TR-82-010

VLSI MATRIX ARITHMETIC ALGORITHMS

Kai Hwang

VLSI MATRIX ARITHMETIC ALGORITHMS

Kai Hwang*
School of Electrical Engineering
Purdue University
W. Lafayette, Indiana, USA

I. INTRODUCTION

This paper presents a new class of matrix algorithms for possible VLSI implementation of large-scale matrix arithmetic solvers. Fast matrix solvers are highly demanded in signal/image processing and in many real-time and scientific applications. After partitioning, only a few functional types of VLSI arithmetic chips are needed in submatrix computations. This partitioned approach is not restricted by problem sizes and thus can be applied to solve arbitrarily large linear systems of equations in an interative fashion. Large-scale matrix computations are needed in solving high-order Linear System of Equations (LSE), $\underline{A} \cdot \underline{x} = \underline{b}$, in many important scientific and engineering application areas. So far, SIMD array processors or pipelined vector supercomputers have been used to solve large LSEs by predeveloped software packages [3,6,14,16]. Fast matrix algorithms for solving LSEs have been suggested by Crout [1], Kant and Kimura [9], Sameh and Kuck [15] and by many other researchers. The recent advent in Very Large Scale Integration (VLSI) microelectronic technology has created a new architectural horizon to implement large-scale vector/matrix computations directly in hardware. [4,5,7,10,12,13, 19,20,29,30].

* This article was written, while the author was visiting the Institute of Information Science, Academia Sinica, and Department of Information Engineering, National Taiwan University, from December 1982 to June 1982.

It has been projected by Mead and Conway [11] that by the late 1980's it will be possible to fabricate 10^7 or 10^8 transistors on a monolithic chip. VLSI computing structures have been suggested by Kung and Leiserson [10], Preparata dn Vuillemin [13], Hwang and Cheng [5], and Nash, et al [12]. A VLSI computing device contains not only a large number of processing cells but also a large number of interconnection paths throughout the integrated chip. The length and organization of these communication paths set a lower bound on the chip area and time delays required for system operations. Systolic VLSI arrays [10] were proposed with a global structure that must be limited in their array sizes due to bounded chip area and I/O packaging contraints.

Based on the state-of-the-art of electronic and packaging technologies, we can only expect VLSI arithmetic devices for regularly structured functions with limited I/O terminals. A modular approach to fabricate VLSI devices is amenable from the viewpoints of feasibility and applicability. We choose a matrix partitioning approach to overcome these technological constraints in constructing large-scale matrix solvers. These "partitioned" algorithms are for modular VLSI implementation of the following four classes of matrix computations :

- L-U decomposition by a new variant of Gaussian elimination.
- Normal inversion of a nonsingular triangular matrix.
- Multiplication of two compatible matrices.
- Solving a triangular system of equations by back substitutioon.

We reserve the parameter, n, for the order of a given dense matrix A, and the parameter, m, as the size of available VLSI arithmetic chips, where the ratio $k = n/m$ is an integer. We shall use boldface capital letters, A, L, U, V,..., to denote $n \times n$ matrices; indexed capitals, A_{ij}, L_{ij}, for $m \times m$ submatrices; boldface lower-case letters, x, b, d,..., as n-element column vectors; and indexed lower-case letters, a_{ij}, x_i,..., as matrix entries or vector components. All analytical results on hardware complexity and system performance are expressed in terms of these parameters n, m, and k under the assumption $n \gg m$, which holds for practical applications. The proposed VLSI matrix solvers can be applied in digital signal processing, structural analysis, Seismic exploitation, fluid dynamics, image processing, pattern recognition, computer-assisted tomography, numeric weather forcasting, artificial intelligence, and various real-time applications [6,7,20,22,25,26,27, 28,29,32]

II. VLSI MATRIX COMPUTATIONS

For L-U decomposition by Gaussian elimination, we consider only nonsingular LSEs in which all the principal minor submatrices of $A = (a_{ij})$ are nonsingular. This provides a necessary and sufficient condition to produce a unique lower triangular matrix $L = (\ell_{ij})$ with $\ell_{11} = \ell_{22} = \dots = \ell_{nn} = 1$, and a unique upper triangular matrix $U = (u_{ij})$ wuch that $L \cdot U = A$. In Crout's reduction method [1], the matrix $A = L \cdot U$ is decomposed according to the following computations for $i = 1,2,\dots, n$.

$$\begin{cases} u_{ik} = a_{ik} - \sum_{j=1}^{i-1} \ell_{ij}\, u_{jk} \quad \text{for } k=i,i+1,\cdots,n. \\ \ell_{ki} = \left[a_{ki} - \sum_{j=1}^{i-1} \ell_{kj}\cdot u_{ji}\right] / u_{ii} \quad \text{for } k=i,i+2,\cdots,n. \\ \text{provided } \ell_{ii}=1 \text{ for } i=1,2,\cdots,n. \end{cases} \tag{1}$$

Crout's method does not require to interchange columns and, thus, eliminates the recording of intermediate results. Instead of dealing with one row or one column at a time, we have modified Crout's method to a new variant of Caussian elimination by processing rows/columns of $m \times m$ submatrices in parallel. This submatrix approach leads to the partitioned L-U decomposition algorithm to be described in Section III.

After the L-U decomposition, one can transform the original system $\underline{A} \cdot \underline{x} = \underline{b}$ to $\underline{L} \cdot \underline{U} \cdot \underline{x} = \underline{b}$ and then to an equivalent triangular system characterized by $\underline{U} \cdot \underline{x} = \underline{L}^{-1} \cdot \underline{b} = \underline{d}$. With this triangularized system, one can compute the solution vector $\underline{x}$ by $\underline{x} = \underline{U}^{-1} \cdot \underline{d}$. The inverse matrices $\underline{U}^{-1}$ and $\underline{L}^{-1}$ always exist, because $\underline{U}$ and $\underline{L}$ are both nonsingular. We denote the inverse matrix $\underline{U}^{-1} = \underline{V} = (v_{ij})$, which is again a triangular matrix with entries calculated by

$$\begin{cases} v_{kk} = 1/u_{kk} \quad \text{for } k=1,2,\cdots,n \\ v_{ij} = -\left(\sum_{k=i+1}^{j} u_{ik}\cdot v_{kj}\right)/u_{ii} \quad \text{for all } j>i. \end{cases} \tag{2}$$

In Section IV, we shall partition the above computations to enable block generation of v_{ij} entries.

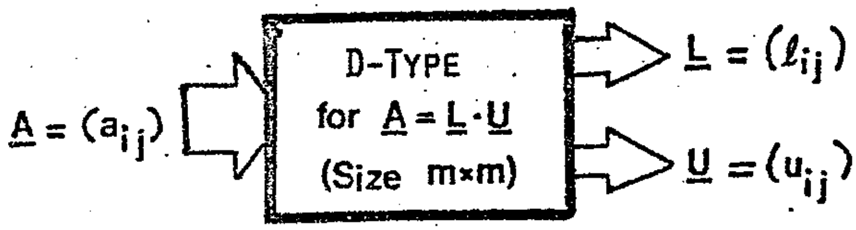
In fact, the (d_j) elements in vector $\underline{d}$ can be generated automatically by applying partitioned L-U decomposition of an $n \times (n+1)$ matrix obtained by adding the $\underline{b}$ vector as the $(n+1)$-th column in matrix $\underline{A}$, that is $a_{i,n+1} = b_i$ for $i = 1,2,\ldots,n$. The solution vector $\underline{x}$ is computed by back substitution. This sequence of computations can be also done in subvectors to be described in Section IV.

$$\begin{cases} x_n = d_n/u_{nn} \\ x_i = \left[d_i - \sum_{j=i+1}^{n} u_{ij}\cdot x_j\right]/u_{ii} \\ \text{for } i=n-1,n-2,\cdots,2,1 \end{cases} \tag{3}$$

III. PRIMITIVE VLSI MATRIX CHIPS

Four primitive types of VLSI arithmetic chip types are functionally introduced in Figs. 1-4. These VLSI chips will be used as building blocks in implementing the partitioned matrix algorithms. These chip types are used to perform $m \times m$ submatrix or m-element subvector computations. Each chip is constructed with a cellular array of multipliers, dividers, and interface latches for pipelined operations [4,10,12]. Detailed schematic logic designs of these primitive VLSI chips can be found in reference [5,7]. Only their functional specifications are given here.

The D-Type chips are for L-U decomposition of each intermediate $m \times m$ submatrix, $A_{rr} = L_{rr} \cdot U_{rr}$, along the principal diagonal of $\underline{A}$ (A_{rr} will be defined shortly). The I-Type chips are for the inversion of triangular $m \times m$ submatrices L_{rr} and U_{rr}. The input/



$$\underline{A} = \begin{bmatrix} a_{11} & a_{12} \cdots & a_{1m} \\ a_{21} & a_{22} \cdots & a_{2m} \\ \vdots & & \vdots \\ a_{m1} & a_{m2} \cdots & a_{mm} \end{bmatrix} = \underline{L} \begin{bmatrix} 1 & & & \\ \ell_{21} & 1 & & \\ \ell_{31} & \ell_{32} & 1 & \\ \vdots & & & \ddots \\ \ell_{m1} & \ell_{m2} & \cdots & \ell_{m,m-1} \; 1 \end{bmatrix} \cdot \underline{U} \begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots u_{1m} \\ & u_{22} & u_{23} & \cdots u_{2m} \\ & & u_{33} & \cdots u_{3m} \\ & & & \ddots \; \vdots \\ & & & u_{mm} \end{bmatrix}$$

Fig.1 Functional specification of D-Type VLSI chips for L-U decomposition of intermediate submatrices

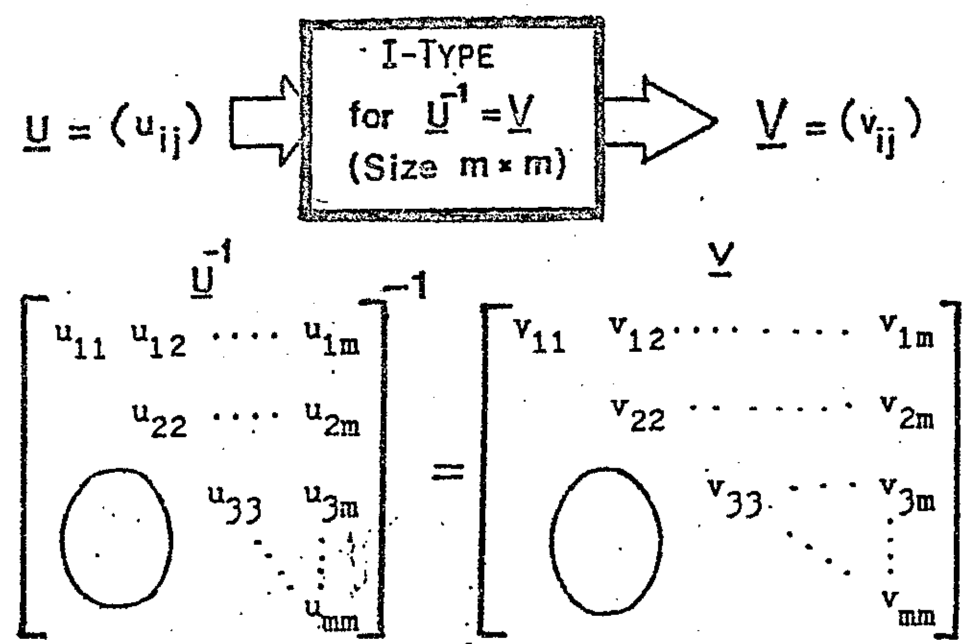


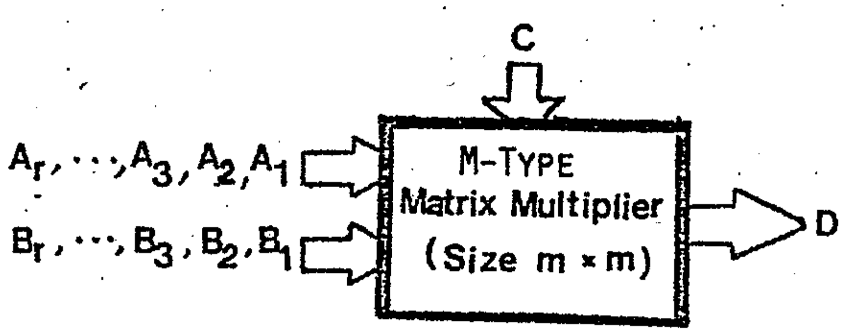
Fig. 2 Functional specification of I-Type chips for submatrix inversion

output arithmetic specifications of D-Type and I-Type chips are shown in Figs.1-2. Both chip types have a fixed delay of 2m time units, where one time unit equals the time required to perform one multiply-add operation, $a \times b + c = d$, or one divide operation, $a/b = c$, by one step processor in the cellular processor array [5,7,10].

The M-Type is the predominant chip type to be used in the construction of various matrix solvers. Accumulative chain matrix multiplications are performed by a M-Type chip as specified in Fig.3. The number, r, of pairs of $m \times m$ matrices to be multiplied and added is determined by the external input sequence. Therefore, the time delay of M-Type chips is equal to $r \cdot m + 1$. The V-Type (Fig.4) chips are deduced from M-Type chips. V-Type performs the accumulative submatrix-vector multiplications. The delay of V-Type chip is also measured as $r \cdot m + 1$. Because each D-Type, I-Type or M-Type VLSI chip contains an array of $m \times m$ step processors [5,7, 10], we consider their interior chip complexity as $O(m^2)$. Each V-Type chip contains a pipeline of m step processors and thus has an interior chip complexity of $O(m)$. The time delays of D-Type and I-Type chips have order $O(m)$ and those for M-Type and V-Type chips are $O(m \cdot r)$, depending also on the number of input pairs.

IV. PARTITIONED L-U DECOMPOSITION

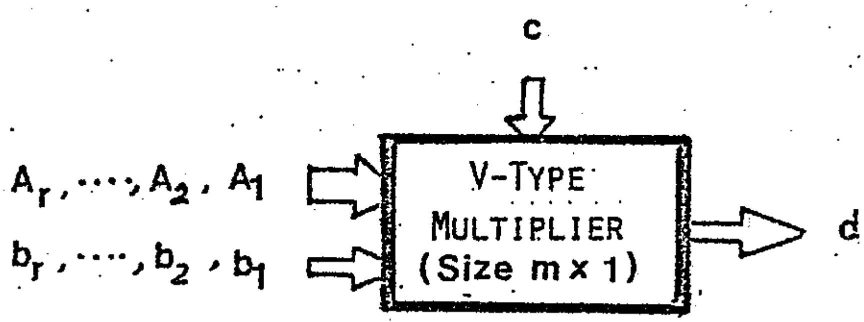
A systolic array of n^2 step processors can perform the L-U decomposition in 4n time units [10]. However, such a systolic array in a single chip may require $4n \times w$ input/output terminals,



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

where C, D, $\{A_i$ and B_i for $i = 1,2,\ldots,r\}$ are $m \times m$ matrices.

Fig.3 Functional specification of M-Type VLSI chips for submatrix multiplication



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

where c, d, and $\{b_i$ for $i = 1, 2, \ldots, r\}$ are $m \times 1$ column vectors, and $\{A_i$ for $i = 1, 2, \ldots, r\}$ are

Fig. 4 Functional specification of V-Type VLSI arithmetic chips where w is the length of matrix elements. For large n (say $n \geq 1000$) with typical operand length w = 32 bits, it is rather impractical to fabricate an $n \times n$ systolic array in a monolithic chip with over $4n \times w = 128,000$ I/O terminals. Our partitioned approach will circumvent this problem by using $m \times m$ VLSI array modules, where m is much smaller than n in at least two orders of magnitude. Of course, I/O port sharing and time-division multiplexing are often used to satisfy the IC packaging constraints, even for small m. [5].

The partitioning method to perform triangular decomposition is illustrated in Fig.5. The given matrix $\underline{A} = (a_{ij})$ is partitioned into k^2 submatrices of order $m \times m$ each. The submatrix computation sequence is also marked. This method is equivalent to Crout's method, when m = 1. However, we assume $m \geq 2$ in general. This sequence of submatrix computations can be best illustrated by partitioning an example matrix $\underline{A}$ of order n = 6 using size m = 2 VLSI chips. Here the ratio k = n/m = 6/2 = 3. (Fig.6) In total, $k \times (k + 1) = 3 \times 4 = 12$ submatrices in $\underline{L}$ and $\underline{U}$ are to be generated for $\underline{L} \cdot \underline{U} = \underline{A}$.

At step 1, we perform L-U decomposition of submatrix A_{11} using a D-Type chip to generate two triangular submatrices L_{11} and U_{11} such that $A_{11} = L_{11} \cdot U_{11}$. Two I-Type VLSI chips are then used to compute the inverse submatrices L_{11}^{-1} and U_{11}^{-1} at step 2. The following matrix multiplications are then performed by 2(k - 1) M-Type chips in parallel.

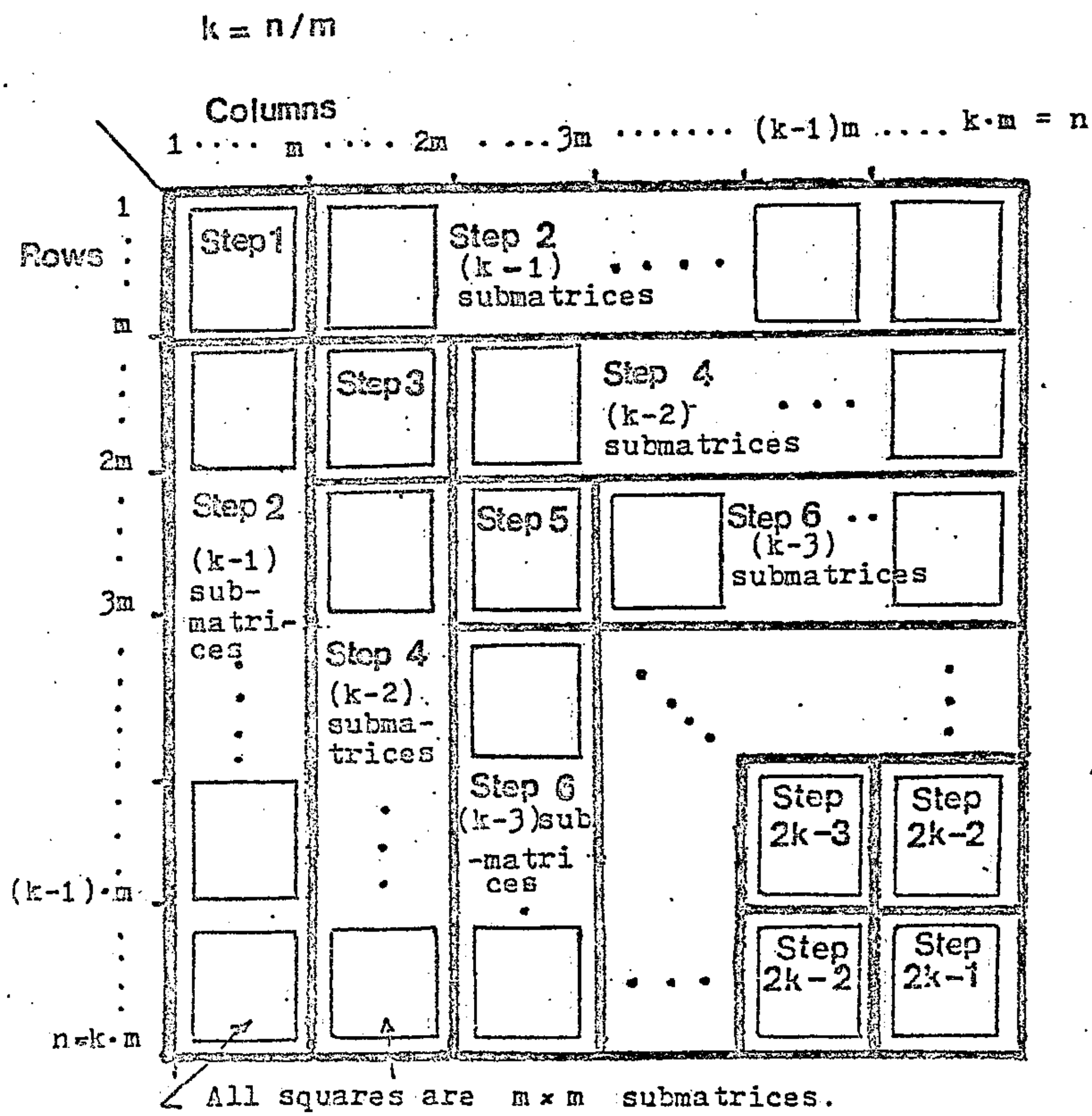
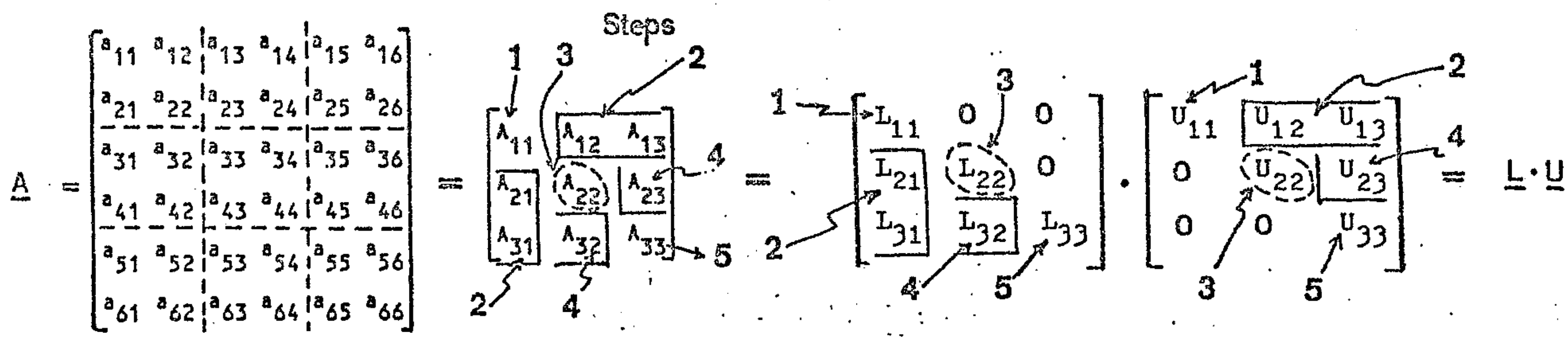


Fig. 5 Partitioning sequence for L-U decomposition of a nonsingular n x n matrix into m x m submatrices in 2k-1 steps, where k = n/m.



$$
\underline{L}\cdot\underline{U} = \left[\begin{array}{cc|cc|cc}
1 & 0 & 0 & 0 & 0 & 0 \\
\ell_{21} & 1 & 0 & 0 & 0 & 0 \\
\hline
\ell_{31} & \ell_{32} & 1 & 0 & 0 & 0 \\
\ell_{41} & \ell_{42} & \ell_{43} & 1 & 0 & 0 \\
\hline
\ell_{51} & \ell_{52} & \ell_{53} & \ell_{54} & 1 & 0 \\
\ell_{61} & \ell_{62} & \ell_{63} & \ell_{64} & \ell_{65} & 1
\end{array}\right] \cdot \left[\begin{array}{cc|cc|cc}
u_{11} & u_{12} & u_{13} & u_{14} & u_{15} & u_{16} \\
0 & u_{22} & u_{23} & u_{24} & u_{25} & u_{26} \\
\hline
0 & 0 & u_{33} & u_{34} & u_{35} & u_{36} \\
0 & 0 & 0 & u_{44} & u_{45} & u_{46} \\
\hline
0 & 0 & 0 & 0 & u_{55} & u_{56} \\
0 & 0 & 0 & 0 & 0 & u_{66}
\end{array}\right]
$$

Step 1. $A_{11} = L_{11} \cdot U_{11}$

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

$U_{12} = L_{11}^{-1} \cdot A_{12}$; $U_{13} = L_{11}^{-1} \cdot A_{13}$

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

$\hat{A}_{23} = A_{23} - L_{21} \cdot U_{13}$; $\hat{A}_{32} = A_{32} - L_{31} \cdot U_{12}$

Step 4. $U_{23} = L_{22}^{-1} \cdot \hat{A}_{23}$; $L_{32} = \hat{A}_{32} \cdot U_{22}^{-1}$

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

Fig.6 Partitioned L-U decomposition of an example matrix of order n = 6 with 2 x 2 (m = 2) VLSI chips in 2k-1 = 2 · 3 - 1 = 5 steps.

$$\begin{cases} L_{p1} = A_{p1} \cdot U_{11}^{-1} & \text{for } p=2,3,\cdots,k \\ U_{1q} = L_{11}^{-1} \cdot A_{1q} & \text{for } q=2,3,\cdots,k \end{cases} \tag{4}$$

For the example 6 × 6 matrix, submatrices L_{21}, L_{31}, U_{12} and U_{13} are generated at step 2 as shown in Fig.6 .

In subsequent steps, we need to generate the following intermediate submatrices using M-Type chips.

$$\hat{A}_{pq} = A_{pq} - \sum_{s=1}^{r-1} L_{ps} \cdot U_{sq} \tag{5}$$

for $p,q=2,3,\cdots,k$.

Local L-U decompositions are then performed on $\hat{A}_{rr}$ at successive odd-numbered steps as shown in Fig.5.

$$L_{rr} \cdot U_{rr} = \hat{A}_{rr} \quad \text{for } r=2,3,\cdots,k \tag{6}$$

The remaining off-diagonal submatrices L_{pr} and U_{rq} are computed by inverting the diagonal submatrices U_{rr} and L_{rr} and then multiplying them by intermediate submatrices $\hat{A}_{pr}$ and $\hat{A}_{rq}$ at successive even-numbered steps. For $r = 2,3,\ldots, k$, we need to compute

$$\begin{cases} L_{pr} = \hat{A}_{pr} \cdot U_{rr}^{-1} & \text{for } p=r+1,\cdots,k \\ U_{rq} = L_{rr}^{-1} \cdot \hat{A}_{rq} & \text{for } q=r+1,\cdots,k \end{cases} \tag{7}$$

For the example 6 × 6 matrix in Fig.6. The intermediate matrix $\hat{A}_{22}$ is computed at step 3. By performing $L_{22} \cdot U_{22} = \hat{A}_{22}$, we obtain two triangular submatrices L_{22} and U_{22} , $\hat{A}_{23}$ and $\hat{A}_{32}$. Inverting these submatrices and then multiplying them to $\hat{A}_{23}$ and $\hat{A}_{32}$, we obtain additional submatrices $U_{23} = L_{22}^{-1} \bullet A_{23}$ and $L_{32} = A_{32} \bullet U_{22}^{-1}$ at step 4. Intermediate submatrix $\hat{A}_{33} = A_{33} - (L_{31} \bullet U_{13} + L_{32} \bullet U_{23})$ is calculated at step 5. Performing L-U decomposition on $\hat{A}_{33}$, we obtain the last two submatrices L_{33} and U_{33} at step 5.

The above interative procedures are summarized in Algorithm 1 for partitioned L-U decompositionof any nonsingular dense matrix $\underline{A}$ of order n. Submatrix computations are specified in groups in Eqs.4,5 and 7. Each group can be computed in parallel within each step by multiple VLSI chips. Submatrix computations can be also computed in sequential order, if only limited number of VLSI chips are available. We shall analyze the hardware chip counts and speed performance of various matrix algorithms in Section VI.

V. PARTITIONED MATRIX INVERSION AND BACK SUBSTITUTION

Partitioned algorithms are developed below for iterative inversion of an n × n nonsingular triangular matrix using I-Type and M-Type VLSI chips. For clarity, we demonstrate the partitioning method by finding the inverse of an example 6 × 6 upper triangular matrix, $\underline{U} = (u_{ij})$ with 2 × 2 array modules (for n = 6 and m = 2). The inverse matrix $\underline{V} = (v_{ij}) = \underline{U}^{-1}$ is partitioned into $k^2 = 9$ submatrices as shown in Fig.7.

First, we perform the inversions of all diagonal submatrices to generate $V_{pp} = U_{pp}^{-1}$ for p = 1,2,...,k. Such inversions are

ALGORITHM 1 (Partitioned L-U Docomposition)

Inputs:

An $n \times n$ dense matrix $\underline{A}=(a_{ij})$ partitioned into k^2 $m \times m$ submatrices A_{ij} for $i,j=1,2,\cdots,k$, where $n=k \cdot m$.

Outputs:

$k \cdot (k+1)$ submatrices L_{pq} for $q \leq p=1,2,\cdots,k$ and U_{rs} for $s \geq r=1,2,\cdots,k$, each of order $m \times m$.

Procedures:

(1) Decompose A_{11} into L_{11} and U_{11} such that $L_{11} \cdot U_{11} = A_{11}$.

(2) Compute inverse matrices L_{11}^{-1} and U_{11}^{-1},

Compute $L_{p1} = A_{p1} \cdot U_{11}^{-1}$ $U_{1p} = L_{11}^{-1} \cdot A_{1p}$ for $p=2,3,\cdots,k$.

(3) For $q \leftarrow 2$ to $(k-1)$ step 1 do

Compute $\hat{A}_{qq} = A_{qq} - \sum_{s=1}^{q-1} L_{qs} \cdot U_{sq}$;

Decompose $\hat{A}_{qq} = L_{qq} \cdot U_{qq}$;

Compute the matrices L_{qq}^{-1} and U_{qq}^{-1}.

For $p \leftarrow (q+1)$ to k step 1 do

Compute $\hat{A}_{pq} = A_{pq} - \sum_{s=1}^{r-1} L_{ps} \cdot U_{sq}$

and $\hat{A}_{qp} = A_{qp} - \sum_{s=1}^{r-1} L_{qs} \cdot U_{sp}$ for $r=\min(p,q)$;

Compute $L_{pq} = \hat{A}_{pq} \cdot U_{qq}^{-1}$; $U_{qp} = L_{qq}^{-1} \cdot \hat{A}_{qp}$.

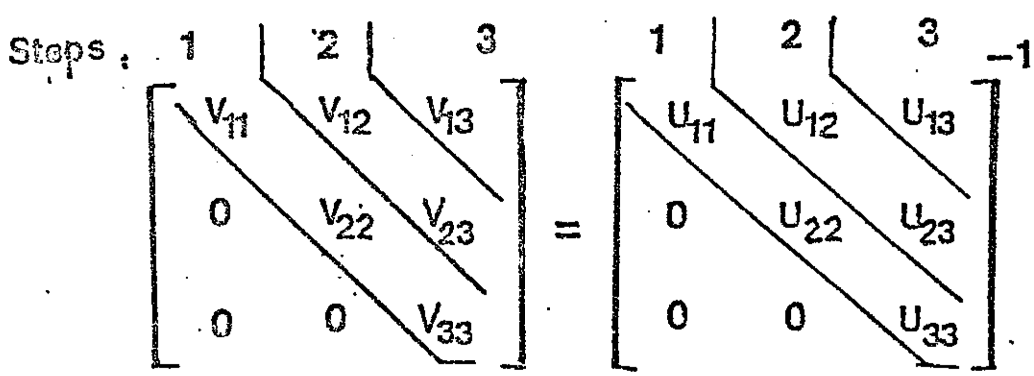
Repeat

Repeat

(4) Compute $\hat{A}_{kk} = A_{kk} - \sum_{s=1}^{k-1} L_{ks} \cdot U_{sk}$;

Decompose $\hat{A}_{kk} = L_{kk} \cdot U_{kk}$

$$\underline{V} = \underline{U}^{-1} = \begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} & u_{15} & u_{16} \\ 0 & u_{22} & u_{23} & u_{24} & u_{25} & u_{26} \\ 0 & 0 & u_{33} & u_{34} & u_{35} & u_{36} \\ 0 & 0 & 0 & u_{44} & u_{45} & u_{46} \\ 0 & 0 & 0 & 0 & u_{55} & u_{56} \\ 0 & 0 & 0 & 0 & 0 & u_{66} \end{bmatrix}^{-1} = \begin{bmatrix} v_{11} & v_{12} & v_{13} & v_{14} & v_{15} & v_{16} \\ 0 & v_{22} & v_{23} & v_{24} & v_{25} & v_{26} \\ 0 & 0 & v_{33} & v_{34} & v_{35} & v_{36} \\ 0 & 0 & 0 & v_{44} & v_{45} & v_{46} \\ 0 & 0 & 0 & 0 & v_{55} & v_{56} \\ 0 & 0 & 0 & 0 & 0 & v_{66} \end{bmatrix}$$



Step 1 : $V_{11} = U_{11}^{-1}$; $V_{22} = U_{22}^{-1}$; $V_{33} = U_{33}^{-1}$

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

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

Fig.7 Partitioned matrix inversion of an example matrix of order $n = 6$ with $m = 2$ submatrix chips in $k = n/m = 6/2 = 3$ steps.

always possible due to the nonsingularity of matrix $\underline{U}$. It follows that k - 1 submatrices in the first off-diagonal are computed at step 2. For the example system in Fig.7, we have to compute V_{12} and V_{23} at step 2, and V_{13} at step 3. These recursive steps for generating the inverse matrix $\underline{V} = \underline{U}^{-1}$ are summarized in Algorithm 2. The computations in Eq.8 can be also performed by M-Type chips. Because of the two-level looping, this algorithm require the same orders of chip count and speed complexity as those for Algorithm 1.

Partitioned multiplication of two large $n \times n$ matrices, say $\underline{A} \cdot \underline{B} = \underline{C}$, is rather straightforward. We include it here for completeness. Basically, each $m \times m$ submatrix C_{pq} of matrix $\underline{C}$ is obtained by performing the cumulative multiplications specified in Eq.9 using exclusively M-Type chips. Partitioned matrix multiplication is specified in Algorithm 3.

Back substitution for solving $\underline{U} \cdot \underline{x} = \underline{d}$ was specified in Eq.3. The method can be partitioned into sections of m-element subvectors. Figure 8 presents the partitioned solution of $\underline{U} \cdot \underline{x} = \underline{d}$ with known $\underline{U} = (u_{ij})$ and $\underline{d} = (d_1, d_2, \ldots, d_6)^T$. Subvectors $[x_5, x_6]^T$, $[x_3, x_4]^T$ and $[x_1, x_2]^T$ are computed sequentially with back substitution. In general, k = n/m steps are needed in the back substitution. Matrix $\underline{U}$ is partitioned into $k \times (k + 1)/2$ submatrices of order $m \times m$. The solution vector $\underline{x}$ is divided into k subvectors and so is the transformed vector $\underline{d}$. Algorithm 4 summarizes the partitioned back-substitution operations. The computations of $\underline{d}_p$ in Eq.10 are carried out by V-Type chips. Boundary conditions $U_{p,k+1} = \underline{0}$ and $\underline{x}_{k+1} = \underline{0}$ were assumed.

ALGORITHM 2 (Partitioned Matrix Inversion)

Inputs:

$m \times m$ submatrices U_{pq} of matrix $\underline{U}=(u_{ij})$ for all $q \geq p=1,2,\cdots,k$.

Outputs:

$k \cdot (k+1)/2$ submatrices V_{pq} of the inverse matrix $\underline{V}=\underline{U}^{-1}$ for all $q \geq p=1,2,\cdots,k$, each of order $m \times m$.

Procedures:

1. For $p \leftarrow 1$ to k step 1 do

$$V_{pp} = U_{pp}^{-1}$$

   Repeat

2. For $q \leftarrow 1$ to $(k-1)$ step 1 do

   For $p \leftarrow 1$ to $k-q$ step 1 do

$$W_{p,p+q} = \sum_{r=1}^{q} U_{p,p+r} \cdot V_{p+r,p+q} \; ; \qquad (8)$$

$$V_{p,p+q} = - V_{pp} \cdot W_{p,p+q} \; .$$

   Repeat

   Repeat

ALGORITHM 3 (Partitioned Matrix Multiplication)

Inputs:

$m \times m$ submatrices A_{pr} and B_{rq} of the matrix $\underline{A}=(a_{ij})$ and matrix $\underline{B}=(b_{ij})$, where $p,q,r=1,2,\cdots,k$.

Outputs:

$m \times m$ submatrices C_{pq} of the resulting product matrix $\underline{C}=(c_{ij})$, where $p,q=1,2,\cdots,k$.

Procedures:

For $p \leftarrow 1$ to k step 1 do

For $q \leftarrow 1$ to k step 1 do

$$C_{pq} = \sum_{r=1}^{k} A_{pr} \cdot B_{rq} \tag{9}$$

Repeat

Repeat

$$\left[\begin{array}{cc:cc:cc} u_{11} & u_{12} & u_{13} & u_{14} & u_{15} & u_{16} \\ 0 & u_{22} & u_{23} & u_{24} & u_{25} & u_{26} \\ \hdashline & & u_{33} & u_{34} & u_{35} & u_{36} \\ \multicolumn{2}{c:}{O} & 0 & u_{44} & u_{45} & u_{46} \\ \hdashline & & & & u_{55} & u_{56} \\ \multicolumn{2}{c:}{O} & \multicolumn{2}{c:}{O} & 0 & u_{66} \end{array}\right] \cdot \left[\begin{array}{c} x_1 \\ x_2 \\ \hdashline x_3 \\ x_4 \\ \hdashline x_5 \\ x_6 \end{array}\right] = \left[\begin{array}{c} d_1 \\ d_2 \\ \hdashline d_3 \\ d_4 \\ \hdashline d_5 \\ d_6 \end{array}\right]$$

$$\begin{bmatrix} x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} u_{55} & u_{56} \\ 0 & u_{66} \end{bmatrix}^{-1} \cdot \begin{bmatrix} d_5 \\ d_6 \end{bmatrix} \qquad \text{(\underline{Step 1})}$$

(Step 2)

$$\begin{bmatrix} x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} u_{33} & u_{34} \\ 0 & u_{44} \end{bmatrix}^{-1} \cdot \left(\begin{bmatrix} d_3 \\ d_4 \end{bmatrix} - \begin{bmatrix} u_{35} & u_{36} \\ u_{45} & u_{46} \end{bmatrix} \cdot \begin{bmatrix} x_5 \\ x_6 \end{bmatrix} \right)$$

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} u_{11} & u_{12} \\ 0 & u_{22} \end{bmatrix}^{-1} \cdot \left\{ \begin{bmatrix} d_1 \\ d_2 \end{bmatrix} - \left(\begin{bmatrix} u_{13} & u_{14} \\ u_{23} & u_{24} \end{bmatrix} \cdot \begin{bmatrix} x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} u_{15} & u_{16} \\ u_{25} & u_{26} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \right) \right\}$$

(Step 3)

Fig. 8 Partitioned VLSI triangular system solver based on subvector back-substitution

<u>ALGORITHM 4</u> (Partitioned Triangular System Solver)

<u>Inputs:</u>

$m \times m$ submatrices U_{pq} of $\underline{U}$ for $q \geq p=1,2,\cdots,k$. The coefficient subvectors $\underline{d}_p$ for $p=1,2,\cdots,k$, each having m consecutive elements of the vector $\underline{d}$.

<u>Outputs:</u>

The subvector $\underline{x}_p$ of the solution vector $\underline{x}=[x_1,x_2,\cdots,x_n]^T$, where $\underline{x}_p=[x_{(p-1)m+1},x_{(p-1)m+2},\cdots,x_{pm}]^T$ for $p=1,2,\cdots,k$.

<u>Procedures:</u>

<u>For</u> $p \leftarrow k$ <u>to</u> 1 in <u>step</u> (-1) <u>Compute</u>

U_{pp}^{-1} from U_{pp} ;

$$\hat{\underline{d}}_p = \underline{d}_p - \sum_{q=p+1}^{k} U_{pq} \cdot \underline{x}_q \; ; \qquad (10)$$

$$\underline{x}_p = U_{pp}^{-1} \cdot \hat{\underline{d}}_p \, .$$

<u>Repeat</u>

VI. VLSI ARCHITECTURES AND PERFORMANCE ANALYSIS

VLSI chip requirements and speed complexity of partitioned matrix algorithms are analyzed in this section. We consider two architectural configurations for the proposed VLSI matrix algorithms. In a strictly parallel configuration, all submatrix operations at each step are performed in parallel by multiple VLSI chips; and thus results in a minimum time delay per each step. The total time delay among all steps is also minimized by overlapping some step operations. In a serial-parallel configuration the number of available VLSI chips in each step in upper bounded. Thus, some parallel-executable operations may have to be executed sequentially. Of course, serial-parallel operations will result in longer time delays due to limited hardware.

To implement Algorithm 1 in hardware, we need to use one D-Type two I-Type, and a large number of M-Type VLSI chips. The number of needed M-Type chips depends on the chosen architectural configuration. In Table 1, we traced the step-by-step operations of Algorithm 1 with a minimum-delay analysis. We have structured the algorithm for minimum data dependency in successive steps. In other words, some adjacent computation steps are overlapped in a lookahead fashion. Maximal concurrencies are achieved by parallelism within each step and overlapping between successive steps. The total time delay of Algorithm 1 implemented in strictly parallel mode is equal to

$$T = 6n + 2n/m - 4m - 2 \doteq O(n), \text{ if } n \gg m \gg 1 \quad (11)$$

For large n, M-Type chips predominate the chip requirement of Algorithm 1. We plot in Fig.9 the actual chip count in successive

Table 1. Time and Hardware Complexities of the Partitioned L-U Decomposition (Algorithm 1)

Step	Submatrix Computations	Time Complexity: Start Time	Time Complexity: Delay	VLSI Chip Count: D-Type	VLSI Chip Count: I-Type	VLSI Chip Count: M-Type
1	$A_{11} = L_{11} \cdot U_{11}$	0	2m	1		
2	L_{11}^{-1}, U_{11}^{-1}	2m	2m		2	
	$L_{p1} = A_{p1} \cdot U_{11}^{-1}$ $U_{1p} = L_{11}^{-1} \cdot A_{1p}$	4m (for p=2,3,...,k)	$m + 1$			2(k-1)
3_2 (q=2)	$\hat{A}_{22}$	5m + 1	m + 1			1
	$\hat{A}_{22} = L_{22} \cdot U_{22}$	6m + 2	2m	1		
	L_{22}^{-1}, U_{22}^{-1}	8m + 2	2m		2	
	$\hat{A}_{p2}, \hat{A}_{2p}$	9m + 1 (for p=3,4,...,k)	m + 1			2(k-2)
	L_{p2}, U_{2p}	10m + 2 (for p=3,4,...,k)	m + 1			
⋮	⋮	⋮	⋮			⋮
3_{k-1} (q=k-1)	$\hat{A}_{k-1,k-1}$	(5m+2)(k-2)-1	(k-2)m+1			1
	$L_{k-1,k-1}$; $U_{k-1,k-1}$	(6m+2)(k-2)	2m	1		
	$L_{k-1,k-1}^{-1}$; $U_{k-1,k-1}^{-1}$	(6m+2)(k-2)+2m	2m		2	
	$\hat{A}_{k,k-1}$; $\hat{A}_{k-1,k}$	(5m+2)·(k-2) +(4m-1)	(k-2)m+1			
	$L_{k,k-1}$; $U_{k-1,k}$	(6m+2)(k-2)+4m	m + 1			2·1
4	$\hat{A}_{kk}$	(5m+2)(k-1)-1	(k-1)m+1			1
	$\hat{A}_{kk} = L_{kk} \cdot U_{kk}$	(6m+2)(k-1)	2m	1		
Total Compute Time		$T = 6n + \frac{2n}{m} - (4m+2) \doteq O(n)$ for $n >> m >> 1$				
Total VLSI Chip Count $M \doteq O(n^2/m^2)$ for $n >> m >> 1$				1	2	$\frac{1}{11}(n/m)^2$

Note: q is the looping index used in Algorithm 1 and k = n/m.

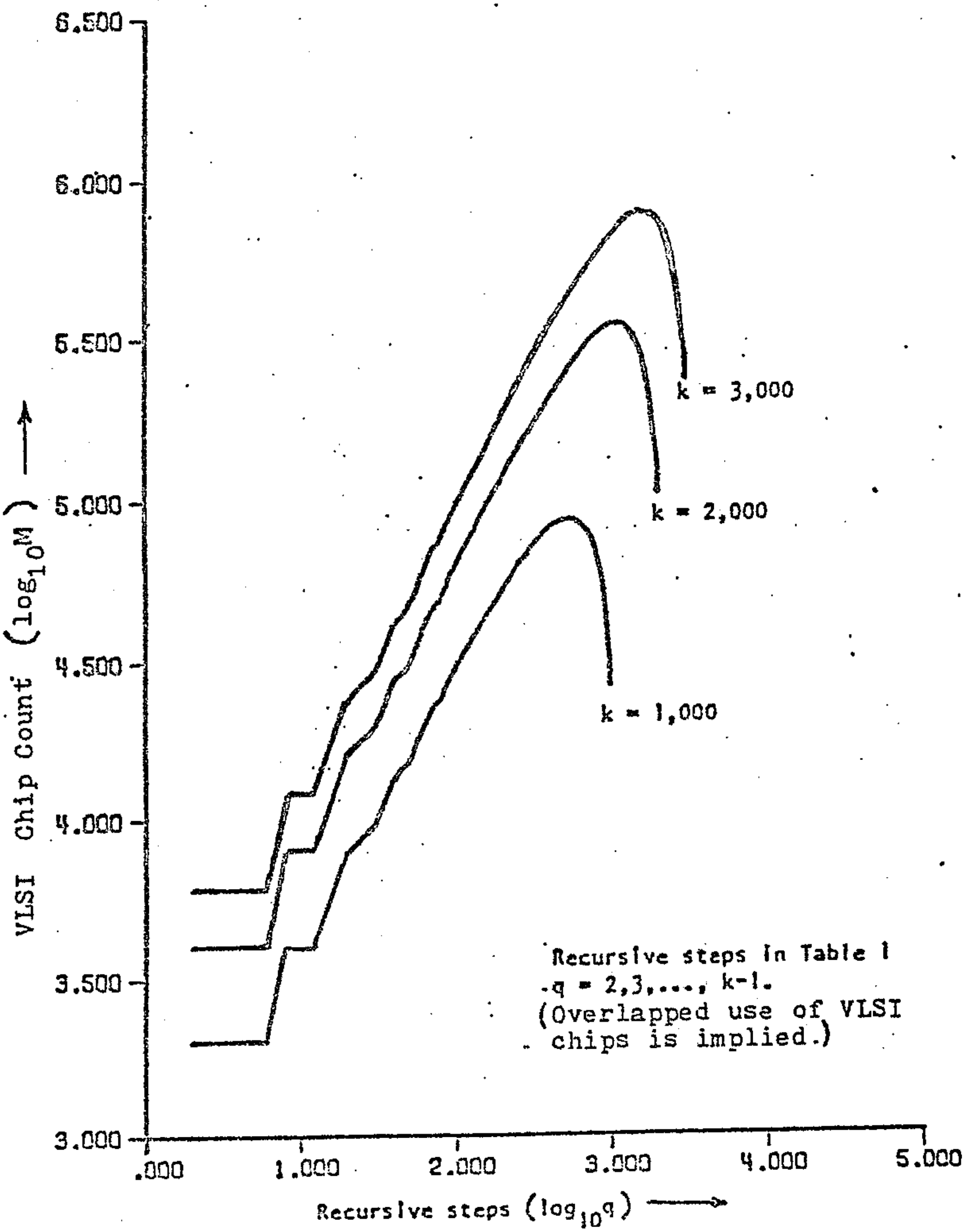


Fig.9 VLSI chip requirements in successive computation steps of Algorithm 1.

steps, 3q for $q = 2,3,\ldots,k-1$, of Algorithm 1. The chip requirement increases steadily until the looping index $q = 13$. The effect of resource sharing between adjacent steps becomes apparent for $q \gtrsim 13$. The peak of each curve corresponds to the minimum number, M, of M-Type chips required to achieve the minimum delay T in Eq.11. This number has been estimated algebraically to be

$$M \doteq n^2/(11m^2) \text{ for } n \gg m \gg 1 \quad (12)$$

From Eqs.11 and 12, we conclude that the partitioned L-U decomposition (Algorithm 1) can be realized with $O(n^2/m^2)$ VLSI chips with interior chip complexity $O(m^2)$.

Using a uniprocessor, $O(n^3)$ time steps are needed to perform the L-U decomposition. It is interesting to note that the triple product of the chip count $O(n^2/m^2)$, the compute time $O(n)$, and the chip size $O(m^2)$ yields the uniprocessor compute time $O(n^3)$; that is

$$O(n^2/m^2) \cdot O(m) \cdot O(m^2) = O(n^3) \quad (13)$$

This property is called conservation law between available hardware chips and achievable speed.

The chip count, $O(n^2/m^2)$, is too high to be of practical value, because of the fact that $n \gg m$. Therefore, we have to bound the chip count with a linear order, $O(n/m)$, in a serial-parallel implementation of the partitioned matrix algorithm. One can use $2n/m - 1$ M-Type chips to implement a serial-parallel architecture for Algorithm 1. Using $O(n/m)$ chips yields the following prolonged time delay for Algorithm 1.

$$T' = n^2/m - n/2 + 2n/m + 17m - 2 \doteq O(n^2/m) \text{ for } n \gg m \gg 1 \quad (14)$$

The conservation law is again preserved in this serial-parallel architecture. In this case, we observe that

$$O(n/m) \cdot O(n^2/m) \cdot O(m^2) = O(n^3) \qquad (15)$$

Similar analyses can be made to estimate the chip counts and time delays for Algorithm 2 and Algorithm 3. As shown in Table 2, both VLSI matrix algorithms can be implemented by $O(n^2/m^2)$ VLSI chips with $O(n)$ time delays for the strictly parallel architecture; and by $O(n/m)$ chips with $O(n^2/m)$ times for the serial-parallel confituration. Only I-Type and M-Type chips are needed in Algorithm 2. Algorithm 3 requires to use only M-Type chips. To solve a triangular LSEs (Algorithm 4), one I-Type chip and $n/2m$ V-Type chips are needed. The total time delay is $O(n)$. Only $O(n/m)$ VLSI chips are used in Algorithm 4. Only the strictly parallel architecture is suggested for constructing the VLSI triangular system solver. Again we have preserved the conservation law. In this case, we observe that

$$O(n/m) \cdot O(n) \cdot O(m) = O(n^2) \qquad (16)$$

where $O(n^2)$ is the compute time of using a uniprocessor to solve a triangular system.

It is obvious that tradeoffs exist between the chip counts and time delays of all partitioned matrix algorithms. The tradeoffs in implementing Algorithm 1 are plotted in Fig. 10. The time delay is a monotonic decreasing function of the chip count. When the chip count exceeds the upper bound M (Eq.12), the minimum time delay is achieved as shown by the flat portion of the curves

Table 2. VLSI Chip Requirements and Speed Performances of Various Partitioned Matrix Algorithms

VLSI Architecture and Complexity / Matrix Algorithm	Strictly Parallel System Architecture with Minimum Time Delays		Serial-Parallel System Architecture with Bounded Chip Count	
	VLSI Chip Count and Types	Total Compute Time	VLSI Chip Count and Types	Total Compute Time
Algorithm 1 for L-U Decomposition	$O(n^2/m^2)$ D, I, M*	$O(n)$	$O(n/m)$ D, I, M*	$O(n^2/m)$
Algorithm 2 for Inversion of Triangular Matrix	$O(n^2/m^2)$ I, M*	$O(n)$	$O(n/m)$ I, M*	$O(n^2/m)$
Algorithm 3 for Matrix Multiplication	$O(n^2/m^2)$ M*	$O(n)$	$O(n/m)$ M*	$O(n^2/m)$
Algorithm 4 for Solving Triangular LSEs	$O(n/m)$ I, V*	$O(n)$		

Note: All measures are based on the assumption $n >> m >> 1$, where n is the matrix order and m is the VLSI chip size.

* Dominating Chip Type to be used.

in Fig.10. By presenting a speed requirement, one can use these curves to decide to minimum number of VLSI chips needed to achieve the desired speed performance. On the other hand, one can predict the speed performance under prespecified hardware allowance. This tradeoff study is necessary for cost-effective design of large-scale matrix system solvers.

VII. VLSI MATRIX ARITHMETIC SOLVERS

Two pipelined matrix solvers are presented below based on the partitioned matrix algorithms. One matrix solver is for the L-U decomposition of an n x n matrix A with m x m VLSI matrix chips. The other is for the inversion of a triangular matrix of order n. Only the serial-parallel architecture is to be presented with $O(n^2/m)$ compute time, $O(n/m)$ chip count, and $O(m^2)$ chip complexity.

A VLSI L-U decomposition pipeline is shown in Figs.11-14 for the case of $k = n \cdot m = 3$. In general, such a pipeline requires to use one D-type, two I-type, and $2k - 1$ M-type submatrix VLSI modules. These VLSI chips are interfaced with high speed latches and feedback connections. Only the snap shops of $2k - 1 = 5$ submatrix steps are shown. During each step, the active chips and data paths are stressed by boldface boxes and data paths.

The matrix inversion algorithm (Fig.7) is realized by the pipeline processor shown in Fig.15 for the case of $k = n/m = 4$. In general, k I-type, and $2(k - 1)$ M-type submatrix multipliers are needed in this serial-parallel implementation of Algorithm 2. The input assignments, data flows at intermediate and output terminals are specified at the attached table for four steps.

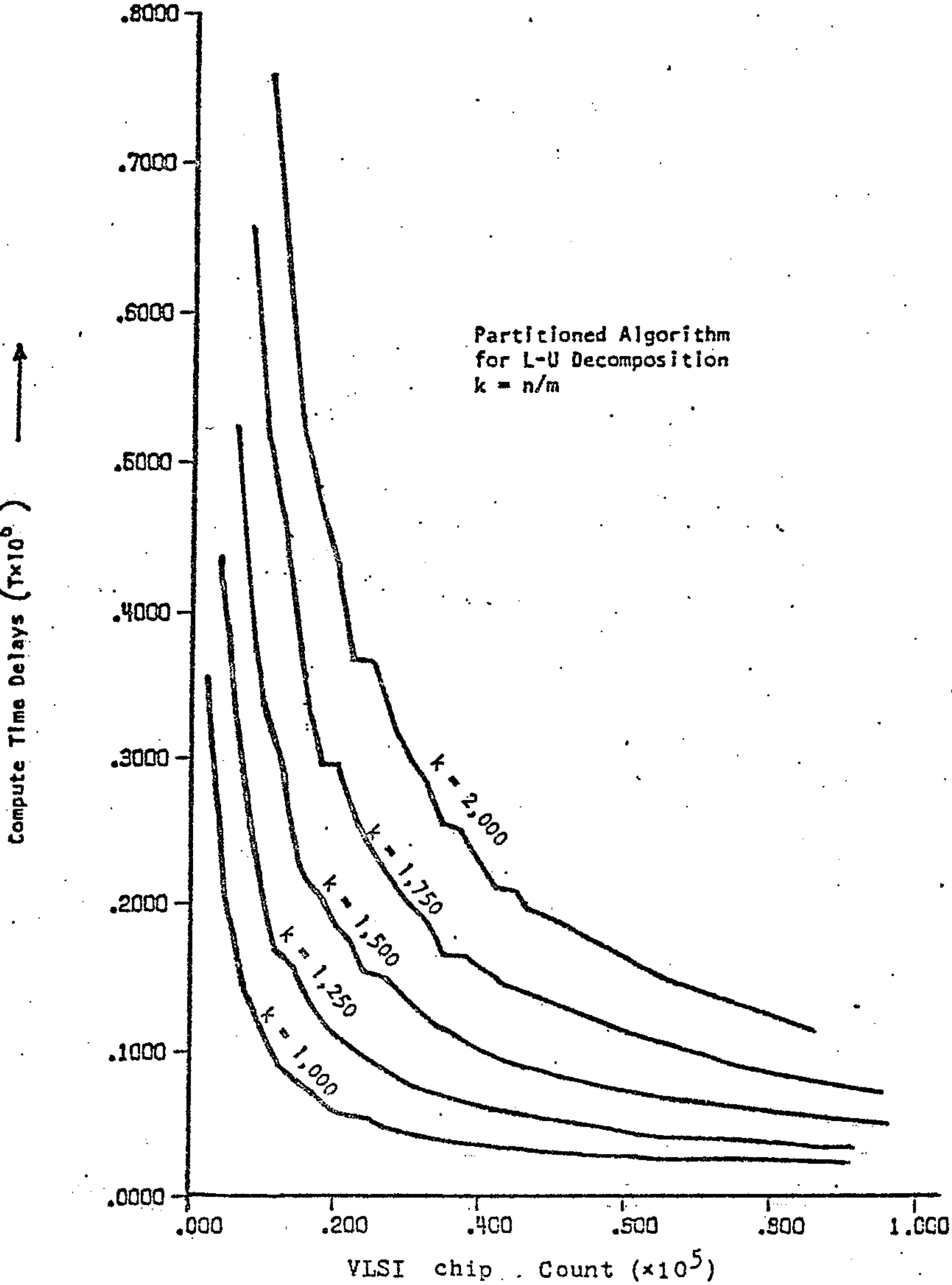
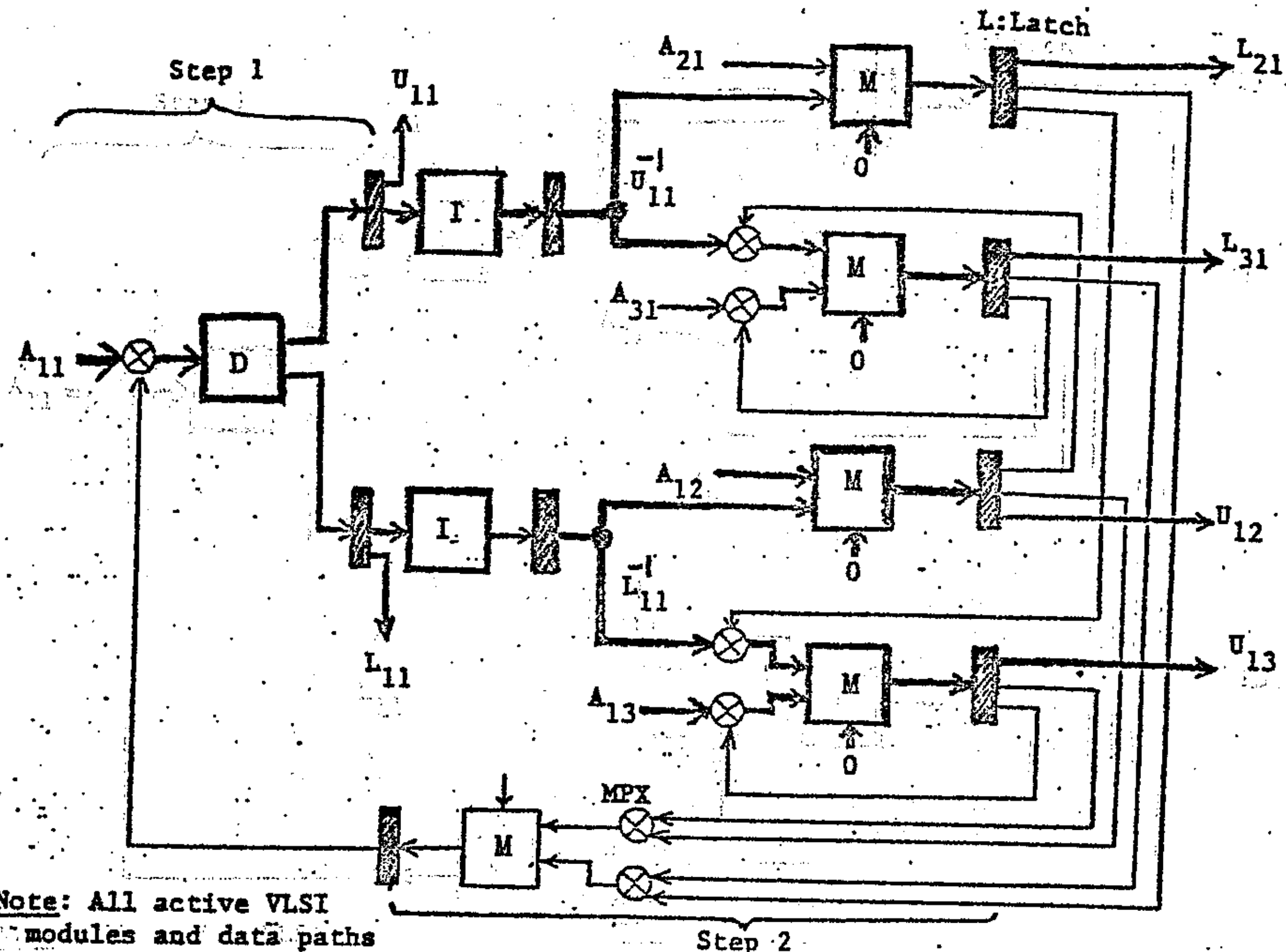


Fig. 10 Compute time delays of Algorithm 1 versus available numbers of VLSI submatrix chips



Note: All active VLSI modules and data paths are darkened.

MPX: Multiplexers.

Fig.11 VLSI pipeline for partitioned L-U decomposition: Steps 1-2 for the example in Fig.6

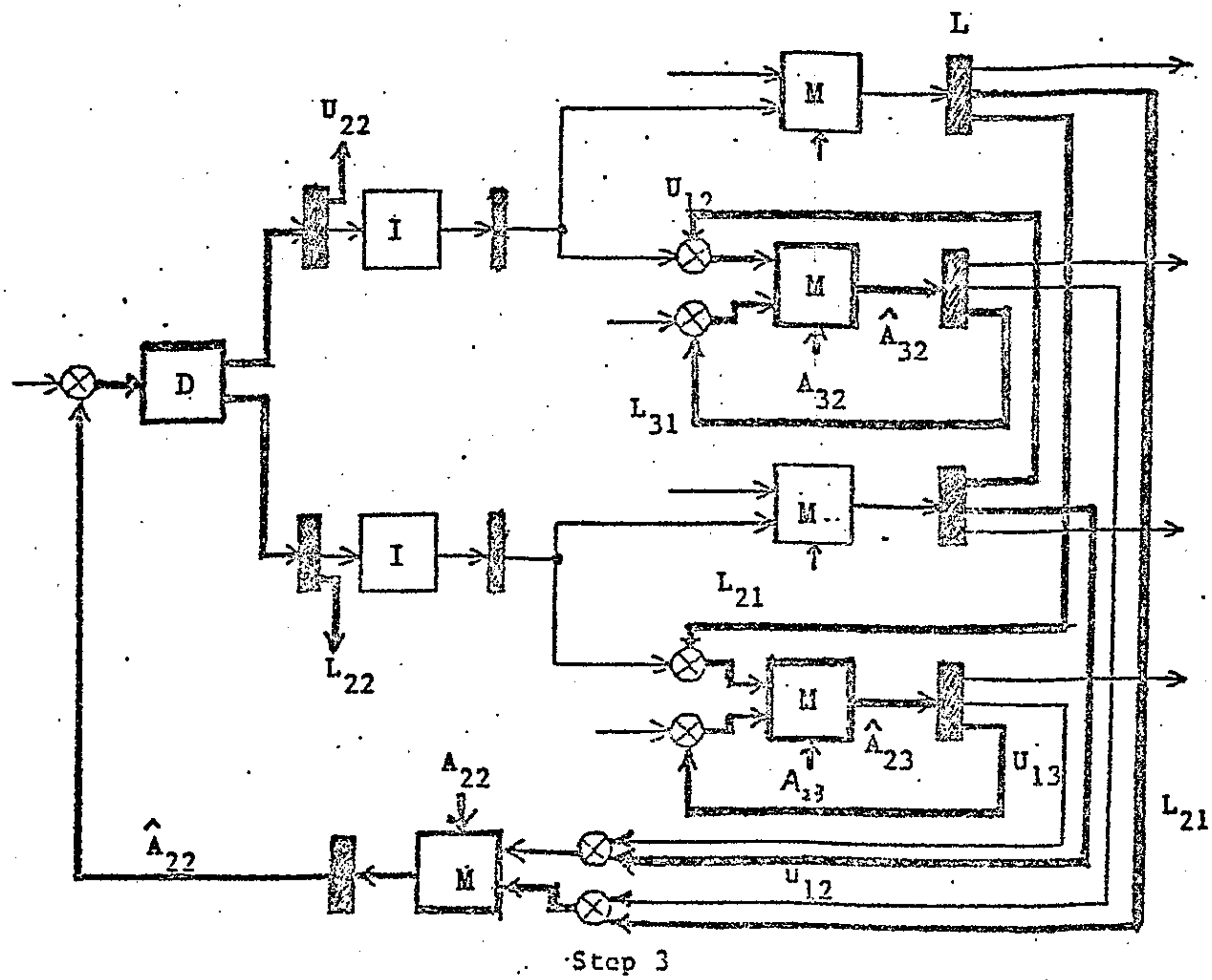


Fig.12 VLSI pipeline for partitioned L-U decomposition: step 3 in Fig.6.

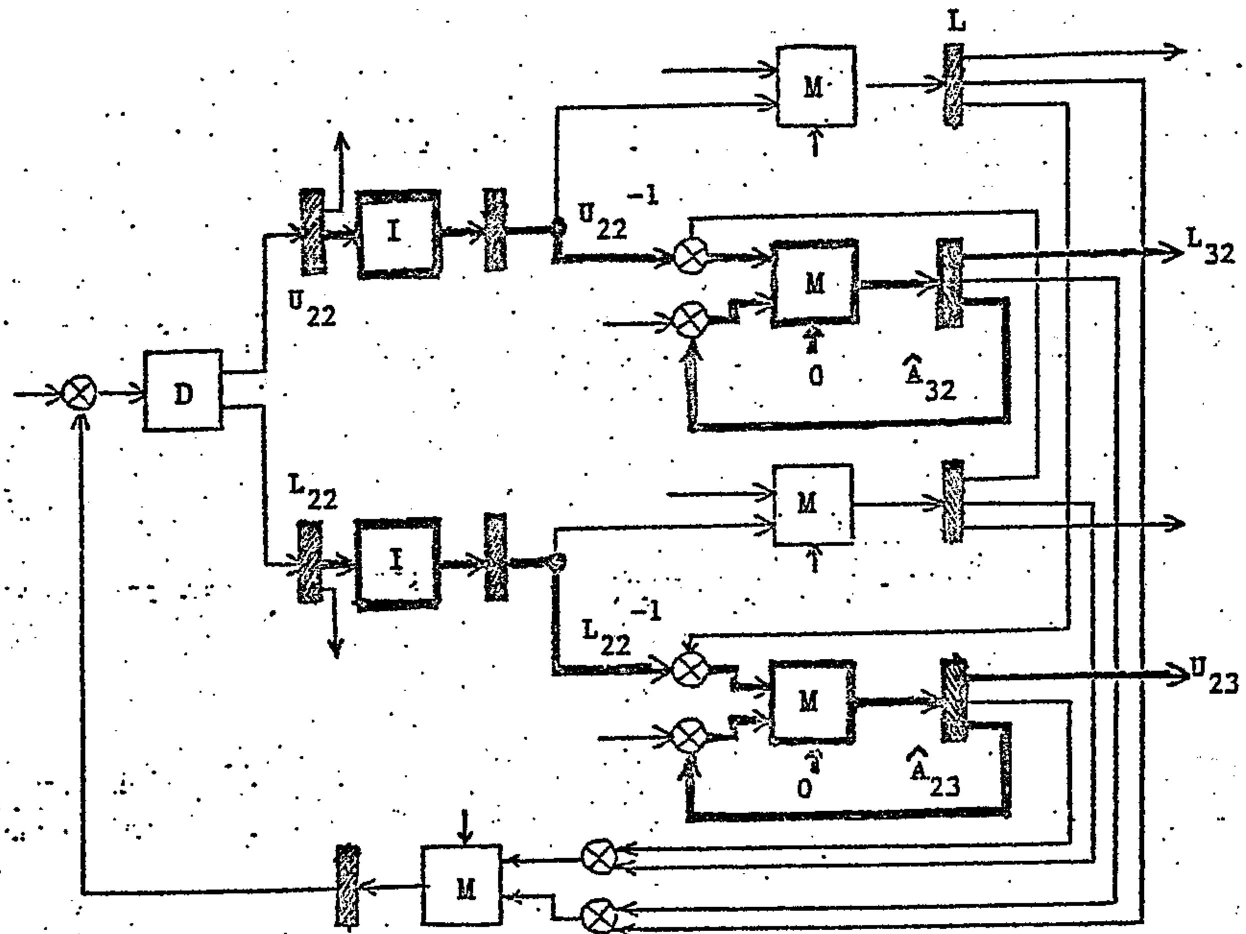


Fig.13 VLSI pipeline for partitioned L-U decomposition: Step 4 in Fig.6.

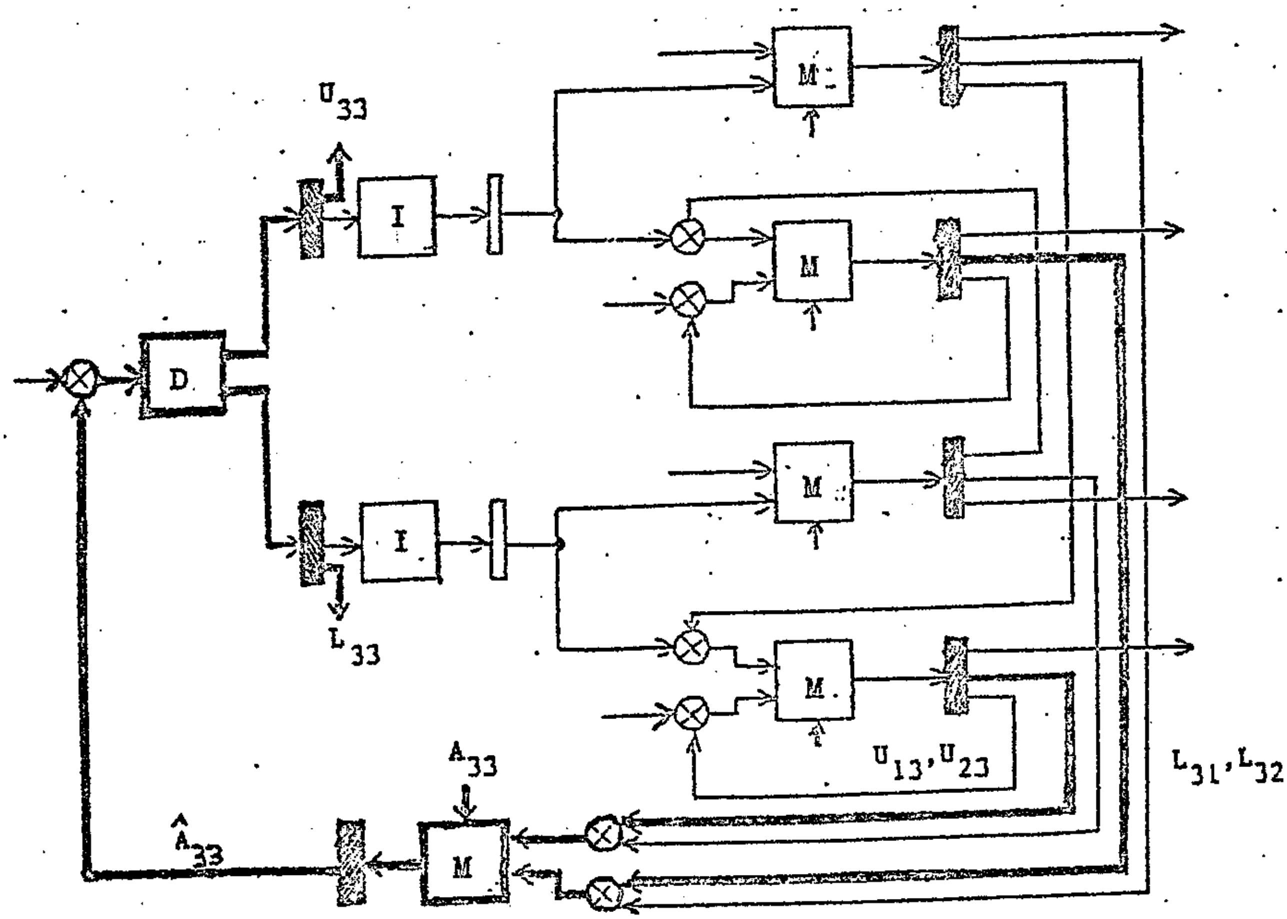
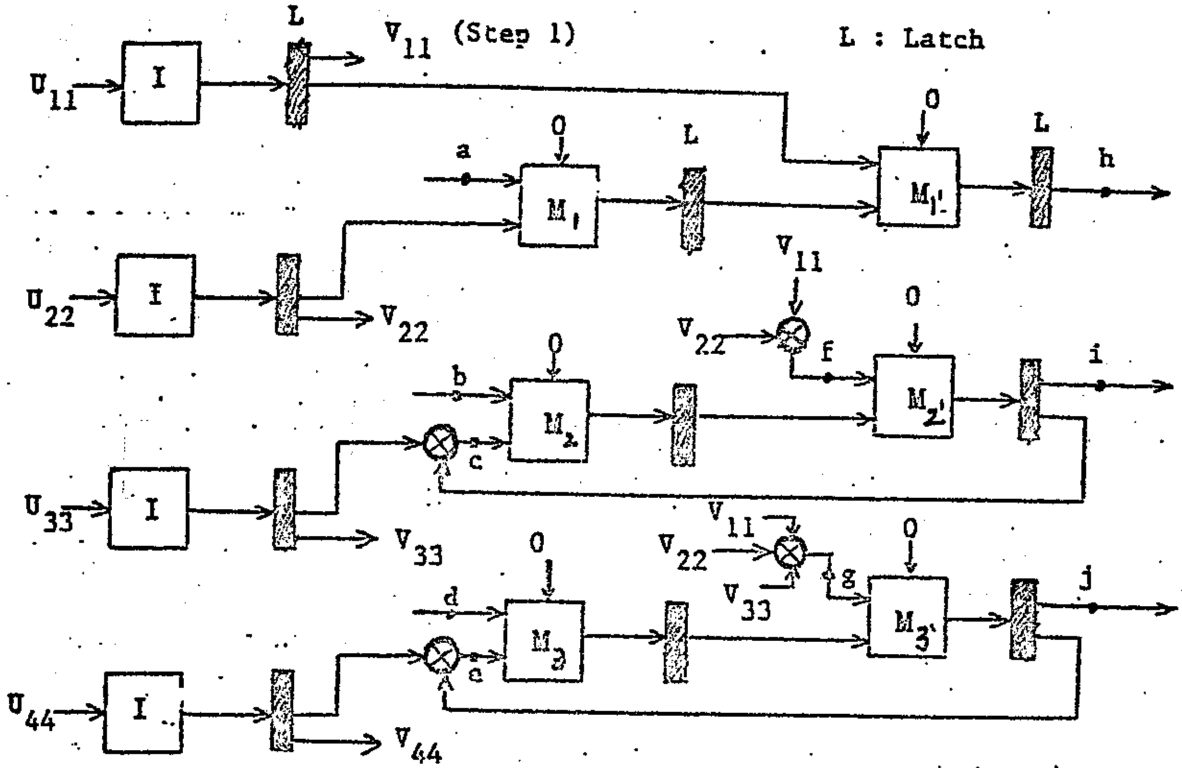


Fig.14 VLSI pipeline for partitioned L-U decomposition: Step 5 in Fig.6.



	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-I modules are active during Step 1.
Modules M_1 and M_1' are active in Step 2.
Modules M_2 and M_2' are active in Steps 2,3.
Modules M_3 and M_3' are active in Steps 2,3,4.

Fig.15 VLSI pipeline for partitioned of an example matrix of order n = 4m.

VLSI matrix solvers have been suggested to implement fast feature extractors and pattern classifiers [19]. Systolic VLSI architectures have been suggested for convolution and resampling [22], for signal/image processing [27], and for other numeric and alphanumerical algorithms [29]. Configurable interconnection networks have been proposed by Synader [31] for designing reconfigurable SIMD array processors or MIMD multiprocessor systems.

VLSI pattern recognizers offer high speed and accuracy which are useful in real-time, on-line, pictorial information processing. This is the first step towards advanced automation and machine intelligence. Recently, many attempts have been made in developing special VLSI devices for signal/image processing and pattern recognition [12,13,14,26,29,40,41,42]. Most of these approaches involve large-scale matrix computations or syntatic parsing operations. We list in Table 3 some candidate PRIP algorithms that might be suitable for VLSI implementation. An example is presented in Fig.16 illustrate the VLSI approach to statistical feature extraction.

The system architecture of an integrated picture processing computer is conceptually illustrated in Fig.17. The system consists of four major subsystems, as shown by the major blocks in the drawing. The host computer can be any one of those existing pattern-analysis computers summarized in [20]. The backend database machine is specially developed for image database management. Either software or hardware approaches can be adopted in developing image database management systems. The front-end communication processor is used to handle terminal activities or to be connected to a computer network for remote users. The shared resource pool contains VLSI functional units or attached special processors for fast PRIP operations as examplified in Table 3. A resource arbitration network is needed between the host processors and the shared resource pool.

Table 3. Pictorial Information Processing Algorithms for Possible VLSI Implementation

Image Processing	Enhancement, Filtering, Fhining, Edge Detection, Segmentation, Registration, Restoration, Clustering, Texture Analysis, Convolution, Fourier Analysis, etc.
Pattern Recognition	Feature Extraction, Template Matching, Statistical Classification, Graph Algorithms, Syntax Analysis, Change Detection, Language Recognition, Scene Analysis and Synthesis, etc.
Image Query Processing	Query Decomposition, Query Optimization, Attribute Manipulation, Picture Reconstruction, Search/Sorting Algorithms, Query-by-Picture-Example Implementation, etc.
Image Database Processing	Relational Operators (JOIN, UNION, INTERSECTION, PROJECTION, COMPLEMENT), Image-Shetch-Relation Conversion, Similarity Retrieval, Data Structures, Priority Queues, Dynamic Programming, Spatial Operators, etc.

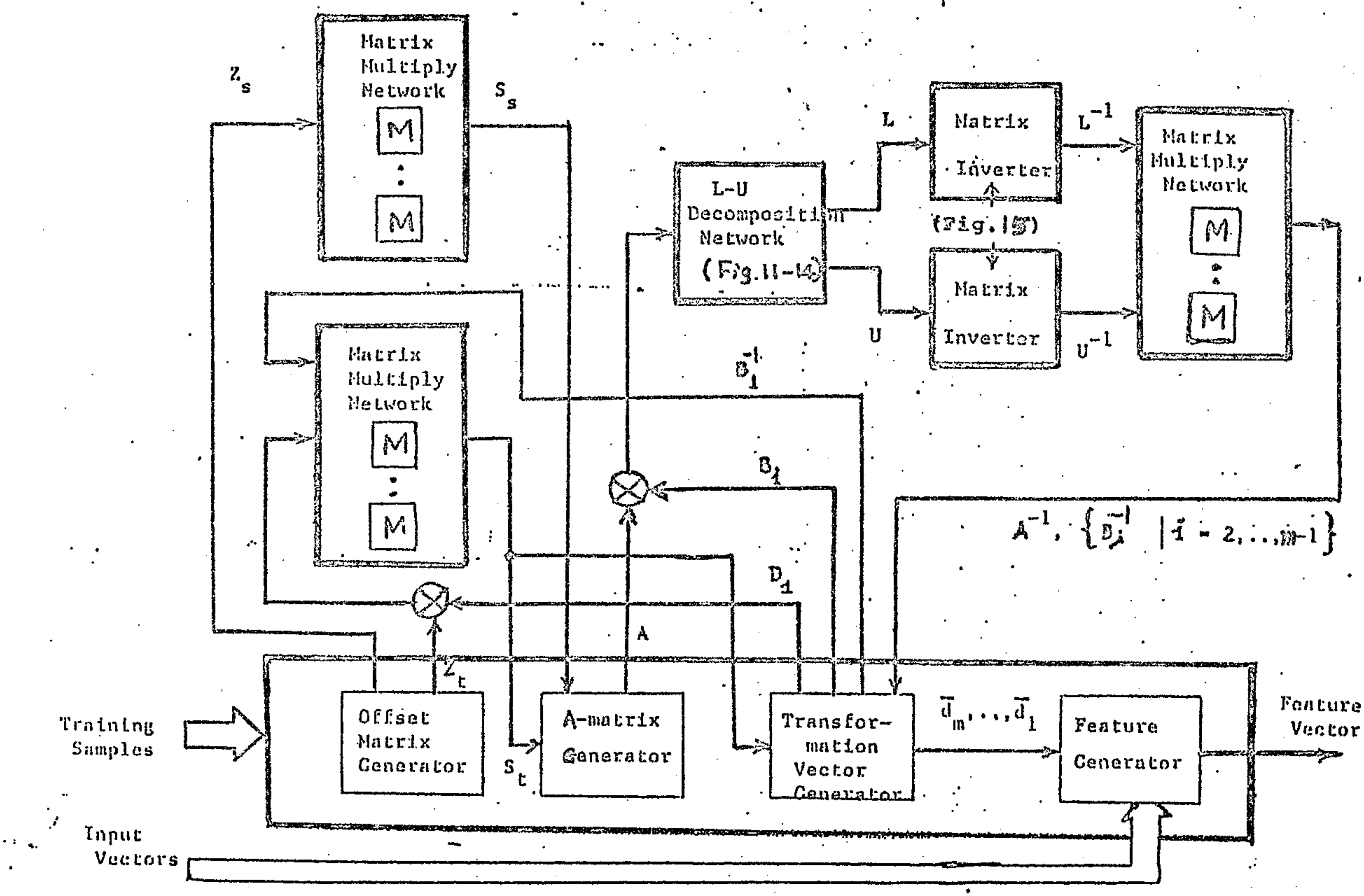


Fig.16 VLSI Matrix manipulation network for statistical feature extraction

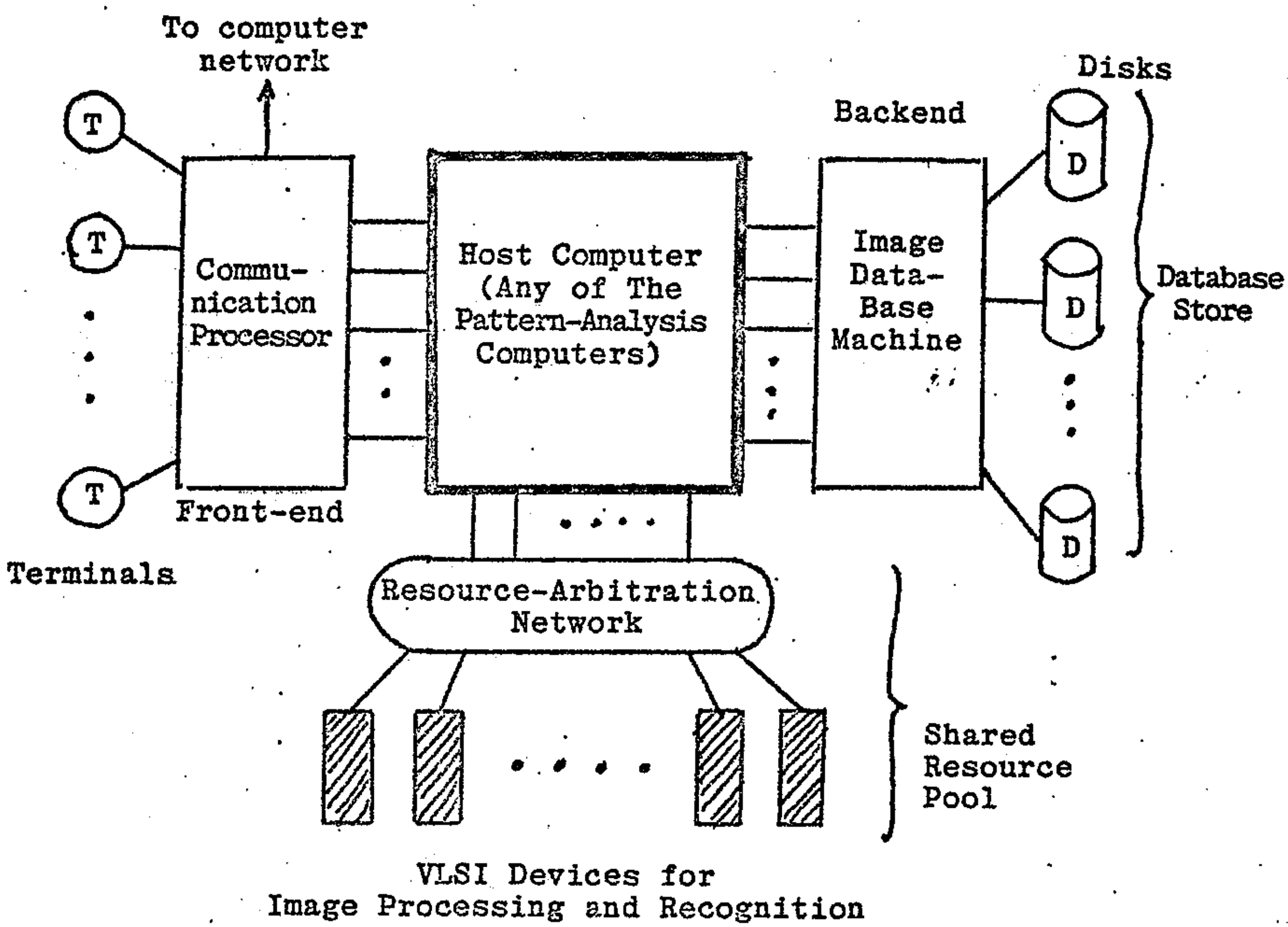


Fig.17 Architecture of the integrated computer system for pattern analysis and image database management

VIII. CONCLUSIONS

Gaussian elimination has been modified in this paper through a block partitioning approach. These partitioned VLSI matrix algorithms can be implemented with $O(n^2/m^2)$ chips in linear time $O(n)$, or with $O(n/m)$ chips in quadratic time $O(n^2/m)$. In either case, we have achieved a significant speedup over the $O(n^3)$ compute time of a serial uniprocessor. M-Type chips are the major type of VLSI chips to be used in implementing matrix solvers. In systolic arrays, the chip complexity is of order $O(n^2)$. In our modular approach, it has been reduced to $O(m^2)$. For $n \gg m$, this implies higher feasibility based on the projected VLSI chip and packaging technologies.

Design tradeoffs of VLSI architectures have to satisfy the conservation law between operating speed and total chip counts. Our partitioned approach offers better extensibility, maintainability, and flexibility to digital system designers. Optimized design must consider high performance/cost ratio. The proposed partitioning matrix algorithms are also suitable for software implementation. The tradeoffs between hardware and software approaches should be an interesting research topic.

Toward the eventual realization of fast VLSI matrix solvers with standard VLSI chips, there are still many practical problems yet to be solved; such as computer-aided layout of VLSI circuits, operand buffering within chips, I/O port sharing and multiplexity, packaging and reliability issues, etc. We firmly believe that partitioning is a logical and feasible approach to designing large-scale matrix manipulation machines. Modulation through algorithmic partitioning is much better off than through unstructured physical circuit partitioning, as far as the systemization of VLSI computing architecture is concerned.

REFERENCES

[1] P. D. Crout, "A Short Method For Evaluating Determinants and Solving Systems of Linear Equations With Real or Complex Coefficients", Proc. of American Inst. of Electrical Engineers, Vol.40, 1941, pp.1235-1240.

[2] D. K. Faddeev and V. N. Faddeeva, Computational Methods of Linear Algebra, translated by R. C. Williams from Russian, W. H. Freeman and Co., San Franxisco, 1963, pp.140-163.

[3] G. Forsythe and C. B. Moler, Computer Solution of Linear Algebraic Systems, Prentice-Hall, Inc., Englewood, N.J. 1967, pp.27-48.

[4] K. Hwang, Computer Arithmetic: Principle, Architecture and Design, John Wiley & Sons, Inc., New York, 1979, Chaps.6 and 8.

[5] K. Hwang and Y. H. Cheng, "Partitioned Algorithms and VLSI Structures for Large-Scale Matrix Compatitions", Proc. of the Fifth Symp. on Computer Arithmetic, IEEE Computer Society, Ann Arbor, Michigan, May 1981, pp.220-230.

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

[7] K. Hwang and S. P. Su, "VLSI Matrix Manipulators for Feature Extraction and Pattern Classification", Technical Report TR-82-001, Institute of Information Science, Academia Sinica, Taipei, Taiwan, Republic of China, March 1982.

[8] E. Isaacson and H. B. Keller, Analysis of Numerical Methods, John Wiley, N. Y. 1966, pp.29-52.

[9] R. M. Kang and T. Kimura, "Decentralized Parallel Algorithms for Matrix Computations, Proc. of Fifth Annual Symp. on Computer Architecture, Palo Alto, Ca., April 1978, pp.96-100.

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

[11] C. Mead and L. Conway, Introduction To VLSI Systems, Addison Wesley Pub. Co., Reading, Mass. 1980, pp.263-332.

[12] J. G. Nash, S. Hansen and G. R. Nudd, "VLSI Processor Array for Matrix Manipulation", in VLSI Systems and Computations, (Kung, et al Editors), Computer Science Press, 1981, pp.367-378.

[13] F. P. Preparata and T. Vuillemin, "Optimal Integrated Circuit Implementation of Triangular Matrix Inversion", Proc. Int'l Conf. Parallel Processing, August 1980, pp.211-216.

[28] K. H. Chu and K. S. Fu, "VLSI Architectures for High-Speed Recognition of General Context-Free Languages and Finite-State Languages", Proc. of 1982 IEEE Symposium Computer Architecture.

[29] H. T. Kung, "Why Systolic Architectures", IEEE Computer Magazine, Jan. 1982 pp. 37-46.

[30] H. T. Kung, et al, (Editors) VLSI Systems and Computations, Computer Science Press, 1981.

[31] L. Synader, "Introduction To The Configurable Highly Parallel Computer", IEEE Computer Magazine, Jan. 1982 pp. 47-64.

[32] N. S. Chang and Y. C. Yuan, "VLSI Design and Verification of a Signal Processing Chip", Proc. of Workshop on CAPAIDM, Hot Springs, Va. 1981, pp. 279-283.