CMU-CS-83-166

Fault-Tolerance and Two-Level Pipelining in VLSI Systolic Arrays

H. T. Kung and Monica S. Lam

Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

November 1983

To appear in *Proceedings of the Conference on Advanced Research in VLSI*, MIT, January 1984.

This research was supported in part by the Office of Naval Research under Contracts N00014-76-C-0370, NR 044-422 and N00014-80-C-0236, NR 048-659, and in part by the Defense Advanced Research Projects Agency (DoD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory under Contract F33615-81-K-1539.

Abstract

This paper addresses two important issues in systolic array designs: fault-tolerance and two-level pipelining. The proposed "systolic" fault-tolerant scheme maintains the original data flow pattern by bypassing defective cells with a few registers. As a result, many of the desirable properties of systolic arrays (such as local and regular communication between cells) are preserved. Two-level pipelining refers to the use of pipelined functional units in the implementation of systolic cells. This paper addresses the problem of efficiently utilizing pipelined units to increase the overall system throughput. We show that both of these problems can be reduced to the same mathematical problem of incorporating extra delays on certain data paths in originally correct systolic designs. We introduce the mathematical notion of a *cut* which enables us to handle this problem effectively.

The results obtained by applying the techniques described in this paper are encouraging. When applied to systolic arrays without feedback cycles, the arrays can tolerate large numbers of failures (with the addition of very little hardware) while maintaining the original throughput. Furthermore, all of the pipeline stages in the cells can be kept fully utilized through the addition of a small number of delay registers. However, adding delays to systolic arrays with cycles typically induces a significant decrease in throughput. In response to this, we have derived a new class of systolic algorithms in which the data cycle around a ring of processing cells. The *systolic ring* architecture has the property that its performance degrades gracefully as cells fail. Using our cut theory for arrays without feedback and the ring architecture approach for those with feedback, we have effective fault-tolerant and two-level pipelining schemes for most systolic arrays.

As a side-effect of developing the ring architecture approach we have derived several new systolic algorithms. These algorithms generally require only one-third to one-half of the number of cells used in previous designs to achieve the same throughput. These new systolic algorithms include ones for LU-decomposition, QR-decomposition and the solution of triangular linear systems.

Table of Contents

List of Figures

1. Introduction

The progression towards increasingly large and complex integrated circuits has been accompanied by smaller device geometries and larger dies, which in turn have led to a decrease in integrated circuit yield. A strategy for increasing the yield involves the design of integrated circuits whose correctness does not require 100 percent of the constituent circuits to be correct. Such approaches fall under the heading of "fault-tolerant" or "restructurable" techniques, and are typically characterized by the inclusion of redundant functional elements and the ability to modify the interconnection structure of the constituent elements. These techniques are particularly important, and frequently applied, in the area of wafer scale integration. A number of different techniques exist for modifying the interconnection structure of integrated circuits. They range from static, pre-packaging approaches (e.g., adding a layer of metalization, laser created/deleted connections) to more dynamic approaches that can be applied after packaging (e.g., fusible links, transistor switching devices)[1].

Fault-tolerant methods are particularly important to systolic array implementations. A unique property of the systolic approach is that as the number of cells grows, the system performance increases proportionally. Thus it is desirable for a systolic array to have as many cells as a given problem can effectively utilize. However, when the number of cells is large, it is inevitable that some of them may fail. Therefore it is important that the systolic arrays be designed to function correctly in spite of the fact that some cells may not (see Figure 1-1 (a)). This paper addresses the problem of how to tolerate the defects once they are located. The fault detection problem, requiring a totally different set of techniques such as voting and self-testing, is beyond the scope of this paper.

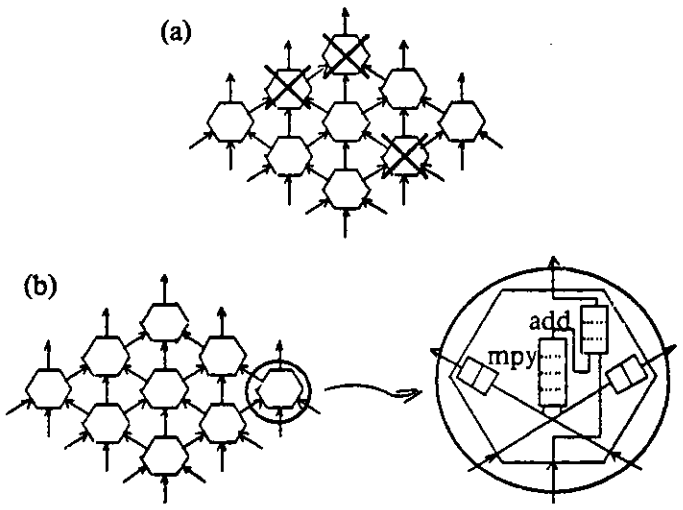

Figure 1-1: Two problems addressed in the paper: (a) fault-tolerance for arrays with faulty cells and (b) two-level pipelining

High throughput floating-point multiplier and adder circuits typically employ three or more pipeline stages[2]. Systolic cells implemented using these units form a *second level of pipelining*[3] in the pipelined organization of systolic arrays (see Figure 1-1 (b)). This additional level of pipelining can greatly increase the system throughput; it is therefore important to be able to systematically transform existing systolic array designs assuming single-stage cells to ones with pipelined cells.

We will show that both the fault-tolerance and the two-level pipelining problems can be solved by the same mathematical reasoning and techniques. Our results imply that once a "generic" systolic algorithm is designed, other versions of the algorithm (for execution on arrays with failed cells, or for implementation using different pipelined processing units) can be systematically derived. The techniques of this paper can also be applied to other computation structures, such as FFT processor arrays and parallel sorting processors.

In the next section we will introduce our approach, using as an example the simplest type of systolic arrays (uni-directional linear arrays). We will discuss our solutions for all systolic arrays without feedback in section 3, and then for those with feedback in section 4. Section 5 includes a summary and some concluding remarks.

2. Fault-Tolerance and Two-Level Pipelining for Uni-directional Linear Arrays

Figure 2-1 depicts a systolic array[4] for the convolution computation with four weights $w_1, \ldots, w_4$. In this array the data flow only in *one* direction, that is, both x_i and y_i move from left to right (with x_i going through an additional "delay register" following each cell). This is an example of a systolic array without feedback cycles—an array where none of the values in any data stream depends on the preceding values in the same stream. (For an example of an array with feedback cycles, see Figure 4-1 (a)).

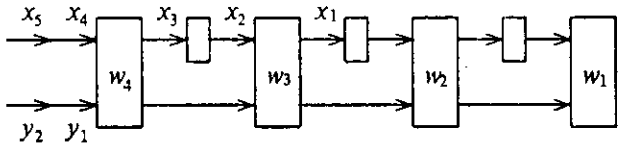

Figure 2-1: Uni-directional linear systolic array for convolution

Suppose that the third cell from the left in the array of Figure 2-1 were to fail. As depicted in Figure 2-2 (a), we would replace the defective cell with two "bypass" registers (shown in dotted lines)—one for the x-data stream, one for the y-data stream. To solve a problem of the same size, the defective array must have one more cell to compensate for the failed cell. It can easily be shown that the new array correctly solves the same problem at the original computational rate of one output per cell cycle. For example, y_1 picks up $w_4 \cdot x_4$, $w_3 \cdot x_3$ and $w_2 \cdot x_2$ at the first, second and fourth cell respectively. The only difference is that the latency of the solution is increased by one cycle. Figure 2-2 (b) depicts the cell specification for this fault-tolerant scheme, using reconfigurable links. Note that the input/output register in a systolic cell can be used as a bypass register in case the cell fails. Therefore no extra registers are needed to implement this fault-tolerant scheme.

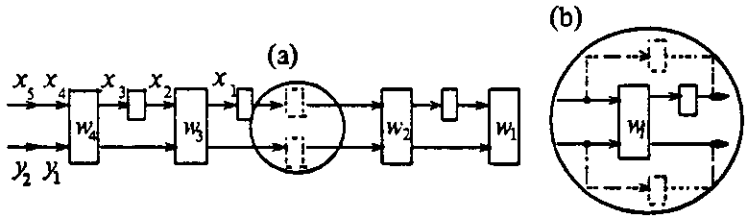

Figure 2-2: (a) Defective cell replaced with registers and (b) cell specification

A basic assumption of this paper is that the probability of the interconnection links and registers failing is negligible. This is reasonable because these components are typically much simpler and smaller than the cells themselves and can be implemented conservatively.

In the proposed scheme data move through all the cells. At failed cells, data items are simply delayed with

bypass registers for one cycle, and no computation is performed (Figure 2-3 (a)). We refer to fault-tolerant schemes of this type as *systolic* fault-tolerant schemes in view of the fact that data travel *systolically* in a defective array from cell to cell, at the original clock speed.

For uni-directional linear arrays, the systolic fault-tolerant scheme proposed here has the distinct advantage that *all* live cells can be utilized (Figure 2-3 (a)). As illustrated by Figure 2-3 (b), fault-tolerant schemes previously proposed in the literature either suffer from low utilization of live cells[5, 6, 7, 8], or reduced throughput due to a slower system clock required by the fact that the communication between logically adjacent cells can now span an arbitrarily large number of failures[9, 10]. Moreover, as will be shown in the next section, this systolic fault-tolerant technique can be generalized to two-dimensional arrays.

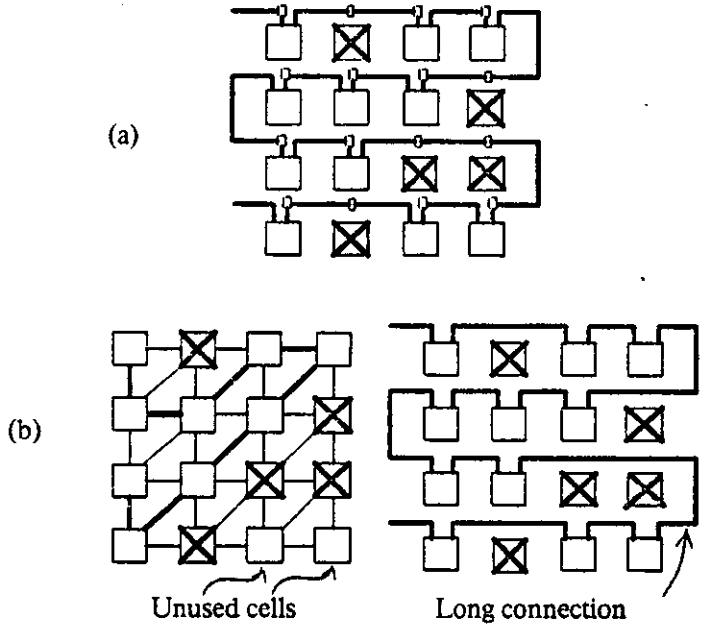

Figure 2-3: (a) Systolic and (b) previous fault-tolerant schemes for uni-directional linear arrays

We now examine more carefully the idea behind our fault-tolerant scheme for the linear array of Figure 2-2. Because of the unit delay introduced by the bypass registers, all the cells after the failed one receive data items one cycle later than they normally would. Since both the x- and y-data streams are delayed by the same amount, the relative alignment between the two data streams remains unchanged. Thus, all the cells after the third one receive the same data and perform the same function, with a one-cycle delay, as would the cell preceding it in a normal array. For this reason, an n-cell, uni-directional, linear array with k defective cells will perform the same computation as a perfect array of $n-k$ cells.

The above reasoning also implies that the correctness of a uni-directional linear array is preserved, if the

same delay of any length of time is introduced uniformly to *all* the data streams between two adjacent cells. This result is directly applicable to the implementation of two-level pipelined arrays. We can interpret the stages in a given pipelined processing unit as additional delays in the communication between a pair of adjacent cells.

Consider, for example, the problem of implementing the systolic array of Figure 2-1 using the pipelined multiplier and adder of Figure 1-1 (b). Since the adder is now a three-stage pipeline unit instead of a single-stage unit, two additional delays are introduced in the y-data path. Thus each cell requires a total number of four delay registers be placed in the x-data path—one is implicit in the original cell definition, one is the delay register in the original algorithm design, and the rest to balance the extra delays in the y-data stream. The resulting two-level pipelined array is depicted in Figure 2-4. This two-level pipelined scheme was proposed previously[3], but we show it here as a special case of a general theory.

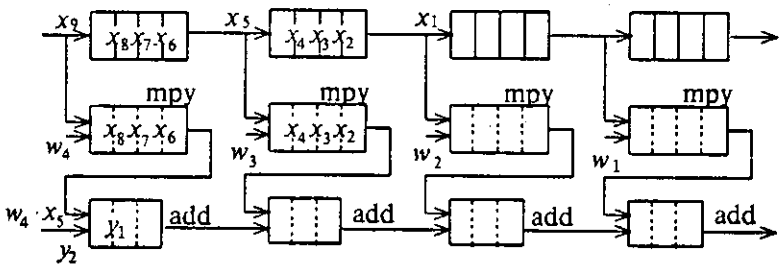

Figure 2-4: Two-level pipelined systolic array for convolution, using pipelined arithmetic units of Figure 1-1 (b)

3. Systolic Arrays without Feedback Cycles

3.1. The *Cut Theorem*

The results of the preceding section can be easily derived from a general theory formulated in terms of a mathematical notion called a *cut*. We model a systolic array as a directed graph, with the nodes denoting the combinational logic and the edges the communication links[11]. The edges are weighted by the number of registers on the links. We say that two designs are *equivalent* if, given an initial state of one design, there exists for the other design an initial state such that (with the same input from the *host*) the two designs produce the same output values (although possibly with a different delay). In other words, as far as the host is concerned the designs are interchangeable provided the differences in the timing of the output are taken into account.

We define a *cut* to be a set of edges that "partitions" the nodes into two disjoint sets, the *source set* and the *destination set*, with the property that these edges are the only ones crossing the boundary and are all directed from the source to the destination set.

Theorem 1: (*Cut Theorem*) For any design, adding the same delay to all the edges in a cut and to those pointing from the host to the destination set of the cut will result in an equivalent design.

Proof: Let S be the original design partitioned by a cut into sets A and B, the source and the destination set respectively. Let S' be the same as S (with its corresponding sets A' and B'), with the difference that d delays are now added onto the edges in the cut.

We will show that S and S' are equivalent in that if we properly initialize S', the output values from A and A' are identical starting from time t_0. Similarly the output values from B and B' are identical, except that the latter lag behind by d cycles.

We define the initial state of A' (at time t_0) to be identical to the state of A at time t_0. Since none of the edges in the cut feed into A', directly or indirectly, nodes in A' behave exactly the same way as the corresponding ones in A and thus produce the same outputs. Therefore, all the inputs arriving at B' are the same as those arriving at B, except that they lag behind by d cycles due to the additional delay registers. If we can define an initial configuration for B' such that at time t_0+d, it reaches the same configuration as B at t_0, then nodes in B' will behave the same way as the corresponding ones in B with a d cycle delay.

We will now proceed to show that such an initial configuration can indeed be defined. First, we let the initial state of B' be identical to the state of B at time t_0-d. Associated with each input edge into set B', e', are the registers $r_1(e'),\ldots,r_d(e')$, where the contents of $r_j(e')$ are moved to $r_{j+1}(e')$ every cycle. We let the initial values of these registers $r_1(e'),\ldots,r_d(e')$ be the values of the corresponding edge in S at time $t_0-1, t_0-2,\ldots,t_0-d$ respectively. This implies that the behavior of B' from time t_0 to t_0+d-1 is identical to that of B from t_0-d to t_0-1 and thus the configuration of B' at t_0+d and that of B at t_0 are identical. □

Since we are concerned with only adding extra delays to an optimized design, we do not need the generality in Leiserson and Saxe's retiming lemma[12, 13]. Thus the result we need requires a far simpler proof.

We will demonstrate that despite its simplicity, the cut theorem is a powerful and convenient tool for designing fault-tolerant and two-level pipelined systolic arrays. For example, as depicted in Figure 3-1 (a), the edges between any two adjacent cells of a uni-directional linear array form a cut. Hence by the cut theorem, we can see immediately that both the defective array of Figure 2-2 (a) and the two-level pipelined array of Figure 2-4 are equivalent to the original array of Figure 2-1. Figure 3-1 (b) depicts a less obvious cut, consisting entirely of all the output edges from the multipliers. This implies that the convolution array will function correctly regardless of the number of pipeline stages present in the multipliers (provided the number is the same for all the multipliers in the array). For instance, if all the four-stage multipliers in Figure 2-4 were replaced with ten-stage multipliers, the resulting systolic convolution array would still be correct.

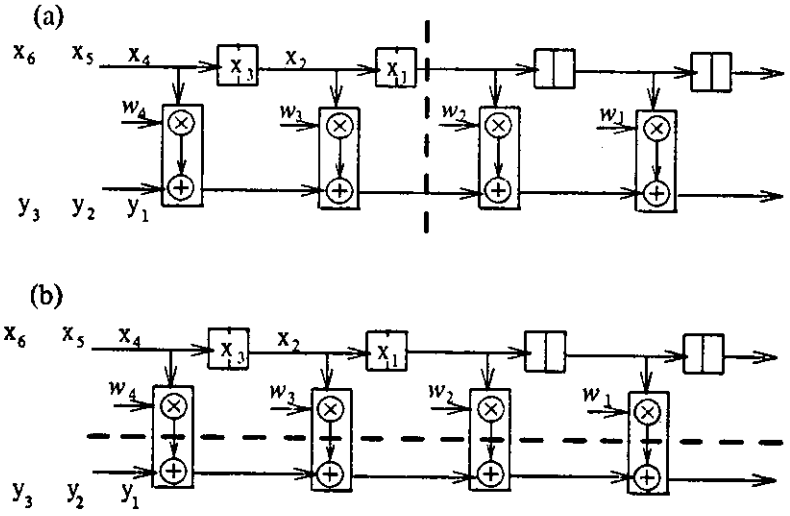

Figure 3-1: Two types of cuts for a uni-directional linear systolic array for convolution

3.2. Systolic Fault-Tolerant Schemes for Two-Dimensional Arrays

We first illustrate the basic techniques by considering the rectangular array of Figure 3-2 (a) where the data move downwards and to the right. Among many other applications, this array can perform matrix multiplication with either an operand or the partial result matrix stored in the array during the computation. Any curve whose slope at any point is within 90 degrees defines a cut, as illustrated in Figure 3-2 (b).

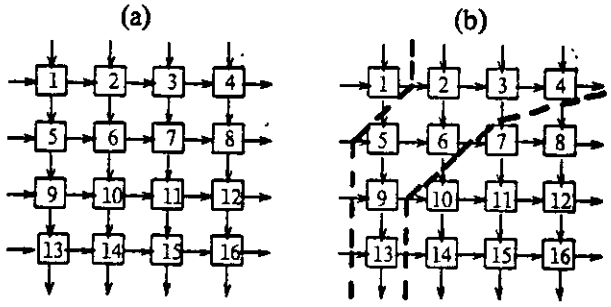

Figure 3-2: (a) Rectangular systolic array without feedback loops and (b) examples of cuts

Suppose that cells 3, 6, 9 and 13 were faulty, as depicted in Figure 3-3 (a). Figure 3-3 (b) illustrates that cell 7 could receive data from its new neighbors, cells 2 and 5, via bypass registers at cells 3 and 6. Figure 3-3 (c) shows that the resultant array can be viewed as a 4×3 systolic array where a unit delay is added to all the edges in a cut. Therefore by the cut theorem, this defective array is equivalent to a flawless 4×3 array.

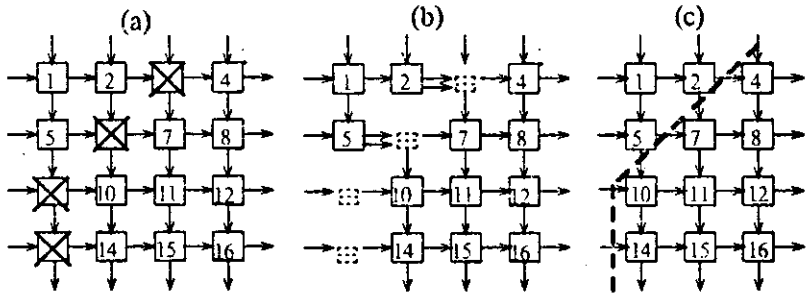

Figure 3-3: (a) Systolic array with defective cells, (b) systolic fault-tolerant scheme using bypass registers and (c) the corresponding cuts

Our simulation results show that while the utilization of the live cells for the above scheme using only bypass registers can be poor, it is greatly improved if an additional "delay register" is provided in each cell. Figure 3-4 (a) depicts a diagonal failure pattern, for which it is possible to prove that no systolic fault-tolerant procedure using only bypass registers can achieve a high utilization of the live cells. If we could introduce an extra delay on each of the data paths other than those on the diagonal, then as shown in Figure 3-4 (b), all the live cells were utilized. We can view this array as a 4×3 array with an additional delay on every edge in the cuts shown in Figure 3-4 (c), thus it is equivalent to a perfect 4×3 array by the cut theorem.

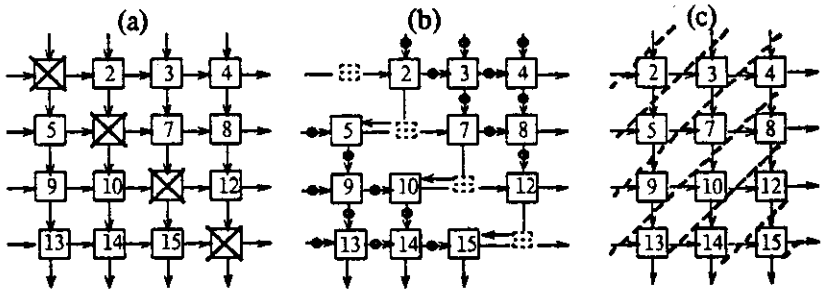

Figure 3-4: (a) Failures on the diagonal, (b) fault-tolerance with delay registers, denoted by black dots and (c) the corresponding cuts

A more realistic example is given in Figure 3-5 (a), where 28 cells in a 10×10 array fail. If only bypass registers are used, it seems that the largest square array that one can implement is 6×6, as depicted by Figure 3-5 (b). However, if one delay register is allowed for each data stream at each cell, it is possible to implement a 7×7 array, as depicted by Figure 3-5 (c).

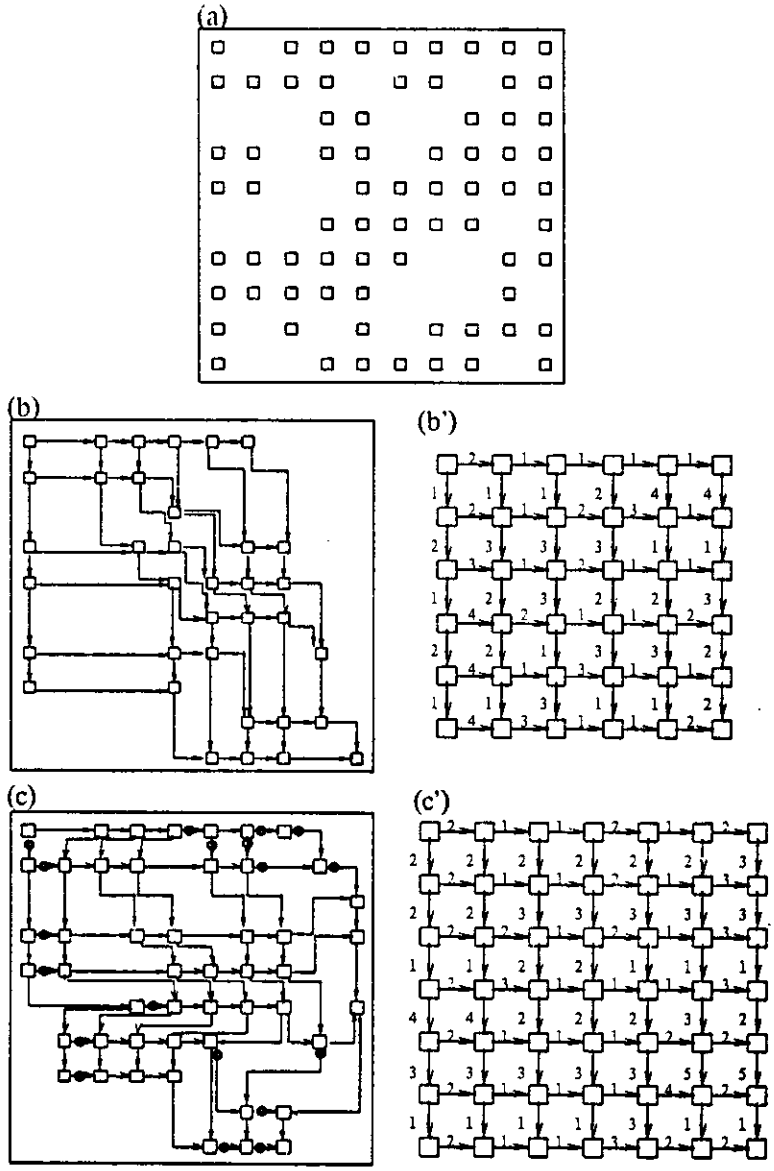

Figure 3-5: (a) Live cells, (b) 36 cells linked to form (b') using only bypass registers and (c) 49 to form (c') if one extra delay register is provided. (Black dots represent delay registers and the weight on each edge indicates the amount of delay)

In general, the more delay registers are provided, the better the utilization. Study is currently underway to examine the tradeoffs between the amount of hardware required and the utilization of the live cells at different failure rates. We note that for systolic arrays made of programmable cells such as the CMU Programmable Systolic Chip (PSC)[14, 15], implementing programmable delay in the data path is straightforward and requires no extra circuitry. If the necessary channel width is also provided, any arbitrary assignment scheme can be implemented. In particular, the previous upper bound results[16, 17] on the maximum connec-

tion length for the reconfiguration of two-dimensional systolic arrays can be directly translated into upper bounds on the maximum programmable delay needed for each data path through each cell.

For the case of two-dimensional arrays, the cut theorem does not lead directly to an effective procedure to obtain a functional array from a defective one. The following theorem, which can be proven to be equivalent to the cut theorem, is useful for this purpose:

> **Theorem 2:** If a systolic design is obtained from an other one by adding delays to some of its edges, then these two designs are equivalent if the total delay added to all the paths between any two nodes is the same.
>
> **Proof:** The result follows directly from the retiming lemma of Leiserson and Saxe[12], by assigning the lag of a node to be the total amount of delay added to any of the paths linking the host to the node. □

For a rectangular or hexagonal array with no feedback cycles (as depicted by Figures 3-2 (a) and 3-6 (a)), the condition in Theorem 2 holds if and only if it holds in every unit square or triangle respectively. Therefore we have a simple criterion for deriving equivalent designs which relies only on "local information". It is used in the heuristic program that generated the configurations of Figure 3-5 (b) and (c).

3.3. Two-Level Pipelining for Two-Dimensional Systolic Arrays

We consider a hexagonal systolic array that can perform band matrix multiplication[18], as depicted in figure 3-6 (a).

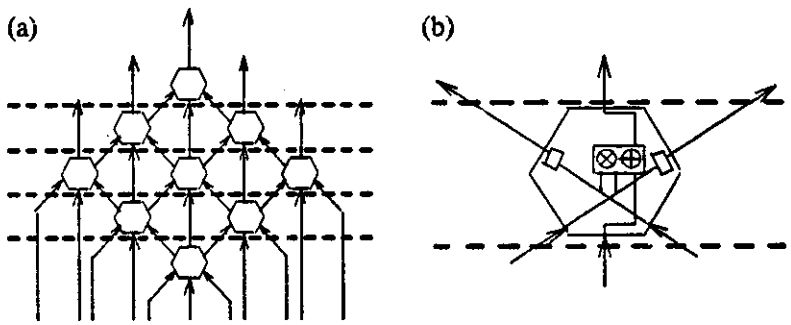

Figure 3-6: (a) Hexagonal systolic array without feedback loops and (b) original cell definition

Two results follow directly from the cut theorem:

1. It is easy to see that the edges under each dashed line in Figure 3-6 (a) define a cut. Every vertical edge, representing the output from an adder (Figure 3-6 (b)), intersects two dashed lines while any other edge intersects only one. Thus by the cut theorem, if the number of pipeline stages in all the adders is increased by $2k$, then for each cell, k delays must be added to the other data paths. Figure 1-1 (b) depicts the case when $k=1$.

2. Consider the output edges of all the multipliers in the array. Like those in the one-dimensional

convolution array (Figure 3-1 (b)), these edges define a cut since none of the outputs from the adders are fed back into the multipliers. By the cut theorem, we can conclude that systolic cells can be implemented using pipelined multipliers of any number of stages without any further modification provided the number of stages is the same for all the multipliers.

4. Systolic Arrays with Feedback Cycles

As previously noted, a cut partitions the nodes of a graph into two sets with data flowing uni-directionally between them. Thus a cut cannot cross feedback cycles in systolic arrays. This is the same as saying that "retiming" preserves the total number of registers in a cycle[12]. In other words, it is impossible to add extra delays on edges in a feedback cycle for fault-tolerance and pipelining purposes. In this section we describe a new technique for treating systolic arrays with feedback cycle. Such arrays include systolic designs for LU-decomposition[19], QR-decomposition[20], triangular linear systems[19] and recursive filtering[21].

4.1. Computation of Simple Recurrences—An Example of Cyclic Systolic Arrays

To illustrate the basic ideas, we consider the computation of the following simple recurrence of size $n-1$:

given the initial values $\{y_0, y_{-1}, \ldots, y_{-n+2}\}$
compute the output sequence $\{y_1, y_2, \ldots\}$ as defined by

$$y_i = \sum_{j=1}^{n-1} y_{i-j}$$

Although summation is used here, the computational structure presented below generalizes to any associative operator. An n-cell systolic array with feedback cycles[21] is capable of performing this simple recurrence computation of size up to $n-1$. Depicted in Figure 4-1 (a) is such an array where $n=6$. The partial sums move down the array from left to right picking up the completed results that move in the opposite direction. The computation of each sum is completed when it reaches the end of the array. Note that this is a 2-slow[11] system, in the sense that only half its cells are active at all time.

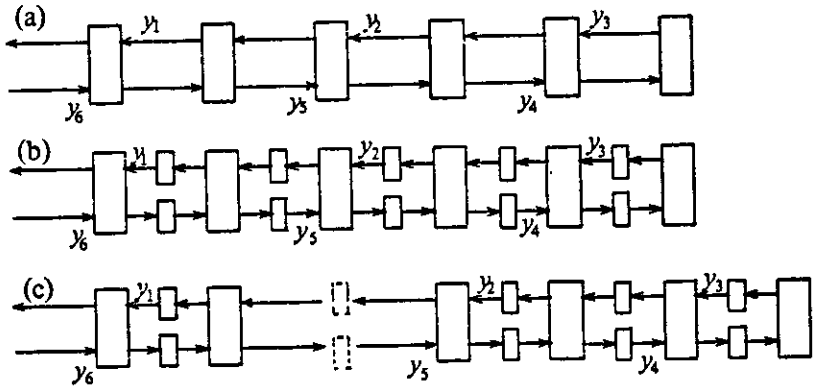

Figure 4-1: Linear array with feedback: (a) original array, (b) reduced throughput and (c) single failure

A naive attempt at achieving fault-tolerance involves slowing the system down even further. In the array of Figure 4-1 (b) data pass through an extra register per cell. This is a 4-slow system, performing the same computation as the 2-slow version, but at half its throughput. Suppose that the third cell from the left were to fail. The original function of the array could be preserved by simply allowing cells 2 and 4 to communicate through a bypass register (as illustrated in Figure 4-1 (c)). A drawback of this approach is that the perfor-

mance of the array degrades rapidly with respect to the number of consecutive failed cells that need to be tolerated. Note that systolic arrays with feedback cycles are initially 2- or 3-slow in general, and in order to tolerate k consecutive failures, the throughput must be further decreased by a factor of $k+1$.

The recurrence of size $n-1$ computed by an n-cell bi-directional linear array (illustrated in Figure 4-1 (a)), can also be implemented on an $n/2$-cell ring with uni-directional data flow (as in Figure 4-2). The systolic ring works as follows. The $n/2$ most recently computed results are stored in each of the $n/2$ cells, while the next $n/2$ partial sums travel around the ring to meet these stored values. A sum is completed as it travels past the cell with the most recently stored value. It is then deposited in the next cell, which contained the oldest value. Meanwhile, the computation of a new value begins in the next successive cell. Figure 4-2 (a) indicates that y_3 has just been added to y_4. Figure 4-2 (b) shows the result of the next cycle—the final value of y_4 has replaced y_1, while y_7 is ready to pick up its first term, y_2. Like the bi-directional systolic array of Figure 4-1 (a), this systolic ring has a computational rate of one output every two cycles. However, since all its cells are active at any time, only half as many cells are needed.

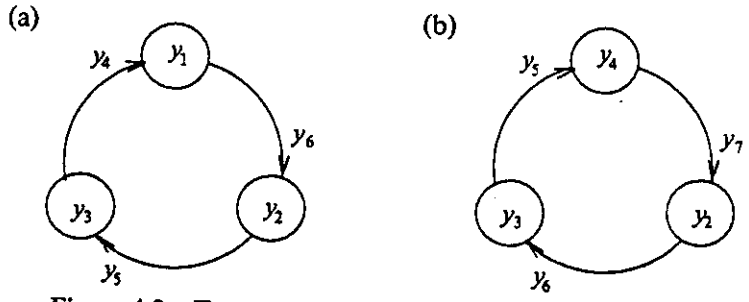

Figure 4-2: Two consecutive snapshots of a systolic ring

More importantly, the throughput of this structure degrades gracefully as the number of defective cells increases. For example, for an array of size n with one cell failure, the reduction in throughput is only $1/(2n-1)$ of the original. A defective ring of 5 cells with 2 failures is illustrated in Figure 4-3. As in an acyclic array, each failed cell in the ring is bypassed with a single register. This ring can solve a problem of size 6 at a throughput of 3 outputs every 8 cycles. In this example, the final values of y_4, y_5, y_6 and y_7 are produced at time instants t_0, t_0+2, t_0+5 and t_0+8, respectively.

4.2. Performance of Systolic Rings

We have shown in the previous section that the ring structure is suitable for solving simple recurrences where each result is dependent on a fixed number of previous results. This characterizes many of the problems solved by systolic arrays with feedback. Before we propose ring algorithms for these problems, we first analyze the performance of the general ring structure. The performance measures of primary interest are the data throughput rate and the amount of hardware required. We will show that a fully functional ring

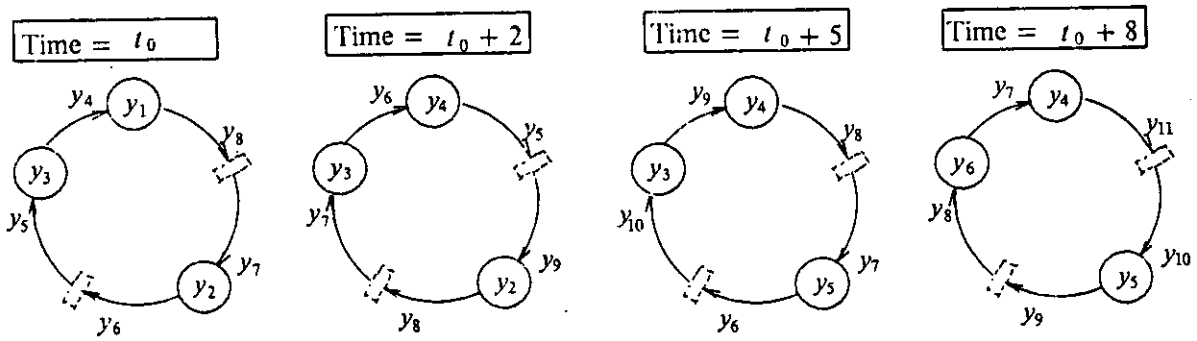

Figure 4-3: Defective systolic ring with 2 faulty cells

typically requires only a fraction of the hardware required by other proposed algorithms and achieves the same throughput. Furthermore, systolic rings are much more amenable to the addition of fault-tolerant features and a second level of pipelining.

To dissociate the issue of the problem size a ring can handle from the analysis of the data throughput, we will first consider the hypothetical case of an "infinite" structure. By "infinite", we mean that this structure has at least as many cells as the number of results we want to compute. Since each cell is used to store only one result, there is no feedback, and this structure can be viewed as a uni-directional linear array. Therefore we can apply the previously derived results to analyze the performance in case of cell failures.

If the ring is flawless, the output stream of each cell is fed directly into the next, and a result is produced every two cycles (independent of the problem size). Let us consider the case where we have k failures every m cells. Similar to the previous case, a defective cell in a ring can be bypassed with a register. While this procedure does not change the functionality of the algorithm, all the actions of the cells following a failed cell are delayed, relative to the one preceding it, by one extra clock cycle. As a result, the action of every $(m-k)$ th live cell has a relative delay of k cycles. Therefore, while a perfect array stores a result in each of the $m-k$ consecutive cells every $2(m-k)$ cycles, an impaired array stores a result in each of the $m-k$ live cells every $2(m-k)+k$ cycles.

> **Lemma 3:** A perfect array of "infinite" size can solve a recurrence problem of any size at a throughput rate of 1/2. If k out of every m cells fail, the throughput is reduced to $(m-k)/(2m-k)$.

In this "infinite" array, each cell is only active for a period of n clock cycles. Moreover, the ith live cell is activated by the arrival of the ith result. Since every result depends on the value of the preceding result, a cell can be activated only after the activation of the preceding cell. Consequently, the string of active cells is always contiguous and its length is the product of the duration of the active period and the throughput rate. If a finite ring is used to solve a particular problem and its size is no smaller than the length of the active string, then it is virtually equivalent to this "infinite" structure. Therefore we have the following results:

Theorem 4: A perfect ring of size m can solve recurrences of sizes up to $2m-1$ at a throughput rate of 1/2. If k cells fail, it can solve problems of sizes up to $2m-k-1$ at a throughput rate of $(m-k)/(2m-k)$. In other words, the reduction in throughput due to the k failures is only $k/(2m-k)$ of the original.

4.3. Two-Level Pipelining for Systolic Rings

By going through a similar argument as previously presented for the two-level pipelined array, we can obtain the following result.

Theorem 5: A systolic ring of m p-stage pipelined cells can solve recurrences of sizes up to $(p+1)m-1$ at a throughput rate of $1/(p+1)$. If k of the m cells fail, this ring can solve problems up to size $(p+1)m-pk-1$ at a throughput rate of $(m-k)/[(p+1)m-pk]$. In other words, the reduction in throughput is only $k/[(p+1)m-pk]$ of the original.

4.4. Other Examples of Systolic Ring Architectures

4.4.1. Solution of Triangular Linear Systems

Let $A=(a_{ij})$ be a nonsingular $n\times n$ band, lower triangular matrix with bandwidth q. Suppose that A and an n-vector $b=(b_1,\ldots,b_n)^T$ are given. The problem is to solve $Ax=b$ for $x=(x_1,\ldots,x_n)^T$. This is a typical recurrence of size $q-1$. A ring of $q/2$ cells is sufficient to solve the problem at a throughput of one result every two cycles. As a comparison, the previous bi-directional linear systolic array[19] has the same throughput, but it uses twice as many cells. The ring is also more robust—with k failures in a ring of m cells, the throughput is only reduced from 1/2 to $(m-k)/(2m-k)$.

Figures 4-4 and 4-5 illustrate the data flow pattern of a perfect 3-cell ring and a 4-cell ring with one failure, respectively, when solving a triangular linear system with bandwidth $q=6$. While this problem size is the largest the former ring can handle, the latter one can solve linear systems with bandwidth up to $q=7$. As a result, the cells in the defective ring of Figure 4-5 are idle one-seventh of the time. In the figure, a cell is assumed to be idle for one cycle if the input has a "don't care" value.

The final step in the computation of each result (x_i) involves a subtraction (from b_i) and a division (by a_{ii}). This needs to be performed by every cell. To avoid having to provide each cell with a division capability and an external data path, we precompute the reciprocals of the diagonals outside the ring and send the additional input (b_i) to the cells via a systolic path.

4.4.2. Triangularization of a Band Matrix

The usefulness of the systolic ring approach is not limited to linear array solutions—Figure 4-6 (a) depicts a two-dimensional ring structure for triangularizing a band matrix A, with bandwidth $w=6$ and $q=3$ sub-diagonals. This ring structure can perform the QR-decomposition, an important computation for linear least squares approximation, and solve linear systems stably using neighbor pivoting[22].

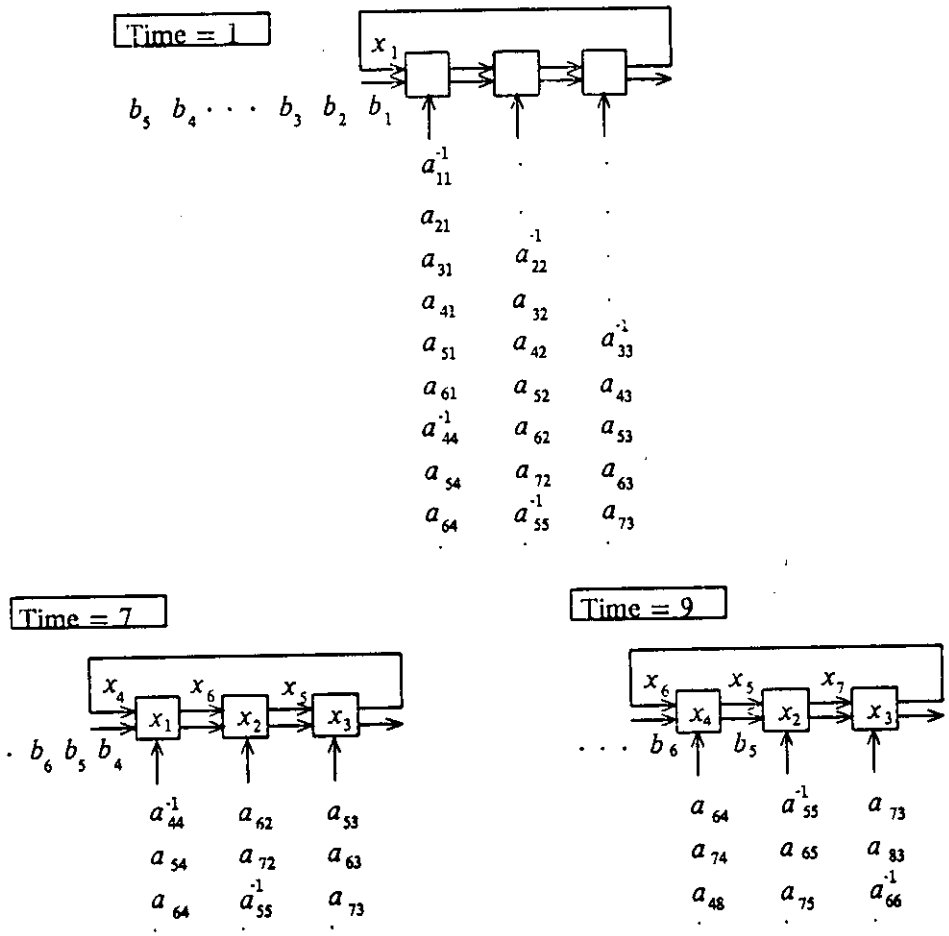

Figure 4-4: Systolic ring for solving triangular linear systems

Each ring in the structure of Figure 4-6 is responsible for the elimination of a subdiagonal, with the bottommost ring handling the bottommost subdiagonal. Consider the operations of a ring, as illustrated by Figure 4-6 (b). The parameters needed for performing the elimination (which for the QR-decomposition are the values defining the Givens rotations) pass around the ring after they are generated. Let p_i be the parameter generated by the element to be eliminated in row i and the element above it. If the data input a_{ij} is not an element of the subdiagonal to be eliminated, it is updated on the arrival of p_i. It is then retained for one cycle to compute with p_{i+1} before it is output to the next ring. If a_{ij} is to be eliminated, it is computed with the stored value, $a_{i-1,j}$, to get p_i, which is then passed down the ring. Thus the output of each ring is the result obtained by eliminating the last subdiagonal of the input array. The uppermost ring outputs the entries of the triangular matrix that we want to compute. Note that corresponding to the elimination of each subdiagonal, a new super-diagonal is created. In the systolic ring, the new elements for this super-diagonal take the place previously occupied by the elements of the eliminated subdiagonal.

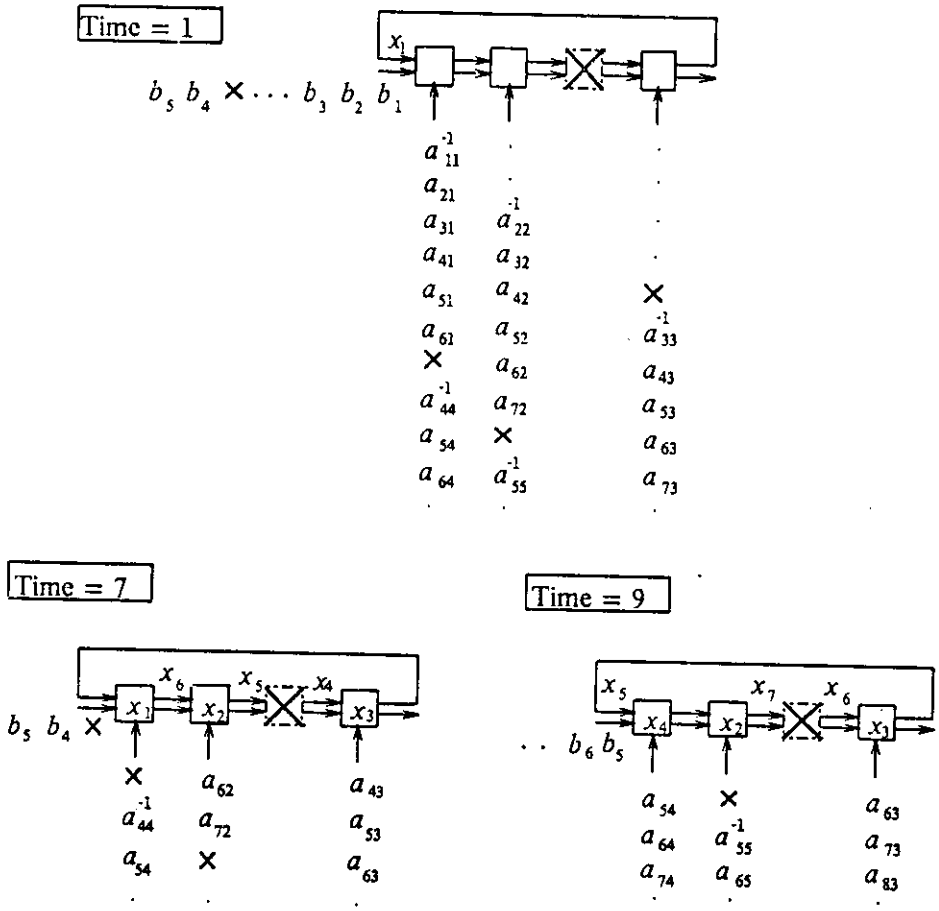

Figure 4-5: A single failure in a systolic ring for solving triangular linear systems

Unlike the data values circulating the rings in the previous examples, the p_i are computed before they are passed around. However, they have the same property that they are produced every two cycles and need to meet with $w-1$ input values before they can be discarded. Therefore, from our previous analysis, q rings of $w/2$ cells each are required for triangularizing a band matrix with bandwidth w and q subdiagonals. For the case of QR-decomposition, it requires half the amount of hardware and achieves the same throughput of a previous solution[20].

Figure 4-7 depicts the fault tolerance scheme for such a structure. If the failed cells are covered by k cuts, then by Theorem 4 the throughput is reduced by $k/(w-k)$ of the original.

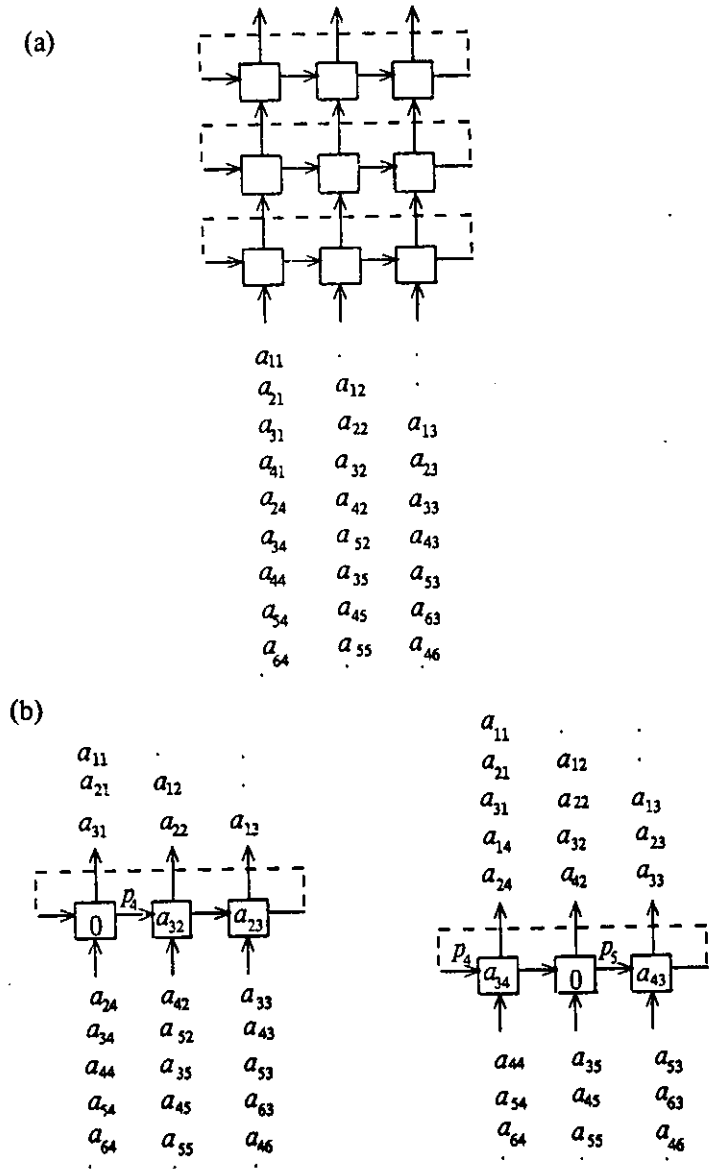

Figure 4-6: (a) Two-dimensional systolic ring structure for matrix triangularization and (b) two snapshots of the bottommost ring

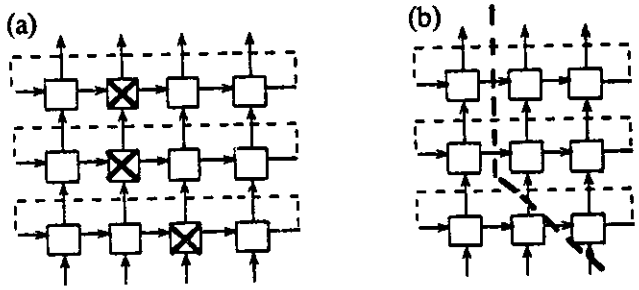

Figure 4-7: (a) Failed cells in a ring architecture for matrix triangularization and (b) the corresponding cut

4.4.3. LU-Decomposition of a Band Matrix

Figure 4-8 depicts a "two-dimensional systolic ring architecture" for the LU-decomposition of a band matrix, $A=LU$. For a given matrix A with bandwidth $2q-1$ we need to use $q/3$ rows of cells, with q cells in each row. The $q/3$ most recently computed rows of u_{ij}'s are stored in the cells as they are generated, while the l_{ij}'s are passed down the rows. Figure 4-9 shows the snapshots of this structure at various stages in the computation. By viewing this structure as an array of rings, its performance can be analyzed using the result of Theorem 5 with parameter $p=2$. The throughput of this array is the same as the previous design[19] which use, however, three times as many cells. Figure 4-10 illustrates how we apply the *cut* technique to this array. Vertical connections have to be provided for linking purposes. Note that if all the faults are covered in k cuts, the decrease in throughput is only $k/(q-2k)$ of the original.

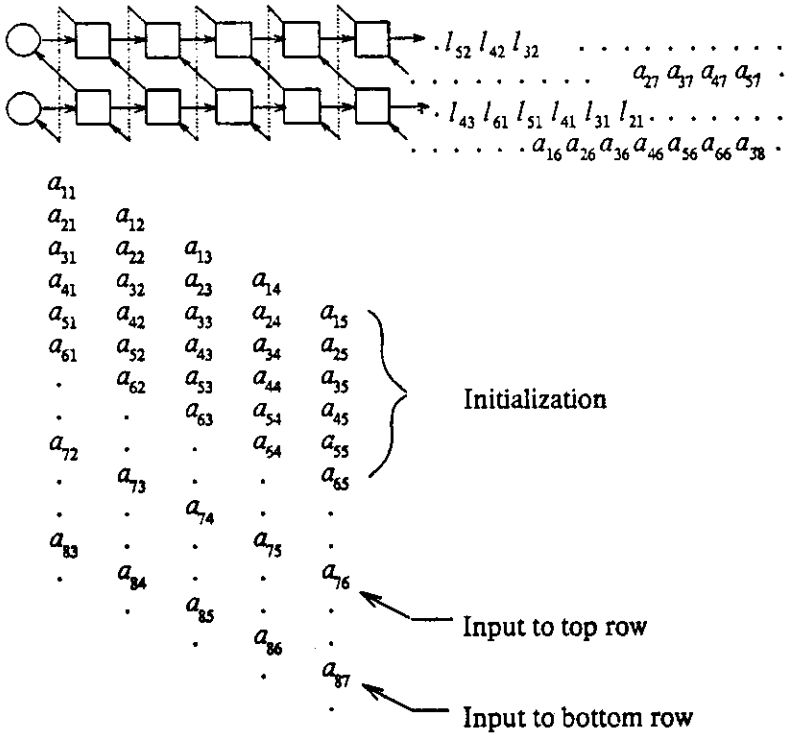

Figure 4-8: Systolic ring architecture for LU-decomposition

4.5. General Remarks on Systolic Rings

The systolic ring architecture has some disadvantages over other systolic architectures, but they are compensated for by its superior fault-tolerance performance. One of the possible disadvantages is that we need to provide an additional data path to unload the values during the computation, as the computed results are continuously stored in the ring. This is, however, not the case for the triangularization schemes of section 4.4.2.

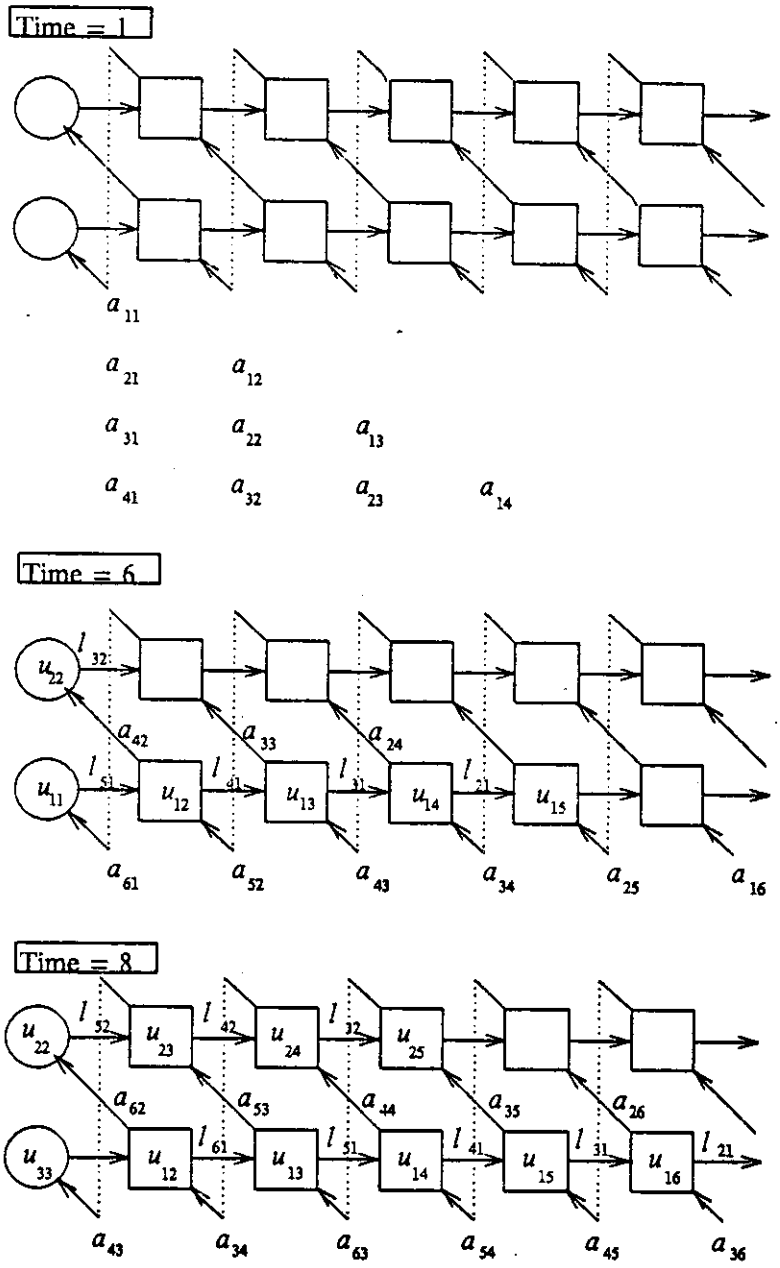

Figure 4-9: Snapshots of a ring architecture for LU-decomposition

In many of the conventional cyclic algorithms, only one or a few boundary cells may require special processing capability and extra input/output bandwidth. However, with some ring architectures, more cells are required to assume the role of a boundary cell. Algorithm-dependent methods can sometimes be used to alleviate the problem of having to provide all these cells with special functionality. For instance, in the previous example of solving triangular linear systems, instead of providing each cell with the capability to divide, we precompute the reciprocals of the diagonals.

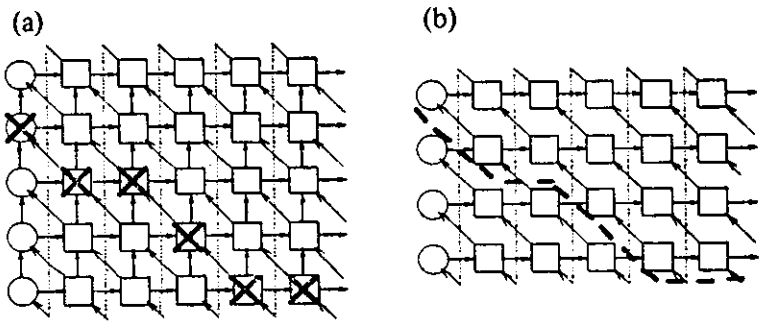

Figure 4-10: (a) Failures in a ring architecture for LU-decomposition and (b) the corresponding cut

Finally we note that the one-dimensional ring can be laid out simply and effectively by folding the array in half. A cell only communicates with a neighboring cell and thus needs only constant length interconnection wires. For a two-dimensional ring, such as the one used for LU-decomposition, it is possible to lay it out by folding each column in half.

5. Summary and Concluding Remarks

Systolic arrays are more specific than general processor arrays, in the sense that data streams in a systolic array move in a prespecified manner. Making explicit use of this additional information, the systolic fault-tolerant approach introduced in this paper is usually more effective than other schemes designed for general processor arrays. In particular the systolic approach requires no increase in interconnection length. This eliminates a source of inefficiency, such as increased system cycle time or driver area, common to most other approaches.

For uni-directional linear arrays, our systolic fault-tolerant technique achieves 100% utilization of live cells, without extra registers nor interconnection links. For two-dimensional arrays without feedback cycles, we have established the basic theory needed for developing efficient systolic fault-tolerant schemes. We expect that if one extra delay register is provided for each data stream at each cell, a reasonably good utilization of live cells can be achieved. We are currently investigating the performance of our techniques for two-dimensional arrays with different degrees of redundancy.

Although many systolic algorithms with feedback have been proposed, some of the same problems to which these algorithms address can also be solved by systolic arrays without feedback. Examples of such problems include convolution, graph connectivity and graph transitive closure[4, 23, 24]. Acyclic implementations usually exhibit more favorable characteristics with respect to fault-tolerance, two-level pipelining, and problem decomposition in general.

For problems that have been solved exclusively by systolic arrays with feedback cycles, the paper introduces a new class of systolic algorithms based on a ring architecture. These systolic rings have the property that the throughput degrades gracefully as the number of failed cells in the rings increases. Furthermore, as a byproduct of the ring architecture approach, we have derived several new systolic algorithms which require only one-third to one-half of the cells used in previous designs while achieving the same throughput.

We have shown that the two-level pipelining problem in systolic arrays can be solved by the same techniques used to solve the fault-tolerance problem. An important task left for the future is the development of software of solving both problems automatically.

References

1. Blankenship, P.E., "Restructurable VLSI Program," Semiannual Technical Summary ESD-TR-81-153, MIT Lincoln Lab, March 1981.

2. Woo, et al., "A 32 Bit IEEE Floating-Point Arithmetic Chip Set," *Proceedings of 1983 International Symposium on VLSI Technology, Systems and Applications*, 1983, pp. 219-222.

3. Kung, H.T., Ruane, L.M., and Yen, D.W.L., "Two-Level Pipelined Systolic Array for Multidimensional Convolution," *Image and Vision Computing*, Vol. 1, No. 1, February 1983, pp. 30-36. An improved version appears as a CMU Computer Science Department technical report, November 1982

4. Kung, H.T., "Why Systolic Architectures?," *Computer Magazine*, Vol. 15, No. 1, January 1982, pp. 37-46.

5. Aubusson, R. C. and Catt, I., "Wafer Scale Integration—A Fault Tolerant Procedure," *IEEE Journal of Solid-State Circuits*, Vol. SC-13, No. 3, June 1978, pp. 339-344.

6. Fussel, D. and Varman, P., "Fault-Tolerant Wafer-Scale Architectures for VLSI," *Proceedings of the 9th International Symposium on Computer Architecture*, April 1982, pp. 190-198.

7. Koren, I., "A Reconfigurable and Fault-Tolerant VLSI Multiprocesor Array," *The 8th Annual Symposium on Computer Architecture*, IEEE & ACM, May 1981, pp. 442.

8. Manning, F.B., "An Approach to Highly Integrated, Computer-Maintained Cellular Arrays," *IEEE Transactions on Computers*, Vol. C-26, No. 6, June 1977, pp. 536-552.

9. Rosenberg, A.L., "On Designing Fault-Tolerant Arrays of Processors," Tech. report CS-1982-14, Duke University, 1982.

10. Rosenberg, A.L., "The Diogenes Approach to Testable Fault-Tolerant Networks of Processors," Tech. report CS-1982-6, Duke University, 1982.

11. Leiserson, C.E., *Area-Efficient VLSI Computation*, PhD dissertation, Carnegie-Mellon University, 1981. The thesis is published by the MIT Press, Cambridge, Massachusetts, 1983.

12. Leiserson, C.E. and Saxe, J.B., "Optimizing Synchronous Systems," *Journal of VLSI and Computer Systems*, Vol. 1, No. 1, 1983, pp. 41-68.

13. Kung, H.T. and Lin, W.T., "An Algebra for VLSI Computation," *Elliptic Problem Solvers II*, Birkhoff, G. and Schoenstadt, A.L., eds., Academic Press, 1983. Proceedings of Conference on Elliptic Problem Solvers, January 1983.

14. Fisher, A.L., Kung, H.T., Monier, L.M. and Dohi, Y., "Architecture of the PSC: A Programmable Systolic Chip," *Proceedings of the 10th Annual International Symposium on Computer Architecture*, June 1983, pp. 48-53.

15. Fisher, A.L., Kung, H.T., Monier, L.M., Walker, H. and Dohi, Y., "Design of the PSC: A Programmable Systolic Chip," *Proceedings of the Third Caltech Conference on Very Large Scale Integration*, Bryant, R., ed., Computer Science Press, Inc., California Institute of Technology, March 1983, pp. 287-302.

16. Leighton, F.T. and Leiserson, C.E., "Wafer-Scale Integration of Systolic Arrays," *Proceedings of 23rd Annual Symposium on Foundations of Computer Science*, IEEE, October 1982, pp. 279-311.

17. Greene, J.W. and Gamal, A.E., "Configuration of VLSI Arrays in the Presence of Defects," Tech. report, Information Systems Lab, Stanford University, May 1983.

18. Weiser, U. and Davis, A., "A Wavefront Notation Tool for VLSI Array Design," *VLSI Systems and Computations*, Kung, H.T., Sproull, R.F., and Steele, G.L., Jr., eds., Computer Science Press, Inc., Computer Science Department, Carnegie-Mellon University, October 1981, pp. 226-234.

19. Kung, H.T. and Leiserson, C.E., "Systolic Arrays (for VLSI)," *Sparse Matrix Proceedings 1978*, Duff, I. S. and Stewart, G. W., eds., Society for Industrial and Applied Mathematics, 1979, pp. 256-282. A slightly different version appears in *Introduction to VLSI Systems* by C. A. Mead and L. A. Conway, Addison-Wesley, 1980, Section 8.3, pp. 37-46.

20. Heller, D.E. and Ipsen, I.C.F., "Systolic Networks for Orthogonal Equivalence Transformations and Their Applications," *Proceedings of Conference on Advanced Research in VLSI*, Massachusetts Institute of Technology, Cambridge, Massachusetts, January 1982, pp. 113-122.

21. Kung, H.T., "Let's Design Algorithms for VLSI Systems," *Proceedings of Conference on Very Large Scale Integration: Architecture, Design, Fabrication*, California Institute of Technology, January 1979, pp. 65-90. Also available as a CMU Computer Science Department technical report, September 1979.

22. Gentleman, W.M. and Kung, H.T., "Matrix Triangularization by Systolic Arrays," *Proceedings of SPIE Symposium, Vol. 298, Real-Time Signal Processing IV*, Society of Photo-Optical Instrumentation Engineers, August 1981, pp. 19-26.

23. Guibas, L.J., Kung, H.T. and Thompson, C.D., "Direct VLSI Implementation of Combinatorial Algorithms," *Proceedings of Conference on Very Large Scale Integration: Architecture, Design, Fabrication*, California Institute of Technology, January 1979, pp. 509-525.

24. Tchuente, M. and Melkemi, L., "Systolic Arrays for Connectivity Problems and Triangularization for Band Matrices," Tech. report R.R. No. 366, IMAG, Institut National Polytechnique de Grenoble, March 1983.