Reconfiguration in the Low and Intermediate Levels of the Image Understanding Architecture

Charles C. Weems, Deepak Rana
David B. Shu, J. Gregory Nash

COINS TR 90-10

February 1990

Reconfiguration in the Low and Intermediate Levels of the Image Understanding Architecture

Charles C. Weems, Deepak Rana
Department of Computer and Information Science
University of Massachusetts
Amherst, MA 01003

David B. Shu, J. Gregory Nash
Hughes Research Laboratories
3011 Malibu Canyon Road
Malibu, CA 90265

ABSTRACT

This paper examines the architectural features of the Image Understanding Architecture that support reconfiguration in its low- and intermediate-level processors. At the low level a variation of the reconfigurable mesh, called the Coterie network is employed to achieve a mode of parallelism that we term "multiassociative." Processors at the intermediate-level are connected via a fully reconfigurable network that is centrally controlled. In addition to the description of these two networks, several examples are presented that illustrate their use.

INTRODUCTION

The motivation for building a "vision machine" or image understanding architecture stems from the need to support a diverse set of complex operations on massive amounts of data at high speeds, and under hard real-time constraints. The design of such a machine must take into consideration the architectural requirements for integrated real-time vision in terms of the type of processing element, control of processing, and communication between processing elements.

The construction of a symbolic description of the environment depicted in an image involves a variety of algorithms and data structures that must be employed at different levels of computational granularity. There are basically three types of computation required with two of these levels obvious: processing of sensory data and processing of world knowledge. The necessity of an intermediate level of processing has been motivated in [Draper, 1988], among others.

Image interpretation may thus be characterized as involving three different levels of processing, each with its own specific class of information,

control and communication. Additionally, those levels must be able to interact through bottom-up transfers of information and top-down control of processing. The low, intermediate, and high levels of representation and processing, together with our understanding of how these levels interact, provides the basis for the design of the Image Understanding Architecture.

OVERVIEW OF THE IMAGE UNDERSTANDING ARCHITECTURE

The Image Understanding Architecture represents a hardware implementation of the three levels of abstraction inherent in our view of computer vision. It consists of three different, tightly coupled parallel processors. These are the Content Addressable Array Parallel Processor (CAAPP) at the low level, the Intermediate Communication Associative Processor (ICAP) at the intermediate level, and the Symbolic Processing Array (SPA) at the high level (Figure 1). The CAAPP and ICAP levels are controlled by a dedicated Array Control Unit (ACU) that takes its directions from the SPA level. In each layer of the IUA the processing elements are tuned to the computational granularity and algorithms required by that particular level of abstraction.

We are currently building a 1/64th slice of the IUA as a proof-of-concept demonstration. The discussion that follows describes the full IUA, except where it is specifically noted that a feature pertains only to the prototype.

At the high level, the IUA is purely a MIMD parallel processor. Additionally, the intermediate and low levels of the IUA may be treated in a variety of modes of parallelism. These include the CAAPP operating in pure SIMD or multiassociative mode, and the ICAP operating in synchronous-MIMD or pure MIMD mode. A brief explanation of how the multiassociative and synchronous-MIMD modes differ from the familiar SIMD and MIMD modes is required. In multiassociative mode, all processors receive the same instruction stream but execute in disjoint SIMD groups, with each group able to operate on locally broadcast values, and to locally compute its own summary values in parallel with all other groups. This allows different parameters to be employed in processing disjoint portions of the image, with all portions being processed simultaneously. In synchronous-MIMD mode, the programming paradigm is more like SIMD than MIMD: the ICAP processors execute the same program but have their own instruction pointers so that they can branch independently, and globally synchronize for each stage of processing. Synchronous-MIMD has the advantage of being as simple to program as a SIMD system but without the time penalty, usually

encountered in SIMD systems, of having to sequentially execute all of the paths in a branching control structure.

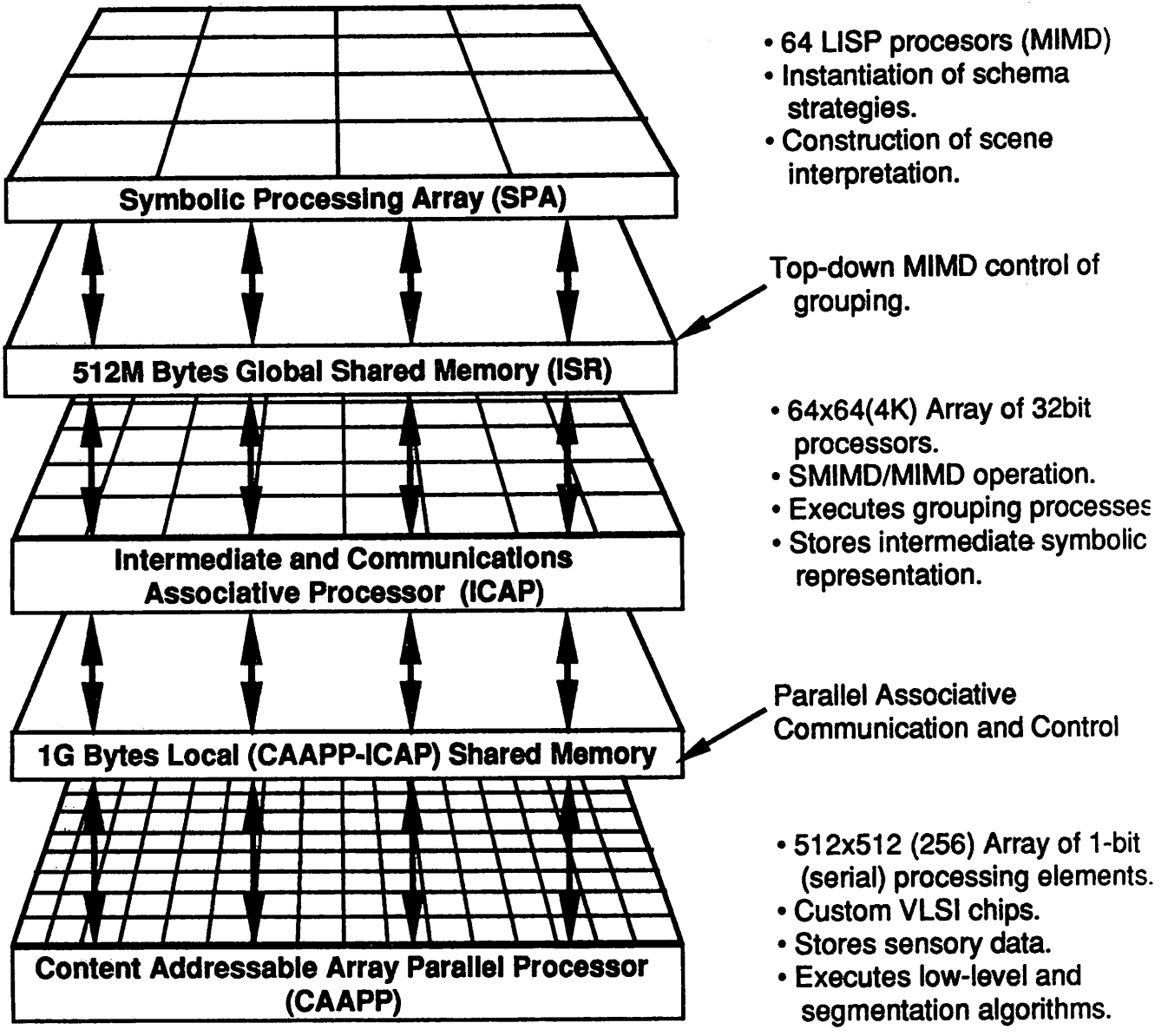

Figure 1. IUA Overview

THE LOW-LEVEL PROCESSOR (CAAPP)

The CAAPP is a 512 X 512 square grid array of custom 1-bit serial processors intended to perform low-level image processing tasks. It is similar in many ways to CLIP-4 [Duff, 1978], MPP [Batcher, 1980], DAP

[Hunt, 1981], GRID [Arvind, 1983], GAPP [Davis, 1984], and the Connection Machine [Hillis, 1986]. However, its architecture is especially oriented towards associative processing with an emphasis on fast global summary feedback mechanisms. The CAAPP is also specifically designed to interact with the ICAP in a tightly coupled fashion for both bottom-up and top-down processing. Thus, the CAAPP has been tailored to permit flexible control, to provide rapid feedback to the controlling processes to enable real-time response to image properties, and to integrate fully into a hierarchically organized vision architecture.

The CAAPP processing elements are linked through a four way (S,E,W,N) communication grid. Each processor can execute an instruction in 100 nanoseconds and contains 5 one-bit registers, an ALU, and 320 bits of RAM that acts as an explicitly managed cache memory. Each element has access to a 32K-bit backing store memory that is dual-ported with the ICAP. The CAAPP chip contains 64 processing elements, is built in 2-micron CMOS via the DARPA MOSIS facility and contains roughly 107,000 transistors. A new version of the chip is also under construction in 1.2 micron CMOS that will contain 256 processors.

One of the principle feedback mechanisms from the CAAPP to the controller is the array-wide logical OR output, called Some/None, which indicates whether any CAAPP cells are in a given state represented by their response bit. It is mentioned here because a similar mechanism figures prominently in the multiassociative processing mode that will be discussed shortly. The Some/None operation is also useful for determining the maximum or minimum value in a selected set of processors.

Communication among CAAPP cells may take place in four different ways. One way is through global feedback and rebroadcast. A second way is via the ICAP: CAAPP data is tranferred to the backing store and the ICAP moves it across the array and places it in the backing store of the appropriate CAAPP cell. A third way uses the nearest neighbor (S,E,W,N) mesh, which allows a CAAPP processor to read a bit from up to two of its neighbors at once. This is similar to the network employed in other mesh-connected SIMD parallel processors. The remaining communication mechanism involves a reconfigurable variation on the mesh, described in the next section, and is a major focus of this article.

THE COTERIE NETWORK

The fourth means of communication among CAAPP processors involves a new and powerful variation on the nearest neighbor mesh called the Coterie network; which was developed in response to the requirement for non-local

communication that was evident in various low-level vision algorithms. One approach to providing non-local communication is to add another network, such as a hypercube or pyramid to the array. However, this significantly increases the complexity and cost of the architecture. Instead, we sought to augment the mesh network with as little extra cicuitry as needed to support the required forms of communication.

It is fairly obvious that an electrically switched mesh allows signals to be sent between distant processors. For example, if one considers a row or column of processors as a bus, a processor could open a switch in one direction, breaking the bus, and then transmit a message in the opposite direction that would propagate at electrical speed as far as the next break. Processors that have not broken the bus are listeners. This scheme is similar to those proposed by [Kumar, 1985], [Miller, 1987], and [Li, 1987], and is a generalization of the propagate operation in the CLIP-4 [Duff. 1978], and the "flash-through" mode of the ILLIAC III [McCormick, 1963]. Because all of the processors in a bus segment can listen to the message being transmitted, there is also some similarity to a broadcast protocol multiprocessor [Levitan, 1984].

We initially considered an implementation with chained pass transistors and buffers, but found the propagation speed to be quite slow. A practical implementation requires the use of precharging and inverted logic to eliminate the buffers. However, once the buffers have been removed from the circuitry, there is no longer any need to restrict a bus to a single talker. If multiple processors output a bit onto a bus segment, the wired-OR of those signals can be read back. To those familiar with associative processors [Foster, 1976], it is immediately obvious that such an operation is equivalent to a Some/None test for responders, except that it is performed independently within each segment.

There is also no need to restrict the buses to rows and columns. They can be any contiguous group of processors in the mesh. In a vision algorithm, each region in an image could be electrically isolated from its neighbors, allowing local broadcast and Some/None operations to occur simultaneously in all regions (see Figure 2). This network has properties that are significantly different from the usual mesh. Because the processors in a group share common properties and purposes, we call the group a Coterie, and hence the name Coterie Network.

For example, suppose that an image is divided into a large number of regions, and that we wish to determine some attribute for each one. In a typical SIMD architecture this would be done by sequentially selecting each region for analysis or in parallel by complex communication between

neighbors where the attribute is computed via a propagating wave that checks region labels at each step. However, using the coterie network, all regions can perform their own local evaluation in parallel without having to check region labels after one initial step of neighbor comparison.

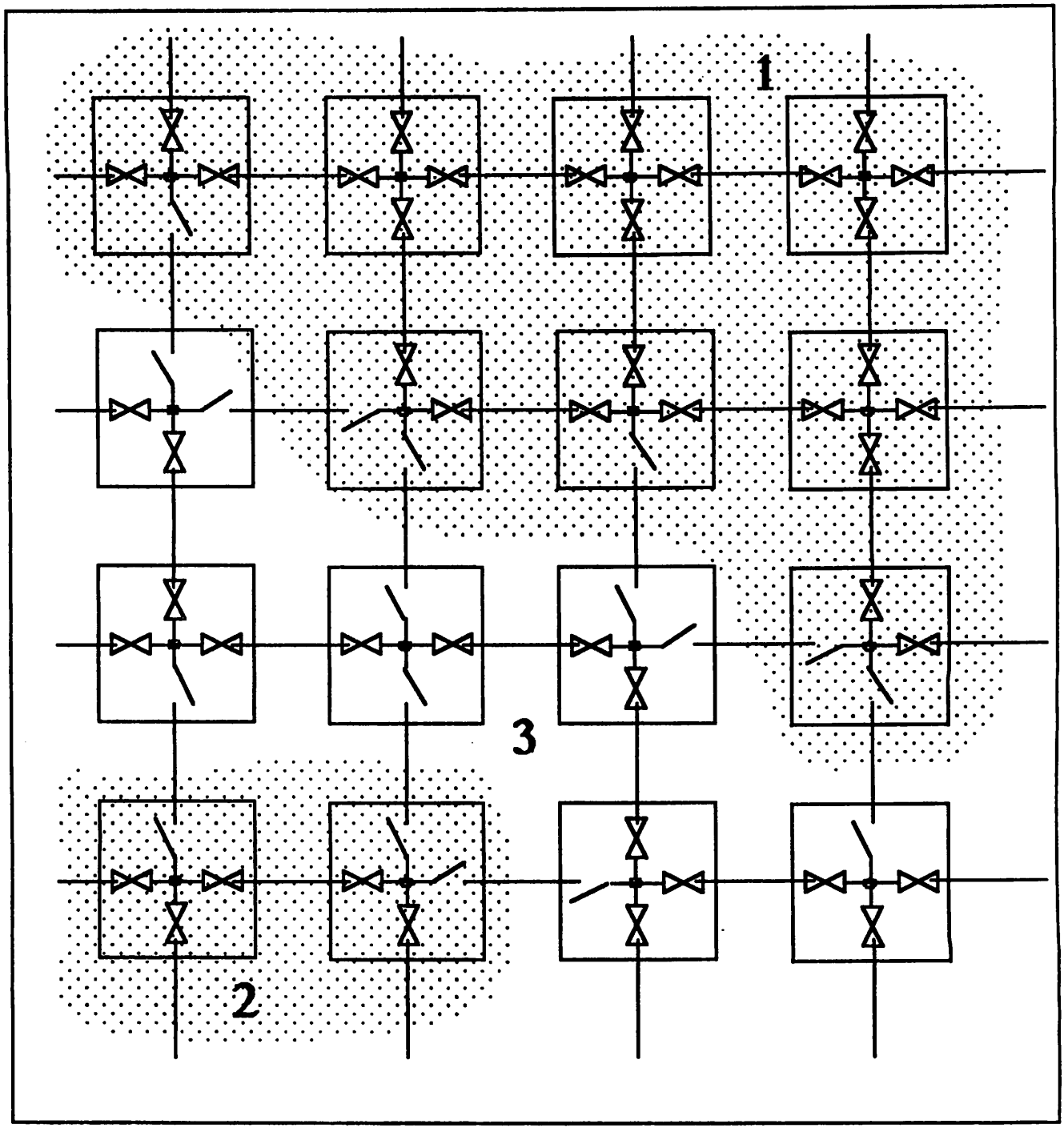

Figure 2. Isolated Processor Groups Corresponding to Regions in an Image

An important result of the ability to test for Some/None in the coteries is that their maximum or minimum values can be determined; which can then be used to label connected components (a frequently used operation in vision). Our simulations show that connected component labelling can be performed in this network in roughly 50 microseconds on a 512 by 512 array, assuming a 100 nanosecond cycle.

In addition to its associative feedback function, the Coterie network can be used to broadcast a value from a single processor to every member of the coterie. This is accomplished by first selecting a single processor within each coterie, using an associative search operation in parallel within all coteries. Subsequent instructions for placing a value onto the network will only be performed by these selected cells. However, all of the cells will perform the operations for reading the value that is on the network. The Coterie Network has many other uses, including matrix arithmetic, FFT, convex hull computation, simulating a pyramid processor, etc.

In our current design, the actual implementation permits signals to cross orthogonally or bypass a node diagonally (see Figure 3). Thus, Coteries need not be physically adjacent to be linked. Note that the Coterie Network is separate from the nearest neighbor mesh, which we refer to as the SEWN Mesh.

The Coterie Network results in a new mode of parallelism that falls between SIMD and MIMD. While there is still a single instruction stream, broadcasting of data values and collection of summary information are no longer restricted to a central entity; a capability that has previously been restricted to MIMD architectures. It may be noted that the Connection Machine also has such capabilities, but the Connection Machine is not a purely SIMD system. Each Connection Machine router node is a finite state machine that operates semi-autonomously. Its router can thus be considered a mono-algorithmic MIMD processor that is attached to the SIMD processor array. The Coterie Network, on the other hand, contains no active logic and operates only in response to broadcast instructions.

ALGORITHMS FOR THE COTERIE NETWORK

Creation of a set of coteries typically begins with opening all of the switches that link processors. Using the SEWN Mesh, the processors compare their own values with the values of their neighbors. They then close the switches that connect them to neighbors with similar properties, leaving open the switches that would connect them to dissimilar neighbors. Of course similarity can be defined by an operation such as a global

broadcast of a threshold and a local comparison. In this way, processors with similar properties establish independent coteries. Because the CAAPP processors can save and restore the switch settings that make up a set of coteries, it is possible to reconfigure the Coterie Network from one processor interconnection pattern to another by broadcasting a single instruction.

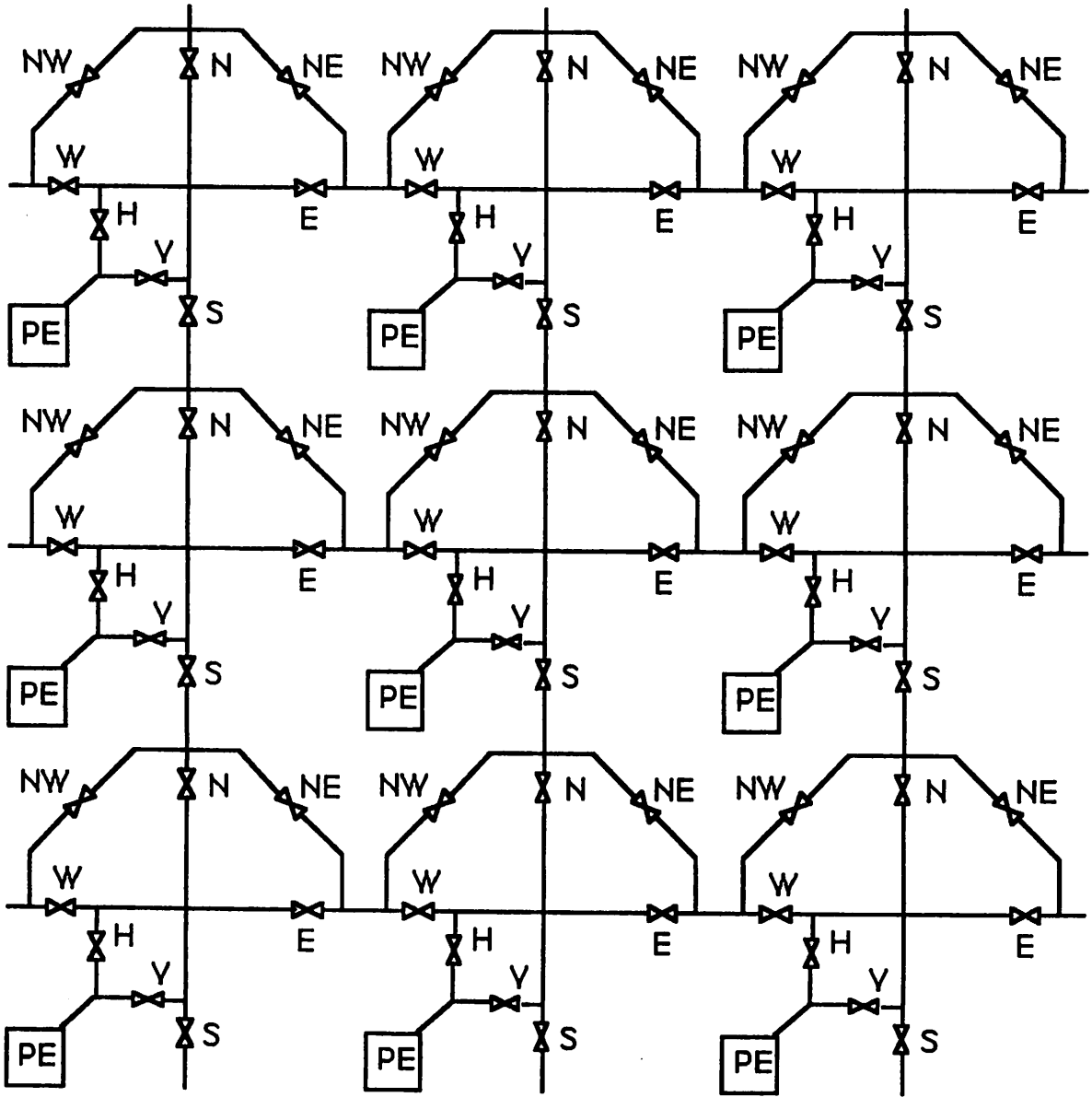

Figure 3. A Three by Three Coterie Network, Showing Crossing and Bypass Links

Label Connected Components

The local associative Some/None operation provided by the Coterie Network is demonstrated by this algorithm. Because finding a maximum uses only broadcast and Some/None feedback, it can be performed locally within a coterie and in parallel with every other coterie. This leads to the simple algorithm for computing a connected component labeling of an image shown in Figure 4.

The algorithm begins by loading each processor with its address in the array from the backing store memory, which serves to give each processor a unique number. Next, the Coterie Network switches are opened between processors that are on region boundaries (i.e. between pairs of processors that have different values), establishing a coterie for each image region. Lastly, all regions in parallel determine their local maximum address value as follows.

The high order bit of the address is loaded into the response register of all active cells and transmitted over the Coterie Network. If any cells in a coterie have their high order bit set, then they are candidates for the maximum value, in which case, any cells in the coterie that have a zero in their high order bit are deactivated. However, if no cells in a coterie have their high order bit set, then none are deactivated because they are all still potential candidates. This process repeats with each successively lower order bit in the address. When the low order bit has been processed, only those cells that contain the maximum value will remain active.

For each iteration, the members of each coterie save the Some/None response so that the maximum value for each coterie is available in all members at the conclusion of processing. Because this value is different for every region, the result is that each connected group of processors is assigned a unique label that is common to every processor within a group.

Create Border Corner Lists

In addition to performing associative Some/None operations, the Coterie network may be used to pass messages across the mesh by providing direct links between non-adjacent processors. In the algorithm shown in Figure 5, it is assumed that some corner detection operation has been performed on the borders of regions in the image. The result is a sparse set of processors that are labelled as corners (ie. those processors whose Corner_Tag field is set to true). One useful feature that can be extracted for each region is a list of the positions of its corners. The following

algorithm forms the corner lists for all regions in parallel. It begins by making each region an independent coterie, using the connected components labelling algorithm. A single cell in each coterie is selected as the coterie leader. In this case, the chosen cell is the member whose cell address equals the region's component label. The leader is responsible for collecting the corners for its region, and passing them to the ICAP which stores them in a list.

```
Load_Processor_Addresses                                    {Intialize}
Coterie_Switches := All_open_but_H_and_V
For Neighbor := North TO West DO                    {Close switches to}
    IF Neighbor.Field = Field THEN                          {= neighbors}
        Coterie_Switch[Neighbor] := Closed
FOR Bit_Num := Address_Length-1 DOWNTO 0 DO              {Coterie     }
    Response := Address[Bit_Num]                         {select Max}
    IF Coterie_Some THEN
        Activity := Response
    Component_Label[Bit_Num] := Coterie_Some!
```

Figure 4. Connected Component Labelling using the Coterie Network

The first corner is determined by selecting the corner cell in each coterie with the maximum address. As part of this process, the coterie leader learns that cell's address and passes it to the ICAP processor associated with the CAAPP chip containing the leader. The selected corner cell is then shut off and the process is repeated so that the next corner is selected. The loop ends when there are no more corners to select, at which point every corner will have been passed to the ICAP by the coterie leader.

This algorithm causes the corner lists to be created in reverse raster-scan order, which is adequate if the regions are simple convex figures. However, for more complex regions, it may be difficult to reconstruct the shape of a region given a corner list in this order. A better arrangement is to list the corners in clockwise (or counterclockwise) boundary traversal order (i.e. the order in which corners would be encountered as the cells at the boundary of the region are traversed, starting from some arbitrary boundary point). The coterie network can also be used to accomplish this task. Although the basic concept for doing this is quite simple, in practice it is complicated by considerations of regions that are one-pixel wide and regions that completely enclose other regions. Because the algorithm is more complex, it is only discussed here in general terms; the detailed discussion will be left to a future paper.

```
Label_Connected_Components
Start_ICAP(Corner_List_Builder)   {A process is started in the ICAP
                                   that will respond to Signal_ICAP
                                   operations by picking up a corner
                                   from the backing store }
Activity := 1!
Leader := Address = Component_Label
Backing_Store_Write(Leader)          {Copy coterie leader tags to
                                                backing store}
Response := Leader
Latch_Local_Count                 {Count leaders in each CAAPP chip}
Signal_ICAP                         {Each ICAP reads its local count
                                      and determines which CAAPP
                                      PEs are coterie leaders}
Activity_&_Response := Corner_Tag
WHILE Some Do                                  {Select greatest corner}
    FOR Bit_Num := Address_Length-1 DOWNTO 0 DO
        Response := Address[Bit_Num]
        Next_Corner[Bit_Num] := Coterie_Some!
        IF Next_Corner[Bit_Num] THEN
            Activity := Response
    Corner_Tag := False              {Disable corner PE when found}
    Backing_Store_Write(Next_Corner)    {Copy corner's address to
                                                backing store}
    Signal_ICAP                      {ICAP picks up a corner address
                                      from each coterie leader}
    Activity_&_Response := Corner_Tag!
END WHILE
```

Figure 5. Creating Lists of Border Corners

As in the preceding algorithm, the first step is to label connected components. After connected component labelling has been performed, each member of a component examines its neighborhood to determine whether any neighboring cell has a different component label. Any cell that has a neighbor belonging to a different region is at the boundary of its own

region. In the simplest case, the cells that are at a region's boundary form a chain that is a one-pixel wide closed loop. Each cell in the chain will have two neighbors; one in the clockwise direction and the other in the counterclockwise direction around the loop. The cells that make up a boundary chain can then set their coterie switches so that the chain becomes a separate coterie. (Some of the complexity of the actual algorithm stems from the situations in which a boundary chain is not a simple closed loop and how the different cases are handled.)

A leader is selected for each of the boundary-chain coteries. Each leader opens the coterie switch connecting it to its counterclockwise neighbor in the loop. The loop is thus transformed into an open figure with the leader at one end. Every cell in the chain that is also tagged as a corner now opens the switch connecting it to its clockwise neighbor in the chain. Next, each leader broadcasts a bit to its coterie. Because the corner cells have broken the coterie, the bit will only reach the first corner clockwise along the boundary from the leader. That corner is then activated and transmits its address back along the coterie to the leader which subsequently passes the address to the ICAP. The active corner then closes the switch to its clockwise neighbor and deactivates itself so that further broadcasts from the leader will pass through it. The process is repeated until all corner cells have been read out to the ICAP. Note that all region boundaries are being processed simultaneously through their coterie chains.

Region Adjacency Graph

A similar algorithm involves collecting a list of adjacent region labels for each region in an image. This algorithm begins by having every boundary cell get the label of its neighbor that is in another region, using the SEWN mesh. Each region then performs a coterie-select-greatest operation on these region labels, and a region label is output to the ICAP via the coterie leader. All boundary cells that have the selected label are then shut off and the test is repeated to obtain the next region label. The process is complete when there are no more labels to output. Because a region label is the address of the coterie leader for a region, and the ICAP processors are spatially collocated with respect to the CAAPP cells, each ICAP processor can directly compute the ID number of the other ICAP processors that contain descriptions of adjacent regions.

Top-Down Directed Region Merging

Once the ICAP processors have collected the information required to describe a specific type of image event, for example lines or regions, it is common to group or merge some of that information. Suppose that a

decision has been made to merge a small region with an adjacent large region. The merging will be done symbolically in the ICAP, but it may also be desirable to have the CAAPP data represent this change as well. In that case, the ICAP processor that contains the smaller region transmits the new region label to the CAAPP by writing the label into the backing store for the region's coterie leader. The ACU then instructs all coterie leaders that have received new labels to broadcast the new label to their coterie and then resign as coterie leader. The cells that are on the common boundary between the two regions then close their coterie switches so that the two coteries are merged into one. The ICAP processor that was responsible for the smaller region then deletes the region's information from its database.

THE INTERMEDIATE LEVEL (ICAP)

The ICAP is designed to manipulate tokens (symbolic descriptions of extracted image events and their associated attributes) at the intermediate level and to support data base functions that allow access to these tokens by grouping processes running on the ICAP and by symbolic interpretation processes, running on the SPA processors. For example, the recognition of a house roof in an image may require the ICAP to group together long, straight, parallel lines, and then to extract parallelograms that are candidate roof outlines. Should the need arise, the results of further processing in the CAAPP can be integrated with the representation in the ICAP because the ICAP representation is in approximate registration with the original image events in the CAAPP.

The ICAP is a square grid (64 X 64) array of Texas Instruments TMS320C25 16-bit digital signal processor (DSP) chips. Each of the 4096 ICAP processors consists of a CPU, 256K bytes of local RAM, 384K bytes of dual-ported memory for interacting with the CAAPP and SPA, and network communication hardware. The ICAP processors operate at 5 million instructions per second and can perform a 16-bit multiply-and-accumulate operation in a single instruction time. In addition to its speed, a digital signal processor was chosen at the ICAP level because its instruction set and arithmetic capabilities are well suited for performing computations in spatial geometry. Three-dimensional geometric projections, computing distances, and matching operations are common operations that may be needed at the intermediate level of vision processing.

Control of the ICAP is provided by the ACU (in Synchronous-MIMD mode) and by the SPA (in MIMD mode). Once sensory events have been extracted and represented symbolically at the intermediate level in the ICAP (and continue to evolve as grouping operations take place), each of the SPA

processors may then query the ICAP in parallel to establish and verify hypotheses. The ICAP provides three different global OR outputs available to the controller that can be used to determine the status of processing in the ICAP array. The choice of meaning for each signal is left up to the programmer. For example, the programmer may choose to have them indicate completion of a task in the ICAP array with or without exceptions. Another use is as an associative Some/None mechanism. A global summation mechanism is also provided that uses the global count hardware in the CAAPP to form a sum of an 8-bit value from each ICAP processor.

Each DSP chip has one serial input port and one serial output port, each of which is capable of a 5M-bit/sec data rate. These serial ports provide the basis for interprocessor communication within the ICAP and as such they form the set of data sources and sinks that are linked by the network described below. In the IUA prototype, which has only 64 ICAP processors, the bit-serial I/O links between the processors are connected through a centrally controlled 64 X 64 bit-serial crossbar switch. Thus it is possible to establish any point-to-point network topology in the prototype ICAP. Various methods of extending the ICAP communications network to the full-size ICAP array are under consideration.

ICAP INTERCONNECTION NETWORK

The ICAP operates in two distinct modes of control. When working with the SPA, the ICAP operates in MIMD mode, but when interacting with the CAAPP, it is most efficient to keep the ICAP processors roughly synchronized. The remainder of this discussion will focus on the latter mode, which as mentioned previously, is called synchronous MIMD, and has a programming model that is similar to SIMD, except that branches can be performed simultaneously rather than sequentially. During synchronous MIMD operation, the ACU manages the stages of processing through barrier synchronization points. Communication between ICAP processors also occurs synchronously via a reconfigurable network that is managed by the ACU. A typical scenario involves the ICAP processors communicating via one connection pattern, then synchronizing at a barrier and waiting for the ACU to reconfigure the network before releasing them.

The ICAP connection network is used to set up a connection pattern between the N output ports of the processors and the N input ports of these same processors. The connection network can be programmed on-line, to make a direct link from the output port of any processor to the input port of one or more processors. We have built a custom VLSI chip, called the PARallel COmmunication Switch (PARCOS), which is capable of broadcasting, allowing the connection network to realize any of the

possible N^N mappings of its input ports onto its output ports. All of the processors can send and receive data on their links at the same time. These links can be changed by the ACU at any time.

The 64-input, 64-output connection network for the IUA prototype uses 2 stages of 32 x 32 PARCOS chips. The PARCOS chips are connected to make a 64 x 64 crossbar switch with broadcast capability as shown in Figure 6. A detailed discussion of the network can be found in [Rana, 1988].

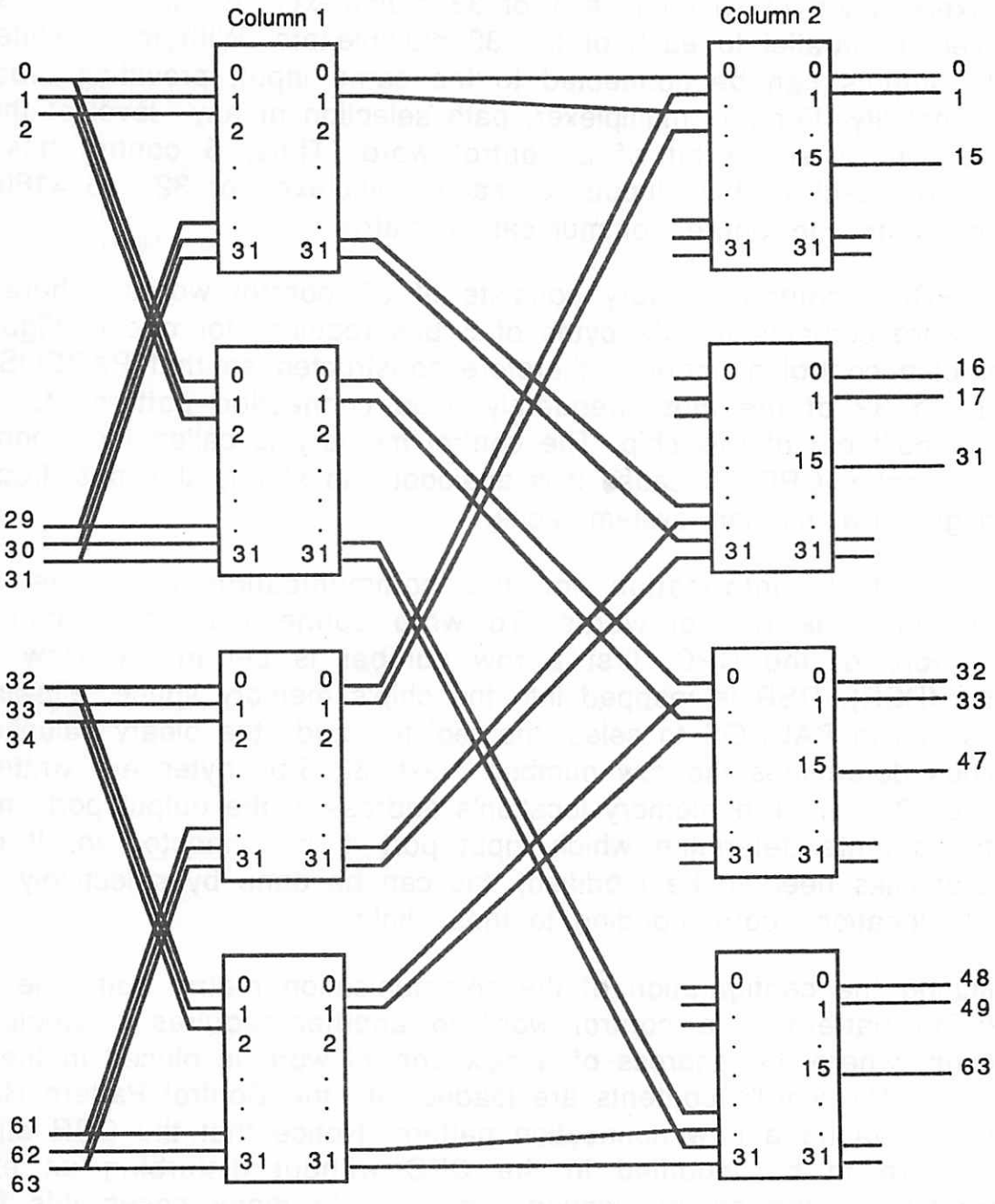

Figure 6. A 64 x 64 Network Built with PARCOS Chips.

The Parallel Communication Switch

The PARCOS chip consists of a communication matrix of 32 bit serial inputs and 32 bit serial outputs, a control memory, a set of registers and associated read/write circuitry. The PARCOS chip organization is shown in Figure 7. Multiple PARCOS chips can be used to build larger connection networks, such as the 64 x 64 network in the IUA prototype.

The communication matrix of PARCOS consists of 32 tree-structured multiplexers, each of which is a 1 of 32 multiplexer. All 32 input lines are connected in parallel to each of the 32 multiplexers. With this architecture, multiple outputs can be connected to the same input, providing broadcast mode capability. For any multiplexer, path selection at any level of the tree is done with a single bit of a control word. Thus, 5 control bits are required to select 1 of 32 inputs for each multiplexer, or 32 x 5 =160 bits for configuring the entire communication matrix.

The PARCOS control memory consists of 32 control words, where each control word contains the 32 bytes of 5 bits required for one configuration. The on-chip control memory is therefore constructed so that PARCOS can hold up to 32 of the most frequently used connection patterns for larger networks built out of this chip. The control memory is called the Connection Pattern Cache (CPC), because it is analogous to storing the most frequently used pages in a memory system cache.

The connectivity information for the communication matrix is stored serially into the control words. To write connectivity information in a control word of the CPC, first a row number is set in the Row Select Register (RSR). RSR is mapped into the chip's memory space, allowing the address bus in PARCOS to select the register, and the binary value on the data lines determines the row number. Next, 32 5-bit bytes are written into addresses 0 - 31. The memory location's address is the output port number and its contents determine which input port it is connected to. If only a subset of links need to be modified, this can be done by selectively writing only into locations corresponding to those links.

Reswitching the configuration of the communication matrix from one stored connection pattern in a control word to another requires a single write instruction, where the address of a new control word is placed in the RSR, and the control word's contents are loaded into the Control Pattern Register (CPR), activating a new connection pattern. Notice that the CPR allows a control word to be modified in the CPC without disturbing an existing configuration in the communication matrix. In many cases this feature

allows the time to write a new connection pattern from the ACU into the CPC to be hidden while the processors are working on an algorithm.

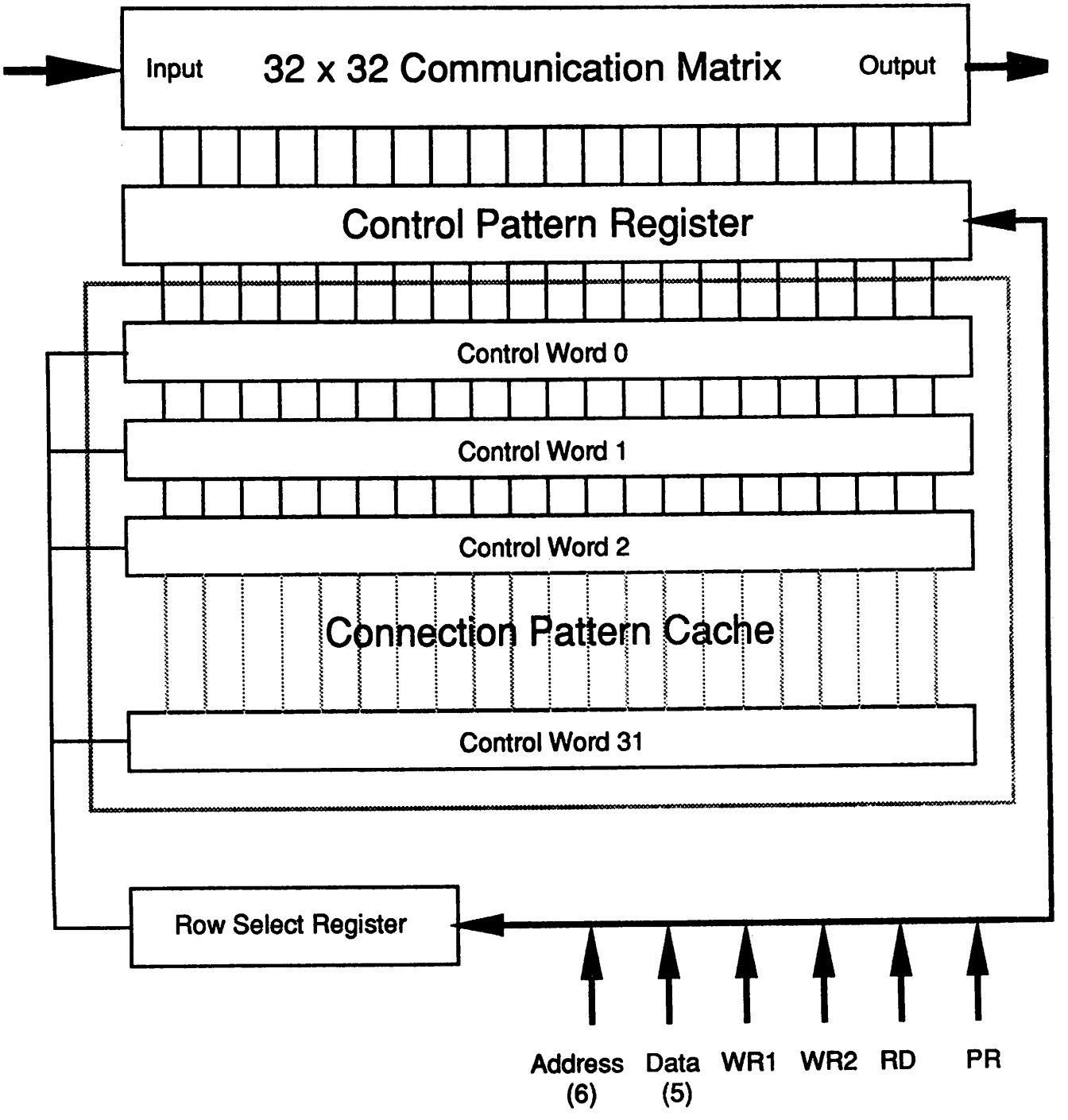

Figure 7. PARCOS Chip Organization

PARCOS is implemented on a single 84 pin, 50,000 device, VLSI chip. It is a full custom design, built out of a 2 micron, P-Well, double metal, scalable

CMOS technology available through MOSIS. Each CPC memory bit is a 6 transistor static RAM cell. The worst case delay in broadcast mode from one input to 32 outputs is less than 50nS.

Future Versions of PARCOS

The design of the PARCOS chip was limited by the number of pins and not by the silicon area. A study for redesigning the PARCOS chip is underway with the goal of providing a 64 x 64 communication matrix with more than one hundred control words. Also, mechanisms will be provided for self routing in these chips in $\theta(1)$ time. A 64 x 64 single chip implementation will allow us to build a 4096-input, 4096-output network by connecting 512 of these chips in a modified 4-stage, strictly non-blocking Clos' [Clos, 1953] topology or by connecting only 192 of these chips in a 3-stage, rearrangeably non-blocking Benes' [Benes, 1962] topology.

Currently it is not possible to directly copy or combine connection patterns in the CPC. Adding the ability to copy one CPC connection pattern into another under the control of a mask register will allow us to build new connection patterns from old ones, which will reduce the time required to create a new pattern in certain cases.

SUMMARY

The Image Understanding Architecture consists of three different parallel processors. Two of these, the low-level (CAAPP) and intermediate-level (ICAP) processors use reconfigurable networks to support intra-level communication between their respective processors.

In the case of the CAAPP, the network is a generally reconfigurable mesh, called the Coterie Network, which permits contiguous processors to be separated into arbitrary, isolated groups called coteries. The coteries are capable of supporting a mode of parallelism called multiassociative processing in which all processors receive the same instruction stream, but the coteries can simultaneously perform independent local broadcast *and* summary operations. The coterie network is especially suited for operations on image components. For example, it can be used to label connected components in 50 microseconds in a 512 x 512 image.

At the intermediate level, a centrally controlled crossbar network utilizing a Connection Pattern Cache is employed when the ICAP processors are operating in synchronous mode. If a new connection pattern for the network has been previously stored in the cache, the network can be reconfigured with a single instruction. Storing a new pattern requires 64 instruction

cycles. The crossbar is built from custom VLSI chips that implement a 32 x 32 network. The custom chips can be connected together in several ways to build larger networks. We are currently working on a version of the chip that will support asynchronous, distrubuted control communication between ICAP processors when they are operating in MIMD mode.

ACKNOWLEDGEMENTS

This work was funded in part by the Defense Advanced Research Projects Agency under contract numbers DACA76-86-C-0015, and DACA76-89-C-0016, monitored by the U.S. Army Engineer Topographic Laboratory; and contract number F49620-86-C-0041, monitored by the Air Force Office of Scientific Research; and by a Coordinated Experimental Research grant (DCA 8500322) from the National Science Foundation.

REFERENCES

[Arvind, 1983] Arvind, D.K., Robinson, I.N., and Parker, I.N., A VLSI Chip for Real-Time Image Processing, Proc. IEEE International Symposium on Circuits and Systems, 1983, pp. 405-408.

[Batcher, 1980] Batcher, K. E., Design of a Massively Parallel Processor, IEEE Trans. Comp., Vol. C-29, No. 9, September 1980.

[Benes, 1962] Benes, V.E., "On Rearrangeable Three-Stage Connecting Networks", Bell Systems Tech. Journal, vol. XLI, no. 5, Sept. 1962, pp 1481-1492.

[Clos, 1953] Clos, C., "A Study of Non-Blocking Switching Networks ", Bell Systems Tech. Journal, vol. 32, no. 2, March 1953, pp 406-424.

[Davis, 1984] Davis, R., Thomas, D., Geometric Arithmetic Parallel Processor-Systolic Array Chip Meets the Demands of Heavy-Duty Processing, Electronic Design, October 31, 1984, pp. 207-218.

[Draper, 1988] Draper, B.A., Collins, R.T., Brolio, J., Griffith, J., Hanson, A.R., Riseman, E.M., The Schema System: Knowledge Based Vision, International Journal of Computer Vision, Vol 2, Number 3, December, 1988.

[Duff, 1978] Duff, M.J.B., Review of the CLIP Image Proceeding System, Procedings of the National Computer Conference, 1978, AFIPS, pp. 1055-1060.

[Foster, 1976] Foster, C. C., Content Addressable Parallel Processors, New York: Van Nostrand Reinhold, 1976.

[Hanson, 1987] Hanson, A.R., and Riseman, E.M., "The VISIONS Image Understanding System.", COINS Technical Report, University of Massachusetts at Amherst. 1987.

[Hillis, 1986] Hillis, D.W., The Connection Machine, MIT Press, Cambridge, 1986.

[Hunt, 1981] Hunt, D.J., The ICL DAP and its Application to Image Processing, in Languages and Architectures for Image Processors (M.J.B. Duff, S. Levialdi eds.), Academic Press, London, 1981.

[Kumar, 1985] Kumar, V.K.P., Raghavendra, C.S., Array Processor with Multiple Broadcasting, Proc. 12th Annual Symp. Computer Architecture, Association for Computing Machinery Press, 1985.

[Lee, 1988] Lee, I., and Smitley, D., "A synthesis algorithm for reconfigurable interconnection networks", IEEE Transactions on computers, June 1988, pp 691 - 699.

[Levitan, 1984] Levitan, S. P., Parallel Algorithms and Architectures: A Programmers Perspective, Ph.D. Dissertation, Computer and Information Science Department, also, COINS Technical Report 84-11, University of Massachusetts at Amherst, May 1984.

[Li, 1987] Li, H., and Maresca, Polymorphic Torus Network, Proc. Intl. Conf. Parallel Processing, Penn State Press, State College, PA, 1987.

[McCormick, 1963] McCormick, B.T., The Illinois Pattern Recognition Computer -- ILLIAC III, IEEE Trans. on Elect. Computers, Dec., 1963, pp. 791-813.

[Miller, 1987] Miller, R., Kumar, V.K.P., Reisis, D., Stout, Q.F., USC Tech. Rept. IRIS #229, Univ. of Southern California, Los Angeles, CA, 1987.

[Rana, 1988] Rana, D., Weems, C.C., and Levitan, S.P., "An easily reconfigurable circuit switched connection network", Proc 1988 IEEE Int Symp on Circuits and Syst, June 1988, pp 247 - 250.

[Weems, 1988] Weems, C.C., Levitan, S.P., Hanson, A.R., Riseman, E.M., Shu, D.B., and Nash, J.G., "The Image Understanding Architecture", International Journal of Computer Vision, Vol 2, Number 3, December, 1988.