Proof: (i) Since the dependent vector d_i is a directed arc from j_1 to j_2 in the DG, j_2 should be executed after j_1, so $\boldsymbol{W}j_2 > \boldsymbol{W}j_1$ and $\boldsymbol{W}(j_2 - j_1) > 0$; then $\boldsymbol{W}d_i > 0$ or $\boldsymbol{W}d_i \geqslant 1$. (ii) Since every node in the space spanned by the projection directions is projected to the same PE, the node space spanned by the projection vectors is represented by

$$\boldsymbol{IS} = a_1\boldsymbol{v}_1 + a_2\boldsymbol{v}_2 + \cdots + a_{n-k}\boldsymbol{v}_{n-k}$$

where $|a_i| \leqslant M_i$. These nodes should not be executed at the same time in the same PE. This means that $\boldsymbol{WIS} \neq 0$, i.e. $|\boldsymbol{WIS}| > 0$, or $|\boldsymbol{WIS}| \geqslant 1$.

These inequalities combined with the former $\boldsymbol{W}d_i \geqslant 1$ constitute a space called the feasible set [21]. To say it more directly, a feasible set is composed of the solutions of a family of linear inequalities. Because the cost function is a linear function of $\boldsymbol{W}$, the optimum solution occurs at corners of the feasible set. Since the minimum grid of the DG in each cartesian co-ordinate is 1, the corners of the feasible set can be found from further derivation of the linear inequalities.

$$|\boldsymbol{W}\boldsymbol{v}_j| \geqslant \left(\sum_i |M_i \boldsymbol{W}\boldsymbol{v}_i|\right) + 1$$

There will be k inequalities to be constructed from k projections. Under the constraints of (i) and (ii), we can find corners of the feasible set to minimise the computation time.

2.4 Relation between timing schedule and space transformation matrix

For performance considerations, the choice of projection directions is a decisive factor for the number of PEs in an array processor, and the timing schedule is a decisive factor for the computation time of an algorithm. In deriving the timing schedule and space transformation matrix, the projection directions are determined first, and this restricts the feasible solution of timing schedule vector as shown in Theorem 2. If the timing schedule is derived without considerations of projection direction, a truly optimum one can be obtained. A designer can check if there is any conflict between these two schedulings. If this conflict does happen, a trade-off between the computation time and the number of PEs should be made.

2.5 Transformation

Once the $\boldsymbol{W}$ and $\boldsymbol{S}$ are determined, the transformation matrix is completed and then the PE index and link are easily obtained by:

(*a*) *Node transformation:* An index $j \in \boldsymbol{J}^n$ (index in the DG) is mapped by

$$\boldsymbol{T}j = [\boldsymbol{WS}]^t j = [t(j)i(j)]^t \tag{10}$$

This means the index j in DG is executed in index i of the corresponding $\boldsymbol{J}^{n-k}$ in array the processor at time $t + k$; $k = \min \boldsymbol{W}j$, for every $j \in \boldsymbol{J}^n$.

(*b*) *Link transformation:* A dependent vector d in the DG is mapped by

$$\boldsymbol{T}d = [\boldsymbol{WS}]^t d = [De(l)l]^t \tag{11}$$

l is a physical directed communication link in the array processor and $De(l)$ is the delay number associated with the link.

Through this transformation, an array processor with execution time is successfully designed without the need for retiming [22]. By choosing a different projection with multiplicities, all the various array designs for a specific algorithm are explored.

3 Design of fault-tolerant array processor

In this Section, the constraints for projection are given so that computation tasks in the DG can be appropriately distributed to PEs which accomplish the concurrent error detection or correction.

3.1 Weighted checksum code

The weighted checksum code (WCC) [6] has been adopted in matrix arithmetic operations for algorithm-based fault tolerance. By the encoding scheme and the preserving property of the matrix arithmetic, the resultant erroneous data produced by the faulty PE can be detected or corrected. In this Subsection, the WCC and the fault model is outlined briefly. The reader can refer to Reference 6 for a detailed description.

Fault model: A module-level fault [4] is assumed in algorithm-based fault tolerance. A module, i.e. the PE here, is allowed to produce any arbitrary logical errors under physical failures mechanisms. This is quite general since it does not assume any technology-dependent fault model. Without loss of generality, a single module error is assumed in this paper. Also, the transmitted links between connected PEs are assumed to be fault-free.

Encoding scheme: A WCC vector with distance three can be expressed as

$$\boldsymbol{A}_c^{\mathrm{T}} = [a_1 \quad a_2 \quad \cdots \quad a_n \quad \mathrm{WCS1} \quad \mathrm{WCS2}] \tag{12}$$

where a_i is a data element in a vector, the superscript T stands for transpose, and WCS1 and WCS2 can be chosen as:

$$\mathrm{WCS1} = [a_1 \quad a_2 \quad \cdots \quad a_n] \cdot [1 \quad 1 \quad \cdots \quad 1] \tag{13}$$

$$\mathrm{WCS2} = [a_1 \quad a_2 \quad \cdots \quad a_n] \cdot [2^0 \quad 2^1 \quad \cdots \quad 2^{n-1}] \tag{14}$$

where $\cdot$ denotes the dot product operation.

It has been proved that the checksum property can be preserved through some basic matrix operation [6]. Based on the basic encoding vector, a matrix can be encoded as either row encoded matrix, column encoded matrix or full encoded matrix according to different matrix arithmetic algorithms.

The WCC check matrix

$$\boldsymbol{H} = \begin{bmatrix} 1 & 1 & \cdots & 1 & -1 & 0 \\ 2^0 & 2^1 & \cdots & 2^{n-1} & 0 & -1 \end{bmatrix} \tag{15}$$

is used to get the syndromes and then the detection or correction procedure can be completed.

3.2 Projection for fault tolerance

Owing to the fixed code distance constraint of three as shown in eqn. 12, there is only one erroneous datum allowed in a WCC vector so that the correction can be accomplished successfully. For appropriately distributing computation tasks (i.e. nodes) to the PEs in the foregoing

systematic design, the projection directions, which assign tasks to PEs so that a single fault in the PEs affects only one datum in a WCC vector, have to be limited. Before proceeding to find the suitable projection directions, the following terms have to be defined first.

Definition 1: Transmission arc
A dependent arc of the DG on which a data stream is just transmitted, not changed iteratively by nodes in the DG.

Definition 2: Iterated arc
A dependent arc of the DG on which data is changed iteratively.

Definition 3: INS (Iterative Node Set)
A set of nodes which is connected by iterated arcs in the DG.

Definition 4: IPES (Iterative PE Set)
A set of PEs which is connected by iterated physical links in an array processor.

Definition 5: Check space
A space consisting of index nodes and their associated data stream f on the arcs, so that f can be checked by same WCC vector, is called a check space.

The check space orientation can be represented by two linear independent vectors. One of the vectors can be chosen as an iterative arc k of the INSs, the other vector (called the check vector) can be chosen from the connection vector of any two nodes of the space. In the example of matrix–matrix multiplication described in a later Section, the check vector can be selected as $[i\ j\ k]^T = [1\ 0\ 0]^T$. The iterated arc is $\boldsymbol{k} = [0\ 0\ 1]^T$. The check space is shown shaded in Fig. 1.

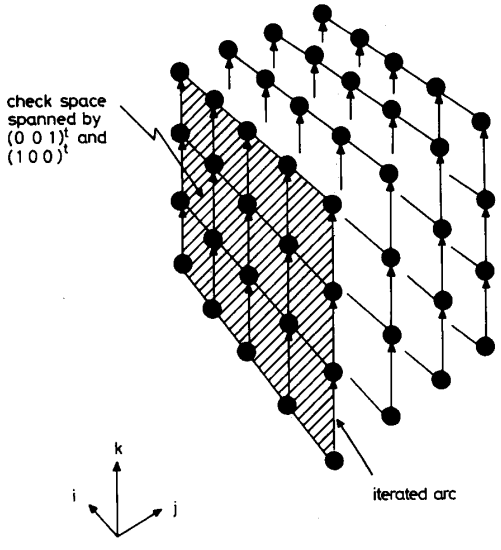

Fig. 1 *Check space of matrix–matrix multiplication*

Given the terminologies above, the theorems to restrict projection directions are then developed. These theorems are developed in the situations that input data of an algorithm are coded in WCC and the property of WCC would be retained in the computation in the fault-free condition.

Lemma 1: Given a DG and a VLSI array derived from projection $\boldsymbol{Pd}$, the single fault in the PEs affects only one datum in a WCC vector if the mapping of INS in the DG to the IPES in array processor is one-to-one.

Proof: From the definition of the INS (or IPES), there is no iterated arc between the INS (or IPES). Our fault model has assumed that the transmission arc is fault-free. Any error in an INS can be detected or corrected by the data redundancy of another INS in the same WCC vector. If the distribution of the INS into the IPES is in a one-to-one manner, any fault in a PE can affect only one INES in a WCC vector.

Lemma 2: Given a DG and a VLSI array derived from projection $\boldsymbol{Pd}$, a single fault in the PEs affects only one datum in a WCC vector if nodes in a different check space can be assigned to the same PE.

Proof: Although a fault in a PE affects data in a different WCC vector, it affects only one datum in the same WCC vector. So, a single fault in the PEs affects only one datum in a WCC vector.

Theorem 3: Given a DG with one iterated arc $\boldsymbol{k}$ and check vectors $\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots$, if the projection is chosen as either $\boldsymbol{k}$ or any vector independent to check space $[\boldsymbol{k}, \boldsymbol{c}_1]$ and check space $[\boldsymbol{k}, \boldsymbol{c}_2]$ and so on, a single fault in the PEs affects only one datum in a WCC vector.

Proof: (*a*) If the projection direction is chosen as $\boldsymbol{k}$, it must be a one-to-one mapping between the INS and the IPES as described in Lemma 1. (*b*) If the projection vector is independent to check space $[\boldsymbol{k}, \boldsymbol{c}_1]$, $[\boldsymbol{k}, \boldsymbol{c}_2]$ and so on, the nodes of a different check space should be projected to a PE as described in Lemma 2.

Theorem 4: Given a DG with iterated arcs $k_1, k_2, k_3, \ldots$, if the projection is chosen as $\boldsymbol{Pd} = a\boldsymbol{k}_1 + b\boldsymbol{k}_2 + c\boldsymbol{k}_3 + \ldots$, or vector independent with column space $[\boldsymbol{k}_1, \boldsymbol{c}]$, $[\boldsymbol{k}_2, \boldsymbol{c}]$, $[\boldsymbol{k}_3, \boldsymbol{c}]$, ..., then a single fault in the PEs affects only one data in a WCC, where a, b and c are real.

Proof: With such a projection direction, the mapping between the INS in the DG and the IPES is one-to-one, as described in Lemma 1.

Theorem 5: The array structure should be at least one-dimensional.

Proof: If the dimension of array processor is less than one, there will be multiple data error in a WCC vector under one PE fault. Such an error cannot be detected or corrected.

Based on the preceding theorems, the appropriate projection direction for fault tolerance can be determined. With the same procedure to find $\boldsymbol{S}$ and $\boldsymbol{W}$ presented in Section 2, an algorithm-based, fault-tolerant array processor can be successfully designed. So, the concepts of the WCC have been blended into the derivation of a linear transformation to form a systematic design method for the fault-tolerant array processor. Based on the method, all various structures of algorithm-specific fault-tolerant array processors can be explored.

4 Design examples

Any algorithm may be implemented by an array structure. The choice of the topology and dimensions of the

array depends on not only the computation time and the number of PEs, but also on factors such as the application environment, the pin number, the pipeline period, and so on. The design methodology is therefore not to provide just one optimal topology of some factors, but to provide different topologies for choice. The methodology presented in this paper is examined in the following examples: matrix–matrix multiplication, and Givens reduction. The DG and array processor are represented with geometry graph to help the illustration of our design methodology. Although, these two algorithm have also been considered in other papers [4, 6], their design structure is just one special case. Here, different topologies and dimensions of the array processor can be derived according to different projection directions. Also, other matrix arithmetic algorithm such as matrix-vector multiplication, LU decomposition, Cholesky decomposition and banded-matrix multiplication can be derived similarly with this methodology [23].

4.1 Matrix–matrix multiplication

For matrix–matrix multiplication

$$\boldsymbol{A}_{m\times r}\boldsymbol{B}_{r\times n}=\boldsymbol{C}_{m\times n} \tag{16}$$

Its WCC form of coding distance 3 may be developed following three distinct forms:

$$\boldsymbol{A}_{(m+2)\times r}\boldsymbol{B}_{r\times n}=\boldsymbol{C}_{(m+2)\times n} \tag{17}$$

or

$$\boldsymbol{A}_{m\times r}\boldsymbol{B}_{rx(n+2)}=\boldsymbol{C}_{mx(n+2)} \tag{18}$$

or

$$\boldsymbol{A}_{(m+1)\times r}\boldsymbol{B}_{r\times(n+1)}=\boldsymbol{C}_{(m+1)\times(n+1)} \tag{19}$$

Eqn. 17 is chosen as our design example. The algorithm is represented in detail as:

INPUT: For $i = 1, 2, \ldots, m+2$

For $j = 1, 2, \ldots, n$

$$a^k_{i1} \leftarrow a^k_i \tag{20}$$

$$b^k_{1j} \leftarrow b^k_j \tag{21}$$

$$c^1_{ij} \leftarrow 0 \tag{22}$$

Next j

Next i

COMPU:

For $i = 1, 2, \ldots, m+2$

For $j = 1, 2, \ldots, n$

For $k = 1, 2, \ldots, r$

$$c^k_{ij} \leftarrow c^{k-1}_{ij} + a^k_{ij-1}b_{i-1j} \tag{23}$$

$$a^k_{ij} \leftarrow a^k_{ij-1} \tag{24}$$

$$b^k_{ij} \leftarrow b^k_{i-1j} \tag{25}$$

Next k

Next j

Next i

OUTPUT:

$$c_{ij} \leftarrow c^r_{ij} \tag{26}$$

$$i = 1, 2, \ldots, m+2; j = 1, 2, \ldots, n$$

On this representation, the dependent vectors associated with the data flow stream are easily obtained from eqns. 23, 24 and 25 respectively,

$$d_a = [0 \quad 1 \quad 0]^T$$

$$d_b = [1 \quad 0 \quad 0]^T$$

$$d_c = [0 \quad 0 \quad 1]^T$$

The DG can easily be plotted as shown in Fig. 1*a*. On this DG the iterated arc is

$$\boldsymbol{k} = [i, j, k]^T = [0 \quad 0 \quad 1]^T$$

The transmission arcs are

$$\boldsymbol{m}1 = [1 \quad 0 \quad 0]^T$$

$$\boldsymbol{m}2 = [0 \quad 1 \quad 0]^T$$

The check vector can be chosen:

$$\boldsymbol{C} = [1 \quad 0 \quad 0]^T \text{ or } [1 \quad 0 \quad 1]^T \text{ or } \ldots$$

The check space is then constructed, for example

$$\boldsymbol{CS} = [\boldsymbol{k}, \boldsymbol{c}] = \begin{bmatrix} 0 & 1 \\ 0 & 0 \\ 1 & 0 \end{bmatrix}$$

Projection direction can be selected as iterated arc $\boldsymbol{k} = [0\ 0\ 1]^T$ or vector independent with column space $\boldsymbol{CS}$ such as

$$\boldsymbol{Pd} = [0 \quad 1 \quad 0]^T, [1 \quad 1 \quad 0]^T, [1 \quad 1 \quad 1]^T, \ldots$$

(*a*) First, selecting $\boldsymbol{Pd} = \boldsymbol{v}_1 = [0\ 1\ 0]^T$, the space transformation matrix is

$$\boldsymbol{S}_i = \left[\boldsymbol{I} - \frac{\boldsymbol{v}_1\boldsymbol{v}_1^t}{\boldsymbol{v}_1^t\boldsymbol{v}_1}\right] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$\boldsymbol{S}$ can then be extracted from the first and third rows as

$$\boldsymbol{S} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The timing schedule $\boldsymbol{W} = [w_1 \ w_2 \ w_3]$ can be chosen based on the following constraints of Theorem 2:

(i) $\boldsymbol{Wd} \geqslant 1$ implies $w_1 \geqslant 1$; $w_2 \geqslant 1$; $w_3 \geqslant 1$

(ii) $|\boldsymbol{Wv}_j| \geqslant (\sum_i |\boldsymbol{M}_i \boldsymbol{Wv}_i|) + 1$ implies $|w_2| \geqslant 1$

For these two inequalities, the only corner of the feasible set is $\boldsymbol{W} = [1\ 1\ 1]$. Combining $\boldsymbol{W}$ and $\boldsymbol{S}$, then transformation matrix is

$$\boldsymbol{T} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

According to the transformation method in Section 2, the array processor is then derived and shown as in Fig. 2*b*.

(*b*) If the linear array structure is a necessary consideration, double projection has to be adopted. Suppose that the following two independent projection directions are chosen:

$$\boldsymbol{Pd}1 = [0 \quad 1 \quad 0]^t = \boldsymbol{v}_1$$

$$\boldsymbol{Pd}2 = [0 \quad 0 \quad 1]^T = \boldsymbol{v}_2$$

The space transformation can then be derived.

$$S_t = \left[I - \frac{v_1 v_1^t}{v_1^t v_1} - \frac{v_2 v_2^t}{v_2^t v_2} \right]$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

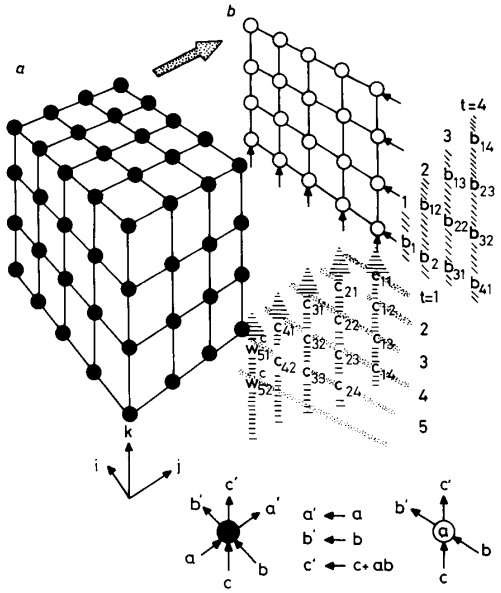

Fig. 2 *(a) DG of matrix–matrix multiplication and (b) array processor with projection along* $[0\ 1\ 0]^T$

S can then be extracted as

$$S = [1 \quad 0 \quad 0]$$

Timing schedule: $M1 = M2 = n - 1$ (defined in Theorem 2):

(i) $Wd \geq 1 \Rightarrow w_1 \geq 1; w_2 \geq 1; w_3 \geq 1$

(ii) $|Wv_j| \geq (\sum_i |M_i Wv_i|) + 1 \Rightarrow |w_2| \geq 1$

that is

$$|w_2| \geq (n-1)|w_3| + 1 \quad \text{or} \quad |w_3| \geq (n-1)|w_2| + 1$$

and minimise the cost function.

$$\text{Computation time} = \max [W(j_2 - j_1)] + 1$$
$$= (n-1)w_1 + (n-1)w_2 + (n-1)w_3 + 1$$

The corners of the feasible set are $W = [1\ n\ 1]$ or $[1\ 1\ n]$. Both of these two selections have introduced equal computation time. If we arbitrarily choose W as $[1\ 1\ n]$, Fig. 3 can be obtained. Comparing this to the array processor of matrix-vector multiplication of Hwang [4], his design is just like Fig. 3. However there B is a vector instead of a matrix.

(*c*) If another projection pair

$$Pd_1 = [0 \quad 0 \quad 1]^T \quad Pd_2 = [1 \quad p \quad 0]^T \quad 0 < p < n$$

is chosen, a linear structure called 'p-diagonalisation' [24] can be derived.

4.2 Given reduction

The givens reduction is an important matrix arithmetic algorithm in the least-squares solution, the eigenvalue

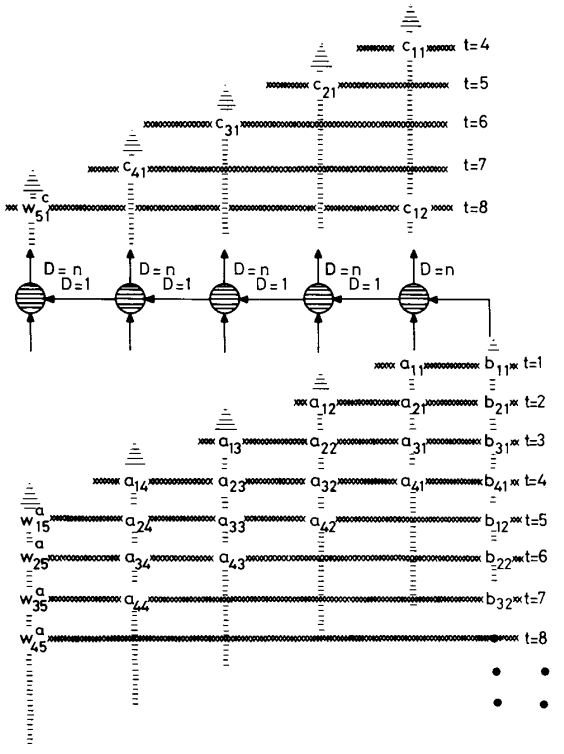

Fig. 3 *Array processor of matrix multiplication after double projection*

calculation and singular-value decomposition, which are quite useful in applications such as beamformer, target tracking and Kalman filter. Its WCC form of the algorithm is

$$A_{m\times(n+1)} \rightarrow R_{n\times(n+1)} \tag{27}$$

and can be represented as follows:

INPUT: $a_{ij}^1 = a_{ij}$ (28)

$i = 1, 2, \ldots, m; j = 1, 2, \ldots, n+1$

COMPU.:

For $k = 1, 2, \ldots, m$

For $i = k, k+1, \ldots, m$

For $j = k, k+1, \ldots, n+1$

$$a_{ij}^k = a_{ij}^{k-1} c_{ij-1}^k - 1_{i-1j}^k s_{ij-1}^k \tag{29}$$

$$l_{ij}^k = a_{ij}^{k-1} c_{ij-1}^k + l_{i-1j}^k c_{ij-1}^k \tag{30}$$

$$c_{ij}^k = \begin{cases} l_{i-1j}^k / [(l_{i-1j}^k)^2 + (a_{ij}^{k-1})^2]^{1/2} & \text{if } j = k \\ c_{ij-1}^k & \text{otherwise} \end{cases} \tag{31}$$

$$s_{ij}^k = \begin{cases} a_{ij}^{k-1} / [(l_{i-1j}^k)^2 + (a_{ij}^{k-1})^2]^{1/2} & \text{if } j = k \\ s_{ij-1}^k & \text{otherwise} \end{cases} \tag{32}$$

Next j

Next i

Next k

OUTPUT: $r_{kj} = l_{mj}^k$ (33)

$j = 1, 2, \ldots, n+1; k = 1, 2, \ldots, m$

In this representation, the dependent vectors associated with the data flow stream can be obtained.

$$\boldsymbol{d}_a = [0 \quad 0 \quad 1]^T$$

$$\boldsymbol{d}_1 = [1 \quad 0 \quad 0]^T$$

$$\boldsymbol{d}_c = \boldsymbol{d}_s = [0 \quad 1 \quad 0]^T$$

DG is plotted in Fig. 4. In the DG, the iterated arcs are

$$\boldsymbol{k}1 = [0 \quad 0 \quad 1]^T \quad \boldsymbol{k}2 = [1 \quad 0 \quad 0]^T$$

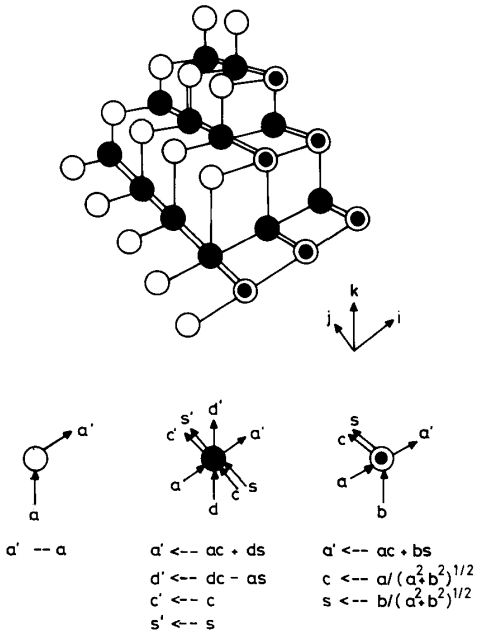

Fig. 4 *DG of Givens reduction*

and the transmission arcs are

$$\boldsymbol{m} = [0 \quad 1 \quad 0]^T \quad \boldsymbol{c} = [0 \quad 1 \quad 0]^T$$

Check spaces are

$$\boldsymbol{CS}_1 = [\boldsymbol{k}, \boldsymbol{c}] = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$\boldsymbol{CS}_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$$

Projection direction can be selected as an iterated arc $\boldsymbol{k}1 = [0\ 0\ 1]^T$ or $\boldsymbol{k}2 = [1\ 0\ 0]^T$ or vector independent with column space $\boldsymbol{CS}_1$ and $\boldsymbol{CS}_2$ such as $[1\ 1\ 1]^T, \ldots$.

(*a*) First selecting $\boldsymbol{v}_1$ as $[1\ 0\ 0]^T$, the space transformation matrix and the timing schedule function can be derived by the same method as matrix–matrix multiplication.

$$\boldsymbol{S}_t = \left[\boldsymbol{I} - \frac{\boldsymbol{v}_1 \boldsymbol{v}_1^t}{\boldsymbol{v}_1^t \boldsymbol{v}_1}\right] = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{S} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Since $\boldsymbol{W} = [1\ 1\ 1]$

$$\boldsymbol{T} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The array processor can then be obtained, as shown in Fig. 5.

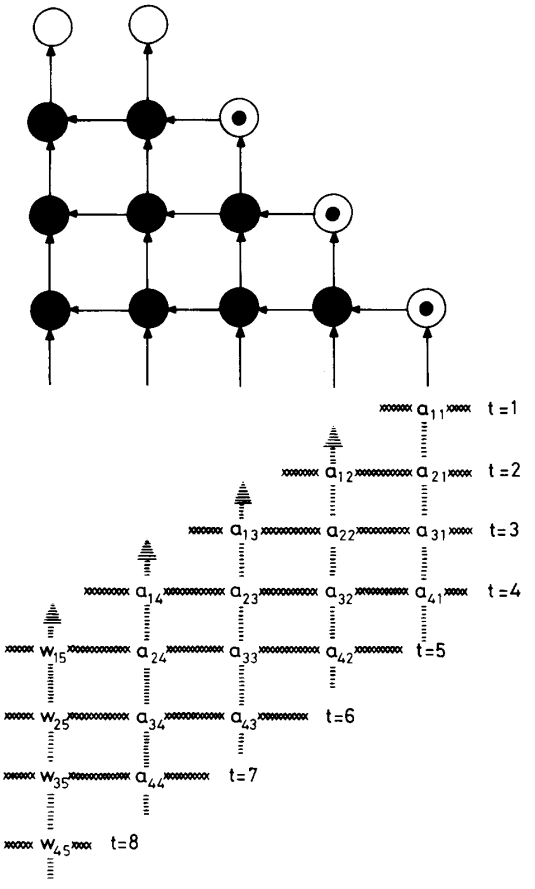

Fig. 5 *Array processor of Givens reduction*

(*b*) Now, double projection is used in the example. Two projection directions are chosen:

$$\boldsymbol{v}_1 = [1 \quad 0 \quad 0]^T$$

$$\boldsymbol{v}_2 = [0 \quad 0 \quad 1]^T$$

The space transformation $\boldsymbol{S}$ can then be obtained as

$$\boldsymbol{S} = [0 \quad 1 \quad 0]^T$$

and the structure of array processor is shown in Fig. 6.

By Theorem 2, $M_1 = m - 1$ and $M_2 = m - 1$

(i) $\boldsymbol{WD} \geqslant 1$ implies $w_1 \geqslant 1; w_2 \geqslant 1; w_3 \geqslant 1$

(ii) $|\boldsymbol{W}\boldsymbol{v}_1| \geqslant (m-1)|\boldsymbol{W}\boldsymbol{v}_2| + 1$ or $|\boldsymbol{W}\boldsymbol{v}_2| \geqslant (m-1)|\boldsymbol{W}\boldsymbol{v}_1|$

That is

$$|w_1| \geqslant (m-1)|w_2| + 1 \quad \text{or} \quad |w_2| \geqslant (m-1)|w_1| + 1$$

Two corners of the feasible set are $\boldsymbol{W} = [m\ 1\ 1]$ and $[1\ 1\ m]$. To minimise the cost function, computation time = max $[\boldsymbol{W}(j_2 - j_1)] + 1 = (m-1)w_1 + (n)w_2 + (m-1)w_3 + 1$, $\boldsymbol{W} = [1\ 1\ n]$ is chosen.

Then the transformation matrix is obtained.

$$T = \begin{bmatrix} n & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

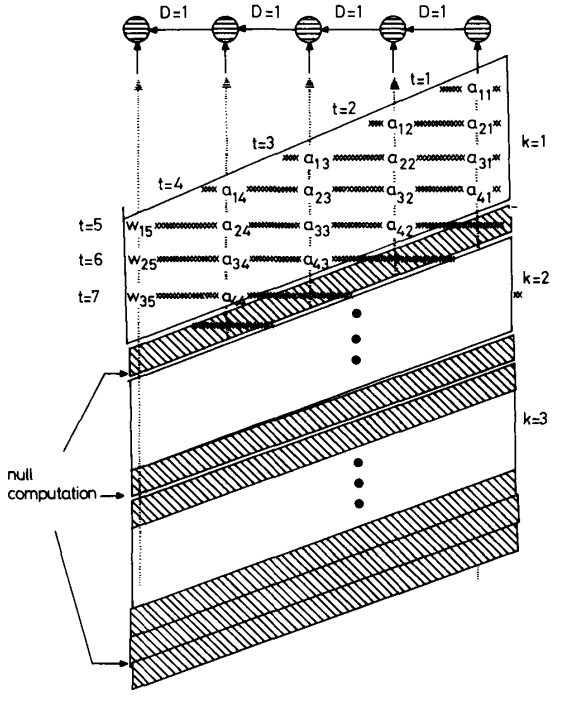

Fig. 6 *Array processor of Givens reduction after double projection*

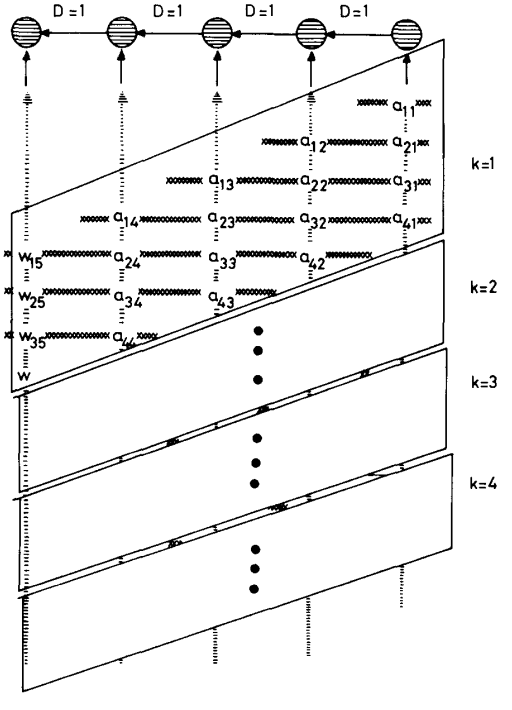

Fig. 7 *Array processor after data compaction*

By this transformation, the array processor can be derived, which is shown in Fig. 6. In this example, the node space to be distributed into a PE has a triangular form instead of a rectangular form. The linear time schedule will introduce some null computation in the array processor as described in Section 2 which is shown in Fig. 6. Data compaction can be applied to increase the PE efficiency as shown in Fig. 7.

(*c*) If another pair of projection direction

$$\boldsymbol{Pd}_1 = [1 \quad 0 \quad 0]^T \quad \boldsymbol{Pd}_2 = [1 \quad 1 \quad 1]^T$$

is chosen, the linear array structure as presented by Jou [6] can then be derived.

6 Conclusion

In this paper we have presented a systematical design methodology for a fault-tolerant array processor. By the methodology, an algorithm of matrix computation can be transformed to a fault-tolerant array processor with different dimensions and topologies. We hope the method with its high flexibility for network topology, high feasibility for CAD and low overhead in hardware and time can promote the design of the fault-tolerant array processor to a system level.

7 Acknowledgment

This work was supported by the National Science Council, Taiwan, ROC, under grant NSC77-0404-E009-04.

8 References

1 KUNG, H.T., and LEISERSON, C.E.: 'Systolic arrays (for VLSI)'. Proc. SIAM Sparse Matrix Symp., 1978, pp. 256–282
2 SAMI, M.G., and STEFANELLI, R.: 'Reconfigurable architectures for VLSI processing arrays', *Proc. IEEE*, 1986, **74**, (5), pp. 712–722
3 SAMI, M.G., and STEFANELLI, R.: 'Self-testing array structures'. Proc. IEEE ICCD, 1984, pp. 677–682
4 HUANG, K.H., and ABRAHAM, J.A.: 'Algorithm-based fault-tolerance for matrix operations', *IEEE Trans. Comp.*, 1984, **C-33**, (6), pp. 518–528
5 BANERJEE, P., and ARRAHAM, J.A.: 'Bounds on algorithm-based fault tolerance in multiple processor systems', *IEEE Trans. Comput.*, 1986, **C-35**, (4), pp. 296–306
6 JOU, J.Y., and ABRAHAM, J.A.: 'Fault-tolerant matrix arithmetic and signal processing on highly concurrent computing structures', *Proc. IEEE*, 1986, (5), pp. 732–740
7 LUK, F.T.: 'Algorithm-based fault tolerance for parallel matrix equation solvers'. SPIE **564** Real Time Signal Processing VIII, 1985, pp. 49–53
8 KUNG, S.Y., CHENG, C.W., and JEN, C.W.: 'Real-time reconfiguration for fault-tolerant VLSI array processors'. Proc. IEEE Real-Time System Symp., Dec. 1986, pp. 46–54
9 GULATI, R.K., and REDDY, S.M.: 'Concurrent error detection in VLSI array structures'. Proc. IEEE ICCD, 1986, pp. 488–491
10 FORTES, J.A.B., and REGHAVENDRA, C.S.: 'Gracefully degradable processor arrays', *IEEE Trans. Comput.*, 1985, (11), pp. 1033–1044
11 KOREN, I.: 'A reconfigurable and fault-tolerant VLSI multiprocessor array'. 8th Symp. Computer Architecture, 1981, pp. 425–442
12 SOMANI, A.K., AGARWAL, V.K., and AVIS, D.: 'A generalized theory for system level diagnosis'. Proc. IEEE ICCD, 1985, pp. 707–711
13 COSENTINO, R.J.: 'Concurrent error correction in systolic architectures', *IEEE Trans. CAD*, 1988, **7**, (1), pp. 117–125
14 CHAN, S.-W., and WEY, C.-L.: 'The design of concurrent error diagnosable systolic arrays for band matrix multiplication', *IEEE Trans. CAD*, 1988, **7**, (1), pp. 21–37
15 MOLDOVAN, D.I.: 'ADVIS: A software package for the design of systolic arrays', *IEEE Trans. CAD*, 1987, **6**, (1), pp. 33–40
16 QUINTON, P.: 'Synthesizing systolic arrays using DIASTOL'. International Workshop of Systolic Arrays, University of Oxford, Department for External Studies, 2nd–4th July 1986, pp. 4.1–4.12

17 KUNG, S.Y., LO, S.C., and LEWIS, P.S.: 'Optimum systolic design for transitive closure and the shortest path problems', *IEEE Trans. Comput*, 1987, **C-36**, (5), pp. 603–614
18 CHEN, M.C.: 'Synthesizing systolic designs'. 1985 International Symposium on VLSI Technology, Systems and Applications, Taipei, Taiwan, May 1985, pp. 209–215
19 CAPPELLO, P.R., and STEIGLITZ, K.: 'Unifying VLSI array designs with geometric transformations'. Proc. Int. Conf. Parallel Processing, 1983
20 KARP, R.M., MILLER, R.E., and WINORRAD, S.: 'The organization of computations for uniform recurrence equations', *J. Ass. Comput. Mach.*, 1967, **14**, (7), pp. 563–590
21 STRANG, G.: 'Linear algebra and its applications' (Academic Press, New York, 1976), pp. 305–323
22 LEISERSON, C., ROSE, F., and SAXE, J.: 'Optimizing synchronous circuitry by retiming'. Third Caltech Conference on VLSI, Pasadena, California, March 1983
23 LIU, C.M., and JEN, C.W.: 'The design of VLSI array processors for concurrent error detection'. Master Thesis of Sciences in Electrical Engineering of Chiao Tung University, Taiwan, June 1987
24 RAMAKRISHNAN, I.V., and VARMAN, P.J.: 'An optimal family of matrix multiplication algorithms linear arrays'. IEEE Parallel Processing, 1985, pp. 376–383