IEICE TRANS. INF. & SYST., VOL.E84–D, NO.12 DECEMBER 2001

PAPER

Reliable Data Routing for Spatial-Temporal TMR Multiprocessor Systems

Mineo KANEKO[†], ***Regular Member***

SUMMARY This paper treats the data routing problem for fault-tolerant systolic arrays based on Triple Modular Redundancy (TMR) in mixed spatial-temporal domain. The number of logical links required in TMR systolic array is basically 9 times larger than the one for corresponding non-fault-tolerant systolic array. The link sharing is a promising method for reducing the number of physical links, which may, however, degrade the fault tolerance of TMR system. This paper proposes several robust data-routing and resource-sharing (plural data transfers share a physical link, or a data transfer and a computational task share a PE as a relay node for the former and as a processor for the latter), by which certain classes of fault tolerant property will be guaranteed. A *stage* and a *dominated set* are introduced to characterize the features of routing/resource-sharing in TMR systems, and conditions on the dominated set and their resultant fault-tolerant properties are derived.

key words: *systolic arrays, fault tolerance, on-line error correction, routing, network architecture,*

1. Introduction

The evolution in VLSI technology allows various complicated arithmetic to be implemented on VLSI chip using multiple, regular connected processing elements, which is named a systolic array. It has a great potential in concurrency and pipeline-ability for resolving large computation problems, such as matrix arithmetic, signal processing and other scientific/engineering applications [1]–[5]. A major drawback of such a high degree of integration is the high possibility of failures in the system. When a target algorithm has been tightly and irredundantly mapped into the spatial-temporal space, a single failure in processing elements (PEs) or communication links makes the entire computing system useless.

Reconfiguration is one of proper approaches to fault tolerant systolic arrays, in which we can fully utilize the modularity of the systolic arrays [13], [14]. However it needs a certain time to achieve reconfiguration, and, in general, the reconfiguration will be applied at fabrication time or with suspending application services. Focusing on on-line error detection and correction, algorithm-based techniques have been shown to be efficient in redundancy overhead [9]–[12]. However, most of algorithm-based techniques can be applied only to linear algebra-based applications. Furthermore, to avoid the error diffusion which causes too many erroneous data to preserve the finite distance property, the mapping from tasks to PEs is strongly limited, which may possibly degrade the concurrency and pipelining-ability.

Triple Modular Redundancy (TMR) is one of the most popular schemes for fault tolerant computing. To preserve the modularity/scalability, connection-locality and massively parallelism of the systolic array, the technique will be applied with cell (PE)-based triplication, and voting mechanism is quipped on each cell to mask internal erroneous data immediately. While the concept of TMR technique is a classical one in a sense, it is an inherent feature of TMR systolic array that hardware redundancy imposed by the triplication can be traded for time redundancy, and we have various design alternatives with different mixtures of hardware-time redundancy.

This paper addresses the data routing problem for fault-tolerant systolic arrays based on TMR in mixed spatial-temporal domain [15]. Since the number of logical links required in TMR systolic array is basically 9 times larger than the one for corresponding non-fault-tolerant systolic array. The link sharing is a promising method for reducing the number of physical links and hence for reducing hardware complexity of the resultant system. Both to increase the possibility of link sharing and to improve the connection locality, the decomposition of a data transfer into a series of data transfers via several PEs is also a candidate technique [16]. However, these modifications may possibly degrade fault tolerance of the TMR systolic array. In literature [15], the decomposition of a data transfer into a series of data transfers is not treated, and literature [16] discusses the scheduling aspect of routing and resource-sharing for data transfers, but does not the fault-tolerance aspect of them. The purpose of this paper is to develop the relation between routing and resource-sharing for data transfers and its resultant tolerance to faults. To achieve it, a stage and a dominated set will be introduced to characterize routing and resource-sharing scheme, and some conditions for routing/resource-sharing to possess certain fault tolerant property will be derived in this paper.

This paper is organized as follows. In Sect. 2, a systematic design procedure for TMR systolic arrays is

Manuscript received December 18, 2000.
Manuscript revised June 22, 2001.

[†]The author is with the School of Information Science, Japan Advanced Institute of Science and Technology, Ishikawa-ken, 923–1292 Japan.

briefly outlined. Problem of physical implementation of data communications and some basic strategies for reducing the complexity of physical links are described in Sect. 3. In Sect. 4, conditions for reliable routing and link-sharing in TMR system is discussed. An design example is shown in Sect. 5. Finally, we conclude our remarks in Sect. 6.

2. Triple Modular Redundancy in Mixed Spatial-Temporal Domain

Data processing on a systolic array system includes computations in each processing element and data communication between processing elements. A systematic design of fault tolerant systolic array based on Triple Modular Redundancy (TMR) begins with the triplication of an original dependence graph, and it is followed by the mapping of the triplicated dependence graph onto physical spatial-temporal space [15].

Throughout this paper, $\boldsymbol{\delta}_0$, $\boldsymbol{\delta}_1$ and $\boldsymbol{\delta}_2$ represent vectors $(0,0)$, $(1,0)$ and $(0,1)$, respectively.

2.1 From Dependence Graph to Triplicated Dependence Graph

Assume that a target algorithm which is to be implemented is given by uniform recurrence equations and a N-dimensional Dependence Graph (DG) $G = (\boldsymbol{V}, \boldsymbol{A}(\boldsymbol{D}))$ is constructed. Each node in $\boldsymbol{V}$ corresponds to a unit computational task and is addressed by a N-dimensional vector, say $v(\boldsymbol{a})$ as a node at $\boldsymbol{a} \in Z^N$. Each directed arc in $\boldsymbol{A}(\boldsymbol{D})$ represents data flow (data dependency). The set of arcs $\boldsymbol{A}(\boldsymbol{D})$ is generated from a set of kernel vectors named *dependence vectors* $\boldsymbol{D} \subset Z^N$ as follows;

$$\boldsymbol{A}(\boldsymbol{D}) = \{\,(v(\boldsymbol{a}), v(\boldsymbol{a}+\boldsymbol{d}))| \\ \boldsymbol{d} \in \boldsymbol{D}, v(\boldsymbol{a}) \in \boldsymbol{V}, v(\boldsymbol{a}+\boldsymbol{d}) \in \boldsymbol{V}\} \quad (1)$$

Furthermore, $\boldsymbol{D}$ is partitioned into two subsets. One is the set of iterated dependence vectors, $\boldsymbol{D}^{(I)}$, along which iteratively updated data pass. The other one is the set of transmission dependence vectors, $\boldsymbol{D}^{(T)}$, along which constant data pass.

Triplicated Dependence Graph (TDG), $G_T = (\boldsymbol{V}_T, \boldsymbol{A}_T(\boldsymbol{D}))$, is a triplicated version of a dependence graph, in which the address of each node is specified with a $(N+2)$-dimensional vector and the set of arcs $\boldsymbol{A}_T(\boldsymbol{D})$ is generated from the original dependence vectors $\boldsymbol{D}$ as shown below.

$$\boldsymbol{V}_T = \{v(\boldsymbol{a}, \boldsymbol{\delta}_i) | v(\boldsymbol{a}) \in \boldsymbol{V}, i \in \{0,1,2\}\} \quad (2)$$

$$\boldsymbol{A}_T(\boldsymbol{D}) = \{(v(\boldsymbol{a}, \boldsymbol{\delta}_i), v(\boldsymbol{a}+\boldsymbol{d}, \boldsymbol{\delta}_j))\,|\boldsymbol{d} \in \boldsymbol{D}, \\ v(\boldsymbol{a}, \boldsymbol{\delta}_i) \in \boldsymbol{V}_T, v(\boldsymbol{a}+\boldsymbol{d}, \boldsymbol{\delta}_j) \in \boldsymbol{V}_T\} \quad (3)$$

Figure 1 shows a simplified model of TDG.

Assuming $\boldsymbol{D} = \{\boldsymbol{d}_1, \boldsymbol{d}_2, \cdots, \boldsymbol{d}_K\}$, in DG, each node $v(\boldsymbol{a})$ first receives K different data from $v(\boldsymbol{a}-\boldsymbol{d}_1)$, $v(\boldsymbol{a}-\boldsymbol{d}_2)$, $\cdots$, $v(\boldsymbol{a}-\boldsymbol{d}_K)$, respectively, it performs its task, and finally it delivers K different data to $v(\boldsymbol{a}+\boldsymbol{d}_1)$, $v(\boldsymbol{a}+\boldsymbol{d}_2)$, $\cdots$, $v(\boldsymbol{a}+\boldsymbol{d}_K)$, respectively. On the other hand, in TDG, each node $v(\boldsymbol{a}, \boldsymbol{\delta}_i)$ receives K sets of three data, each set from $v(\boldsymbol{a}-\boldsymbol{d}_k, \boldsymbol{\delta}_0)$, $v(\boldsymbol{a}-\boldsymbol{d}_k, \boldsymbol{\delta}_1)$ and $v(\boldsymbol{a}-\boldsymbol{d}_k, \boldsymbol{\delta}_2)$, $k \in \{1,2,\cdots,K\}$, and it takes majority over each set of three data to get error-corrected K inputs. Then the node performs its task and distributes each of K results to three nodes $v(\boldsymbol{a}+\boldsymbol{d}_k, \boldsymbol{\delta}_0)$, $v(\boldsymbol{a}+\boldsymbol{d}_k, \boldsymbol{\delta}_1)$ and $v(\boldsymbol{a}+\boldsymbol{d}_k, \boldsymbol{\delta}_2)$, $k \in \{1,2,\cdots,K\}$. As a result, in TDG, any single node fault among three nodes $v(\boldsymbol{a}, \boldsymbol{\delta}_0)$, $v(\boldsymbol{a}, \boldsymbol{\delta}_1)$ and $v(\boldsymbol{a}, \boldsymbol{\delta}_2)$ is tolerable for each $\boldsymbol{a}$.

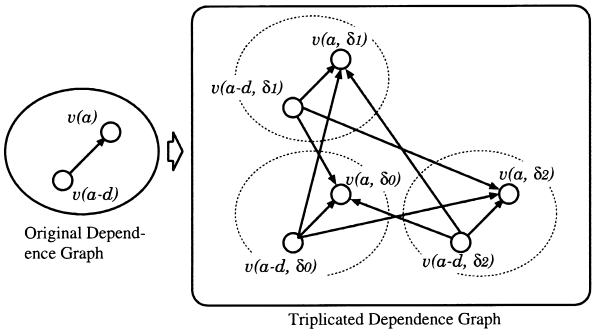

Fig. 1 Triplicated dependence graph.

2.2 Transformation to Systolic Array Processing

The space transformation matrix $\boldsymbol{P} = [\boldsymbol{P}_O, \boldsymbol{P}_R]$ with its dimension $M \times (N+2)$ and the timing schedule function $\boldsymbol{W} = [\boldsymbol{W}_O, \boldsymbol{W}_R]$ with its dimension $1 \times (N+2)$ define the mapping from TDG to array processing. That is, a PE address $\boldsymbol{p}$ in M-dimensional space and a control step t for a node $v(\boldsymbol{a}, \boldsymbol{\delta}_i)$ to be assigned will be given as,

$$\begin{bmatrix} \boldsymbol{p}^T \\ t \end{bmatrix} = \overbrace{\begin{bmatrix} \boldsymbol{P}_O & \boldsymbol{P}_R \\ \boldsymbol{W}_O & \boldsymbol{W}_R \end{bmatrix}}^{N\quad 2} \begin{bmatrix} \boldsymbol{a}^T \\ {\boldsymbol{\delta}_i}^T \end{bmatrix} \quad (4)$$

Using the same $\boldsymbol{P}$ and $\boldsymbol{W}$, a new data dependence vector $(\boldsymbol{d}_k, \boldsymbol{\delta}_j - \boldsymbol{\delta}_i)$ in TDG is mapped as,

$$\begin{bmatrix} {\boldsymbol{\ell}_k^{(i,j)}}^T \\ De(\boldsymbol{\ell}_k^{(i,j)}) \end{bmatrix} = \begin{bmatrix} \boldsymbol{P}_O & \boldsymbol{P}_R \\ \boldsymbol{W}_O & \boldsymbol{W}_R \end{bmatrix} \begin{bmatrix} {\boldsymbol{d}_k}^T \\ (\boldsymbol{\delta}_j - \boldsymbol{\delta}_i)^T \end{bmatrix} \quad (5)$$

where $\boldsymbol{\ell}_k^{(i,j)}$ represents the spatial difference between a source PE and a destination PE of data transfer, and $De(\boldsymbol{\ell}_k^{(i,j)})$ represents the time difference (the number of control steps) between the data creation at a source PE and the data consumption at a destination PE. In the following, $\boldsymbol{\ell}_k^{(i,j)}$ will be called *communication vector*.

In choosing the space transformation matrix and the timing schedule function, the following constraints are claimed.

$$i \neq j \Rightarrow \boldsymbol{P}_R {\boldsymbol{\delta}_i}^T \neq \boldsymbol{P}_R {\boldsymbol{\delta}_j}^T \tag{6}$$

$$\boldsymbol{W}\,(\boldsymbol{d}_k, \boldsymbol{\delta}_j - \boldsymbol{\delta}_i)^T = \boldsymbol{W}_O {\boldsymbol{d}_k}^T + \boldsymbol{W}_R(\boldsymbol{\delta}_i - \boldsymbol{\delta}_j)^T \geqq 1 \tag{7}$$

$$(\boldsymbol{a}, \boldsymbol{\delta}_i) \neq (\boldsymbol{b}, \boldsymbol{\delta}_j) \Rightarrow \begin{bmatrix} \boldsymbol{P}_O\ \boldsymbol{P}_R \\ \boldsymbol{W}_O\ \boldsymbol{W}_R \end{bmatrix} \begin{bmatrix} \boldsymbol{a}^T \\ {\boldsymbol{\delta}_i}^T \end{bmatrix} \neq \begin{bmatrix} \boldsymbol{P}_O\ \boldsymbol{P}_R \\ \boldsymbol{W}_O\ \boldsymbol{W}_R \end{bmatrix} \begin{bmatrix} \boldsymbol{b}^T \\ {\boldsymbol{\delta}_j}^T \end{bmatrix} \tag{8}$$

The first claim guarantees a single erroneous data among three replicas by a single faulty processor, the second one is for preserving data dependency in temporal domain, and the third one for avoiding execution conflicts.

Since every computation is triplicated and results are completely exchanged right after the computations, any single PE failure among three PEs onto which triplicated nodes $v(\boldsymbol{a}, \boldsymbol{\delta}_0)$, $v(\boldsymbol{a}, \boldsymbol{\delta}_1)$ and $v(\boldsymbol{a}, \boldsymbol{\delta}_2)$ are respectively mapped is tolerable.

3. Physical Implementation of Logical Links

Figure 2 shows an original DG for a matrix multiplication, in which $\boldsymbol{d}_1 = (0, 1, 0)$ and $\boldsymbol{d}_2 = (1, 0, 0)$ are two transmission dependence vectors, and $\boldsymbol{d}_3 = (0, 0, 1)$ is an iterated dependence vector. From this original DG, first, TDG will be constructed within $(3 + 2)$-dimensional space, and next nodes are mapped onto spatial-temporal space by the space transformation matrix and timing scheduling function. Figure 3 shows a rough sketch of resultant PE array and input/output sequences to the system designed with the following transformation matrix and timing scheduling function.

$$\boldsymbol{P} = \overbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \end{bmatrix}}^{\boldsymbol{P}_O} \overbrace{\begin{bmatrix} -1 & -1 \\ 0 & -1 \end{bmatrix}}^{\boldsymbol{P}_R} \tag{9}$$

$$\boldsymbol{W} = \overbrace{\begin{bmatrix} 1 & 1 & 2 \end{bmatrix}}^{\boldsymbol{W}_O} \overbrace{\begin{bmatrix} 0 & 0 \end{bmatrix}}^{\boldsymbol{W}_R} \tag{10}$$

In Fig. 3, a box with $P_{i,j}$ denotes a PE and a_{ij} (ith row jth element in A), b_{ij} (ith row jth element in B) and c_{ij} (ith row jth element in C) are the primary input/output data, which are drawn on "control step"× "primary input/output port" plane. Data transfers (and physical links) between PEs are excluded in this figure, since their complete drawing falls into a complicated figure. On the other hand, Table 1 shows communication vectors generated in this design, and Fig. 4 draws the set of data transfers needed for $v(\boldsymbol{a}, \boldsymbol{\delta}_0)$ (mapped onto $P_{i,j}$), $v(\boldsymbol{a}, \boldsymbol{\delta}_1)$ (mapped onto $P_{i-1,j}$) and $v(\boldsymbol{a}, \boldsymbol{\delta}_2)$ (mapped onto $P_{i-1,j-1}$) to execute their own tasks, and each of those data transfers corresponds to one of communication vectors shown in Table 1. As we have mentioned before, each communication vector represents relative position of a destination PE from a source PE for a data transfer, and $P_{i,j}$, $P_{i-1,j}$ and $P_{i-1,j-1}$ are any three PEs with the same relative position (their absolute positions are determined depending on the value of $\boldsymbol{a}$) in the PE array shown in Fig. 3. Note

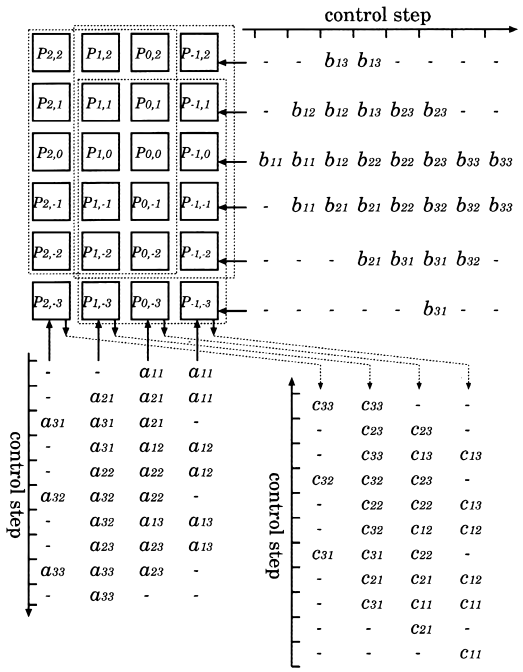

Fig. 3 A systolic implementation of TDG for the matrix multiplication: PE array and input/output sequences. Three dotted rectangles show three ranges onto each of which $\{v(\boldsymbol{a}, \boldsymbol{\delta}_0) | v(\boldsymbol{a}) \in \boldsymbol{V}\}$, $\{v(\boldsymbol{a}, \boldsymbol{\delta}_1) | v(\boldsymbol{a}) \in \boldsymbol{V}\}$ or $\{v(\boldsymbol{a}, \boldsymbol{\delta}_2) | v(\boldsymbol{a}) \in \boldsymbol{V}\}$ is mapped.

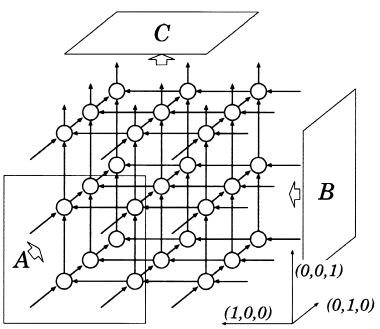

Fig. 2 DG for $\boldsymbol{C} = \boldsymbol{A} \times \boldsymbol{B}$.

Table 1 List of communication vectors.

generated from $\boldsymbol{d}_1$	generated from $\boldsymbol{d}_2$	generated from $\boldsymbol{d}_3$
$\boldsymbol{\ell}_1^{(0,0)} = (0, 1)$	$\boldsymbol{\ell}_2^{(0,0)} = (1, 0)$	$\boldsymbol{\ell}_3^{(0,0)} = (0, -1)$
$\boldsymbol{\ell}_1^{(0,1)} = (-1, 1)$	$\boldsymbol{\ell}_2^{(0,1)} = (0, 0)$	$\boldsymbol{\ell}_3^{(0,1)} = (-1, -1)$
$\boldsymbol{\ell}_1^{(0,2)} = (-1, 0)$	$\boldsymbol{\ell}_2^{(0,2)} = (0, -1)$	$\boldsymbol{\ell}_3^{(0,2)} = (-1, -2)$
$\boldsymbol{\ell}_1^{(1,0)} = (1, 1)$	$\boldsymbol{\ell}_2^{(1,0)} = (2, 0)$	$\boldsymbol{\ell}_3^{(1,0)} = (1, -1)$
$\boldsymbol{\ell}_1^{(1,1)} = (0, 1)$	$\boldsymbol{\ell}_2^{(1,1)} = (1, 0)$	$\boldsymbol{\ell}_3^{(1,1)} = (0, -1)$
$\boldsymbol{\ell}_1^{(1,2)} = (0, 0)$	$\boldsymbol{\ell}_2^{(1,2)} = (1, -1)$	$\boldsymbol{\ell}_3^{(1,2)} = (0, -2)$
$\boldsymbol{\ell}_1^{(2,0)} = (1, 2)$	$\boldsymbol{\ell}_2^{(2,0)} = (2, 1)$	$\boldsymbol{\ell}_3^{(2,0)} = (1, 0)$
$\boldsymbol{\ell}_1^{(2,1)} = (0, 2)$	$\boldsymbol{\ell}_2^{(2,1)} = (1, 1)$	$\boldsymbol{\ell}_3^{(2,1)} = (0, 0)$
$\boldsymbol{\ell}_1^{(2,2)} = (0, 1)$	$\boldsymbol{\ell}_2^{(2,2)} = (1, 0)$	$\boldsymbol{\ell}_3^{(2,2)} = (0, -1)$

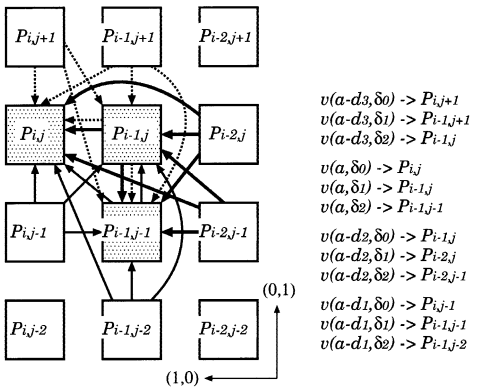

Fig. 4 Data transfers needed for $v(\boldsymbol{a}, \boldsymbol{\delta}_0)$, $v(\boldsymbol{a}, \boldsymbol{\delta}_1)$ and $v(\boldsymbol{\delta}a, \boldsymbol{\delta}_2)$ to execute their tasks in the array processing shown in Fig. 3. Bold solid arrows, thin solid arrows and bold broken arrows indicate data transfers corresponding to dependence vectors $\boldsymbol{d}_1$, $\boldsymbol{d}_2$ and $\boldsymbol{d}_3$, respectively. The right hand side shows the mapping from operation nodes to PEs.

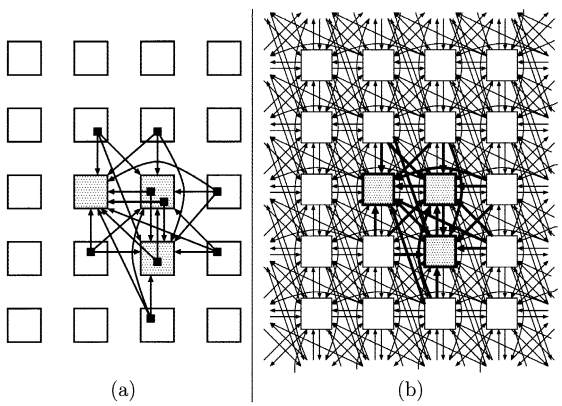

Fig. 5 Routing and network architecture-I. (a) Each data transfer is performed by a direct connection. (b) The overall network architecture. PEs and links with bold lines indicate resources contained in a single stage (see Sect. 4).

that, for $P_{i,j}$, $P_{i-1,j}$ and $P_{i-1,j-1}$ near the boundary of PE array, the data transfer from a source PE would be a primary input from the outside to the PE array.

Figure 5 shows the network architecture† designed by the realization of each logical link with a physical link which directly connects a source PE and a destination PE of a data transfer. In this implementation, logical links having the same source PE and the same destination PE are realized with a single physical link.

Here we encounter the problems; (1) Is it possible to reduce architectural complexity? (2) Is a fault-tolerant property preserved even if plural logical link (data transfers) share a single physical link?

In general, by the multiplication of data dependency, each dependence vector in $\boldsymbol{D}$ for an original DG is multiplicated into 3^2 dependencies, and logical communication links which are generated from $|\{\boldsymbol{\ell}_k^{(i,j)} | \boldsymbol{d}_k \in \boldsymbol{D}, i, j \in \{0,1,2\}\}| = |\boldsymbol{D}| \times 3^2$ communication vectors should be implemented physically on a processor array.

The reduction of the link complexity can be

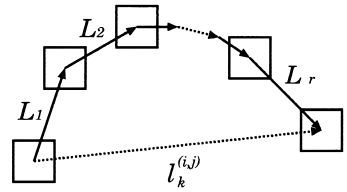

Fig. 6 Decomposition of a communication vector.

achieved by the following factors.

(A) Removing a self-looped link $\boldsymbol{\ell}_k^{(i,j)} = \boldsymbol{O}$.

(B) When the following holds for an original transmission dependence vector $\boldsymbol{d}_k \in \boldsymbol{D}^{(T)}$, data transfer along $\boldsymbol{\ell}_k^{(j_{r-1}, j_r)}$ can be omitted.

$$\boldsymbol{\ell}_k^{(i,j_0)} + \boldsymbol{\ell}_k^{(j_0,j_1)} + \boldsymbol{\ell}_k^{(j_1,j_2)} + \cdots + \boldsymbol{\ell}_k^{(j_{r-2},j_{r-1})} + \boldsymbol{\ell}_k^{(j_{r-1},j_r)} = \boldsymbol{O} \tag{11}$$

(C) If two communication vectors are identical, say $\boldsymbol{\ell}_k^{(i,j)} = \boldsymbol{\ell}_\ell^{(m,n)}$, and the mapping of nodes $\{v(\boldsymbol{a}, \boldsymbol{\delta}_i) | v(\boldsymbol{a}) \in \boldsymbol{V}\}$ and the one in $\{v(\boldsymbol{a}, \boldsymbol{\delta}_m) | v(\boldsymbol{a}) \in \boldsymbol{V}\}$ are spatially overlapped, then two kinds of data transfer, one is along $\boldsymbol{\ell}_k^{(i,j)}$ and the other is along $\boldsymbol{\ell}_\ell^{(m,n)}$, can share a single physical link.

(D) When a communication vector is decomposed into more than one vector, such as,

$$\boldsymbol{\ell}_k^{(i,j)} = L_1 + L_2 + \cdots + L_r \tag{12}$$

then the data transfer along $\boldsymbol{\ell}_k^{(i,j)}$ can be replaced with the series of data transfers along L_1, L_2, $\cdots$, and L_r (Fig. 6). Such decomposition increases the possibility of link sharing and also may contribute to shorten the length of a physical link.

To achieve the decomposition of a communication vector and link sharing, further investigations on both conflict-free scheduling of multiple data communication on a physical link [15], [16] and the reliability degradation imposed by the resource sharing (plural logical links (data transfers) share a physical link, or a data transfer and a computational task share a PE as a relay node for the former and as a processor for the latter) are needed. This paper treats the latter problem, and show some classes of fault-tolerance guaranteed by the appropriately restricted link sharing. Unfortunately, we have not clarified yet the relation between the factor (B) and the reliability, and we will exclude (B) from the discussions in this paper.

4. Reliable Routing and Resource Sharing

We consider a system which consists of physical PEs and physical links. A physical PE is responsible not only for the execution of tasks mapped on it but also

†Figure 5 (b) shows a part of network architecture for a TMR matrix multiplication of a larger size. Note that some physical links are omitted for boundary PEs.

for relaying data transfer. On the other hand, a physical link is just a wire having two ends each of which connects to a physical PE.

We treat both PEs and links as subjects to be faulty, and the number of faults is counted with the number of faulty

Assumption 1: When a PE is faulty, data transfer through this PE as well as computations mapped onto this PE are assumed to be all erroneous. When a physical link is faulty, data transfer through this link are assumed to be all erroneous.

Consider an original DG having a set of dependence vectors $\boldsymbol{D} = \{\boldsymbol{d}_1, \boldsymbol{d}_2, \cdots, \boldsymbol{d}_K\}$. After the conversion from a DG to a TDG followed by the transformation to systolic array processing, a physical PE, onto which a node $v(\boldsymbol{a}, \boldsymbol{\delta}_i)$ in the TDG is mapped, is denoted in short as $P(\boldsymbol{a}, \boldsymbol{\delta}_i), i \in \{0, 1, 2\}$. We will use "*the computation results of* $P(\boldsymbol{a}, \boldsymbol{\delta}_i)$" to represent the computation results with respect to the node $v(\boldsymbol{a}, \boldsymbol{\delta}_i)$.

On the other hand, a *route* of data transfer in the physical domain is defined, in this paper, as an alternate sequence of PEs and links which begins with an out-going link of a source PE and ends with a destination PE, where neighboring PE and link are physically connected. It is assumed that individual data transfer from one PE to another is carried out along a single route.

A resource (a PE or a link) p is said to "*dominate* $P(\boldsymbol{a}, \boldsymbol{\delta}_j)$ *with respect to* $P(\boldsymbol{a} - \boldsymbol{d}_k, \boldsymbol{\delta}_i)$" if p is contained in the route for data transfer from $P(\boldsymbol{a} - \boldsymbol{d}_k, \boldsymbol{\delta}_i)$ to $P(\boldsymbol{a}, \boldsymbol{\delta}_j)$ which corresponds to the communication vector $\boldsymbol{\ell}_k^{(i,j)}$.

Furthermore, we will introduce following notations for convenience's sake.

$$\mathcal{R}(\boldsymbol{a}) \triangleq \{P(\boldsymbol{a}, \boldsymbol{\delta}_0), P(\boldsymbol{a}, \boldsymbol{\delta}_1), P(\boldsymbol{a}, \boldsymbol{\delta}_2)\} \tag{13}$$

$$\begin{aligned}&\mathcal{D}(p, \boldsymbol{a}, \boldsymbol{d}_k, \boldsymbol{\delta}_i)\\ &\quad \triangleq \{P(\boldsymbol{a}, \boldsymbol{\delta}_j) \,|P(\boldsymbol{a}, \boldsymbol{\delta}_j) \in \mathcal{R}(\boldsymbol{a}) \text{ is dominated}\\ &\qquad \text{by } p \text{ with respect to } P(\boldsymbol{a} - \boldsymbol{d}_k, \boldsymbol{\delta}_i)\}\end{aligned} \tag{14}$$

$$\begin{aligned}\mathcal{S}(\boldsymbol{a}) \triangleq \Big\{p \,|p \text{ dominates an element in } \mathcal{R}(\boldsymbol{a})\\ \text{with respect to an element}\\ \text{in } \textstyle\bigcup_{k=1}^{K} \mathcal{R}(\boldsymbol{a} - \boldsymbol{d}_k)\Big\}\end{aligned} \tag{15}$$

Note that $\mathcal{R}(\boldsymbol{a})$ is a set of three different PEs each of which executes one of the triplicated tasks $v(\boldsymbol{a}, \boldsymbol{\delta}_0)$, $v(\boldsymbol{a}, \boldsymbol{\delta}_1)$ and $v(\boldsymbol{a}, \boldsymbol{\delta}_2)$.

We will refer to $\mathcal{S}(\boldsymbol{a})$ as a *stage*, and use "*the outputs of* $\mathcal{S}(\boldsymbol{a})$" to represent the computation results of $P(\boldsymbol{a}, \boldsymbol{\delta}_0)$, $P(\boldsymbol{a}, \boldsymbol{\delta}_1)$ and $P(\boldsymbol{a}, \boldsymbol{\delta}_2)$. Figure 7 illustrates a stage, and Fig. 8 shows an example of $\mathcal{D}(p, \boldsymbol{a}, \boldsymbol{d}_k, \boldsymbol{\delta}_i)$.

Note that, from the definition,

1. $P(\boldsymbol{a}, \boldsymbol{\delta}_i)$ dominates itself with respect to each element in $\bigcup_{k=1}^{K} \mathcal{R}(\boldsymbol{a} - \boldsymbol{d}_k)$.

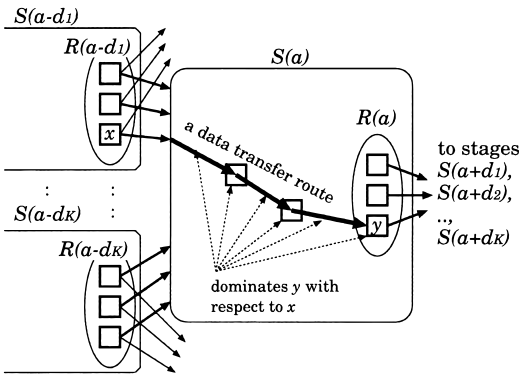

Fig. 7 Illustration of *stage*.

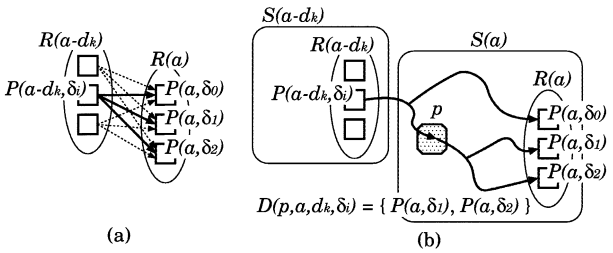

Fig. 8 Example of $\mathcal{D}(p, \boldsymbol{a}, \boldsymbol{d}_k, \boldsymbol{\delta}_i)$: (a) Bold arrows and broken arrows are data transfers from $\mathcal{R}(\boldsymbol{a} - \boldsymbol{d}_k)$ to $\mathcal{R}(\boldsymbol{a})$. (b) Routes for data transfers which are represented with bold arrows in (a), where p is either a PE or a physical link. In this case, $\mathcal{D}(p, \boldsymbol{a}, \boldsymbol{d}_k, \boldsymbol{\delta}_i) = \{P(\boldsymbol{a}, \boldsymbol{\delta}_1), P(\boldsymbol{a}, \boldsymbol{\delta}_2)\}$.

2. $P(\boldsymbol{a} - \boldsymbol{d}_k, \boldsymbol{\delta}_i)$ does not dominate any element in $\mathcal{R}(\boldsymbol{a})$ with respect to $P(\boldsymbol{a} - \boldsymbol{d}_k, \boldsymbol{\delta}_i)$, except the case $P(\boldsymbol{a} - \boldsymbol{d}_k, \boldsymbol{\delta}_i) = P(\boldsymbol{a}, \boldsymbol{\delta}_j)$ for some $P(\boldsymbol{a}, \boldsymbol{\delta}_j) \in \mathcal{R}(\boldsymbol{a})$. ($P(\boldsymbol{a} - \boldsymbol{d}_k, \boldsymbol{\delta}_i)$ may possibly dominate some element in $\mathcal{R}(\boldsymbol{a})$ with respect to some element in $\left(\bigcup_{k=1}^{K} \mathcal{R}(\boldsymbol{a} - \boldsymbol{d}_k)\right) \setminus \{P(\boldsymbol{a} - \boldsymbol{d}_k, \boldsymbol{\delta}_i)\}$.
3. Each stage $\mathcal{S}(\boldsymbol{a})$ has K preceding stages (some may be primary inputs) $\mathcal{S}(\boldsymbol{a} - \boldsymbol{d}_1)$, $\mathcal{S}(\boldsymbol{a} - \boldsymbol{d}_2), \cdots, \mathcal{S}(\boldsymbol{a} - \boldsymbol{d}_K)$, and has K succeeding stages (some may be primary outputs) $\mathcal{S}(\boldsymbol{a} + \boldsymbol{d}_1)$, $\mathcal{S}(\boldsymbol{a} + \boldsymbol{d}_2), \cdots, \mathcal{S}(\boldsymbol{a} + \boldsymbol{d}_K)$.

At a stage $\mathcal{S}(\boldsymbol{a})$ in our triplicated systolic processing, $P(\boldsymbol{a}, \boldsymbol{\delta}_i)$ first receives $K \times 3$ data elements, each of which is sent from one in $\bigcup_{k=1}^{K} \mathcal{R}(\boldsymbol{a} - \boldsymbol{d}_k)$ via elements in $\mathcal{S}(\boldsymbol{a})$. For each $k \in \{1, 2, \cdots, K\}$, three data elements from elements in $\mathcal{R}(\boldsymbol{a} - \boldsymbol{d}_k)$ are then taken their majority to be the correct k-th input. After that, the PE computes K results using these data, and finally it sends K results to the following PEs. In general, each one of K results does not always depend all of K inputs. That is, even if a PE receives some inputs which are uncorrectable by voting, some of the computation results may possibly be error-free. However, in this paper, we will neglect the details of input/output dependency within a node, and will employ the following model on the correctness of the computation results for making our discussions concise.

Assumption 2: Computation results of $P(\boldsymbol{a}, \boldsymbol{\delta}_i)$ are all error-free if and only if

1. $P(\boldsymbol{a}, \boldsymbol{\delta}_i)$ is fault-free, and
2. $P(\boldsymbol{a}, \boldsymbol{\delta}_i)$ receives K sets of at least two correct data,

otherwise they are all erroneous.

When every element in $\mathcal{R}(\boldsymbol{a})$ yields correct outputs, the stage $\mathcal{S}(\boldsymbol{a})$ is said to be *fair*. When exactly one element in $\mathcal{R}(\boldsymbol{a})$ yields erroneous outputs and the other yield correct outputs, the stage is said to be *marginally fair*. Otherwise, it is said to be *unfair*.

When, for every triplicated primary outputs, at least two among them are correct, i.e., every primary output is either correct or correctable by voting, the entire TMR system is said to be *normal*.

The following Lemma 1 is trivial from the nature of the TMR system.

Lemma 1:

(1) A stage $\mathcal{S}(\boldsymbol{a})$ is fair if every preceding stage of the stage $\mathcal{S}(\boldsymbol{a})$ is either fair or marginally fair and all of the elements within $\mathcal{S}(\boldsymbol{a})$ are fault free.
(2) A stage $\mathcal{S}(\boldsymbol{a})$ is unfair if at least one of the preceding stages of the stage $\mathcal{S}(\boldsymbol{a})$ is unfair.
(3) A TMR system is normal if and only if every stage in the system is either fair or marginally fair.

The remaining part of this section is used for filling the gap between (1) and (2) of Lemma 1 for guaranteeing every stage to be fair or marginally fair, i.e., the system to be normal, with respect to a certain class of fault patterns.

Lemma 2: Under the condition that all of the preceding stages of the stage $\mathcal{S}(\boldsymbol{a})$ are fair, the stage $\mathcal{S}(\boldsymbol{a})$ is either fair or marginally fair for any single fault within $\mathcal{S}(\boldsymbol{a})$ iff

$$\forall p \in \mathcal{S}(\boldsymbol{a}),$$
$$\left| \bigcup_{k=1}^{K} \bigcup_{i \neq j} [\mathcal{D}(p, \boldsymbol{a}, \boldsymbol{d}_k, \boldsymbol{\delta}_i) \cap \mathcal{D}(p, \boldsymbol{a}, \boldsymbol{d}_k, \boldsymbol{\delta}_j)] \right| \leqq 1 \quad (16)$$

holds, where $\bigcup_{i \neq j}$ denotes the union over $0 \leqq i \leqq 2$, $0 \leqq j \leqq 2$, $i \neq j$.

Proof of Lemma 2:

Necessity:

If there exists $p \in \mathcal{S}(\boldsymbol{a})$ such that

$$\left| \bigcup_{k=1}^{K} \bigcup_{i \neq j} [\mathcal{D}(p, \boldsymbol{a}, \boldsymbol{d}_k, \boldsymbol{\delta}_i) \cap \mathcal{D}(p, \boldsymbol{a}, \boldsymbol{d}_k, \boldsymbol{\delta}_j)] \right| \geqq 2, \quad (17)$$

it implies that at least one of the three cases holds.

(a) There exist k, i, $i' \neq i$, and P_α, $P_\beta \in \mathcal{R}(\boldsymbol{a})$, $P_\alpha \neq P_\beta$, such that both P_α and P_β are dominated by p with respect to both $P(\boldsymbol{a} - \boldsymbol{d}_k, \boldsymbol{\delta}_i)$ and $P(\boldsymbol{a} - \boldsymbol{d}_k, \boldsymbol{\delta}_{i'})$ (Fig. 9 (a)).
In this case, when p is faulty, both P_α and P_β receive at least two erroneous k-th inputs.

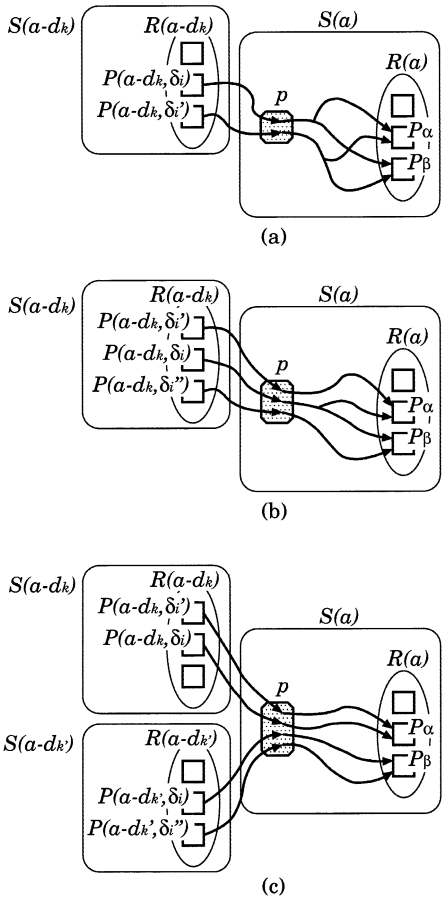

Fig. 9 Explanatory figures for necessity proof of Lemma 2.

(b) There exist k, $\{i, i', i''\} = \{0, 1, 2\}$, and P_α, $P_\beta \in \mathcal{R}(\boldsymbol{a})$, $P_\alpha \neq P_\beta$, such that P_α is dominated by p with respect to both $P(\boldsymbol{a} - \boldsymbol{d}_k, \boldsymbol{\delta}_i)$ and $P(\boldsymbol{a} - \boldsymbol{d}_k, \boldsymbol{\delta}_{i'})$ and at the same time P_β is dominated by p with respect to both $P(\boldsymbol{a} - \boldsymbol{d}_k, \boldsymbol{\delta}_i)$ and $P(\boldsymbol{a} - \boldsymbol{d}_k, \boldsymbol{\delta}_{i''})$ (Fig. 9 (b)).
In this case, when p is faulty, both P_α and P_β receive at least two erroneous k-th inputs.

(c) There exist k, $k' \neq k$, $\{i, i', i''\} = \{0, 1, 2\}$, and P_α, $P_\beta \in \mathcal{R}(\boldsymbol{a})$, $P_\alpha \neq P_\beta$, such that P_α is is dominated by p with respect to both $P(\boldsymbol{a} - \boldsymbol{d}_k, \boldsymbol{\delta}_i)$ and $P(\boldsymbol{a} - \boldsymbol{d}_k, \boldsymbol{\delta}_{i'})$ and at the same time P_β is dominated by p with respect to both $P(\boldsymbol{a} - \boldsymbol{d}_{k'}, \boldsymbol{\delta}_i)$ and $P(\boldsymbol{a} - \boldsymbol{d}_{k'}, \boldsymbol{\delta}_{i''})$ (Fig. 9 (c)).
In this case, when p is faulty, P_α receives at least two erroneous k-th inputs, and P_β receives at least two erroneous k'-th inputs.

In every case among the above, when p becomes faulty, the computation results of P_α and P_β are both erroneous, and the stage $\mathcal{S}(\boldsymbol{a})$ becomes unfair.

Sufficiency:

The condition (16) implies that, for any single resource p, at least two elements in $\mathcal{R}(\boldsymbol{a})$ are not contained in the set;

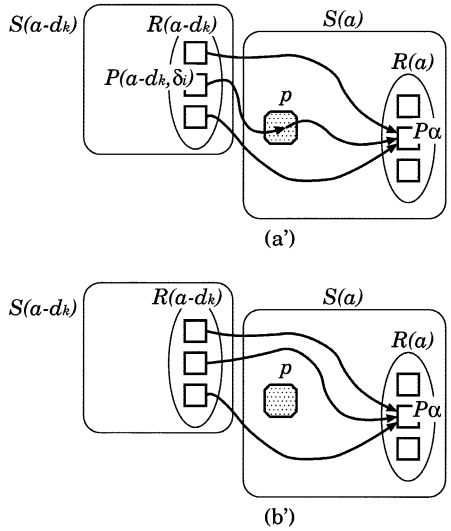

Fig. 10 Explanatory figures for sufficiency proof of Lemma 2.

$$\mathcal{D}_{\cup}(p,\boldsymbol{a}) \stackrel{\triangle}{=} \bigcup_{k=1}^{K} \bigcup_{i \neq j} [\mathcal{D}(p,\boldsymbol{a},\boldsymbol{d}_k,\boldsymbol{\delta}_i) \cap \mathcal{D}(p,\boldsymbol{a},\boldsymbol{d}_k,\boldsymbol{\delta}_j)]$$

Let $P_\alpha \in \mathcal{R}(\boldsymbol{a})$ be a such PE for a certain p, i.e. $P_\alpha \notin \mathcal{D}_{\cup}(p,\boldsymbol{a})$.

(a′) When there exist k and i such that $P_\alpha \in \mathcal{D}(p,\boldsymbol{a},\boldsymbol{d}_k,\boldsymbol{\delta}_i)$, then

$$P_\alpha \notin \mathcal{D}(p,\boldsymbol{a},\boldsymbol{d}_k,\boldsymbol{\delta}_j),\ j \in \{0,1,2\}\backslash\{i\}$$

As a result, even if p is faulty, P_α can receive two correct redundant k-th inputs without passing through p, and it can recover the correct k-th input (Fig. 10(a′)).

(b′) When $P_\alpha \notin \mathcal{D}(p,\boldsymbol{a},\boldsymbol{d}_k,\boldsymbol{\delta}_i)$ for all $i, 0 \leqq i \leqq 2$ for some $k, 1 \leqq k \leqq K$, it is trivial that, even if p is faulty, P_α can receive three correct redundant k-th inputs (Fig. 10(b′)).

Since either (a′) or (b′) holds for any resource p, even if any single fault occurs in $\mathcal{S}(\boldsymbol{a})$, at least two elements in $\mathcal{R}(\boldsymbol{a})$ can yield correct outputs.

End of proof.

Theorem 1: If the condition given by Eq. (16) holds for every stage in a TMR system, then the entire TMR system is tolerable for any single fault within each two successive stages (in the sense that the sum of the number of faults in $\mathcal{S}(\boldsymbol{a})$ and the number of faults in $\mathcal{S}(\boldsymbol{a}-\boldsymbol{d}_k)$ equals to one).

Proof of Theorem 1:
We will use the fact that the original computation algorithm to be implemented is given by the uniform recurrence equations.

Suppose that the entire system is not normal. Let $\mathcal{S}(\boldsymbol{a})$ be an unfair stage and every one of its preceding stages and ancestor stages is either fair, marginally fair or triplicated primary input (treated as to be fair). Such $\mathcal{S}(\boldsymbol{a})$ always exist, since the TDG under consideration does not contain directed loops. If the stage $\mathcal{S}(\boldsymbol{a})$ is fault free, then at least one of the stages of $\mathcal{S}(\boldsymbol{a})$ should be unfair. However it contradicts to the choice of $\mathcal{S}(\boldsymbol{a})$. If the stage $\mathcal{S}(\boldsymbol{a})$ contains a single fault, then every preceding stage of $\mathcal{S}(\boldsymbol{a})$ are fault-free from the condition of Theorem 1. Together with the assumption that every immediate preceding stage of each immediate preceding stage of $\mathcal{S}(\boldsymbol{a})$ is either fair or marginally fair, every immediate preceding stage of $\mathcal{S}(\boldsymbol{a})$ is fair (Lemma 1). From Lemma 2, the stage $\mathcal{S}(\boldsymbol{a})$ should be either fair or marginally fair, and it contradicts the choice of $\mathcal{S}(\boldsymbol{a})$. As a result, the proposition "the entire system is not normal" is denied.

End of proof.

Note that, if $p \in \mathcal{S}(\boldsymbol{a}) \cap \mathcal{S}(\boldsymbol{a}-\boldsymbol{d}_k)$ exists, faulty p is a single fault for $\mathcal{S}(\boldsymbol{a})$ and also a single fault for the immediate preceding stage $\mathcal{S}(\boldsymbol{a}-\boldsymbol{d}_k)$, and we can not apply Theorem 1 to this case. In other words, Theorem 1 does not allow the sharing of hardware resource (PEs and links) between two successive stages.

Lemma 3: When every preceding stage of the stage $\mathcal{S}(\boldsymbol{a})$ is either fair or marginally fair, and

$$\forall p \in \mathcal{S}(\boldsymbol{a}), \left| \bigcup_{k=1}^{K} \bigcup_{i=0}^{2} \mathcal{D}(p,\boldsymbol{a},\boldsymbol{d}_k,\boldsymbol{\delta}_i) \right| \leqq 1 \qquad (18)$$

holds, then, for any single fault within $\mathcal{S}(\boldsymbol{a})$, the stage $\mathcal{S}(\boldsymbol{a})$ is either fair or marginally fair.

Proof of Lemma 3:
For any element $p \in \mathcal{S}(\boldsymbol{a})$, at least two elements P_α, P_β in $\mathcal{R}(\boldsymbol{a})$ can receive all their inputs without passing through p. Then at least two elements in $\mathcal{R}(\boldsymbol{a})$ yield correct outputs for any single fault within $\mathcal{S}(\boldsymbol{a})$ even if some of immediate preceding stages are marginally fair.

End of proof.

Theorem 2: If the condition given by Eq. (18) holds for every stage in the TMR system, then the entire TMR system is tolerable for any single fault within each stage.

Proof of Theorem 2:
The proof can be done with similar way to the one for Theorem 1.

Considering the entire system is not normal, let $\mathcal{S}(\boldsymbol{a})$ be an unfair stage and every one of its immediate preceding stages and ancestor stages is either fair, marginally fair or primary input. Such $\mathcal{S}(\boldsymbol{a})$ always exists when the entire system is not normal. However, from Lemma 3, $\mathcal{S}(\boldsymbol{a})$ can not be unfair, and it contradicts that the entire system is not normal.

End of proof.

From Theorems 1 and 2, we can see the tradeoff between the limitation on resource sharing and tolerable fault patterns, i.e., the set of feasible implementa-

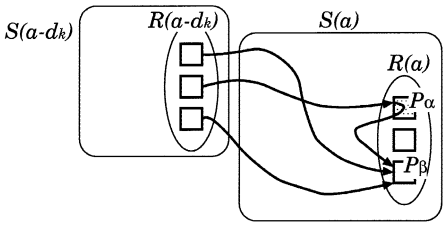

Fig. 11 Explanatory figure for proof of Lemma 4.

tions under Theorem 2 (Eq. (18)) is a subset of the one under Theorem 1 (Eq. (16)), while the set of fault patterns tolerable under Theorem 2 is a superset of the one under Theorem 1. In this sense, the following lemma and theorem offer another pair of the limitation on resource sharing and tolerable fault patterns, which is in between Theorem 1 and Theorem 2.

Lemma 4: When every preceding stage is fair, and

$$\left|\bigcup_{k=1}^{K}\bigcup_{i\neq j}[\mathcal{D}(p,\boldsymbol{a},\boldsymbol{d}_k,\boldsymbol{\delta}_i)\cap\mathcal{D}(p,\boldsymbol{a},\boldsymbol{d}_k,\boldsymbol{\delta}_j)]\right| = \begin{cases}0:\ \forall p\in\mathcal{S}(\boldsymbol{a})\backslash\mathcal{R}(\boldsymbol{a})\\ 1:\ \forall p\in\mathcal{R}(\boldsymbol{a})\end{cases} \tag{19}$$

holds, then the stage $\mathcal{S}(\boldsymbol{a})$ is fair for any single fault within $\mathcal{S}(\boldsymbol{a})\backslash\mathcal{R}(\boldsymbol{a})$ and it is marginally fair for any single fault within $\mathcal{R}(\boldsymbol{a})$.

Proof of Lemma 4:
For $p\in\mathcal{S}(\boldsymbol{a})\backslash\mathcal{R}(\boldsymbol{a})$, $P_\alpha\in\mathcal{R}(\boldsymbol{a})$ and k; if there exists i such that $\mathcal{D}(p,\boldsymbol{a},\boldsymbol{d}_k,\boldsymbol{\delta}_i)$ contains $P_\alpha\in\mathcal{R}(\boldsymbol{a})$, then $P_\alpha\notin\mathcal{D}(p,\boldsymbol{a},\boldsymbol{d}_k,\boldsymbol{\delta}_j)$ with respect to $j\in\{0,1,2\}\backslash\{i\}$. Otherwise $P_\alpha\notin\mathcal{D}(p,\boldsymbol{a},\boldsymbol{d}_k,\boldsymbol{\delta}_j)$ with respect to $j\in\{0,1,2\}$. It means that any $P_\alpha\in\mathcal{R}(\boldsymbol{a})$ can receive at least two k-th inputs without passing through p (refer to Fig. 10), and it can have correct k-th input even if p is faulty. As a result, every element in $\mathcal{R}(\boldsymbol{a})$ can have K correct inputs and can yield correct outputs, i.e., the stage $\mathcal{S}(\boldsymbol{a})$ is fair, for any single fault within $\mathcal{S}(\boldsymbol{a})\backslash\mathcal{R}(\boldsymbol{a})$.

Next we will consider the case $p=P_\alpha\in\mathcal{R}(\boldsymbol{a})$. Note that P_α is contained $\mathcal{D}(P_\alpha,\boldsymbol{a},\boldsymbol{d}_k,\boldsymbol{\delta}_i)$ for every possible i and k. For any k, if $P_\beta\in\mathcal{R}(\boldsymbol{a})$, $P_\beta\neq P_\alpha$, is contained in $\mathcal{D}(P_\alpha,\boldsymbol{a},\boldsymbol{d}_k,\boldsymbol{\delta}_i)$ with certain i, then it is not contained in $\mathcal{D}(P_\alpha,\boldsymbol{a},\boldsymbol{d}_k,\boldsymbol{\delta}_j)$ with $j\in\{0,1,2\}\backslash\{i\}$ (Fig. 11). Otherwise $P_\beta\notin\mathcal{D}(P_\alpha,\boldsymbol{a},\boldsymbol{d}_k,\boldsymbol{\delta}_j)$ for j, $j\in\{0,1,2\}$. In both cases, P_β can receive at least two k-th inputs without passing through P_α. Totally, every element in $\mathcal{R}(\boldsymbol{a})$ can receive at least two k-th inputs without passing through any one of the other two elements in $\mathcal{R}(\boldsymbol{a})$, and hence it can have correct k-th input even if one of the other two element in $\mathcal{R}(\boldsymbol{a})$ is faulty. As a result, even if one of the elements in $\mathcal{R}(\boldsymbol{a})$ is faulty, the other two elements can yield correct outputs, i.e., the stage $\mathcal{S}(\boldsymbol{a})$ is marginally fair.

End of proof.

Theorem 3: If the condition given by Eq. (19) holds for every stage in a TMR system, then the entire TMR system is tolerable for any single fault within a stage $\mathcal{S}(\boldsymbol{a})$ and its preceding $\mathcal{R}(\boldsymbol{a}-\boldsymbol{d}_k)$, in the sense that the number of faulty elements in $\mathcal{S}(\boldsymbol{a})$ plus the number of faulty elements in $\mathcal{R}(\boldsymbol{a}-\boldsymbol{d}_k)$ equals to 1.

Proof of Theorem 3:
Suppose that the entire system is not normal. Let $\mathcal{S}(\boldsymbol{a})$ be an unfair stage and every one of its immediate preceding stages and ancestor stages is either fair, marginally fair or primary input (treated as to be fair).

If all immediate preceding stages of the stage $\mathcal{S}(\boldsymbol{a})$ are fair or primary inputs, the stage $\mathcal{S}(\boldsymbol{a})$ should be either fair or marginally fair for any single fault within $\mathcal{S}(\boldsymbol{a})$, which contradicts the proposition.

When an immediate preceding stage of $\mathcal{S}(\boldsymbol{a})$, say $\mathcal{S}(\boldsymbol{a}-\boldsymbol{d}_{k_1})$, is marginally fair, we can find a sequence of marginally fair stages $\mathcal{S}_1=\mathcal{S}(\boldsymbol{a}-\boldsymbol{d}_{k_1})$, $\mathcal{S}_2=\mathcal{S}((\boldsymbol{a}-\boldsymbol{d}_{k_1})-\boldsymbol{d}_{k_2})$, $\cdots$, $\mathcal{S}_R=\mathcal{S}((\boldsymbol{a}-\boldsymbol{d}_{k_1}-\cdots-\boldsymbol{d}_{k_{R-1}})-\boldsymbol{d}_{k_R})$, where $R\geqq 1$, $\mathcal{S}_r$ is an immediate preceding stage of $\mathcal{S}_{r-1}$, and immediate preceding stages of $\mathcal{S}_R$ are fair stages and primary inputs. For the consistency purpose, we let $\mathcal{S}_0=\mathcal{S}(\boldsymbol{a})$. From Lemma 4, $\mathcal{R}(\boldsymbol{a}-\boldsymbol{d}_{k_1}-\cdots-\boldsymbol{d}_{k_R})$ should contain single fault. From the limitation of fault occurrence given in Theorem 3, $\mathcal{S}_{R-1}$ should be fault-free. Furthermore each of immediate preceding stages of $\mathcal{S}_{R-1}$ is either fair or marginally fair. As a result, $\mathcal{S}_{R-1}$ should be fair, which contradicts the proposition for any value of $R(\geqq 1)$.

End of proof.

We have obtained three theorems on the relation between data transfer routing and fault tolerance property, each of which provides a sufficient condition on data transfer routing for "any" single fault within a certain range being tolerable.

Unfortunately, applications of Theorem 1 and Theorem 3 are seriously restricted, since resource sharing between successive stages is not allowed. Resource sharing between successive stages may occur depending on (1) the selection of the space transformation matrix $\boldsymbol{P}$ (affects PE sharing) and (2) data transfer routing scheme (affects PE sharing and link sharing). It is preferable, for preserving high utilization of PEs, to select the space transformation matrix so that PEs are shared by successive stages, as link the design example demonstrated in Fig. 3. So we need to investigate further into Theorem 1 and Theorem 3 to make them widely applicable to various designs with less limitations on the selection of the space transformation matrix.

5. Examples of Network Configuration

Here we demonstrate some data transfer routing schemes for TMR version of matrix multiplication shown in Sect. 3. Since, as we have mentioned before, the space transformation matrix used for that systolic

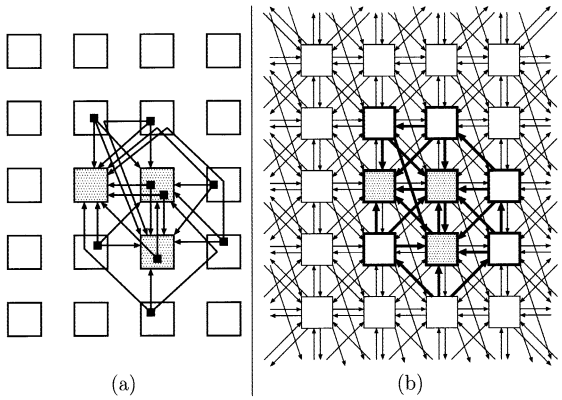

Fig. 12 Routing and network architecture-II. (a) Set of routing for communication vectors. (b) Overall network architecture.

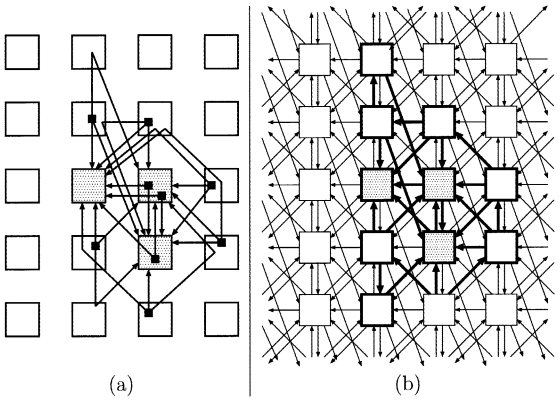

Fig. 13 Routing and network architecture-III. (a) Set of routing for communication vectors. (b) Overall network architecture.

array induces PE sharing between successive stages, Theorem 2 is applied for network configurations shown in the following. Note again that the scheduling aspect of data transfer is not discussed throughout this paper.

First of all, we will check the direct implementation (network architecure-I) shown in Fig. 5. In this implementation, every data transfer is realized by a direct physical link from source PE to destination PE, and the problem is whether some class of fault tolerant property is preserved or not when all data transfers with a common source PE and an common destination PE are achieved by a single physical link. Theorem 2 gives us a positive answer, that is, any single fault within a single stage is tolerable, where a single stage $\mathcal{S}(\boldsymbol{a})$ in this case consists of the set of three PEs $\mathcal{R}(\boldsymbol{a})$ and the set of physical links as;

$$\{(P(\boldsymbol{a}-\boldsymbol{d}_k,\boldsymbol{\delta}_i),P(\boldsymbol{a},\boldsymbol{\delta}_j))| \\ k\in\{1,2,3\},i\in\{0,1,2\},j\in\{0,1,2\}, \\ P(\boldsymbol{a}-\boldsymbol{d}_k,\boldsymbol{\delta}_i)\neq P(\boldsymbol{a},\boldsymbol{\delta}_j)\}$$

Resources within a single stage are indicated by gray boxes (PEs in a $\mathcal{R}(\boldsymbol{a})$) and bold allows (physical links) in Fig. 5. Since, in this implementation, each PE in a stage $\mathcal{S}(\boldsymbol{a})$ dominates only itself and each physical link in the stage dominates only one PE in $\mathcal{R}(\boldsymbol{a})$, the condition Eq. (18) holds and hence Theorem 2 holds.

Figure 12 and Fig. 13 show two sets of possible routing for logical links with communication vector decompositions (a), both of which meet the condition in Theorem 2, and their corresponding final network architectures ((b): network architecture-II and network architecture-III). In (b) of these figures, gray boxes denote elements in $\mathcal{R}(\boldsymbol{a})$, and bold boxes (PEs) and bold arrows (links) together with gray boxes denote resources in a stage $\mathcal{S}(\boldsymbol{a})$ for some value of $\boldsymbol{a}$. Note that a bold white box plays only the role to relay data transfer in this certain stage $\mathcal{S}(\boldsymbol{a})$, while it may execute node operation in other stage with different value of $\boldsymbol{a}$.

Table 2 summarizes some numerical features of those three networks. From this table, we can see the tendency that, when we try to reduce the number of links per PE, the communication vector decomposition should be applied aggressively which results in larger number of PEs and larger number of PE-to-PE data transfers contained in a stage. As a result, since a stage is a set of resources in which "any" single fault is guaranteed to be tolerable (by Theorem 2), the smaller the number of links per PE is, the lower the reliability in a statistical measure is expected to be.

Another appearance of such tendency is shown in Fig. 14 which illustrates the range in which a second faulty PE is not guaranteed to be tolerated (from The-

Table 2 Numerical comparison between different network configurations.

Network	I	II	III
Communication vector decomposition	No	Yes	Yes
# of outgoing edges per PE	14	9	7
# of PEs in a stage	3	8	10
# of links in a stage	22	22	24
# of PE-to-PE data transfers in a stage	24	30	32

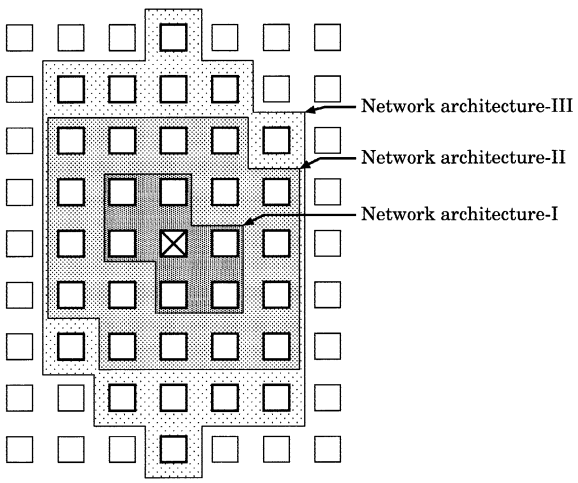

Fig. 14 Different ranges in which a second faulty PE is not guaranteed to be tolerated depending on the data routing scheme.

orem 2) under the existence of a faulty (permanent) PE marked by a cross. Network architecure-I has the smaller such range than network architecure-II or III has, which indicates that network architecure-I tolerates the denser faulty PE distribution than network architecure-II or III does.

While our configurations are guaranteed to tolerate any single fault in every stage, the tolerance (or intolerance) to two or more faults in each stage depends on a fault pattern for the entire array system, and its analysis is not yet developed. So, the exact evaluation of the reliability in statistical measure remains as a future work, and we will evaluate reliability lower bounds based only on single-fault/stage tolerance property.

Now we consider a matrix multiplication of $L \times M$ matrix A and $M \times N$ matrix B, which is implemented on $L \times (M+N-1)$ PE array (nominal version) and $(L+1) \times (M+N)$ PE array (TMR version). We assume that PE fault and link fault occur randomly, and we let p and q are reliabilities of a single PE and a single physical link, respectively. For simplicity, we will use $n = L \times (M+N-1)$ and $n' = (L+1) \times (M+N)$.

In a nominal (non-fault-tolerant) implementation (remove all redundant computations and data transfers (redundant PEs and physical links) from Fig. 5, for details please refer to Fig. 5 in [15]), each PE has 3 incoming physical links, and we treat a PE and its incoming physical links as a unit to compute reliability. It is clear that the reliability of a single unit is given as $r_0 = p \cdot q^3$, and the overall reliability R_{nominal} is given as,

$$R_{\text{nominal}} = {r_0}^n = (pq^3)^n$$

With respect to TMR version with network architecture-I, similar to the previous one, we treat a PE and its incoming physical links as a unit, and its reliability is given as $r_1 = p \cdot q^{14}$. As we have shown in Fig. 14, when a unit (a PE and its incoming links) becomes faulty, 6 other units around it are not allowed to be faulty (tolerance is not guaranteed). Hence the overall reliability $R_{\text{TMR-1}}$ is given as,

$$\begin{aligned} R_{\text{TMR-I}} &\geqq r_1^{n'} \\ &\quad + n' r_1^{n'-1}(1-r_1) \\ &\quad + \frac{n'(n'-7)}{2!} r_1^{n'-2}(1-r_1)^2 \\ &\quad \vdots \\ &\quad + \frac{n'(n'-7)\cdots(n'-7(k-1))}{k!} r_1^{n'-k}(1-r_1)^k \\ &\quad \vdots \\ &\triangleq R_{1\text{L}} \end{aligned} \tag{20}$$

For simplicity, we assume that n' can be divided by 7 evenly. Then,

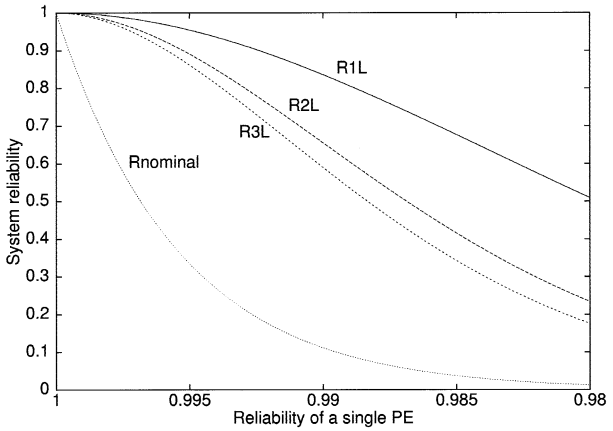

Fig. 15 Reliability comparison. R_{nominal}, $R_{1\text{L}}$, $R_{2\text{L}}$ and $R_{3\text{L}}$ denote overall reliabilities of nominal implementation, TMR with network architecture-I, TMR with network architecture-II and TMR with network architecture-III, respectively.

$$\begin{aligned} &R_{1\text{L}} \\ &= \sum_{k=0}^{n'/7} \frac{n'(n'-7)\cdots(n'-7(k-1))}{k!} r_1^{n'-k}(1-r_1)^k \\ &= r_1^{6n'/7} \sum_{k=0}^{n'/7} {}_{n'/7}C_k 7^k r_1^{(n'/7-k)}(1-r_1)^k \\ &= r_1^{6n'/7} \left(r_1 + 7(1-r_1)\right)^{n'/7} \\ &= r_1^{6n'/7} \left(7-6r_1\right)^{n'/7} \end{aligned} \tag{21}$$

Similarly, the lower bounds of the overall reliability for TMR with network architecture-II ($R_{2\text{L}}$) and for TMR with network architecture-III ($R_{3\text{L}}$) are given using $r_2 = p \cdot q^9$ and $r_3 = p \cdot q^7$ as,

$$R_{2\text{L}} = r_2^{22n'/23} \left(23 - 22r_2\right)^{n'/23} \tag{22}$$

$$R_{3\text{L}} = r_3^{34n'/35} \left(35 - 34r_3\right)^{n'/35} \tag{23}$$

Originally, the success of highly integrated systems will be supported by reliable components, and the overall reliability at near around $p = 1$, $q = 1$ may be an important measure. To evaluate and to compare such values for network architecture-I, II and III, the second order approximation on Taylor expansion around $p = 1$, $q = 1$ will be adopted, and its results are given as follows.

$$R_{1\text{L}} \approx 1 - 3n' \left((1-p) + 14(1-q)\right)^2 \tag{24}$$

$$R_{2\text{L}} \approx 1 - 11n' \left((1-p) + 9(1-q)\right)^2 \tag{25}$$

$$R_{3\text{L}} \approx 1 - 17n' \left((1-p) + 7(1-q)\right)^2 \tag{26}$$

We can see $R_{1\text{L}} > R_{2\text{L}} > R_{3\text{L}}$ under the situation where the reliability of a PE is much lower than that of a physical link, while $R_{1\text{L}} > R_{3\text{L}} > R_{2\text{L}}$ may arise when a physical link is relatively unreliable.

Finally of this section, the numerical comparison of reliabilities R_{nominal}, $R_{1\text{L}}$, $R_{2\text{L}}$ and $R_{3\text{L}}$ under

$L = M = N = 10$ is shown in Fig. 15. While the graph is drawn with its x-axis p for convenience, we have assumed in those computations that the failure rate of a single link is 1/20 times smaller than the one of a single PE, i.e., $(1 - q) = (1 - p)/20$. From this figure, we can verity numerically the tendency: the smaller the number of links per PE is by resource sharing, the lower the overall reliability is.

6. Conclusions

In this paper, relations between data transfer routing/resource-sharing and fault tolerant property in TMR systolic array systems are investigated, and three theorems have been obtained. It is notable that these three theorems line up in the tradeoff between limitation of data transfer routing/resource-sharing and tolerable fault patterns. We can find another type of design tradeoff from design examples (network architecture-I, II and III) based on Theorem 2; the smaller the number of links per PE is, the lower the reliability in a statistical measure becomes. It is because to reduce the number of links per PE needs aggressive communication vector decomposition, which results in a TMR system in which each stage (a set of resources in which a single fault is tolerated by Theorem 2) contains a larger number of PEs and PE-to-PE data transfers.

While Theorem 2 is applicable to variety of systolic operations with various selections of the space transformation matrix and the timing schedule function, applications of Theorem 1 and Theorem 3 are seriously restricted, since resource sharing between successive stages is not allowed. Also discussions in this paper including these three theorems assume Assumption 1 and Assumption 2, which are somewhat pessimistic as a model of error generation/propagation. The study on the fault tolerance under more precise model of the correctness of computation results is a future problem. Further discussion together with the scheduling problem of data transfer toward physical link minimization and tighter evaluation of overall system reliability are also left for future works.

Acknowledgement

This work is supported in part by the Ministry of ESSC, Japan under Grant-in-aid for Scientific Research No.09450158 and by the Research Body CAD21, Tokyo Institute of Technology, Japan.

References

[1] H.T. Kung, "Why systolic architectures?" IEEE Trans. Comput., vol.c-31, no.1, pp.37–46, 1982.
[2] S.Y. Kung, "On supercomputing with systolic/wavefront array processors," Proc. IEEE, vol.72, no.7, pp.867–884, 1984.
[3] P. Quinton, "Automatic synthesis of systolic arrays from uniform recurrent equations," Proc. IEEE 11th Int. Sym. on Computer Architecture, pp.208–214, 1984.
[4] D.I. Moldovan, "On the design of algorithms for VLSI systolic arrays," Proc. IEEE, vol.71, no.1, pp.113–120, 1983.
[5] S.-J. Jou and C.-W. Jen, "Design of a systolic array system for linear state equations," IEE Proc., vol.135, Pt.G, no.5, pp.211–218, 1988.
[6] J.H. Kim and S.M. Reddy, "A fault-tolerant systolic array design using TMR method," Proc. IEEE Int. Conf. Computer Design, pp.769–773, 1985.
[7] H.T. Kung and M.S. Lam, "Fault-tolerance and two-level pipelining in VLSI systolic arrays," MIT Conference on ADV. Research in VLSI, pp.74–83, 1984.
[8] R.J. Cosentino, "Concurrent error correction in systolic architecture," IEEE Trans. Comput.-Aided Des. Integrated, vol.7, no.1, pp.117–125, 1988.
[9] K.-H. Huang and J.A. Abraham, "Algorithm-based fault tolerance for matrix operations," IEEE Trans. Comput., vol.c-33, no.6, pp.518–528, 1984.
[10] J.-Y. Jou and J.A. Abraham, "Fault-tolerant matrix arithmetic and signal processing on highly concurrent computing structures," Proc. IEEE, vol.74, no.5, pp.732–741, 1986.
[11] B. Vinnakota and N.K. Jha, "Synthesis of algorithm-based fault-tolerant systems from dependence graphs," IEEE Trans. Parallel and Distributed Systems, vol.4, no.8, pp.864–874, 1993.
[12] C.M. Liu and C.W. Jen, "Design of algorithm-based fault-tolerant VLSI array processor," IEE Proc., vol.139, Pr.E, no.6, pp.539–547, 1989.
[13] H.F. Li, R. Jayakumar, and C. Lam, "Restructuring for fault-tolerant systolic arrays," IEEE Trans. Comput., vol.38, no.2, pp.307–311, 1989.
[14] C.W.H. Lam, H.F. Li, and R. Jayakumar, "A study of two approaches for reconfiguring fault-tolerant systolic arrays," IEEE Trans. Comput., vol.38, no.6, pp.833–844, 1989.
[15] M. Kaneko and H. Miyauchi, "A systematic design of fault tolerant systolic arrays based on triple modular redundancy in time-processor space," IEICE Trans. Inf.& Syst., vol.E79-D, no.12, pp.1676–1689, Dec. 1996.
[16] M. Kaneko, H. Miyauchi, and C.-S. Park, "Link sharing for fault tolerant systolic arrays based on mixed spatial-temporal triple modular redundancy," Proc. IEEE APCCAS'96, pp.472–475, 1996.

Mineo Kaneko received the B.E., M.E. and Dr.E. degrees in Electrical and Electronic Engineering, from Tokyo Institute of Technology, Tokyo, Japan, in 1981, 1983, and 1986, respectively. From 1986 to 1996, he was with the Department of Electrical and Electronic Engineering, Tokyo Institute of Technology. Currently, he is a Professor in the School of Information Science, Japan Advanced Institute of Science and Technology. His research interests include circuit theory and CAD for VLSIs, fault tolerance through algorithm-level redundancy and/or reconfiguration and analog & digital signal processing. He has received best paper awards from IEEE Asia-Pacific Conference on Circuits and Systems in 1992 and 1994. He is a member of IEEE.