WAFER-SCALE INTEGRATION AND TWO-LEVEL PIPELINED IMPLEMENTATIONS OF SYSTOLIC ARRAYS

by

H.T. Kung, Monica S. Lam

DRC-15-25-84

December, 1984

Wafer-Scale Integration and Two-Level Pipelined Implementations of Systolic Arrays

H. T. Kung and Monica S. Lam

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

February 1984

This paper is to appear in *Journal of Parallel and Distributed Processing*, Vol. 1, No. 1, 1984.

A preliminary version appears in *Proceedings of the MIT Conference on Advanced Research in VLSI*, January 1984, pp. 74-83.

✓ 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.

620.0042
C28d
DRC-15-[illegible]

Abstract

This paper addresses two important issues in systolic array designs. How do we provide fault-tolerance in systolic arrays for yield enhancement in wafer-scale integration implementations? And, how do we design efficient systolic arrays with two levels of pipelining? The first level refers to the pipelined organization of the array at the cellular level, and the second refers to the pipelined functional units inside the cells.

The fault-tolerant scheme we propose replaces defective cells with clocked delays. This has the distinct characteristic that data can flow through the array with faulty cells at the original clock speed. We will show that both the defective cells under this fault-tolerant scheme and the second level pipeline-stages can simply be modeled as additional delays in the data paths of "generic" systolic designs. We introduce the mathematical notion of a *cut* to solve the problem of how to allow for these extra delays while preserving the correctness of the original systolic array designs.

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 and ring architectures for arrays 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. The 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

In recent years many systolic algorithms have been designed and several prototypes of systolic array processors have been constructed[1, 2, 3, 4]. Major efforts are currently devoted to building systolic arrays for large, real-life applications. In this paper, we will consider two implementation techniques for building high-performance systolic arrays: wafer-scale integration (WSI) and fabrication using pipelined components.

Fabrication flaws on a wafer are inevitable. It is necessary for a WSI circuit to be "fault-tolerant" so that wafers with defective components can still be used. A common approach is to include redundant circuitry in the design and avoid defects by programming the interconnection of the constituent elements. In particular, the laser-programming technology has been applied successfully to program the redundant circuitry in VLSI RAMs as a yield enhancement measure[5]. The MIT Lincoln Laboratory[6] has also been experimenting on the use of laser-programmable links to build wafer-scale processor arrays.

Systolic arrays are well-suited to the application of wafer-scale integration. They consist of large numbers of small and identical (thus interchangeable) cells and their regular and localized interconnection greatly simplify the problem of routing around defective cells. On the other hand, systolic architectures guarantee full exploitation of their constituent cells to achieve maximum parallelism. The more cells an array has, the more powerful it is. Wafer-scale integration has the potential to provide a very cost-effective and reliable way of implementing high-performance systolic systems.

Before WSI systolic arrays can become a reality, we must solve the problem of how to construct fault-tolerant arrays. After the cells are tested (by wafer-probing, for example), how do we route around the defects to build a functional array? (See Figure 1-1 (a)). This paper describes a "*systolic*" approach which provides fault-tolerance at a very low cost and admits of a graceful degradation in performance as the number of defects increases.

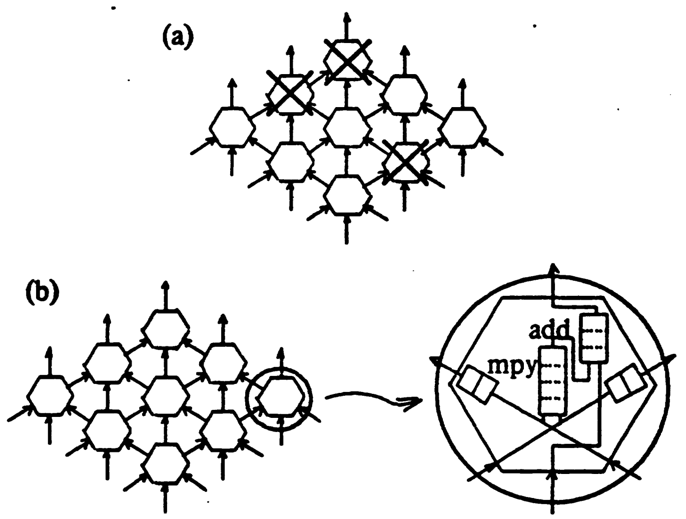


Figure 1-1: Two problems addressed in the paper: (a) fault-tolerance for arrays with faulty cells and (b) two-level pipelining

The use of pipelined components for implementing cells of systolic arrays is especially attractive for applications requiring floating-point operations. Commercially available floating-point multiplier and adder chips can deliver up to 5 MFLOPs per device. To achieve such high throughput, they typically have three or more pipeline stages[7]. These components, when used to implement systolic cells, form a *second level of pipelining*[8], the first being the pipelined organization of systolic arrays at the cellular level. While this additional level of pipelining can increase the system throughput, it considerably complicates the design of systolic array algorithms. Our solution to this problem is to devise a methodology to transform existing systolic designs which assume single-stage cells to arrays consisting of 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.

In the next section we will introduce our approach to the problems, using as an example the simplest type of systolic arrays—uni-directional linear arrays. As we will see, systolic arrays without feedback admit of a much simpler solution and they will be discussed in section 3. In section 4, we will propose a new architecture, the "systolic ring", which can be used in place of many systolic arrays with feedback cycles and are much more amenable to fault-tolerant measures. 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[9] 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)).

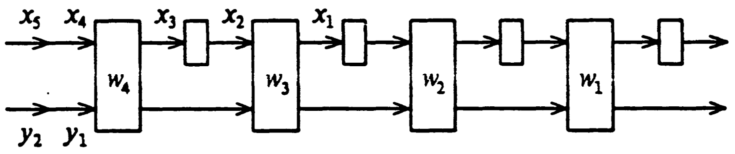


Figure 2-1: Uni-directional linear systolic array for convolution

Depicted in 2-2 (a) is an example of a 5-cell array with one faulty element. The defective cell in the middle is replaced with two "bypass" registers (drawn in dotted lines)—one for the x-data stream and one for the y-data stream. It can easily be shown that this array correctly solves the same problem as the array of Figure 2-1. 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 degradation in performance due to the defect is slight. The maximum convolution computed by this array in one pass can have only 4 rather than 5 weights, and the latency of the solution is increased by one cycle. However, the computational throughput, often the most important factor in performance, remains the same at one output per cell 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.

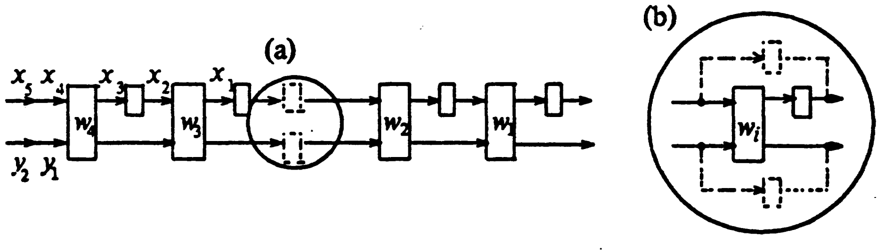


Figure 2-2: (a) Defective cell replaced with registers and (b) cell specification

A basic assumption of this paper is that the probability of the interconnection links and registers failing is very small and thus negligible. This is reasonable because these components are typically much simpler and smaller than the cells themselves. Furthermore, they can be implemented conservatively and/or with high redundancy to increase the yield.

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 call fault-tolerant schemes of this type *systolic* 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 advantages that it utilizes *all* the live cells and maintains the throughput rate of a flawless array (Figure 2-3 (a)). As illustrated in Figure 2-3 (b), other fault-tolerant schemes previously proposed in the literature either suffer from low utilization of live cells[10, 11, 12, 13], or reduced throughput due to a slower system clock required by the fact that the communication between logically adjacent cells can now span a large number of failures[14, 15, 16, 17]. Moreover, as will be shown in the next section, our systolic fault-tolerant technique can be generalized to two-dimensional arrays.

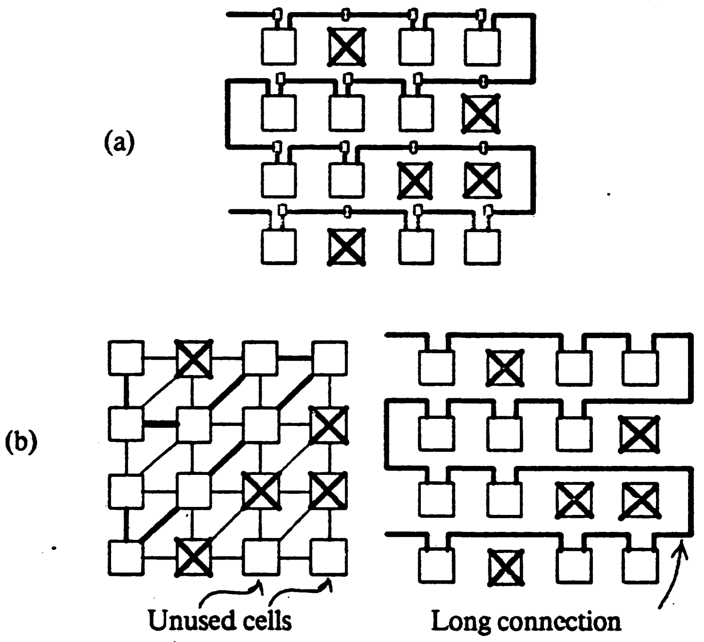


Figure 2-3: (a) Systolic and (b) previous fault-tolerant schemes for uni-directional linear arrays

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, the second is the delay register in the original algorithm design, and the last two are to balance the two new delays in the y-data stream. The resulting two-level pipelined array is depicted in Figure 2-4. This design has been proposed previously[8], but it is reproduced here as a special example of a general theory.

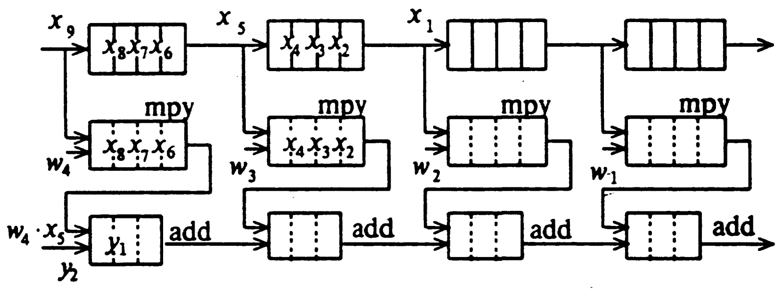


Figure 2-4: Two-level pipelined systolic array for convolution, using pipelined arithmetic units of Figure 1-1 (b)

3. Systolic Arrays without Feedback Cycles

From the previous section we see that both the defective cells in a fault-tolerant array and the pipeline-stages in systolic cells can simply be modeled as additional delays in the data paths. Thus by solving the one problem of how, if possible, to allow for additional delays in systolic designs, we can transform generic systolic designs to fault-tolerant or two-level pipelined designs. A general theory of adding and removing register delays to a system has been proposed by Leiserson and Saxe[18] in the context of optimizing synchronous systems.

3.1. The *Cut Theorem*

We model a systolic array as a directed graph, with the nodes denoting the combinational logic and the edges the communication links[19]. 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, i.e., the outside world) the two designs produce the same output values (although possibly with a constant delay). In other words, as far as the host is concerned, the designs are interchangeable provided the possible 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 in a graph into two disjoint sets, the *source set* and the *destination set*, with the property that these edges are the only ones connecting nodes in the two sets and are all directed from the source to the destination set.

We say that a systolic design is a "delayed" system of another design if the former differs from the latter by having additional delays on some of the communication links. Thus the graph representations of the two designs are the same except for the weights on the edges that correspond to the communication links with additional delays.

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 by properly initializing S' (at t_0), the output values from A and A' will be identical and that the output values from B are the same as those from B', but lagging behind by d clock cycles.

We define the initial state of A' 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.

Let $r_1(e'), \ldots, r_d(e')$ be the delay registers on any edge in the cut, e', with $r_1(e')$ being closest to the source node and $r_d(e')$ closest to the destination node. First, we assign the initial state of B' to be identical to the state of B at time t_0-d. We then initialize the registers $r_1(e'), \ldots, r_d(e')$ with the values of the data on the corresponding edge in S at time $t_0-1, t_0-2, \ldots, t_0-d$, respectively. In this way, the input data received by the nodes in set B' from time t_0 to t_0+d-1 is identical to those received by B from t_0-d to t_0-1 and so the configuration of B' at t_0+d and that of B at t_0 are identical. Since the outputs from A' are the same as those from A, all the inputs arriving at B' starting from time t_0+d are the same as those arriving at B, except that they lag behind by d cycles due to the additional delay registers. Therefore the nodes in B' will behave the same way as the corresponding ones in B with a d cycle delay. □

We say that a delayed system S' is *derivable* from S if there exists a set of cuts $C_1, C_2, \ldots, C_n$ with their cut delays $d_1, d_2, \ldots, d_n$, such that

$$\forall e' \in S', \text{ number of additional delays on } e' = \sum_{\{i \mid e' \in C_i\}} d_i.$$

Since equivalence is associative, the cut theorem implies that if a "delayed" design is derivable from the original design then the two designs are equivalent.

Since a cut partitions the nodes of a graph into two sets with data flowing uni-directionally between them, it cannot cross any feedback cycle. On the other hand, for any given edge not in a feedback cycle, we can always construct a cut set that contains it. Therefore any number of delays on the data paths in a graph without feedback can always be incorporated if we have the option of inserting other delays into the system.

3.2. Linear Arrays Without Feedback

We will now apply the above results to the examples we discussed previously. 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.

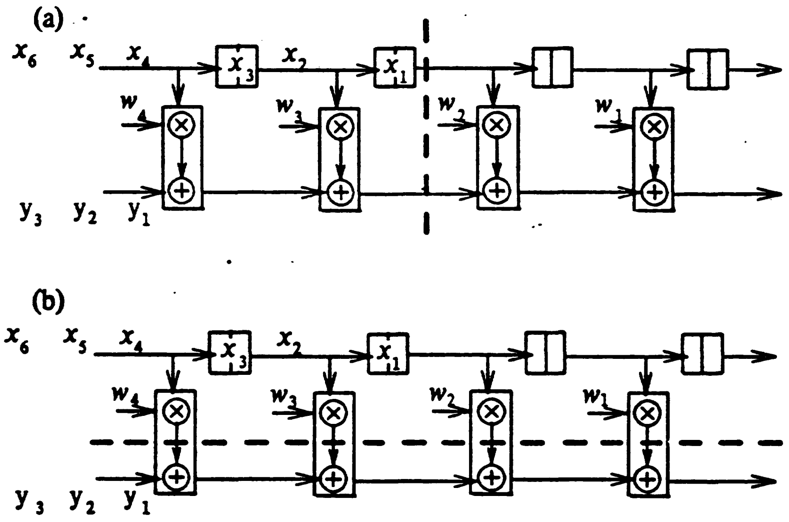


Figure 3-1: Two types of cuts for a uni-directional linear systolic array for convolution

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

It is just as simple to apply the cut theorem to two-level pipelined arrays of two dimensions. Let us consider the example of a hexagonal systolic array that can perform band matrix multiplication[20] (Figure 3-2 (a)). Two results follow directly from the cut theorem:

1. The edges under each dashed line in Figure 3-2 (a) define a cut. All vertical edges, each representing an adder's output (Figure 3-2 (b)), intersect two dashed lines while all the other edges intersect 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 $k=1$.

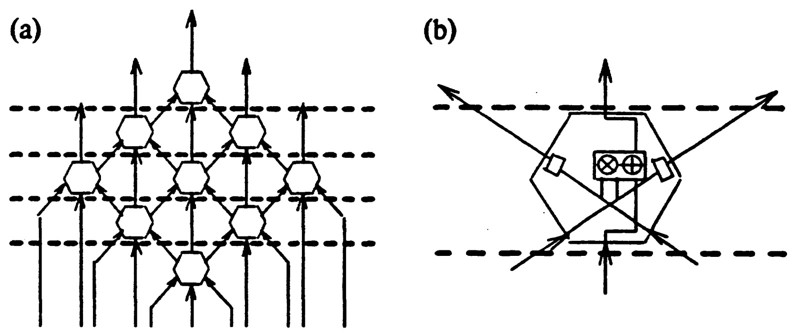


Figure 3-2: (a) Hexagonal systolic array without feedback loops and (b) original cell definition

2. Consider the output edges of all the multipliers in the array. Like those in the uni-directional linear 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 conclude that these 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.

3.4. Two-Level Pipelining for the FFT Processor Arrays

The cut theorem can be applied to two-level pipelined designs for any processor arrays without cycles. We consider here as an example, the well known processor array for computing fast Fourier transforms (FFTs). For an n-point FFT, the array has $\log_2 n$ stages of $n/2$ processors for performing butterfly operations. The data are shuffled between any two consecutive stages according to a certain pattern[21, 22]. Figure 3-3 depicts the so-called constant geometry version of the FFT algorithm (for $n=16$), that allows the same pattern of data shuffling to be used for all stages.

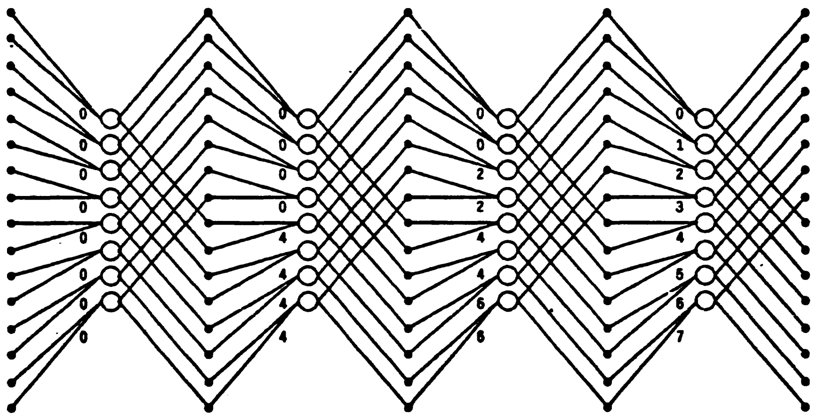


Figure 3-3: Constant geometry version of the FFT algorithm

In the figure the processors for butterfly operations are represented by circles, and the number h by an edge indicates that the result associated with the edge must be multiplied by ω^h, where ω is a primitive n-th root of unity.

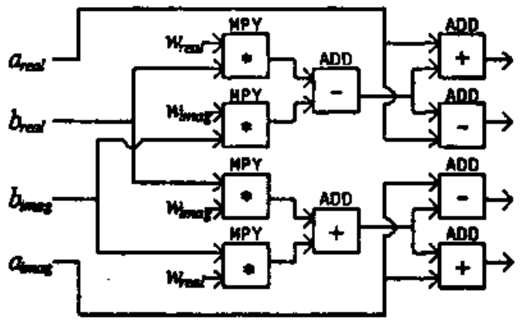
A butterfly operation,

$$(<*nal+\dot{J}*imag)^{\pm}(breal+Jb\dot{r}magX w_{real}+jw_{imag})$$
$$= \qquad \cdot w_{imag})]+j[a_{imag}\pm(b_{real}\cdot w_{imag}+b_{imag}\cdot w_{real})].$$

involves four real multiplications and six real additions. Figure 3-4 (a) depicts a straightforward processor implementation for the butterfly operation using four multipliers and six adders. The time that the processor takes to perform a butterfly operation is the total delay of one multiplier and two adders.

To increase the throughput for calculating butterfly operations, we implement the processors with pipelined multipliers and adders. Suppose that these functional units each have five pipeline stages, as in the case of some recent floating-point chips[7]. By the cut theorem, the pipeline delays on the *b^* and *b^ag* data paths have to be balanced by the same number of delays on the fl^/ and a'_{maz} input lines. The two-level pipelined design of the processor is shown in Figure 3-4 (b).

(a)



(b)

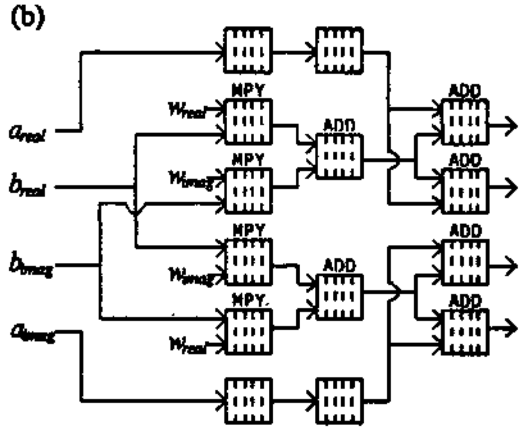
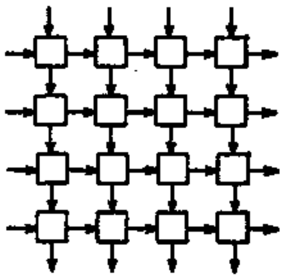


Figure 3-4: (a) Processor for the butterfly operation, and (b) corresponding two-level pipelined processor

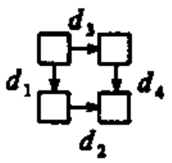
3.5. Systolic Fault-Tolerant Schemes for Two-Dimensional Arrays

Let us consider as an example the rectangular array of Figure 3-5 (a) where the data move forwards and downwards. 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. We will first discuss the constraints that a correct implementation must satisfy and then we will study several redundancy schemes.

(a)



(b)



$d_x + d_2 = </_3 + d_4$

Figure 3-5: (a) Rectangular systolic array without feedback loops and (b) local correctness criterion

3.5.1. The Local Correctness Criterion

By exploiting the regularity in systolic arrays, the following theorem reduces the problem of establishing equivalence between two designs to smaller problems which can be solved using only "local information".

Theorem 2: Let S be a mesh-connected systolic design without feedback and S' be a "delayed" version. S' is equivalent to S if for each square of adjacent cells in the grid, the number of delays on each of the two paths joining the two diagonally-opposite corners is the same.

Proof: Let V_i and E_i be the nodes and (vertical) edges in the ith column in grid S'. We form two subgraphs G_1' and G_2', such that G_1' contains all the nodes and edges to the left of the ith column and G_2' contains all those to the right, and in addition, they each contain V_i and E_i. We will first show that graph S' is derivable from S if subgraphs G_1' and G_2' are derivable from the corresponding subgraphs in S, G_1 and G_2, respectively.

Let C be a cut in subgraph G_1'. If C does not intersect E_i, all the nodes in V_i must belong to the destination set of the cut. Since there are no direct links between any nodes in the source set and the nodes in $S'-G_1'$, C is also a cut in S'. By the same token, any cut in subgraph G_2' that does not contain any edges in E_i is a cut in S'.

It is obvious that a cut can have at most one edge in E_i. Suppose the cuts C_1 in G_1' and C_2 in G_2' both contain the same edge e in E_i. For both subgraphs, all the nodes in V_i that are above e belong to the source set, and those below belong to the destination set. We observe that $C_1 \cup C_2$ partitions the nodes of S' also into a source set and a destination set, with the former being the union of the source sets in the two subgraphs and the latter the union of destination sets. Therefore, $C_1 \cup C_2$ is a cut in S'.

Without loss of generality, let the delay associated with all the cuts be 1. (A cut with d delays is equivalent to d identical cuts, each with 1 delay.) If G_1' and G_2' are derivable from G_1 and G_2 respectively, then for each edge $e \in E_i$ with $d(e)$ delays, there exist exactly $d(e)$ cuts containing e in each of the two subgraphs. Therefore all the cuts containing edges in E_i in the two subgraphs can be paired up to form cuts in S'. We have already shown that the cuts in the subgraph that do not contain any edges in E_i are also cuts in S'. Therefore if G_1' and G_2' are derivable from G_1 and G_2 respectively, then S' is also derivable from S.

The above result implies that we can cut up the grid S' into vertical strips and show that S' is equivalent to S by proving the equivalence of each of the strips. By applying the same argument on the horizontal links, we can further subdivide the strips into squares, each containing only four cells. The equivalence problem is now reduced to solving the equivalence for each of the squares. An edge from each of the two paths that connect the two diagonally-opposite corners constitute a cut. Therefore if the number of delays on each of the two paths of a square is the same, then the square is derivable from its counterpart in S. If this condition holds for each square, then S' is derivable from, and thus equivalent to S. □

The criterion for correctness as derived from this theorem is represented graphically in Figure 3-5 (b).

This theorem can be generalized to any array where we can find paths that partition the graph representing the array into disjoint subgraphs. For example, in the case of a hexagonal array without feedback cycles (Figure 3-2 (a)), the constraints for equivalence are simply reduced to the local criterion that for each unit triangle of three adjacent cells, the number of delays on each of the two paths connecting two of the corners of the triangle has to be the same.

3.5.2. Redundancy Schemes

The utilization of live cells for the rectangular systolic array of Figure 3-5 (a) depends on the availability of two hardware resources: delay registers in the live cells and the channel width. The results of Section 3.1 imply that if sufficient delay registers are available in the cells, the "systolic" approach can fully utilize all the live cells without any penalty to the throughput rate of the system. In general, a lower utilization can be expected with a smaller number of delay registers. The other factor that might decrease the utilization is the channel width. If there are not sufficient tracks in the channels, we might not be able to implement the interconnection desired.

We have conducted several experiments to study the tradeoff between the utilization of live cells and the required hardware resources. We implemented four heuristic programs modeling different redundancy schemes. We ran Monte Carlo simulations on three different array sizes and cell failure rates ranging from 5% to 65%. The distribution of defects is assumed to be identical for all the cell locations on the wafer. The different schemes are described in the following and their examples are illustrated in Figure 3-6.

1. *No additional hardware.* Because of the limitation in routing, we resort to a simple scheme where for each defective cell, we skip either the row or the column that contains the cell. The criterion of correctness is trivially satisfied. A greedy algorithm is used here; the row or column containing the most failures is eliminated first.

2. *No delay register and unlimited channel width.* In this scheme, all the cells in the final array are chosen so that the links only point in the forward or downward direction. This guarantees that the number of delays on each link is equal to the manhattan distance between the two end points of the link, and thus the local correctness criterion is satisfied for each unit square of the array. Our basic strategy is to build the array row by row, picking as the next cell the one that satisfies the criterion and excludes the least number of live cells from being used. A simple maze-runner is also implemented to determine the number of tracks required for interconnection.

3. *One delay register per data path and unlimited channel width.* The additional delay increases the flexibility in the assignment scheme, but it also complicates the algorithm of the program. We modified the program in scheme 2 such that if necessary, a delay register may be added to the new edges being created and to the old edges that are in the same row or column as the new ones, provided, of course, they do not have delay registers on them already.

4. *Unlimited delay registers and unlimited channel width.* How many delay registers are necessary to achieve 100% utilization? The scheme we chose requires delay registers be placed *only* on the *logically* vertical connections and none on the horizontal ones. The n live cells are partitioned into horizontal strips, each containing $\sqrt{n}$ cells. The cells in each strip are connected to form the rows and then connected to the corresponding cells in their neighboring rows. Delays are then assigned only to the logically vertical connections to satisfy the correctness criterion.

The empirical results are shown in Figure 3-7. Each data point represents the average value over 100 trials. These results indicate that unless the cell yield is exceptionally high, redundancy is essential (see Figure 3-7 (a)). The channel width is generally not a bottleneck. While low yields and poor utilization increase the length of the path between two logically neighboring cells, they also open up more space for routing. For the range of array sizes and cell yields in our simulations, three redundant tracks are found to be sufficient for schemes 2 and 3, and five for scheme 4.

The expected utilizations with zero and one delay register are shown in Figures 3-7 (b) and (c). The larger the array size, the more hardware delay registers are needed to get the same utilization. This is obvious since the set of constraints that have to be satisfied by a larger array is a superset of those satisfied by a smaller

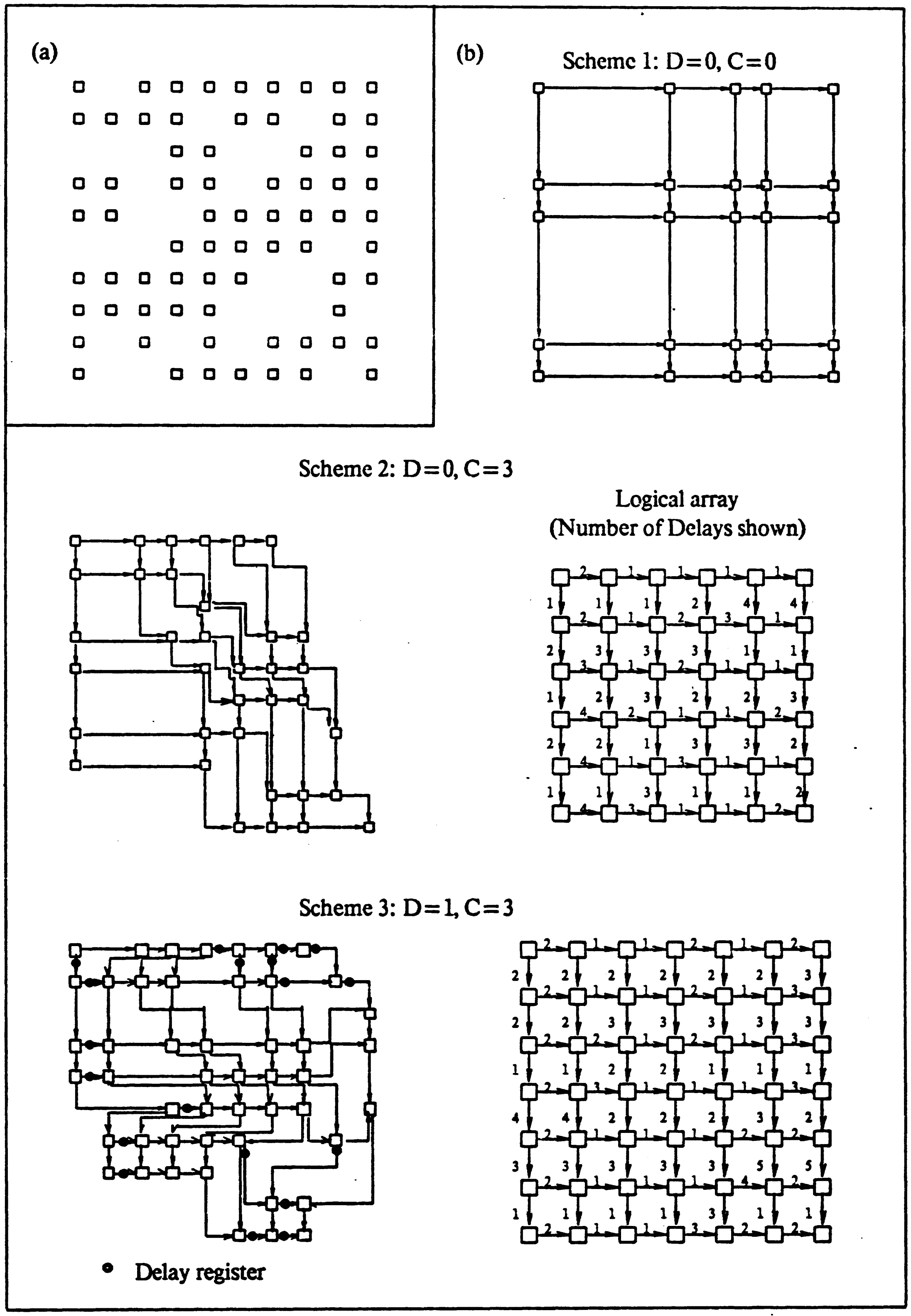
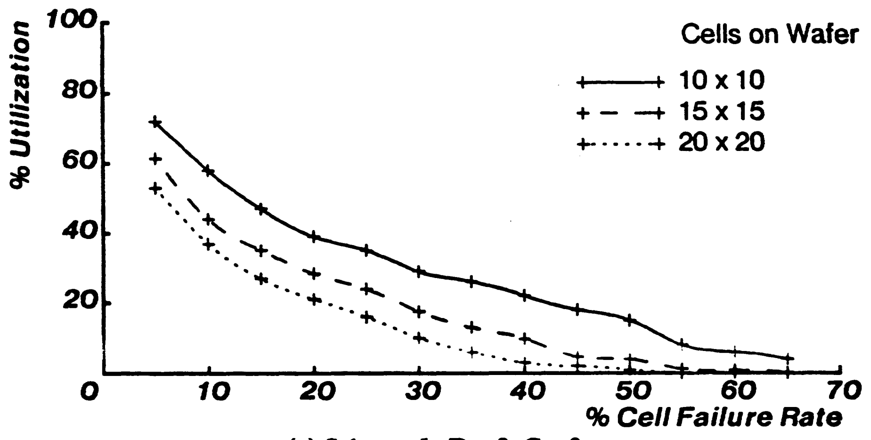
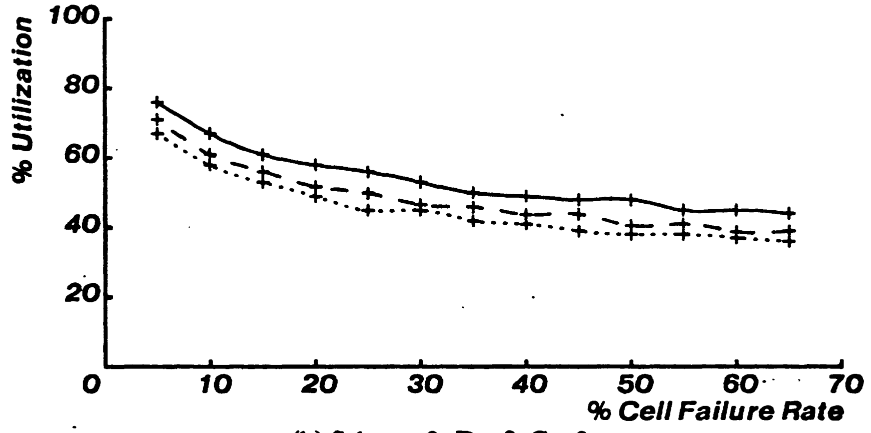


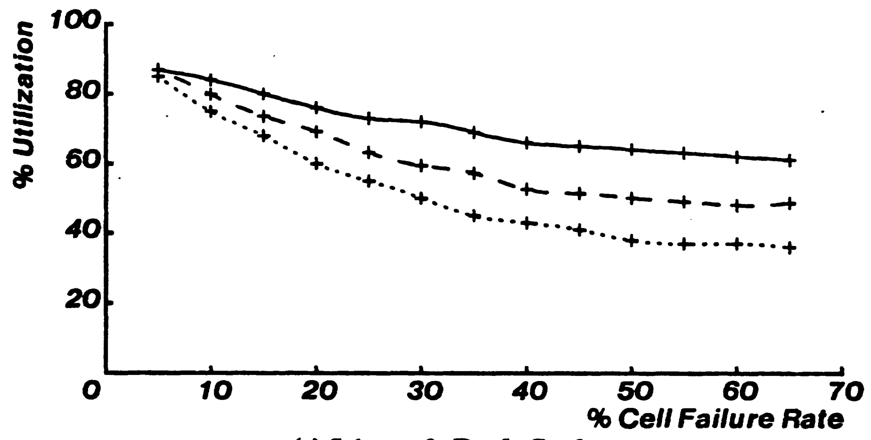
Figure 3-6: (a) Live cells (72 out of 100 cells) and
(b) array configurations under different redundancy schemes
(D represents number of delay registers and C the redundant tracks in channel)



(a) Scheme 1: D=0, C=0



(b) Scheme 2: D=0, C=3



(c) Scheme 3: D=1, C=3

Figure 3-7: Utilization under different redundancy schemes
(D represents number of delay registers and C the redundant tracks in channel)

array. We have to bear in mind, however, that the cells in a larger array are typically smaller and thus have lower failure rates. From Figure 3-8, we see that the maximum number of delay registers required on the logically vertical links to achieve 100% utilization is approximately equal to the number of cells on a side of a wafer. We note that for systolic arrays composed of programmable cells such as the CMU Programmable Systolic Chip (PSC)[23, 24], implementing programmable delay is straightforward and requires no extra circuitry.

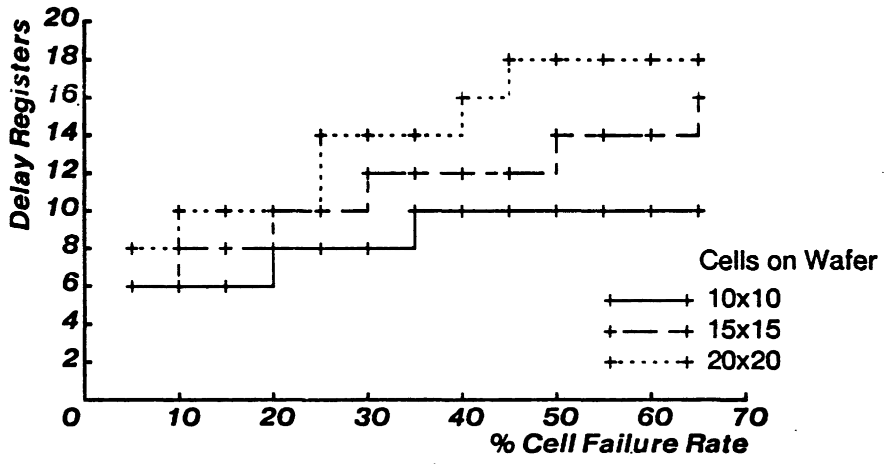


Figure 3-8: Maximum number of delays required by 98% of the trials to achieve 100% utilization using scheme 4

These experiments give us a general idea of the expected efficiency of the different redundancy schemes using the systolic approach. In-depth studies using a more precise model are necessary to determine the optimal or near-optimal redundancy scheme for any particular application. Probabilistic analyses[16, 17] have been performed for other fault-tolerant schemes where utilization is limited by the maximum length of interconnection allowed.

4. Systolic Arrays with Feedback Cycles

In this section we will describe a new technique for treating systolic arrays with feedback cycles. Such arrays include systolic designs for LU-decomposition[25], QR-decomposition[26], triangular linear systems[25] and recursive filtering[27].

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 computation structure presented below generalizes to any associative operator. An n-cell systolic array with feedback cycles[27] 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, y_4', y_5', y_6', move down the array from left to right picking up the completed sums that are moving in the opposite direction, y_1, y_2, y_3. The computation of each sum is completed when it reaches the end of the array. Note that this is a 2-slow[19] system, in the sense that only half its cells are active at all time.

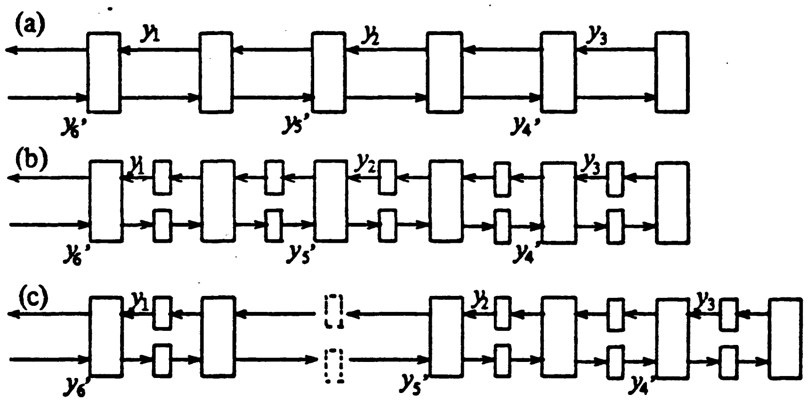


Figure 4-1: Linear array with feedback: (a) original array, (b) reduced throughput and (c) single failure

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 performance 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 (a)). 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. Every two cycles, a sum is completed and a new computation begins. For example at time 0 in Figure 4-2 (a), y_4' is ready to pick up its last term y_3 while y_6' is ready for its first term y_1. The final value of y_4 then travels to cell "a" to replace y_1. At time 2, y_5'

and y_7' will pick up their last and first terms respectively. 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, all its cells are active at any time, therefore only half as many cells are needed.

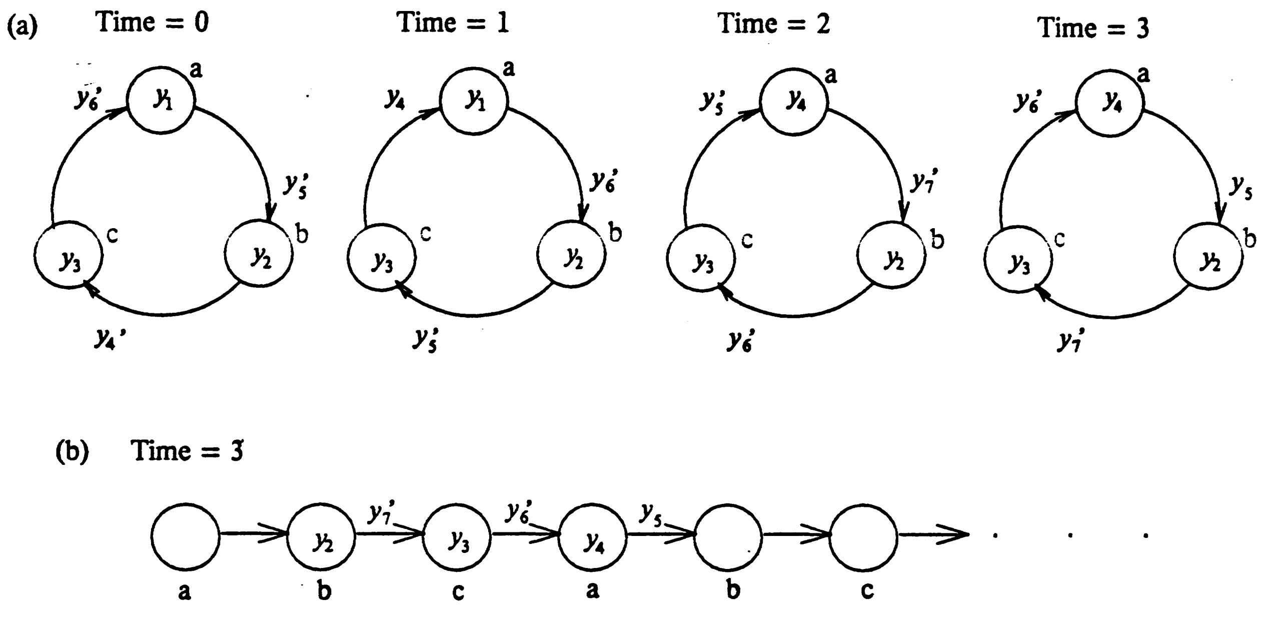


Figure 4-2: (a) Four consecutive snapshots of a systolic ring and (b) its unrolled structure

4.2. Fault-Tolerant Systolic Rings

Systolic rings require not only less cells than other designs solving the same problems, they also degrade gracefully as the number of defective cells increases.

Each cell in the systolic ring computes with a stored result for a period of $2n$ cycles before the result is replaced by a new value. The ring can be unrolled to form a linear array where each cell stores only one result in its whole lifetime, as shown in Figure 4-2 (b). This transformation reduces the ring structure to one without feedback, and thus allows us to analyze its fault-tolerant behavior using the results of the preceding section.

Figure 4-3 (a) shows an example of a 4-cell systolic ring with one defect and Figure 4-3 (b) shows its unrolled version. A defect in the ring of m cells translates to a defect in every block of m cells in the linear array. Recall that in an array without feedback, the bypass registers corresponding to the defects will cause a delay in the action of the cells but not the functionality of the array. It is therefore the case that the defective ring computes the recurrence correctly. However, due to the delay through the defective cell, the $m-1$ live cells produce results at a reduced rate of $m-1$ outputs every $2m-1$ $(=2[m-1]+1)$ cycles.

Although a defective systolic ring solves problems at a slower rate than a flawless ring with the same number of live cells, it can solve larger problems. The additional delay through the defective cell means that the live cells have an extra clock cycle before they have to store a new result. This cycle can be effectively used to compute with one more recurrence term. Figure 4-3 (a) shows the ring of $m-1$ live cells solving a maximum size problem with $2m-2$ $(=[2(m-1)-1]+1)$ recurrence terms. The following theorem summarizes the result of this section.

Theorem 3: 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

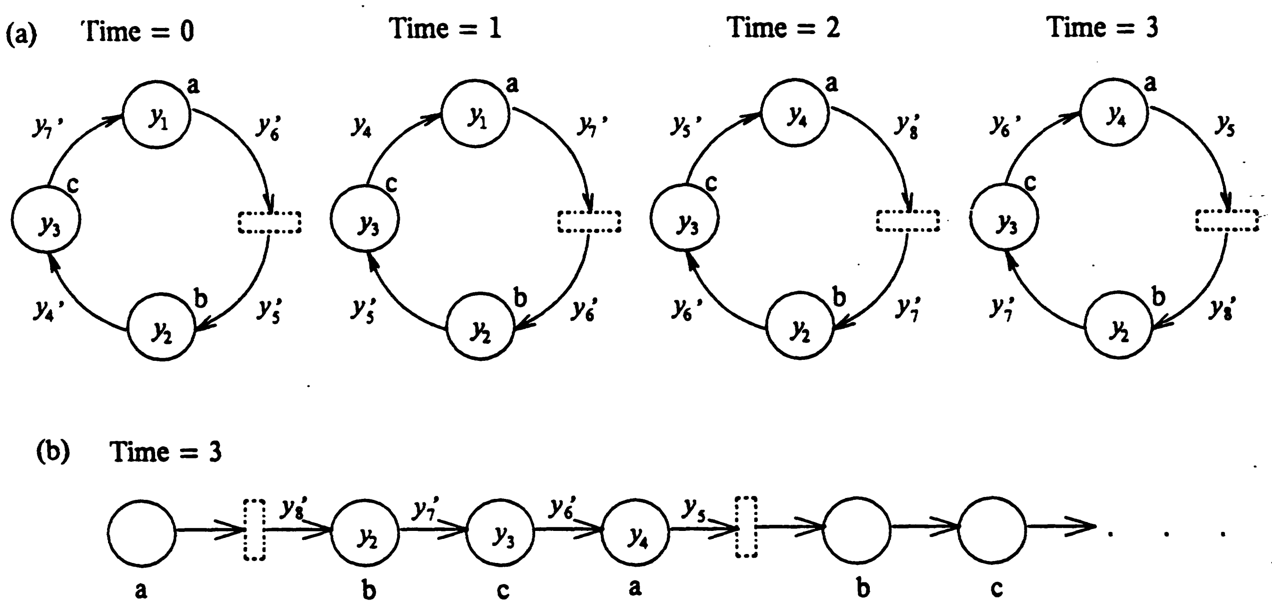


Figure 4-3: (a) Four consecutive snapshots of a systolic ring with one failure and (b) its unrolled structure

$(m-k)/(2m-k)$. 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 above for the two-level pipelined array, we can obtain the following result:

Theorem 4: 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

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. We will describe some of the examples in this section.

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 can be viewed as a recurrence problem 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[25] 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

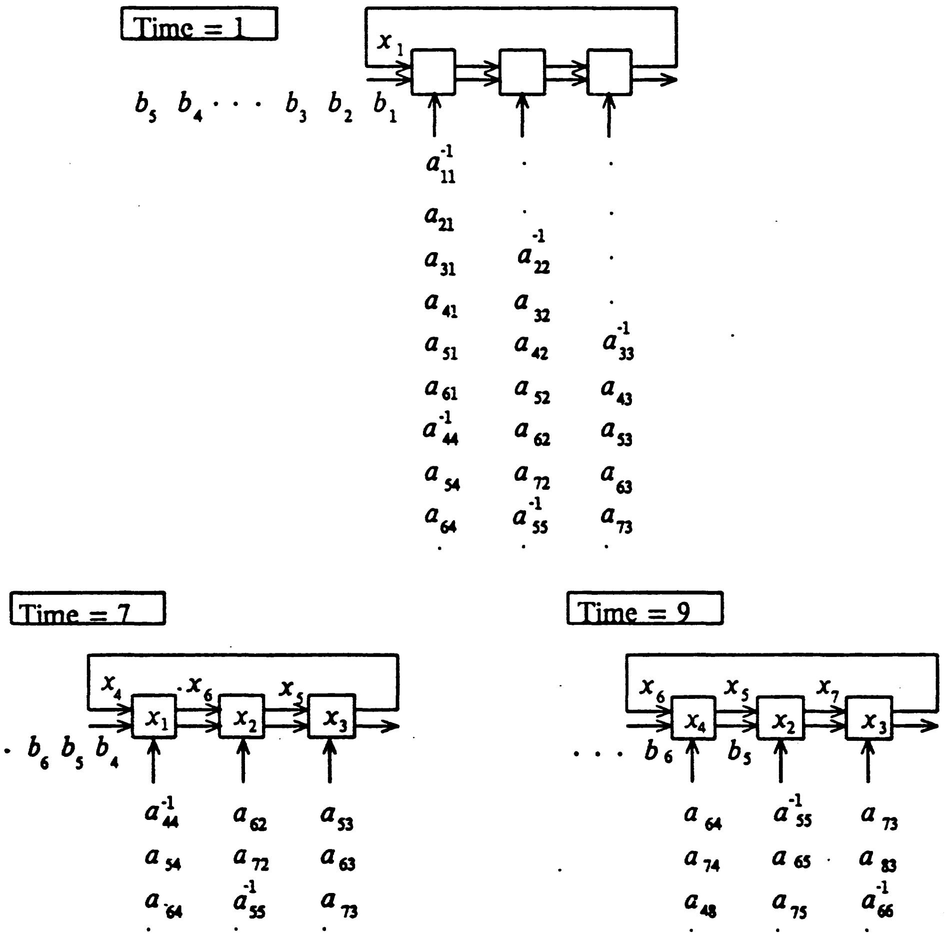


Figure 4-4: Systolic ring for solving triangular linear systems

largest the former ring can handle in one pass, 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, denoted by "×".

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.

The layout of a ring of processors is very straightforward, as shown in Figure 4-6. Similar to uni-directional linear arrays, defects on a ring can simply be bypassed via the cells' input/output registers.

4.4.2. Triangularization of a Band Matrix

The usefulness of the systolic ring approach is not limited to linear array solutions—Figure 4-7 (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 it can also solve linear systems using the stable computation technique of neighbor pivoting[28].

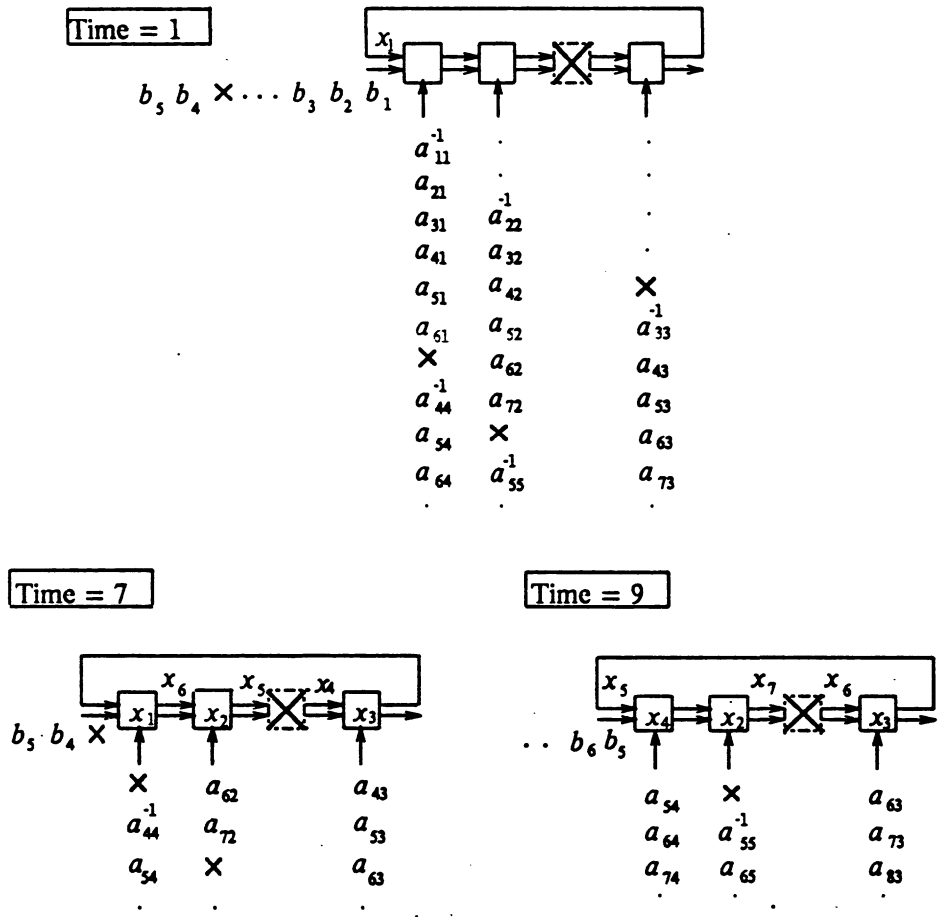


Figure 4-5: A single failure in a systolic ring for solving triangular linear systems

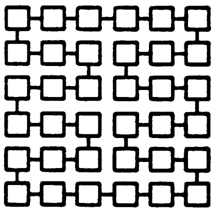


Figure 4-6: Layout of a Systolic Ring

Each ring in the structure of Figure 4-7 eliminates a subdiagonal, with the bottommost ring handling the bottommost subdiagonal. The operations of a ring are illustrated by Figure 4-7 (b). The parameters needed for performing the elimination (e.g. Givens rotations for QR-decomposition) pass around the ring after they are generated. Suppose p_i is the parameter generated by the element eliminated in row i and the element above it. If the data input a_{ij} is not on the subdiagonal to be eliminated, it is updated on the arrival of p_i. It stays in the cell for one cycle to compute with p_{i+1} and then moves on 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. 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 correspond-

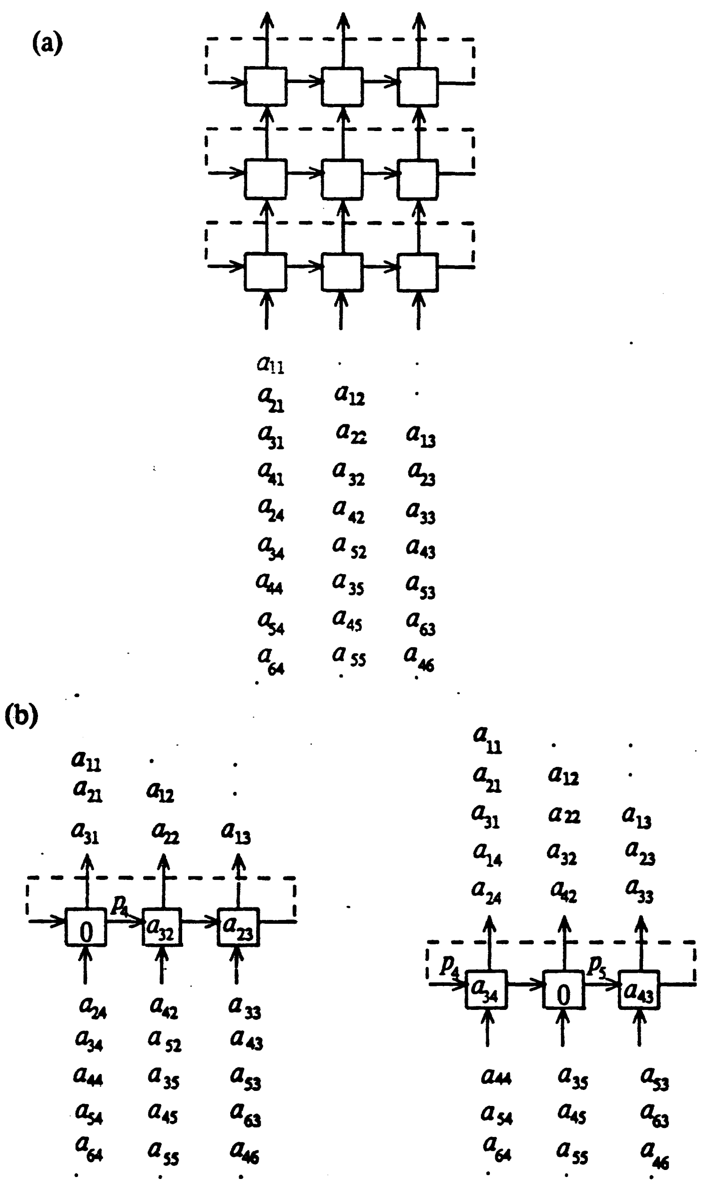

(a)
(b)
a_{11} a_{21} a_{31} a_{41} a_{24} a_{34} a_{44} a_{54} a_{64}
a_{12} a_{22} a_{32} a_{42} a_{52} a_{35} a_{45} a_{55}
a_{13} a_{23} a_{33} a_{43} a_{53} a_{63} a_{46}
0
p_4
p_5
a_{32}
a_{23}
a_{34}
a_{43}

Figure 4-7: (a) Two-dimensional systolic ring structure for matrix triangularization and (b) two snapshots of the bottommost ring

ing 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 sub-diagonal.

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. This architecture requires about half the amount of hardware and achieves the same throughput of a previous solution of QR-decomposition[26]. An efficient layout of this ring architecture is shown in Figure 4-8 (a). Every two consecutive rows correspond to one ring.

The analysis of the fault-tolerant behavior of this ring structure is very similar to the one-dimensional ring. A system with n rings can be unrolled to form a mesh-connected acyclic array with n cells on one side and an "unbounded" number on the other. The throughput rate is reduced from 1/2 to $n/(2n+k)$ if k defects are tolerated in each of the n rings in the final array. Also, by applying theorem 2, we can simplify the correctness constraints on the final configuration to get a local criterion that has to be satisfied by each unit square in the logical grid. This criterion is depicted in Figure 4-8 (b).

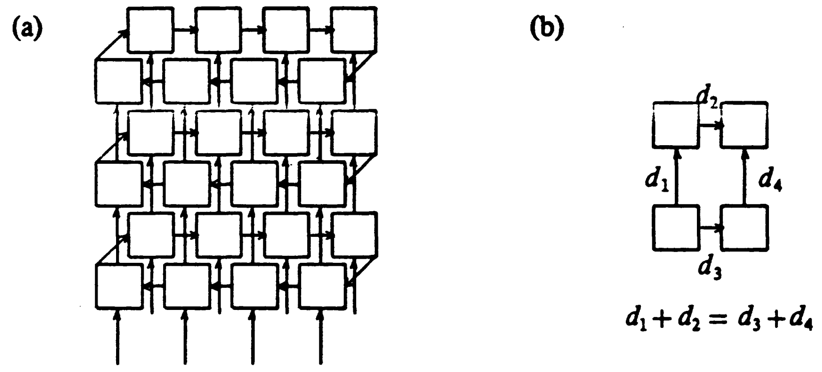


Figure 4-8: (a) Layout of the two-dimensional ring structure for matrix triangularization, and (b) the local correctness criterion

4.4.3. LU-Decomposition of a Band Matrix

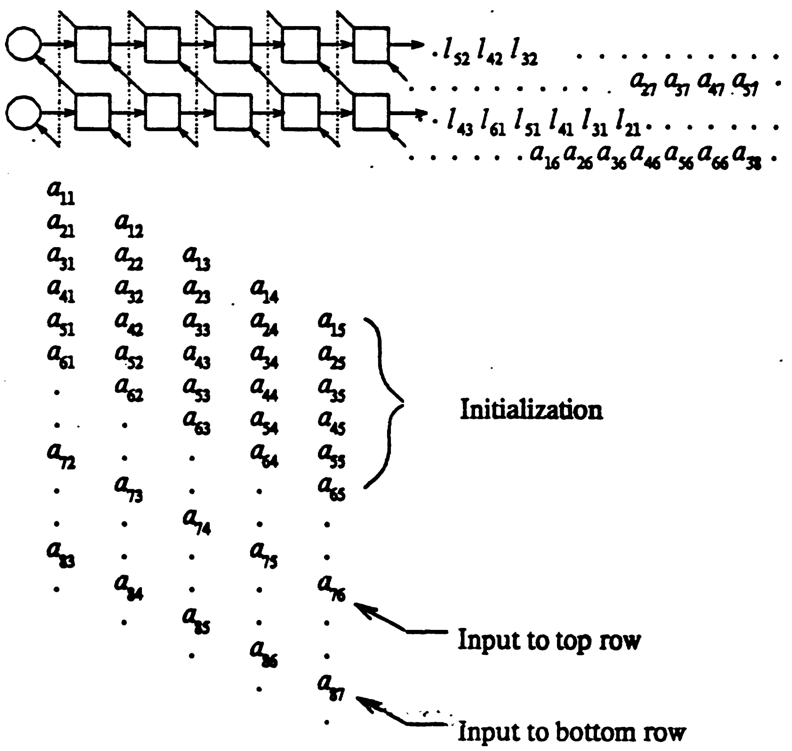

$l_{52}\, l_{42}\, l_{32}$
$a_{27}\, a_{37}\, a_{47}\, a_{57}$
$l_{43}\, l_{61}\, l_{51}\, l_{41}\, l_{31}\, l_{21}$
$a_{16}\, a_{26}\, a_{36}\, a_{46}\, a_{56}\, a_{66}\, a_{38}$
Initialization
Input to top row
Input to bottom row

Figure 4-9: Systolic ring architecture for LU-decomposition

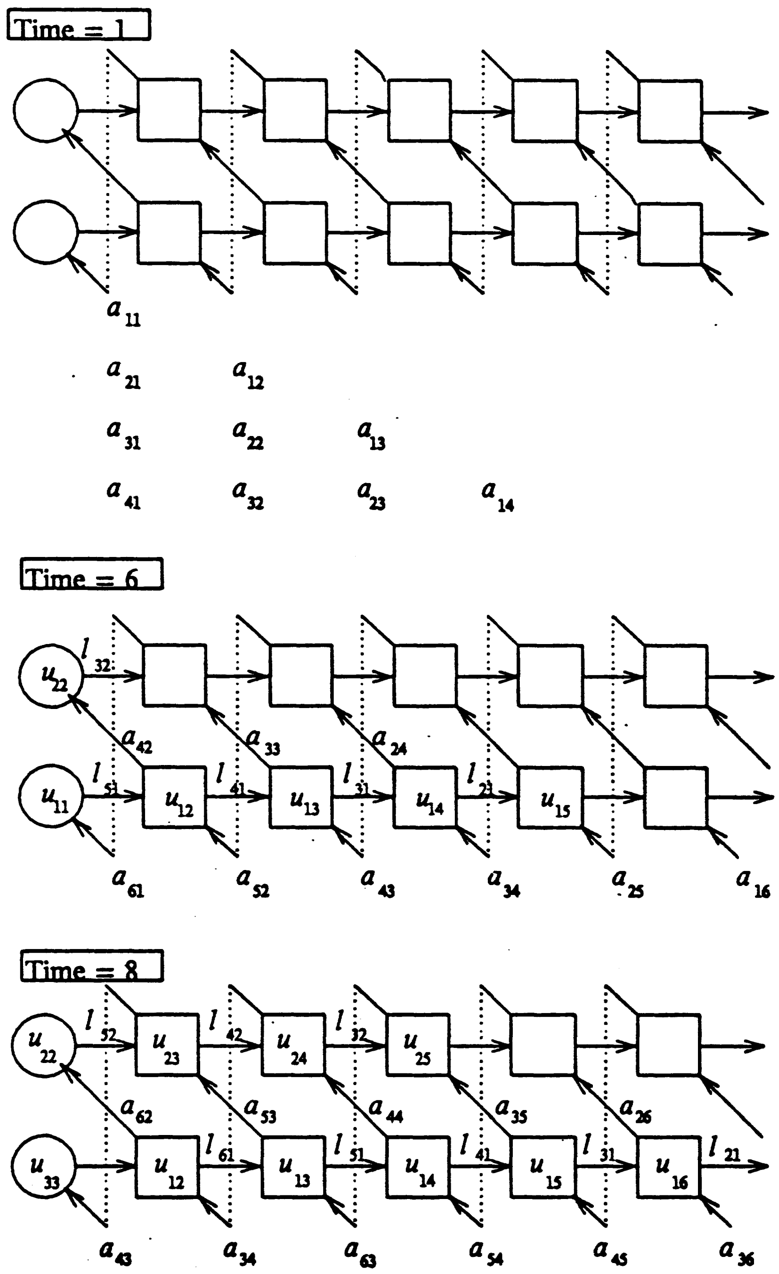

Time = 1
a_{11}
a_{21} a_{12}
a_{31} a_{22} a_{13}
a_{41} a_{32} a_{23} a_{14}
Time = 6
u_{22} l_{32}
a_{42} a_{33} a_{24}
u_{11} l_{51} u_{12} l_{41} u_{13} l_{31} u_{14} l_{21} u_{15}
a_{61} a_{52} a_{43} a_{34} a_{25} a_{16}
Time = 8
u_{22} l_{52} u_{23} l_{42} u_{24} l_{32} u_{25}
a_{62} a_{53} a_{44} a_{35} a_{26}
u_{33} u_{12} l_{61} u_{13} l_{51} u_{14} l_{41} u_{15} l_{31} u_{16} l_{21}
a_{43} a_{34} a_{63} a_{54} a_{45} a_{36}

Figure 4-10: Snapshots of a ring architecture for LU-decomposition

Figure 4-9 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 4-10 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 4 with parameter $p=2$. The throughput of this array is the same as the previous design[25] which uses, however, three times as many cells.

This two-dimensional ring architecture admits of a surprisingly efficient layout. See Figure 4-11 (b). The numbers on the cells indicate the original row the cells are in. This layout can be obtained by the following method. Starting with the original architecture (Figure 4-11 (a)), we first bring the top and bottom rows together and get a cylindrical structure. We then expand the space between each row by one cell's length, so that if we flatten out the cylinder, the consecutive rows in the "front" and "back" surfaces will be interleaved. But before we flatten out the cylinder, we first "twist" it by one cell's length in the direction that shortens the inter-row links.

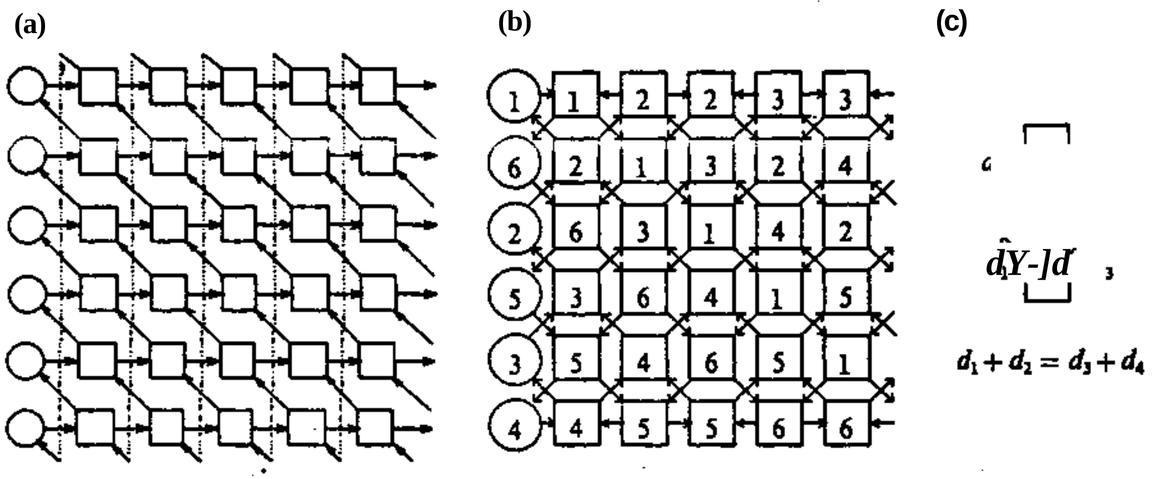


Figure 4-11: (a) Two-dimensional ring architecture for LU-decomposition, (b) its layout and (c) the local correctness criterion

By going through the analysis of the unrolled structure, we get the following results. If *k* faults are bypassed in the communication links in each of the column of connections in Figure 4-11 (b), then the throughput is reduced from 1/3 to *ti/(3n+ k)* where *n* is the number of rows in the final array. Also, the local criterion that has to be satisfied by each group of four cells is illustrated in Figure 4-11 (c).

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 4A2.

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, every cell is required to assume the role of a boundary cell. Algorithm-dependent methods can sometimes be used to alleviate the problem of having to provide each cell 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.

5. Summary and Concluding Remarks

The fault-tolerant approach proposed in this paper is tailored to systolic arrays. By using the additional information about systolic data flows we are able to design schemes that are usually more effective than other schemes designed for general processor arrays. Our systolic fault-tolerant scheme has the characteristic that the maximum interconnection length is not increased. 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, the utilization of live cells on a wafer increases with the number of redundant channels and delay registers available in the cells. The number of delay registers needed to achieve the same utilization also increases with the cell failure rate and the size of the original array on the wafer. Our empirical studies indicate for a wafer with *nxn* cells, approximately *n* delay registers per cell are needed to achieve 100% utilization.

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[9,29,30]. 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, this 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 to solve both problems automatically.

References

, M. I. T., November 1979. , bibdate = "Wed Nov 24 17:46:23 1982" ,)
, M. I. T., July 1980. , bibdate = "Wed Nov 24 17:46:59 1982" ,)

1. Evans, R.A., Wood, D., Wood, K., McCanny, J.V., McWhirter, J.G. and McCabe, A.P.H., "A CMOS Implementation of a Systolic Multi-Bit Convolver Chip," *VLSI '83*, Anceau, F. and Aas, E.J., eds., North-Holland, August 1983, pp. 227-235.

2. Kung, H.T., "On the Implementation and Use of Systolic Array Processors," *Proceedings of International Conference on Computer Design: VLSI in Computers*, IEEE, November 1983, pp. 370-373.

3. Symanski, J.J., "NOSC Systolic Processor Testbed," Tech. report NOSC TD 588, Naval Ocean Systems Center, June 1983.

4. Yen, D.W.L. and Kulkarni, A.V., "Systolic Processing and an Implementation for Signal and Image Processing," *IEEE Transactions on Computers*, Vol. C-31, No. 10, October 1982, pp. 1000-1009.

5. Smith, R.T., Chlipala, J.D., Bindels, J.F.M, Nelson, R.G., Fischer, F.H. and Mantz, T.F. , "Laser Programmable Redundancy and Yield Improvement in a 64K DRAM," *IEEE Journal of Solid-State Circuits*, Vol. SC-16, No. 5, October 1981, pp. 506-514.

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

7. Woo, B., Lin, L., and Owen, R.E., "ALU, Multiplier Chips Zip through IEEE Floating-Point Operations," *Electronics*, Vol. 56, No. 10, May 19 1983, pp. 121-126.

8. 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

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

10. 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.

11. 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.

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

13. 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.

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

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

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.E1., "Configuration of VLSI Arrays in the Presence of Defects," Tech. report, Information Systems I^b, Stanford University, May 1983.

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

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

20. 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.

21. Rabiner, L.R. and Gold, B., *Theory and Application of Digital Signal Processing.* Prentice-Hall, Englewood Cliffs, New Jersey, 1975.

22. Stone, H.S., "Parallel Processing with the Perfect Shuffle," *IEEE Trans. Computers,* Vol. C-20, February 1971, pp. 153-161.

23. 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.

24. 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.

25. 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.

26. 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.

27. 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.

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

29. Guibas, LJ., Kung, H.T. and Thompson, CD., "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.

30. 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.