Rationale For A Heterogeneous Parallel Image Understanding Architecture

Charles C. Weems

COINS TR 90-79

Computer & Information Science
Lederle Graduate Research Center
University of Massachusetts
Amherst, MA 01003

September 1990

Rationale for a Heterogeneous Parallel Image Understanding Architecture

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

ABSTRACT

This paper provides an overview of the architectural requirements for parallel processing in support of real-time, knowledge-based computer vision. One goal is to provide an appreciation for the diversity, complexity, and computational intensity of vision processing. We begin with a high-level description of common vision algorithms, analyze their requirements in terms of the inherent structures that are present, and how these are related to the processing elements, control of processing, and communication in parallel processing. Our conclusion is that these factors combine to defy solution with traditional architectural approaches to parallel processors, and that a heterogeneous parallel processor will best support the processing employed in vision.

Distinctly different forms of parallel processing can be applied at three levels of computational granularity, corresponding to levels of abstraction in the image interpretation process. Each level has its own unique computation, communication, and control requirements. In addition, interaction between levels must take place via parallel data and control paths.

The paper concludes with an overview of the Image Understanding Architecture (IUA), a multilevel parallel processor designed to meet the requirements of image understanding. A proof-of-concept prototype of 1/64th of the IUA is currently being constructed by the University of Massachusetts and Hughes Research Laboratories.

1. INTRODUCTION

Machine vision is one of the most computationally intractable domains of artificial intelligence research. A scenario with video input might require an interpretation to be updated 30 times per second, corresponding to video frame rate. Each color frame contains three quarters of a million color-intensity data values (pixels) at 512 x 512 resolution. Performing a single operation on each pixel translates to 23 million instructions per second. Of course, the computation to perform image interpretation is more than one operation per pixel. Many researchers believe that one hundred thousand times that amount is required. 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), a very great computational rate is nevertheless required.

A typical goal of an image understanding system is to construct a three dimensional model of the image sensor's environment. Such an interpretation may require the identification of hundreds of objects of many different types [Draper, 1989]. Other goals might include the determination of the

sensor's position with respect to landmarks; the identification of independently moving objects in the environment; or the detailed inspection of an object. Complicating factors include the presence of noise, occlusion, uneven lighting, shadows, specular reflections, and motion effects.

Vision researchers [Hanson, 1986] have shown that pattern recognition techniques, by themselves, are inadequate for this task. Consider an image of a window: Parts of the window can be reflective, transparent, or both; it also introduces distortion and specularity. Despite the fact that the window has no characteristic pattern, we perceive it as a separate object because of our knowledge of the properties of glass, and the window's surroundings. Even if the window were perfectly transparent, the presence of the window-frame would allow us to infer the existence of a pane of glass. In fact, much of what we "see" in natural scenes is really inferred from partial information.

It is clear that vision involves both sensory and knowledge-based processing. However, between these two levels of abstraction it is useful to introduce one or more levels of symbolic processing. Symbols range in complexity from descriptions of extracted image events (such as edges or regions) through perceptually useful groupings of events (such as geometric figures or surfaces), to abstract descriptions of object parts or entire objects. Although the level of abstraction of these intermediate representations varies, they share a similarity of structure and processing. Thus, vision researchers tend to classify algorithms and representations into three levels: low (sensory), intermediate (symbolic), and high (knowledge-based). For the computer architect, a vision system therefore presents three distinct sets of requirements.

The requirements at each level can be broken into three categories: computation, communication, and control. Computational requirements are related to the operations and data types most frequently used in a level. Communication requirements depend upon the type of data used, and how it is combined, replicated, or shared within and between levels. Control requirements are based on the algorithmic control structures and data structures used within a level, and the interdependencies between operations both within and between levels.

2. REPRESENTATIONS, OPERATIONS, AND ALGORITHMS USED IN VISION

A survey of the literature [Hanson, 1978b, Ballard,1982, Nevatia, 1982, Uhr, 1986, Kanade 1987, Arbib, 1987, Levialdi, 1988] reveals that vision researchers use many techniques. This section lists some of the operators, algorithms, and representations that are used. It is not a complete survey of every approach being used. However, the following tables are representative of a majority of the work being done.

We begin with commonly used representations of scene data, model data, and world knowledge. It should be noted that there is some disagreement among researchers as to the level of abstraction that should be assigned to each type of representation. This disagreement stems in part, from the fact that a representation may be used by algorithms at more than one level. For example, a low-level edge detector may output a list of edges that is operated on by both a low-level edge-linking process and an intermediate-level geometrical grouping process. In a sense, the edge list is used both at the low and intermediate levels.

For the purpose of creating a loose taxonomy, the levels of abstraction will be somewhat arbitrarily defined in the following way. If a representation is based on arrays of numerical data that are in registration with or correspond directly to image data, then it is low level. Those representations based on symbolic descriptions of extracted image events, or view-specific symbolic instantiations of stored models and knowledge are intermediate level. Representations that are view- or scene-independent are high level. A high-level representation thus characterizes general models and knowledge, as well as view-independent 3-D models and knowledge of the current environment.

The following table lists common representations, categorized according to these definitions of the low, intermediate, and high levels.

Low Level
Images (arrays of pixels representing intensity, depth, color, etc.)
Labeled images (regions, edges, vertices, surfaces, texture, etc.)
Expanded or contracted images (multiresolution images)
Motion or stereo displacement arrays, and normalized flow-fields
Fourier-space arrays, Hough-space arrays, feature-spaces, etc.
Images projected from stored models
Intermediate Level
Local and global statistical measures and histograms representing general characteristics of an image or scene, coordinates of focus of expansion, motion parameters, independent trajectories
Symbolically coded edges and lines (straight line, spline fit, chain-code, width, contrast across, end points, etc.)
Symbolically coded regions (convex hull, quad-tree, Euler number, spectral measures, texture measures, medial axis, etc.)
Symbolically coded surfaces (plane fit, generalized cylinder, generalized cone, polygon list, etc.)
Groups of coded edges, regions, surfaces, etc. (lists of related tokens, graph structures describing relationships, etc.)
Symbolic projections of stored models as edges, regions, surfaces, groups of tokens, etc. corresponding to the current view
High Level
Semantic networks
Production rules, expert knowledge
Control and interpretation strategies (procedural knowledge)
Knowledge sources (general, object specific, token specific, etc.)
Hypotheses (object identification, relational, behavioral, etc.)
Constraints
Stored plans and current instantiations of plans
General object models (physical, behavioral), maps, etc.
Environmental models (instantiated object models and relationships, current behavior, predicted behavior, etc.)

Table 1. Representations Used at Different Levels of Abstraction in Vision

Operations and algorithms applied to the various representations are considered next. These are again divided into low, intermediate, and high levels. However, here the distinction is less clear. Obviously, if an algorithm takes an image as input and produces an image as output, it is low level. But what is an algorithm that spans the low and intermediate level representations, transforming an image into a symbolic representation? For simplicity, an algorithm is defined to be at the level of its input representation, since most algorithms work predominantly with their input, generating the output at the end of some sequence of steps.

The definition is neither complete nor robust. For example, an algorithm that takes image data and integrates it with an existing list of edges would be both low and intermediate, because it has inputs that are at both levels of abstraction. In many cases, it may be possible to split the algorithm into two logical parts, one at each level. An algorithm that immediately transforms an image into a symbolic

representation that is used for the majority of its processing also does not fit the definition of low-level processing, but could be similarly split into low and intermediate parts. Although the following lists attempt to place each algorithm or operation into one level, it must be remembered that some of them span levels. In analyzing architectural requirements, it is necessary to decompose those algorithms into their component operations at each level.

There are three categories of low-level processing that seem to outnumber all others: transformation operations, edge detectors and operators, and region segmentations and operators.

Transformations	Edges	Regions
Threshold	Kirsh Operator	Connected Components
Gaussian Convolution	Sobel Operator	Local Histogram Segment.
Laplacian Convolution	Roberts Cross	Feature Space Clusters
Difference of Gaussians	Canny Operator	Multiresolution Segment.
Median Filter	Nevatia-Babu Operator	Region Growing
Bandpass Filter	Hough Transform	Region Splitting
Histogram	Template Matching	Blackboard Based Segment.
Statistical Measures	Edge Linking	Expand and Contract
Covariance	Edge Thinning	Medial Axis
Correlation	Contour Tracing	Bounding Rectangle
Fourier Transform	Chain Coding	Convex Hull
Gray-level Correction	Spline Fitting	Area
Hadamard Transform	First Difference	Eccentricity
Hilbert Transform	Fourier Curve Fitting	Euler Number
Walsh Transform	Conic Curve Fitting	Compactness
Kalman Filter	End-point Fitting	Quad-tree Encoding
Predictive Compression	Average Contrast	Region Adjacency List
Geocorrection	Length	Concavity Tree
	Straight Line Fitting	Boundary Tracing

Table 2. Examples of the Three Most Common Classes of Operations in Low Level Vision

In addition to these three large groups, there are low-level operations related to stereo, motion, texture, surfaces, and others that are not so easily classified.

Surface	**Stereo**
Surface Detection	Correspondence Matching
Shape from Shading	Stereo Displacement Field
Shape from Texture Gradient	Depth from Stereo
Shape from Depth Map	**Texture**
Plane Fitting	First Order Texture Measures
Generalized Cone Fitting	Second Order Texture Measures
Fourier Shape Description	Energy Measures (Entropy, etc.)
Motion	Fourier Texture Measures
Moravec Interest Operator	Markov Random Field Measures
Flow Field Normalization	Structural Texture Descriptions
Flow Field Decomposition	Texture Segmentation
Focus of Expansion Determination	**Miscellaneous**
Depth from Motion	Orthogonal & Oriented Windowing
Motion Detection	Corner Detection
Moving Point Trajectory	Std. Deviation of Euclidean Distance

Table 3. Additional Operations Found in Low Level Vision

The intermediate level is where instantiated models and extracted image events meet. In many cases, the image events must be grouped together or organized to facilitate matching with models. Models must usually be transformed to match them with structures derived from the scene. The representative intermediate level operations are thus divided into three sets: grouping, model transformations, and model matching. There are other intermediate level operations, but these three types account for the majority.

Grouping	**Model Matching**
Parallel Lines	Topological Match
Orthogonal Lines	Geometric Match
Vertex Grouping	Subgraph Isomorphism
Line Classification	Double Subgraph Isomorphism
Colinear Lines	Boundary Evaluation
Coplanar Surfaces	Association Graph Construction
Intersecting Lines and Surfaces	Merit Function
Temporal Grouping	Matching Metrics
Model Transformations	Relaxation Labelling
Scaling	Clique Finding
Translation	Minimum Cost Path
Rotation	Minimum Spanning Tree
Perspective Projection	Temporal Matching

Table 4. Common Operations in Intermediate Level Vision

The high level is responsible for maintaining long-term knowledge, and representing the short-term model of the current environment and the system's status. It must select candidate models and instantiate them for matching, control the matching process at the intermediate level, evaluate the matches to generate identification hypotheses regarding scene elements, compare hypotheses to detect conflicts, control the resolution of conflicts, combine information from hypotheses to verify or refute them, and infer missing portions of the environmental model from partial interpretations. If

the system is endowed with actuators, the high level may also plan and carry out actions within the environment.

Semantic Network Traversal	Parameter Optimization
3-D Model Construction	Relational Matching
Model Driven Control	Production Rules
Inference	Backtrack Searching
Network Predicates	Logic
Hypothesis Generation	Theorem Proving
Hypothesis Verification	Cooperative Problem Solving
Belief Maintenance	Constraint Satisfaction
Goal Achievement	Planning
Learning	Subgoal Establishment

Table 5. Common Operations in High Level Vision

3. A VISION PROCESSING SCENARIO

To establish a context for the following discussion, one sequence of operations is described that could be used to interpret an image. This is an oversimplification of how the UMass VISIONS system works [Hanson, 1978a, Draper, 1987]. It would be impossible to completely discuss the full interpretation process within the space of this paper. The purpose here is to show the types of processing and interactions that take place in a context larger than a single algorithm.

The processing is initiated with a region segmentation, the first step of which is an edge-preserving smoothing operator that involves a few iterations of a 3 X 3 window convolution. The next step is segmentation [Nagin, 1982, Beveridge, 1987], which detects clusters in 1-dimensional local histograms within 16 X 16 subimages. A local count is recorded for each range corresponding to buckets in a local histogram. (Actually, the algorithm utilizes windows that overlap by 4 pixels in each direction, forming 24 X 24 pixel histograms and requiring more complex communication than there is room to discuss.) Next, the histograms are searched for peaks and valleys, by applying criteria defining clusters of values. Communication with processing in neighboring subimages is used to consistently extract labels of peaks to be associated with pixels in windows, and generate a cluster label plane. Connected components are formed within the 16 X 16 windows, and then region merging is performed to remove the artificial seams of the 16 X 16 windows to produce the final segmentation.

Another part of interpretation involves line extraction. The straight line algorithm [Burns, 1986] begins by applying two 3 X 3 convolutions (the Sobel gradient operator) on the original image. Edge pixels are then assigned coarse orientation labels. Connected components labelling of the orientation label plane produces regions of pixels with similar gradient orientations, each with a unique label. Short lines (i.e.. regions with a small set of pixels of similar orientation) that result from this process are selected and then saved for later use as a texture measure. The parameters describing the remaining "long" lines are then computed by fitting a planar intensity surface to the pixel values in each region. Collinear segments that may have resulted from excessive fragmentation of longer lines in the original image are linked together. The result is a set of tokens that describe straight lines of various lengths, corresponding to events in the image [Brolio, 1989].

The next phase is construction of a feature database at the intermediate level, consisting of the extracted regions and lines represented as tokens with feature vectors. For example, features

associated with a line token might be its length, orientation, the contrast across it, adjacent regions, end points, etc. At this point, processing essentially shifts to the intermediate level and additional low-level processing could take place in a pipeline fashion; which could involve stereo or motion analysis, or simply preparing the next frame for merging with the token database.

The next major step involves geometric grouping of lines based on collinearity, parallelism, and orthogonality to establish more complex geometric structures [Beveridge, 1987, Boldt,1989]. Then, sets of object constraints derived from the knowledge base are applied to the intermediate-level tokens to form initial object hypotheses. Constraints are expressed as minimum and maximum values on token attributes that form a range defining acceptable tokens for initial object hypotheses [Hanson, 1987]. For each hypothesis, a score and threshold are generated for each token set.

The last phase uses the results of the previous two to extend hypotheses, detect conflicts between them, and resolve those conflicts. It is at this point that high-level processing takes a greater role. Triggered by the initial object hypotheses (and later by expectations), different object schemas [Draper, 1987] each apply grouping strategies to the tokens to verify existing hypotheses and establish new ones. These strategies use the token sets at the intermediate level both singly and in combination, depending on the known or expected relationships between the features describing the object, by issuing commands that refer to the token labels and other verification strategies. Through a global blackboard the schemas incrementally 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.

4. ANALYSIS AND SUMMARY OF REQUIREMENTS

4.1. Methodology

One could analyze the architectural requirements of the algorithms listed in section 2 using traditional methods, by embedding them in applications and gathering statistics on instruction set usage and memory access patterns. However, these methods assume a basic machine organization has been defined, and guide the optimization of design parameters within that organization. The result would probably be a von Neumann organization tailored to be significantly faster for vision than standard uniprocessors. But, even the most highly tuned von Neumann machine will fail to meet the computational demands of vision.

The design of a successful vision machine requires the architect to take a more abstract view of the design problem. Vision, both in humans and artificial systems is a highly structured mechanism that lends itself well to massive parallelism in a wide variety of forms. *The challenge to the architect is to capitalize on the structures present in vision to obtain an optimal machine organization.* At the same time, there is the need to balance the use of special hardware with generality and flexibility, because vision is an evolving discipline.

This section examines the operations, algorithms, and representations identified in section 2, by noting the structures that are present. In each case, three types of structure are considered: data structure, control structure, and communication structure. In this context, a data structure is any organization of the data that facilitates and simplifies processing or that has the potential for parallelism. A control structure is any organization of control that is regular or has separable components. A communication structure is any pattern of data movement or combination.

Once the inherent structures are identified, an organization can be established for a vision architecture. Then, having reduced the design space to an organization, traditional approaches can be used with appropriate modifications to tune the architecture for better performance. It should be noted that in this article, our goal is to establish a qualitative appreciation for the requirements of vision. The detailed quantitative analysis of the requirements would fill many more papers [Weems, 1988, 1990d].

4.2. Low Level Requirements

Starting with the low level, the first obvious structure is the two-dimensional array. The array stems directly from the physical structure of typical sensors. Array elements resulting from sensory input are typically 8-bit integer values, although some sensors have more or less precision, and a few output floating point. Computed values often require more precision than sensory data (for example, the weighted sum produced in a convolution, prior to normalization), and may have sufficient dynamic range to warrant scaled integer, fixed point, or floating point. On the other hand, many operations result in a binary (one bit) image. Pixel coordinates are also used; in a 512 x 512 image each ordinate is a 9-bit value. Color is usually represented by a triad of sensor values, for example, a 24-bit quantity consisting of three 8-bit integers, one each for red, green, and blue intensity. Orthogonal and oriented windowing are sometimes used to restrict processing of other operators to a particular region of interest in an image.

Thresholding, gray-level correction, and dynamic range compression and expansion are examples of pointwise operations on arrays; these operations involve only the communication of global quantities to each of the elements in an array, or communication between corresponding elements of arrays of identical size and shape. Convolutions, such as the Gaussian, Sobel, Kirsh, etc. involve combining an array element's value with those of its neighbors, often modified according to a mask. The mask is typically square, and commonly varies in size from 2 x 2 to 21 x 21 elements. The median filter similarly requires communication in a local neighborhood, but involves sorting the neighboring values instead of applying arithmetic functions. Altogether, several hundred pointwise operations and dozens of convolutions may take place in the interpretation of one image.

Contour tracing, chain coding, corner detection, edge length, edge linking, etc. require that elements be assembled into roughly pixel-wide chains corresponding to edges or discontinuities in an image. Information is passed between neighboring elements within chains, sometimes from one end of an open chain to the other end, or between nearby chains. Computing the average contrast across an edge requires values to be combined from image elements adjacent to the edge elements, and then the combined values are averaged along the length of the edge chain. These operations are usually performed on all edges in an image, or some selected subset of the edges. The various curve fitting operations gather up the elements that constitute an edge and compute the best fit of a particular type of curve to those points. The result is an intermediate level representation of the edge, i.e. as a token with a set of descriptors. Interpreting one image may require all these operations to be performed, some more than once.

The region oriented operations divide the image array into irregularly shaped, contiguous groups of elements. Connected components labelling propagates labels within these groups. Local histogram segmentation establishes region groups by computing histograms within small, overlapping windows to identify local features that are then labelled. The local feature labels are then merged with those in adjacent windows in order to deal with regions that cross window boundaries. Expand, contract, and medial axis, operations require processing (comparison with neighbors, and relabelling) to take place at region boundaries. The convex hull also requires boundary processing, but communication is along the boundary, rather than across it. Operations such as area, compactness, eccentricity, and Euler number collect information from each region to produce

descriptive scalar values. Determining the adjacent regions involves gathering labels from elements adjacent to region boundaries. Transforming a region into a quad-tree representation gathers information from windows of varying size that overlay the region. Again, an interpretation can require all of these operations, along with other region operations that have not been mentioned. Some of the operations, such as local histogram segmentation, may be repeated several times for different image features.

Some operations result in a non-image array, such as the Fourier and Hough transforms. The Fourier requires a complex pattern of data exchanges across the array, but this long-distance communication is amenable to off-line optimization. The Hough transform also combines data across the array into different locations in the resultant array, but the pattern of access is data dependent and must be established on-line. An interpretation may involve tens of these non-image transforms.

The result of statistical measures (existence, count, mean, variance, standard deviation, etc.) are single values that characterize some set of array elements. These are typically used in decision making processes and should thus be computed quickly, even though doing so requires that information be gathered from the entire array. A similar operation is the histogram, which divides a set of elements into subsets that are counted. The histogram array is typically one-dimensional, but multi-dimensional histograms are also employed. Usually, the histogram is then searched for peaks and valleys that denote features in the image. An interpretation can require tens to hundreds of statistical measures to be computed, depending upon the application.

The various "shape-from-x" and surface detection operations usually involve computing a gradient from some image feature, segmenting the gradient image, and then attempting to fit a surface to regions of the gradient. The first step typically gathers information from local neighborhoods. The second step may involve local histogram segmentation, or connected component labelling. The last step requires information from a region to be gathered so that a surface can be fit to it. These operations may be repeated for different features during the interpretation.

Motion and stereo are closely related in the types of processing performed. Each begins by computing the displacement between features in two (or more) images. The features can be dense, as with pixel registration, or sparse, as with matching edges or points of high curvature. Constraints on the matching between images may be available (e.g. epipolar lines in stereo). A displacement field (for stereo) or flow field (for motion) is the result. Normally the camera parameters are known in stereo, and a depth map may be directly computed from the displacement field. When the parameters of motion are known, it is similarly possible to directly compute a depth map. However, if they are not known, then they must first be extracted from the flow field. One approach to motion parameter extraction begins with smoothing the flow field by averaging neighboring flow vectors. The smoothed flow field is decomposed into translational and rotational motion components via some global optimization algorithm. It should be noted that there are ambiguous cases that can not be distinguished for six-degree-of-freedom motion, but typically some approximation of the camera motion is known and used to resolve the ambiguities. Usually, stereo and motion processing only occur once as part of an interpretation. However, the resulting depth map can require processing, segmentation and analysis that is comparable in complexity to that for the original image.

Simple approaches to texture use arithmetic operations on local neighborhoods of pixels to determine features. Another approach is to use template matching within local neighborhoods to identify specific textures of interest. More global approaches, such as Fourier measures and Markov Random Fields, identify patterns that repeat across larger areas of an image.

To briefly summarize, low level vision is dominated by the processing of arrays of numerical information. These are most frequently image-sized arrays, whose elements are integers of varying precision, mostly binary or in the range of eight to sixteen bits. However, floating point, fixed

point, and scaled integer representations are also used. Computations are predominantly standard arithmetic and Boolean operations, although square roots, trigonometric functions, and sorting are not uncommon.

Operations take place between corresponding elements of identically sized arrays, between elements and their neighbors, between elements and the elements of mask arrays, and between elements and global constants. This implies the need for local neighborhood communication, and broadcast of values. Other operations require data to be extensively rearranged in the array, both in predetermined and data dependent patterns. Statistical measures and histograms impose a requirement for quickly condensing data from an entire array, from a subset of array elements, or within independent edges or regions.

Generally, low level operations have a single control thread, or at most a few potential threads depending on branches in the control structure. For example, if an algorithm processes those pixels with values above a threshold one way and those below the threshold in another way, there are two potential control threads. Furthermore, the single thread may require that different values be broadcast within regions or edges, or that the same statistical measure be independently computed across regions or edges.

Lastly, there is a requirement for rapid input of sensory data. Obviously, no matter how fast the processor, it will not achieve real-time performance if it must spend all of its time reading in the data. What is not so obvious is the flexibility that is required in performing input. For example, a vision system may sample the image stream at less than frame rate, say every tenth of a second. Or, a system may need to vary its sampling rate, as when a time-consuming interpretation causes it to fall behind and it must subsample the image stream to catch up. Another useful feature is the ability to look backward in time. For instance, when an independently moving object is identified, reviewing previous frames may help to refine its trajectory. For development and experimentation, real-time output is highly desirable, especially with the capability to drive multiple display devices simultaneously so that partial results can be viewed in relation to each other.

4.3. Intermediate Level Requirements

Intermediate level data types are more varied than at the low level. Edges, lines, vertices, regions, and surfaces may be represented by record structures, linear lists, linked lists, quad trees, etc. These symbolic representations will be referred to here as tokens. Groups of tokens may be stored in linear or linked lists, trees, or general graph structures, depending upon the relationships to be represented. Models are represented in many ways, such as polygon lists, wire frames, parametric surfaces, etc. Once instantiated and projected to facilitate matching, their representations are similar to tokens.

Grouping of tokens takes many forms. For example, edges may be grouped in a specific part of the image to form a particular figure; collinear lines may be grouped across the image to infer continuous lines occluded by a foreground object; surface patches may be grouped into larger surfaces or geometric arrangements; edges and surfaces may be grouped to facilitate fitting specific curves; tokens from successive frames may be grouped to determine trajectories of objects; etc. All of these processes involve selection of tokens via partial matches of their attributes, which is equivalent to associative retrieval. The selected tokens are then linked together into larger structures. Tokens may be grouped locally (when one of the constraints is spatial distance) or globally. One interpretation can require dozens of different grouping operations.

Thus, the operations required for grouping are storage, access, creation, and structuring of tokens, comparison of token values with global values, computation of distances, computation of angular and other relationships, optimized geometric fits to sets of tokens, computation of intersection points and lines, comparisons between token values, gathering tokens together locally and globally,

inserting tokens into lists, and linking tokens into tree and graph structures. The transformation of tokens from one representation or structure to another must also be supported.

The processing required for model transformations depends on the model representation. For example, scaling, rotation, and translation may all be elements of the formula for a parametric surface representation. However, instantiating and projecting the 3-D surface model for 2-D matching may require the evaluation of a higher-order polynomial at many points. On the other hand, rotation, translation, scaling, and projection of a wire frame model may be accomplished with fairly straightforward matrix arithmetic. It may be necessary to instantiate tens to hundreds of models for one scene, depending upon the application.

Matching generally requires element by element comparison of a 2-D model-based structure with a token structure originating from image data. The elements of a match are traversal of the two structures, comparison of elements or evaluation of their similarity, and global summary of multiple matches to make a final selection. The traversal of the structures is complicated by their differences. Usually, the structure derived from the image has both missing and extra components in comparison with the model-based structure. In addition, the token structure may provide alternatives for each element, or elements may have associated uncertainty values. Matching is thus usually part of an optimization process, using one of the popular approaches. As a result, dozens of different matches may have to be tried for each object in a scene, to complete an initial interpretation.

Comparison of elements can involve lengthy computations that result in multiple matching parameters. Traversal of structures can range from simply accessing a linear list to the double subgraph isomorphism problem. In practice, other constraints usually limit the number of graph traversals required.

Control at the intermediate level is more complex than at the low level. Multiple threads of control are the rule, rather than the exception. However, those threads may or may not be independent. For example, in grouping collinear lines, there will usually be a phase of comparing lines and selecting best candidates, followed by a combining phase. Within each phase, separate threads of control can be used to process independent groups of lines, but until all candidate groupings have been evaluated, the combining phase cannot begin. There are many instances at the intermediate level where the same sort of operation is applied to a set of tokens, but each token is dealt with differently. Thus, there are potentially many control threads that maintain a loose global synchronization.

Other operations at the intermediate level may be completely independent. For example, the high level may direct the intermediate level to match a set of independent models against token data. These operations can have separate control threads. Of course, each match may also involve multiple, possibly synchronous, threads. It should also be noted that some of the associative retrieval operations are simple enough to require only a single thread of control. Thus, the intermediate level should provide for tightly synchronized, loosely synchronized, and unsynchronized control.

4.4. High Level Requirements

Processing at the high level is not as well defined as at the low and intermediate levels. The is insufficient space to describe multiple approaches to high level vision, so the blackboard model [Erman, 1980, Nii, 1986], will be the only one used in the discussion that follows. In particular, its use in the UMass VISIONS System [Parma, 1980, Weymouth, 1986] is examined, although similar operations occur in other blackboard systems. Because a vision architecture must also support the other approaches, the best architectural strategy, is to maximize generality and flexibility in high level processing.

The blackboard model involves independent, concurrent, knowledge-based processes, that communicate via a central message facility (the blackboard). These entities are called "schemas" or "frames" and have access to a set of "knowledge sources". Schemas, knowledge sources, and the blackboard may all be arranged hierarchically. A knowledge source is a code module having access to information that it can manipulate at the request of a schema. Depending upon the complexity of an application, a system may have only a few knowledge sources, or hundreds of them. A schema uses information from knowledge sources to produce "hypotheses," which it posts on the blackboard. A typical system may have tens or hundreds of schemas, usually one for each class of object, with subsidiary schemas for recognizing object parts. For example, a "road line" schema sends a request to an "edge" knowledge source for all parallel edges that meet certain constraints. The schema combines this with color and region data from other knowledge sources to hypothesize lane division stripes in a road scene. The hypothesis is posted on the blackboard, and used by a "road" schema to support to its hypothesis for a road in a certain part of the scene.

Once a hypothesis is posted, it is tested for conflicts with others on the blackboard. A schema may have several strategies for resolving a conflict, such as requesting more information from a knowledge source to strengthen its hypothesis or withdrawing a weak hypothesis if the conflicting one has a high measure of confidence. Schemas may also examine the blackboard for information that helps to confirm or refute a hypothesis.

Each schema performs a unique set of operations in generating hypotheses; which may be as simple as evaluating a set of production rules, to following a very elaborate and computationally intensive strategy for evaluating data from knowledge sources. Posting a hypothesis on the blackboard involves modifying a shared data structure, with all the requisite precautions for avoiding access conflicts.

Schemas may be responsible for selecting models from the knowledge base and instantiating them, or possibly directing their instantiation by the intermediate level. In a sense, the knowledge base is a high level knowledge source. Other knowledge sources operate at the intermediate level (performing grouping and matching operations, for example), or even at the low level (such as resegmenting a portion of an image). In general, the high level can make use of any processes at either the low- or intermediate-levels via the knowledge source mechanism. Regardless of the level, the schemas control the functioning of the knowledge sources.

Each schema can have an independent thread of control, with minimal synchronization overhead for accessing the blackboard. Within schemas there are potentially multiple control threads, such as the parallel evaluation of production rules. Each knowledge source can also be considered an independent thread spawned by a schema. Searching of the blackboard or the knowledge base is another potential source of parallelism. Asynchronous communication between schemas and knowledge sources, schemas and the blackboard, and between schemas and the knowledge base are implicitly required.

To summarize, the high level must support intensive arithmetic, logical, and symbolic operations on structured data types, some of which are shared. It must be capable of extracting information from and controlling processing at the intermediate and low levels. It must be able to quickly search both the shared blackboard and the knowledge base. The high level must support independent control threads and communication mechanisms.

4.5. General Requirements and Issues

In this section a few general issues are presented, some being implicit in the preceding discussion, and others not previously addressed. It has been implicit that a vision architecture must

simultaneously maintain the low, intermediate, and high-level representations and support processing at all levels simultaneously. The processing at the different levels can overlap, and doing so is probably a prerequisite for real-time performance.

Also implicit is application of different types of processing to selected subsets of the data in the low level. As at the intermediate level, this is equivalent to associative retrieval and processing. Another requirement is for a global control mechanism to manage processing within and between levels; this mechanism may be centralized, or distributed, or a combination of the two.

Hard real-time constraints demand not only high mean performance, but predictable performance. For instance, sophisticated, multi-level caches provide high performance on average, but a cache miss may have a significant performance penalty and cannot be predicted. Thus, a hard deadline can only be guaranteed under the assumption of worst-case cache access, defeating the purpose of the cache. Similarly, elaborate instruction pipes that have to flush and restart at random also pose a problem in real-time systems.

Real-time processing also requires that global summary information, such as statistical measures at the low level, be computed quickly to facilitate control decisions. In a machine whose operands are images, a global summary operation is equivalent to a conditional test and branch in a uniprocessor. For a uniprocessor engaged in a real-time application, a very slow test and branch operation would be disastrous. The same is true of an image processor, and of global summary operations at the intermediate level.

Beyond these requirements, two significant architectural implications can be derived from the preceding discussion. First, each level of abstraction requires different types of computational elements, different control mechanisms, and different forms of communication. Second, highly parallel communication and control is required *between* the levels.

4.6. Summary of Requirements

For ease of reference the computation, communication, and control requirements of the three levels are summarized in the following tables.

Low Level	Intermediate Level	High Level
Fine grained	Medium grained	Coarse grained
Arithmetic	16-bit integer arith.	32-bit integer
Logical	32-bit integer/f.p. arith.	32-bit floating point
Comparisons	Symbolic processing	List processing
8- to 16-bit integer	Record processing	Graph processing
Real, floating point	Compare/matching	Process creation & control
Arrays (image size)	Graph processing	Knowledge base proc.
Global statistics	Geometry	Rule evaluation
Local statistics	Matrix arithmetic	

Table 6. Computational Requirements of Each Level

Low Level	Intermediate Level	High Level
Short messages	Medium messages	Long messages
Global broadcast	Global broadcast	Shared structures
Local broadcast	Subset broadcast	Blackboard
Global summary	Global summary	Knowledge base access
Local summary	Down to low level	Down to lower levels
Neighborhood	Neighborhood	Knowledge sources
Over image elements	Over lists, graphs	With actuators
Off-line routing	Up to high level	With global control
On-line routing	Collection (grouping)	With sensor controls
Fast, flexible I/O	Dense routing	
Up to intermediate		

Table 7. Communication Requirements of Each Level

Low Level	Intermediate Level	High Level
Fine grained	Medium grained	Coarse grained
One or few threads	Related threads	Independent threads
Associative select	Independent threads	Distributed control
Multi-associative select	Associative select	Asynchronous
Synchronous	Synchronous	Data locking, sharing
Central control	Asynchronous	Process spawning
Coordinate w/ int. lev.	Central control	Resource allocation
High level control	Distributed control	Manage lower levels
Input and output	High level control	

Table 8. Control Requirements of Each Level

5. AN ARCHITECTURAL OUTLINE

5.1. Homogeneous Versus Heterogeneous Approaches

Vision systems can be mapped into many types of parallel architecture. For example, it might be argued that the diversity of processing demands a general-purpose multiprocessor; but this strategy is probably suboptimal at the low level which is characterized by data parallelism. Greater efficiency would be obtained at the low level by devoting equivalent resources to data-parallel hardware. On the other hand, a data parallel array does not support the independent control threads found in high-level processing. Clearly, no homogeneous parallel processor can simultaneously support all of the levels with optimal efficiency, as is necessary to achieve real-time vision.

The alternative to a homogeneous system is a heterogeneous one. It is possible to design a parallel processor for each level that is tuned to that level's requirements. Combining multiple tuned parallel processors also has the advantage that it allows processing to take place optimally at all levels simultaneously. However, one cannot simply connect three independent parallel processors together as this is unlikely to provide the necessary inter-level communication and control. The processors must be designed to include mechanisms for interacting with the other levels.

5.2. Parallel Communication and Control Between Processing Levels

A central characteristic of image interpretation is the bi-directional flow of communication and control between all levels. Specific communication operations must be performed rapidly for different interpretation strategies to be tried dynamically within one or a few frame times.

In the upward direction, communication consists of image abstraction and segmentation results, grouped collections of lower-level abstractions, plus 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 control of knowledge-directed processing and grouping operations, commands for selecting subsets of the image to constrain processing, modification of parameters of lower-level processes, and requests for additional information from the intermediate representation. Thus, the tendency is for data to flow upward and control to pass downward.

Communication between processing levels involves rapid transfer and evaluation of information from lower to higher levels, and the ability to exercise control from higher to lower levels. Based on our experience with highly parallel algorithms, we believe that associative processing [Foster, 1976] is a key element of the communication paradigm between each pair of levels and the control paradigm for the low and intermediate levels in the hierarchy.

In an associative system, constraints can be broadcast for selecting pixels, or symbolic tokens for selective processing. In this way higher levels control lower levels. Many associative systems also provide special hardware to test and/or count the response that comes after processing data. These global summary mechanisms allow fast conditional testing so that decisions can be made quickly. Thus, the lower levels provide feedback to higher ones.

Global summary mechanisms provide one way flow of information upward. However, when a large number of values must be transferred between levels, a parallel data path between adjacent levels is required. For example, by connecting spatially collocated processors in adjacent pairs of levels with independent data paths. Parallel data transfer permits the use of a processing strategy in which each level transforms 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 which treats the level below as an associative data base.

Parallel transfer can also be used to flexibly pass control information from higher to lower levels. This would allow, for example, initial low- and intermediate-level processing to take place with a single controller and later processing to take place with multiple controllers. 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.

5.3. Processing Levels

At the low level, there is as much potential for parallelism as there are pixels in the image. In a few instances, such as multi-sensor processing, there is an even greater potential. However, every operation does not achieve this level of parallelism. For example, extracting the convex hull of a region may only involve a few hundred points in an image.

If an application does not require maximum speed, a low-level processor can be designed to support less that the maximum potential parallelism, and thereby reduce the system's cost. A processor of this type has the advantage that few of its resources are wasted, and it has high utilization. But for

the most demanding applications, maximum potential parallelism should be supported so each operation completes in minimum time, even though processors may be unused in some operations. The remainder of this discussion assumes the latter case.

Because most sensors are arranged as square grids, a processor-per-pixel system will most naturally follow the same arrangement. The processor should support nearest neighbor communication, and the ability to quickly pass information along edges and across regions, and to permute array values in both predetermined and data dependent patterns. The low level must also support rapid calculation of global statistics, and the calculation of similar functions within regions or edges in parallel.

Control of the low level involves far fewer instruction streams than pixels. However, it must be possible to select any subset of the processors to carry out an operation. It must also be possible for the intermediate- and high-level processors to transmit parameters to the low level to support top-down local control of array-wide operations. Of course, the low level must be able to transmit its results in parallel to the intermediate level. Similarly, it should support parallel input of image data from a staging memory, to minimize I/O overhead.

The intermediate level does not necessarily benefit the fixed processor arrangement that the low level does, although it is useful to associate each intermediate-level processor with a fixed group of low-level processors so that spatial relationships between certain structures in the two levels can be preserved. There is not as much potential data parallelism at the intermediate level because the data is more abstract , and each datum carries more information. However, several thousand tokens may be processed simultaneously by one grouping or matching process. A second approach is to exploit the control parallelism by running tens to hundreds of these processes simultaneously, perhaps with subsidiary data parallelism.

Communication at the intermediate level must allow token grouping and access by knowledge sources. Knowledge source access depends upon whether token structures are shared, or copied and distributed. Some systems may use both access approaches, and thus depend upon both shared memory and message passing capabilities. As noted earlier, some operations also require global summary information to be computed in a timely manner.

The intermediate level must support both central control (which is used when interacting with the low level) and distributed control (for working under the direction of the high level). There is thus the need for mechanisms to globally synchronize the processors, and to let them run independently. Also, it must be possible for the high level to issue commands to the intermediate level to execute knowledge sources.

The high-level processors can be general purpose in nature, as long as they support multiprocessing with shared data access in some manner. However, it may be desirable to augment the processors with hardware to support evaluation of rules, inference, logic programming, and list processing.

Communication in the high level requires access to various shared structures, such as the token databases, different levels of the blackboard, and the knowledge base. In simpler systems, this access can be supported by a common shared memory. However, in larger applications it may be necessary to partition the memory and may even be useful to augment it with processing capabilities..

The high level must support tens to hundreds of object (and object part) schemas running in parallel. However, perhaps only a hundred or so schemas might be executing at any one time in a complex system (the others will be waiting for results from knowledge sources or other schemas). Thus, the high level must support low-overhead dynamic scheduling (under real-time constraints in many cases).

As can be seen from the preceding discussion, a vision architecture requires a complex and unique combination of computational power, communication, and control. In the next section, we briefly discuss a specific architecture designed with these requirements in mind.

6. OVERVIEW OF THE IMAGE UNDERSTANDING ARCHITECTURE (IUA)

The Image Understanding Architecture consists of three different, tightly coupled parallel processors. These are the Content Addressable 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 1). The CAAPP and ICAP levels are controlled by a dedicated Array Control Unit (ACU) that takes its directions from the SPA level. In each layer of the IUA the processing elements are tuned to the computational granularity and algorithms required by that particular level of abstraction. A 1/64th slice of the IUA is currently being built by the University of Massachusetts and Hughes Research Laboratories as a proof-of-concept demonstration. The brief discussion that follows describes the full IUA, except where it is specifically noted that a feature pertains only to the prototype. More information on the design of the IUA can be found in [Weems, 1989], [Weems, 1990a], [Weems, 1990b], [Rana, 1988], and [Rana, 1990]. Discussions of software support for the IUA can be found in [Weems, 1990c], [Herbordt, 1990], and [Scudder, 1990].

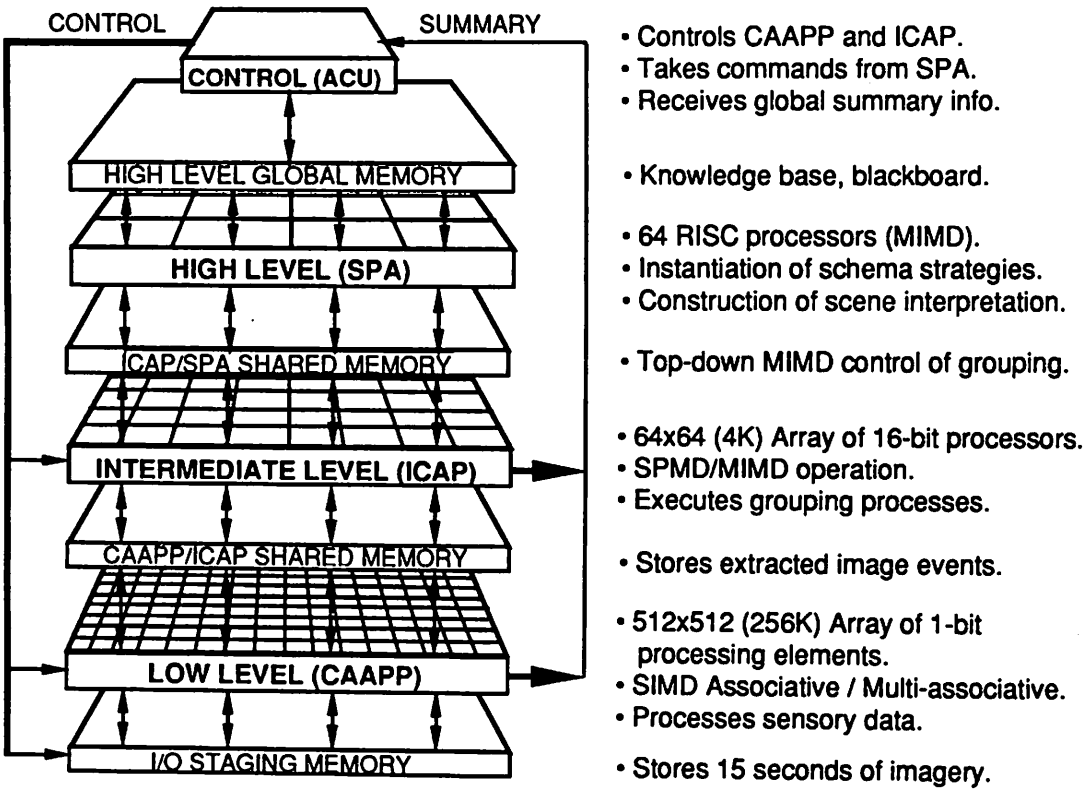

Figure 1. IUA Overview

At the high level, the IUA is purely a MIMD parallel processor, while the intermediate and low levels of the IUA may be treated in a variety of modes of parallelism. The CAAPP operates in SIMD associative or multi-associative mode, and the ICAP operates in synchronous-MIMD or pure MIMD

mode. The multi-associative and synchronous-MIMD modes are explained in the following sections, but will be briefly introduced here. In multi-associative mode, the CAAPP cells execute the same instruction stream but in disjoint groups, with each group able to operate on locally broadcast values, and to locally compute its own summary values in parallel with all other groups. This allows different parameters to be employed in processing disjoint portions of the image, with all portions of an image being processed simultaneously. In synchronous-MIMD mode, the ICAP processors execute the same program but have their own instruction pointers so that they can branch independently, and globally synchronize for each stage of processing. Synchronous-MIMD has the advantage of being as simple to program as a SIMD system but without the time penalty, usually encountered in SIMD systems, of having to sequentially execute all of the paths in a branching control structure.

6.1. The CAAPP (low) Level

The CAAPP is an image-sized array of custom 1-bit serial processors, and is similar in many ways to CLIP-4 [Duff, 1978], MPP [Batcher, 1980], DAP [Hunt, 1981], GRID [Arvind, 1983], GAPP [Davis, 1984], and the Connection Machine [Hillis, 1986]. However, its architecture is especially oriented towards associative processing with an emphasis on fast global summary feedback mechanisms supported in hardware. The CAAPP is also specifically designed to interact with the ICAP in a tightly coupled fashion.

The CAAPP processing elements are linked through a four way mesh which is augmented with circuitry that allows certain types of long distance communication to take place quickly. Each processor can execute an instruction in 100 nanoseconds and contains 5 one-bit registers, an ALU, data routing circuitry, and 320 bits of RAM that acts as an explicitly managed cache memory. Each element has access to a 32K-bit backing store memory that is dual-ported with the ICAP. The backing store is also referred to as the CAAPP-ICAP shared memory (CISM). The CAAPP also provides a special I/O staging memory that can buffer up to 15 seconds of imagery at 30 frames per second. Any frame in the staging memory can be transferred into the CAAPP processor array in 7.2 microseconds. The CAAPP can also store an image into the staging memory in a similar amount of time.

A key to integrating the CAAPP into the IUA is its fast global summary mechanisms. The array-wide logical OR output, called Some/None, indicates whether any CAAPP cells are in a given state. The counting operation reports the number of cells in a given state, and is used to gather statistics about an image and the results of processing. Some/None takes one instruction cycle to report, and counting takes 18 cycles.

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 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 remaining communication mechanism is described in the next section.

6.2. The Coterie Network

The fourth means of communication is via multi-associative broadcast and response, which is supported by a variation on the mesh called the Coterie network. This is similar to the flash-through mode of Illiac-3 [McCormick, 1963], the reconfigurable buses proposed by Kumar [Kumar, 1987],

Miller and Stout [Miller, 1987] and the polymorphic torus proposed by Li [Li, 1987], but differs in that it allows general reconfiguration of the mesh, and multiple processors to write to the mesh at the same time. By adding the simple switch network shown in Figure 2, it is possible, under program control, to create independent groups of processors that share a local bus. The isolated groups of processors can then execute globally broadcast instructions using locally broadcast data, or local summary information, which permits SIMD 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 attribute for each of the regions. In a typical SIMD architecture this would be done by sequentially selecting each region or in parallel via a propagating wave that checks region labels at each step. However, using the Coterie network the regions can be isolated from each other and perform their own local evaluation in parallel. Note that the Coterie Network is separate from the nearest neighbor mesh, which we refer to as the SEWN Mesh.

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 those of their neighbors, and close the switches that connect them to neighbors with similar properties. It should be fairly obvious 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 in a single cycle, it is possible to quickly reconfigure the Coterie network from one interconnection pattern to another.

Within a coterie, there is a bit-wide bus that all of the processors are connected to. Each active CAAPP processor may be instructed to output a bit onto the bus or read whatever value is currently on the bus. When more than one processor writes to the bus, the result is the logical OR of the values on the bus. The bus is thus functionally equivalent to the global Some/None circuit except that its output is locally formed and only available within a coterie. If only one processor is selected for writing, the Coterie network broadcasts its value to every member of the coterie. The local summary and broadcast processes can occur in every coterie in parallel. The Coterie network also permits parallel-prefix algorithms to be performed within coteries, thereby enabling operations such as enumeration, counting, and summing to take place locally.

6.3. Communication Between the CAAPP and ICAP

The principal mechanism for transferring data between the CAAPP and ICAP is the CISM (or backing store). Each ICAP processor has access to the 256K-byte block of memory that also acts as the 32K-bit backing store for each of the 64 CAAPP cells associated with an ICAP processor. Swapping between the CAAPP and CISM is accomplished by dual-porting a portion of the on-chip CAAPP memory. 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.

6.4 The ICAP

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

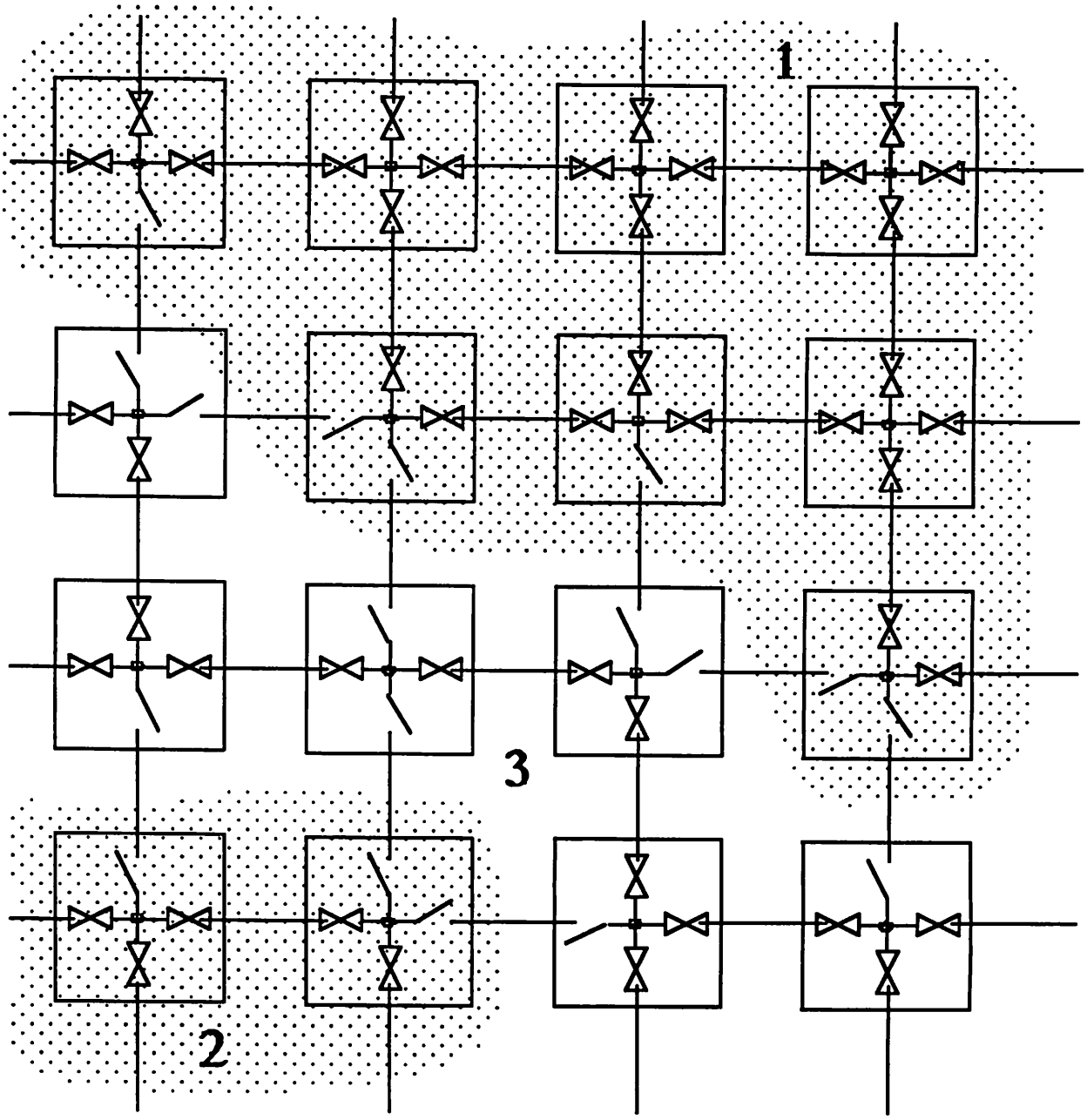

Figure 2. The Coterie Network

Control of the ICAP is provided by the ACU (in Synchronous-MIMD mode) and by the SPA (in MIMD mode). Once sensory events have been extracted and represented symbolically in the ICAP, the SPA processors may then direct the ICAP processors to execute procedures that serve as knowledge sources. The ICAP provides three different global OR outputs available to the controller that can be used to determine the status of processing in the ICAP array. The choice of meaning for each signal is left up to the programmer. For example, they can be used as an associative Some/None mechanism, or to indicate exceptions in processing. A global summation mechanism is

provided for gathering statistical information, and forms a sum of an 8-bit value from each ICAP processor.

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. In the IUA prototype, which has only 64 ICAP processors, the bit-serial I/O links between the processors are connected through a centrally controlled 64 X 64 bit-serial crossbar switch. Thus it is possible to establish any point-to-point network topology in the prototype ICAP. Various methods of extending the ICAP communications network to the full-size ICAP array are under consideration.

6.5. Communication Between the ICAP and SPA

The ICAP-SPA Shared Memory (ISSM) provides the principal communication path between the top two levels of the IUA. It is viewed as an I/O device by each ICAP processor. 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.

6.6. The SPA

The SPA processors operate in MIMD mode with communication through a shared blackboard. 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 workstation class processor, augmented with a symbolic co-processor. A separate research investigation within the UMass VISIONS project is currently exploring the implementation of high-level algorithms and data structures using a commercially available shared-memory multiprocessor [Draper, 1988]. 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. We are also considering an augmented shared memory, such as a multi-port associative (or content-addressable) memory.

6.7. The Array Control Unit

The ACU contains two separate processors that can issue instructions to the CAAPP (and control the ICAP as described below). The two processors are referred to as the Macro-controller and the Micro-controller.

The Macro-controller is a standard RISC processor 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, it can only issue instructions at about one-quarter of the rate that the CAAPP can execute them. The second method is to issue subroutine calls to the Micro-controller.

The Micro-controller is a custom processor, driven by horizontal microcode and capable of issuing an instruction to the CAAPP every 100 nanoseconds. The Micro-controller will have a large library of CAAPP routines in its program memory, any of which can be called by the Macro-controller. The

dual processor arrangement 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 the only source of instructions for the CAAPP is the ACU, the ICAP processors each have their own program memory. The ICAP program memory is loaded with a large library of service routines upon system initialization. The ACU issues instructions to the ICAP 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. An ICAP user program is typically just an execution script of calls to the ICAP library. Thus, the ACU and ICAP interact very little when a program is running in the ICAP; the exception is when the ICAP program reaches a global synchronization point -- this must be mediated by the ACU. The ACU can also set the ICAP to operate in MIMD mode, by turning control over to a task queuing program in the ICAP processors. The queuing 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 between ICAP processors.

The ACU thus supports 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.

6.8. Current Implementation Status

Low level (CAAPP) custom VLSI processing element chips, containing 64 processors, have been fabricated and tested. The intermediate level parallel communication switch (PARCOS), a 32 x 32 bit-serial crossbar with a connection pattern cache, has been fabricated in custom VLSI and tested. A custom feedback concentrator chip, which supports Some/None, Some/All, and Count, has also been built and tested. The 1/64th scale demonstration prototype of the IUA is being built at Hughes Research Labs and is scheduled for completion in mid-1990 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.

6.9. Further Development

Several refinements are being considered for future versions of the machine, in particular to the ICAP communication structure and the SPA. One extension is to augment the ICAP circuit switched network that is centrally controlled with a distributed control routing network. We are also exploring various means of increasing the circuit density in order to reduce the physical size of the machine. For example, a 256 processor CAAPP chip is already being designed and a 1024 PE chip is envisioned in a few years. We are also looking at substrate technology that will permit several low- and intermediate-level processor chips and their memory to be integrated into a two by four inch carrier.

7. CONCLUSIONS

Knowledge-based machine vision is both a complex and computationally intensive problem. As such, it presents a unique set of challenges to the computer architect. In order to supply the requisite processing power, it is necessary to first analyze the data, communication, and control structures used in vision systems, in order to capitalize on their inherent parallelism. Those structures can then be translated into architectural requirements for computation, communication, and control. Once this

is done, it becomes possible to outline a machine organization to which traditional methods of architectural analysis and tuning can be applied.

Vision is characterized by three distinct levels of abstraction, each of which utilizes significantly different representations, communication structures, and algorithms. The computational requirements of each level are so different from the other two that no single, homogeneous parallel processor can efficiently support processing at all three levels. Instead, we propose that a heterogeneous processor, consisting of three levels that correspond to the levels of abstraction in vision, will be best suited for real-time vision. Furthermore, each level must be designed from the start to be integrated with the complete system, in order to avoid bottlenecks in communication between levels.

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.

8. ACKNOWLEDGEMENTS

This work was funded in part by the Defense Advanced Research Projects Agency under contract numbers DACA76-86-C-0015, and DACA76-89-C-0016, monitored by the U.S. Army Engineer Topographic Laboratory; and contract number F49620-86-C-0041, monitored by the Air Force Office of Scientific Research; and by a Coordinated Experimental Research grant (DCA 8500322) from the National Science Foundation. The author would like to thank Edward M. Riseman and Allen R. Hanson for their helpful comments in preparation of the manuscript, and David B. Shu and J. Gregory Nash at Hughes Research Laboratories for their contributions to the IUA design and development.

9. REFERENCES

[Arbib, 1987], Arbib, M.A., Hanson, A.R. (eds.), Vision, Brain, and Cooperative Computation, MIT Press, Cambridge, MA, 1987.

[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.

[Ballard, 1982] Ballard, D.H., Brown, C.M., Computer Vision, Prentice Hall Inc., Englewood Cliffs, NJ, 1982.

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

[Boldt, 1989] Boldt, M., Weiss, R., Riseman, E., Token-Based Extraction of Straight Lines, IEEE Trans. Sys. Man, and Cybernetics, V.19, No. 6, Nov-Dec, 1989, pp. 1581 - 1594

[Brolio, 1989] Brolio, J., Draper, B.A., Beveridge, J.R., Hanson, A.R., ISR: A Database for Symbolic Processing in Computer Vision, IEEE Computer, Vol. 22, No. 12, December, 1989, pp. 22-30.

[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.

[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, 1987] 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, 1989] Draper, B.A., Collins, R.T., Brolio, J., Griffith, J., Hanson, A.R., Riseman, E.M., The Schema System: Knowledge Based Vision, International Journal of Computer Vision, Vol. 2, Number 3, 1989.

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

[Duff, 1986] Duff, M.J.B. (ed.), Intermediate-Level Image Processing, Academic Press, London, 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.

[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. (Eds.), Computer Vision Systems, New York: Academic Press, 1978.

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

[Hanson, 1987] Hanson, A.R., Riseman, E.M., From Image Measures to Object Hypotheses, Submitted to IEEE PAMI.

[Herbordt, 1990] Herbordt, M.T., Weems, C.C., Message Passing Algorithms for a SIMD Torus with Coteries, Proc. ACM Intl. Symposium on Parallel Algorithms and Architectures, Crete, July, 1990.

[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.

[Kanade, 1987] Kanade, T. (ed.), Three Dimensional Machine Vision, Kluwer Academic Publishers, Norwell, MA, 1987.

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

[Levialdi, 1988] Levialdi, S. (ed.), Multicomputer Vision, Academic Press, San Diego, CA, 1988.

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

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

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

[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 Intelligence, 4:263-277, 1982.

[Nevatia, 1982] Nevatia, R., Machine Perception, Prentice Hall, Englewood Cliffs, NJ, 1982.

[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.

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

[Rana, 1990] Rana, D., Weems, C.C., A Feedback Concentrator for the Image Understanding Architecture, Proc. 1990 IEEE Intl. Conf. on Pattern Recognition, Atlantic City, NJ, June 1990.

[Scudder, 1990] Scudder, M., Weems, C.C., An Apply Compiler for the CAAPP, COINS TR #90-60, University of Massachusetts, 1990.

[Uhr, 1987] Uhr, L.M., Evaluation of Multicomputers for Image Processing, Academic Press, Orlando, FL, 1986.

[Weems, 1988] Weems, C.C., Hanson, A.R., Riseman, E.M., Rosenfeld, A., An Integrated Image Understanding Benchmark: Recognition of a 2-1/2D "Mobile", Proc. IEEE Conf. on Computer Vision and Pattern Recognition, Ann Arbor, MI, June 5-9, 1988.

[Weems, 1989] Weems, C.C., Levitan, S.P., Hanson, A.R., Riseman, E.M., Shu, D.B., Nash, J.G., The Image Understanding Architecture, Intl. Journal of Computer Vision, 2, 251-282 (1989), Kluwer Academic Publishers, Boston, MA.

[Weems, 1990a] Weems, C.C., Rana, D., Reconfiguration in the Low and Intermediate Levels of the Image Understanding Architecture, in Reconfigurable SIMD Parallel Processors, Hungwen Li (ed.), Prentice Hall, Englewood Cliffs, N.J., 1990. Also, COINS TR# 90-10.

[Weems, 1990b] Weems, C.C., Rana, D, Shu, D.B., Nash, J.G., A Progress Report on the Development of the Image Understanding Architecture, Proc. IEEE Intl. Conf. on Pattern Recognition, Atlantic City, NJ, June, 1990.

[Weems, 1990c] Weems C.C., Burrill, J.R., The Image Understanding Architecture and its Programming Environment, in Parallel Architectures and Algorithms for Image Understanding, V.K. Prassana-Kumar (ed.), Academic Press, Orlando, FL, 1990.

[Weems, 1990d] Weems, C.C., Riseman, E.M., Hanson, A.R., Rosenfeld, A., An Integrated Image Understanding Benchmark for Parallel Processors, COINS TR# 90-##, University of Massachusetts at Amherst, also submitted to the Journal of Parallel and Distributed Computing.

[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.