THE UNIVERSITY OF MICHIGAN

COMPUTING RESEARCH LABORATORY[1]

A CLASS OF CELLULAR ARCHITECTURES TO SUPPORT PHYSICAL DESIGN AUTOMATION

R.A. Rutenbar, T.N. Mudge and D.E. Atkins

CRL-TR-10-83

FEBRUARY 1983

Room 1079, East Engineering Building
Ann Arbor, Michigan 48109
USA
Tel: (313) 763-8000

[1]This work was supported in part by NSF Grant No. MCS-8009315 and MCS-8007298. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the funding agencies.

A Class of Cellular Architectures to Support Physical Design Automation

by

R. A. Rutenbar, T. N. Mudge and D. E. Atkins

Computing Research Laboratory
Department of Electrical and Computer Engineering
University of Michigan
Ann Arbor, MI 48109

Abstract

Special-purpose hardware has been proposed as a solution to several increasingly complex problems in design automation. This paper examines a class of cellular architectures--called *raster pipeline subarrays*--applicable to physical design automation problems represented on a cellular grid. Machines with this architecture were first employed for cellular image processing, and many similarities exist between problems in grid-based DA and problems in cellular image processing and pattern recognition. A review of machines designed for cellular image processing shows how DA machines proposed/constructed for grid-based problems fit naturally into a taxonomy of image processors; a review of some of the mathematical tools developed to formalize pattern recognition problems shows how they can be usefully applied to DA problems. Implementations of design rule checking and routing algorithms are described in detail for an existing raster pipeline subarray machine called a *cytocomputer*. Experimental results using this hardware are encouraging, and extensions to large, practical problems are studied. Based upon these studies we define the architecture and necessary performance characteristics of a raster pipeline subarray machine optimized specifically for grid-based DA applications. The merits of such an architecture are evaluated in the context of practical special-purpose hardware.

Key words--design automation, special-purpose hardware, design rule checking, routing, image-processing.

1. Introduction

The successful implementation of increasingly complex integrated systems has been made possible only because of the existence of increasingly sophisticated CAD tools. Traditional CAD research--for example, the mathematical analysis of design automation (DA) algorithms and data-structures, the application of software structuring techniques to chip layout, and the use of data bases to manage the design process--has resulted in the production of software tools running on conventional serial computers. These tools are limited in three fundamental ways: by the inherent complexity of the problem, by the efficiency of the coded implementation, and by the resources of the machine on which the code runs. Recently, to overcome these three limitations serious attention has been given to the application of special-purpose hardware to CAD problems [1]-[18]. The strategy is to map the structure of the problem onto a specially designed architecture to avoid these three limitations by optimizing the hardware to exploit the parallelism inherent in the problem, by replacing some software with hardware and firmware, and by including precisely those resources critical to the solution of the problem.

In this paper we will examine a class of architectures suitable for physical design problems represented on a fixed cellular grid. Problems such as design rule checking (DRC), device extraction, placement and routing are often represented and solved in the framework of a cellular grid. An immediate candidate architecture for these sorts of problems is the cellular array. Indeed, advances in technology have heralded a renaissance for the cellular array. However, traditional cellular architectures--rectangular arrays of simple processors with memory and connections to neighbors--are not the only machine organization capable of efficient solution of grid-based DA problems. In fact, architectures for solving grid-based problems have been studied extensively in the field of cellular image processing. Useful parallels may be drawn between architectures for cellular image processing and for grid-based

This work was supported in part by NSF grants MCS-8009315 and MCS-8007298.

physical DA; such architectures are essentially characterized by the manner in which storage and processing power are allocated to each cell in the problem. As part of our examination this paper reports the results of DA experiments performed with a cytocomputer[1], a special-purpose architecture originally designed for image processing applications [19]-[21]. The discussion sets in context the results summarized in [15]. A cytocomputer is a pipeline of 3 × 3 subarray processors accepting image data as a raster input stream and producing a raster output stream. For the purposes of this discussion, we shall refer to the class of architectures which includes the cytocomputer as *raster pipeline subarray architectures*. The results presented here indicate that raster pipeline subarray architectures represent an architectural alternative that in many situations delivers a more cost-effective solution than the traditional array approaches.

The paper is organized as follows. The image processing analogy is enlarged to show how an explicit taxonomy of image processors effectively categorizes the diverse collection of special machines proposed/constructed to solve grid-based DA problems. We also show how some of the mathematical tools developed to solve pattern recognition problems of image processing applications can be fruitfully applied to DA problems; in particular a formalism using the morphological operators of Serra and Matheron is discussed. With this background, cytocomputer architecture is considered in detail. Next, cytocomputer implementations for design-rule checking and routing are examined. Experiments were performed using a Model II cytocomputer configured with one processing stage[2] and controlled by an interpreter that allows the machine to be used interactively for algorithm development. Concrete algorithms and execution times are given for width/spacing checks applicable, for example, to the simple Mead and Conway NMOS rules [22], and for a maze-router connecting two-point nets in a single conductor layer. Extensions to more complex problems are

---

[1] *Cytocomputer* is a trademark of the Environmental Research Institute of Michigan for image processing systems.

considered, and the strengths of the raster pipeline subarray are evaluated in the context of DA problems. Based upon these studies, we propose an outline for an optimized DA machine with a raster pipeline subarray organization.

2. Cellular Architectures and Algorithms

Research in special-purpose architectures for image processing spans more than two decades and has accelerated with recent advances in hardware technology [23]-[28]. Although we have remarked that the analogy between image processing and grid-based DA problems is conceptually useful, it should not be taken too far. For example, the statistical inference and frequency-domain signal processing component characterizing much image processing work is wholly absent in DA work. Our primary interest is in structural pattern recognition and pattern manipulation. From this slightly different perspective we construct a taxonomy of image processors emphasizing:

(1) how storage is allocated to the cells of an image being processed,

(2) how processing power is applied to cells or groups of cells,

(3) how processing elements and storage elements are interconnected.

It will be shown how many grid-based DA architectures fit naturally into this scheme. In particular, the place of the cytocomputer is described in this scheme. After this discussion of hardware, mathematical tools developed for pattern recognition are described along with their application to DA work.

2.1. DA Architectures in an Image-Processor Taxonomy

An essential problem in image processor architecture is *windowing* [26]. Because real images span the range 10^2 to 10^5 cells or pixels (picture elements) on a side, it is generally impossible to allocate a unique physical processor to each cell

---

[2] Multiple processing stages are essential for practical applications. Results of earlier work on design rule checking using a simpler 88-stage TTL prototype of a Model I cytocomputer appear in [13],[14].

in the image; rather, the image must be manipulated in subsections or windows. The shapes of these discrete sections, their acquisition, their path to and from processing elements, and the amount of parallelism in data-movement and data-manipulation define the architecture.

Figure 1 shows a taxonomy emphasizing these features. It has three salient points. First and foremost, because our primary interest is in DA architectures and their classification, this scheme is just large enough to contain most of the interesting grid-based DA architectures of which we are aware; image processing architectures (interesting in their own right) which have no close analogue in DA machines (e.g., pyramid machines) have simply been left out. Second, it is explicitly a hierarchical classification in contrast to other classifications [23], [26]. To specify a machine by its parents in the hierarchy gives its concise relationship to other machines, highlighting critical similarities and differences. Third, and related to this hierarchy, it places cytocomputers in the hierarchy to show their natural relationship with subarray organizations, rather than in a separate category disjoint from all other machine organizations.

At the first level the hierarchy divides into two basic machine organizations. As noted in [23] there are machines whose architectures are dominated by a central bus structure or, more generally in our terminology, by an Interconnection-Network (ICN) structure (see Fig. 2). The other basic organization is, as expected, the array structure (see Fig. 3). By array structure we mean specifically the existence of one or more rectangular arrays of interconnected processor/storage elements and the machinery to move data through these arrays. Each of these two basic organizations are subdivided into two classes.

ICN structured machines are classified as using either a single bus or a routing-network. For example PICAP II [29] employs a single high-speed bus to connect image memories, a neighborhood processor, and a filter processor. On the other hand,

the proposed PASM architecture [30] plans to employ multi-path routing-networks to connect a set of processor/memory subsystems.

The class of array-structured machines is also divided into two subclasses. Adopting Preston's terminology [26], array structured machines are classified as being subarrays or full-arrays. The distinction here is somewhat unclear, depending not only on structural differences, but also on the size of the array involved. The full-array is what would be labeled a traditional cellular array: a matrix of processor/memory pairs each connected locally to its neighbors. Machines in this class include CLIP4 [31], a 96 × 96 array of simple bit-processors each with a 32 bit RAM; the Distributed Array Processor (DAP) [32], a 64 × 64 array with 4K-bit storage per processor; the Massively Parallel Processor (MPP) [33], 128 × 128 processors with 1K-bit storage per processor; and the Adaptive Array Processor (AAP) [34], whose building block is a single chip 8 × 8 array with 96 bits of storage per processor. Some of the machines with large memories in each cell (e.g., MPP and DAP) incorporate some notion of image-folding; images larger than the physical array are folded into several planes and mapped onto the storage available in each cell. In addition, DAP provides for limited global communication by including an additional bus for each row and column of cells, enabling complete row-vectors and column-vectors to be accessed and moved around the array.

The subarray class is also subdivided, and is characterized by the range of subarray sizes and the connections between distinct subarrays. A subarray is just an array which is usually much smaller than the images to be processed; it is a processing window. The smallest subarray is a single neighborhood, 3 × 3 on a square grid, while the largest is generally in the region between 16 × 16 and 32 × 32. The simplest subarray is the class of raster single-subarrays (see Fig. 4). The basic idea is to process the image data in a serial stream (raster order) as it passes by a subarray processor. To do this, shift registers are introduced as buffers for a few rows of the image. As the stream passes through the buffers and the processor, enough of

the image is present to insure that each neighborhood eventually arrives at the subarray and is processed. The raster idea is especially attractive considering that the source of many real images is serial: disks, CCD-cameras, or host-computer-memory. The GLOPR machine [35] is a very early example of this organization. The subarray class is not restricted to a single subarray processing element; the second subclass contains the multiple-subarray machines. It too is subdivided. Because the single subarrays just discussed can output a data stream with a format identical to the input stream, it is possible to connect several such machines in a pipeline, the output of each being chained to the input of the next. Here the individual processors are called stages, and the entire machine is a raster pipeline subarray (see Fig. 5). This is the organization of a cytocomputer, the subject of the next section. Multiple subarrays are not restricted to a raster input format. More generally these non-raster organizations use several interconnected subarray memories and subarray processors to concurrently process several pieces of an image (see Fig. 6). It should be clarified here that the major difference between these machines and the apparently similar ICN based machines is a matter of emphasis. Of primary interest in multiple subarrays are the number, size, and speed of the subarray buffers and processors, with their interconnections being of secondary interest. In ICN structures much of the architecture is subordinated to the interconnection scheme. A machine in the non-raster multiple subarray class is the Preston-Herron Processor (PHP) [36], in which three image memories communicate with sixteen table-driven processors.

We have provided a concise overview of image processing architectures to classify the diverse set of proposed/constructed DA machines. It will be shown that many grid-based DA machines are related through the previous taxonomy. The existence of some superficial architectural similarities between these two classes of machines is not surprising given some of the similarities between cellular image processing tasks and grid-based physical DA tasks. However, it is significant that for most of these DA machines there is a precise analogue in the image processing world.

This implies that future DA hardware can profit from the work being done on image processing architectures by applying and extending this work rather than rediscovering it. Our primary motivation is to develop the relationship between raster pipeline subarray architectures and other DA architectures.

Full-array structures have been particularly popular for physical DA. Breuer and Shamsa [7] have proposed a single chip 256 × 256 array of finite-state machines to perform unit-cost Lee routing and a multi-chip 1024 × 1024 machine. Iosupovicz [12] discusses the details of such a routing machine based on an interconnection of smaller more modular building block chips. Adshead [2], [3] has reported successful application of the DAP machine to problems in maze-routing and logic simulation. Routing involves folding the cell-map representing a large gate-array onto the physical array; simulation involves coordinating updates to the states of logic elements distributed around the array. Blank [5], [6] has proposed two array architectures for solving general bit-map DA problems: a Bit Map Processor (BMP) and a Virtual Bit Map Processor (VBMP). A BMP is a standard array of 1024 × 1024 simple processors each with memory. A VBMP is a 32 × 32 array incorporating special hardware to support folding a larger virtual grid onto the physical array. Large amounts of memory, 1K-bits for accumulators and 16K-bits for general registers, reside at each node of the matrix, and the edge and corner cells include special mechanisms for dealing with border effects and neighborhoods straddling physical boundaries. Each cell in the array can also be individually addressed via row and column lines to provide some global communication. Simulations for grid-based design-rule checking and simple maze-routing have been constructed, and a prototype cell similar to a single BMP array cell has been fabricated. Hong et al. [11] describe a Physical Design Machine based on an array of commercial microprocessors, which also incorporates provisions for folding large problems onto the array. An 8 × 8 prototype with 15K bytes per processor is operational, and claims are made that a 32 × 32 structure would likely suffice for all real problems. Sophisticated global-routing algorithms have been

implemented and run on modest test grids [16].

Bus structured machines have also been constructed. Damm et al. [9] have built a Lee-routing engine by modifying a commercial minicomputer. A special cell-memory, a hardware "kernel" of routing operations, and an additional bus interconnecting them were added to optimize routing performance. Successful operation with printed circuit boards has been reported. (Outside the area of grid-based physical DA, the Yorktown Simulation Engine (YSE) [17], an event-driven logic-simulator, is an ICN structured machine. Up to 256 logic processors, each storing and updating logic elements, communicate over a cross-bar switch.)

Subarray architectures also appear. Seiler [18] has developed a hardware implementation of Baker's raster DRC [37] using a raster subarray. The processing section uses a few custom PLA-based chips to perform width checks, edge checks, and logical combination of mask layers in a small window. A feedback mechanism with shrink/expand templates is provided to enable larger width checking using multiple passes through the processor. The raster pipeline cytocomputer [13]-[15] has been used to perform DRC and routing; these algorithms are the subject of succeeding sections.

2.2. Cytocomputer Architecture

With this background, we proceed to describe cytocomputer architecture in detail. The difficulty in producing a cost-effective full-array architecture for image-processing applications led the Environmental Research Institute of Michigan (ERIM) to develop the alternative architecture of the cytocomputer. In [20] Lougheed and McCubbrey detail some of the practical advantages accruing to a low-complexity high-bandwidth cytocomputer image processor structure when compared with full-arrays. In the following we describe the system-level architecture of a cytocomputer, the structure of each stage, and the performance characteristics of the machine on which our experiments were done.

Figure 7 shows a cytocomputer at the system level. Note that the machine is connected as a peripheral to a host computer. The host sends instructions to the cytocomputer controller, a microprogrammed unit that programs the individual stages in the pipeline and manages the movement of data through it. An image can move through the cytocomputer in two ways: (1) the host sends an image to the image buffer, then the controller repeatedly sends it through the pipeline and back into the buffer until processing is done, and finally the host retrieves the processed image from the buffer; (2) the host sends the serial image stream directly into the pipeline, managed by the controller, and receives the output stream directly. As expected, the second approach is slower because the host's operating system mediates much of the transfer.

Figure 8 shows the structure of a cytocomputer stage. Common to both the cytocomputer and full-array, each discrete processing step consists of transforming each cell in an image based on its current value and the values of its neighbors. A cytocomputer datapath is a serial pipeline of subarray processing stages with a common clock in which each stage performs a single *neighborhood transformation* on an entire image. Images enter the pipeline as a stream of eight-bit pixels in sequential line-scanned format and progress through the pipeline of processing stages at a constant rate. Shift registers within each stage store two contiguous scan lines while window registers hold the nine neighborhood pixels which constitute the 3 × 3 subarray input of a stage. At each discrete time step a new pixel is clocked into each stage. Simultaneously, the contents of all shift registers are shifted one element. Each stage performs a programmed transformation of the center pixel based on the values of the center and its eight neighbors. Neighborhood transformations are computed within the data-transfer clock period, allowing the output of each stage to appear at the same rate as its input. Following the initial delay to fill the pipeline, processed images are produced at the same rate they are entered.

To visualize the transformation process, consider a 3 × 3 window moving across an image as shown in Figure 9. Cell $A_{6,6}$ in the raster stream has entered the stage and the complete 3 × 3 neighborhood of cell $A_{5,5}$ is now stored in the stage subarray. (That the cells appear reflected in the subarray is of no consequence; the requirement is that *all* cells of a neighborhood be available in the subarray.) Performing a single neighborhood transformation now produces the modified value $A_{5,5}^{new}$ which is injected into the output stream. On the next cycle, $A_{6,7}$ enters the stage and the computed value $A_{5,6}^{new}$ is output. In this example the latency of a stage--the number of cycles between $A_{i,j}$ entering and $A_{i,j}^{new}$ leaving--is always the same as the number of cells in the raster stream between $A_{5,5}$ and $A_{6,6}$ (shaded in Fig. 9). For an image N cells wide this latency is N + 2 clock cycles. A pipeline can be viewed as a series of 3 × 3 stage windows following each other across the image, each one processing the previous stage's output.

The following steps outline the processing that occurs in one stage in one clock period [19] (refer to the numbered steps in Fig. 8):

(1) A pixel entering the stage is biased (normalized) or has some of its bits masked.

(2) The nine cells of the stored neighborhood are transformed into a nine-bit vector. Each bit is essentially a true/false decision about each neighbor: $cell_5 = constant_5$, $cell_0 > threshold_0$, for example. This vector is an address into a table.

(3) A new value for the center cell is selected from the following: the current value of the center, the largest value in the neighborhood, or the value in the table addressed by the nine-bit address just computed.

(4) The center value from (3) is used to address another table to produce a modified center value. This table is used typically for Boolean operations on bits or for further arithmetic biasing of this value.

(5) The center value from (4) is unbiased and unmasked. It is possible, for example, to alter a single bit in the center cell as a function of all the bits of the neighborhood without disturbing the other bits in the center.

These bias-values, masks, constants, and tables comprise the instructions for a single stage. Because the stage depends heavily upon table look-up the format of the data in each cell is arbitrary: each 8-bit cell can be viewed as eight independent 1-bit fields, as a few multi-bit fields, or as a single encoded integer.

Equation 1 approximates the time, T, required for a cytocomputer with P pipeline stages to perform K operations on each 3×3 subarray of an $M \times N$ image. Let t_c be the stage cycle time in sec/pixel, then:

$$T \approx \lceil K/P \rceil (P(N+2)t_c + MNt_c). \quad (1)$$

The $\lceil K/P \rceil$ term is the number of passes through the pipeline. The last term is the time required to pass once through the pipeline. It is the sum of two terms: the time until an output appears at the last stage of the pipe (called the *latency*), and the time to move all MN cells of the image through the pipeline. If K is not a multiple of P, then the unused processing stages in the last pass through the pipeline are programmed to pass the image stream with negligible delay.

Cytocomputers exist in both MSI and LSI implementations. Existing MSI implementations usually have between one and ten stages. A single chip CMOS version of a single stage, excluding line buffers, has been fabricated. The chip incorporates all functions of the previous generation MSI stage hardware and has a cycle time $t_c \approx$ 1-2 μs. Because of the considerable size reduction, a 100-stage cytocomputer pipeline will fit on a modest number of circuit boards. All experiments reported in this paper were performed on a single-stage MSI TTL prototype of this chip with $t_c = 2$ μs and a 256K-byte image buffer.

2.3. Formalism for Cellular Algorithms

We shall briefly describe a formalism called *mathematical morphology*, developed by Serra and Matheron [38], [39], which introduces several useful operators for dealing with patterns in images and also introduces an algebraic framework in which to manipulate them. It treats a two-dimensional binary image as a set of points (e.g., the opaque points on a transparent mask) where a point is an ordered pair of coordinates in the plane. In our case the plane is a digital rectangular grid; this is a practical but not a mathematical necessity--practical insights may be gained from the study of ideal problems in the continuous Euclidean plane.

Given that A and B are binary images, and hence sets, we have all the usual set-theoretic operators (i.e., the usual Boolean operators): intersection, $A \cap B$; union, $A \cup B$; difference, $A - B$; and complement, $\bar{A}$. Also, the usual operations of addition and scaling can be applied to points represented as coordinates. Next, define the notion of *translation* of a set by a point. The set A translated by point p, denoted A_p is:

$$A_p = \{ a + p \mid a \in A \},$$

which is just A with its *local* origin moved to p. With translation we can define the two essential primitives of *dilation* and *erosion*. Dilation, $\oplus$, and erosion, $\ominus$, are defined as follows:

$$A \oplus B = \bigcup_{b \in B} A_b \qquad A \ominus B = \{ p \mid B_p \subseteq A \}$$

The dilation $A \oplus B$ is the union of translations of A by points from B. The erosion $A \ominus B$ is the set of points to which we can translate B and still have it contained in A. Loosely, we can think of dilation and erosion as formal generalizations of the intuitive ideas of expanding and shrinking. However, erosions and dilations are defined for arbitrarily complex sets A and B, whereas expands and shrinks are usually specified with simple patterns (sets).

The utility of this formalism rests on the fact that all these operations are *local* in nature. Each operation can be reduced to a sequence of neighborhood transformations in the 3 × 3 subarray of a cytocomputer stage. Algorithms can be designed at a high level and then decomposed into an executable sequence of cytocomputer operations, thus minimizing the need for ad hoc pattern specification.

To illustrate the above ideas, consider Figure 10 in which image A is eroded by the elementary image B. The points in the result are those points in A to which the origin of B (marked + in its center) can be translated and still have B contained in A. A cytocomputer stage implements this operation by a straightforward template match on each 3 × 3 subarray of pixels; the central pixel is set to 1 just if the neighborhood configuration contains B. In this example B is an elementary image (also called a structuring element [38]) because it fits in the subarray of a stage. In general, operations with a larger more complex B will be decomposed into a sequence of elementary operations suitable for execution in the stage hardware. The algebra includes identities which permit simplifications similar to those done in Boolean algebra. For example, to compute $A \ominus B$ when B is a dilation of elementary images:

$$B = B_1 \oplus B_2 \oplus B_3 \oplus B_4$$

it is sufficient to compute:

$$A \ominus B = ((((A \ominus B_1) \ominus B_2) \ominus B_3) \ominus B_4)$$

which can be done directly by four stages.

Two operators defined as compositions of the dilation and erosion primitives are also particularly useful, as will be seen in later sections. These are called *opening* and *closing*. If X and S are sets, then:

$$X \textit{ opened by } S = X_S = (X \ominus S) \oplus S$$

$$X \textit{ closed by } S = X^S = (X \oplus S) \ominus S$$

Interpret sets X and S as geometric shapes in a plane. Then X opened by S is the set of points in X touched by S as S slides around inside X. Closing has a similar

interpretation for the complement of shape X.

Figure 11 demonstrates all these operations on two simple plane figures.

3. Design Rule Checking

The design rules are a set of geometrical constraints that the masks of the wafer fabrication process must satisfy. The two general approaches to the implementation of DRC's reflect the data-structure chosen for the IC mask. Geometric-shapes checkers perform checks on masks represented as sets of intersecting polygons or rectangles [40]. Grid-based checkers work with a mask represented as a grid whose cells are labeled according to the presence or absence of particular mask layers. Both nonuniform grids (the chips are dissected into contiguous rectangles of arbitrary size, e.g. [41]) and uniform grids (the cells are squares) have been used. Raster-scan approaches have been developed [37], [42] which access a uniform grid in raster order and check *local* design rules; the idea is to pass a small window over the grid and identify the local violation-patterns appearing in the window.

Roughly speaking, a design rule checker performs the following on the geometric shapes comprising a mask:

> **Connectivity Resolution:** discrete shapes on the same layer are merged into a single larger shape if they overlap; connectivity is similarly assessed across several layers, e.g., across contact windows.
>
> **Layer Combination:** new layers are created from Boolean combinations of existing layers, e.g., the intersection of several layers.
>
> **Tolerance Checks:** tests are made to determine whether a local group of shapes on one or more layers satisfies some spatial constraint, e.g., corner/edge separation, incursion, inclusion, exclusion, size, area, perimeter.

When a mask is represented as a grid, local connectivity and layer combination are easily computed. Overlapping shapes automatically become a single entity as the

cells within the shapes are labeled as belonging to a particular layer, and Boolean combinations performed globally across several layers are simply performed on each cell in the mask. Global connectivity is harder to resolve because it involves propagating nodal connectivity information around the cells of the grid. Such global data movement is generally inefficient if we are restricted to local processing of cells. Tolerance checks are more interesting since they require not just cell by cell processing but also pattern recognition operations on spatially distributed groups of cells.

Accordingly, this section describes tolerance checks implemented on a cytocomputer. The checks are motivated by the NMOS design rules of [22]; we employ a uniform grid of $\lambda \times \lambda$ cells where λ is the basic length unit in which design rules are expressed.

3.1. DRC Algorithms

To illustrate the cellular DRC, we describe two different algorithms to perform a width-3λ tolerance check on an orthogonal mask. This check identifies regions of a mask less than 3λ wide.

In these algorithms a single mask is represented by a binary image occupying one bit in each 8-bit cell of the cytocomputer input image. Each algorithm produces another binary image, stored one bit per cell, indicating the locations of width violations. In a complete DRC all masks of interest--metal, polysilicon, diffusion, for example--are stored in these parallel bit-planes. The 8-bit cytocomputer datapath allows up to eight bit-planes--input masks, intermediate results, final error locations--to be processed simultaneously.

The first algorithm is based on the simple observation illustrated in Figure 12. At the left is a mask represented in the continuous plane on which we wish to perform a width-W check. Let a disk of diameter W slide around the inside of the mask to all possible locations at which it may be completely contained. This is illustrated in the

middle of Figure 12. While it slides, note those points covered by the disk and trace the path of its center. It is clear the disk should not pass through regions which are too narrow, i.e., regions which fail the width test. The result is shown at the right of the figure. Except for some square-corner effects, those regions left uncovered all violate the width test. Note also that the region traced out by the center of the disk is not connected across the diagonal neck of the mask.

With these observations one can construct an executable width-3λ algorithm. Let M be the mask-image to be checked. Because a real mask will have square corners, replace the diameter-3 disk with a 3 × 3 square, S. The operation of finding all points covered by S as it slides in M is precisely the opening $M_S = (M \ominus S) \oplus S$ from the previous section. The region traced out by the center of S is just the erosion $M \ominus S$; call this C. The algorithm can be outlined as follows:

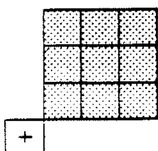
Algorithm 1 : Width-3λ Test

Step 1: Open M with S. Areas of M not in the opening are errors.

Step 2: Erode M by S to get C. This is the path traced by the center of S.

Step 3: Tag the northeast corner of each component of C; call this set of points T_{NE}. This prepares to identify the regions of M which restrict the passage of S by finding the breaks in C.

Step 4: Dilate T_{NE} over M with the following shape (marked + at its origin):



This dilation intersects the southwest corner of another region of C if and only if a break has occurred along a northeast/southwest axis. Mark the points in this intersection as errors; they indicate a diagonal width violation.

Steps 5,6: Similar to steps 3,4 to find errors along the northwest/southeast axis.

This algorithm tags any region smaller than $3\lambda \times 3\lambda$, and tags the *north* side of each pinched-neck diagonal width violation. It is generic in the sense that it can implement any width-W check. For a width-W check, S is a W $\times$ W square and is used to locate regions smaller than W $\times$ W. The figure of step-4 is a digital approximation of a radius-W quarter-circle and is used to find breaks in C, the path traced by the center of S. Breaks are characterized by diagonally adjacent corners of components of C separated by no more than W. For example, if we mark one set of corners, T_{NE}, the northeast corners, and then search nearby each for an unconnected southwest corner, each region searched takes the shape of a radius-W quarter-circle. Note that in the above case with $W = 3\lambda$ a radius-3λ circle is approximated as a 3 $\times$ 3 square.

The second algorithm is again based on the observation that the center of the square S will not trace a single connected region as it slides through a figure with width violations. This algorithm uses the notion of a *homotopic thinning* [38] which conceptually shrinks a figure without changing its connectivity, i.e., without introducing holes or breaking the figure into several pieces. Figure 13 illustrates a simple thinning. Thinning is implemented by sequential erosions of the mask-image conditioned so that removal of a point does not alter the local connectivity of any figure. The algorithm can be summarized as follows.

Algorithm 2 : Width-3λ Test

Step 1: Open M by S. Any regions of M not in the opening are errors.

Step 2: Erode M by S to get C, then thin the opening from Step 1 and compare to C. Regions present in the thinning and not on the erosion mark diagonal width errors, i.e., they bridge the gaps between the disconnected blocks of C and identify the errors.

These algorithms illustrate the utility of the formalism presented in the previous section. Algorithms are expressed as sequences of operators working on geometric objects; altering the size of these objects does not alter the basic algorithm. These abstract operators are formally decomposed into a set of concrete operations that can be performed by the hardware. This formalism frees these algorithms from much of the tedious detail of pattern specification inherent in pattern matching.

3.2. DRC Experimental Results

Algorithm 1 requires 8 pipeline stages but was run in 15 stages due to constraints imposed by the software that programs the stage. Algorithm 2 required 6 stages and was run in 6. Figure 14 shows the results of running each algorithm on a simple 64 × 64 test pattern. All errors are correctly detected. On the single-stage cytocomputer each of these tests required about 0.1 seconds to run. Depending on the sophistication of the checking needed and the amount of extra processing required to put meaningful labels on errors, a DRC for this class of rules on five NMOS mask layers will take between 150 and 250 stages. Table 1 estimates the time required to run such a DRC on a $2000\lambda \times 2000\lambda$ chip for several different pipeline lengths. We assume a cycle time $t_c = 1\mu s$, and a sufficient mask-data transfer rate to keep pace with the pipeline, i.e., the mask can be stored or regenerated with negligible delay for multiple passes through the pipeline.

Pipeline Length	Estimated DRC Time in Seconds	
	150-Step-DRC	250-Step-DRC
1	600.	1000.
10	60.	100.
100	8.4	10.5
250	4.5	4.5

Table 1. Estimated DRC Time, $2000\lambda \times 2000\lambda$ IC

3.3. Enlarging the Scope of Application

It is clear from Table 1 that with a modest pipeline (10-100 stages) a chip represented as a grid with a few million cells (a typical image) can be checked against rules equal in complexity to those of [22] in 1-2 minutes.

However, several potential difficulties arise if this methodology is extended to very large chips, for example to commercial microprocessors. There are two essential problems: the *size* of a large layout, and the *quality* of a DRC for such a layout.

The size issue concerns the representation of a large chips on cellular grids. For example, if f is the minimum feature size of a given layout, some commercial microprocessors when drawn on a grid of $f \times f$ squares require 10 to 15 million cells for resolution [43]; for reference, a 300 mil × 300 mil chip drawn on a 1 μm grid has about 60 million cells. Given that a complete DRC will require several passes through a practical pipeline (10-100 stages) the question is how to generate, store or regenerate the required chip image. There are at least two potential solutions:

(1) Dedicate a large disk and a large image buffer to the pipeline. Disks storing 100M-bytes with 1M-byte/sec block transfer rate are common, and the cost-per-bit of disk storage is declining. One pass through the pipeline has the following outline:

```
BEGIN
   Generate mask image at host machine and store to disk ;
   WHILE ( more segments of mask-image ) DO
       BEGIN
           Load next segment of mask into image-buffer;
           Process buffer through pipe;
           Store segment back on disk;
       END;
END;
```

A complete DRC will require several passes after initial mask image generation. The key constraints here are the mask-generation rate of the host, the I/O speed of the disk, and the size of the image-buffer. As an example, consider a

system with the parameters of Table 2. This simple sequential model gives a rough time of 1800 seconds for the data-movement and processing required to produce an error-image for this chip on the disk (1000 seconds to make the image initially, 4 × 64 iterations of a roughly 3 second disk-buffer-pipeline-buffer-disk processing cycle[3]). About 60% of this is the time needed for the host to make the mask-image. Software overhead will likely increase this by at least 50%.

(2) Dedicate special-purpose hardware to the task of mask-image generation. The preceding analysis assumes that the mask-image is fully instantiated on the disk with no encoding. If some pre- and post-processing is available at each end the of the pipe, a simple run-length coding of each line will reduce storage requirements and hence data transfer time. More complex schemes such as hierarchical bit-maps [44], [45] might also be possible. Seiler [18] has discussed a single-chip polygon-to-raster converter allowing a polygon-based mask structure to be rapidly processed by a raster subarray DRC machine.

We conclude that some combination of dedicated storage and special hardware is sufficient to manage the size problem for large chips.

System Parameter	Value
Mask-Image Size	64M-bytes (8096 × 8096)
Host Processing Rate	64K cells/sec
Disk Transfer Rate	1M-byte/sec (average)
Pipeline Length	64 stages
Stage Cycle Time t_c	1 μs
Image-Buffer Size	1M-byte
Complete DRC Length	256 steps

Table 2. DRC System Example Parameters

[3]Note that it is possible to process the mask-image in pieces *without* overlapping or extra processing to correct boundary effects. Consider that, if each stage has line buffers longer than the width of the image, the entire image can be streamed through the pipeline and processed. Partition the image into horizontal bands spanning the whole width of the mask-image. Load a band into the image buffer, process through the pipe but suspend the pipeline clock when the last cell enters the first stage, return the processed portion of the image to the disk, load the next band into the buffer, and restart the clock. This simulates an unbroken image stream.

The second and perhaps more serious problem is the issue of quality: a cellular DRC imposes restrictions on the layout of a chip and on the geometry-rules to be checked. Layouts represented with a polygon data-structure may contain features of arbitrary shape and arbitrary size; polygon checkers can usually resolve electrical connectivity and use this information in tolerance checks. Cellular checkers restrict the layout to a uniform grid, restricting all features to orthogonal boundaries (no oblique lines) and all distances to multiples of the unit cell size. Electrical connectivity is not usually available during tolerance checking, resulting in nuisance errors. These issues are addressed below.

(1) Many layouts do not require features of arbitrary size. Although memories or performance-optimized components require finer resolution of features for which grid-based checkers are inappropriate, there are common applications which have less stringent performance constraints but which still require much real estate and which can be satisfactorily layed out on a cellular grid. The recent popularity of simplified design rules in several technologies [46],[47] leads us to conclude that grid-based design may be equally acceptable.

(2) Some oblique lines are representable on cellular grids at the expense of increased storage or processing. A scheme for 45° lines using several bits-per-cell appears in [41]; the morphological operators are applied to this scheme in [48]. A method for 45° interconnect based on conditional labeling of cells appears in [18]. Most layouts are primarily orthogonal. It has been argued that obliques are a questionable luxury that may become too expensive to check in the face of VLSI complexity [49].

(3) Lack of electrical connectivity information is not unique to cellular checkers[4]. Some recent university systems, e.g. [50], [51], have chosen not to implement

[4] Connectivity extraction is not impossible here but it is expensive. Baker's raster checker [37] includes a separate node-extraction phase, but the information is not used during the actual DRC. The overhead of storing connectivity information, e.g., a node number in each cell, is expensive. Note that in a non-uniform grid [41] it is less expensive because there are fewer cells.

obliques and/or connectivity in order to pursue other research directions. Only rules based explicitly on electrical information, e.g. fanout rules, are compromised here. Purely geometrical rules of considerable complexity--reflection rules, transistor size rules, for example--checked with layer combinations and tolerance tests in commercial polygon packages are likewise checkable on a cellular grid; the only drawback is the possibility for nuisance errors due to connectivity pathologies.

Given the existence of a class of large designs that can be appropriately represented on cellular grids, the strong demand for a comprehensive DRC at the end of the design process with the accompanying long execution time for a typical software DRC (often measured in days on a mainframe, e.g. [52]), and the increasing demand for checking during design-entry, we conclude that, by providing a reasonable quality DRC in the time range 10 minutes - 1 hour even for very large problems, the special hardware examined above is a potentially useful DA tool.

4. Routing

Maze-routing is a natural application for a cellular architecture. The continuing viability of Lee-type routers in both PC board and LSI applications is indicated by recent surveys [53], [54]. Much research has focused on modifications of the basic Lee algorithm [55] seeking to improve the efficiency of software implementations [56]-[58]. This section describes a cytocomputer routing experiment using a specially designed version of the Lee algorithm for two-point nets with a single conductor layer. Note that a cytocomputer router is analogous to the inner-loop of a software implementation: its only job is to find a path between two points on a grid. The host processor must deal with wire-list determination, ordering, and unroutable connections. Nevertheless, a hardware inner-loop will enhance the performance of the complete routing system.

4.1. Routing Algorithms

We treat here the implementation of a unit-cost Lee-type maze-runner. The cytocomputer's 8-bit datapath precludes a scheme with arbitrary weights (except for very small test grids where no path to be completed exceeds 256 cells in length). The activity in each cell of the routing grid is encoded into the following alphabet:

S, T	source and target cells
□, ■	free and blocked cells
←, ↑, ↓, →	arrows on wavefront pointing back to the source
T←, T↑, T↓, T→	target labeled by wavefront (path found)
ST	source labeled during target back-trace

This is essentially the encoding proposed for the array architecture in [7] where the expanding wavefront is composed of source-pointing arrows.

The routing of a single source-to-target path has three phases:

Wavefront-Expansion: iteratively expand from the source a wavefront of back-pointing arrows; arrows on one wave frontier begin a path to the source and are equidistant from the source. Continue until the target cell is reached.

Back-Trace: trace a path from the target back to the source along the arrows.

Clean-Up: remove extraneous labeled cells and relabel the new path as a future obstacle.

Each of these phases is implemented as an algorithm which changes a cell in the grid depending on its neighbors. We refer to a cell's neighbors using compass directions. It is useful here to view a cytocomputer as emulating a full-array of finite-state machines.

WAVE-EXPAND places an arrow in a cell if it is bordered by an active wavefront. Any label in $\beta = \{ \leftarrow, \uparrow, \downarrow, \rightarrow, S \}$ can be on a wave border. Each cell C in the grid is tested as follows:

```
WAVE-EXPAND:
  IF C ∈ { □ , T }
  THEN BEGIN
```

```
    IF north ∈ β
    THEN attach ↑ to C
    ELSE IF west ∈ β
         THEN attach ← to C
         ELSE IF east ∈ β
              THEN attach → to C
              ELSE IF south ∈ β
                   THEN attach ↓ to C
END
```

When more than one labeling may be chosen, the arbitrary order of tests chooses directions in the order ↑, ←, →, ↓. Note that the operation *attach* does the obvious function on the alphabet, e.g., □ attach ← = ←, T attach ↑ = T↑. Implemented on a cytocomputer WAVE-EXPAND requires one stage, i.e., a P-stage pipeline can perform P WAVE-EXPANDS in one pass. This step is repeated until the target T is labeled with an arrow. Ideally we should stop when the target cell is labeled. However, the target may actually be reached in the middle of the pipeline and we cannot simply stop the raster stream. Hence, the target will probably be overrun and a few extraneous cells will be labeled. The issue is how to determine when the target is reached. On the current hardware the best solution is to put this operation in the pipeline controller's microcode, allowing it to check the image buffer after each pass through the pipe and signal the host when appropriate. A slower approach is to return some of the image to the host after K passes through the P-stage pipe (where $K \times P \approx$ expected length of the path being routed) allowing the host to check the target and repeat as necessary. In our experiments the image was examined by the host after every 100 passes through the single stage pipeline.

BACK-TRACE simply traces the source-pointing arrows from the target back to the source, attaching a T to each cell C on the unique path traced.

```
BACK-TRACE:
  IF C ∈ β AND
    ((north = T↓) OR (west = T→) OR (east = T←) OR (south = T↑))
  THEN attach T to C
```

Unlike WAVE-EXPAND, BACK-TRACE is completely sequential. Hence, like the decision to terminate WAVE-EXPAND, it is best implemented in the controller's microcode or,

less efficiently, in the host. Nevertheless, our experiments used the pipeline for this step to avoid the overhead of removing the image from the environment of the interpreter controlling the cytocomputer. One stage implements one BACK-TRACE step.

CLEAN-UP labels the new path as an obstacle and removes all other cells labeled in WAVE-EXPAND; it requires one stage.

```
CLEAN-UP:
  IF C ∈ { ■ , T↑, T←, T→, T↑}
  THEN relabel C as ■
  ELSE relabel C as □
```

Figure 15 shows a symbolic example of these three processes.

4.2. Routing Experimental Results

Figure 16 shows the results of an experiment in which a path of length 700 was routed on a 200 × 200 grid. Although this test required about 120 seconds to run on a single-stage machine (neglecting host overhead to terminate WAVE-EXPAND), straightforward expansion to more pipeline stages drastically reduces this time. For example, a 64-stage system will require about 2 seconds for the same connection. If BACK-TRACE and WAVE-EXPAND termination are implemented in microcode, the time to route a wire becomes wholly dominated by the time to do WAVE-EXPAND. Assuming microcode support, a sufficiently large image buffer, and $t_c = 1.0\,\mu s$, Table 3 estimates the time required to route wires of varying lengths on a 500 × 500 grid as a function of cytocomputer pipeline length.

Pipeline Length	Estimated Routing Time in Seconds		
	Wire-Len 100	Wire-Len 500	Wire-Len 1000
1	25.1	125.	251.
10	2.55	12.8	25.5
100	.300	1.50	3.00
250	.376	.751	1.50

Table 2. Estimated Routing Time, 500x500 grid.

With these parameters a 64-stage machine could place 500 wires of length 100 in this grid in about 5 minutes.

4.3. Enlarging the Scope of the Application

It is clear from the preceding discussion that a machine with a modest pipeline (10-100 stages) and microcode support can route grids representing many PC boards (12 inches per side with 1/40th inch spacing is roughly a 500 × 500 grid) and small gate-arrays (1000x1000 grid) in 10-100 minutes. This section discusses extensions to larger grids and more sophisticated routers.

Expansion to larger grid sizes can be accommodated with larger image buffers. Size is less of a problem here than with a DRC: a 2000×2000 grid requires only 4M-bytes of memory. With additional microcode support it becomes unnecessary to send the entire grid through the pipeline during wavefront expansion. Because a wave-front can expand at most K cells in any direction after K WAVE-EXPANDS [59] it is only necessary to send through a P-stage pipe a grid large enough to contain the wavefront after P steps. Hence, placing a short wire in a large grid requires no more time than that required to place the same wire in a small grid.

More sophisticated routers can be accommodated at the cost of more passes through the pipeline. Multi-point nets can be implemented directly for the router of the preceding section using the strategy of [7]: first connect two points, relabel the entire path as a source, choose another point on the net as a target and route again. Multi-layer routing is possible if we store extra layer information in each 8-bit cell and use a more complex WAVE-EXPAND algorithm at a cost of two or more stages per expansion step. Via-exclusion can similarly be handled with a more complex CLEAN-UP step. These trade-offs are desirable when a more complex router produces higher quality connections and hence more completions. As proposed in [16], several non-interacting connections can be routed simultaneously on a single large grid with appropriate partitioning.

Alternative schemes, such as channel-routers and line-routers, have in many applications supplanted maze-routers. Although maze-routers offer a wide range of routing performance their slow execution rate restricts them to those last few connections unroutable by any other means. However, a hardware maze-router running faster than software implementations of the other schemes removes these restrictions and makes maze-routing an efficient scheme for use on all connections.

5. Raster Pipeline Architectures

We have used the phrase raster pipeline subarray to encompass a class of machine organizations. A cytocomputer is a particular member of this class, a machine optimized for specific pattern recognition operations. The single-pipe, 8-bit wide datapath, and table-driven organization of a cytocomputer closely matches it to its intended applications. Nevertheless, it would be surprising if this organization was optimal for DA work. In this section we propose a design for a pipeline stage more closely matched to the DA applications previously described. We also discuss the merits of raster pipeline subarray systems with respect to the practical considerations of selecting special-purpose hardware.

5.1. An Optimized Raster Pipeline DA Architecture

There are three essential characteristics of grid-based DA problems: a wide range of grid sizes including large grids with $10^8 - 10^9$ cells, a wide range of data--bits, fields, integers--required in each cell, and a wide range of processing--pattern recognition, field manipulation, integer arithmetic--required on each subarray of cells. The 8-bit table-driven structure of current cytocomputers is insufficient to handle these types of data formats and data manipulation. Moreover, the table-driven model does not simply scale up to wider datapaths. Table look-up is impractical for more than 12 or 13 bits, and hence, field manipulation achieved only with direct table look-up is impossible for wider data. Arithmetic capabilities are also limited in current

cytocomputers. This section will outline a stage structure capable of handling these problems.

A given DA algorithm is mapped into a sequence of single-stage operations performed in the pipeline; the number of pipeline passes necessary to implement the algorithm dominates the execution time for the algorithm. To minimize this time, it is desirable to incorporate as much hardware in each stage as is necessary to perform each algorithm step in one stage. For example, in a DRC it should be possible to perform pattern recognition steps on several independent mask-planes simultaneously in a single stage. Also, it should be possible to perform one WAVE-EXPAND step for a maze-router using arbitrary integer weights/penalties in a single stage.

Figure 17 shows the structure of such a stage. It resembles a cytocomputer stage in that there is subarray storage, line-buffering, and a datapath using table look-up. However, the following new features are incorporated:

Wide Datapath: 24-32 bits wide in all storage and processing sections. This accommodates the need to support several data formats in each cell.

Subarray Access: the 3 × 3 subarray window is now more than simply nine neighbors; with a 32-bit datapath the subarray can be viewed as a 3 × 3 × 32 array of bits, accessible as nine 32-bit words and thirty-two 9-bit image-planes. This accommodates the need to perform several pattern processing steps on independent mask-planes in one stage.

ALU, Field Manipulation, Table Look-Up: The datapath now has a full-width ALU adding complete arithmetic capabilities. Table look-up is still provided but only for the low-order bits of the datapath; 12-13 bit look-up (4K-8K tables) is practical. To line up data for the table, field alignment in the form of barrel-shifts in 2-4 bit increments is provided at both ends of the datapath. Integers, multi-bit fields, and bit-planes can coexist in a single cell; arithmetic, logic, and table substitution can be performed on any of these formats. Temporary

storage similar to the subarray is provided for stage-intermediate results.

Datapath Instructions: Explicit control of the flow through the datapath of a stage is provided by a controller with its own instruction-set. Each instruction operates on one *minor-cycle* of the stage clock (similar but less flexible minor-cycles exist in current cytocomputers). Several instructions are stored in a stage and executed in order. Each instruction determines the source, processing and destination of datapath operands. If storage permits, literal operands from these instructions could also be injected into the datapath.

Note that this structure resembles that of a microprogrammed bit-slice machine. The primary departures are the subarray access mechanisms, the explicit support for tables and fields, and the desire to fit *everything* on as few chips as possible.

This structure realizes the goal of minimizing the number of stages required to implement DA algorithms. Consider a DRC application: several independent mask-planes are processed on successive minor-cycles by accessing different bit-planes in the subarray and operating on each with transformations stored previously in the datapath table. A more general WAVE-EXPAND step is done in one stage: 8 cycles to determine the bordering cell with minimum/maximum weight, one cycle to add/subtract this from the central cell, and one cycle to update any necessary flags.

Table 4 gives the performance goals for such a stage. Several trade-offs are apparent. The datapath width affects the complexity of the ALU, the subarray and

Parameter	Value
Datapath Width	24-32 bits
Line-Buffer Length	4K-16K
Stage Processing Rate	1 μs
Minor-cycles (instructions)	10-12, ≈ 100 ns/cycle
Table Look-Up	4K-8K words
Field-Alignment	Barrel Shift, 2-4 bit increments
Temporary Storage	4-9 full-width words

Table 4. Performance Goals for DA Pipeline Stage

temporary storage, and the instruction storage. Overall pipeline rate impacts the number of feasible minor-cycles, and the line-buffer and table access times. With a semi-custom implementation (gate-arrays and 64K-bit memories) a single stage will require about 10 chips. With a custom implementation, large memories (e.g., 256K-bit to 1M-bit chips [60]), and relaxed pin constraints [61] a 3-chip stage is possible: one chip each for the line-buffers, the stage-processor, and the tables.

We have not yet addressed the appropriate length for a pipeline of these stages. Most applications argue for very long pipes, for example, to handle efficiently the placement of long wires in a grid. However, it is usually not the case that all stages are required at all steps of the algorithm. Global decisions may be necessary after short processing sequences, the grid size may need alteration (e.g., for routing with dynamically changing bounding perimeters), or a temporary image may need to be stored somewhere. For these situations a long pipeline will be underutilized. The solution shown in Figure 18 employs multiple shorter pipelines. Here, pipeline k can be connected to pipeline (k+1) mod 2 to form longer pipes. More importantly, several short pipes can be concurrently performing different tasks: short DRC steps, independent path connections, for example. (Conceivably, several independent users can have a short dedicated pipe if appropriate multiple I/O channels are available.) This organization requires only the addition of switching multiplexors at the front of each pipe, and a small *format-processor* to choose which bits of which streams are placed in the single final output stream. The complexity of such a system is not excessive; assuming a 3-chip stage, a 1-chip switch, and a 10-chip format-processor a subsystem with four 32-stage pipelines will require about 400 chips.

5.2. Practical Considerations for DA Architectures

Several criteria are available with which to evaluate the merits of any proposed special-purpose machine [2]. Practical trade-offs among cost, speed, expandability,

and range of application are commonly considered. In this section we briefly discuss some of the essential characteristics of raster pipeline subarray machines for DA work.

Expandability: a machine based on a pipeline of homogeneous stages is inherently modular. Adding stages is straightforward and practical. In addition, the loose coupling of the major system components--disk, image-memory, controller, pipeline--permits independent upgrading of any component.

Cost/Performance Range: both cost and performance are proportional to pipeline length and image-buffer size. A low-end system will have only a single short pipe and small buffer. A high-end system will have several long pipes, a large buffer, and a dedicated disk.

Direct Accommodation of Large Problems: the only limitation on the grid-size is the length of the line-buffers in each pipeline stage. Assuming a very large $M \times N$ image can be generated, it is possible for the pipeline to process this image directly, for any M, if each line-buffer length exceeds N.

Application Range: clearly a variety of DRC and routing tasks can be performed. Any problem that can be represented on a cellular grid and that is characterized by local functional dependencies among cells is a candidate application.

The primary weakness of these architectures is the restriction on global and conditional data-manipulation imposed by the pipeline structure. A P-stage pipeline of neighborhood-processing stages can, in one pass, move any pixel by P cells in any direction. The cost to move just one cell this distance is same as the cost to move all cells this distance: one pass through the pipe. If one cell in an image requires knowledge about another cell a distance $D \gg P$ cells away, the cost is $\lceil D/P \rceil \gg 1$ passes through the pipe. It is similarly difficult for a state change in one cell to influence globally all subsequent processing steps in the pipeline. If this state change

occurs in the middle of a full pipeline with output already streaming from the final stage it is impossible backup the raster, reprogram the stages, and rerun the image. However, neither of these problems is serious enough to warrant abandoning pipeline-based machines. With hardware/firmware support the pipeline controller can perform much of this data-movement and examination, e.g., WAVE-EXPAND termination.

It is useful here to compare raster pipeline subarray machines with full-arrays on some of these points. The pipeline structure easily accommodates additional processing stages. Arrays are generally not designed to accommodate additional processors. Large arrays with thousands of processors are usually restricted to simple but fast bit-sequential processors; algorithms may be lengthy because of this bit-level processing but overall speed can be significant just because of the enormous number of processors. Pipelines with 10-100 processors can afford more complex stages; the goal is to incorporate as much processing power in each stage as possible (e.g., the ability to perform one WAVE-EXPAND step for a general weighted maze-runner in one stage) to minimize the number of passes through the pipeline. Both arrays with large memories at each node and pipelines with long line buffers can deal directly with large problems. However, arrays are limited by the total storage available across all nodes, whereas pipelines are limited by the length of the line buffers. Consider, for example, that both a 64 × 64 array with 4K-bits per node and a 64-stage 32-bit wide pipeline with 4K-word line buffers require 16M-bits of storage. A 704 × 704 × 32-bit grid can be folded directly onto the array effectively filling up all storage; any larger image must paged in and out of this storage. A 4K × 4K × 32-bit image can be directly streamed through the pipeline. Both arrays and pipelines benefit uniformly from improvements in device density and speed: incorporating more stages (processors) onto a chip allows the construction of larger pipelines (arrays). There will inevitably be some point at which chip count for a large pipeline system matches that of a large array. In this situation the particular structure of the

problems at hand will determine the choice of hardware.

6. Conclusions

The class of raster pipeline subarray architectures encompasses a range of machines and applications. We have shown that an appropriate raster pipeline subarray organization is able to support several grid-based DA applications, the principal strength of this organization being the wide cost/performance range achievable with a modular pipeline structure. Results from experimental DA algorithms running on cytocomputers are encouraging. We are currently interfacing a cytocomputer with a new host environment--a VAX[*] 11/780 running UNIX[**]-- and developing new, more comprehensive DRC and routing packages to run on this hardware. The intent is to develop potential applications by studying prototype algorithms running on the machine in a real environment. Moreover, it will enable us to identify more of the performance bottlenecks to realistic applications discussed previously, and hence to refine the optimized machine architecture for DA proposed here.

Acknowledgments

The authors are grateful to the Environmental Research Institute of Michigan for early access to several cytocomputers, and wish to thank Robert Lougheed in particular for his support.

[*] *VAX* is a trademark of the Digital Equipment Corporation.
[**] *UNIX* is a trademark of Bell Laboratories.

References

[1] M. Abramovici, Y. H. Levendel and P. R. Menon, "A logic simulation machine," *Proc. 19th Design Automation Conf.*, pp. 65-73, June 1982.

[2] H. G. Adshead, "Towards VLSI complexity: The DA algorithm scaling problem: Can special DA hardware help?" *Proc. 19th Design Automation Conf.*, pp. 339-344, June 1982.

[3] H. G. Adshead, "Employing a distributed array processor in a dedicated gate-array layout system," *Proc. ICCC*, pp. 411-414, Oct. 1982.

[4] R. L. Barto, S. A. Szygenda and E. W. Thompson, "Architecture for a hardware simulator," *Proc. ICCC-80*, pp. 891-893, 1980.

[5] T. Blank, M. Stefik and W. van Cleemput, "A parallel bit map processor architecture for DA algorithms," *Proc. 18th Design Automation Conf.*, pp. 837-845, June/July 1981.

[6] T. Blank, "A bit map architecture and algorithms for design automation," Ph.D. Thesis, Dept. of EE, Stanford Univ., Stanford CA., Sept. 1982.

[7] M. A. Breuer and K. Shamsa, "A hardware router," *Jour. of Digital Systems*, vol. IV, issue 4, pp. 393-408, 1981.

[8] C. R. Carroll, "A smart memory array processor for two layer path finding," *Proc. 2nd Caltech Conf. on Very Large Scale Integration*, Jan. 1981.

[9] E. Damm, H. Gethoeffer and K. Kaiser, "Hardware support for automatic routing," *Proc. 19th Design Automation Conf.*, pp. 219-223, June 1982.

[10] M. M. Denneau, "The Yorktown simulation engine," *Proc. 19th Design Automation Conf.*, pp. 55-59, June 1982.

[11] S. J. Hong, R. Nair and E. Shapiro, "A physical design machine," in *VLSI 81*, J. P. Gray, Ed., London: Academic Press, pp. 346-365, 1981.

[12] A. Iosupovicz, "Design of an iterative array maze router," *Proc. ICCC*, pp. 908-911, 1980.

[13] T. N. Mudge, R. M. Lougheed and W. B. Teel, "Design rule checking for VLSI circuits using a cellular computer," *Abstracts of the 1981 ACM Computer Science Conf.*, St. Louis, pp. 29, Feb. 1981.

[14] T. N. Mudge, R. M. Lougheed and W. B. Teel, "Cellular image processing techniques for checking VLSI circuit layouts," *Proc. of the 1981 Conf. on Information Sciences and Systems*, The Johns Hopkins University, pp. 315-320, March 1981.

[15] T. N. Mudge, R. A. Rutenbar, R. M. Lougheed and D. E. Atkins, "Cellular image processing techniques for VLSI circuit layout validation and routing," *Proc. 19th Design Automation Conf.*, pp. 537-543, June 1982.

[16] R. Nair, S. J. Hong, S. Liles and R. Villani, "Global wiring on a wire routing machine," *Proc. 19th Design Automation Conf.*, pp. 224-231, June 1982.

[17] G. F. Pfister, "The Yorktown simulation engine: Introduction," *Proc. 19th Design Automation Conf.*, pp. 51-54, June 1982.

[18] L. Seiler, "A hardware assisted design rule check architecture," *Proc. 19th Design Automation Conf.*, pp. 232-238, June 1982.

[19] R. M. Lougheed, D. L. McCubbrey and S. R. Sternberg, "Cytocomputers: Architectures for parallel image processing," *Proc. IEEE Workshop on Picture Data Description and Management*, Aug. 1980.

[20] R. M. Lougheed, D. L. McCubbrey, "The cytocomputer: A practical pipelined image processor," *Proc. 7th Annual International Symp. on Computer Architecture*, pp. 271-277, May 1980.

[21] S. R. Sternberg, "Language and architecture for parallel image processing," in *Pattern Recognition in Practice*, E. S. Gelsema and L. N. Kanal, Eds., Amsterdam: North Holland Publishing Co., 1980.

[22] C. Mead and L. Conway, *Introduction to VLSI Systems*, Reading: Addison-Wesley, 1980.

[23] P. E. Danielsson and S. Levialdi, "Computer architectures for pictorial information systems," *Computer*, vol. 14, no. 11, pp. 53-67, Nov. 1981.

[24] *Languages and Architectures for Image Processing*, M. Duff and S. Levialdi, Eds., London: Academic Press, 1981.

[25] M. Kidode, "Image processing machines in Japan," *Computer*, vol. 16, no. 1, pp. 68-80, Jan. 1983.

[26] K. Preston, "Cellular logic computers for pattern recognition," *Computer*, vol. 16, no. 1, pp. 36-47, Jan. 1983.

[27] K. Preston, M. J. B Duff, S. Levialdi, P. Norgren and J. Toriwaki, "Basics of cellular logic with some applications in medical image processing," *Proc. of the IEEE*, vol. 67, no. 5, pp. 826-856, May 1979.

[28] *Multicomputers and Image Processing: Algorithms and Programs*, K. Preston and L. Uhr, Eds., New York: Academic Press, 1982.

[29] D. Antonsson et al., "PICAP - A system approach to image processing," *IEEE Trans. Computers*, vol. C-31, no. 10, pp. 997-1000, Oct. 1982.

[30] H. J. Siegel, et al., "PASM: A partitionable SIMD/MIMD system for image processing and pattern recognition," *IEEE Trans. Computers*, vol. C-30, pp. 934-947, Dec. 1981.

[31] M. J. B. Duff, "Review of the CLIP image processing system," *Proc. National Computer Conf.*, pp. 1055-1060, 1978.

[32] J. K. Iliffe, *Advanced Computer Design*, London: Prentice Hall, Chap. 12, 1982.

[33] K. E. Batcher, "Architecture of a massively parallel processor," *Proc. 7th Annual Symp. on Computer Architecture*, pp. 168-174, 1980.

[34] M. Aoki et al., "An LSI adaptive array processor," *Proc. ISSCC*, pp. 122-123, Feb. 1982.

[35] K. Preston and P. E. Norgren, "Interactive image processor speeds pattern recognition," *Electronics*, vol. 45, p. 89, 1972.

[36] J. M. Herron, J. Farley, K. Preston and H. Sellner, "A general-purpose high-speed logical transform image processor," *IEEE Trans. Computer*, vol. C-31, no. 8, pp. 795-800, Aug. 1982.

[37] C. M. Baker, "Artwork analysis tools for VLSI circuits," M.S. Thesis, MIT, Cambridge, MA, 1980.

[38] J. Serra, *Mathematical Morphology and Image Processing*, London: Academic Press, 1981.

[39] G. Matheron, *Random Sets and Integral Geometry*, New York: John Wiley & Sons, 1975.

[40] H. S. Baird, "Fast algorithms for LSI artwork analysis," *Jour. of Design Automation and Fault Tolerant Computing*, vol. 2, no. 2, May 1978, pp. 179-209.

[41] P. Losleben and K. Thompson, "Topological analysis for VLSI circuits," *Proc. 16th Design Automation Conf.*, pp. 461-473, June 1979.

[42] R. Eustace and A. Mukhopadhyay, "A deterministic finite automaton approach to design rule checking for VLSI," *Proc. 19th Design Automaton Conf.*, pp. 712-717, June 1982.

[43] E. H. Frank and R. F. Sproull, "Testing and debugging custom integrated circuits," *Computing Surveys*, vol. 13, no. 4, pp. 425-452, Dec. 1981.

[44] M. Marek-Sadowska and W. Maly, " A hierarchical layout description for artwork analysis of VLSI IC," *Proc. ICCC-82*, pp. 419-422, Oct. 1982.

[45] J. Wilmore, "The use of bit maps in designing efficient data bases for integrated circuit layout systems," *Jour. of Digital Systems*, vol. IV, issue 1, pp. 71-95, 1980.

[46] R. F. Lyon, "Simplified Design Rules for VLSI Layouts," *Lambda*, vol. II, no. 1, pp. 54-59, 1st Quarter, 1981.

[47] T. W. Griswold, "Portable design rules for bulk CMOS," *VLSI Design*, vol. III, no. 5, pp. 62-67, Sept./Oct. 1982.

[48] R. A. Rutenbar, "A cellular framework and techniques for physical design automation problems," CRL Tech. Report CRL-TR-6-83, University of Michigan, Jan. 83.

[49] P. Losleben, "Computer aided design for VLSI," in *Very Large Scale Intergration VLSI: Fundamentals and Applications*, D. F. Barbe, Ed., Springer-Verlag, 1980.

[50] J. L. Bentley, D. Haken and R. W. Hon, "Fast geometric algorithms for VLSI tasks," *Proc. COMPCON-80*, pp. 88-92, 1980.

[51] M. H. Arnold and J. K. Ouserhout, "Lyra: A new approach to geometric layout rule checking," *Proc. 19th Design Automation Conference*, pp.530-536, June 1982.

[52] S. E. Bello, J. L. Hoffman, R. I. McMillan and J. A. Ludwig, "VLSI hierarchical design verification," *Proc. ICCC-82*, pp. 530-533, Oct. 1982.

[53] M. A. Breuer, A. D. Friedman and A. Iosupovicz, "A survey of the state of the art in design automation," *Computer*, vol. 14, no. 10, pp. 58-75, Oct. 1981.

[54] J. Soukup, "Circuit layout," *Proc. of the IEEE*, vol. 69, no. 10, pp. 1281-1304, Oct. 1981.

[55] C. Y. Lee, "An Algorithm for Path Connections and Its Applications," *IRE Trans. on Electronic Computers*, vol. EC-10, September 1961, pp. 346-358.

[56] F. Rubin, "The Lee path connection algorithm," *IEEE Trans. Computer*, vol. C-23, pp. 907-914, Sept. 1974.

[57] J. H. Hoel, "Some variations of Lee's algorithm," *IEEE Trans. Comput.*, vol. C-25, pp. 19-24, Jan. 1976.

[58] J. Soukup, "Fast maze router," *Proc. 15th Design Automation Conf.*, pp. 100-101, June 1978.

[59] S. Akers, "Routing," in *Design Automation of Digital Systems*, vol. 1, M. Breuer, Ed., Englewood Cliffs, NJ: Prentice Hall, Chapter 6, 1972.

[60] M. A. Fischetti, "VLSI/LSI components," *Spectrum*, vol. 20, no. 1, pp. 43-47, Jan. 1983.

[61] S. R. Parris and J. A. Nelson, "Practical packaging considerations in VLSI packaging," *VLSI Design*, vol. III, no. 6, pp. 44-49, Nov./Dec. 1982.

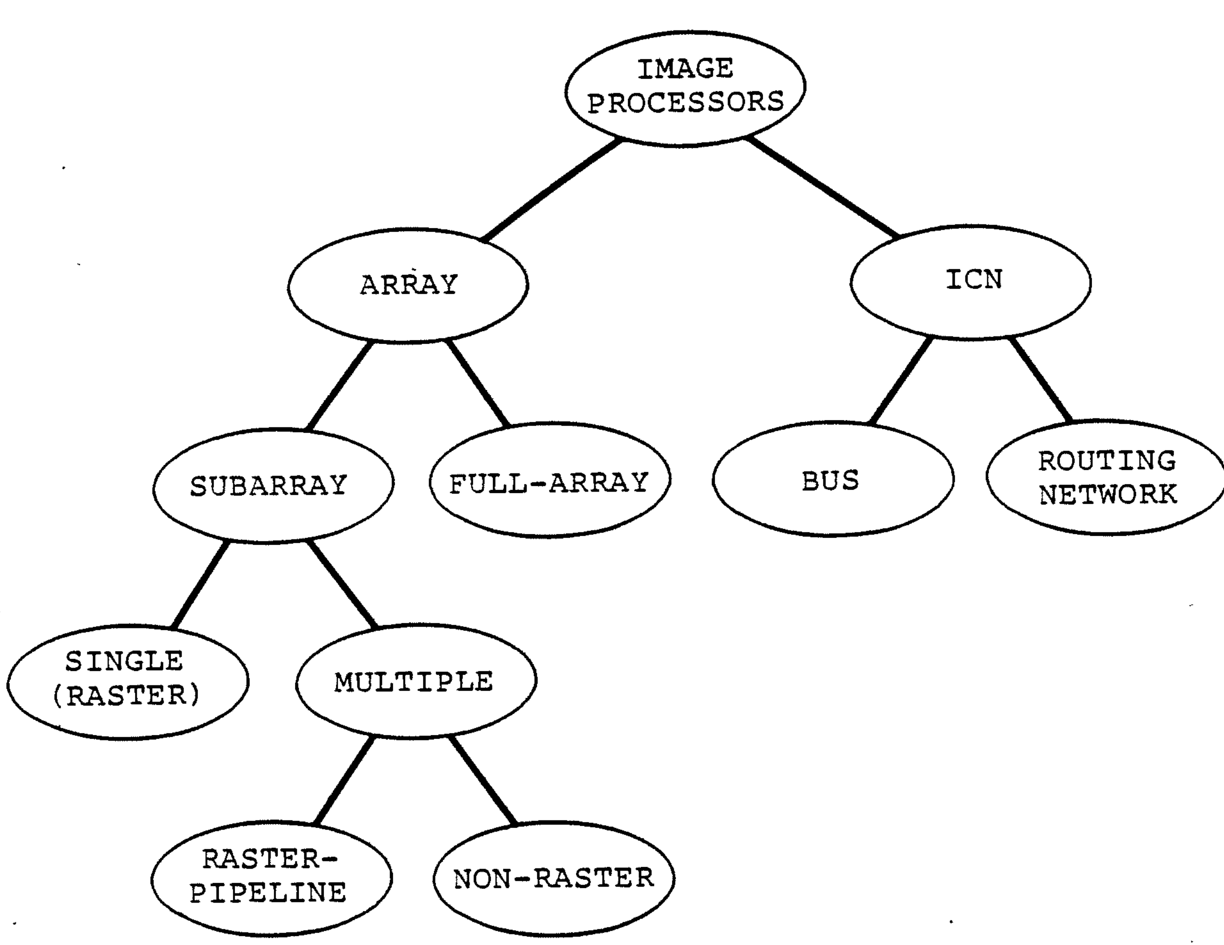


Fig. 1. Taxonomy of image processors.

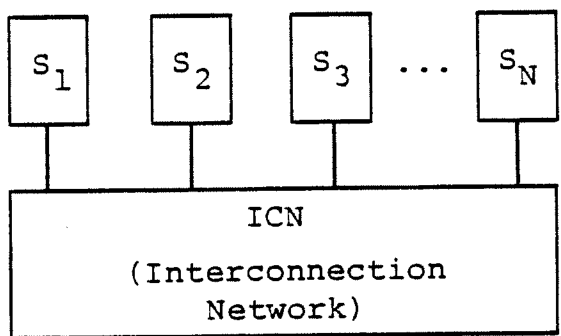


Fig. 2. ICN (Interconnection Network) architecture.
S_i is a processor/memory subsystem.

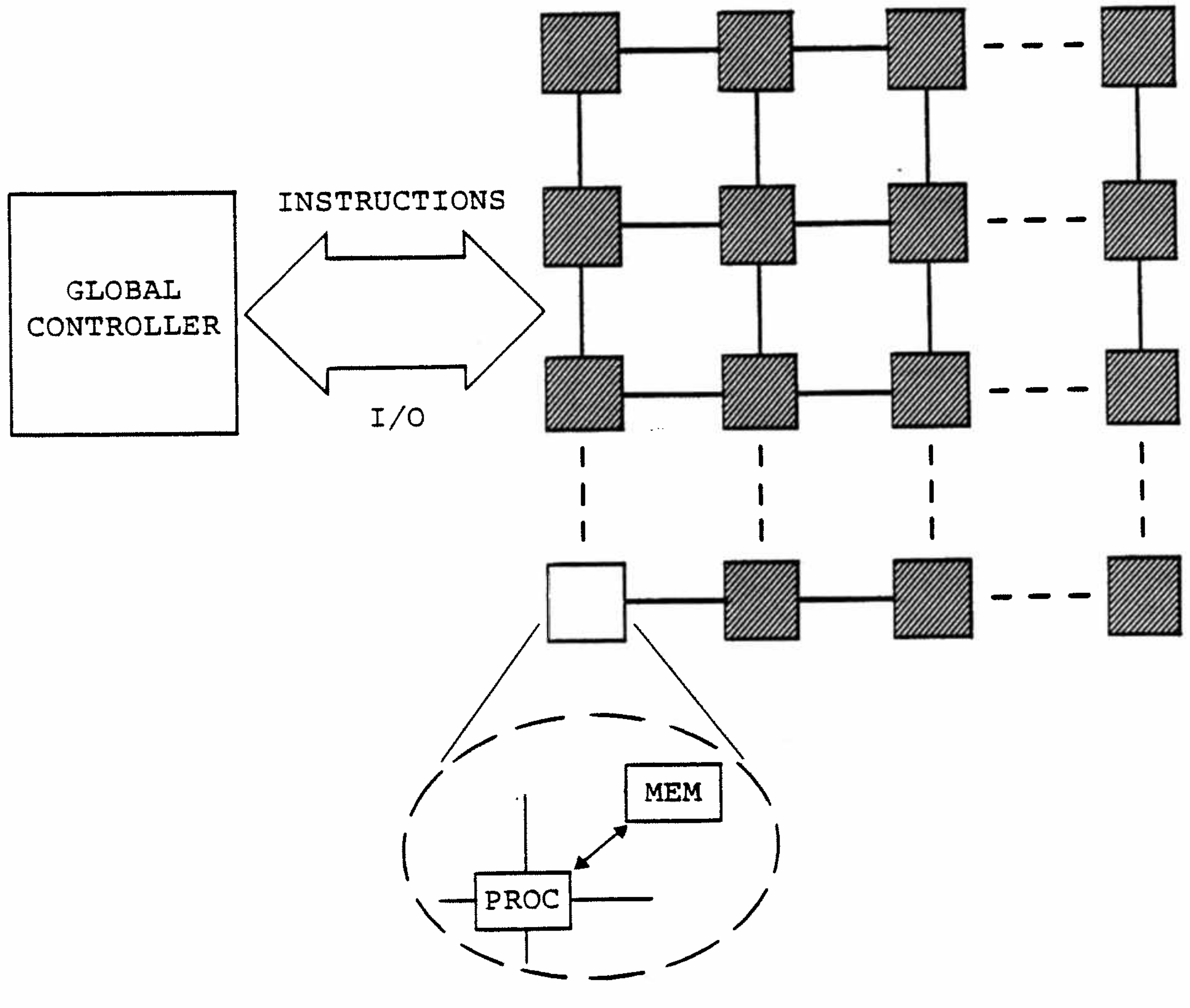


Fig. 3. Array architecture.

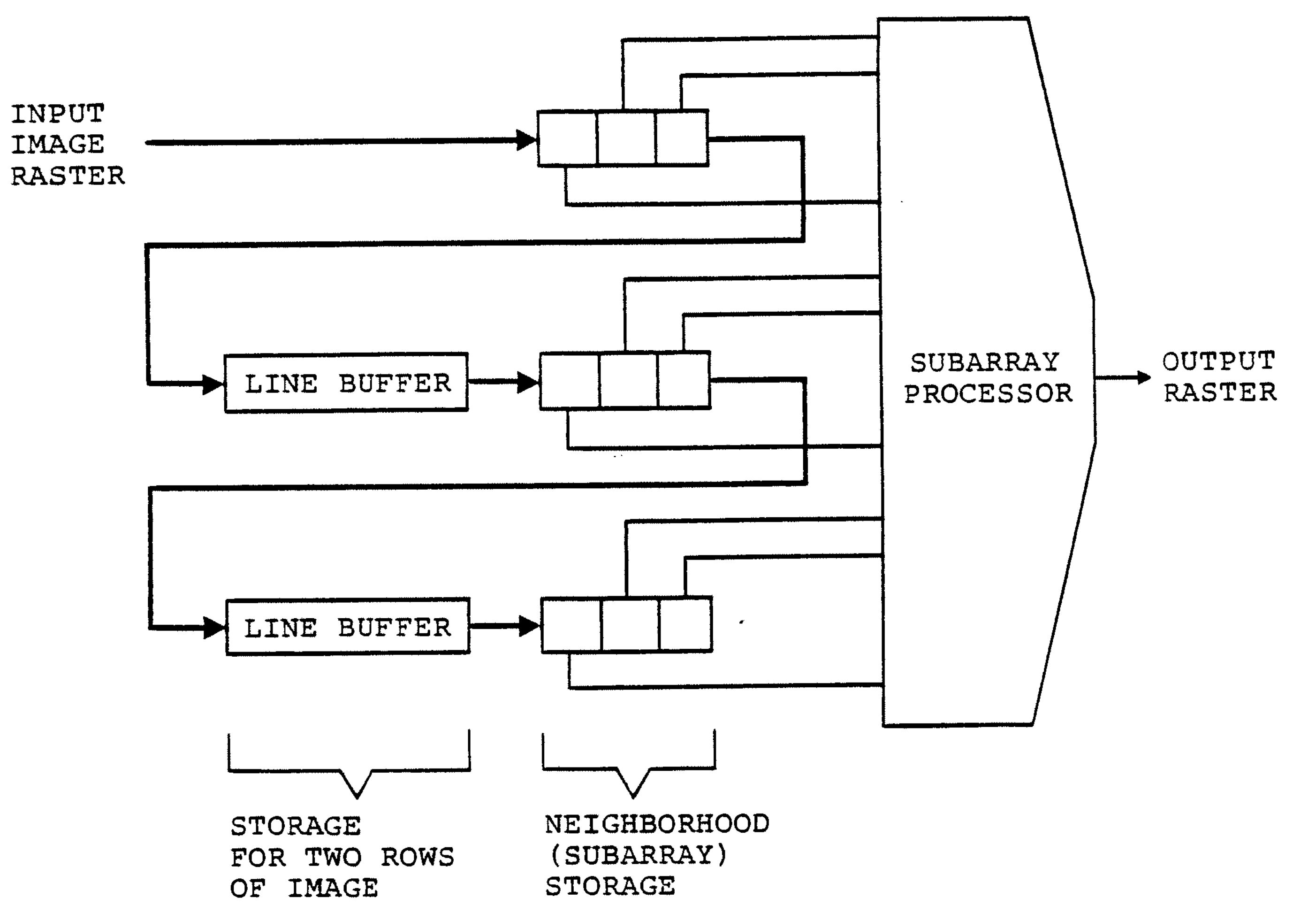


Fig. 4. Raster single subarray architecture.

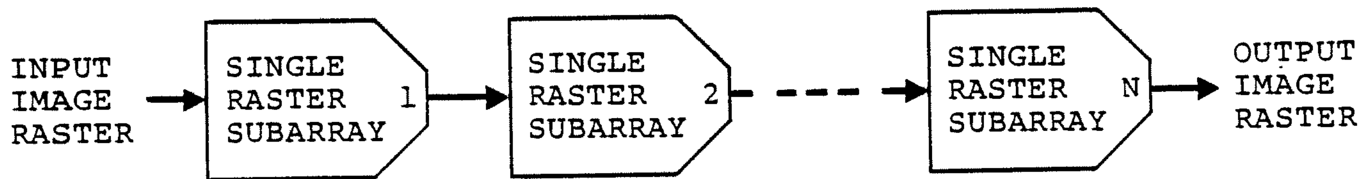


Fig. 5. Raster pipeline subarray architecture.

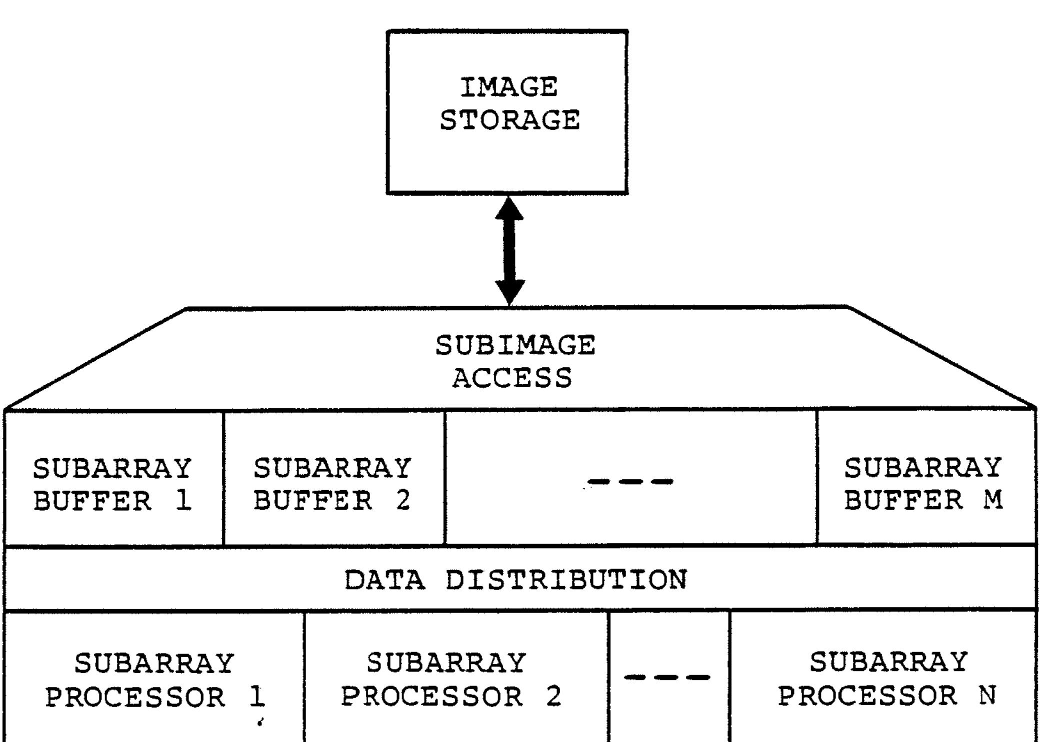


Fig. 6. Non-raster multiple subarray architecture.

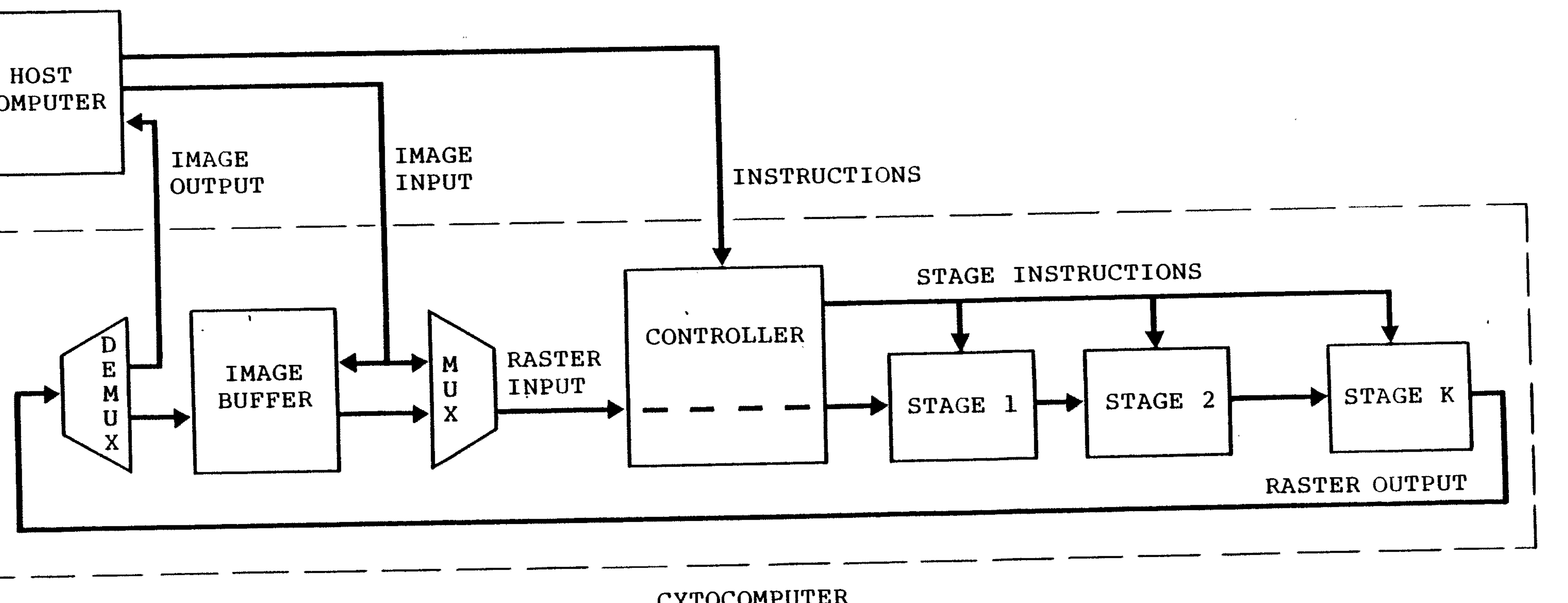


Fig. 7. Cytocomputer system architecture.

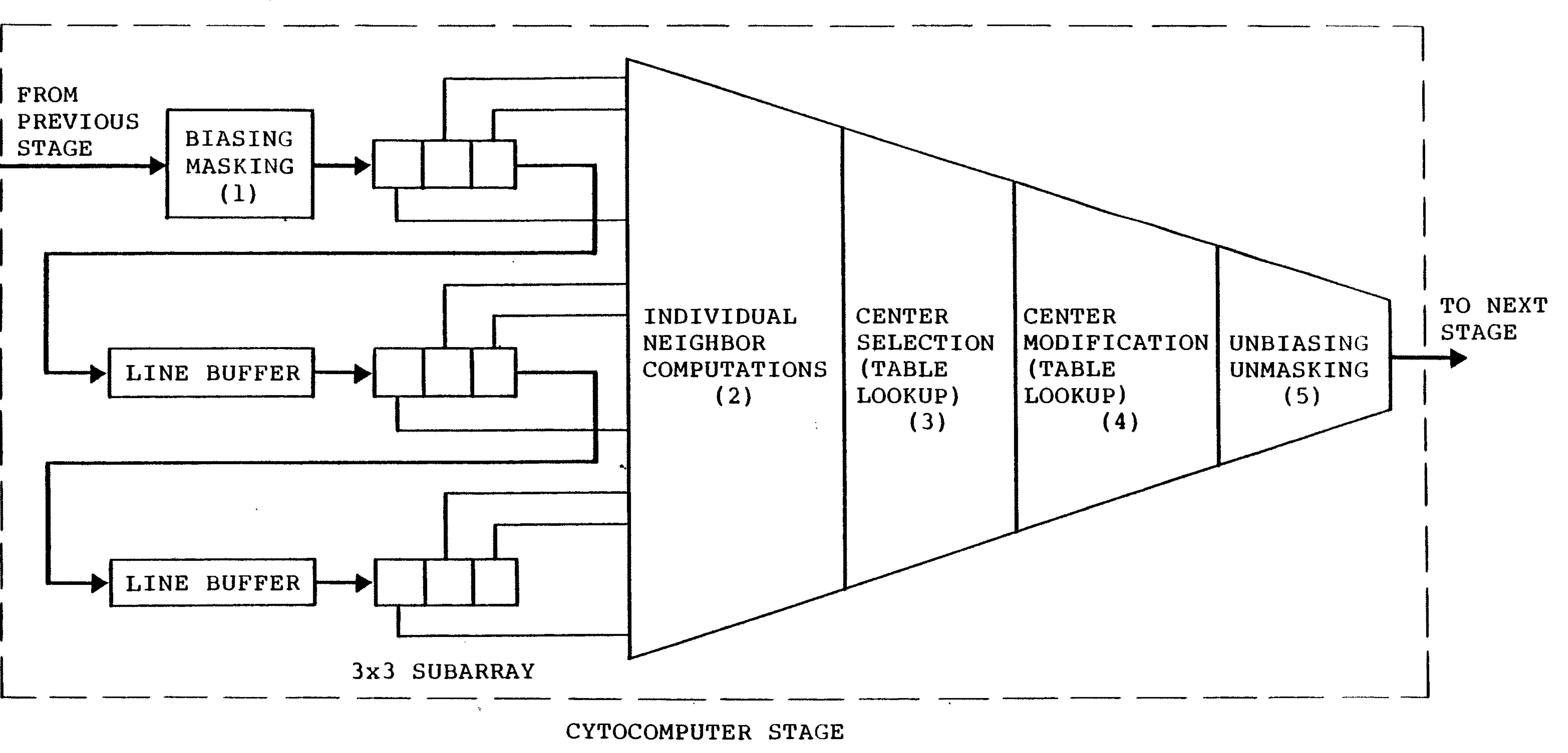


Fig. 8. Cytocomputer stage architecture. See text for description of operations (1)-(5).

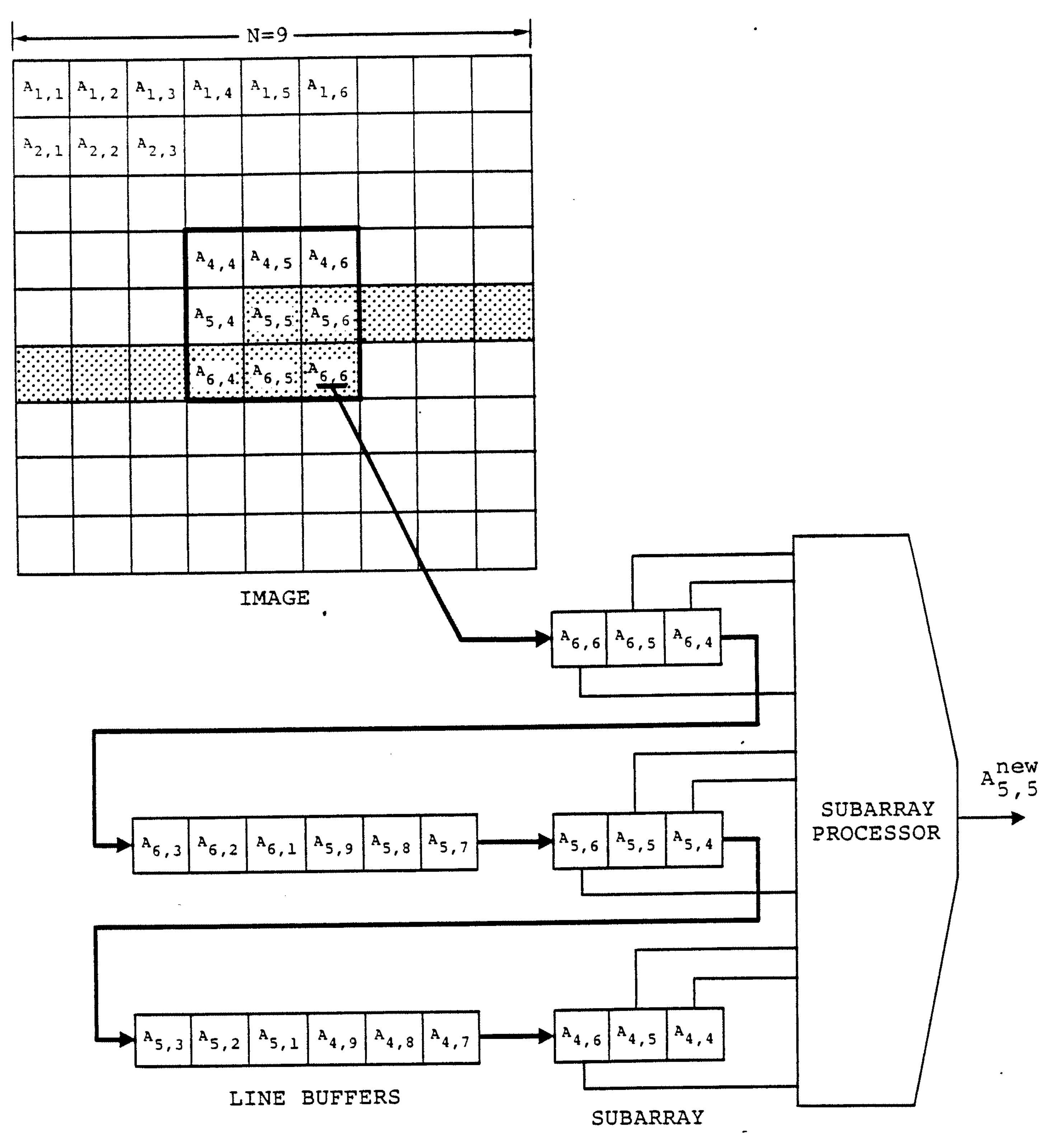


Fig. 9. Illustration of a single neighborhood transformation.

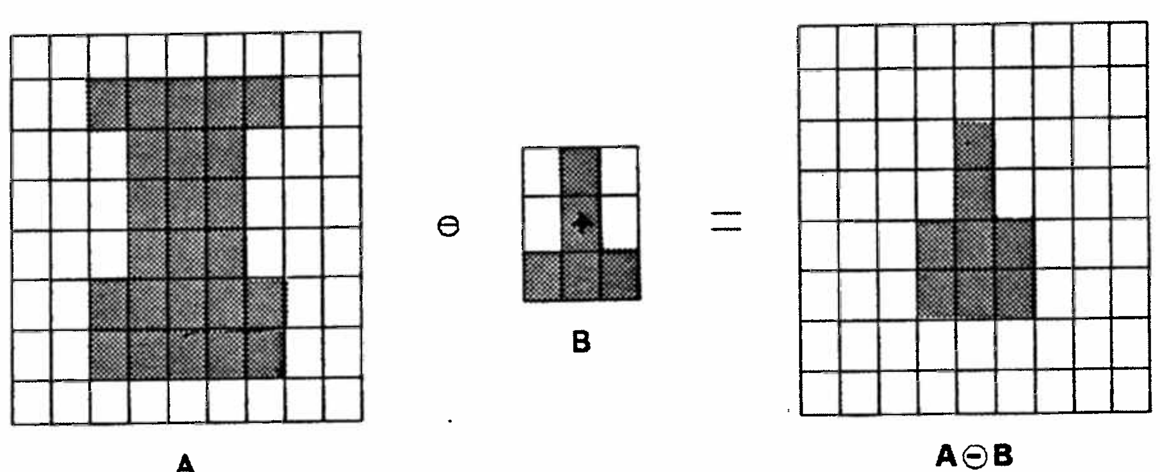


Fig. 10. Illustration of erosion $A \ominus B$.

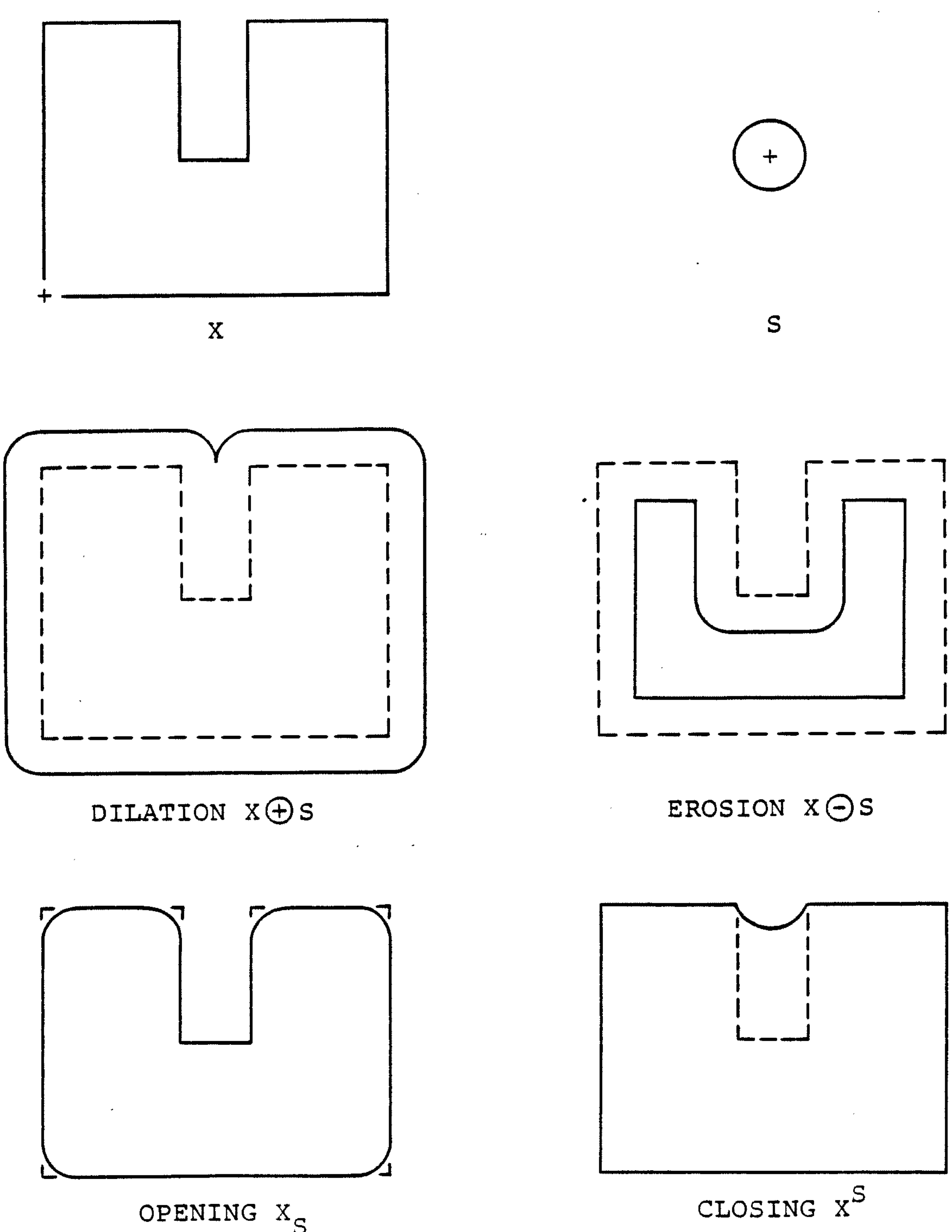


Fig. 11. The dilation, erosion, opening, and closing of figure X by figure S.

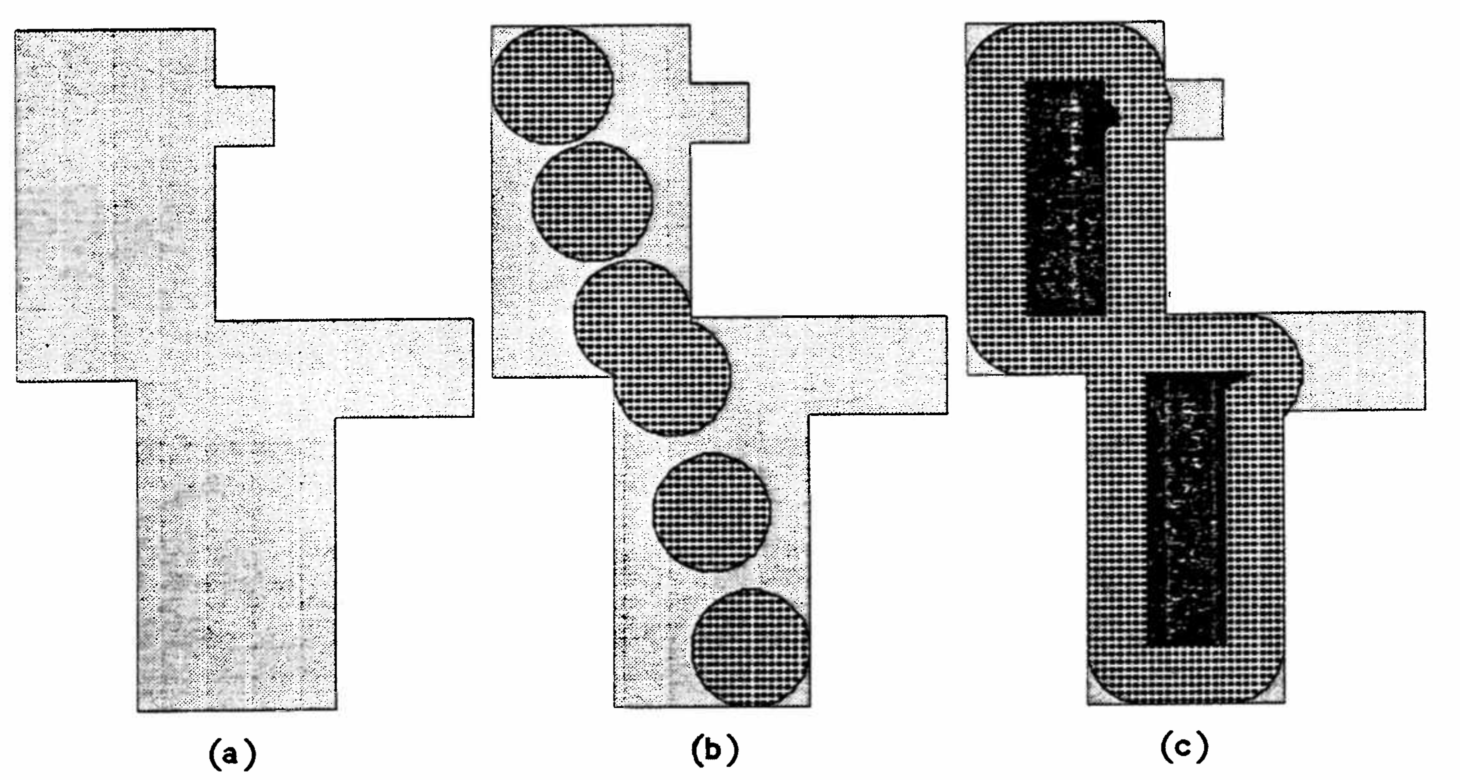


Fig. 12. Geometric basis for Algorithm 1 width-W check.
(a) Mask to be checked.
(b) Sliding diameter-W disk.
(c) Regions covered by disk (dotted) and traced by center (black).

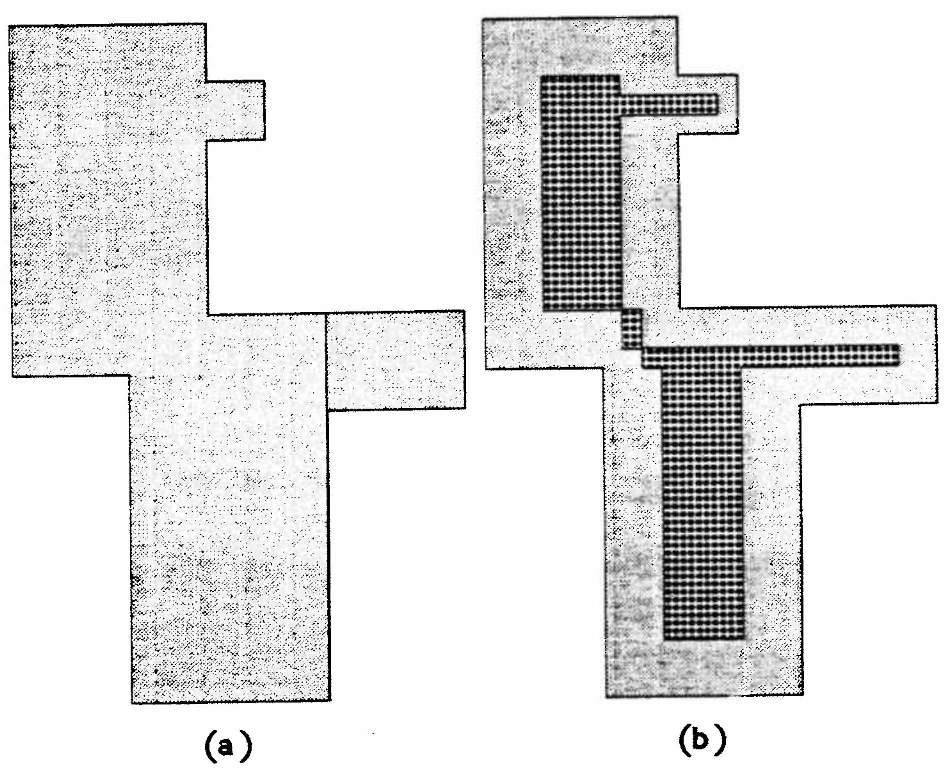
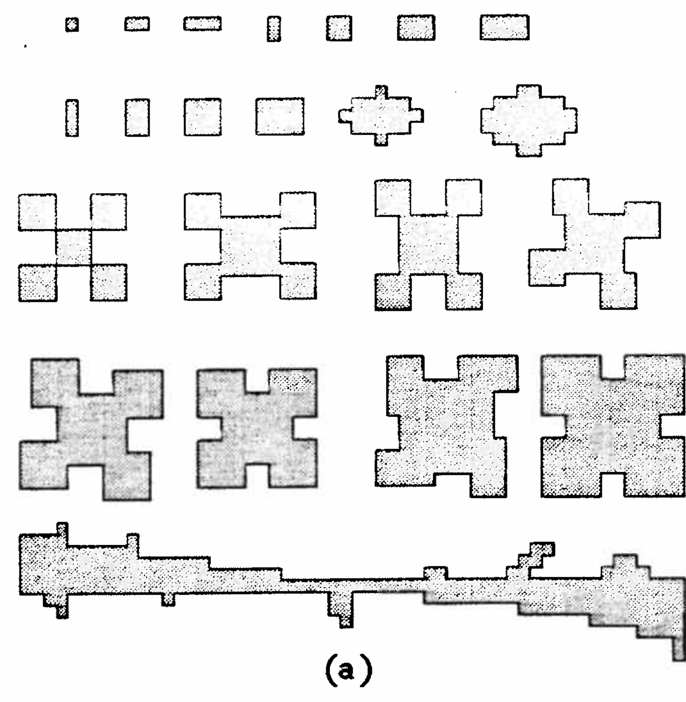


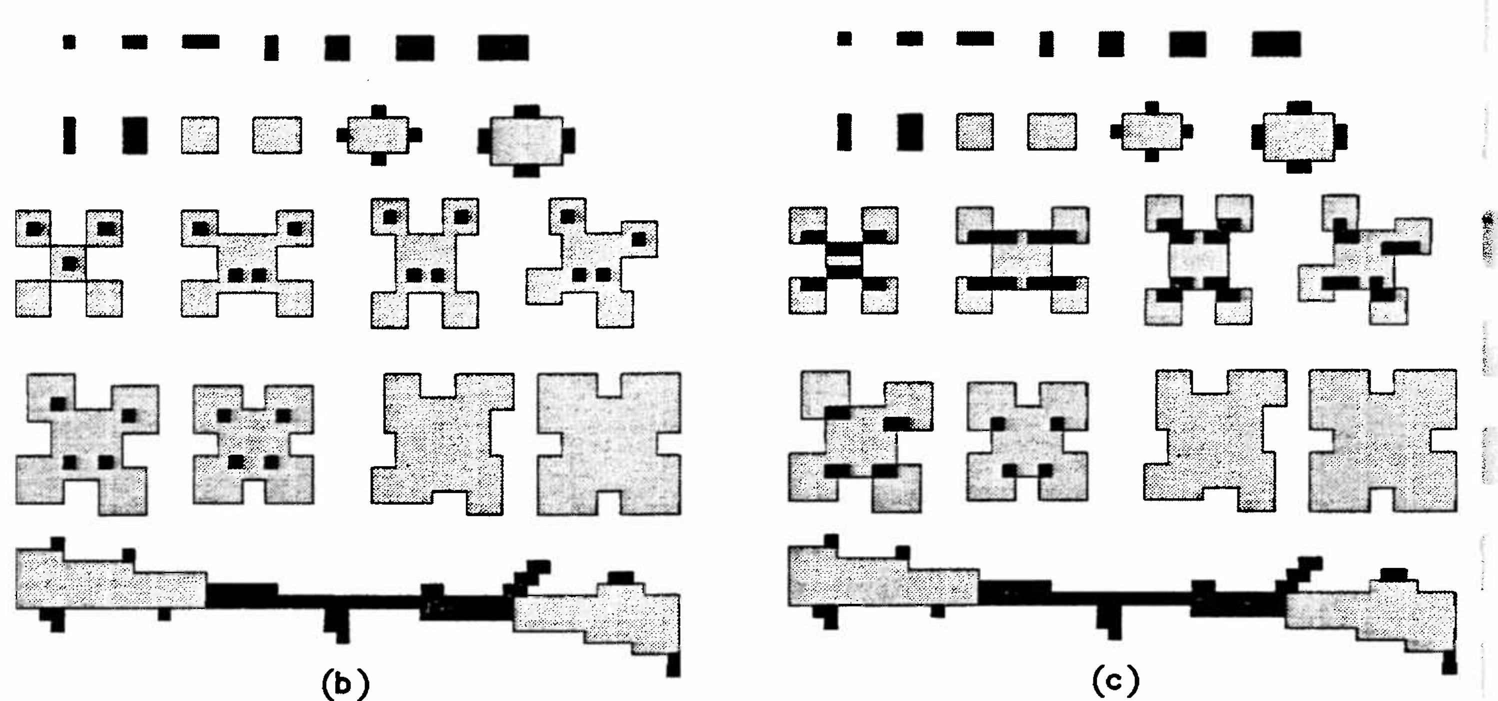
Fig. 13. Geometric basis for Algorithm 2 width-W check.

(a) Original mask to be checked.

(b) Thinned result (dotted).



(a)



(b) (c)

Fig. 14. Results of width-3λ checks on 64λ × 64λ mask.

(a) Mask to be checked.

(b) Algorithm 1 labels errors in black. Regions too small are black; narrow necks are tagged by a black square north of violation.

(c) Algorithm 2 labels errors in black. Regions too small are black; narrow necks are tagged by a black line across violation.

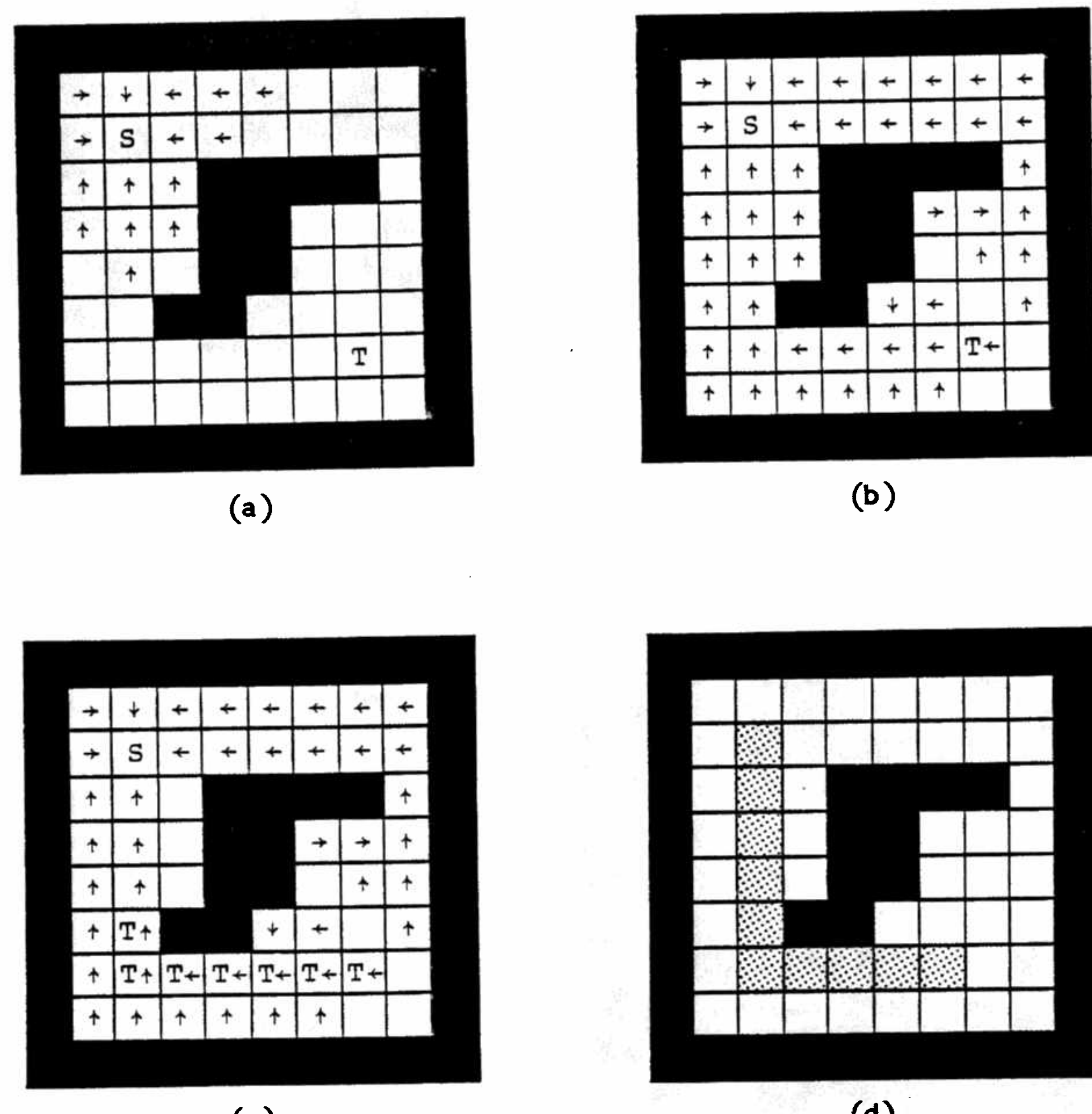


Fig. 15. Routing process.

(a) WAVE-EXPAND in progress.

(b) Target labeled; WAVE-EXPAND terminated.

(c) BACK-TRACE in progress.

(d) Path complete after CLEAN-UP.

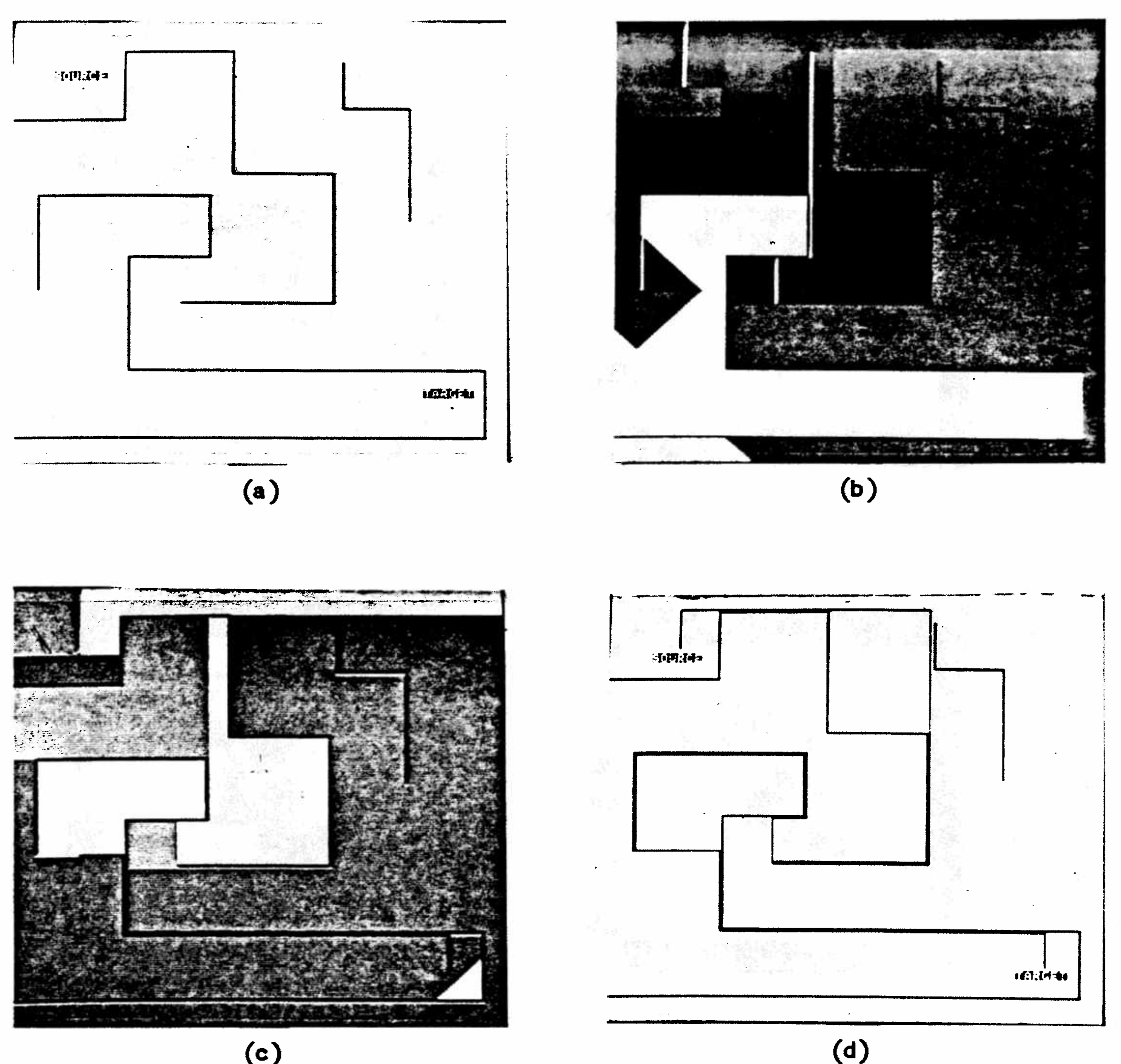


Fig. 16. Result of routing 700-cell path on 200 × 200 grid.
(a) Grid with Source (top left) and Target (lower right).
(b) WAVE-EXPAND in progress (wavefronts are shaded).
(c) BACK-TRACE in progress (path is shaded).
(d) Final path after CLEAN-UP.

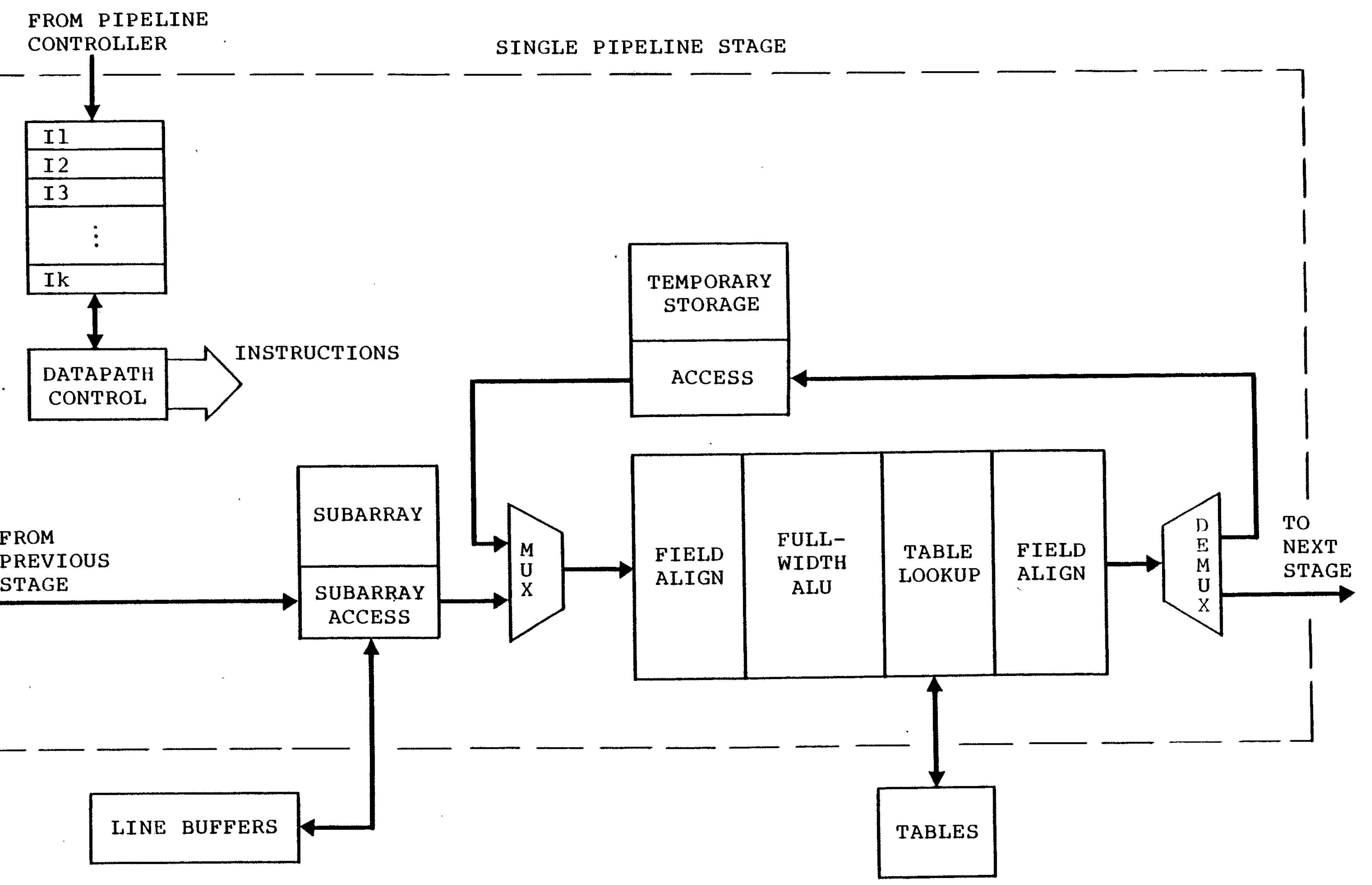


Fig. 17. Stage architecture for raster pipeline subarray DA machine.

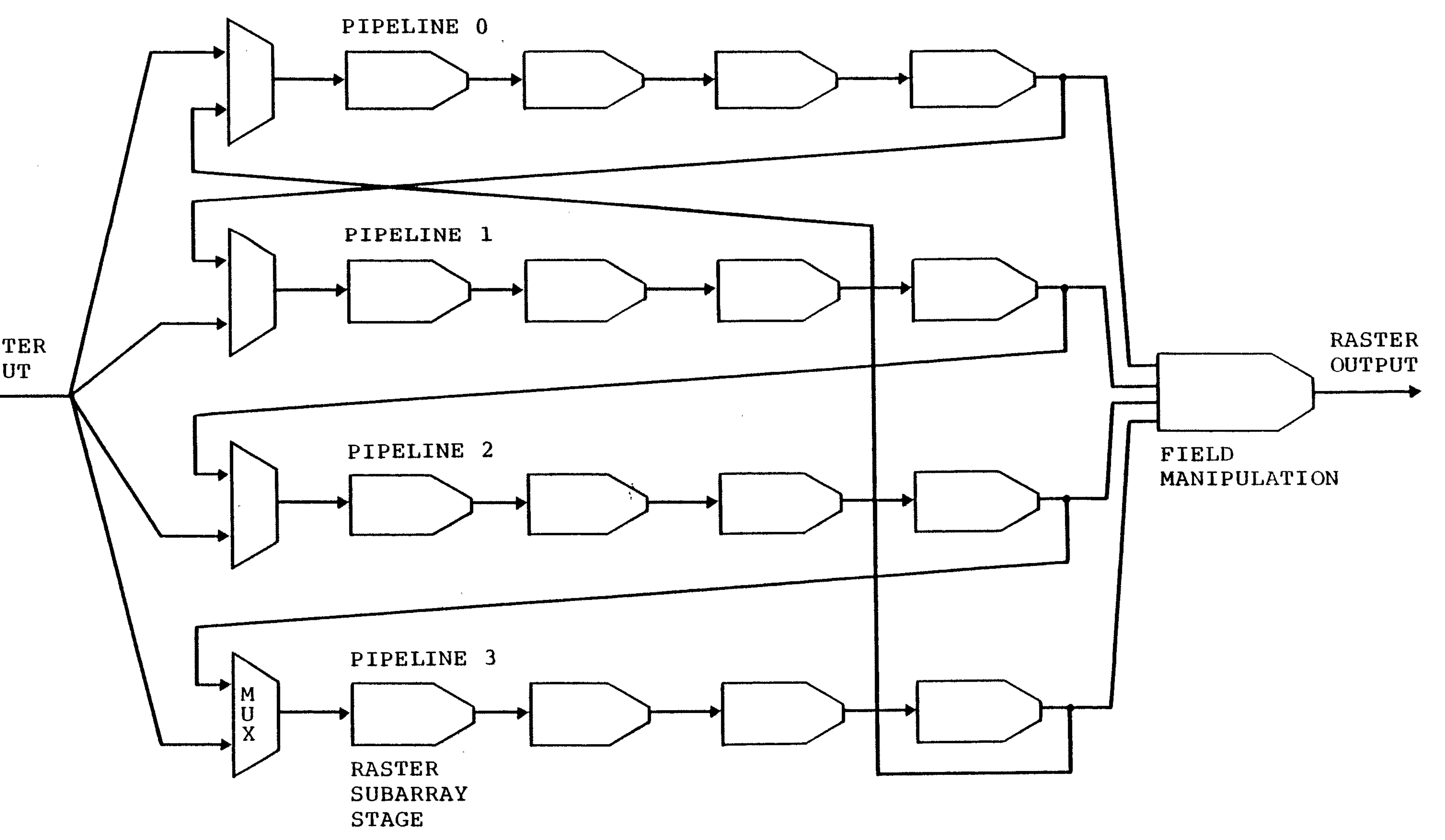


Fig. 18. Pipeline architecture for raster pipeline subarray DA machine.