THE IMAGE UNDERSTANDING ARCHITECTURE

Charles C. Weems, Steven P. Levitan*
Allen R. Hanson, Edward M. Riseman
Computer and Information Science Department
University of Massachusetts

J. Gregory Nash, David B. Shu
Hughes Research Laboratories

COINS Technical Report 87-76

*Current address: Department of Electrical Engineering, Benedum Hall,
University of Pittsburgh, Pittsburgh, PA 15261

This research was supported, in part, by the Advanced Research Projects Agency of the Department of Defense via Contract No. F49620-86-C-0041 monitored by the Air Force Office of Scientific Research under Contract No. DACA76-86-C0015 monitored by the Engineer Topographic Laboratory

ABSTRACT

This paper provides an overview of the Image Understanding Architecture (IUA), a massively parallel, multi-level system for supporting real-time image understanding applications and research in knowledge-based computer vision. The design of the IUA is motivated by considering 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 IUA integrates parallel processors operating simultaneously at three levels of computational granularity in a tightly-coupled architecture. Each level of the IUA is a parallel processor that is distinctly different from the other two levels, in order to best meet the processing needs at each of the corresponding levels of abstraction in the interpretation process. Communication between levels takes place via parallel data and control paths. The processing elements within each level can also communicate with each other in parallel, via a different mechanism at each level that is designed to meet the specific communication needs of each level of abstraction.

An associative processing paradigm, that provides a simple yet general means of managing massive parallelism, has been utilized as the principle control mechanism at the low and intermediate levels. Control of processing in these levels is based upon their rapid responses to queries, involving partial matches of processor memory to broadcast values. This has been enhanced with hardware operations that provide for global broadcast, local compare, Some/None response, responder count, and single responder select.

The low-level, called the Content Addressable Array Parallel Processor (CAAPP), is a 512×512 array of 1-bit serial processors designed to operate on arrays of pixels and to construct intermediate-level tokens from events in an image. At the intermediate level, an array of 64×64 16-bit processors, called the Intermediate Communications and Associative Processor (ICAP), are used for retrieving, comparing, and matching tokens, computing geometric relationships between tokens, and constructing new tokens that describe more abstract entities. At the high level, called the Symbolic Processing Array (SPA), a set of 64 processors capable of executing LISP programs supports computation involving inference, hypothesis generation and verification, analysis of uncertainty, model-based processing, and control of processing at the lower levels. To demonstrate how the IUA may be used for vision processing, several simple algorithms and a typical interpretation scenario on the IUA are presented.

We believe that the IUA represents a major step towards the development of a proper combination of integrated processing power, communication, and control required for real-time computer vision. A proof-of-concept prototype of 1/64th of the IUA is currently being constructed by the University of Massachusetts and Hughes Research Laboratories.

TABLE OF CONTENTS

1. Introduction....1
2. Parallel Architectures for Knowledge-Based Vision....5
   - 2.1 Architectural Requirements for Vision....5
   - 2.2 Processing Characteristics for Multi-Level Architectures....6
   - 2.3 Communication Between Processing Levels....8
     - 2.3.1 Associative Communications and Control....8
     - 2.3.2 Parallel Data Transfer and and Control Between Levels....10
   - 2.4 Review of Existing Architectures....10
     - 2.4.1 Mesh-Connected Arrays....11
     - 2.4.2 Pyramid Processors....12
     - 2.4.3 Hypercube Processors....14
     - 2.4.4 Mesh-Plus-Hybercube-Connected Arrays....15
     - 2.4.5 Shared Memory Multiprocesors....15
     - 2.4.6 Systolic Processors....17
3. Overview of the Image Understanding Architecture (IUA)....17
   - 3.1 The Three Processing Levels....20
   - 3.2 The CAAPP Processing Elements....21
   - 3.3 Associative Feedback from the CAAPP....22
   - 3.4 Intra-Level Communication Within the CAAPP....24
   - 3.5 The CAAPP Coterie Network....25
   - 3.6 Inter-Level Communication Between the CAAPP and ICAP....27
   - 3.7 Some Example Algorithms for the CAAPP....28
   - 3.8 ICAP Structure and Control....30
   - 3.9 ICAP Communication....31
   - 3.10 The SPA: High Level Processing....32
   - 3.11 The ACU: Controlling the CAAPP and the ICAP....33

4. A Vision Processing Scenario For the IUA.....34
5. Image Understanding Benchmark Performance.....36
6. Current Implementation Status.....38
7. Further Development.....40
8. Conclusion.....40
9. Acknowledgements.....41
10. References.....42

1. INTRODUCTION

Machine vision is one of the most computationally intractable domains of artificial intelligence research. It requires that an interpretation of a changing scene be updated as video frames arrive at 30 frames per second. Each video frame contains three quarters of a million color-intensity data values which comprise the picture elements (pixels) of the image. Performing a single operation on each of these pixels requires executing about 23 million instructions per second just to keep up with the input. Of course, the computation needed to perform image interpretation is much more than one operation on every pixel in an image. Many researchers believe it is in the range of 3 to 5 orders of magnitude more computation. Although "real-time" interpretation does not mean that a full interpretation must be completed with each new frame (because much of the previous interpretation may be re-used, and some vision tasks may not require such frequent updating of an interpretation), nevertheless a very large computational rate is required.

One goal of machine vision is the construction of a symbolic description of the environment depicted in an image. Such an interpretation involves not only labeling certain regions in an image, or locating a single object in the viewed scene, but often requires the construction of a three-dimensional model of the surroundings, with associated identification in the image of the two-dimensional projections of these three-dimensional models. Figures 1 and 2 show a simple example of an interpretation of a house scene via labels of object classes. The immense amount of computation necessary is only one aspect of the problem. A variety of algorithms and data structures must be employed at different levels of computational granularity. In the treatment that follows we will argue that there are three types of computation required. Based upon the past 25 years of research in artificial intelligence and computer vision, two of these levels are obvious: processing of sensory data and processing of world knowledge. The necessity of an intermediate level of processing will be motivated below.

Sensory processing principally involves classical image processing techniques such as contrast enhancement, and computer vision techniques of edge detection, region segmentation, and feature extraction. The principal unit of information that is being processed at the sensory level is the pixel, consisting of the color or intensity values of the image, and possibly depth data for the visible surface element associated with each pixel.

Because of the inherent ambiguities that are present in images of natural scenes, it is rarely possible to construct an interpetation directly from the pixel data. Additional knowledge must be used to reduce local ambiguity and to infer portions of objects that are missing in an image due to effects such as occlusion or shadows. Inference via stored knowledge and the reduction of ambiguity from "low-level" sensory processing are a part of what is referred to as "high-level" or knowledge-based vision processing. Without knowledge-based processing it would be impossible to interpret large portions of many images. The approach to knowledge-based vision that follows is derived from the VISIONS system development project for the analysis of natural scenes at the University of Massachusetts [Hanson, 1974, 1978a, 1978b, 1986; Parma ,1980; Reynolds et al., 1984; Weymouth 1986]. Here we will present an extremely brief overview, and refer the reader to the companion

Figure 1. Raw Image

paper on image interpretation [Draper, 1987].

The successful functioning of an interpretation system involves hypothesizing scene and object parts from the low- and intermediate-level abstractions. These hypotheses are used to access symbolic knowledge structures (called schemas) which capture object descriptions and contextual (relational) constraints derived from prototypical scene situations. The hierarchically organized schemas contain interpretation strategies whose execution involves top-down control of the intermediate grouping strategies. The interpetation relates image-specific events and their descriptions to the elements of world knowledge used to construct the interpretation.

There is no simple computational transformation that will map arrays of sensory data onto the stored symbolic concepts represented in a knowledge base. It has become generally accepted that many levels of representation (data abstraction) and many stages of processing must take place to reliably interpret a scene. We will refer to the set of representations between the sensory data and the symbolic structures associated with objects in the environment as the intermediate level representation or in the VISIONS system terminology, the Intermediate Symbolic Representation (ISR) (See figure 3).

The intermediate level of representation bridges the gap between the low and high levels. At the intermediate level the basic unit of information is a symbolic description

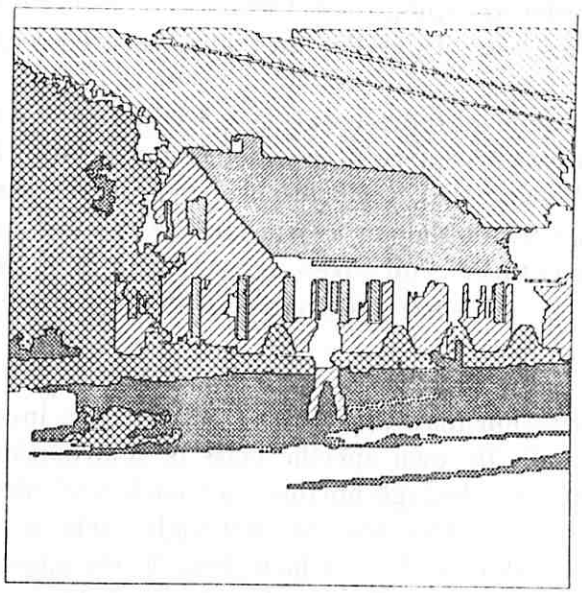


Figure 2a. Example Image Interpretation

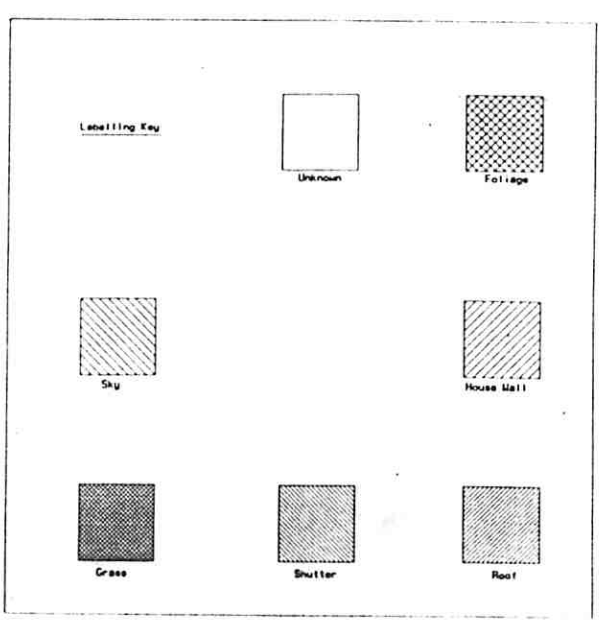


Figure 2b. Labeling Key

of an image event extracted from the image data, and referred to as a token. Examples of token classes (or token types) are lines, regions, and surfaces, and in general any extractable sensory event that is useful for interpretation. Processing at the intermediate level may then involve grouping these token events into more complicated structures such as rectangles, planes, sets of parallel lines, and textured regions.

The intermediate level representation is treated as a symbolic database by the interpretation processes which query it in order to form initial hypotheses about the content of the image. Following this, verification of hypotheses and resolution of conflicting hypotheses often requires further processing in the low and intermediate levels. Additional goal-oriented segmentation and token extraction may be necessary and almost certainly there will need to be goal-oriented control of the grouping processes in order to perform effective object matching.

Image interpretation may thus be characterized as involving three different levels of processing, each with its own specific class of information. 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 those levels interact, provides the basis for the design of our Image Understanding Architecture (IUA) presented in section 3.

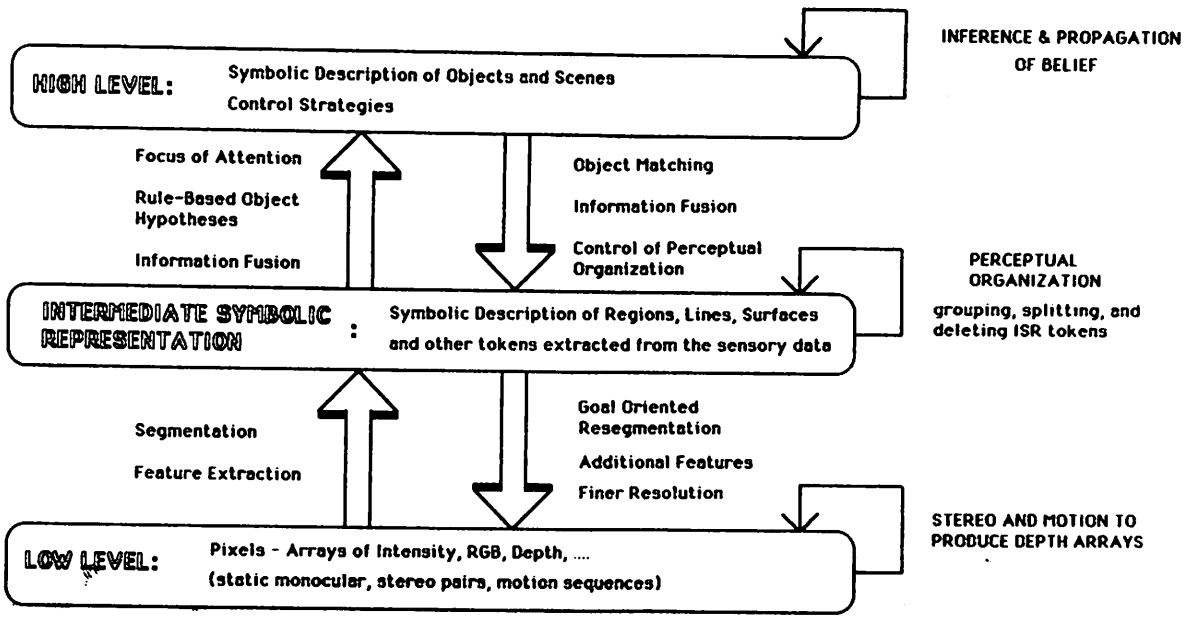


Figure 3. UMass VISIONS Image Interpretation System

2. PARALLEL ARCHITECTURES FOR KNOWLEDGE-BASED VISION

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 to provide an environment for experimentation that minimizes the software effort that is necessary to build a vision system. In this section we discuss the requirements for such an architecture and review some existing architectures and organizations.

2.1 Architectural Requirements for Vision

Some of the architectural issues to be addressed for vision come directly from the specific requirements of the problem as discussed in the introduction:

* The ability to transform pixel data (i.e. an image) into a set of meaningful symbols that describe it.

* The ability to process both pixel and symbol data in parallel.

* The ability to simultaneously maintain the low, intermediate, and high-level representations.

* Fast I/O and processing rates for huge amounts of sensory and intermediate level data.

* The ability to select particular subsets of data for varying types of processing.

* Summary feedback and evaluation mechanisms that allow focusing of attention and data-directed processing, without having to dump processing results to some "host" for external evaluation.

Beyond these requirements, two significant architectural implications can be derived from an understanding of our approach to the computer vision problem:

* Multiple levels of representation and stages of processing are essential and require very different types of processing elements.

* Fine grained and high speed communication and control is required both among the processes at each level and between the different processing levels.

The following discussion examines the computational requirements needed at the different processing levels and then moves on to discuss the communication and control issues involved in vision processing.

2.2 Processing Characteristics for Multi-Level Architectures

The three levels of abstraction discussed above have been mapped into three levels of architectural requirements with each level having the appropriate architecture for the set of tasks at that level. The simplest level is the low level where uniform computation is applied at local points, usually at each pixel, across the image. At the low level we need to perform operations such as smoothing, edge detection, region segmentation, feature extraction, and feature matching between frames in motion and stereo processing. Here each pixel, or a neighborhood around each pixel, must be processed (usually repeatedly); consequently the typical organization is SIMD. Some of these operations may also require processing over somewhat larger image neighborhoods such as in correlation between stereo and motion frames.

In addition to high-speed low-level operations, there is a need to be able to quickly load the data and to test the results of partial processing. To perform multiple iterations of

low-level operations under control of higher-level processes requires fast, concise feedback from the low-level processes to the intermediate- and high-level processes as well as data-dependent control of the low-level processes by the higher ones (discussed in Section 2.3).

At the intermediate-level our concerns are for comparing and manipulating the symbolic tokens associated with the extracted image events, such as regions, lines, and surfaces. The types of operations needed at this level involve indexing tokens by their content, which is referred to as associative retrieval, and comparing token attributes and relations. The processes are used to partition, aggregate, and in general transform these tokens into structures that more naturally match the object representation.

The intermediate level also requires specialized processors. Large numbers of line and region fragments can be generated by even the most effective low level algorithms. To perform grouping operations (e.g., merging and splitting of regions, or linking and reorganizing lines) one needs a large amount of "less local" communication. Fragments of lines must be matched and merged across large fractions of the image. Similarly, regions need to be merged and compared with others from possibly non-contiguous areas. Although there has not been as large a body of research on grouping and organizing surface patches, we belive that the processing issues for surfaces at the intermediate-level are equivalent to those for regions and lines.

Data reduction through abstraction is fundamental to intermediate-level processing. For it to be done quickly requires architectural support for both inter-level and intra-level communication as well as a flexible repertoire of data manipulation instructions. Thus, the intermediate level must operate as a server for the queries made as a result of object and scene processing at the high level, support data reduction processes, and provide control and evaluation mechanisms for the lower level processes.

Interpretation operations at the high level generate and test hypotheses based on available data provided by the low and intermediate levels of processing and request new data to be abstracted from the image if needed. Image interpretation involves processes that construct complex descriptions, during which many hypotheses are put forth; only those which can be satisfied by matches to two-dimensional and three-dimensional object models and their contextual constraints are accepted. The computational and communication needs of these processes should be provided by high level processors which form the third tier of processing power needed to solve the image understanding problem.

Thus at the high-level support must be provided for semantic processing involving mechanisms for focus of attention, the formation and verification of object hypotheses, knowledge-based inference using complex control strategies from multiple knowledge sources, and reasoning under uncertainty. This type of processing involves extensive symbolic computation. The high-level may also be concerned with geometric manipulation of three-dimensional models, which requires significant amounts of numeric computation.

2.3 Communication Between Processing Levels

A central characteristic of image interpetation is the bi-directional flow of communication and control up and down through all representation and processing levels. These capabilities allow multiple image operations to be performed, processing results to be evaluated, and algorithms to be re-applied with different parameters on different parts of the image. Specific communication operations must be performed in microseconds or in tenths of microseconds in order for different interpretation strategies to be tried dynamically within one or a few frame times.

In the upward direction, the communication consists of image abstraction and segmentation results from multiple algorithms, and possibly from multiple sensory sources. It also involves the communication of a set of attributes describing the token associated with each extracted image event; these are stored in a symbolic representation at the intermediate-level. From the intermediate- to high-level the upward communication consists of grouped collections of lower-level abstractions, their descriptions and responses to queries about token attributes and relations. In addition, summary information and statistics allow processes at the higher-levels to assess the success of lower-level operations. In the downward direction the communication consists of knowledge-directed processing and grouping operations, commands for selecting subsets of the image for specifying further processing in particular portions of the image, modification of parameters of lower-level processes, and requests for additional information in terms of the intermediate representation.

2.3.1 Associative Communications and Control

From the preceding discussion, it can be seem that communication between processing levels involves the rapid transfer and evaluation of information from lower to higher levels, and the ability to exercise fine grained control from higher to lower levels. Based on our experience with highly parallel algorithms [Levitan, 1987], we believe that the best way to meet these requirements of high speed, fine grained communication and control at the low and intermediate levels is with associative processing techniques. There are four processing capabilities that are key to associative computation [Foster, 1976]:

* Global Broadcast/Local Compare/Activity Control
* Some/None Response
* Count Responders
* Select a Single Responder

Associative processing can best be understood by an example of a single controller (a teacher) interacting with an associative array (a class of students). If the teacher needs to know if any student in a class has a copy of a particular book, the teacher can simply state, "If you have the book, raise your hand." The students each make a check, in parallel, and respond appropriately. This corresponds to a broadcast operation of a controller and a local comparison operation at each pixel in an array, to check for a particular value. Both operations assume that the local processors have sufficient computational capability to

perform the comparison.

Query and response is just the first part of associative processing. We have thus far described only a content addressable ("If your hand is up, I'm talking to you.") scheme. To perform associative processing, we must be able to conditionally generate symbolic tags based on the values of data and use those tags for further processing. For the following example, suppose that enchancement, connected components, and feature extraction for color, texture, size, and intensity have already been performed on the image data. It would then be possible to broadcast a query such as, "select all regions whose size is greater than a certain threshold value, and label them LARGE-REGION". This associative tag is not particularly interesting by itself but, when the controller starts performing multiple logical operations on tags, the processing becomes more interesting. Since each pixel (or region) could have multiple tags based on its extracted features, spatial location, and relationships to other pixels (or regions), it is possible to perform operations such as, "select LARGE-REGIONS with SKY-BLUE-COLOR, LOW-TEXTURE, in TOP-OF-IMAGE, ADJACENT-TO-TREE, and ADJACENT-TO-ROOF; then label these regions POSSIBLE-SKY." As processing continues only subsets of the pixels or regions are involved in any particular operation, but note that all pixels or regions with a given set of properties are being processed in parallel.

The ability to associate tags with values is half the battle for high speed control. We also need to get responses back from the array quickly. Forcing the teacher to ask each student if they have their hand up defeats the process. The teacher can see immediately if any of the students have their hands up, and can quickly count how many do. Similarly, a Some-response/No-response (referred to as Some/None) wire running through the pixel array allows the controller to immediately determine properties about the data in the array, and therefore the state of processing in the array *without looking sequentially at the data values themselves.*

Additionally, fast hardware to perform a count of the responders allows the controller to see summary information about the state of the data in the array. We can write programs that can *conditionally* perform operations based on the overall state of the computation. By using the properties of the radix representation of numeric values in the array, we can also use the counting hardware to sum the values in the array. The ability to sum values gives us the power to compute statistical measures such as mean and variance.

A single-responder-select operation provides one mechanism for transferring non-summary information from a lower to a higher level in the vision processing hierarchy. The higher level may select a single cell or token in a lower level, based upon a partial match of its information fields, and read it out. Selecting a single responder (also referred to as "Select-First") is useful when a group of tokens have been associatively selected by matching some of their attributes to broadcast values, and it is then desired to read out other attributes values for individual members of the set of selected tokens. Another purpose is to select single cells as representatives of groups (regions or segments) which can then be used to store facts about all cells in that group. For instance one cell associated with a single pixel in a region might keep the average color intensity and

variance for the entire region.

These examples of students, pixels, and regions illustrate the power of associative processing. Associative processing is used as the communications paradigm between each pair of levels and as the control paradigm for the low and intermediate levels in the hierarchy. Criteria are broadcast for selecting pixels, or regions, or symbolic tokens for selective processing. In this way higher levels control lower levels. It is possible to test and/or count the response that comes after processing data to allow conditional branching for the next step of processing in a given algorithm. Thus, the lower levels provide feedback to higher ones.

2.3.2 Parallel Data Transfer and Control Between Levels

As mentioned above the single-responder-select operation is useful when a small number of data values must be copied to a higher level. However, when a large number of values must be transferred between levels, a parallel data path between adjacent levels is more appropriate. A simple means of achieving this is by connecting spatially collocated processors in adjacent pairs of levels with independent data paths so that inter-level data transfers may be performed in parallel.

The parallel data transfer mechanism permits the use of a general processing strategy in which each level is used to transform a lower-level representation into a higher-level representation. All or part of this new representation may then be extracted by the next higher level of processing which then treats the level below it as an associative data base.

Since the parallel transfer mechanism also permits the passing of control information from higher to lower levels, the granularity of control can be made to vary. This would allow, for example, initial processing to take place in a coarse-grained control mode with a single controller (SIMD) and later processing to take place in a fine-grained control mode where multiple controllers are active (Multi-SIMD, or MIMD). The former is useful for generating initial hypotheses about an image, while the latter is better suited to resolving multiple local conflicts between those hypotheses, and to filling in details of the interpretations.

As can be seen from the preceding discussion, a vision architecture requires a complex and unique combination of processing power, communication, and control. The remainder of this section examines existing parallel architectures with regard to these requirements.

2.4. Review of Existing Architectures

In order to meet the computational requirements of image interpretation, it is clear that the power of parallel processing must be utilized. Existing parallel architectures, however, do not simultaneously address the varying computational needs of computer vision, although any given architecture may perform quite well on some portion of the

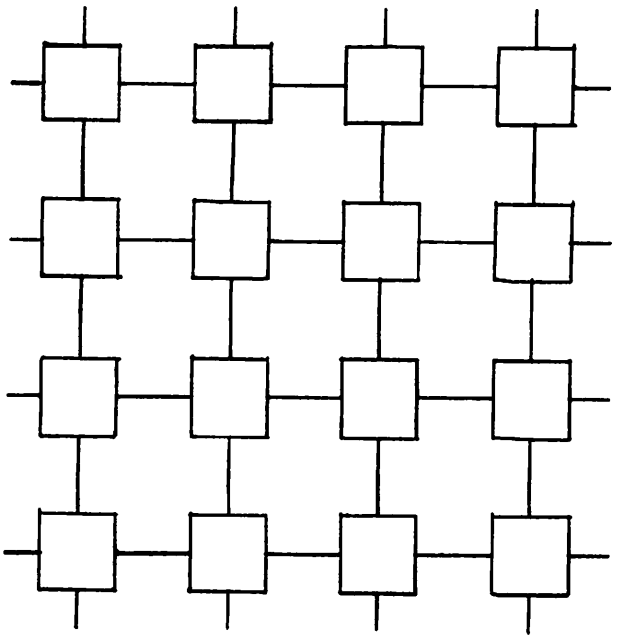


Figure 4. A 4-Connected Mesh of Processors

vision problem. This section will review six general classes of existing architectures and discuss their advantages and disadvantages with respect to solving the vision problem. The six classes are mesh-connected arrays, pyramids, hypercubes, mesh-plus-hypercube-connected arrays, shared memory systems, and systolic arrays.

2.4.1 Mesh-Connected Arrays

These machines are an obvious choice for image processing applications. A typical machine of this class has thousands of small processors that are connected to their nearest 4,6, or 8 neighbors. (Figure 4) Processors respond in SIMD mode to instructions that are broadcast by a central controller.

The advantage of this architecture is that images map quite naturally onto its structure. When the image size matches the size of a mesh- connected architecture, maximum parallelism can be obtained for operations that involve computations on individual pixels or small neighborhoods of pixels.

This type of architecture has several disadvantages for vision processing. Foremost of these is a lack of MIMD processing capability that is necessary to support high-level vision. In addition, there are many low- and intermediate-level vision algorithms that

involve the grouping or matching of image structures which are spatially distant in an image. In a mesh-connected architecture, however, communication of information across long distances is very time consuming. Lastly, in order to economically build an array with thousands of processors, the individual processors must be quite small. For example, typical processor will have a 1-bit ALU, a small amount of memory, and communication links to its immediate neighbors. Although such processors are adequate for performing pixel operations, they lack the power that is needed for manipulating intermediate-level tokens; for example, computing trigonometric functions on floating-point values is very slow on a 1-bit processor.

Some examples of mesh connected architectures are CLIP-4 [Duff, 1978], MPP [Batcher, 1980], DAP [Hunt, 1981], GRID [Arvind, 1983], and GAPP [Davis, 1984]. Each of these machines has its own special features, but all of them have the same general form. One additional characteristic of all of these machines is a lack of emphasis on global summary feedback mechanisms to permit rapid evaluation of processing results. Although it is possible for these machines to transfer their results to another machine for evaluation, doing so is a very time consuming operation. This reflects the fact that they have been designed as stand-alone image processors used primarily for image enhancement in which the results of processing are intended for interpretation by humans rather than forming the first stage of an autonomous vision system.

2.4.2 Pyramid Processors

An extension of the mesh-connected array concept is the multi-resolution pyramid or quad-tree. This structure has been proposed in a variety of forms [Uhr, 1972; Hanson, 1974, 1980; Rosenfeld, 1986; Tanimoto, 1983], but the essential idea is that an image-sized mesh-connected array is augmented by layers of successively lower resolution mesh-connected arrays. Each array in a pyramid processor is typically one fourth as large as the array below it, hence the term "pyramid". Except for those processing elements at the bottom level, each processing element in a pyramid is also connected to four processors in the array below it(Figure 5).

The pyramid processor provides the capability for quickly changing the resolution of an image, which can significantly improve the execution speed of some low-level algorithms (especially those that depend upon communication between cells that are spatially distant in an image). However, pyramid processors are more difficult to build than mesh-connected arrays because they have a more complex arrangement of communication links and require twice the number of processing elements as a mesh-connected array in order to achieve the same image resolution. Hence, no pyramid processors have been built commerically.

Even though a pyramid architecture has multiple levels of processing elements, it should not be concluded that a pyramid is suitable for implementing the multiple levels required for computer vision. The pyramid architecture implements an image resolution hierarchy, whereas computer vision requires an architecture that implements a hierarchy of abstraction levels. In a pyramid architecture, all of the processors are identical and execute

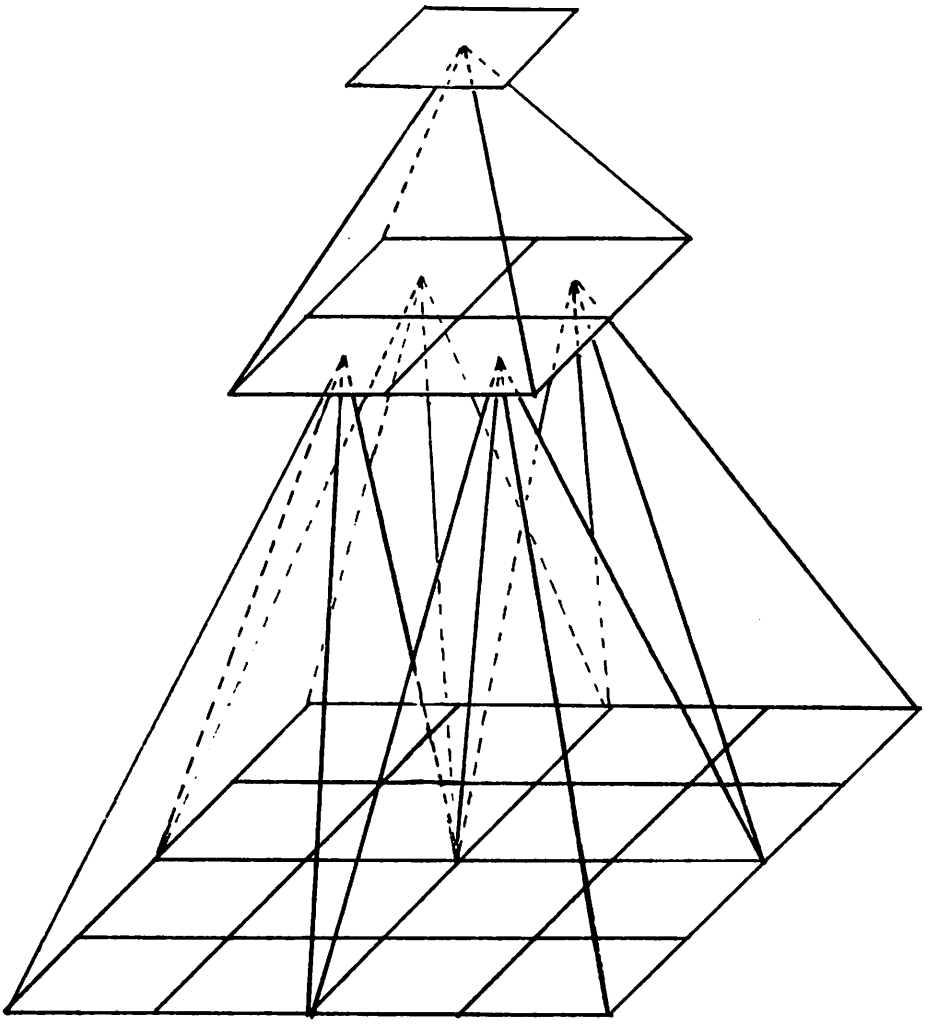


Figure 5. A Pyramid of Processors

in SIMD fashion. A vision machine, on the other hand, clearly requires different types of processors at different levels, and a variety of modes of parallelism including both SIMD and MIMD. At successively higher levels of a vision machine, the number of processing elements may decrease, but their complexity will increase so that, physically, each level of a vision machine will have roughly the same amount of circuitry. This contrasts with a pyramid architecture, in which the amount of circuitry decreases by a factor of four with each successively higher level.

Pyramid architectures are thus well suited to performing low-level vision operations. However, they do not provide the flexibility or type of processing power required at the intermediate- and high-level of vision.

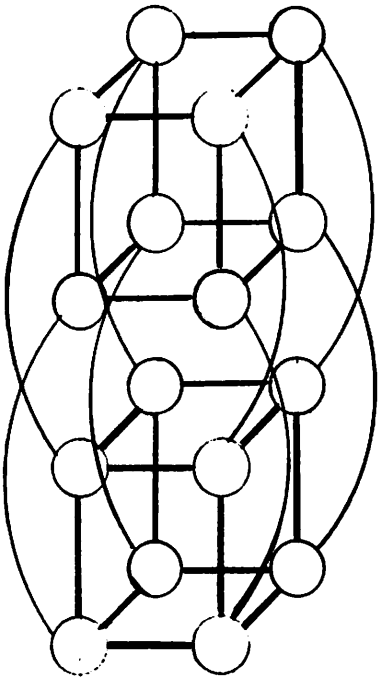


Figure 6. A Hypercube of Dimension 4.

2.4.3 Hypercube Processors

Machines of this class consist of processors connected by communication links whose arrangement is topologically equivalent to an n-dimensional cube. A hypercube machine of dimension d will have 2^d processors, each with d communication links to other processors. Each processor is at most d-1 links away from any other processor (Figure 6). Machines of this class range in size from tens to hundreds of processors, although at least one system with a thousand processors has been proposed [NCube, 1985]. A processor in a machine of this type is typically an 8,16, or 32 bit microprocessor with local memory of at least 64K-bytes and as much as 2.5 M-bytes.

The advantage of the hypercube architecture is that it provides rapid communication between distant processors. Although hypercube machines with large dimensionality can be built, current systems do not provide the same amount of parallelism as mesh-connected arrays, and thus are not as efficient for low-level vision processing. Depending upon the number and sophistication of the processing elements, a hypercube machine could be used for either intermediate- or high-level vision processing. However, the distributed nature of the data memory in a hypercube architecture can make the management of large, globally shared, data structures more difficult. For example, a global blackboard at the high level must be accessible by all processors on an equal basis. Existing machines also tend to have processors and memories that are too small to support high-level processing, but these

deficiencies are being addressed in newer models of these machines.

Some examples of hypercube architectures are the Intel Hypercube [Rattner, 1985], NCube [NCube, 1985], and Cosmic Cube [Seitz, 1985].

2.4.4 Mesh-Plus-Hypercube-Connected Arrays

In addition to the pyramid approach, one way to eliminate the local neighborhood communication limitation on a mesh-connected array is to add a set of hypercube communication links between the processors. (Although a mesh may be embedded topologically in a hypercube network, the result is usually slower and more cumbersome than a true mesh for image processing operations.) This provides the ability to work with images and to transmit information over longer distances. A machine of this type can be used for both low-level vision processing and some types of intermediate-level processing. However, just as with the mesh or pyramid, the lack of MIMD processing capability precludes its use for high-level vision. In addition, the low and intemediate level processing cannot take place concurrently in this type of machine, which is a necessity for real-time interpretation.

An example of this type of machine is the Connection Machine [Hillis, 1986].

2.4.5 Shared Memory Multiprocessors

Machines of this class are MIMD parallel processors in which each processing element is a 16- or 32-bit microprocessor that has access to a large global memory. In some cases the processing elements may also have a smaller amount of local memory. There are two general methods for linking processors to memory. The simplest method involves a high-speed bus to which all of the processors and memories are attached (Figure 7). Such machines typically have fewer than 30 processors (although systems with hierarchical bus architectures are being built that will have larger numbers of processors). A more complex method involves the use of a multistage switching network that provides links between processors and memory on a demand basis (Figure 8). Machines of this type have been built with up to 128 processors.

The advantage of a shared memory is the ease of implementing a large shared data structure, such as a blackboard. Thus, this class of machine is thus best suited for high-level vision tasks. Although it is possible to program a shared memory machine to perform image processing operations in parallel by dividing the image up among the processors, the result is far less effective for example, than when, for example, a mesh-connected processor is used. In addition to the much lower level of parallelism, this reduced effectiveness stems from the MIMD nature of the machine class. Each parallel operation must be created as a new process on the system, incurring substantial operating system overhead. Also, whenever the processes must interact, they suffer a time penalty in synchronizing with each other. Because the actual image processing operations execute relatively quickly when they are divided among multiple processors, the process start-up and synchronization overhead

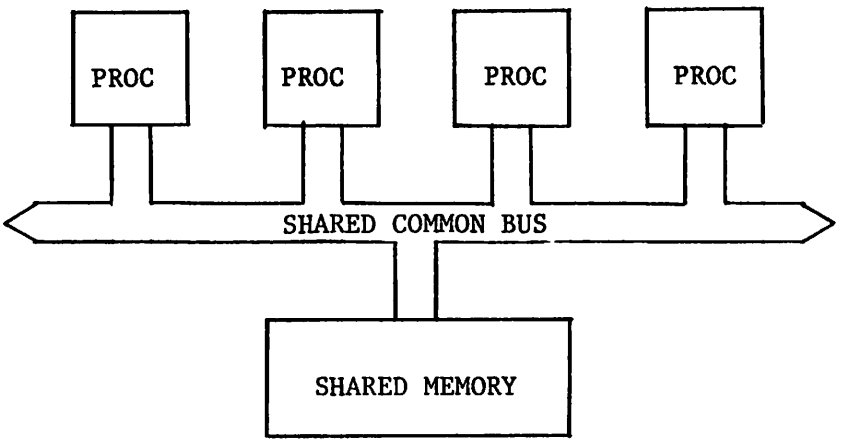


Figure 7. Common Bus Shared Memory Multiprocessor

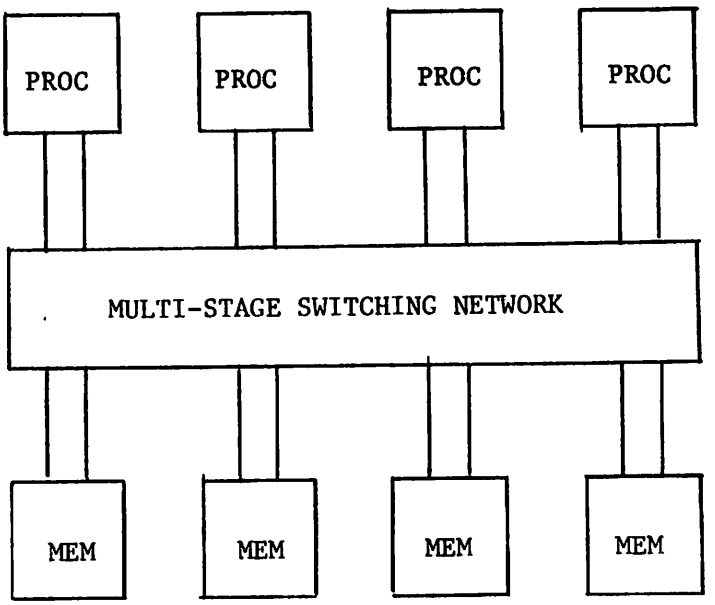


Figure 8. Switching Network Shared Memory Multiprocessor

rapidly grows to dominate the total processing time. In many cases, increasing the number of concurrent processes that are operating on an image beyond a certain point will cause the total execution time to increase.

Machines in this class can be used for intermediate-level vision, but they still do not obtain the greatest possible parallelism at this level. There are thousands of tokens that can be processed in parallel and only tens of processors in a typical machine of this class.

Two examples of the bus architecture shared memory are the Encore Multimax [Encore, 1986] and the Sequent Balance [Sequent, 1986] machines. Examples of shared memory systems that use multistage networks are the BBN Butterfly [Crowther, 1985] and the IBM RP-3 [Pfister, 1985].

2.4.6 Systolic Processors

These machines take their name from physiology. The term "systolic" refers to the contraction of the heart that rhythmically forces the blood forward. A typical systolic processing element has a small set of inputs and outputs. On each machine cycle it takes values from its inputs, performs an operation on them, and passes the results to its outputs. These systolic elements are chained together to form a systolic array. Data is pumped into one end of the array, passes through the elements in serial fashion, and the results emerge at the other end of the array. The systolic array can also be thought of as a pipeline with a series of pumping (or processing) stations. Once the pipe is filled with data, all of the processing stations are operating on values in parallel (Figure 9). Systolic array elements can be either general-purpose programmable function units or special-purpose fixed function units. The latter are not very useful for computer vision because of their inflexibility. The primary advantage provided by programmable systolic architectures is high performance for low cost. They are best suited for image processing tasks, but can work well with any application that involves large arrays of data. The main disadvantage of the systolic array is that any evaluation of the processing results must wait until all of the data has passed through the array. If a systolic array processes an image in one frame time, then this restriction has the effect of only allowing the controlling process to make a decision and change the array's programmed functions once each frame time. In a systolic array, it is thus much more difficult for a vision system to quickly and flexibly adapt its processing strategy to the actual characteristics of an image.

Of course, systolic arrays are inappropriate for high-level vision because most high-level processing involves data structures that are more sophisticated than numerical arrays. Systolic arrays are also not designed to support the MIMD style of processing that is characteristic of high level processing. One example of a systolic array that is being used for image processing is the CMU WARP [Kung, 1984].

3. OVERVIEW OF THE IMAGE UNDERSTANDING ARCHITECTURE (IUA)

The Image Understanding Architecture represents a hardware implementation of the three levels of abstraction embedded in our view of computer vision. Overall it consists of three different, closely coupled parallel processors. These are the Content Addressable

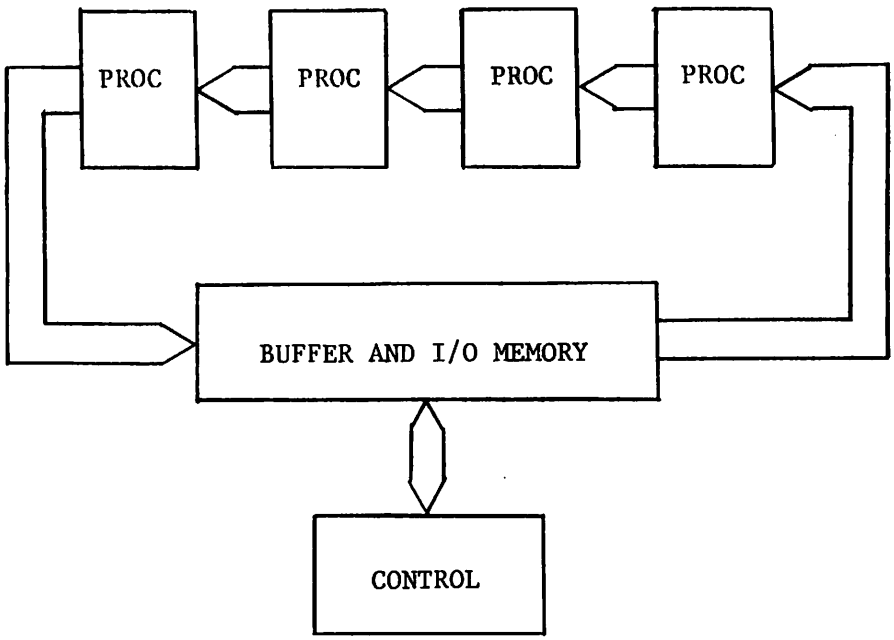


Figure 9. A Systolic Array

Array Parallel Processor (CAAPP)* at the low level, the Intermediate Communications Associative Processor (ICAP) at the intermediate level, and the Symbolic Processing Array (SPA) at the high level (Figure 10). The CAAPP and ICAP levels are controlled by a dedicated Array Control Unit (ACU) that takes its directions from the SPA level.

The three levels of the IUA implement a hierarchy of abstraction which corresponds to the three levels of abstraction (representation and processing) described previously. In each layer of the IUA the processing elements are tuned to the computational granularity and algorithms required by that particular level of abstraction. For example, it is inappropriate to try to run LISP in parallel at the lowest level, because the low level is primarily concerned with fast pixel operations.† Thus, the low-level processors are tuned for real-time image processing operations. At the highest level, on the other hand, symbolic AI processing

---

* The term "content-addressable" is a synonym for "associative" and is an alternate term that now is not as widely used as it was when some of our work began [Foster, 1976, Weems, 1984a.]

† Note that by parallel LISP we mean running independent LISP programs concurrently on multiple processors, and not simply processing data structrues in parallel as is done in Connection Machine LISP [Hillis, 1986]

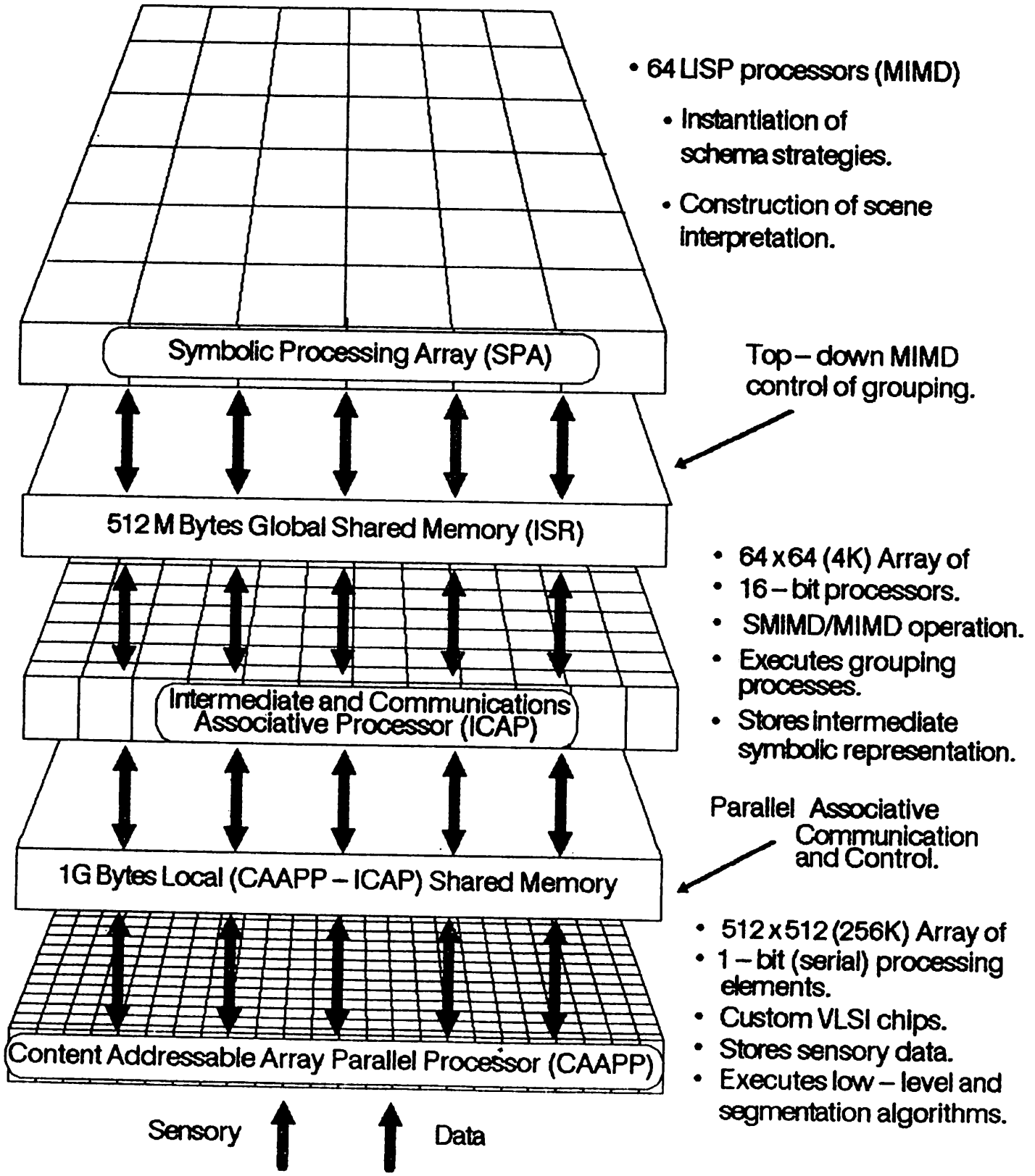


Figure 10. UMass IUA Overview

will be the main objective, so the high-level processors are selected for their ability to run LISP code efficiently.

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 to allow multiple hypotheses from the SPA to be evaluated in parallel at the lower levels. These include the CAAPP operating in pure SIMD or multi-SIMD mode, and the ICAP operating in synchronous-MIMD or pure MIMD mode. A brief explanation of how the multi-SIMD and synchronous-MIMD modes differ from the familiar SIMD and MIMD modes is required. In multi-SIMD mode, the CAAPP cells excute in disjoint SIMD groups, with each group receiving a different instruction stream. This allows different algorithms to be run on disjoint portions of the image, with each algorithm peforming pixel-parallel operations within its sector of the image. In synchronous-MIMD mode, the programming paradigm is more like SIMD than MIMD: the ICAP processors execute similar instruction streams, 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.

3.1 The Three Processing Levels

The CAAPP is a 512×512 square grid array of custom 1-bit serial processors intended to perform low-level image processing tasks. Although each processing element provides a small amount of procesing power in comparison to a modern uniprocessor, the simplicity of the processing elements allows us to place 64 of them on a single VLSI chip and thus the machine may be economically constructed. The CAAPP 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]. The architecture of the CAAPP, however, is especially oriented towards associative processing with an emphasis on global summary feedback mechanisms. This reflects the goal of our work, which is automated real-time vision without human intervention, where all of the processing and interpretation must be done by the system itself as opposed to the speeding up of simple image processing tasks and/or low-level computer vision. Thus, the CAAPP has been tailored to permit flexible control, and to provide rapid feedback to the controlling processes so that they may exercise control in response to actual image properties.

The intermediate level is implemented by the ICAP, which is also a square grid associative array, but is composed of fewer but more powerful processing elements; the ICAP is a 64×64 array of off-the-shelf 16-bit digital signal processor chips (Texas Instruments TMS320C25). Each ICAP cell is associated with an 8×8 sub-array of CAAPP cells, to which it has access.

The ICAP is designed to manipulate the tokens at the intermediate level and to support the data base functions which allow the symbolic interpretation processes, running on the SPA processors, to access these tokens. 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. The symbolic representation of a line in the ICAP would consist of a unique label for the line and a set of fields which quantify its attributes. Such attributes may include end points, orientation, contour length, relative curvature, direction of curvature, average contrast across the line, labels of adjacent regions, nearby endpoints, and pointers to nearby or related lines. The CAAPP is used to develop the intermediate symbolic representation, which is then passed to the ICAP for further processing. Should the need arise, the results of re-segmentation in the CAAPP can be integrated with the representation in the ICAP. The ICAP representation facilitates this processing capability because it is in approximate registration with the original image events in the CAAPP (there is a 64 to 1 processor ratio between spatially collocated CAAPP and ICAP elements).

The 64 SPA processors are powerful 32-bit, general purpose microprocessors intended to run LISP for performing high-level symbolic operations, and for controlling sub-array processing in the ICAP and CAAPP arrays. From the point of view of the SPA, the lower levels appear as an intelligent database that is stored in a shared global memory and maintained by a set of concurrent processes. The shared memory decouples the SPA processes from the locality of information in the image. An SPA process simply makes a request to the database and then waits for completion of the request before accessing the database to get the results.

The SPA processors will run a LISP-based blackboard system [Erman, 1980, Nii, 1986, Draper, 1987a,b] in which various object schemas will cooperate and compete in the generation and verification of hypotheses about the content of the image and its relationship to models of the environment. From the point of view of the blackboard system, the CAAPP and ICAP will appear as knowledge sources at different levels of abstraction. The various schemas in the system can activate different processes in the CAAPP and ICAP either for the full array or for independent sub-arrays. Thus, the SPA processors operate in MIMD mode with communcation through the blackboard, thereby allowing flexible horizontal and vertical parallel computation through the three levels.

The IUA is conceived as a stand-alone image interpretation system. Although a host processor is attached to the global controller for the IUA, the host is intended to serve the IUA rather than the other way around. The IUA host will provide a software development environment, and an access point for users of the IUA. The host may be used for examining the results of processing on the IUA, or for monitoring processes in the IUA. However, the host system does not take part in the actual image interpretation process. The ACU, which is an integral part of the IUA system, is responsible for managing the CAAPP and ICAP arrays in cooperation with the SPA, and for providing the interface to the host system.

3.2 The CAAPP Processing Elements

The CAAPP consists of a 512×512 array of bit-serial processing elements that are

linked through a four way (S,E,W,N) communications grid which is augmented with circuitry that allows certain types of long distance communication to take place quickly (see section 3.5). Each processor contains 320 bits of RAM, 5 one-bit registers (A-Activity Control, B-Secondary Acitivity Pattern, X-Response, Y-General Purpose, Z-Arithmetic Carry), an ALU and data routing circuitry. It is important that the primary working data memory be on the same chip as the processing elements, in order to maximize access speed, and to keep the number of pins on the chip within reasonable limits. However, each element also has access to a 32K-bit backing store memory that is dual-ported with the ICAP. The backing store is also referred to as the CAAPP-ICAP shared memory (CISM). Figure 11 shows the CAAPP cell architecture, and Figure 12 presents the instruction set for the CAAPP. The CAAPP processing element is very simple, but it is this simplicity that allows 64 of them to be placed on a single integrated circuit, and that also permits the CAAPP to execute instructions with a cycle time of 100 nanoseconds.

The initial development of this design is described in greater detail in [Weems 1984a]. Since that time significant work has gone into improvements to the CAAPP processing element [Weems, 1984b, 1985]. The current design of the CAAPP processing elements has been achieved through two iterations of reduced instruction set analysis and redesign of the processing element architecture.

We are currently preparing a 2-micron CMOS implementation of the CAAPP chip design with the cooperation of Hughes Research Laboratories. Each chip has 64 processing elements and contains approximately 90,000 devices, of which 50 percent are in the on-chip memory. This is roughly the same number of devices found in 16-bit microprocessor chips, but was much simpler to design and implement because of the repetitive nature of the parallel processor cell design. Figure 13 is a photomicrograph of a CAAPP test chip that contains 32 processing elements.

3.3 Associative Feedback from the CAAPP

The key to integrating the CAAPP into the IUA is its combination of associative feedback and control mechanisms. The principle feedback mechanism in the CAAPP is the array-wide logical OR output, called Some/None, which indicates whether any CAAPP cells are in a given state represented by the response bit (X register). At the end of each instruction cycle the logical OR of the response bit from every processing element is automatically available at two different levels. The global controller receives a Some/None signal for the full array, while the ICAP processors receive the Some/None indication for that portion of the CAAPP array connected to each of them.

A count of all responding cells is also available at the global controller and ICAP level. The counting operation is used to gather statistics about an image and the results of processing. For example, through counting we may quickly determine the mean and standard deviation of an attribute value for a given set of processors (see section 3.8). The 8×8 CAAPP sub-array counts are available at each corresponding ICAP at the end of each CAAPP instruction cycle. The global controller develops the full array count

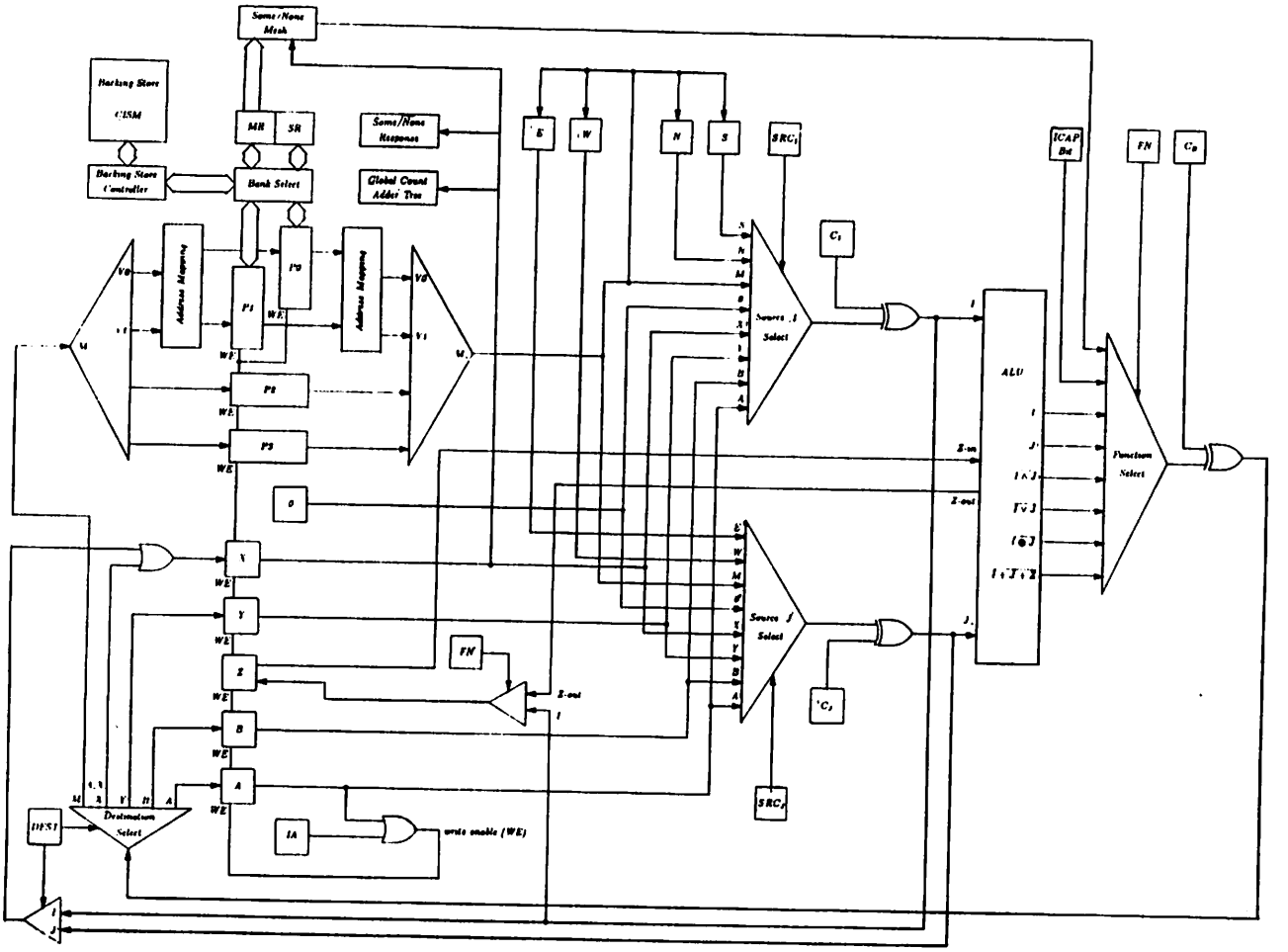
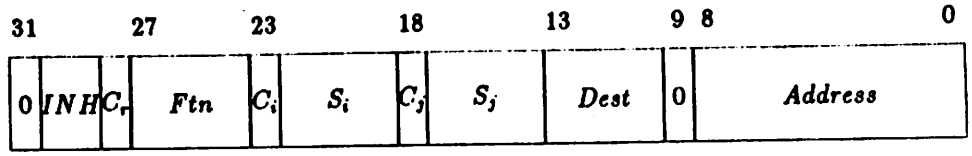


Figure 11. CAAPP Cell Architecture



INH	(Inhibit)
0	Non-inhibit — always active
1	Inhibit if A = 0
2	Inhibit if A = 0 or S/N = Some
3	Inhibit if A = 0 or S/N = None

$C_{i,j,r}$	
0	TRUE
1	Complement $S_{i,j}$, *Result*

	S_i	S_j	*Ftn*	*Dest*	
0	zero-reg	zero-reg	$Some/None \Rightarrow Dest$	X_{pe}	0
1	C	C	$S_i \Rightarrow Dest$	A, X	1
2	S	E	$S_j \Rightarrow Dest$	$A, X \Leftarrow S_i$	2
3	N	W	$\overline{S_i \bigwedge S_j} \Rightarrow Dest$	$A, X \Leftarrow S_j$	3
4	Y	Y	$\overline{S_i \bigvee S_j} \Rightarrow Dest$	Y	4
5	X	X	$\overline{S_i \bigoplus S_j} \Rightarrow Dest$	X	5
6	B	B	$\overline{S_i + S_j + Z} \Rightarrow Dest$	B	6
7	A	A	$ICAP\ C \Rightarrow Dest$	A	7
8	memory	memory	$S_i \Rightarrow Z$	memory	8
9	—	—	$memory \Rightarrow MR$	—	9
10	—	—	$memory \Rightarrow MR, SB$	—	10
11	—	—	$MR \Rightarrow memory$	—	11
12	—	—	$MR, SB \Rightarrow memory$	—	12
13	—	—	—	—	13
14	—	—	—	—	14
15	—	—	—	—	15

Figure 12. CAAPP Cell Instructions

through a polling mechanism that takes 16 CAAPP instruction times (1.6 microseconds) to complete. However, the polling operation is independent of processing in the CAAPP cells and so the CAAPP may continue to operate while a count is being formed.

The Select–First operation is used to choose a single CAAPP cell from among multiple responders for readout or processing by the global controllor or the ICAP processors. As necessary, the data in these selected CAAPP cells can be moved up to the ICAP level. The Select-First operation can be applied to the CAAPP array as a whole, or to independent, contiguous groups of CAAPP cells in parallel.

3.4 Intra-Level Communication Within the CAAPP

Communication among CAAPP cells may take place in four different ways. One way is through global feedback and rebroadcast. This method is used when all or most of the CAAPP processors must be told the value of one of the processors (e.g. broadcasting the maximum value so that all cells can normalize their values). A second way is via the ICAP. In some cases it is more efficient to transfer CAAPP data to the backing store and

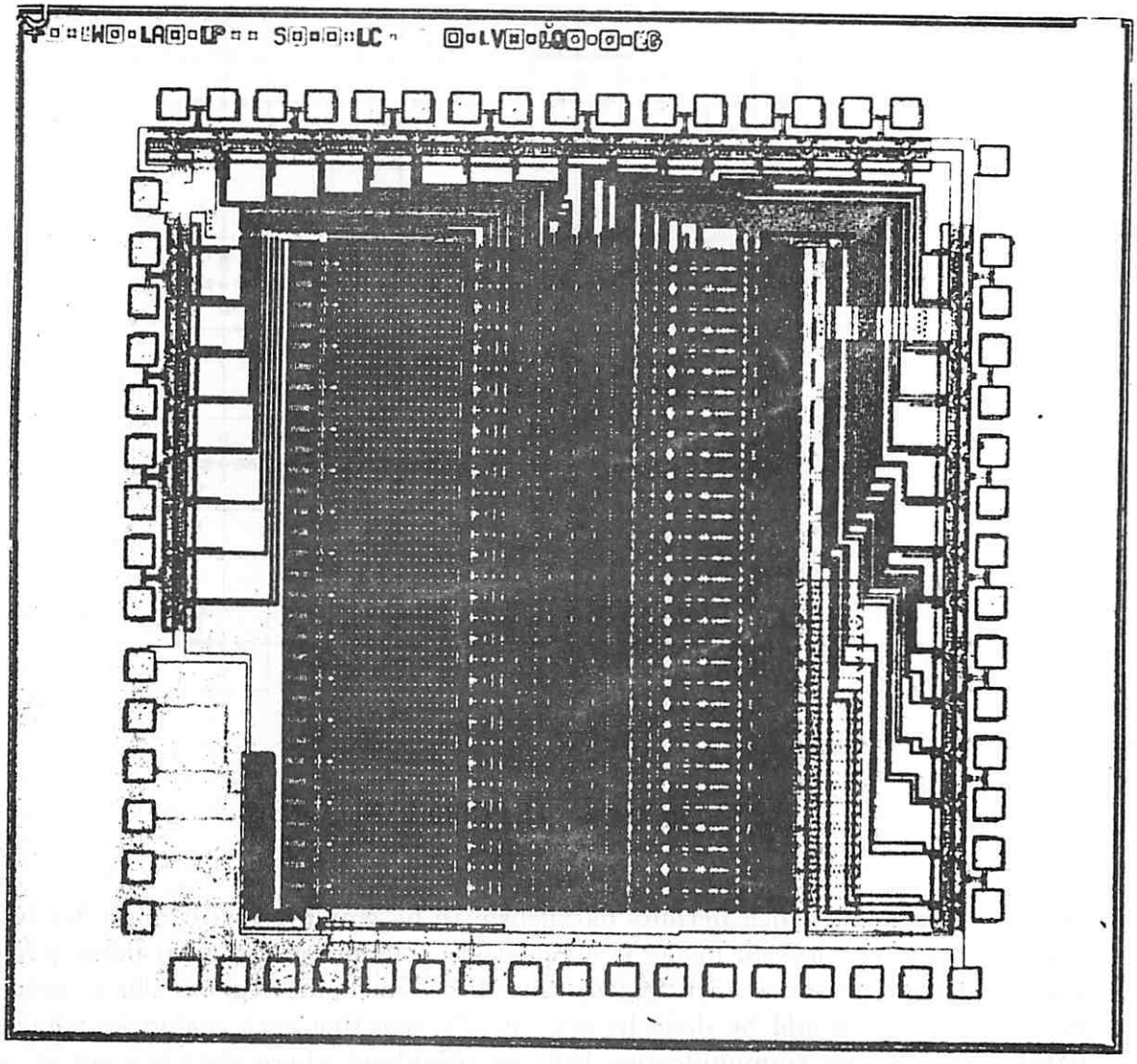


Figure 13. Photomicrograph of CAAPP Test Chip

let the ICAP move it across the array and place it in the backing store of the appropriate CAAPP cell. The third way uses the nearest neighborhood (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 last communication mechanism within the CAAPP is presented in the next section.

3.5 The CAAPP 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. By adding the simple switch network shown in Figure 14, it is possible to create independent groups of processors that share a local associative Some/None feedback circuit. The isolated groups of processors can then respond to globally broadcast instructions in a locally data-

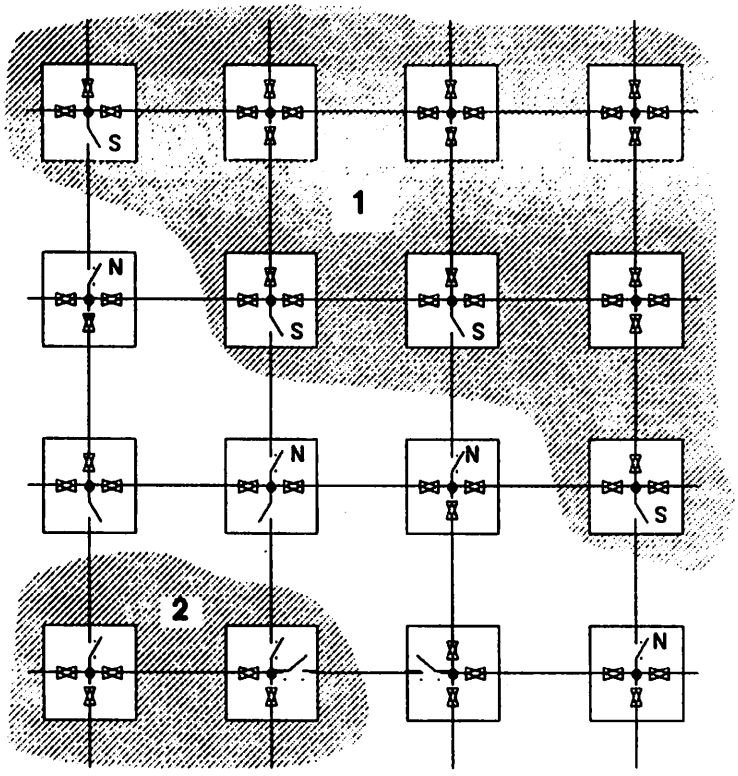


Figure 14. The Coterie Network

dependent fashion, which permits parallelism to be employed with more flexibility. For example, suppose that an image is divided into a large number of regions, and that we wish to determine some statistical measure for each of the regions. 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 statistics are accumulated via a propagating wave that checks region labels at each step. In the CAAPP, however, all regions can perform their own local evaluation in parallel without having to check region labels after one initial step of neighbor comparison.

The name Coterie Network is based upon the similarity of the isolated processor groups to a "coterie": that is, a group of people who associate closely because of common purposes, interests, etc. [Random, 1968]. The isolated groups of processors are thus referred to as coteries. Note that the Coterie Network is a switched mesh that is under program control, and is separate from the nearest neighbor mesh, which we refer to as the SEWN Mesh.

This added form of parallelism is possible because, as is shown in Figure 14, the processors at the edge of a region can break their electrical connections to any of their adjacent processors that are not members of that same region. Thus, the processors can be reorganized into isolated groups, so that within each group, the processors share a common associative Some/None feedback path. When a single processor is active within each coterie, then this path may also be used to broadcast a value from the active processors

to the rest of the cells in their coteries, as in a Broadcast Protocol Multiprocessor [Levitan, 1984].

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 disimilar 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. It should be fairly obvious to the reader that, among other things, each region of a segmentation could be a coterie of cells. 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.

Within a coterie, there is a mesh of wire that all of the processors are connected to. The ACU may instruct each active CAAPP processor to output a bit onto its coterie's mesh and then read whatever bit value is currently on the mesh within its coterie. When more than one processor in a coterie tries to output a bit onto the mesh, the value that appears on the wire is the logical OR of the output bits of all of the processors in the coterie. The shared mesh is thus functionally equivalent to the global Some/None feedback circuit except that its output is only locally formed and available within a coterie. In addition to its associative feedback function, the coterie mesh 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 parallel within all coteries. Subsequent ACU instructions for placing a value onto the mesh 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 mesh. In this way the coterie network can be used for local broadcasting of data values. The local feedback and broadcast processes can occur in every coterie in parallel.

3.6 Inter-Level Communication Between the CAAPP and ICAP

The principle mechanism for transferring data between the CAAPP and ICAP is the CISM (or backing store). Besides the normal program and data memory, each ICAP processor has access to the a 256K-byte block of memory that is shared with the corresponding 64 CAAPP processors. This 256K-byte memory also acts as the 32K-bit by 64-bit backing store for the on-chip CAAPP memory and is the primary communications path between CAAPP and ICAP. Swapping to and from the CISM is done through dual-porting of a portion of the on-chip CAAPP memory. Although access to the off-chip memory is about 10 times slower for the CAAPP than access to the on-chip memory, the dual-porting arrangement permits double-buffered swapping to take place while the CAAPP is processing data in other memory segments. When data is moved between the CAAPP and the CISM it goes through an automatic corner turning mechanism that provides bit-serial data access to the CAAPP and byte-parallel access to the ICAP.

```
-- Beginning with the high-order bit
FOR Bit_Num := Field_Length - 1 DOWNTO 0 DO
    Response := Field[Bit_Num]        --Put bit in response register
    IF Some                           --If any cell has a 1 in this bit
      THEN
        Activity := Response          --Then turn off activity in cells
                                      --with a 0 in this bit
```

Figure 15. Finding a Maximum Value in the CAAPP

In the event that large amounts of CAAPP data must be moved up to the SPA or controller, the data is first transferred in parallel to the ICAP processors which are also linked to the SPA processors through a dual-ported common memory.

3.7 Some Example Algorithms for the CAAPP

The simple CAAPP algorithm in Figure 15 demonstrates how the CAAPP is programmed, and the importance of rapid feedback from the CAAPP to its controller. This algorithm selects all active cells that contain the globally maximum value in a given field. In addition, the maximum value is available in the controller at the end of this operation. Selecting the maximum array value requires three CAAPP instruction cycles for each bit in the field.

The algorithm begins by loading the high order bit of a given field into the response register of all active cells. The global controller then tests the Some/None output of the array. If any cells have their high order bit set, then they are candidates for the maximum value; in which case, any cells that that have a zero in their high order bit are then deactivated. However, if no cells 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 field. When the low order bit has been processed, only those cells which contain the maximum value will remain active. For each iteration, the controller saves the Some/None response so that the maximum value is available in the controller at the conclusion of processing. This takes 24 CAAPP instruction cycles (2.4 microseconds) for an 8-bit value.

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 labelling of an image shown in Figure 16. The algorithm begins by loading each processor with its address in the array from the backing store memory. This serves to give each processor a unique number. Next,

```
Load_Processor_Addresses
Coterie_Switches:=(Open, Open, Open, Open)        - -Initialize coterie
                                                  - -switches
FOR Neighbor:= North TO West DO                   - -Initialize flag for
    Equal:= True                                  - -each neighbor
                                                  - -For each bit in field
    FOR Bit_Num:=Field_Length -1 DOWNTO 0 DO
                                                  - -Compare own bit
                                                  - -with neighbor
        Equal:=Equal AND(Neighbor.Field[BitNum]=Field[BitNum])
    IF Equal                                      - -If field value matches
                                                  - -neighbor, bit for bit
        THEN          - -then close the coterie switch to connect with
            Coterie_Switch[Neighbor]:=Closed      - -that neighbor
                                  - -Find maximum addresses in coteries
FOR Bit_Num:= Address_Length -1 DOWNTO 0 DO
    Response:=Address[Bit-Num]                    - -Put bit in response register
    IF Coterie_Some                               - -If any cell has a 1 in this bit
        THEN
            Activity:=Response                    - -Then turn off activity in
                                                  - - cells with
                                                  - -with a 0 in this bit
```

Figure 16. Connected Component Labelling using the Coterie Network

the Coterie Network switches are opened between processors that are on region boundaries (i.e. between pairs of processors that have diffeent values) establishing a coterie for each image region. Lastly, all regions in parallel determine their local maximum address value. (This is the same algorithm as for finding a maximum value in the entire array except that the coterie Some/None response is used in place of the global Some/None response to control the setting of activity.) As part of finding the maximum, every processor in a coterie stores the maximum address value for all cells in its coterie in its own memory. 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. From our electrical simulations of the Coterie Network, we calculate that this algorithm will take approximately 50 microseconds to execute.

These algorithms use only the Some/None Response form of feedback. The response count is of equal importance in many of our algorithms. For example, we can form a histogram of any numerical feature in the CAAPP using the response count. This is quite simple to do: For each bucket in the histogram we associatively select those cells whose values fall within the range of the bucket by broadcasting the minimum and maximum value of the range, comparing with the cell's value, and appropriately setting the response

```
Sum:=0                                    - -Initialize sum
FOR Bit_Num:=Iligh_Order DOWNTO Low_Order DO
                                          - -Count each bit in
    Response:=Field[Bit Num]              - -field and add to
    Sum:=Sum*2+Response_Count                 - -sum, scaling appro-
priately
Response:=Activity                        - -Count number of active
                                          - -cells
Mean:=Sum/Response_Count                  - -and compute mean.
```

Figure 17. Computing the Mean of Values in Selected CAAPP Cells Using the Response Count Operation

register to 0 or 1; then a count of the responding cells gives the histogram bucket value. The time to form the histogram is thus proportional to the number of buckets in the histogram and is independent of the number of values in the array.

Figure 17 gives a CAAPP algorithm that uses the response count to compute the mean of the values stored in selected cells. The algorithm begins by summing the values in the selected cells. Starting with the high order bit, each bit of the cell's values is separately counted. The counts are each added to the overall sum after being appropriately scaled by a power of two. The algorithm concludes by setting the response bit equal to the activity bit so that the response count will be the number of active cells, and dividing the sum by that count to get the mean of the values.

Of course, in addition to processing that is oriented around associative feedback, the CAAPP is able to perform the usual image processing and low level vision algorithms that do not depend upon feedback to the controller. For example, Gaussian convolution, Sobel operator, Canny operator, K-curvature, border following, etc. all use the mesh connected operations that are typical of this class of machines. Presenting the CAAPP implementations of these and other algorithms here would require far too much additional space and will be left to a future paper.

3.8 ICAP Structure and Control

The ICAP is a square grid (64×64) array of Texas Instruments TMS320C25 processor chips. It is intended to perform intermediate level processing on image events that are in physical registration with the pixel data. Each ICAP processing element is associated with an 8×8 subarray of processors in the CAAPP array. An ICAP processor has access to data stored in any of the 64 CAAPP processors with which it is associated. Each ICAP processor also has access to the global summary information for those 64 processors.

There are two reasons for choosing 64 as the number of CAAPP processors to associate

with each ICAP processor. First, it is convenient to associate one ICAP with each CAAPP chip since this greatly simplifies the interface between the two levels. Second, with a 512×512 array at the bottom level and three processing levels plus a single ACU, a uniform inter-level connectivity is provided by a 64 to 1 reduction factor between each of the levels. Finally, given the inter-level communication rates and processing loads for the processing elements at each level, we find that 64 to 1 is a reasonable processor ratio between each pair of levels.

Each of the 4096 ICAP processors consists of a CPU that can perform 16-bit operations, 256K bytes of local RAM, dual ported memory for interacting with the CAAPP and SPA, and neighbor communications hardware (discussed in the next section). The hardware for an ICAP cell consists of an off-the-shelf CPU and memory, plus a custom VLSI circuit that provides inter-level communications and control functions. The Texas Instruments TMS320C25 operates 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, we chose to use a digital signal processor at the ICAP level because its instruction set and arithmetic capabilities are well suited for performing computations in spatial geometry. (Three-dimensional geometric projections and computing distances are among the more frequently employed operations 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 an intermediate symbolic representation has been developed in the ICAP, (and continues 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 responses and a global summation value as feedback to the ACU. The three global OR responses from the ICAP (called Done-1, Done-2, and Done-3) are used by the controller 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 correspond to the three interrupt levels in the TMS320C25, so that they indicate when all processors have serviced a given interrupt level. Another use is to indicate completion of a task in the ICAP array with or without exceptions. Yet another use for the ICAP global OR responses is as an associative Some/None mechanism. The global summation mechanism uses the global count hardware in the CAAPP to form a sum of an 8-bit value from each ICAP processor. Thus the ICAP-sum and CAAPP-count operations cannot occur simultaneously.

3.9 ICAP Communication

The ICAP communicates in the vertical dimension via two sets of shared memory structures. There is one layer of shared memory between the CAAPP and the ICAP, and another layer between the ICAP and the SPA. The CAAPP-ICAP Shared Memory (CISM or backing store), has already been discussed (section 3.6).

The ICAP-SPA Shared Memory (ISSM) is viewed as an I/O device by each ICAP processor. A given ICAP processor can write (or read) up to 256 16-bit words to (from)

an I/O buffer in the ISSM. The ICAP then initiates a block transfer between the I/O buffer and a page of its choosing in the ISSM RAM. The ISSM RAM associated with each ICAP holds 128K bytes (64K 16-bit words). An ICAP processor may only access the 128 K byte segment of ISSM that is associated with it, however, each SPA processor has global access to the entire ISSM for all of the ICAP processors. This structure allows processes in the SPA to access the results of ICAP processing regardless of their spatial locations in the array.

The horizontal links between the ICAP processors provide the intra-level communications necessary for grouping and merging processes to operate on token attributes and token relations within the intermediate symbolic representation. These links are circuit-switched bit-serial transmission lines and provide a 5 M-bit/second data transfer rate between ICAP processors. The topology of this network includes a mesh with additional hypercube links that provide for both local neighborhood and long distance communication across the array. The ICAP network is centrally switched by the ACU following a global synchronization of the ICAP processors.

3.10 The SPA: High Level Processing

The detailed architecture of the SPA has not yet been fully defined. In the first prototype of the IUA, which is a 1/64th vertical slice of the full IUA, the SPA will be a single Motorola M68020 class processor. A separate research investigation within the UMass VISIONS project, is currently exploring the implementation of cooperative algorithms and data structures using a shared-memory multiprocessor at the SPA level [Draper, 1987]. This experience is providing additional direction to the future scaling up of the IUA at the SPA level.

Currently, the full SPA is envisioned as consisting of 64 or more processors, each capable of running LISP. Each processor will have some local memory and will have access to a global shared memory that will include the ISSM, and the blackboard. To the SPA processors the ACU appears to be just another SPA processor. Any SPA processor may send a request for service at the ICAP or CAAPP levels to the ACU by posting its request in the shared memory. An SPA processor can also direct processing at the ICAP level by transferring execution scripts to individual ICAP processors via the ISSM.

The SPA is being designed to support parallel schema-based (or frame-based) processing within the framework of a blackboard architecture [Draper, 1987b]. In such a system, various schemas are activated in parallel via focus-of-attention strategies to interpet results of initial bottom-up processing. The schemas generate hypotheses which activate verification processes in the form of other schemas. These, in turn, exercise control over the ICAP and CAAPP to perform top-down processing and re-evaluation. The schema processes cooperate and compete via messages posted on the system blackboard.

3.11 The ACU: Controlling the CAAPP and ICAP

The purpose of the Array Control Unit is primarily to issue SIMD instructions to the CAAPP. However, it is also responsible for coordinating execution in the ICAP processors whenever they are operating in synchronous-MIMD mode, or interacting closely with the CAAPP. One major design goal for the ACU was to maximize the rate at which instructions are issued to the CAAPP. This meant that the overhead for controlling loops, branches, and subroutine calls in the ACU had to be minimized. A second major design goal for the ACU was to minimize the cost of implementing a complete development environment for it. Preferrably, the ACU would execute a commonly used instruction set so that software could be transported from an existing machine.

Clearly, the first goal required a custom processor, while the second goal dictated an off-the-shelf processor. The solution to this dilemma was to incorporate both into the ACU design. Thus, the ACU contains two separate processors that can issue instructions to the CAAPP (or to the ICAP as described below). The two processors are referred to as the Macro-controller and the Micro-controller.

The Macro-controller is a Motorola M68020-based system that brings with it the wide range of software tools that are available for that processor. It can issue instructions to the CAAPP in two ways. The simplest way is to take direct control of the instruction bus and write out data values that will be interpreted as instructions by the processor arrays. Even at its maximum rate, however, the 68020 can only issue instructions at about one-tenth of the rate that the CAAPP can execute them. The second method for the Macro-controller to issue instructions is to issue subroutine calls to the Micro-controller.

The Micro-controller is a custom processor, driven by horizontal microcode. It is capable of issuing an instruction to the CAAPP every 100 nanoseconds, with minimal overhead for loop, branch, and subroutine control. The Micro- controller communicates with the Macro-controller through a set of dual-ported registers. The Micro-controller has a large library of CAAPP routines in its program memory, any of which can be called by the Macro-controller by writing the appropriate parameters into the dual-ported registers and issuing an interrupt to the Micro-controller. When the Micro-controller completes execution of a CAAPP routine, it returns a status flag to the Macro-controller which may then issue a new call.

The routine-calling mechanism permits the user to write applications in a high-level language for the Macro-controller, and yet obtain good peak instruction rates for operations on the CAAPP. Although this does not provide 100% utilization of the CAAPP, it is reasonable to expect 50% utilization in many cases, which should be adequate for most research and development situations.

When an application requires maximum performance from the CAAPP, it must be micro-coded for execution on the micro-controller. The micro-controller is a complete processor, capable of executing general purpose programs. Thus, for real-time applications, a typical development scenario would involve rapid prototyping of an implementation on

the Macro-controller, followed by migration of the high-level code into Micro-controller instructions. In the future, tools and compilers will be developed for the Micro-controller that will aid the code migration processes. Ilowever, it is reasonable to assume that the Macro-controller development environment will always be more attractive. Thus, it is expected that the two-stage development process will remain as the standard approach to implementing real-time applications on the IUA.

Although the only source of instructions for the CAAPP is the ACU, the ICAP processors each have their own 128K-byte program memory. The ICAP progam memory is loaded with a large library of service routines upon system initialization. The way in which the ACU issues instructions to the CIAP is by storing a user program into ICAP program memory and then issuing an interrupt to the ICAP that causes it to jump to the user program. (The program is broadcast to all of the ICAP program memories in parallel.) An ICAP user program is typically just an execution script of calls to the ICAP library. Thus, once a user program has been started in the ICAP, the ACU and ICAP interact very little (the exception is when the ICAP program reaches a global synchronization point: synchronization of the ICAP is mediated by the ACU). The ACU can also set the ICAP to operate in MIMD mode. This is done by turning control over to a task queing program in the ICAP processors. The queing program reads execution scripts from the ISSM according to a predefined protocol. When the ICAP is executing in MIMD mode, it depends upon the SPA to provide coordination of any required synchronization of ICAP processors. The SPA processors can also issue processing requests to the ACU when an array-wide operation is required.

The architecture of the ACU thus provides for the close interaction between the CAAPP and ICAP during the initial phases of interpreting an image. However, the ACU also permits the CAAPP and ICAP to work independently, with the ICAP taking directions from the SPA as the high level interpretation processes come into play. This allows the CAAPP to concurrently perform additional-low level processing, such as integrating information from other sensors or starting to process the next image. In the section that follows, which is an interpretation scenario for the IUA, it should be kept in mind that the ACU is actually managing all of the processing in the CAAPP and ICAP.

4. A VISION PROCESSING SCENARIO FOR THE IUA

The discussion that follows describes one possible sequence of operations on the IUA that could be used to form an interpretation from an image. This is actually a gross oversimplification of how the UMass VISIONS system is used to interpret an image. It would be impossible to provide a complete discussion of the full interpretation process on the IUA within the space available in this paper. Our purpose here is merely to show the types of processing and interactions that can take place in the IUA.

The processing is initiated with a region segmentation of the image. The first step is to apply an edge-preserving smoothing operator. We use an algorithm which involves five iterations of a 3×3 window convolution on the CAAPP. The next step is a region

segmentation [Beveridge, 1987], which uses local histograms within 16×16 windows. In brief, each ICAP computes a histogram for the 8×8 tile of CAAPP cells associated with it. This is done by broadcasting series of value range comparisons and performing local count operations that are applied to the CAAPP. The ICAP simply records the local count for each range selection, corresponding to buckets in the histogram. The ICAPs then merge their histograms through communication with their horizontal neighbors in the same 16×16 window. (Actually, the windows overlap by 4 pixels in each direction, forming 24×24 pixel histograms and requiring a bit more complex communication at the CAAPP and ICAP levels than we have room to discuss here.) Next, the ICAPs search their histograms for peaks and valleys, applying various criteria for defining clusters of values. The ICAPs communicate with their neighbors to consistently extract labels of peaks to be associated with pixels and generate a cluster label plane that is returned to the CAAPP through the backing store. The CAAPP's form connected components within 16×16 windows, and then perform region merging to remove the artificial seams of the 16×16 windows in order to produce the final segmentation.

Another part of the interpretation process involves line extraction. We use the Burns straight line algorithm [Burns, 1986] which begins by applying two 3×3 convolution windows to compute the Sobel gradient operator on the original image in the CAAPP (this step can actually be done in parallel with ICAP processing for the region segmentation). Edges are assigned coarse orientation labels by broadcasting a table of orientation ranges to the CAAPP processors. When a processor's value falls within an orientation range that is being broadcast, it stores the associated orientation label. Using the Coterie Network, a connected components labelling algorithm is run on the orientation label plane producing regions of pixels with similar gradient orientations, each with a unique label. Short lines (ie. regions with a small set of pixels of similar orientation) that result from this process are associatively selected and then saved for later use as a texture measure. The parameters describing the remaining "long" lines are transferred to the backing store so that the ICAP has access to them. The ICAP processors then compute for each region the parameters of a representative line by fitting a planar intensity surface to the pixel values in the region. The ICAP array links collinear segments that may have resulted from excessive fragmentation of longer lines in the original image. The result is a set of tokens that describe straight lines of various lengths which correspond to events in the image.

The next phase of processing results in the construction of a feature database for the extracted tokens–the Intermediate Symbolic Representation (ISR)–that is stored at the ICAP level. The CAAPP and ICAP together compute various feature values that describe the regions and lines that have been extracted from the image. For example, some features associated with a line might be its length, orientation, the contrast across it, adjacent regions, end points, etc.

At this point, the ICAP essentially takes over the processing. Our simulations indicate that the preceding operations take on the order of 20 milliseconds to perform. During the remainder of the scenario the CAAPP is free to receive another image and begin to do similar or other types of low-level processing in a pipeline fashion. This could involve stereo or motion analysis, or simply preparing the next frame for merging with the token database.

Next, the ICAP applies sets of object constraints retrieved from the knowledge base in the SPA shared memory and applies these constraints to the tokens in the ISR that are resident in ICAP memory in order to form initial hypotheses. Constraints are minimum and maximum values on token attributes that form a range on accepting tokens as object hypotheses. For each hypothesis, a score and threshold are generated within each ICAP for its token set. During this phase the ICAP processors are essentially running in MIMD mode.

The next major step involves geometric grouping of lines based on collinearity, parallelism, and orthogonality to abstract more complex geometric structures. The ICAP returns to synchronous-MIMD operation in order to facilitate the exchange of information between ICAP processors.

The last phase uses the results of the previous two phases to extend hypotheses, detect conflicts between them, and resolve those conflicts. It is at this point that the SPA enters the picture. The ICAP processors transfer selected token labels to the memory that is shared with the SPA. The different object schemas in the SPA apply various grouping strategies to the tokens by issuing commands to the ICAP processors that refer to the token labels and object verification strategies. Through a global blackboard the schemas incrementally attempt to resolve conflicts and find a consistent set of hypotheses with proper spatial and spectral relationships. Algorithms from the previous phases may be selectively repeated with different parameters and for different goals as different strategies for object verification are applied in different areas of the image to arrive at a consistent interpretation of the scene.

5. IMAGE UNDERSTANDING BENCHMARK PERFORMANCE

Figure 18 shows estimated execution times for the IUA on a suite of image understanding tasks that comprise the first DARPA Image Understanding Benchmark [Rosenfeld, 1987]. It should be noted that these tasks only exercise the CAAPP and ICAP levels of the IUA and, except for one instance, the tasks do not require inter-level communication. As such, they can only be viewed as isolated processing steps that will not give any real test of the integrated processing required for an interpretation. An Integrated Image Understanding Benchmark is currently in preparation that will address the issue of inter-level communication.

The times that are shown in Figure 18 are based, as much as possible, on execution of programs with our instruction-level simulator. In several instances, however, fully

simulating execution of the task was too time consuming, and we thus elected to simulate key portions of the task and extrapolate the total execution time.

The first two tasks begin with an 8-bit digital image of size 512×512 pixels. The first task involves convolution with an 11×11 mask and detection of zero crossings in the resultant image. The second task requires lists of the border pixels, as defined by the zero-crossings, to be output. Both of these tasks are performed by the CAAPP level of the IUA.

The next pair of tasks begins with a 1-bit digital image of size 512×512 pixels. In the first of these two tasks, all connected components which have an image value of 1 are grouped into regions that are assigned unique positive-integer labels. The second task involving the binary image requires the formation of the Hough transform of the input image, with the result being an array of size 360×512 where the first dimension represents theta and the second represents rho. Both of these tasks are also performed exclusively by the CAAPP.

The next three tasks begin with a set of 1000 real coordinate pairs defining 1000 points in a plane. The output of the first task is the convex hull of the set of points. The second task results in the Voronoi diagram for the points. The third task requires construction of the minimal spanning tree across the set of points. The ICAP level of the IUA was used exclusively for the convex hull and minimal spanning tree tasks. For the Voronoi diagram task we elected to use an algorithm that required the CAAPP and ICAP to work together (Other algorithms exist that could be run at either the CAAPP or ICAP level alone.) To faciliate execution by the CAAPP, we also chose to convert the real input representation (for which no precision was specified) into an internal 24-bit fixed-point representation. In the other tasks, it was assumed that a real value was in IEEE 32-bit floating point representation.

The vertex visibility task begins with 1000 triples of triples of real coordinates in the range of 0 to 1000. These coordinates specify the vertices of 1000 opaque triangles in three-dimensional space. The task requires a list of all vertices that are visible from the origin to be output. This task is performed using only the ICAP level of the IUA.

The last task in the figure required that a minimum cost path be found between two points in a graph with 1000 vertices of order 100 where the edges of the graph are weighted by non-negative real values. Only the ICAP level of the IUA was used in performing this task.

As can be seen from the figure, the IUA performed quite well on this suite of tasks. These timings do not take into consideration the fact that two thirds of the IUA was left idle in every case but one. The IUA could easily execute several of these tasks concurrently, without even making use of the SPA level.

	Task	Execution Time (milliseconds)	Processor
1.	Convolution, zero crossing	0.2	CAAPP
2.	Output border list	0.2	CAAPP
3.	Connected components	0.05	CAAPP
4.	Hough transform	27.0	CAAPP
5.	Convex hull	15.0	ICAP
6.	Voronoi diagram	50.0	ICAP/CAAPP
7.	Minimual spanning tree	124.0	ICAP
8.	Visibility of vertices	290.0	ICAP
9.	Minimum cost path	1.0	ICAP

Figure 18. Image Understanding Benchmark Performance

6. CURRENT IMPLEMENTATION STATUS

At present a 1/64th scale demonstration prototype of the IUA is being built at Hughes Research Labs. This vertical slice of the machine is scheduled for completion in 1988 and will include 4096 CAAPP cells, 64 ICAP cells, a single SPA processor and the ACU. The entire prototype will plug into a single-user workstation that will serve as a host (Figure 19).

The prototype will physically consist of a 16 by 20 inch mother-board with 83 daughter-boards attached to it. Eighty of the daughter-boards are 4 by 2.5 inches and the remaining three are 20 by 2.5 inches in size. Of the 80 small daughter-boards, 16 are for clock distribution and signal buffering. The other 64 contain the ICAP and CAAPP processors and their memories. The three larger daughter-boards provide the controller interface, feedback concentration, and ICAP communications network switching. The mother-board also includes a dual-ported frame-buffer memory that allows high speed image input and output.

Each processor daughter-board will contain a single custom VLSI chip, a TMS320C25, 256K bytes of static RAM, 384K bytes of dual-ported dynamic ram, and tri-state bus buffers. The single custom chip holds the 64 CAAPP processors with their local memories, the backing store controller, a refresh controller for the dynamic RAM, and arbitration logic for the various devices that must access the bus of the associated ICAP processor.

The custom VLSI chip is being designed in 2-Micron CMOS with 2 metal layers. A 32 processor chip has been fabricated through the MOSIS facility. A first run of the complete 64 processor custom chip is scheduled for Fall of 1987.

Our original software simulator [Weems, 1984a] has been re-written to run on a Texas Instruments Odyssey parallel signal coprocessor system in a Texas Instruments Explorer.

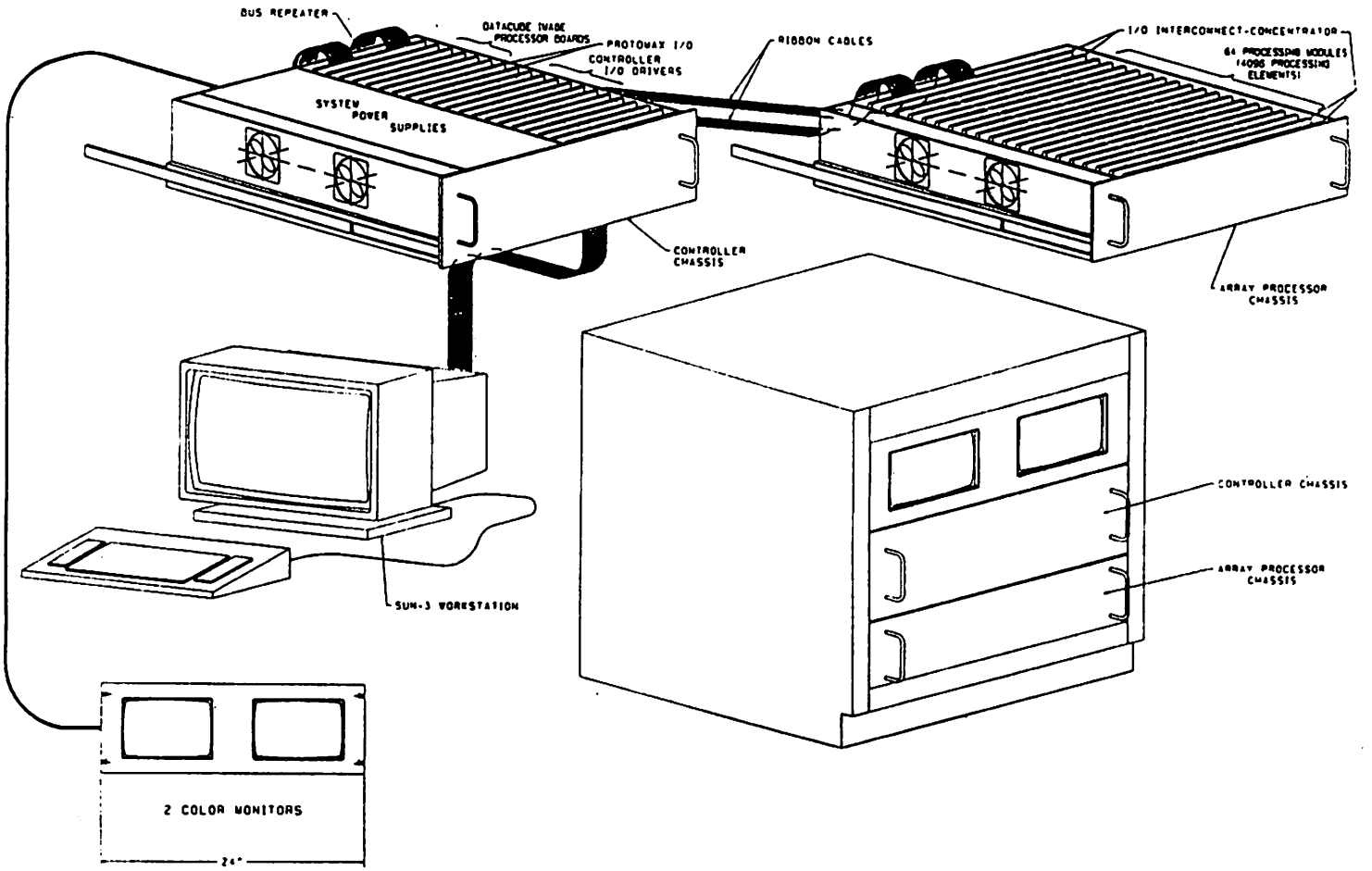


Figure 19. Image Understanding
Architecture Test Bed

The Odyssey, which contains four TMS32020 processors, simplifies emulation of the ICAP processor and greatly improves the execution times for CAAPP simulations over our VAX-based simulator. The Odyssey-based simulator provides the capability of one-fourth of an IUA mother-board, and also permits us to closely mimic the interactions of the three processing levels down to the signal level.

A VAX-based simulator has been written for the CAAPP, Coterie Network, and Backing Store. This simulator is written in C and is intended to be portable to other systems. For example, we are currently moving the simulator to a SUN-3 workstation.

7. FURTHER DEVELOPMENT

Several refinements are being considered for future versions of the machine, in particular to the ICAP communication structure and the CAAPP to ICAP control interface. One possibility being explored is to give the ICAP the ability of interacting with a local CAAPP controller. The CAAPP would then be able to execute in Multi-SIMD mode at the chip level: each 8×8 section could then execute a separate instruction stream. Another extension would be to augment the ICAP circuit switched network that is centrally controlled with a distributed control routing network. Finally, we are also examining the implications of augmenting the SPA shared memory with a Content Addressable Input Output Memory (CAIOM) [Levitan, 1984] to facilitate blackboard management.

8. CONCLUSION

The three-level structure of the Image Understanding Architecture supports the necessary hierarchy of abstractions for the different representations and operations that we believe are needed to generally solve the vision problem. Each level is constructed to perform a suite of tasks most appropriate for that level of abstraction.

The CAAPP is optimized to perform local operations on neighborhoods of pixels and to provide feedback to the higher levels of processing about the state of the computation and statistics about low-level data. It excels at very tightly-coupled fine-grained parallelism. The mapping of one pixel onto each processor ensures that the maximum amount of parallelism available in the low-level vision tasks will be utilized.

The ICAP is designed to support the necessary tasks of building an intermediate symbolic representation of the image and operating on that representation. These operations need two primary capabilities: data manipulation and communication. The data representations used by the CAAPP need to be transformed by the ICAP into a more accessible format, and then passed to neighboring ICAP cells to perform merging and grouping operations.

The high level tasks which perform knowledge-based inference and manipulation of object models are run in the SPA. To support distributed artificial intelligence process-

ing powerful processors are needed with large amounts of memory. The communication between processes will primarily be in terms of a blackboard system managed by the processes themselves. As these processes run and make requests via the ACU to the ICAP (and sometimes directly to the CAAPP) they will extract information about the image and post the results of their analysis on the blackboard for other processes to use. The end result will be an interpretation of the image achieved by cooperation of the set of object processes.

Beyond simply testing our hardware design, our ultimate goal for the prototype is to provide a powerful interim development environment for image understanding and parallel processing research. A simulated parallel processor has simply been too slow to permit any significant amount of experimentation. Once our prototype is running, we will be able to perform more total computation in the first few minutes of execution time than we have been able to do in our previous five years of simulation.

9. ACKNOWLEDGEMENTS

This research was supported, in part, by the Advanced Research Projects Agency of the Department of Defense via Contract No. F49620-86-C-0041 monitored by the Air Force Office of Scientific Research and under Contract No. DACA76-86-C0015 monitored by the Engineer Topographic Laboratory. We would also like to thank Dr. Caxton C. Foster for his long-term contribution of ideas in content addressable memories and processing that provide a foundation to this work.

10. 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.*,29:1-9, September 1980.

[Beveridge, 1987] Beveridge, J.R., Griffith, J., Kohler, R.R., Hanson, A.R., Riseman, E.M., Segmenting Images Using Localized Histograms and Region Merging (in preparation).

[Burns, 1986] Burns, J.B., Hanson, A.R., Riseman, E.M., Extracting Straight Lines, IEEE Trans. Pattern Analysis and Machine Intelligence, 8:425-455, 1986.

[Crowther, 1985] Crowther, W., Goodhue, J., Starr, E., Thomas, R., Milliken, W., Blackadar, T., Performance Measurements on a 128-Node Butterfly Parallel Processor, Proc. of the Intl. Conf. on Parallel Processing, 1985, pp. 531-540, IEEE Computer Soc. Press, Washington, DC.

[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, 1986] Draper, B., Hanson, A., and Riseman, E., A Software Environment for High Level Vision, COINS Technical Report, University of Massachusetts at Amherst, (in preparation) 1986.

[Draper, 1987a] Draper, B.A., Collins, R.T., Brolio, J., Griffith, J., Hanson, A.R., Riseman, E.M., "Tools and Experiments in the Knowledge- Directed Interpretation of Road Scenes", Image Understanding Workshop Proceedings, Morgan Kaufmann, Los Altos, CA, 1987.

[Draper, 1987b] Draper, B.A., Collins, R.T., Brolio, J., Griffith, J., Hanson, A.R., Riseman, E.M., The Schema System: Knowledge Based Vision (in preparation).

[Duff, 1978] Duff, M.J.B., Review of the CLIP Image Processing System, Processings of the National Computer Conference, 1978, AFIPA, pp. 1055-1060.

[Encore, 1986] Encore Computer Corp., promotional literature, Marlborough, MA, 1986.

[Erman, 1980] Erman, L., et al., The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty, *Computing Surveys*, 12:213-253, 1980.

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

[Hanson, 1984] Hanson, A. R. and Riseman, E. M., Preprocessing Cones: A Computation Structure for Scene Analysis, COINS Technical Report 74C-7, University of Massachusetts at Amherst, September 1974.

[Hanson, 1978a] Hanson, A. R., Riseman, E. M., VISIONS: A Computer System for Interpreting Scenes. In: *Computer Vision Systems*, A. R. Hanson, and E. M. Riseman (Eds.), New York: Academic Press, 1978, pp. 303-333.

[Hanson, 1978b] Hanson, A. R., Riseman, E. M., Segmentation of Natural Scenes. In: *Computer Vision Systems*, A. R. Hanson, and E. M. Riseman (Eds.), New York: Academic Press, 1978, pp. 129-163.

[Hanson, 1980] Hanson, A. R., Riseman, E. M., Processing Cones:A Computational Structure for Scene Analysis for Image Analysis. In: *Structured Computer Vision*, S. Tanimoto and A. Klinger (Eds.), New York: Academic Press, 1980.

[Hanson, 1986] Hanson, A. R., Riseman, E. M., A Methodology for the Development of General Knowledge-Based Vision Systems, (to appear). In: *Vision, Brain, and Cooperative Computation*, M. Arbib and A. Hanson (Eds.), Cambridge, MA: MIT Press, 1986.

[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.

[Kung, 1984] Kung, H.T., Menzilcioglu, O., Warp: A Programmable Systolic Array Processor, Proc. SPIE Symp. Vol 495, Real-Time Signal Processing VII, August 1984.

[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.

[Levitan, 1987] Levitan, S. P., Measuring Communication Structures in Parallel Architectures and Algorithms, (to appear). In: *The Characteristics of Parallel Algorithms*, L. Jamieson, D. Gannon, and R. Douglass (Eds.), Cambridge, MA: MIT Press, 1987.

[Nagin, 1982] Nagin, P.A., Hanson, A.R., Riseman E.M., Studies in Global and Local Histogram-Guided Relaxation Algorithms, IEEE Trans. Pattern Analysis and Machine Intellience, 4:263-277, 1982.

[NCube, 1985] NCube Corp, Promotional literature, Beaverton, OR, 1985.

[Nii, 1986] Nii, H.P., The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures, AI Magazine, Vol 7, Number 2, pp.38-53.

[Parma, 1980] Parma, C. C., Hanson A. R., and Riseman, E. M., Experiments in Schema-Driven Interpretation of a Natural Scene, COINS Technical Report 80-10, University of Massachusetts at Amherst, April 1980. Also in: *NATO Advanced Study Institute on Digital Image Processing*, R. Haralick and J. C. Simon (Eds.), Bonas, France, 1980.

[Pfister, 1985] Pfister, G., Brantley, W., George, D., Harvey, S., Kleinfelder, W., McAuliffe, K., Melton, E., Norton, V., Weiss, J., The IBM Research Parallel Processor Prototype (RP3): Introduction and Architecture, Proc. of th Intl. Conf. on Parallel Processing, 1985, pp. 764-771, IEEE Computer Soc. Press, Washington, DC

[Random 1968] The Random House Dictionary of the English Language, College Ed., L. Urdang, S. Flexner, Eds., Random House, N.Y., 1968.

[Rattner, 1985] Rattner, Justin, Concurrent processing: A new direction in scientific computing, Proceedings of the National Computer Conference, 1985.

[Reynolds, 1984] Reynolds, G., Irwin, N., Hanson A. R., and Riseman, E. M., Hierarchical Knowledge-Directed Object Extraction Using a Combined Region and Line Representation, *Proc. of the Workshop on Computer Vision: Representation and Control*, Annapolis, Maryland, April 30 - May 2, 1984, pp. 238-247.

[Rosenfeld, 1986] Rosenfeld, A., The prison machine: An alternative to the pyramid, J. Parallel and Distributed Computing 3:404-411.

[Rosenfeld, 1987] Rosenfeld, A., A Report on the DARPA Image Understanding Architectures Workshop, Proceedings of the 1987 DARPA Image Understanding Workshop, Morgan Kaufmann, Los Altos, CA, 1987.

[Seitz, 1985] Seitz, Charles L., The Cosmic Cube, Communications of the ACM, 28-1, January 1985, pp 22-33.

[Sequent, 1986] Sequent Computer Systems, promotional literature, Beaverton, OR, 1986.

[Tanimoto, 1983] Tanimoto, S., A Pyramidal Approach to Parallel Processing, *Proc. 10th Annual International Symp. on Computer Architecture*, Stockholm, Sweden, June, 1983.

[Uhr, 1972] Uhr, L., Layered Recognition Cone Networks that Preprocess, Classify and Describe, *IEEE Trans. Comp.*,21:758-768, 1972.

[Weems, 1984a] Weems, C. C., *Image Processing on a Content Addressable Array Parallel Processor*, Ph.D. Dissertation Computer and Information Science Department, also, COINS Technical Report 84-14, University of Massachusetts at Amherst, September 1984.(a)

[Weems, 1984b] Weems, C. C., Levitan, S. P., Foster, C. C., Riseman, E. M., Lawton, D. T., and Hanson, A. R., Development and Construction of a Content Addressable Array Parallel Processor (CAAPP) for Knowledge-Based Image Interpretation, *Proc. Workshop*

on Algorithm-Guided Parallel Architectures for Automatic Target Recognition, Leesburg, VA, July 16-18, 1984, pp. 329-359.(b)

[Weems, 1985] Weems, C. C., The Content Addressable Array Parallel Processor: Architectural Evaluation and Enhancement, *Proc. IEEE International Conference on Computer Design: VLSI in Computers*, Port Chester, New York, October 7-10, 1985. pp. 500-503.

[Weymouth, 1986] Weymouth, T. E., *Using Object Descriptions in a Schema Network for Machine Vision*, Ph.D. Dissertation, Computer and Information Science Department, also, COINS Technical Report 86-24, University of Massachusetts at Amherst, 1986.