PUMPS Architecture for Pattern Analysis and Image Database Management

FAYÉ A. BRIGGS, MEMBER, IEEE, KING-SUN FU, FELLOW, IEEE, KAI HWANG, SENIOR MEMBER, IEEE, AND BENJAMIN W. WAH, MEMBER, IEEE

Abstract—**The PUMPS architecture consists of P task processing units (TPU) which share a pool of special peripheral processors, VLSI functional units, and a common two-dimensional shared memory (SM) via a block transfer oriented interconnection network. A shared cache is provided between the TPU's and SM for efficient MIMD interprocessor communication. The SM is also connected via a backend database management network (BDMN) with distributed control to the file memories, which are disk-based database storage devices.**

A front end communication processor (FECP) is used to switch the control of I/O terminals. The FECP and the BDMN manage the relationally structured image databases in addition to providing the efficient processing capabilities for classical file manipulative primitives. The FECP and BDMN have unique features such as a language interface for relational image database management using query-by-picture example (QPE) and spatial operators.

Special VLSI functional units for pattern analysis and recognition of context-free languages and finite-state languages are presented. To effectively use the multiple resources on the system, they must be scheduled efficiently. A distributed scheduling algorithm is discussed with respect to the cube and Omega networks. Lastly, the architecture of a database machine for image processing is discussed. Special hardware for selection and histogramming are presented.

Index Terms—**Image database management, multiprocessor architecture, pattern analysis, reconfigurable architecture, resource sharing, VLSI computing structures.**

I. INTRODUCTION

PATTERN analysis tasks require a wide variety of processing techniques and mathematical tools. Computational processes required for both numerical and combinatorial algorithms are summarized in Table I (found on p. 981). The table reveals the computation-intensive properties, such as the solution of linear system of equations (LSE), ordered search and sorting, and constrained optimizations, that are frequently used in image analysis. This table is not meant to be exhaustive.

In most pattern analysis systems, large single processor computers are employed to process pictorial information. These computers are often designed for general applications and are not tailored for the special needs of pattern analysis and image database management. Many image analysis tasks require only repetitive Boolean operations or simple arithmetic operations defined over extremely large arrays of picture elements. Hence, the use of a large general-purpose computer results in intolerable waste of resources. Moreover, a rigidly structured architecture does not lend itself to the flexibility required to process a wide spectrum of pattern recognition and image processing (PRIP) algorithms. Various computer architectures have been proposed in the past for pattern recognition and image analysis, and a survey of some of these has been given in [16]. The existing pattern analysis system architectures consist of the array-structured machines such as ILLIAC IV, BSP, and STARAN to the multi-mini/microprocessor systems, such as C.mmp and C.m*. Other image processing systems are the real-time cellular-logic based CLIP-4 and the Toshiba TOSPICS which has a large image memory designed to reduce data transfer time [16]. Most of the above machines are too rigid in their configurations. Yet other systems, such as the PM^4 [5] and PASM [37], which limit their degree of reconfiguration to SIMD and/or MIMD, have been proposed.

These architectures do not satisfactorily address the bottleneck problems of manipulating large image databases, especially in a real-time environment. In a study on the effect of computer architecture on algorithm decomposition, a conclusion was reached that cost-effective solutions to many applications such as pattern analysis problems require some form of functional specialization in the computer architecture [22]. A general-purpose multiprocessor system which is designed basically for a time-sharing environment is not tailored to efficient implementation of specific applicative domain which requires the execution of special classes of algorithms.

It is well known that certain algorithms are more suited to SIMD array processing, and some to MIMD multiprocessing. However, there are also some classes of image algorithms in which there is a need to update a common database (such as the same copy of a histogram), or the need to efficiently process a moving image, in which pipelining is most adequate. In order to meet the processing requirements of the explosive amount of pictorial information, concurrency must be exploited maximally at all levels of a computation. At the instruction preprocessing level, pipelining can be implemented within the processor. At the arithmetic and logic operation level, pipelining and parallelism can be implemented in VLSI or special attached processing units. These units, when integrated with an efficient multiprocessing system which is organized at the control level in a relatively tightly coupled system, enable configurations of parallelism (synchronized or asynchronous)

Manuscript received January 18, 1982; revised July 7, 1982. This work was supported by the National Science Foundation under Grant ECS-80-16580. The project was conducted at the School of Electrical Engineering, Purdue University, West Lafayette, IN. The alphabetical order of the authors' names does not reflect the degree of their contributions.

F. A. Briggs is with the Department of Electrical Engineering, Rice University, Houston, TX 77001.

K.-S. Fu, K. Hwang, and B. W. Wah are with the School of Electrical Engineering, Purdue University, West Lafayette, IN 47907.

0018-9340/82/1000-0969$00.75

or of macropipelines [19]. The multiprocessor architecture manages the different sets of shared resources in the system. In addition, an efficient database system which is harmoniously integrated into the reconfigurable multiprocessor system will alleviate most of the inefficiencies exhibited by the rigid architectures. The reconfigurability implies that for each given parallel algorithm, an optimum path may be selected through the structure to efficiently execute the set of tasks. Moreover, the system utilization and throughput may be increased by having several algorithms in execution concurrently in different partitions of the system.

The PUMPS architecture described in this paper exploits the new emerging VLSI computing structures and special peripheral processors such as the attached array processor AP120B [14]. The PUMPS configuration also results from the need to relax the processing modes defined in the PM^4 architecture [5]. PUMPS organization introduces features such as distributed operating system, resource sharing and cost-effective resource pool configuration for a given applicative domain and workload. The PUMPS architecture can thus be easily reconfigured to suit a specially chosen application area.

The emphasis of application of the architecture in this paper is in pattern analysis and image database management. A general block diagram of a pattern analysis system, is shown in Fig. 1. There are essentially five processing stages. In the preprocessing stage, enhancement operations are performed to restore the noisy blurred patterns. The enhanced patterns are then segmented according to region features. The classification stage recognizes the segmented parts by indicating its membership in the pattern classes. Finally, structure analysis is performed to produce description and interpretation of the pattern information.

The pattern analysis system above is often applied to a continuous sequence of patterns. The processing time of such a repetitive task can be reduced by partitioning the task into a sequence of processes which can be executed on a macropipeline [19] configuration of the PUMPS. Each process can be executed in an autonomous unit, called a segment, which operates concurrently with other segments. In the PUMPS, a segment can be an SISD, SIMD, MIMD, pipeline unit, or specially dedicated VLSI processors. Hence, a macropipeline configuration of the PUMPS is composed of segments which are logically arranged so that consecutive processes of a task can be assigned to distinct segments of the macropipeline for processing. In general, the concept of macropipelining is of great importance for the flexibility and efficiency of PUMPS for processing PRIP algorithms [6].

In Section II, we outline the architectural features of PUMPS. The design issues in the configuration of macropipelining are enumerated in Section III. In Section IV, three examples of VLSI functional units are shown. These functional units are designed for matrix manipulation and recognition of context-free languages and finite-state languages. In Section V, the design of a distributed resource arbitration network is presented. In Section VI, the design issue of a database machine for image processing are discussed, and VLSI design for selection and histogramming is presented. Section VII provides some concluding remarks and directions for future studies.

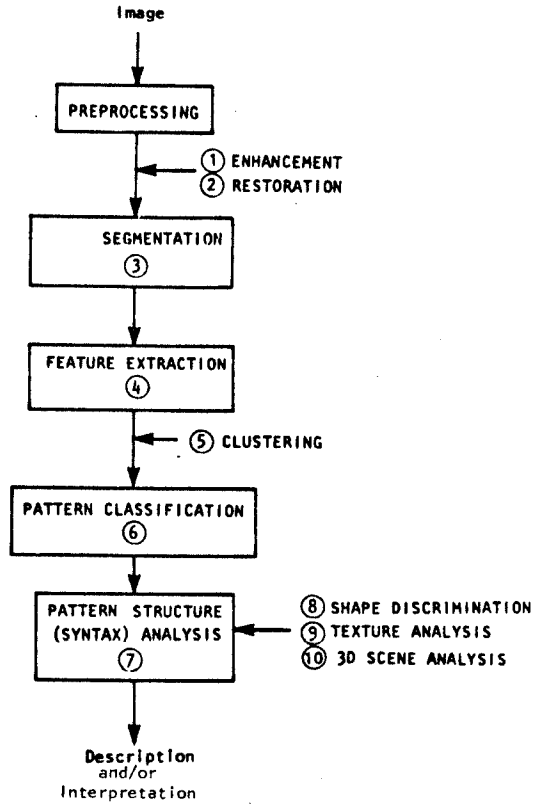


Fig. 1. A general flow diagram of pattern analysis system.

II. System Architecture of PUMPS

An integrated image analysis and image database processing system is depicted in Fig. 2. It consists of three closely cooperating subsystems, namely, the man–machine interface using a command interpreter, the image analysis subsystem, and the database management subsystem. The PUMPS architecture integrates these subsystems into an efficient multiprocessor system.

PUMPS is a high performance multiprocessor computer operating with SISD/MIMD task processors and a set of shared peripheral processors and VLSI units (PPVU's). A block diagram showing the major components in PUMPS is given in Fig. 3. In general, there are *P task processing units* (TPU's) in the system, each of which is multiprogrammed. They also can operate in an interactive fashion through the shared PPVU's and shared memory systems. The TPU's can communicate with each other via the task processor communications (TPC) bus. This intercommunication medium is very effective in passing interrupts, synchronization, and other control signals. All the TPU's are connected to the shared-resource pool of special PPVU's via a *special resource arbitration network* (SRAN). This network provides connections between each TPU and the desired PPVU. The SRAN is a low conflict interconnection network. Besides connecting any TPU to any PPVU, it must provide for arbitrary inter-PPVU paths. As VLSI technology develops, modular switches will become more cost-effective because of their regular and local connections [15]. In case several TPU's reference the same functional unit, some priority must be established to resolve the conflicts.

The allocation of the shared resources in the pool to the

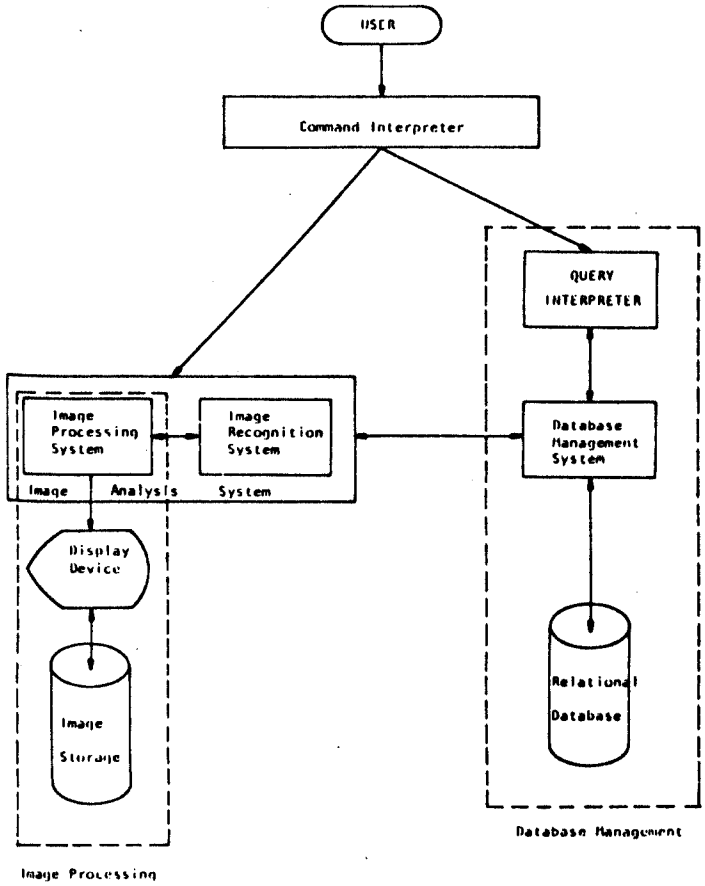


Fig. 2. Integrated image analysis and database management system.

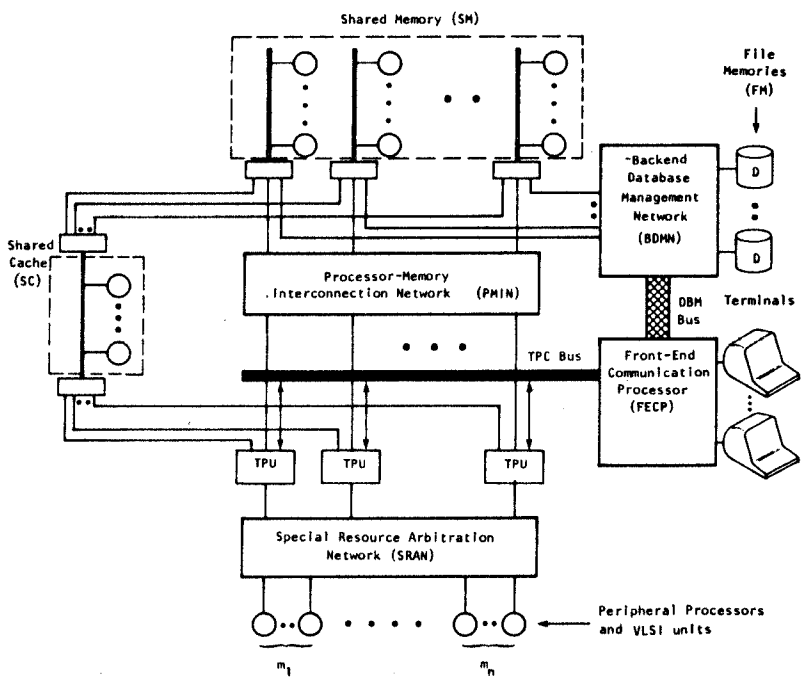


Fig. 3. The system architecture of PUMPS.

TPU's depends on the computational requirements of the active processes. The allocation is considered dynamic. Furthermore, the selection of the resource types in the pool is tailored to special application requirements. For example, one may wish to include an FFT processor, a histogram analyzer, and some VLSI array or pipeline processors in the pool for image analysis applications. In this sense, PUMPS has a dynamically reconfigurable structure. Different applicative environments may be equipped with different functional units. The remaining system resources, such as TPU's and shared

memories, are designed for general-purpose MIMD computations.

The TPU's perform three basic types of functions: 1) dispatching and initiating PPVU tasks, 2) executing purely sequential tasks, and 3) participating in MIMD processes and running the operating system. In order to perform the first type of function, a local *task memory* (TM) is provided within each TPU as depicted in Fig. 4. The TM is partitioned into several segments. These consists of the unmapped memory, which is used for the operating system kernel and device drivers, the local image buffers, and the local scratch-pad. The local image buffers are shared between the *task processor* (TP) and the PPVU's. The PPVU's are generally passive and the TPU, acting as a controller, must provide the PPVU with a continuous flow of data when the PPVU is processing a task. A resource controller and data channels (RCDC) in the TPU is used to format and channel the data between the TM and the PPVU.

To match the speed of the TP, a local *task cache* (TC), is provided between the TP and TM. The TC stores the most recently used private instructions and data of the active processes assigned to the TPU. Due to the locality property of programs, most of the references can be made in the cache. A multiprocessor system with private caches may encounter data coherence problems in which several copies of the same block of shared data may exist in different caches at any given time [12]. Basically, two methods have been proposed to solve this coherence problem. In the first method, whenever a processor attempts to update a variable in a TC, the possible copies of the variable in other caches must be modified accordingly before the process execution proceeds. This requires maintaining a central copy of the directories of each cache or selectively tagging cache blocks with common variables [7]. This method, although efficient, may be cost-prohibitive. The second method was proposed for the C.mmp, in which shared writable data are noncacheable and reside in shared memory. In the PUMPS architecture a compromise solution was sought to solve the cache consistency problem by tagging the data as private (*P*-data) or shared (*S*-data). The shared writable data are referenced in a unique cache (SC) shared by all the processors. An analysis of the performance of the shared cache concept shows its potential effectiveness [11].

The PUMPS has a distributed memory organization using virtual memory addressing based on paged segments. Within the TC, misses are serviced by a task memory management unit (TMMU) which initiates the block transfer from the TM to the TC, provided the block exists in the TM. If the block does not reside in the TM but is known to the process, a TM page fault occurs which is also serviced by the page-fault handler that resides in the TMMU. The occurrence of a TM page fault causes the current active process in the TP to be blocked, whereupon a task switch is made to a runnable process also residing in the TPU. By distributing the memory management functions over the TMMU's entire system, the TP's are relieved of performing memory management functions and thereby increase their effectiveness in performing useful computations. Such a memory management scheme was proposed in the PM^4 system.

The interleaved TM in TPU serves as a high speed buffer between the TPU and the *shared memory* (SM). The SM

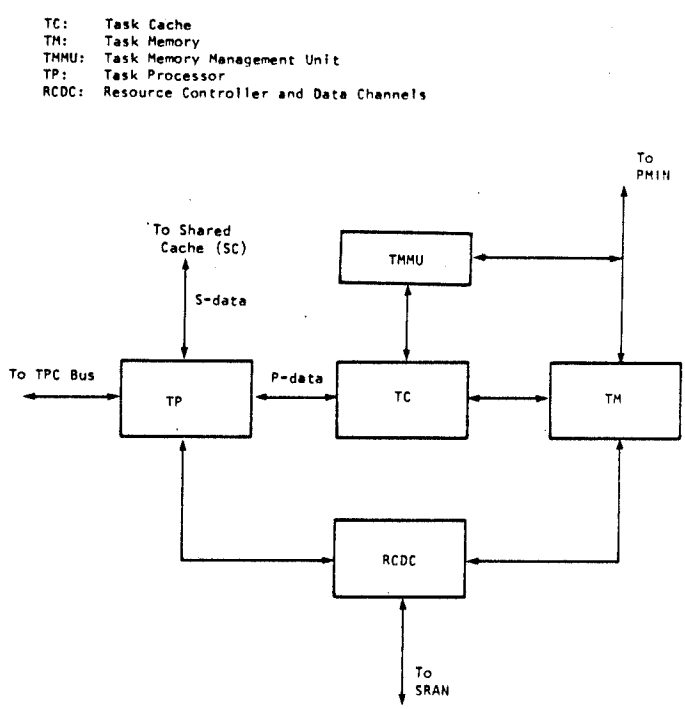


Fig. 4. Architecture of a task processing unit (TPU).

consists of semiconductor memories organized with multiported l lines and m memory modules per line, to permit efficient block transfers of information [11]. A block-transfer oriented *processor-memory interconnection network* (PMIN), such as the delta network [32], is used between the TPU's and the SM's. The shared memories are also connected to the *file memories* (FM) via a *backend image database management network* (BDMN), which is designed to handle the data transfers from multiple disks. The FM, together with a backend computer and the BDMN, comprise the database machine for image processing (DMIP).

The architectural design of the database machine for image processing is shown in Fig. 5. It consists of three parts: a set of data modules, each of which includes a disk with the associated cellular logic for processing picture queries, a backend computer, and the BDMN. The design of an efficient image database system of PUMPS differs from the conventional database design for alphanumeric information processing, because of the large amounts of imagery data involved [6]. The need to interface with image processing packages [8] and the necessity of providing a high-level language interface to the users, complicates the design issues. However, the design of the DMIP is based on the identification of the properties of various image data manipulation and retrieval operators. These operators are interpretable via a language interface which permits a logical representation of the images. To permit a high degree of concurrency of accesses to the DMIP, its control is distributed. The control mechanism requires the partitioning and replication of images and the placement of these partitions on the secondary storage. The images on the file memory are also dynamically reorganized for efficient retrievals. The design of the DMIP will be discussed in more detail the Section VI.

A *front-end communication processor* (FECP) is used to switch the control of I/O terminals and performs preprocessing

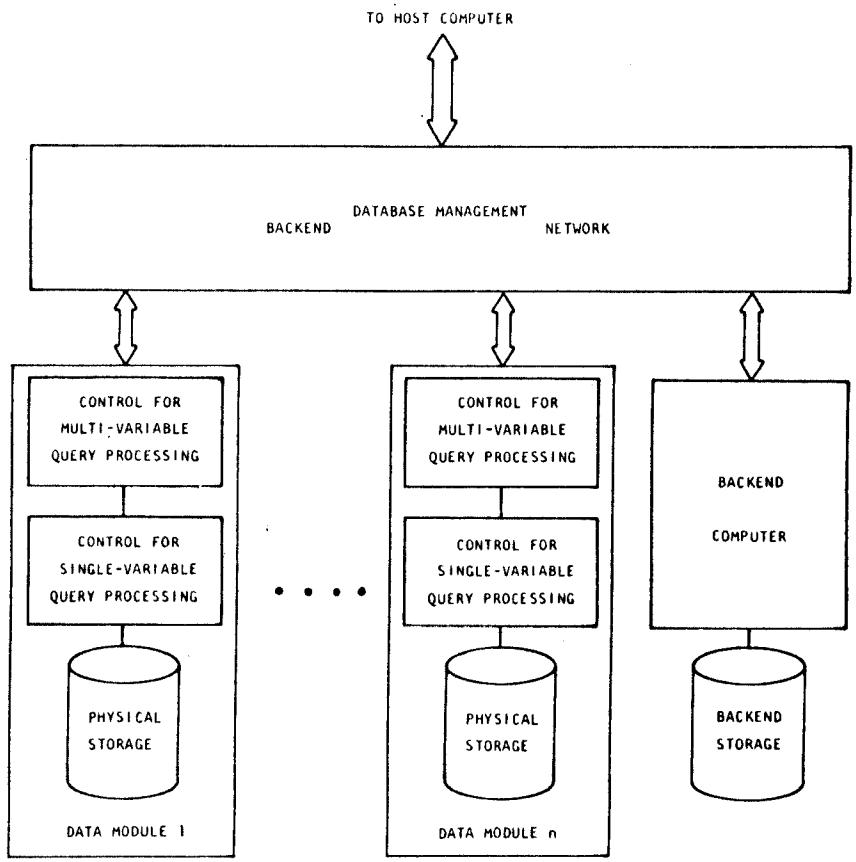


Fig. 5. Database machine for image processing.

task such as interactive file editing. The FECP together with the BDMN provide efficient processing capabilities for classical file manipulative primitives in addition to providing the users access to the whole multiprocessor system.

III. Configurations of Macropipelines in the PUMPS

In general, the implementation of the possible configurations of macropipelines in the PUMPS requires additional hardware support. Each processing segment or peripheral processor and VLSI unit (PPVU) has one full duplex buffers for data operands. The buffers are configured to enhance the access times of the operands and to control the synchronization of data inputs to the PPVU. The output of one segment is stored in the buffer of the next segment in the macropipeline. This buffer is used as the input to this next segment as shown in Fig. 6. Such structures of macropipelines characterize most image processing algorithms [30]. Many examples of macropipelines are found in the literature: real-time vision systems [29], analysis of motion [1], image reconstruction from projections [13], radar signal processing [2], and air traffic control [19].

For the PUMPS architecture, each buffer B_{ij} in Fig. 6 is partitioned into several independent logical modules, whose sizes are software selectable. Two successive segments cooperate in a producer/consumer relationship. One segment stores its output as successive batches of data in consecutive modules of the local buffer of the next segment, where they are processed sequentially. Of course, the consumer or producer must be suspended when the intermediate buffer is empty or full,

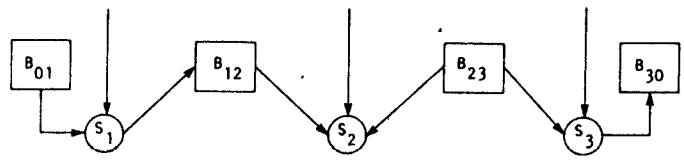


Fig. 6. Macropipeline in PUMPS architecture with S_i a TPU or PPVU.

respectively. P and V operations can be used to enforce the producer/consumer relationship. The flow of data through a PPVU is coordinated by a small local control unit. In practice, a segment may receive input data from the preceding segment and also from a TPU memory. The communication between the two segments of a macropipeline might be reduced to a few words, in which case, the local buffers are sufficient. In some cases, the local buffers are used to limit the congestion at the TPU's local memories when a picture is being streamed between two consecutive segments.

For efficient macropipelining, the algorithms used for two consecutive segments should be input/output compatible. For example, if one segment produces an output image in row-major format (line by line), then the succeeding segment should consume it also in row-major format to avoid additional segments for data format transformations (e.g., transposition). In this context, the choice of the PPVU's will depend on the typical system workload. The selection of the number of PPVU's of each type was discussed in [6].

If the execution times of the segments of the pipe are dissimilar, the overall performance of the task on the macropipeline configuration may be poor [23]. The performance is also affected by the randomness of the processing times on the TPU [11]. The processing times on a PPVU are generally

deterministic in nature. Some of the inputs to the pipeline may have deterministic interarrival times and others are random. A general study which takes into account these various aspects, would indicate decomposition strategies for good performance of a macropipeline configuration.

Macropipelines are generally efficient when the same task is repeated on successive input data sets. This is typical of real-time processing applications. In such applications, the impact on the performance of the pipeline startup time is reduced [23]. However, if the processing is performed on one input data set, a speed-up is still obtainable by applying the techniques discussed above. In this case, the intermediate data is not buffered between two segments but is forwarded directly to the next segment provided they are I/O compatible. The speed-up results by avoiding data manipulation which takes place in buffers. This technique is a generalization of *chaining* as implemented in the Cray-1 computer.

IV. Special VLSI Functional Units for Pattern Analysis

A typical task requires the coordination of different processes with varying characteristics. Some of these processes run more efficiently on dedicated PPVU's. When a particular process is executed frequently in a workload, it may become cost-effective to attach a special functional unit to the computer system for its processing, as provided by the PUMPS architecture. Examples of the special functional units are FFT analyzers and VLSI chips for solving linear system of equations [25]. Many computations required in image processing and pattern recognition are suitable candidates for VLSI realization. It is highly desired to develop VLSI computing structures for image smoothing, image registration, edge detection, image segmentation, texture analysis, multistage feature selection, syntatic pattern recognition, pictorial query processing, and image database management, etc. The ultimate purpose is towards the development of an effective, real-time, pattern-analysis and image-understanding system. The potential gain in VLSI hardware approaches lies not only in speed but also in reliability and cost-effectiveness.

A. VLSI Matrix Arithmetic Solvers

Many pattern analysis programs require to perform large-scale matrix/vector computations. These include matrix multiplication, L-U decomposition of a dense matrix, the inversion of a triangular matrix, and the solution of triangular systems of equations. Hwang and Cheng [25] have proposed a partitioned approach to realize these large-scale matrix computations by modularly structured VLSI arithmetic devices. Only four basic types of VLSI arithmetic chips are needed in implementing the partitioned matrix algorithms. Functional specifications of these primitive chip types are given in Fig. 7. Detailed logic design of these basic VLSI modules can be found in [24]. The four types, D, I, M, and V, perform submatrix L-U decomposition, inversion of triangular submatrix, cumulative submatrix multiplications, and cumulative matrix-vector multiplications, respectively. The following example shows how to triangularize a nonsingular dense matrix using the partitioned approach with VLSI matrix arithmetic modules of size $r \times r$, where n is the order of the given dense matrix A, and the case $k = n/r = 3$ is shown in the following example:

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

where all A_{ij}, L_{ij}, U_{ij} are $r \times r$ submatrices. L_{11}, L_{22}, L_{33} are lower triangular submatrices with all diagonal elements equal to 1. U_{11}, U_{22}, U_{33} are upper triangular submatrices.

Partitioned L-U Decomposition:

Step 1: $A_{11} = L_{11} \cdot U_{11}$ (Type D)

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 I and M)

Step 3: $\hat{A}_{22} = A_{22} - L_{21} \cdot U_{12} = L_{22} \cdot U_{22}$
$\hat{A}_{23} = A_{23} - L_{21} \cdot U_{13}$; $\hat{A}_{32} = A_{32} - L_{31} \cdot U_{12}$
(Types M and D)

Step 4: $U_{23} = L_{22}^{-1} \cdot \hat{A}_{23}$; $L_{32} = \hat{A}_{32} \cdot U_{22}^{-1}$
(Types M and I)

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

After decomposition, we need only to solve the triangular system as demonstrated below.

Partitioned VLSI Solution of a Triangular System of Equations:

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

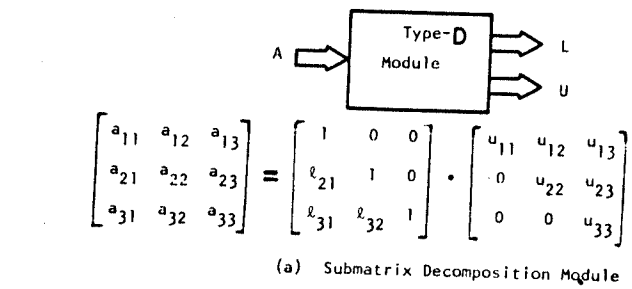
where X_i and D_i for $i = 1, 2, 3$ are $r \times 1$ column subvectors.

Step 1: $X_3 = U_{33}^{-1} \cdot D_3$ (Types I and V)

Step 2: $X_2 = U_{22}^{-1} \cdot (D_2 - U_{23} \cdot X_3)$
(Types I and V)

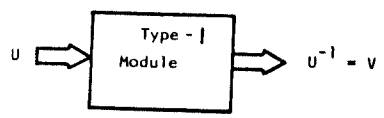
Step 3: $X_1 = U_{11}^{-1} \cdot [D_1 - (U_{12} \cdot X_2 + U_{13} \cdot X_3)]$
(Types I and V).

Speed performance and hardware chip count of these partitioned matrix algorithms are analyzed in [25]. With a strictly parallel architecture, we can achieve linear speed $O(n)$ with $O(n^2/r^2)$ VLSI chips. With a serial-parallel architecture, we can achieve $O(n^2/r)$ speed with $O(n/r)$ VLSI chips. The dominating chip type to be used will be the M-type as shown in Fig. 7. The functional design of a pattern classifier using the proposed VLSI matrix arithmetic chips is shown in Fig. 8. With sufficient training feature samples, we can classify the



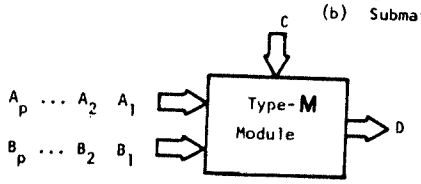
$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ \ell_{21} & 1 & 0 \\ \ell_{31} & \ell_{32} & 1 \end{bmatrix} \cdot \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

(a) Submatrix Decomposition Module



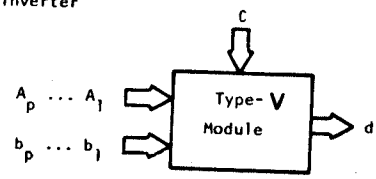
$$\begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}^{-1} = \begin{bmatrix} v_{11} & v_{12} & v_{13} \\ 0 & v_{22} & v_{23} \\ 0 & 0 & v_{33} \end{bmatrix}$$

(b) Submatrix Inverter



$D = C + \sum_{i=1}^{p} A_i \cdot B_i$ where C, D, and $\{A_i$ and B_i for $i = 1,\ldots,p\}$ are r x r matrices.

(c) Matrix Multiplier



$d = c + \sum_{i=1}^{p} A_i \cdot b_i$ where c, d, $\{b_i$ for $i = 1,\ldots,p\}$ are r x 1 column vectors, and $\{A_i$ for $i = 1,\ldots,p\}$ are r x r matrices.

(d) Matrix-Vector Multiplier

Fig. 7. Primitive VLSI matrix arithmetic modules.

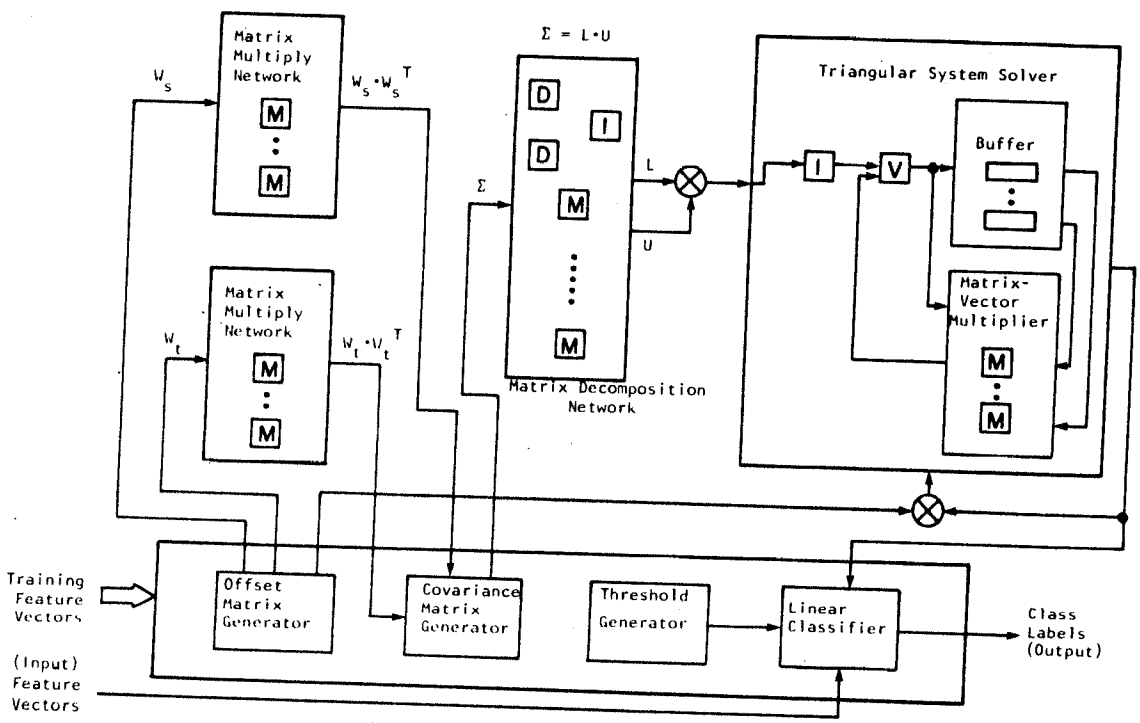


Fig. 8. Functional design of a pattern classifier using the proposed VLSI matrix arithmetic modules.

input pattern vectors (features) by Fisher's linear criterion. All the matrix manipulation networks in Fig. 8 can assume pipelined structures as detailed in [26]. A high degree of overlapped operations are performed in successive submatrix computation steps.

Feature extraction and pattern classification are the initial candidates for possible VLSI implementation [26]. 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 arithmetic solvers.

B. Recognition of Context-Free Languages

We present in this section a VLSI architecture for the high-speed recognition of context-free languages [9]. The architecture for context-free languages consists of $n(n+1)/2$ identical cells and is capable of recognizing an input string of length n in $2n$ time units. The architecture for finite-state language recognition consists of n cells and can recognize a string of length n in constant time. This will be presented in the next section. Since both architectures have characteristics such as modular layout, simple control and dataflow pattern, high degree of multiprocessing and/or pipelining, etc., they are very suitable for VLSI implementation.

The recognition methods employed will be based on the Cocke-Kasami-Younger algorithm [20] and its extension to finite-state languages recognition.

Let $G = (N, \Sigma, P, S)$ be a grammar in Chomsky normal form (with no null rule) and let $A = a_1 a_2 \cdots a_n$, $n \geq 1$, be a string where, for $1 \leq k \leq n$, $a_k \in \Sigma$. Form the strictly upper-triangular $(n+1) \times (n+1)$ recognition matrix T as follows, where each element $t_{i,j}$ is a subset of N and is initially empty (note: O-origin addressing convention is used for matrices).

Begin
  Loop1: **For** $i := 0$ **to** $n-1$ **do**
    $t_{i,i+1} := \{A \mid A \rightarrow a_{i+1}$ is in $P\}$;
  Loop2: **For** $d := 2$ **to** n **do**
    For $i := 0$ **to** $n-d$ **do**
      Begin
        $j := d + i$;
        $t_{ij} := \{A \mid$ there exists k,
          $i+1 \leq k \leq j-1$
          such that $A \rightarrow BC$
          is in P for some
          $B \in t_{i,k}, C \in t_{k,j}\}$
      End
End

If the element $t_{0,n}$ of the recognition matrix contains the start symbol S, then the string is accepted; otherwise the string is rejected.

The VLSI architecture for implementing the Cocke-Kasami-Younger algorithm in parallel can be divided into two parts, the preprocessing requirements and the hardware design. Preprocessing requirements are those tasks that are mostly input independent and therefore are required to be computed only once initially. The hardware design is the part that uses the preprocessing results and performs the recognition in $2n$ time units.

The preprocessing requirements have three parts. First, each distinct nonterminal of the grammar is numbered in ascending order. That is, if the nonterminal set is $\{S, A\}$ then the corresponding numbered nonterminal set will be $\{A_1, A_2\}$ where $A_1 = S$ and $A_2 = A$. Similarly, we also number the terminals of the grammar in this way. That is, if the terminal set is $\{b, a\}$ then the corresponding numbered terminal set will be $\{a_1, a_2\}$ where $a_1 = b$ and $a_2 = a$. With the grammar rules rewritten according to the numbered terminal and nonterminal sets, we can proceed to the next stage of the preprocessing requirements.

The next stage of preprocessing requirements is to construct a coded production table for the hardware. After we have done all the coding, we will have a coded production table for the production matching operation. The table will then be loaded into the memory module of each cell in the architecture during the initialization phase.

The last preprocessing requirement is to code every input string according to a special format. This corresponds to loop 1 of the Cocke-Kasami-Younger algorithm. First, we build a code table similar to the one before, except that we use terminals this time. That is, assume we have n terminals labeled $a_1, a_2, \cdots, a_n$, then for each terminal found in the order of labels, the corresponding set of nonterminals that derive it are coded in the same way as before. Now we can use this table to code the input strings by using table scanning. The table scanning can be done either in the host computer or on-the-fly by a simple content addressable memory [34], which contains this code table.

With the exception of the table scanning, all the other preprocessing tasks are input independent. Therefore, they are computed only once initially and the rest of the recognition tasks will be carried out by hardware.

In order to implement the Cocke-Kasami-Younger algorithm efficiently in hardware, the VLSI structure is chosen to be the same as the strictly upper triangular recognition matrix T (see Fig. 9). On the other hand, data in each matrix element are represented by using an s-bit binary word as described in the preprocessing tasks. The hardware design can be subdivided into two portions: the dataflow requirement and the functional units design. The dataflow requirement specifies the necessary data communications between cells, whereas the functional units design handles the required operations on input data within a cell.

The dataflow requirement is specified in loop2 of the Cocke-Kasami-Younger algorithm. It is easy to see that element (ij) of the recognition matrix needs to receive data from elements (ik) and (kj), for $i < k < j$. Observe also that there is no data dependency among the elements on a particular diagonal of the recognition matrix; therefore, we can compute the elements on a diagonal in parallel if the required data for each element is arranged to arrive at the right moment.

Let $t = j - i$ be the distance between cell (ij) and the boundary (see Fig. 9). The result of a cell at distance t will be

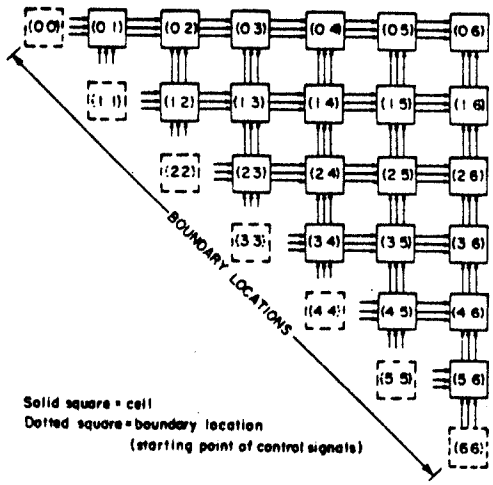


Fig. 9. The architecture for recognition of context-free languages, with $n = 6$.

ready at time $2t$; the cell then transmits its result upwards and to the right. This result travels at a rate of one cell per time unit for t additional time units and then slows down to one cell in every two time units until the recognition is completed. Using this algorithm, a network with $n(n+1)/2$ cells will require $2n$ time units for the final result to be available. Detailed schematic logic design of the recognition cells can be found in [9]. A VLSI architecture for the recognition of context-free languages using Earley's algorithm has also been proposed recently [41].

C. Finite-State Languages Recognition

Since finite-state languages form a subset of context-free languages, the Cocke–Kasami–Younger algorithm and hence, the architecture in Section IV-B, can also be used to recognize finite-state languages. In this section, a more efficient method for finite-state language recognition, which is derived from the Cocke–Kasami–Younger algorithm will be employed [9]. The VLSI architecture using this algorithm can be made to have constant response time by using pipelining if the number of cells is sufficient.

The method used here is derived from the Cocke–Kasami–Younger algorithm. This is done by observing the fact that finite-state grammar is a kind of right-linear grammar which is a special case of the algorithm. The modified algorithm for recognition of finite-state languages is given as follows:

Modified Algorithm: Let $G = (N, \Sigma, P, S)$ be a finite-state grammar (with no null rule) and let $A = a_1 a_2 \cdots a_n$, $n \geq 1$ be a string, where for $1 \leq i \leq n$, $a_i \in \Sigma$. Form the linear recognition array T as follows, where t_i is a subset of N:

Begin

$t_1 := \{A \mid A \rightarrow a_n \text{ is in } P\}$;

Loop 1: **For** $i := 2$ **to** n **do**

Begin

$j = n - i$

$t_i := \{A \mid A \rightarrow a_{j+1} B \text{ is in } P$ for some $B \in t_{i-1}\}$

End

End

If the element t_n of the linear recognition array T contains the start symbol (or final state) then the string is accepted; otherwise the string is rejected.

The architecture design is again divided into two parts: the preprocessing requirements and the hardware design. The preprocessing requirements are about the same as before.

In order to utilize the processors' organization with the chosen data structure, the architecture also employs a linear array structure of size n as shown in Fig. 10, where A is the input string. On the other hand, the same data representation in Section IV-B for nonterminals in each array element is used. Notice that the computation of the next element of T in this algorithm depends only on the present element. Therefore, pipelining technique can be applied to make the recognition in constant time. The design a typical cell for the recognition of finite-state languages can be found in [9].

V. Distributed Scheduling of Resources on Interconnection Networks

We have discussed earlier that available resources on the system are arbitrated by the SRAN. In this section, we discuss a distributed algorithm for this purpose [40].

In general, an interconnection network routes requests from a set of source points to a set of destination points (they may coincide with each other). In a *resource sharing interconnection network* (*RSIN*), the destination points are identical (or sets of identical) resources for which requests or tasks can be delegated to. Examples of these are special purpose VLSI chips. In this respect, jobs initiated at source processors can be sent to any one of the free resources of a given type at the destination. This is the important point that differentiates RSIN's from interconnection networks using address mapping. Another application is to use the same set of destination points as the source points. In this case, we have a load balancing network which can rebalance the load in the system dynamically. RSIN's can also be used as distribution networks in dataflow machines.

Since the system operates continuously, requests from source processors can be initiated at random times. At any time, a set of processors may be making requests and a set of resources are free. It is the function of a scheduler to set the RSIN in order to connect the maximum number of resources to the processors, that is, to have the maximum resource utilization.

The earliest study of RSIN has been done with respect to centralized computer systems. A unibus is used in a time-shared fashion for connecting peripheral I/O devices to the CPU. Multiple time-shared buses have been used in the PLURIBUS minicomputer multiprocessor [31]. A cross-bar switch has been used in C.mmp although the network is mostly used in address mapping mode. This network permits full interconnection capability between any source and destination ports. As long as each source port addresses a unique destination port, there is no blocking in the network and all messages can be routed through the network simultaneously. The single or multiple buses are a source of bottleneck, and are the

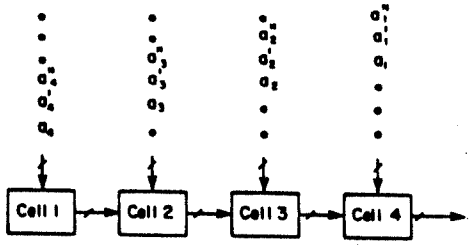


Fig. 10. Architecture for recognition of finite-state languages with $n = 4$.

east expensive designs. The cross-bar switch is the most ex-ensive network but has the least degree of blocking. A com-romise is to use a less expensive network than the cross-bar witch and one that has less blocking probability than the single us systems. This has been studied with respect to the Banyan etwork [35]. In these studies, it is shown that when a pro-essor makes a request for multiple resources, allocating re-ources with smaller distance functions reduces the amount f network blockage caused by the allocation of these resources 8]. A tree network is proposed to aid the scheduler in noosing a resource to allocate. The tree network has a delay $O(\log_2 n)$ in selecting a free resource (n is the total number resources).

A few comments can be made about the previous studies. irst, the scheduling algorithms are centralized. For mapping requesting processors to n resources, the scheduling algo-thm has a worst case complexity of $O(n * \log_2 n)$. This mplexity depends on the number of requesting processors. his is practical when n is small or when requests are not very equent. Second, for scheduling requests on interconnection tworks with logarithmic delays such as the binary n-cube, nyan, and Omega, no optimal scheduling algorithm has en established.

The Omega [27] and binary n-cube [33] networks are osen for their simplicity and versatility. The basic element these networks is a two input, two output interchange box ich allows a straight or a diagonal connection. For a network nnecting N inputs to N outputs (N is a power of 2), there are $_2 N$ stages and $N/2 * \log_2 N$ interchange boxes. The delay the network is therefore $O(\log_2 N)$.

Simulation results presented in [15] show that with $N = 8$, ere is a message blocking probability of about 30 percent ng address mapping. We show in this section that there is tually no blocking when these networks operate as a IN.

The results are obtained by exhaustive enumeration of all possible combinations of connections for a subset of re-esting processors and free resources. Because of the large mber of combinations, only networks with $N = 8$ can be ulated. Even in this case, the total number of possible nbinations is over 8 million. The large number of combi-ions is attributed to the fact that the order of connections mportant. For a set of i requesting processors and j free ources, there are $\binom{i}{j} j!$, $(i \geq j)$ or $\binom{j}{i} i!$, $(j > i)$ possible ered connections.

A faster method is developed by observing that each box can et in two states. With 12 interchange boxes, there are 2^{12} 096 states or possible connections. These 4096 possible connections are arranged into multiple trees so that the maximum number of connections can be found efficiently. Using this method, the enumerations were completed using 10 hours of CPU time and 64 kbytes of memory on a VAX 11/780.

A selected set of the simulation results are plotted in Fig. 11 for the blocking probability. These results are based on the assumption that the network is completely free before the allocations. The average processor blocking probability is defined as the ratio of the number of allocated processors to the minimum of the number of requesting processors and the number of free resources. It is seen that the blocking probability and the standard deviation of processor allocations are very small. We can conclude that with a good scheduling algorithm, these networks serve almost equally as well as the cross-bar switch for resource sharing.

The centralized scheduling algorithm has a high overhead when the number of processors and resources to be scheduled is large since every requesting processor has to be scheuled sequentially. In a distributed algorithm, all the requesting processors are scheduled in parallel. The resource scheduling overhead is, therefore, proportional to the delay time in the network and independent of the number of requesting processors.

The distributed algorithm is implemented by distributing the scheduling intelligence into the interconnection network so that there is no centralized control. Each exchange box can resolve conflicts and route requests to the appropriate destination. If a request is blocked, it will be sent back to the originating exchange box in the previous stage. Request routing is thus dynamic, and all the exchange boxes operate independently.

The algorithm consists of two phases. In the first phase (*resource phase*), information concerning the number of free resources is passed from the resource side to the processor side. In the second phase (*request phase*), the network propagates the requests from the processors to the resources. This uses the information that is obtained in the resource phase. When multiple signals are pending in a box, priority must be set to determine the order of servicing these requests.

The algorithm described above does not preclude dynamic operation. In fact, requests can be initiated at random times and they will be routed to a free resource or be rejected. The operation of the exchange box can be completely asynchronous. An accepted request is known to a processor when an acknowledgment is received along the data link. A request is rejected when a rejection signal is received by the processor. A rejected request can be retried later.

The performance of the distributed algorithm is again plotted in Fig. 11. It is seen that the blocking probability is less than 19 percent in all cases and compares favorably with the optimal case. The maximum average delay time for a request to access a free resource or be rejected is 4.2 units of time (not shown) in which the delay through an exchange box is 1 unit. The delay time of the algorithm is dependent on the delay in the network and not on the number of requesting processors. Further discussion of the network can be found in [40].

The scheduling algorithm discussed above can be applied in the database machine design in which multiple subsets of identical VLSI chips are applied. This is discussed in the next section.

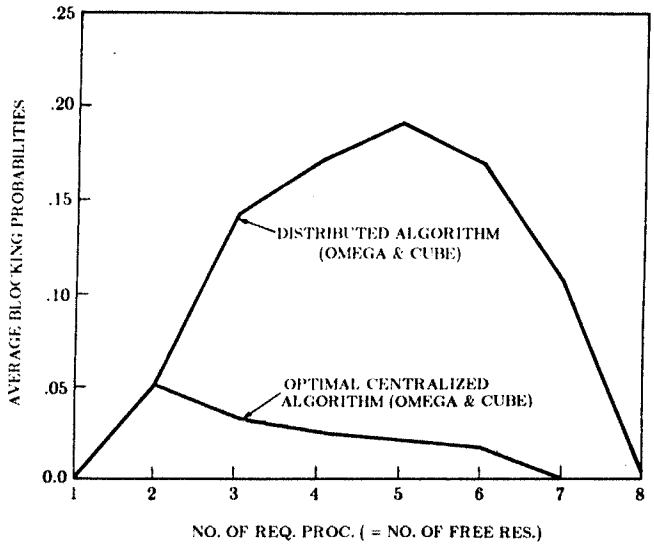


Fig. 11. Blocking probability for resource allocations on Omega and binary n-cube networks of size $N = 8$.

VI. Backend Image Database Management System

In the design of PUMPS, imagery data must be staged to the shared memory (SM) before they are accessible by the TPU's. This is efficient only when the same data are repeatedly used. In this case, the overhead of the transfer is balanced by the efficiency in processing. On the other hand, some image processing operations resemble the conventional database processing operations, namely, a large amount of data is processed a small number of times. In this case, it is more efficient to process the data directly on the storage medium and eliminates the congestion in the processor-memory interconnection network (PMIN). Furthermore, it relieves the TPU's to process operations that are more CPU-bound. The database machine for image processing (DMIP) is the result of an architectural approach which distributes processing power closer to devices on which data are stored and offloads database processing functions from the main computer.

The motivations behind designing image database machines are threefold. First, data transfer over the memory hierarchy is expensive and therefore, it is preferable to process the image without the intermediate data transfer. Second, the cost of hardware, in particular VLSI, is decreasing. This allows many specialized functions to be implemented directly in hardware. Lastly, the size of image database is growing rapidly and the need for distributed processing of imagery operations is inevitable.

Nearly all the research on database machines are directed towards conventional databases which can be classified into four types. The first type is the backend system which utilizes minicomputers to enhance the database processing of large host computers. The functions of the backend system include access validation, storage management, concurrency control, and I/O control. IDM by Britton Lee, Inc. is an example of this class.

The second type of database machine utilizes the single-instruction multiple-data stream (SIMD) principle. This concept is extended from backend machines in which the database processing functions are moved to a lower level. In this class, simpler, less costly processors are dedicated to a small block of data. This concept is exemplified by the logic-per-track device in which processing logic are duplicated for each track of the disk and the keys in different tracks are searched in parallel. Examples of this design are CASSM [28], RAP [36], and DBC [3]. If the replication goes further down to the bit level, an associative memory results. This is exemplified by RELACS [4], in which STARAN is used as the associative memory.

The third type of database machine design is based on the multiple-instruction multiple-data stream (MIMD) principle. This design is exemplified by DIRECT [10] in which the processing logic are interconnected with an array of CCD memory modules through a cross-bar switch. A variable number of processors can be assigned to process a database query. This design offers more flexibility and better load balancing and allows the processors to be shared among the storage modules.

The last type of database machine design is based on the distributed system principle, utilizing modules which consist of a storage medium with enhanced intelligence, and which communicate with each other through a communication network. DIALOG is an example of this class [39]. The advantages of this type of database machine are its flexibility and its expandability. Furthermore, heterogeneous storage devices can be accommodated in the system. Techniques developed for the optimization of distributed databases are also applicable for the design of this class of database machines.

A database machine for image processing can be designed as one or a combination of the aforementioned classes. The following functional features are identified. High level database functions such as selection, projection, and join are implemented. These operations are useful for manipulating the image database. On the other hand, low level image processing operations such as histogramming and edge detection are also implemented. An image database machine is, therefore, a conventional database machine enhanced with low level image processing hardware.

We have previously studied the design of a relational database system for images—IMAID [8], [17]—and a relational database machine—DIALOG. IMAID is designed as an integrated database system interfaced with an image understanding system for the efficient storage and retrieval of images and pictures. By using image processing and pattern recognition manipulation functions, structures and features of images are extracted and integrated into relational databases. A relational query language, query-by-pictorial example (QPE) [8] is introduced for manipulating queries regarding spatial relations as well as conventional queries. Conventional database manipulation operators such as selection and join are used to manipulate imagery objects stored in relations.

A general assumption about VLSI chips are that they are inexpensive. For complex operations, this is not really true due to the fact that external control, timing, memory, and software must be provided. Furthermore, as the types of VLSI chips increase and the degree of replication is large, the system becomes expensive. A solution to this problem is to use a resource

sharing interconnection network so that a pool of common resources can be used. This concept is illustrated in Fig. 12. VLSI chips are distributed into each storage module. They can be used for real-time off-the-track processing. A pool of common resources are also shared among the storage modules. The resource-sharing interconnection network connects these resources to the storage medium. We present in the remainder of this section a specialized VLSI design that can be used to perform real-time histogramming and selection from data off the disk.

Fig. 13 shows the architecture of the selection processing module. There is a common part of the circuit that is used for both selection and histogramming. This consists of n circulating registers which contain keys to be matched. The timing and control module controls the associative logic which is capable of performing equality match, proximity, and threshold operations [34]. A subset of the associative logic is enabled and selected keys are compared with the data output from the disk. The multiple match resolution circuit is used to select the first 1 from a set of responding 1's and is similar to those designed for associative memories [34]. The software programmable logic array (SPAL) computes the output $Z = f(x_1, \cdots, x_n)$ with a user selected function f of n variables. The output Z is used to control the gate which filters off unnecessary information from the disk. Since the selection function does not change throughout the matching of the entire database, the function f can be input before the selection begins. f is represented in disjunctive normal form and each term of f can be input sequentially. If n is larger than the number of possible inputs of a single SPAL, the n inputs can be partitioned into sets and fed into multiple SPAL's. Multiple outputs $Z_1, Z_2, \cdots, Z_k$ will be obtained and supplied to a single SPAL to compute Z. The circuit can be extended systematically into a tree of SPAL's.

The selection circuit operates in periods in which a period is the length of time for a database record (tuple) to be output from the disk. The database data are fed to all the associative logic. As selected fields of the record are detected, a subset of the associative logic is enabled and the results of the matching are stored in temporary flip flops. After the end of record is detected, the values $X_1, \cdots, X_n$ stored in the temporary flip flops are output to the SPAL, which computes the value Z and decides whether the record is to be discarded.

The advantages of this selection circuit are threefold.

1) *Generality:* The design is more general than the finite state automaton approach [21] and allows arbitrary logic functions (with limitation imposed by the size and number of the registers) to be implemented. It follows the parallel discrete comparators approach of Stelhorn [38] except that all the functions are implemented in hardware and is, therefore, much faster.

2) *Reconfigurability:* The size of a key is not limited to the size of a register. For equality and threshold operations, suppose it is needed to compute $D \geq B$ where D is a field of the database tuple and B is a key. If B and D have a size greater than the size of the register, they can be partitioned into k subkeys, $B = B_1, \cdots, B_k, D = D_1, \cdots, D_k$ and the operation $D \geq B$ is equivalent to $(D_1 > B_1) \vee ((D_1 = B_1) \wedge (D_2 > B_2)) \cdots \vee ((D_1 = B_1) \wedge (D_2 = B_2) \wedge (D_k > B_k))$. Therefore, the registers can be reconfigured to perform operations on keys longer than the size of a register.

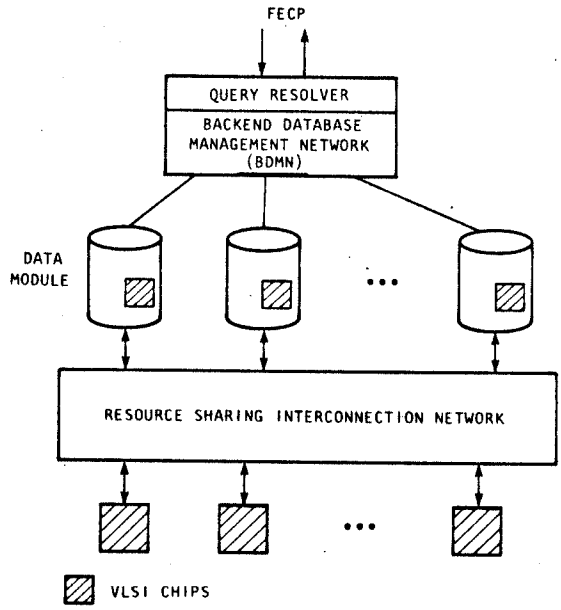


Fig. 12. A conceptual view of an image database machine.

3) *Speed:* Because the computation of the function f by the SPAL is completely overlapped with the associative matching of the next record, the operation is pipelined and can result in a higher throughput than conventional selection circuits where the function f is computed simultaneously with the matching operations.

To use the circuit to compute the histogram of an image, the registers will be set to the different thresholds and the associative logic are set to perform the less than functions. As each pixel of image is output, all the registers with keys smaller than the value of the pixel is enabled and enable the SPAL. The multiple match resolution circuit selects the correct interval where the counter should be incremented. Based on the results of the counters, a second pass of the imagery data would allow thresholding to be performed.

VII. Conclusions

The PUMPS architecture generates a number of related research topics that need to be investigated. One basic topic is the development of an appropriate operating system and language that can be used to effectively control and express the pattern analysis and image database management tasks. Another operating system related task is the memory management method and its implementation. Pattern analysis algorithms cannot be classified as having properties of the more general programs. For example, an algorithm for performing a specific function on a VLSI processor may be well behaved. In which case, parameters such as looping distance and page reference string will be well defined for a given task dimension. With such parameters we can develop an efficient image buffer management policy and determine appropriate buffer size for each VLSI unit. VLSI technology can be explored for fast manipulation and update of relational image database and permit a modular growth.

Implementation of a processing task on the PUMPS ar-

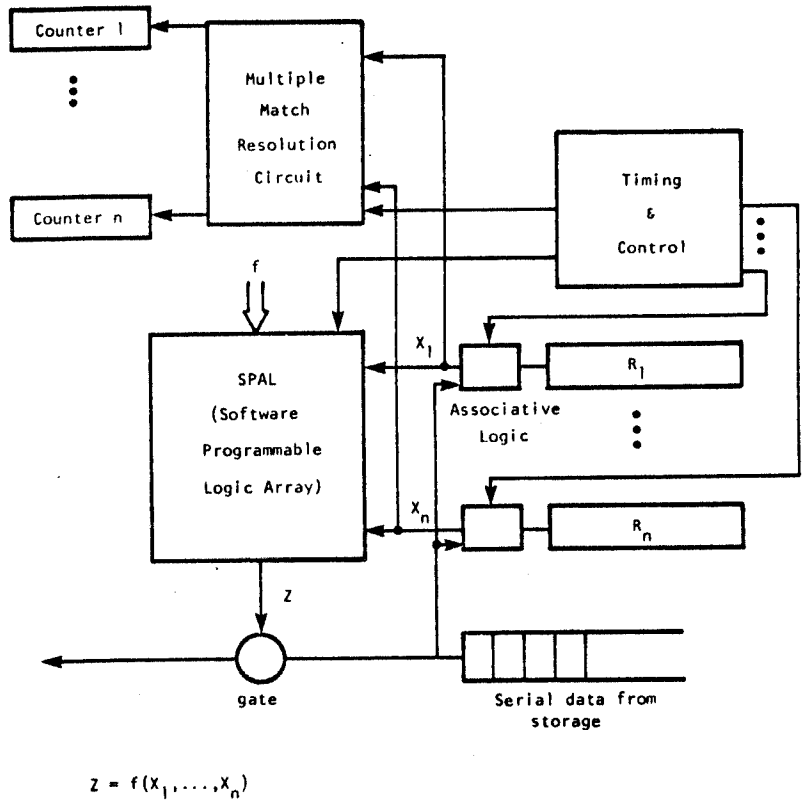


Fig. 13. Architecture of the selection processing module for selection, projection and histogramming.

TABLE I
IMAGE ANALYSIS AND DATABASE MANAGEMENT PROBLEMS AND THEIR REQUIRED COMPUTATIONS

Computational Tasks Required \ Image Analysis and Database Problems	Image Enhancement	Image Restoration	Image Segmentation	Feature Extraction & Selection	Clustering Techniques	Statistical Pattern Classification	Pattern Structure (Syntax) Analysis	Shape Discrimination	Texture Analysis	Scene Analysis	Alphanumeric and Image Database Management	Image Query Language Processing
Iterative relaxation methods	X	X	X		X			X		X		
Constrained and unconstrained optimizations	X	X	X	X		X		X				
Matrix manipulation (Solving Algebraic equations)		X		X	X	X						
Transformation and convolution	X	X		X					X			
Interpolation and functional evaluation		X	X	X				X	X			
Histogramming	X		X	X					X			
Logical operations (boolean)	X	X	X	X		X	X			X	X	X
Similarity retrieval and spatial operations								X			X	
Set Manipulation and relational operations			X		X		X			X		
Ordered search or heuristic search	X		X		X		X	X		X		
Sorting, pattern matching and graph operations					X		X	X		X	X	
Backtracking and syntax parsing							X	X	X	X		X
Language translation and compiling					X		X					X
Image Display Operations									X		X	

chitecture requires the efficient allocation and scheduling of system resources. In particular, the effective utilization of the PPVU's is important. The match of the set of PPVU configurations with the specific application needs makes the PUMPS very attractive. The unification of the image database management subsystem with the user-oriented multiprocessor system in the PUMPS permits an effective on-line processing of pattern analysis problems and the interactive management of imagery data. Continued research is needed for the development of specialized VLSI functional chips. Some of these algorithms are listed in Table I. Research in the effective scheduling of resources also warrant further study. Lastly, the

system must be evaluated and simulated over various applications related to pattern analysis and image processing.

REFERENCES

[1] D. P. Agrawal and R. Jain, "Computer analysis of motion using a network of processors," in *Proc. 5th Int. Conf. Pattern Recognition*, Dec. 1980.

[2] C. V. Armstrong *et al.*, "An adaptive multimicroprocessor array computing structure for radar signal processing applications," in *Proc. 6th Annu. Symp. Comput. Arch.*, 1979.

[3] J. Banerjee, D. K. Hsiao, and K. Kannon, "DBC—A database computer for very large data bases," *IEEE Trans. Comput.*, vol. C-28, June 1979.

[4] P. B. Berra and E. Oliver, "The role of associative array processors in data base machine architecture," *IEEE Computer*, Mar. 1979.

[5] F. A. Briggs, K. S. Fu, K. Hwang, and J. H. Patel, "PM^4—A reconfigurable multiprocessor system for pattern recognition and image processing," in *Proc. NCC*, 1979, pp. 255-265.

[6] F. A. Briggs, K. Hwang, K. S. Fu, and B. W. Wah, "PUMPS architecture for pattern analysis and image database management," in *Proc. 1981 IEEE Comput. Soc. Workshop Comput. Arch. Pattern Analysis and Image Database Management*, Virginia, Nov. 11-13, 1981, pp. 178-187.

[7] L. M. Censier and P. Feautrier, "A new solution to coherence problems in multicache systems," *IEEE Trans. Comput.*, vol. C-27, Dec. 1978.

[8] N. S. Chang and K. S. Fu, "A relational data base system for images," in *Pictorial Information Systems*, S. K. Chang and K. S. Fu, Eds. New York: Springer-Verlag, 1980.

[9] K. H. Chu and K. S. Fu, "VLSI architectures for high speed recognition of general context-free languages and finite-state languages," in *Proc. 9th Annu. Symp. Comput. Arch.*, Austin, TX, Apr. 1982.

[10] D. J. DeWitt, "DIRECT—A multiprocessor organization for support relational data base management systems," *IEEE Trans. Comput.*, vol. C-28, June 1979.

[11] M. Dubois and F. A. Briggs, "Efficient interprocessor communications for MIMD multiprocessor systems," in *Proc. 8th Ann. Symp. Comput. Arch.*, 1981, pp. 187-196.

[12] ——, "Effects of cache coherency in multiprocessors," in *Proc. 9th Annu. Symp. Comput. Arch.*, Apr. 1982. pp. 299-308.

[13] E. J. Farrell, "Processing limitations of ultrasonic image reconstruction," in *Proc. Conf. Pattern Recognition Image Process.*, June 1978.

[14] *AP-120B Processor Handbook*, Floating Point System, Inc., Pub. 7259-02, Portland, OR.

[15] M. A. Franklin, "VLSI performance comparison of Banyan and crossbar communications networks," *IEEE Trans. Comput.*, vol. C-30, Apr. 1981.

[16] K. S. Fu, "Special compute architectures for pattern recognition and image processing," in *Proc. 1978 Nat. Comput. Conf.*, pp. 1003-1013.

[17] K. S. Fu and N. S. Chang, "An integrated image analysis and image database management system," in *Proc. COMPCON Fall '82*, Washington, DC, Sept. 1982.

[18] L. R. Goke, "Banyan networks for partitioning multiprocessor systems," Ph.D. dissertation, Univ. Florida, Gainesville, 1976.

[19] W. Handler, "The concept of macropipelining with high availability," *Elektronische Rechenanhagan*, no. 15, pp. 269-274, 1973.

[20] M. A. Harrison, *Introduction to Formal Language Theory*. Reading, MA: Addison-Wesley, 1978, pp. 417-500.

[21] L. A. Hollar and D. C. Roberts, "Current Research into specialized processors for text information retrieval," in *Proc. of 4th Int. Conf. Very Large Databases*, Berlin, Germany, 1978, pp. 270-279.

[22] R. Hon and D. R. Reddy, "The effect of computer architecture on algorithm decomposition and performance," in *High-Speed Computers and Algorithm Organization*, Kuck *et al.*, Eds. New York: Academic, 1977, pp. 411-421.

[23] K. Hwang, S. P. Su, and L. M. Ni, "Vector processing computer architecture," in *Advances in Computers*, Yovits, Ed., vol. 20. New York: Academic, 1981, pp. 115-197.

[24] K. Hwang and Y. H. Cheng, "Partitioned matrix algorithms and VLSI structures for large-scale matrix computations," in *Proc. IEEE 5th Symp. Comput. Arch.*, May 1981, pp. 222-232.

[25] ——, "Partitioned matrix algorithms for VLSI arithmetic systems," *IEEE Trans. Comput.*, to be published, 1982.

[26] K. Hwang and S., P. Su, "VLSI matrix manipulators for feature extraction and pattern classification," School Elec. Eng., Purdue Univ., West Lafayette, IN, Tech. Rep. TR-EE 81-44, 1981.

[27] D. Lawrie, "Access and alignment of data in an array processor," *IEEE Trans. Comput.*, vol. C-24, pp. 215-255, Dec. 1975.

[28] G. J. Lipovski, "Architectural features of CASSM: A context addressed segment sequential memory," in *Proc. 5th Annu. Symp. Comput. Arch.*, May, 1978.

[29] G. Nicolac and K. H. Hohne, "Multiprocessor system for the real-time digital processing of video-image series," *Elektronische Rechenanhagen*, no. 21, 1979.

[30] R. Nudd, "Image understanding architectures," in *Proc. AFIPS*, 1980.

[31] S. M. Ornstein *et al.*, "Pluribus—A reliable multiprocessor," in *Proc. AFIPS 1975 Nat. Comput. Conf.* Montvale, NJ: AFIPS Press, 1975, pp. 551-559.

[32] J. H. Patel, "Performance of processor-memory interconnection for multiprocessors," *IEEE Trans. Comput.*, vol. C-30, pp. 771-780, Oct. 1981.

[33] M. C. Pease, "The indirect binary n-cube microprocessor array," *IEEE Trans. Comput.*, vol. C-26, May 1977, pp. 458-573.

[34] C. V. Ramamoorthy, J. L. Turner, and B. W. Wah, "A design of a cellular associative memory for ordered retrieval," *IEEE Trans. Comput.* vol. C-27, Sept. 1978.

[35] B. D. Rathi, A. R. Tripathi, and G. J. Lipovski, "Hardwired resource allocators for reconfigurable architectures," in *Proc. 1980 Int. Conf. Parallel Process.*, Aug. 1980, pp. 109-117.

[36] S. A. Schuster *et al.*, "RAP.2—An associative processor for data base and its applications," *IEEE Trans. Comput.*, vol. C-28, June 1979.

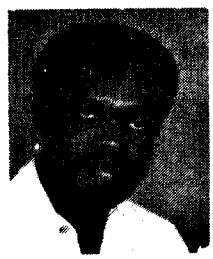
[37] H. J. Siegel *et al.*, "PASM: A partionable SIMD/MIMD system for image processing and pattern recognition," *IEEE Trans. Comput.*, vol. C-30, pp. 934-947, Dec. 1981.

[38] W. H. Stelhorn, "A processor for direct scanning of text," in *Proc. 1st Workshop Comput. Arch. Non-Numeric Process.*, Dallas, TX, 1974.

[39] B. W. Wah and S. B. Yao, "DIALOG—A distributed processor organization for database machine," in *Proc. Nat. Comput. Conf.*, vol. 49, 1980, pp. 243-253.

[40] B. W. Wah and A. Hicks, "Distributed scheduling of resources on interconnection networks," in *Proc. Nat. Comput. Conf.* AFIPS Press, June 1982.

[41] Y. T. Chiang and K. S. Fu, "A VLSI architecture for fast context-free language recognition (Earley's algorithm)," in *Proc. 3rd Int. Conf. Distributed Comput. Syst.*, Oct. 18-22, 1982.



Fayé A. Briggs (M'77) received the B.Eng. degree from Ahmadu Bello University, Nigeria, in 1971, the M.S. degree from Stanford University, Stanford, CA, in 1974, and the Ph.D. degree from the University of Illinois, Urbana-Champaign, in 1977, all in electrical engineering.

He is currently an Associate Professor of Electrical Engineering and Computer Science at Rice University, Houston, TX. This summer he was on leave at the IBM Thomas J. Watson Research Center, Yorktown Heights, NY. From 1976 to 1982 he was an Assistant Professor with the School of Electrical Engineering at Purdue University, West Lafayette, IN. His current interests include multiprocessor and pipelined computer systems, memory organizations, performance evaluation, operating system, and VLSI computing structures.

Dr. Briggs is a member of the Association for Computing Machinery and the IEEE Computer Society.

King-Sun Fu (S'56-M'59-SM'69-F'71) received the Ph.D. degree in electrical engineering from the University of Illinois, Urbana, in 1959.

He is presently a Goss Distinguished Professor of Engineering and Professor of Electrical Engineering at Purdue University, West Lafayette, IN. He received the Herbert N. McCoy Award for Contributions to Science in 1976, the Outstanding Paper Award of the IEEE Computer Society in 1977, and the 1981 ASEE Senior Research Award. He is the Vice President for Publications of IEEE Computer Society, an Associate Editor of IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS, *Pattern Recognition*, *Computer Graphics and Image Processing*, *Journal of Cybernetics*, and *Information Sciences*, and a member of the Editorial Board of the *International Journal of Computer and Information Sciences*. He is the first President of the International Association for Pattern Recognition (IAPR). He is author of the

books *Sequential Methods in Pattern Recognition* and *Syntactic Methods in Pattern Recognition*, published by Academic Press in 1968 and 1974, respectively, *Statistical Pattern Classification Using Contextual Information*, published by Wiley in 1980, and *Syntactic Pattern Recognition and Applications*, published by Prentice-Hall in 1982.

Dr. Fu is a member of the National Academy of Engineering and Academia Sinica, and is a Guggenheim Fellow.

Kai Hwang (S'68-M'68-SM'81) received the Ph.D. degree in electrical engineering and computer science from the University of California, Berkeley, in 1972.

He is presently an Associate Professor of Computer Engineering, School of Electrical Engineering, Purdue University, West Lafayette, IN. During 1981 and 1982, he visited Academia Sinica and National Taiwan University as a Visiting Professor in Computer and Information Engineering. His current research interest lies mainly in advanced computer architecture including array/pipeline vector computers, multiprocessor and computer networks, VLSI computing structures, dataflow and supercomputer applications. He has published numerous technical papers and book chapters in the areas of computer architecture, arithmetic systems, sequential machines, file structures, performance modeling, and parallel processing, and is the author of *Computer Arithmetic: Principles, Architecture and Design* (New York: Wiley, 1979). He also coauthored *Computer Architecture and Parallel Processing* to be published in the McGraw-Hill Computer Science Series.

Dr. Hwang is a distinguished visitor of IEEE Computer Society and a member of the Association for Computing Machinery.

Benjamin W. Wah (S'74-M'77-S'78-M'79) received the B.S. and M.S. degrees in electrical engineering and computer science from Columbia University, New York, NY, in 1974 and 1975, and the M.S. and Ph.D. degrees in engineering from the University of California, Berkeley, in 1976 and 1979, respectively.

Currently, he is an Assistant Professor of Electrical Engineering at Purdue University, West Lafayette, IN. His research interests include parallel computer architecture, distributed computer systems, and theory of computing.

PICCOLO Logic for a Picture Database Computer and Its Implementation

KAZUNORI YAMAGUCHI AND TOSIYASU L. KUNII, MEMBER, IEEE

***Abstract*—The logic named PICCOLO for a picture database computer and its implementation is presented. The logic is shown to have three major advantages. One advantage is that the computer design based on this logic can handle a universal variety of pictorial data structures. Another advantage is that a set of data generated by rules such as texture distribution rules is stored in the picture computer efficiently. The third advantage is that this logic can serve as the basis of a logic for parallel processing machines. For implementation of the logic, a new methodology named architecture engineering is introduced as an architecture/design oriented methodology. Implementation case studies show the usefulness of the methodology. Two implementations on an abstract machine that are also on a parallel LISP machine are reported.**

***Index Terms*—Architecture engineering, language directed machine, parallel processing, PICCOLO, picture database, relational model, VLSI technology.**

Manuscript received January 11, 1982.

The authors are with the Department of Information Science, Faculty of Science, University of Tokyo, Hongo, Tokyo 113, Japan.

I. INTRODUCTION

RECENTLY, much data have been handled in a form of pictures or images. LANDSAT remote sensing and computerized tomography are examples of data gathering intensive applications. Another group of examples are data generation intensive applications such as CAD (computer-aided design) and CAI (computer-assisted instruction). By storing these data in an integrated database, the handling and sharing of these data become much easier. Recently, several works have been done for constructing such database systems called *picture database systems*. INFADS by Kunii [1], [2], GADS by Carlson [3], GRAIN by Chang [4], and QBPE (query-by-pictorial-example) by Fu [5] are examples aiming at a picture database system. These database systems are founded on the relational model because the other data models such as a network data model and a hierarchical data model lack the data independence which is one of the essential requirements for a database system. However, the relational model again lacks the capability to handle a pictorial data