TR-82-001

VLSI MATRIX MANIPULATORS FOR FEATURE EXTRACTION AND PATTERN CLASSIFICATION

Kai Hwang, Senior Member IEEE,

and S. P. Su, Student Member, IEEE

ABSTRACT :

In statistical methods for feature extraction and pattern classification, large-scale matrix computations are often performed over large image data bases. Based on Hwang/Cheng partitioned matrix algorithms [8], we develop a class of matrix manipulation networks with modular VLSI arithmetic chips. These special-purpose VLSI matrix manipulators are effective in extracting features from raw images and in classifying patterns over a large feature space. Such VLSI pattern analyzers are highly demanded in real-time pictorial information processing and in artificial intelligence applications.

Special matrix/vector computations are analyzed for possible VLSI solution of pattern recognition problems. We emphasize computational complexity and achievalbe speedups by VLSI hardware approaches. Basic VLSI arithmetic modules, partitioned matrix algorithms, and functional structures of the VLSI feature extractor and pattern classifier are presented. Performance analysis and hardware design tradeoffs are also provided.

INDEX TERMS :

Very Large Scale Integration (VLSI), pattern recognition, computer arithmetic, feature extraction, matrix computations, pictorial information processing, real-time applications, computer architecture.

This research was supported in part by U.S. National Science Foundation under grant ECS-80-16580 and in part by Academia Sinica, Taiwan, China.

Kai Hwang and S. P. Su are with the School of Electrical Engineering, Purdue University, Lafayette, Indiana 47907. A preliminary version of this paper, entitled "A Partitioned Matrix Approach to VLSI Pattern Classification", was presented in the IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management, Hot Springs, Virginia, Nov. 11-13, 1981.

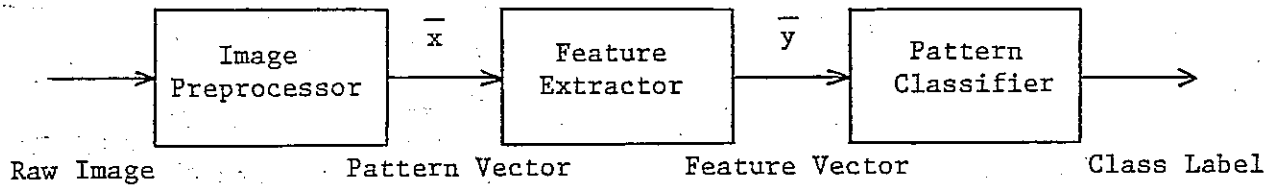
1. INTRODUCTION

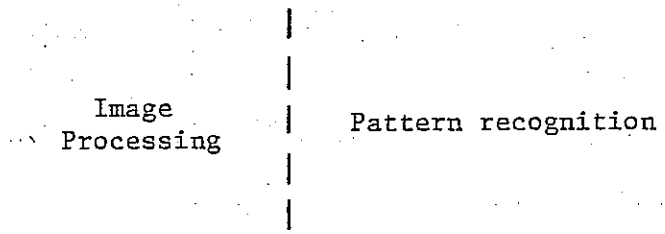
Computationally speaking, a statistical pattern recognition system consists of three phases as illustrated in Fig. 1. The preprocessing includes such image operations like enhancement, restoration, segmentation and etc. [13]. After preprocessing, the picture will be smoothed, segmentized, and represented by an n-dimensional input vector $\bar{x} = (x_1, x_2, \ldots, x_n)^T$. The middle stage is for feature extraction transforming the vector $\bar{x}$ to a feature vector $\bar{y} = (y_1, y_2, \ldots, y_m)^T$ in an m-dimensional feature space ; where $n >> m$. After the features are extracted, an intelligent pattern classifier is needed to determine the membership of the pattern being examined among known classes of petterns. Either statistical methods [6] or syntactic methods [5] can be used to classify the patterns.

Raw

Statistical decision-theoretic approach is used to partition the feature space into mutually exclusive regions (subspaces), where each region corresponds to one known pattern class. Large scale matrix computations are often involved in the recognition porcess. Most frequently performed operations are matrix multiplication, inversion, L-U decomposition, system triangularization, and back subsitution, etc. [3,6]. When dealing with large input space, the size of the matrices to be manipulated becomes too large to be handled efficiently by conventional Single-Instruction-Single-Data(SISD) serial computers through software approaches. Single-Instruction-Multiple-Data(SIMD) array processors and pipelined vector processors, like Illiac IV, BSP, ASC, STAR, CYBER-205, Cray-1, etc, have been suggested to handle large-scale matrix/vector computations [9]. However, these supercompouers may not be necessarily cost-effective in performing the simple but repetitive computations required in image processing and pattern recognition.

Recent advances in VLSI microelectronic technology has triggered the thought of implementing some pattern-analysis matrix algorithms directly in hardware. Such VLSI pattern recognizers should be very useful in real-time, on-line, pictorial





$$\bar{x} = (x_1, x_2, \ldots, x_n)^T$$

$$\bar{y} = (y_1, y_2, \ldots, y_m)^T$$

where $n >> m$

Fig.1. Three phases of a pattern analysis system

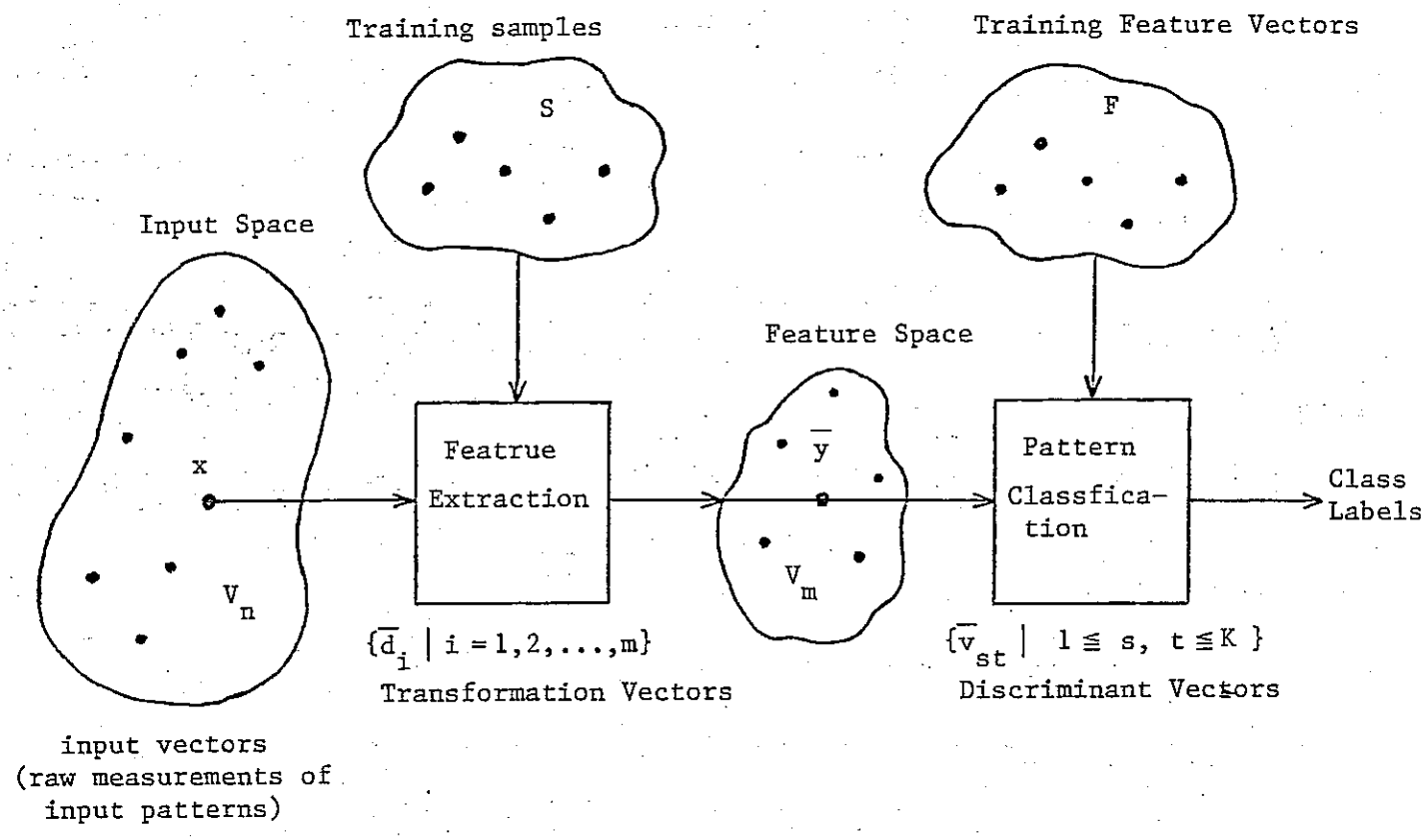
information processing or for artificial intelligence applications. VLSI systolic arrays have been suggested in [10] for matrix manipulations. Recently, many attempts were made to extend the systolic concept to VLSI signal/image processing [1,11,12,16,17,18]. In this paper, we develop a class of VLSI matrix manipulators for fast feature extraction and pattern classification.

We choose a modular approach to VLSI feature extraction/pattern classification based on Hwang-Cheng matrix partitioning algorithms [8]. Only limited types of VLSI matrix manipulating chips are needed to construct the hardware pattern recognizer. We shall first analyze the involved matrix/vector computations and then synthesize the needed VLSI matrix manipulators. The partitioned matrix computations are algorithmically specified with complexity analysis. For clarity purpose, we illustrate the design by constructing a linear pattern recognizer for a two-class problem environment. The results obtained in the sequel can be systematically generalized in a multi-class environment. The performance of the proposed VLSI feature extractors and pattern classifier will be compared with that of using a conventional SISD serial computer.

2. MATRIX COMPUTATIONS IN PATTERN RECOGNITION

A computational model of a statistical pattern recognition system is illustrated in Fig.2. All input vectors, $\bar{x}$ (the raw patterns to be recognized), form the input space V_n. To design a feature extractor, one has to produce a set of m tranformation vectors $\{\bar{d}_i \mid i=1,2,\ldots,m\}$ with the aid of a set S of training samples with known classes. Each $\bar{d}_i$ is an $n \times 1$ column vector. We denote the j-th sample of class s as $x_j^{(s)}$. The output of the extractor is the feature vector, $\bar{y}$, which is related to the input $\bar{x}$ by the following linear transformation.

$$\bar{y} = D \cdot \bar{x} \qquad (1)$$



$$\bar{y} = D \cdot \bar{x} \quad \text{(Linear transformation)}$$

$$f_{st}(\bar{y}) = \bar{v} \cdot \bar{y} + \alpha \quad \text{(Linear discriminant functions)}$$

$$\text{for } s \neq t \ \varepsilon \{1,2,\ldots,K\}$$

Fig.2 Computational model of a statistical pattern recognition system.

where $D = [\bar{d}_1, \bar{d}_2, \ldots, \bar{d}_m]^T$ is the $m \times n$ transformation matrix. Eigenvectors have been used to determine this matrix D in Karhunen-Loéve expansion [2], in Kukunaga-Koontz [6], and Chien and Fu [2]. Foley and Sammon [3] introduced a discriminating method to determine an optimal set of transformation vectors based on maximum separability instead on best fitting. We choose to modify Foley-Sammon algorithm to allow modular VLSI implementation of the feature extractor.

Let N_s be the number of training samples and μ_s be the sample mean for pattern class s (s=1,2, in a two-class environment). The sample offset is denoted by $z_j^{(s)} = x_j^{(s)} - \mu_s$ for $j = 1,2,\ldots,N_s$. An $n \times N_s$ sample offset matrix is formed by $Z_s = [z_1^{(s)}, z_2^{(s)}, \ldots, z_{N_s}^{(s)}]$, where each $z_j^{(s)}$ is an $n \times 1$ column vector. The $n \times n$ within class scatter matrix S_s (for class s) is obtained by performing an orthogonal matrix multiplication.

$$S_s = Z_s \cdot Z_s^T \tag{2}$$

In Foley-Sammon method, a weighted scatter matrix is defined between the pattern classes s and t.

$$A_{st} = c \cdot S_s + (1-c) \cdot S_t \tag{3}$$

where $0 \leqq c \leqq 1$ is determined by a "generalized" Fisher criterion as described in [3]. For a two-class environment in which s = 1 and t = 2, we simply denote $A_{st} = A_{12} = A$. Let $\mu = \mu_1 - \mu_2$ be the mean difference. We define a $h \times h$ matrix $B_h = (b_{ij})$ for $h = 1,2,\ldots,m-1$, where $b_{ij} = \bar{d}_i^T \cdot A^{-1} \cdot d_j$ for $1 \leqq i, j \leqq h$. We rewrite Foley-Sammon algorithm as follows :

GENERATION OF TRANSFORMATION VECTORS

Step 1. Initialize $i = 1$ and $B_1^{-1} = b_{11}^{-1}$. Compute $\bar{d}_1$ by

$$\bar{d}_1 = \alpha_1 \cdot A^{-1} \cdot \mu \tag{4}$$

where α_1 is a scalar constant chosen to satisfy $\bar{d}_1 \cdot \bar{d}_1^T = 1$ and $\alpha_1^2 = (\mu^T \cdot [A^{-1}]^2 \cdot \mu)^{-1}$

Step 2. Increment $i \leftarrow i + 1$.

Step 3. Compute the i-th transformation vector $\bar{d}_i$ by

$$\bar{d}_i = \alpha_i \cdot A^{-1} \cdot (\mu - [\bar{d}_1, \bar{d}_2, \ldots, \bar{d}_{i-1}] \cdot B_{i-1}^{-1} \cdot \beta) \tag{5}$$

where $\beta = [1/\alpha_1, 0, 0, \ldots, 0]^T$ is a $(i-1) \times 1$ column vector and α_i is a normalizing constant such that $\bar{d}_i^T \cdot \bar{d}_i = 1$. Go to Step 2, if $i < m$. Halt if $i \geq m$.

The computations specified in Eqs.4 and 5 involve the inversions of A and B_i for $i = 1, 2, \ldots, m-1$, which are very lengthy. Instead of using the recursive matrix inversion method suggested in [3], we shall describe in section 3 a block-partitioning method to generate the inverse matrices, A^{-1} and B_i^{-1}, through "partitioned" L-U decomposition.

We use linear discriminant functions for multiclass pattern classification [3,6,13,15]. To distinguish among K pattern classes, $K(K-1)/2$ pairwise discriminant functions are needed. The discriminant function between two distinct classes s and t is defined by

$$f_{st}(\bar{y}) = \bar{v}_{st}^T \cdot \bar{y} + \alpha_{st} \tag{6}$$

where $\bar{y} = (y_1, y_2, \ldots, y_m)^T$ is the feature vector, $\bar{v}_{st} = (v_1, v_2, \ldots, v_m)^T$ is called the discriminant vector and α_{st} is a scalar threshold constant. All discriminant vectos, $\bar{v}_{st}$, and threshold constants, α_{st}, are determined with the aid of a set F of training feature vectors with known class labels. For a two-class recognition system, we simply write Eq.6 as $f(\bar{y}) = \bar{v}^T \cdot \bar{y} + \alpha$. The feature pattern $\bar{y}$ is classified into class s, if $f_{st}(\bar{y}) \geq 0$; and into class t, if otherwise.

We choose to implement Fisher's method in generating the discriminant vector $\bar{v}$ from the training feature vectors. The threshold constant α can be set to be "zero" with appropriate choice of the coordinate system. Let $y_j^{(s)}$ be the j-th training feature vector of class s, for $j = 1, 2, \ldots, M_s$. We denote the feature mean difference as $\theta_{st} = \theta_s - \theta_t$ and feature offset vector as $w_j^{(s)} = y_j^{(s)} - \theta_s$. Again, we define an $m \times M_s$ feature offset matrix $W_s = [w_1^{(s)}, w_2^{(s)}, \ldots, w_{M_s}^{(s)}]$ for each class $s = 1, 2, \ldots, K$. The covariance matrix for class s is computed by

$$\Sigma_s = \frac{1}{M_s - 1} \cdot W_s \cdot W_s^T \qquad (7)$$

where Σ_s has dimension $m \times m$. We denote the convariance sum matrix $\Sigma_{st} = \Sigma_s + \Sigma_t$. The following computation steps are needed to generate the discriminant vector.

GENERATION OF DISCRIMINANT VECTORS

Step 1. Compute the feature mean differences $\theta_{st} = \theta_s - \theta_t$ for all distinct class pairs. Generate the feature offset matrices W_s for all classes by subtracting the mean θ_s from each training feature vector $y_j^{(s)}$.

Step 2. Perform matrix multiplication as specified in Eq.7 to generate convariance matrices Σ_s for all classes. Pairwise add the covariance matrices to produce Σ_{st} for all $s \neq t$.

Step 3. Per each pair of classes, solve the following linear system of equations to determine the m-dimensional vector $\bar{v}_{st}$.

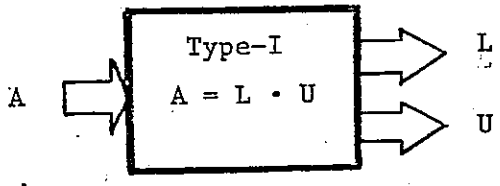
$$\Sigma_{st} \cdot v_{st} = \theta_{st} \qquad (8)$$

We use a system triangularization approach to solve Eq.8. Again, partitioned matrix computations are performed by VLSI modules in finding the L-U decomposition of Σ_{st} and then obtaining $\overline{v}_{st}$ through back substitution.

3. PARTITIONED MATRIX ALGORITHMS

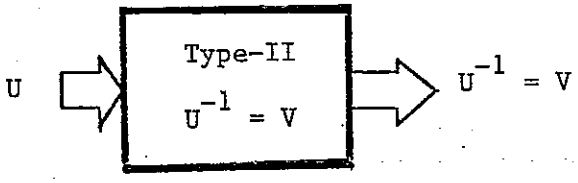
Four basic types of VLSI arithmetic modules are functionally specified in Fig.3. These VLSI chip types will be used as building blocks in implementing the partitioned matrix algorithms. Detailed cellular logic design of these pipelined arithmetic modules can be found in reference [8]. Only their functional specifications are presented here. The Type-I module is used for L-U decomposition of an $r \times r$ "intermediate" submatrix of a given high-order matrix of dimension $n \times n$ (or $m \times m$), where $n >> m >> r$. The Type-II modules are for the inversion of triangular submatrices of size $r \times r$. Both Type I and Type II modules have a fixed time delay of $2r$ time units, where each time unit equals the clock time for one multiply-add operation, $a + b * c = d$, or for one divide operation, $a/b = c$, by one step processor in the systolic array [10] or in the cellular pipelines [8].

The Type-III module performs accumulative matrix multiplications. The number, t, of pairs of $r \times r$ matrices to be multiplied is determined by the external input sequence. This leads to the time delay of $r \cdot t + 1$ for a Type-III module. The Type-IV module, performing additive multiplications of submatrix-vector pairs, is reduced from Type III and, thus, has the same modular delay. Type-I, Type-II, and Type-III modules are each constructed with $r \times r$ step processors with interior chip complexity $O(r^2)$. Type IV module constains r step processors forming a linear pipeline with interior chips complexity $O(r)$.



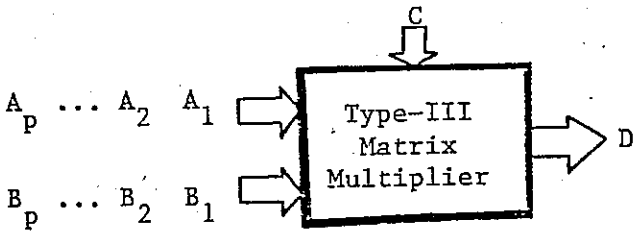
$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ \ell_{21} & 1 & 0 \\ \ell_{31} & \ell_{32} & 1 \end{pmatrix} \cdot \begin{pmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{pmatrix}$$

(a). Type-I VLSI Module (r = 3 shown)



$$\begin{pmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{pmatrix}^{-1} = \begin{pmatrix} v_{11} & v_{12} & v_{13} \\ 0 & v_{22} & v_{23} \\ 0 & 0 & v_{33} \end{pmatrix}$$

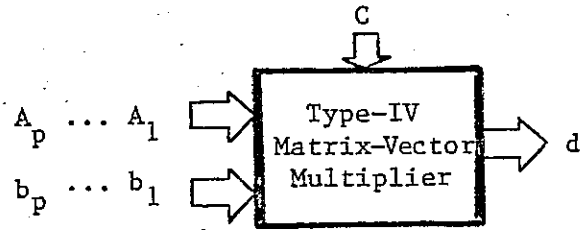
(b). Type-II VLSI module (r = 3 shown)



$$D = C + \sum_{i=1}^{p} A_i \cdot B_i$$

where C, D, A_i and B_i for $i = 1,\ldots,p$ are all $r \times r$ matrices

(c). TYPE-III VLSI Module



$$d = c + \sum_{i=1}^{l} A_i \cdot b_i$$

where c, d and b_i for $i = 1,\ldots,p$ are all $r \times 1$ column vectors and A_i for $i = 1,\ldots, p$ are all $r \times r$ matrices.

(d). Type-VI VLSI module

Fig.3 Primitive matrix manipulating VLSI modules of size r.

Four important matrix algorithms were shown partitionable by Hwang and Cheng [8]. Instead of presenting the detailed algorithm steps, we show the involved matrix computations with partitioning diagrams in Fig.4. Each "capital" entry in the matrix diagrams represents an $r \times r$ submatrix. The given matrix has order n(or m). We assume $n = r \cdot k$ for some integer k. The given matrix A is thus partitioned into $k^2 = n^2/r^2$ submatrices. For clarity purpose, we illustrate an example case of $n/r = k = 3$ in Fig.4.a. The L-U decomposition of matrix A can be done in $2k - 1 = 2 \cdot 3 - 1 = 5$ submatrix steps. All the diagonal elements of L equal to 1. All diagonal submatrices of U are upper triangular matrices. Submatrix computations in each step are specified along with the needed VLSI module types.

The partitioned matrix multiplication, $C = A \cdot B$, can be done in one step by computing all k^2 $r \times r$ product submatrices $\{C_{ij} \mid C_{ij} = \sum_{s=1}^{k} A_{is} \cdot B_{sj}$ for i,j = 1,2,...,k} in parallel with k^2 Type-III modules. The inversion of an n n triangular matrix can be done in k submatrix computation steps (The case of k = 4 is shown in Fig.4.b). Figure 4.c shows the VLSI solution of a triangular linear system of equations in (k = 3) subvector steps. Speed performance and hardware requirements of these four partitioned matrix algorithms are summarized in Table 1.

4. VLSI MATRIX MANIPULATORS

In this section, we present two pipelined matrix manipulation networks using the primitive VLSI arithmetic modules with interior complexity $C = O(r^2)$ as shown in section 3. One manipulator is for L-U decomposition of an $n \times n$ matrix A. The other is for the inversion of a triangular matrix of order n. As shown in Table 1, Linear computation time, O(n) can be achieved at the expense of using $O(n^2/r^2)$ VLSI modules for both algorithms. For $n >> r$, the quadratic chip count becomes too large to be cost-effective. Therefore, we prefer to implement the partitioned matrix algorithms in a serial-parallel approach. Such serial-parallel mode results in a linear VLSI chip count, O(n/r). Of course, the

$$(n \times n)$$

$$A = \begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix} = \begin{pmatrix} L_{11} & 0 & 0 \\ L_{21} & L_{22} & 0 \\ L_{31} & L_{32} & L_{33} \end{pmatrix} \cdot \begin{pmatrix} U_{11} & U_{12} & U_{13} \\ 0 & U_{22} & U_{23} \\ 0 & 0 & U_{33} \end{pmatrix} = L \cdot U$$

Note :

1. All A_{ij}, L_{ij}, U_{ij} are $r \times r$ submatrices. $n = 3r$
2. L_{11}, L_{22}, L_{33} are lower triangular submatrices with all diagnoal elements equal to 1.
3. U_{11}, U_{22}, U_{33} are upper triangular submatrices.

Step 1. $A_{11} = L_{11} \cdot U_{11}$ (Type I).

Step 2. $L_{21} = A_{21} \cdot U_{11}^{-1}$; $L_{31} = A_{31} \cdot U_{11}^{-1}$

$U_{12} = L_{11}^{-1} \cdot A_{12}$; $U_{13} = L_{11}^{-1} \cdot A_{13}$ } (Types II and III)

Step 3. $\hat{A}_{22} = A_{22} - L_{21} \cdot U_{12} = L_{22} \cdot U_{22}$

$\hat{A}_{23} = A_{23} - L_{21} \cdot U_{13}$; $\hat{A}_{32} = A_{32} - L_{31} \cdot U_{12}$ } (Type III, I)

Step 4. $U_{23} = L_{22}^{-1} \cdot \hat{A}_{23}$; $L_{32} = \hat{A}_{32} \cdot U_{22}^{-1}$ (Type III, II)

Step 5. $\hat{A}_{33} = A_{33} - (L_{31} \cdot U_{13} + L_{32} \cdot U_{23}) = L_{33} \cdot U_{33}$ (Type III, Type I)

Fig.4 Hwang-Cheng partitioned matrix algorithm : Part(a) L-U decomposition of a matrix A with dimension $n \times n$, where $n = 3r$.

$$U^{-1} = \begin{pmatrix} U_{11} & U_{12} & U_{13} & U_{14} \\ 0 & U_{22} & U_{23} & U_{23} \\ 0 & 0 & U_{33} & U_{34} \\ 0 & 0 & 0 & U_{44} \end{pmatrix}^{-1} = \begin{pmatrix} V_{11} & V_{12} & V_{13} & V_{14} \\ 0 & V_{22} & V_{23} & V_{24} \\ 0 & 0 & V_{33} & V_{34} \\ 0 & 0 & 0 & V_{44} \end{pmatrix} = V$$

Step 1. $V_{11} = U_{11}^{-1}$; $V_{22} = U_{22}^{-1}$; $V_{33} = U_{33}^{-1}$; $V_{44} = U_{44}^{-1}$ (Type I)

Step 2. $V_{12} = -V_{11} \cdot (U_{12} \quad V_{22})$; $V_{23} = -V_{22} \cdot (U_{23} \quad V_{33})$;

$V_{34} = -V_{33} \cdot (U_{34} \quad V_{44})$ (Type III)

Step 3. $V_{13} = -V_{11} \cdot (U_{12} \cdot V_{23} + U_{13} \cdot V_{33})$ (Type III)

$V_{24} = -V_{22} \cdot (U_{23} \cdot V_{34} + U_{24} \cdot V_{44})$

Step 4. $V_{14} = -V_{11} \cdot (U_{12} \cdot V_{24} + U_{13} \cdot V_{34} + U_{14} \cdot V_{44})$ (Type III)

(b) Partitioned inversion for an example triangular matrix of order $n = 4r$

$$\begin{pmatrix} U_{11} & U_{12} & U_{13} \\ 0 & U_{22} & U_{23} \\ 0 & 0 & U_{33} \end{pmatrix} \cdot \begin{pmatrix} X_1 \\ X_2 \\ X_3 \end{pmatrix} = \begin{pmatrix} D_1 \\ D_2 \\ D_3 \end{pmatrix}$$

Note :

X_i and D_i for $i = 1,2,3$ are $r \times 1$ column subvectors

Step 1. $X_3 = U_{33}^{-1} \cdot D_3$ (Type II, IV)

Step 2. $X_2 = U_{22}^{-1} \cdot (D_2 - U_{23} \cdot X_3)$ (Type II, IV)

Step 3. $X_1 = U_{11}^{-1} \cdot [D_1 - (U_{12} \cdot X_2 + U_{13} \cdot X_3)]$ (Types II, IV)

(c) Partitioned solution of a triangularized linear system of equations for $n = 3r$.

Fig.4 (continued) Huang-Cheng partitioned matrix algorithms : Part (b) and Part (c)

Table 1. Speed Performance and Hardware Requirement of Partitioned Matrix Algorithms

Operation mode / Algorithm	Strictly Parallel Operation		Serial-Parallel Operation	
	VLSI Chip Count	Compute Time	VLSI Chip Count	Compute Time
L-U Decomposition (Fig.4a,5)	$O(n^2/r^2)$	$O(n)$	$O(n/r)$	$O(n^2/r)$
Triangular Matrix Inversion (Fig.4b,6)	$O(n^2/r^2)$	$O(n)$	$O(n/r)$	$O(n^2/r)$
Matrix Multiplication	$O(n^2/r^2)$	$O(n)$	$O(n)$	$O(n^2/r)$
Solving Triangular Linear System of Equations	$O(n/r)$	$O(n)$		

n : Matrix order

r : Module size

time delay is prolonged to be $O(n^2/r)$, still much faster than $O(n^3)$ delay with the use of a uniprocessor.

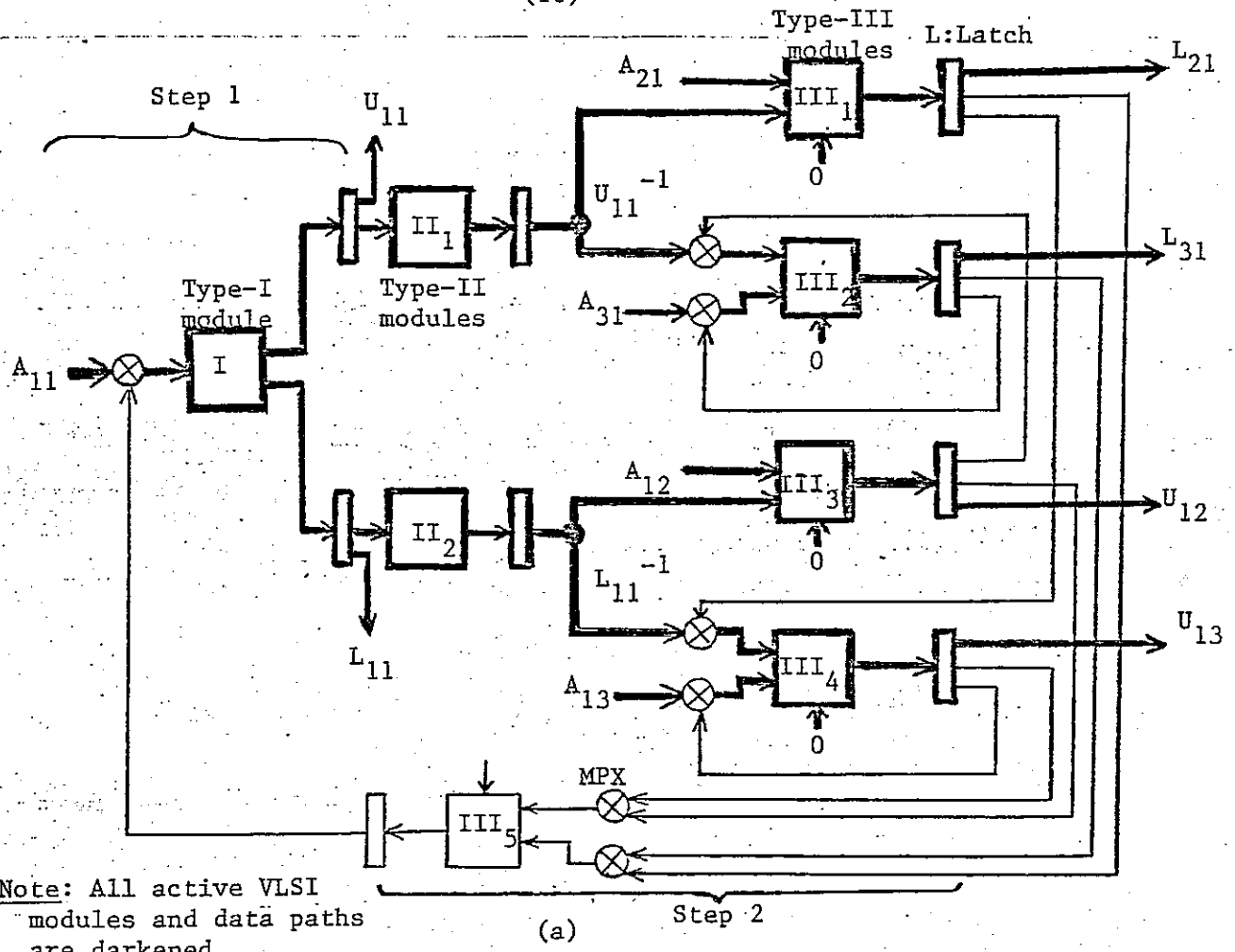
A VLSI L-U decomposition network in shown in Fig.5 for the case of $k = n/r = 3$. In general, such a network requires to use one Type-I module for submatrix L-U decomposition, two submatrix inverters (II_1 and II_2 as shown), and $2k - 1$ Type III modules for additive submatrix multiplications. These arithmetic modules are interfaced with high-speed latch memories to yield pipelining operations with feedback connections. For the example matrix decomposition shown in Fig.4.a, we show the snap shots of the $2k - 1 = 5$ submatrix computing steps in Fig.5.a,b,c,d. The active modules and data paths involved in each step are stressed by "boldface" boxes and lines in the snap shots.

Multiplexers are used to select the appropriate input to the functional modules at different steps. Note that the steps are divided according to the boundary of submatrix operations. Each step may require different number of pipeline cycles to complete the operation. To decompose an $n \times n$ matrix A into two triangular matrices, L and U. such that $A = L \cdot U$, the network shown in Fig.5 requires the following computation time

$$T_1 = n^2/2r - n/2 + 2n/r + 17r - 2$$
$$\doteq O(n^2/r) \qquad \text{for } n >> r \tag{9}$$

The total chip count of this L-U decomposition network equals

$$M_1 = 1 + 2 + (2n/r - 1) = 2n/r + 2$$
$$\doteq O(n/r) \qquad \text{for } n >> r \tag{10}$$



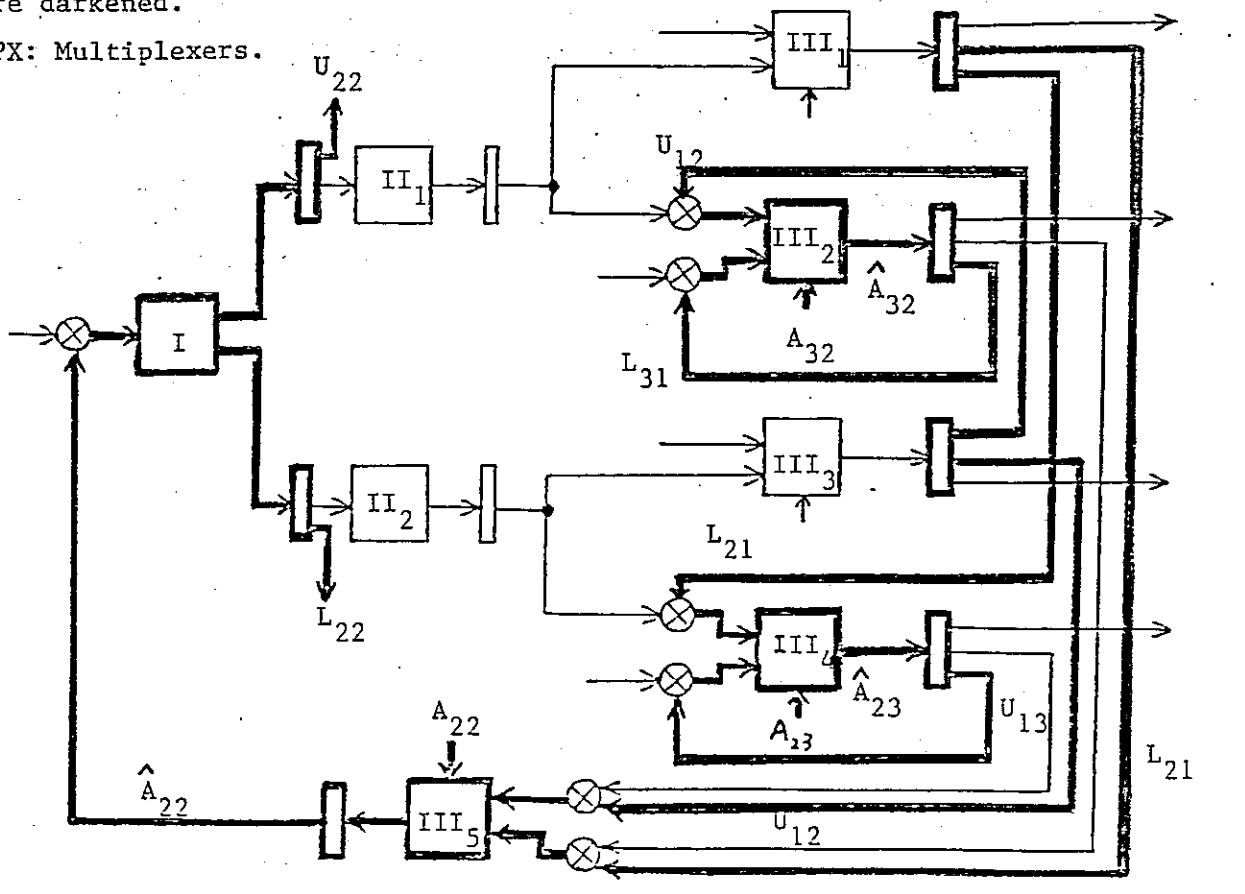
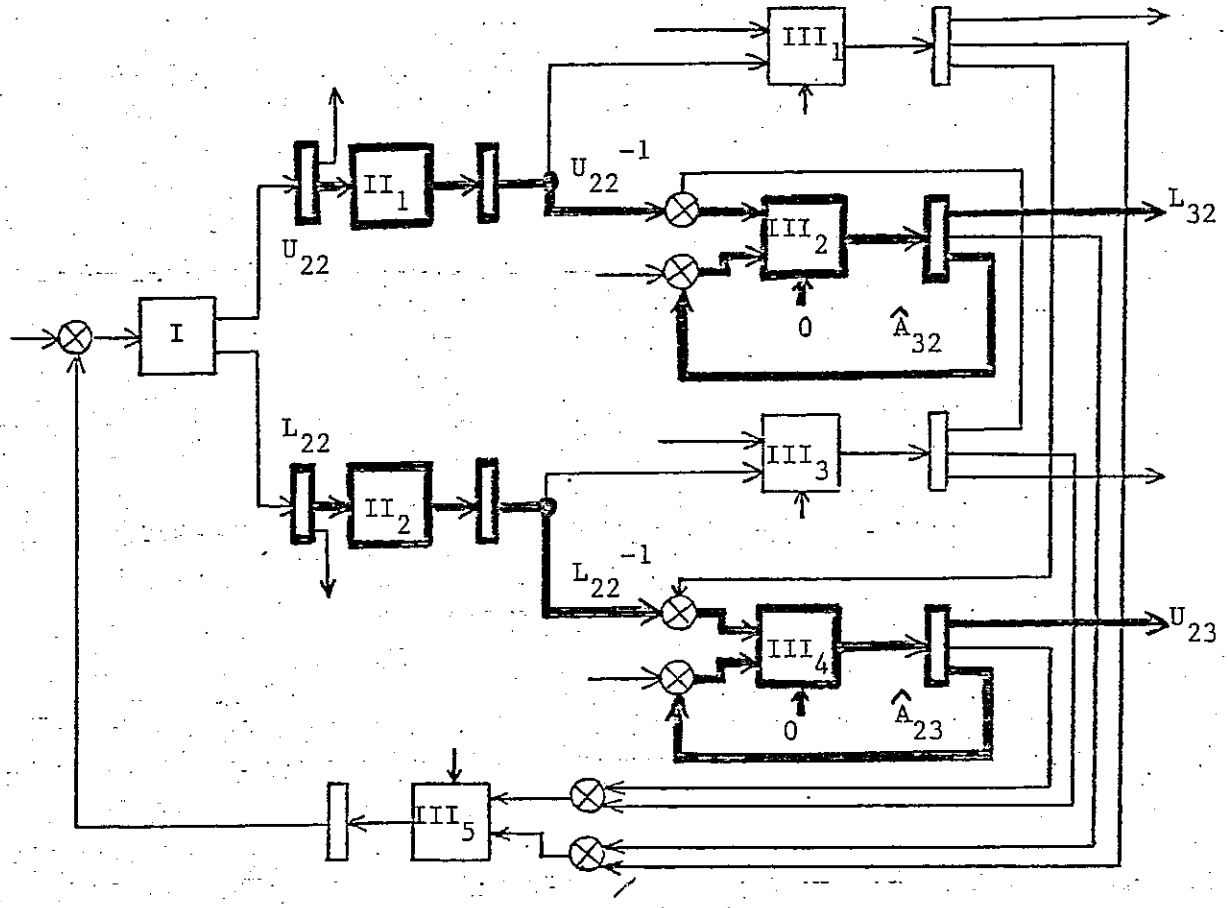
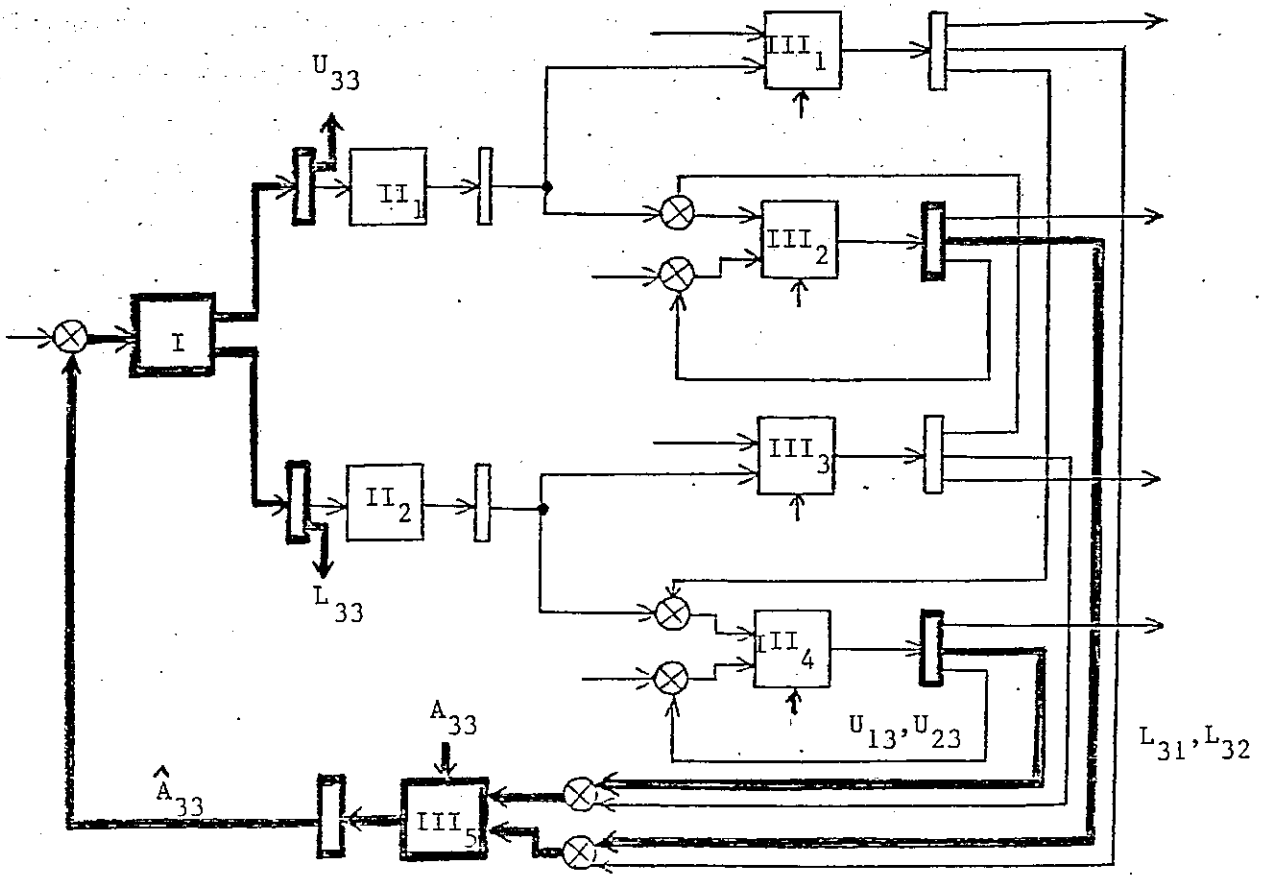


Fig.5 Functional design of a pipelined VLSI matrix network for partitioned L-U decomposition : (a) Step 1 and Step 2, (b) Step 3. (The network for k = n/r = 3 is shown).



(c) Step 4



(d) Step 5

Fig.5 (continued) : (c) Step 4, (d) Step 5.

It is interesting to note that $T_1 \cdot M_1 \cdot C = O(n^2/r) \cdot O(n/r) \cdot O(r^2) = O(n^3)$, where $O(n^3)$ is the time complexity of implementing the L-U decomposition algorithm by a uniprocessor.

The matrix inversion algorithm in Fig.4.b is realized by the pipelined matrix manipulator in Fig.6 for the case of $k = 4$. In general, inverting an $n \times n$ triangular matrix requires to use k Type-II submatrix inverters and $2(k-1)$ Type-III submatrix multipliers. Thus the total module count equals

$$M_2 = k + 2(k-1) = 3k - 2$$
$$\simeq O(k) = O(n/r) \quad \text{for } n >> r \tag{11}$$

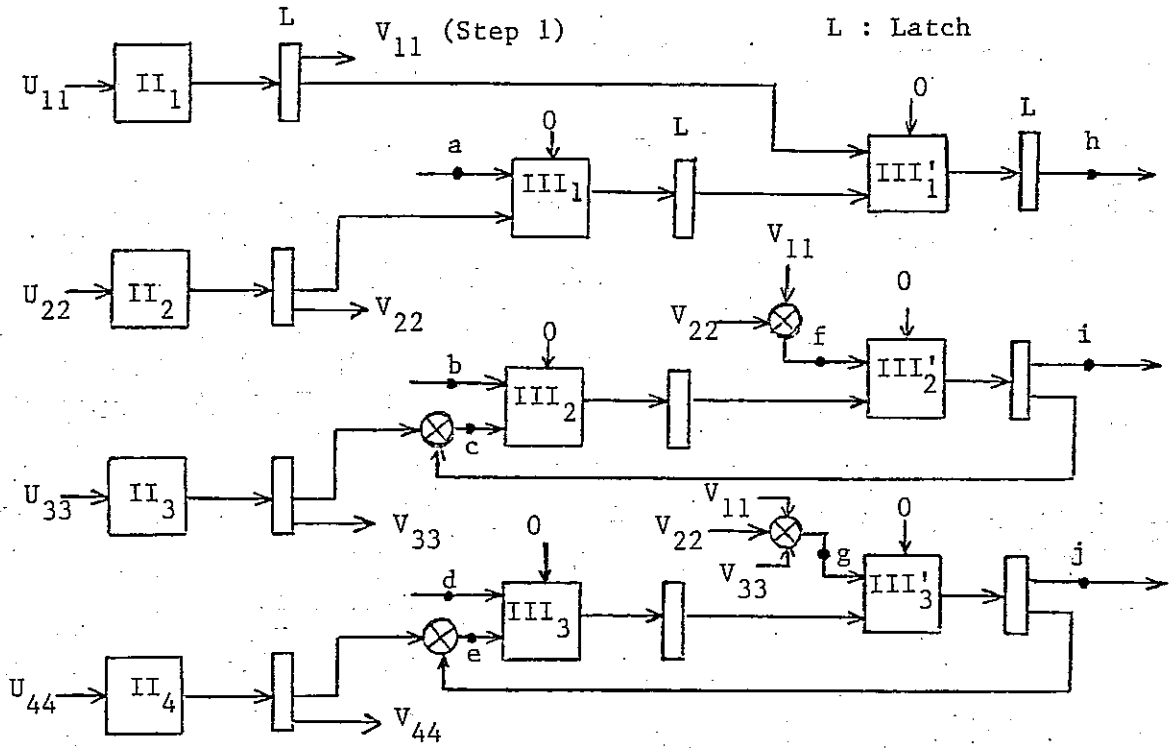
The input assignments and data flows at intermediate and output terminals are specified at the attached table for the four submatrix steps. The total time of using this network to generate the inverse matrix $V = U^{-1}$ is estimated to be

$$T_2 = n^2/2r + n/2 + 2n/r + r - 2$$
$$= O(n^2/r) \text{ for } n >> r \tag{12}$$

We shall use these two matrix manipulation pipelines in the construction of the feature extrator and pattern classifier.

5. VLSI FEATURE EXTRACTION/PATTERN CLASSIFICATION

In a feature extraction process, the major computations are to perform $Z_s \cdot Z_s^T$, $Z_t \cdot Z_t^T$, and $[\bar{d}_1, \bar{d}_2, \ldots, \bar{d}_{i-1}] \cdot B_{i-1}^{-1}$, and to generate the inverse of matrices A and B_i for all $i = 1, 2, \ldots, m-1$. We denote the integer ratio $N_s/r = h_s$ and $N_t/r = h_t$. Figure 7 shows the functional design of a VLSI feature extraction network. The Offset Matrix Generator produces matrix Z_s and Z_t by subtraction.



	Terminals							Output		
	a	b	c	d	e	f	g	h	i	j
Step 2	U_{12}	U_{23}	V_{33}	V_{34}	V_{44}	V_{22}	V_{33}	V_{12}	V_{23}	V_{34}
Step 3		U_{12} U_{13}	V_{23} V_{33}	U_{23} U_{24}	V_{34} V_{44}	V_{11}	V_{22}		V_{13}	V_{24}
Step 4				U_{12} U_{13} U_{14}	V_{24} V_{34} V_{44}		V_{11}			V_{14}

Note : All 4 Type-II modules are active during Step 1.
Modules III_1 and III'_1 are active in Step 2.
Modules III_2 and III'_2 are active in Steps 2,3.
Modules III_3 and III'_3 are active in Steps 2,3,4.

Fig.6 Functional design of a pipelined VLSI matrix network for partitioned matrix inversion. (The network for k = n/r = 4 is shown).

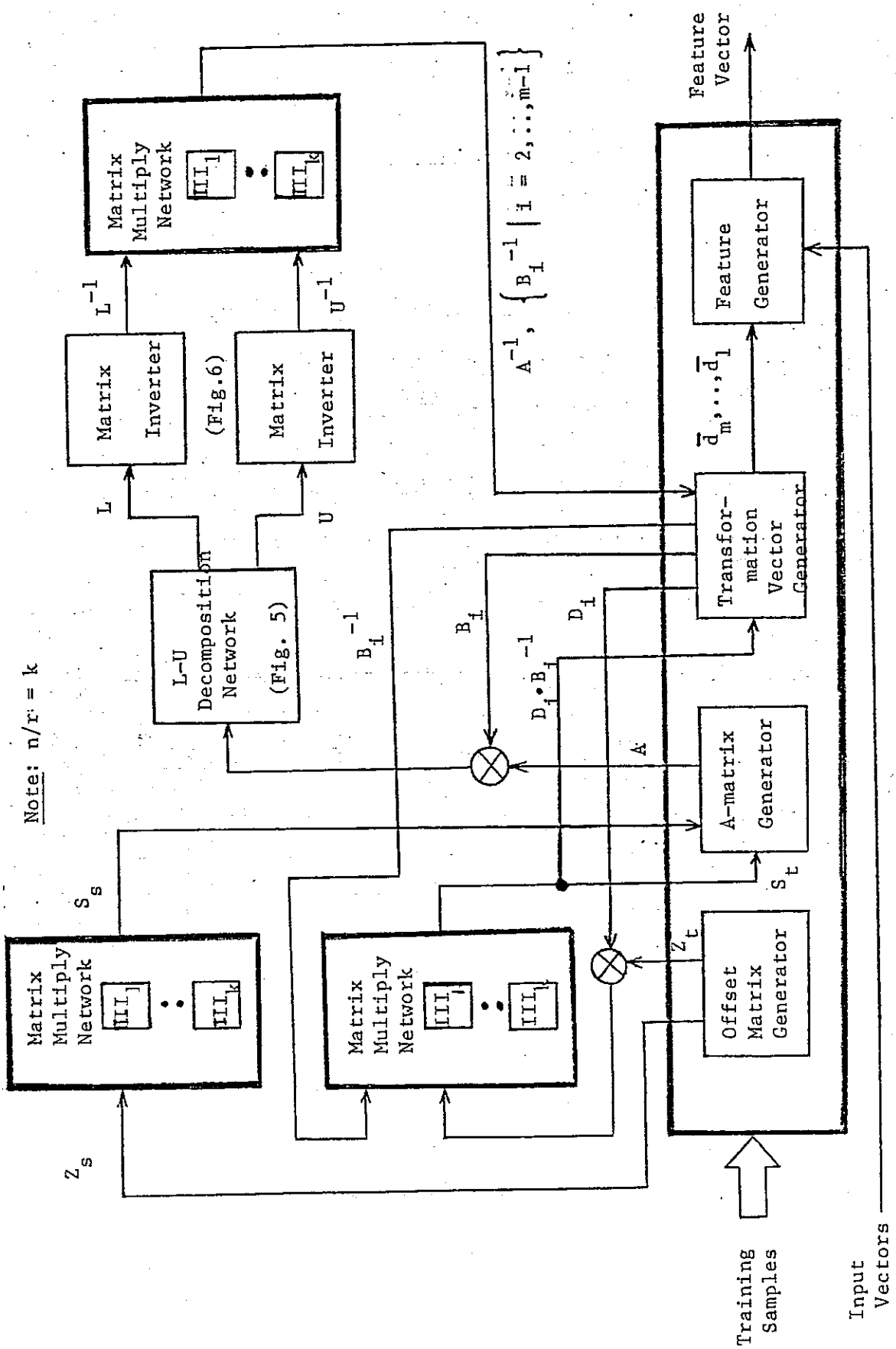


Fig.7 Functional design of a feature extractor using the proposed VLSI matrix manipulators.

A-matrix Generator performs weighting and addition of Z_s and Z_t. The Transformation Vector Generator computes the vectors $\bar{d}_1, \bar{d}_2, \ldots, \bar{d}_m$. The Feature Generator performs the linear transformation specified in Eq.1.

Two Matrix Multiply Networks are used to compute the scatter matrices S_s and S_t. Each matrix multiply network consists of n/r independently operated Type-III VLSI modules. The L-U decomposition is based on the design shown in Fig.5. The inversions of triangular matrices L and U are generated by the network designed in Fig.6. We generate the inverse of matrix A by $A^{-1} = (L \cdot U)^{-1} = U^{-1} \cdot L^{-1}$. One L-U decomposition network, two VLSI matrix inverters, and one matrix multiply network are used for generating the matrix A^{-1}.

The hardware used to generate A^{-1} can be also used to obtain the inverse matrices B_i for all $2 \leqq i \leqq m-1$. The order of each matrix B_i equals i. The ratio i/r may not be necessarily an integer. We can always augment B_i to become B_h having order $h = \lceil k/r \rceil \cdot r$, which is an integer multiple of the module size r and, thus, implementable with the proposed VLSI modules. B_h is related to B_i by

$$B_h = \begin{pmatrix} B_i & 0 \\ 0 & I_j \end{pmatrix} \tag{13}$$

where I_j is an identity matrix of order $j = h - i$. The inverse of B_h can be computed by

$$B_h^{-1} = \begin{pmatrix} B_i^{-1} & 0 \\ 0 & I_j \end{pmatrix} \tag{14}$$

Let $D_i = [\bar{d}_1, \bar{d}_2, \ldots, \bar{d}_i]$. The matrix multiplication $D_i \cdot B_i^{-1}$ is performed by the same matrix multiply network generating the scatter matrix S_t. If i/r is not an integer, the matrices D_i and B_i^{-1} can be also augmented with zeros and

identity matrix in order to use size-r VLSI modules.

The time delay of a Fisher linear classifier is attributed mainly by the computations of $W_s \cdot W_s^T$, $W_t \cdot W_t^T$, and by solving Eq.8. To solve a dense system $\Sigma \cdot \bar{v} = \bar{\theta}$, first we perform L-U decomposition of matrix Σ to yield $(L \cdot U) \cdot \bar{v} = L \cdot (U \cdot \bar{v}) = \bar{\theta}$. This represents the solution of two triangular subsystems. The forward elimination is specified by $L \cdot \bar{\delta} = \bar{\theta}$, and the back substitution corresponds to $U \cdot \bar{v} = \bar{\delta}$. The solutions of these two subsystems lead to the solution of the original system charaterized by $\Sigma \cdot \bar{v} = \bar{\theta}$. Figure 8 shows the functional design of a VLSI linear pattern classifier. The Offset Matrix Generator performs matrix scaling and addition to yield the matrix Σ. The Threshold Generator produces the threshold constant α according to a given optimizing criterion. The matrices W_s and W_t are computed in parallel. Each matrix multiply network contains m/r Type-III VLSI modules. The L-U Decomposition Network triangularizes the matrix Σ. The Triangular System Solver solves the two triangular sybsystems, $L \cdot \bar{\delta} = \bar{\theta}$ and $U \cdot \bar{v} = \bar{\delta}$, sequentially. There are O(m/r) VLSI modules in the triangular system solver. The matrix-vector multiplier consists of β Type-IV VLSI modules, where $\beta = \lceil ((m/r - 3)r - 2)/(2r + 2) \rceil + 1$. For $m >> r$, $\beta = m/2r$. Maximal overlapped operations are performed in successive steps of using the matrix-vector multiplier. β subvector data buffers are used for iterative back substitution. The reason to use β identical buffers for storing multiple solution vectors is to enable parallel processing in the matrix-vector multiplier.

6. PERFORMANCE ANALYSIS AND CONCLUSIONS

Matrices involves in feature extraction have order n and involved in pattern classification have order m. All VLSI modules manipulate submatrices of order r. The proposed VLSI matrix manipulators perform efficiently under the assumption $n >> m >> r$. The systolic arrays are globally structured with

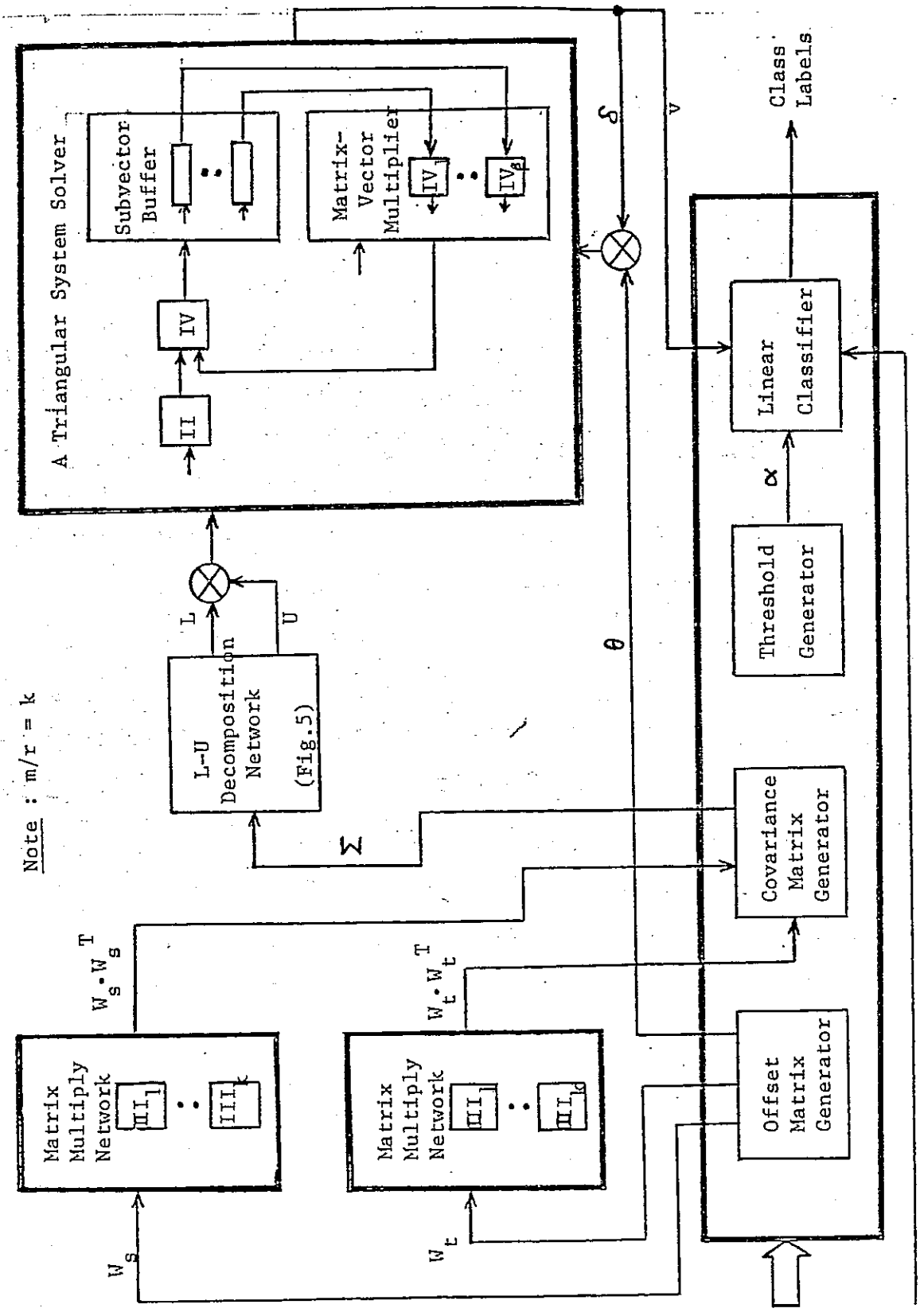


Fig.8 Functional design of a pattern classifier using the proposed VLSI matrix manipulators.

chip complexity $O(n^2)$ or $O(m^2)$. We have reduced the chip complexity to $O(r^2)$ through modular approach. Modulation makes it more feasible for IC fabrication and I/O packaging, and also provides better extensibility. The hardware L-U decompositor (Fig.5) and matrix inverter (Fig.6) each has a speedup of $O(n^3)/O(n^2/r)$ or $O(m^3)/O(m^2/r)$ over the conventional uniprocessor. The hardware chip counts have orders $O(n/r)$ and $O(m/r)$ respectively.

The matrix multipliers for computing the scatter matrices $Z_s \cdot Z_s^T$, $W_s \cdot W_s^T$, and the product matrices $D_i \cdot B_i^{-1}$ have, each, a chip count $O(n/r)$ in Fig.7 and $O(m/r)$ in Fig.8. We generate the resulting product matrix one row at a time with the same speedup as those for the L-U decompositor and matrix inverter. Of course, if $O(n^2/r^2)$ or $O(m^2/r^2)$ Type-III submatrix multipliers are used, the product matrix can be generated in linear time $O(n/r)$ or $O(m/r)$. The choice is up to cost effectiveness. The triangular system solver used in the pattern classifier (Fig.8) has a speedup of $O(m^2)/O(m/r)$ over the uniprocessor approach. The chip count in this solver is $O(m/r)$ with chip complexity $O(r)$ for the Type-IV modules. We have explored maximal concurrency in each submatrix computation step in these partitioned matrix operations. With pipelining, high degree of overlapped operations between successive submatrix steps are also explored.

Design tradeoffs in developing VLSI feature extractor and pattern classifier have to satisfy the conservation law between operating speed and hardware chip count, as demonstrated by the fact the $T \cdot M \cdot C = O(n^3)$, where T, M, and C are the compute time, chip count, and chip complexity respectively. For L-U decomposition, matrix inversion, and matrix multiplication, the following alterative choices can be made :

$$T \cdot M \cdot C = \begin{cases} O(n^2/r) \cdot O(n/r) \cdot O(r^2) = O(n^3) \\ \qquad \text{for serial-parallel mode} \\ O(n) \cdot O(n^2/r^2) \cdot O(r^2) = O(n^3) \\ \qquad \text{for strictly parallel mode.} \end{cases} \tag{15}$$

For pattern classification, the variable "n" in Eq.15 should be replaced by "m". For the triangular system solver in Fig.8, the following conservation law must be satisfied.

$$T \cdot M \cdot C = O(m) \cdot O(m/r) \cdot O(r) = O(m^2) \tag{16}$$

Feature extration and pattern classification are the initial condidates for possible VLSI implementation. We demonstrated only the VLSI realization of the Foley-Sammon feature extraction method and of the Fisher's linear classifier. Other methods such as the eigenvector approaches to feature selection and Bayes quadratic discriminant functions should be realizable with the proposed "partitioned" matrix manipulators. Many other computations required in image processing and pattern recognition are also good candidates for VLSI realization. It is highly desired to develop VLSI computing structures also for smoothing, image registration, edge detection, image segmentation, texture analysis, multi-stage feature selection, syntatic pattern recognition, pictorial query processing, and image database management, etc. We hope that this initial effort will inspire further research projects towards the development of effective real-time pattern-analysis and image-understanding system with the emerging VLSI technology. The potential gain lies not only in speed but also in realiabitity and cost-effectiveness.

REFERENCES :

[1] Chang, N.S. and Yuan, Y.C., "VLSI Design and Verification of a Signal Processing Chip", Proc. of Workshop on Computer Architecture for PAIDM, IEEE Computer Society, Hot Springs, Virginia, Nov. 1981, pp.279-283.

[2] Chien, Y. T. and Fu, K. S., "On The Generalized Karhumen-Loive Expression", IEEE Trans. Information Theory, Vol.ITT-13, July 1967, pp.518-520.

[3] Foley, D. H., and Sammon, J. W. Jr., "An Optimal Set of Discriminant Vectors," IEEE Trans. Comp., March 1975, pp.281-289.

[4] Foster, M. J., and Kung, H. T., "The Design of Special-Purpose VLSI Chips," Computer Magazine, Jan. 1980, pp.26-40.

[5] Fu, K. S., Syntactic Methods in Pattern Recognition, Academic Press, New York, 1976.

[6] Fukunaga, K., Introduction to Statistical Pattern Recognition, New York, Academic Press, 1972.

[7] Hwang, K., Computer Arithmetic: Principles, Architecture, and Design, John Wiley, New York, 1979, Chaps.6 and 8.

[8] Hwang, K., and Cheng, Y. H., "Partitioned Matrix Algorithms and VLSI Structures for Large-Scale Matrix Computations," IEEE 5th Symp. on Computer Arithmetic, May 1981, pp.222-232.

[9] Hwang, K., Su, S. P., and Ni, L. M., "Vector Computer Architecture and Processing Techniques," Advances in Computers, Vol.20, (M. C. Yovits, editor), Academic Press, New York, 1981, pp.115-197.

[10] Kung, H. T., and Leiserson, C. E., "Systolic Arrays (for VLSI)," in Sparse Matrix Proc., (Duff. I. S. et al. editors), Society for Indust. and Appl. Math., Pa. 1979, pp.256-282.

[11] Kung, H. T., and Picard, R. L., "Hardware Pipelines for Multi-dimensional Convolution and Resampling, Proc. Workshop on Computer Architecture for Pattern Analysis and Image Database Management, IEEE Computer Society, Hot Springs, Va., Nov. 11-13, 1981, pp.273-278.

[12] Kung, H. T., Ruane, L. M., and Yen, D. W. L., "A Two-Level Pipelined Systolic Array for Convolution," in VLSI Systems and Computations (Edited by H. T. Kung, et al), Computer Science Press, 1981, pp.255-264.

[13] Peterson, D. W., and Mattson, R. L., "A Method of Finding Linear Discriminant Functions for a Class of Performance Criteria," IEEE Trans. Information Theory, July 1966, pp.380-387.

[14] Rosenfeld, A., and Kak, A., Digital Picture Processing, Academic Press, 1976, New York, N. Y.

[15] Sammon, J. W. Jr., "Interactive Pattern Analysis and Classification," IEEE Trans. Computers, July 1970, pp.594-616.

[16] Swartzlander , E. E., "VLSI Architecture," in Very Large Scale Integration (VLSI): Fundamentals and Applications, (Edited by D. F. Barbe), Spring-Verlay, New York, 1980.

[17] Young, T. Y., and Liu, P. S., "VLSI Arrays and Control Structure for Image Processing," Proc. Workshop on Computer Architecture PAIDM, IEEE Computer Society, Hot Springs, Va., Nov. 11-13, 1981, pp.257-264.

[18] Yen, D. W. L., and Kulkarni, A. V., "The ESL Systolic Processor for Signal/Image Processing," Proc. of 1981 Workshop on Computer Architecture for PAIDM, Hot Springs, Va., Nov. 11-13, 1981, pp.265-272.