Parallel Processing in the DARPA Strategic Computing Vision Program

Charles C. Weems, University of Massachusetts at Amherst
Christopher Brown, University of Rochester
Jon A. Webb, Carnegie Mellon University
Tomaso Poggio, Massachusetts Institute of Technology
John R. Kender, Columbia University

TO BRING THE POWER OF PARALLEL PROCESSING TO VISION SYSTEMS, RESEARCHERS IN THE STRATEGIC COMPUTING PROGRAM DEVELOPED NEW HARDWARE, SOFTWARE TOOLS, ALGORITHMS, AND PERFORMANCE METRICS. WHAT ARE THE RESULTS OF THIS EFFORT? A BETTER UNDERSTANDING OF BOTH PARALLEL ARCHITECTURE AND MACHINE VISION.

THE COMPUTATIONAL DEMANDS of image understanding are among the most extreme of any computer application. A medium-resolution color image contains 750,000 bytes, and images are typically acquired 30 times a second, giving an input data rate of 23.6 megabytes per second. Many operations must be performed on each pixel to extract the simplest features from an image; yet this is only the beginning of the interpretation process.

A typical goal of an image-understanding system is to construct a three-dimensional model of the image sensor's environment. Such an interpretation might require identifying hundreds of different objects. Other goals might include determining the sensor's position with respect to landmarks, identifying independently moving objects, or inspecting objects. Complicating factors include noise, occlusion, uneven lighting, shadows, specular reflections, and motion effects.

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 the window's surroundings and our knowledge of the properties of glass. Even if the window were perfectly transparent, we could infer the existence of a pane of glass from the presence of the window frame. In fact, much of what we "see" in natural scenes is really inferred from partial information.

Fortunately, the tremendous computational demands of vision are balanced by the potential for applying parallel processing. For example, vision clearly involves both sensory (low-level) and knowledge-based (high-level) processing. Between these two levels of abstraction it is useful to introduce one or more levels of symbolic (intermediate-level) 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.

Each of these abstraction levels can be processed in parallel with the other levels. Furthermore, within each level there is a great deal of potential parallelism. Low-level vision is amenable to single-instruction multiple-data stream, data parallel, or pipelined parallel computation. (The sidebar on page 37 provides definitions for these and other terms found below.) Intermediate-level vision can involve both single-instruction multiple-data and multiple-instruction multiple-data stream parallelism — the former to process the thousands of extracted image events and abstract symbols that are common in an interpretation, and the latter to permit independent, simultaneous operations on collections of symbolic data. High-level processing can involve multiple cooperating and competing interpretation strategies, access to a large knowledge base, transformations and projections of models, and so on.

Another approach would be to use multiple visual modules running in parallel to analyze a scene. Texture, color, motion, depth, and object boundary analysis can all proceed simultaneously. Further, the vision problem is simplified if the seeing agent has active control over its sensors. Thus, large-grain parallelism is also appropriate for simultaneously planning actions, updating asynchronous databases, monitoring physical progress, and allocating sensors.

Because of the size and complexity of vision systems, it is impractical to use a software simulation of a parallel processor to explore parallelism in vision. Conventional uniprocessors are barely sufficient for vision research in which real-time performance is not required. Interposing a layer of architectural simulation further reduces their performance to the point that research is difficult to conduct, and real-time experiments are out of the question. To bootstrap research into parallelism for vision, parallel hardware was needed.

Of course, new parallel hardware is useless without software tools. Vision research has extensive requirements for software development environments. Thus, given the hardware with its basic set of tools, additional tools to support vision research were still necessary. Once the tools were available, developing applications became feasible, and researchers began to provide feedback to the architects and toolsmiths.

However, as useful as subjective and anecdotal evidence is, architects also need empirical data on which to base design decisions. The next step was to develop mechanisms for evaluating machine performance on vision tasks. This article examines each step in this process: the hardware that was built, the tools that were developed, the applications that were implemented, and the performance evaluation that was carried out.

New parallel hardware

Toward the goal of developing advanced parallel-processing architectures, DARPA's Strategic Computing architecture program built several new and powerful parallel processors. Of these, three of the more mature approaches were considered germane to vision: Single-instruction multiple-data, multiple-instruction multiple-data, and systolic processing. The vision community chose three commercially built examples reflecting these approaches, respectively: the Connection Machine, the Butterfly, and the Warp. In addition to their usefulness for vision, the three types of parallelism are widely applicable to many other domains. A fourth approach, more specific to vision, was also selected for implementation as a noncommercial, experimental system. This machine, the Image-Understanding Architecture, involves a heterogeneous combination of parallel processors with single-instruction multiple-data, multiple-instruction multiple-data, and other capabilities.

CONVENTIONAL UNIPROCESSORS ARE BARELY SUFFICIENT FOR VISION RESEARCH IN WHICH REAL-TIME PERFORMANCE IS NOT REQUIRED.

The Connection Machine. The Connection Machine[1] is a massively parallel single-instruction multiple-data processor, produced by Thinking Machines Corporation, that has been successfully applied to many problems in low- and intermediate-level vision. Several different programming models have emerged, since the machine provides multiple communication modes.

The Connection Machine model CM-2 can be configured with between 16,384 and 65,536 processors operating under a single instruction stream. Each processor has a 1-bit arithmetic logic unit with 64 Kbytes of memory (optionally with a floating-point arithmetic accelerator shared among 32 processors). The processors have two modes of communication: the North, East, West, South network and the router. The NEWS network provides communication between neighboring processors in a rectangular grid of arbitrary dimension. The router sends messages between processors in the machine via a hypercube connection network. The Connection Machine also has facilities for returning to the host machine the results of various operations on a field in all the processors; it can return the field's global maximum, minimum, sum, logical AND, and logical OR.

To manipulate data structures with more elements than the number of processors in the array, the Connection Machine supports virtual processors. A single physical processor can operate as a set of virtual processors by serializing operations in time and partitioning each processor's memory. This is otherwise invisible to the user.

Figure 1 shows how the Connection Machine is used at MIT, where it and a movable two-camera eye-head system comprise a laboratory environment called the MIT Vision Machine. Columbia University researchers have also used the Connection Machine to develop new software tools and applications.

The Warp. Another computer employed in DARPA vision research is the Warp, designed at Carnegie Mellon University in prototype form. General Electric Corporation constructed the production version and distributed it to several DARPA sites, including Martin Marietta Corporation, Hughes Aircraft Corporation, Advanced Decision Systems, ESL, the University of Maryland, Analytical Sciences Corporation, and Science Applications International Corporation. The second prototype and first production machines were installed at Carnegie Mellon.

The heart of the Warp computer is a short linear array, normally consisting of 10 cells (although up to 20 cells have been used), each of which is a 10-Mflops computer.[2] Cell-to-cell communication has high bandwidth (40 Mbytes per second) and low latency (200 nanoseconds). The cells have local program and data memory, and can be programmed in a Pascal-level language

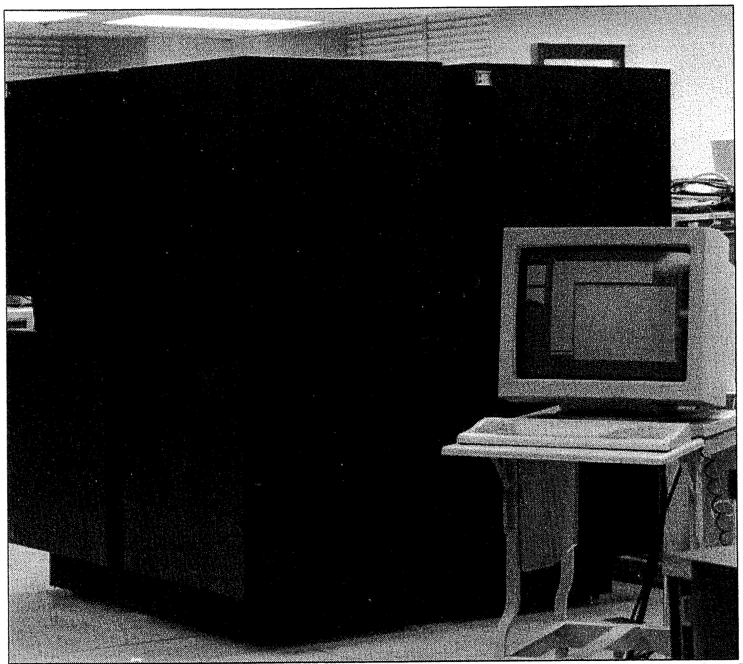
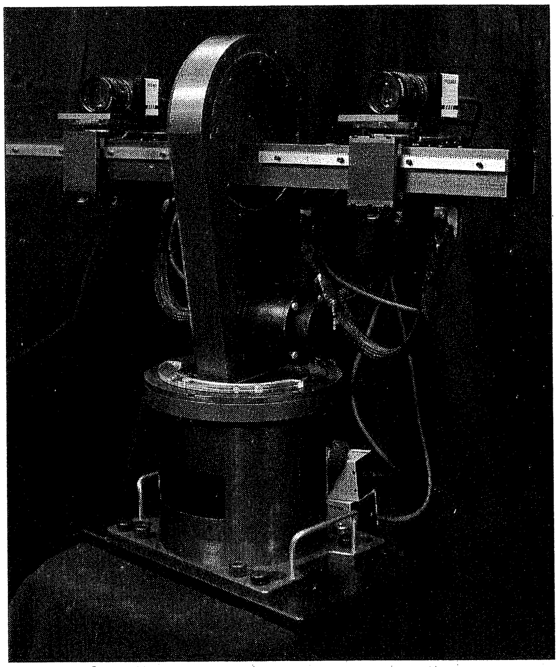



Figure 1. The MIT Vision Machine: the Connection Machine (left) and the eye-head system (right).

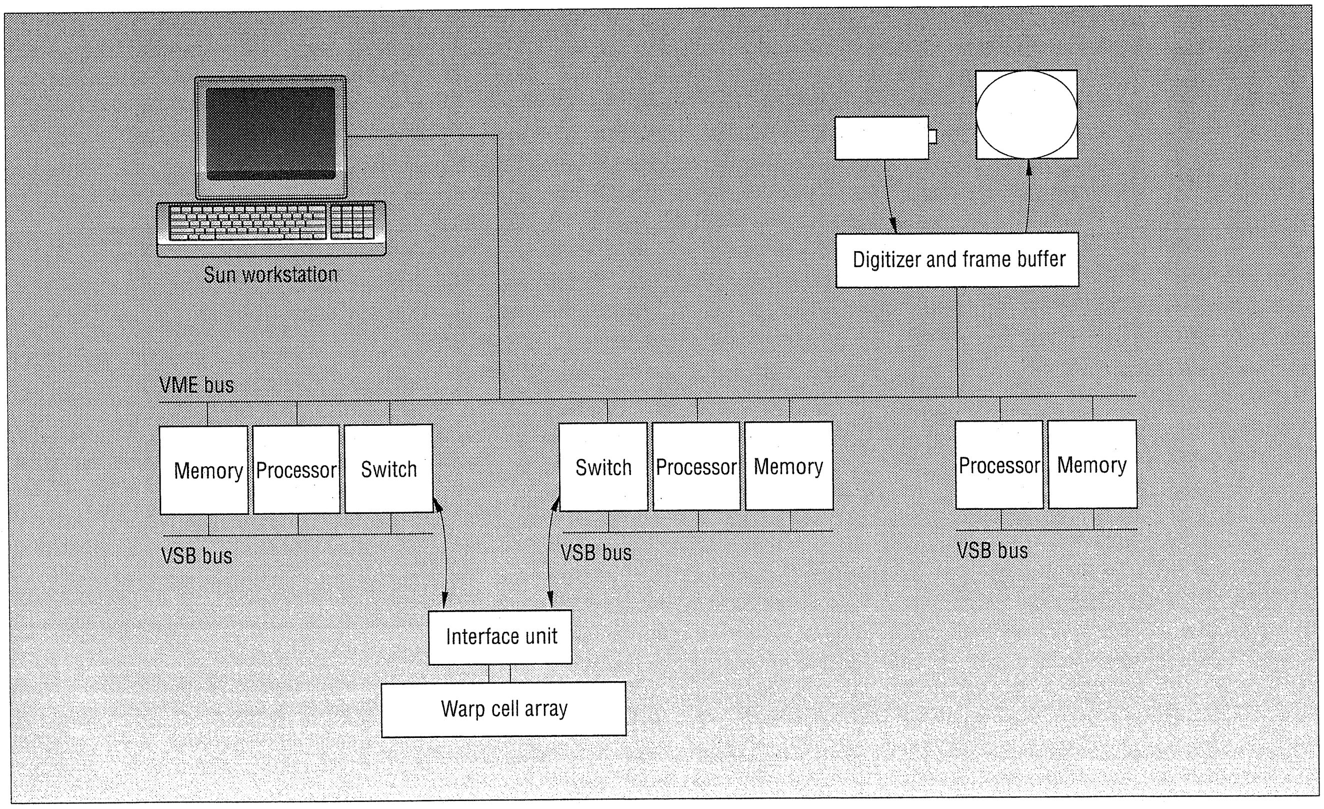


Figure 2. The Warp.

called W2, which supports communication between cells using asynchronous word-by-word send and receive statements. The systolic array is attached to an external host, which communicates with the array via a separate memory and an interface unit (see Figure 2). The external host is attached to a digitizer and frame buffer, and to a workstation that provides the user interface.

One of the Carnegie Mellon Warp machines was installed on the Navlab robot

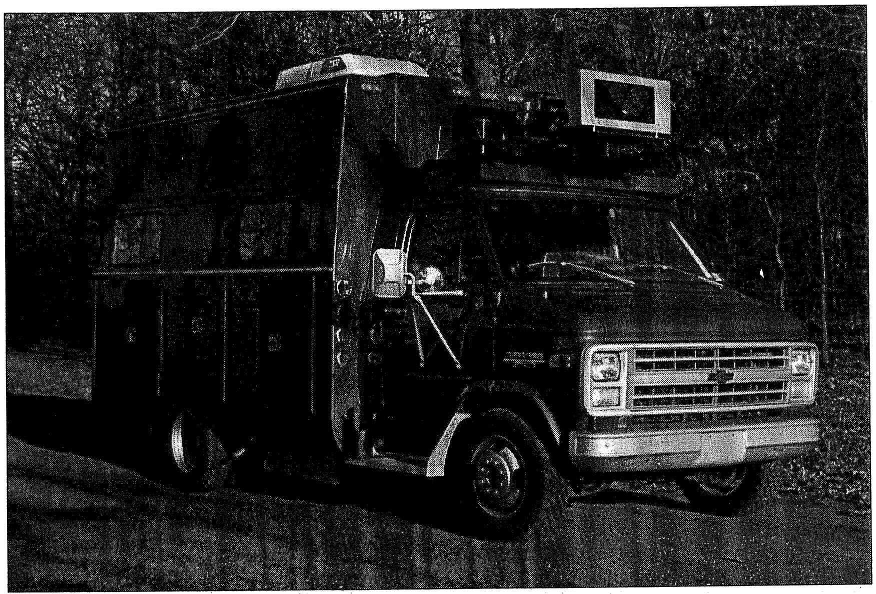


Figure 3. The Navlab.

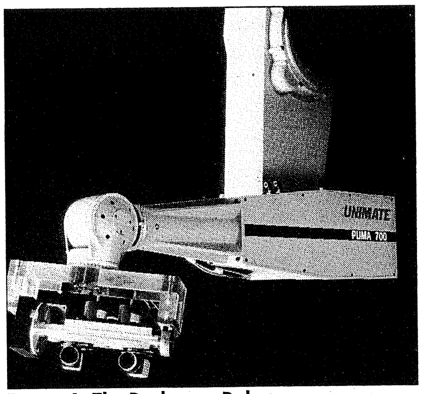


Figure 4. The Rochester Robot.

vehicle, a custom-built, fully instrumented testbed for robot navigation, based on a standard van chassis (see Figure 3). The Navlab is equipped with cameras, a laser range finder, a global positioning system receiver, and an inertial navigation system. The Navlab carries all of its computational equipment on board, making it a fully autonomous robot vehicle.[3]

The Butterfly and Max-Video. The third DARPA-based commercial system that affected vision research is the Butterfly Plus parallel processor, developed by Bolt Beranek and Newman/Advanced Computers Incorporated. The latest version has 128 nodes, each consisting of an MC68020 processor, an MC68851 memory management unit, an MC68881 floating-point unit, and 4 Mbytes of memory. The Butterfly is a shared-memory multiprocessor with nonuniform memory access times; each processor can directly access any memory in the system through a network (called an omega network or butterfly network) of 4×4 crossbar switches. The network assures that one processor can access another's memory in a "reasonable" time — approximately 4 log n times slower than it can access its own memory for an n-processor computer.

At the University of Rochester, the Butterfly has a VME bus connection that mounts in the same card cage as a Max-Video system and motor controller boards for a binocular robot head (see Figure 4). The Max-Video system is a commercially built pipelined parallel image-analysis system. It allows an arbitrary number of independent boards to be cabled together to achieve many frame-rate image-analysis capabilities, including digitization and storage, 8×8 or larger convolution, arbitrary signal-processing computations, pixelwise image processing, crossbar image-pipeline switching for dynamic reconfiguration of the image pipeline, lookup tables, histogramming, and feature location. The Butterfly can control Max-Video boards through programmable registers.

The Image-Understanding Architecture. The University of Massachusetts and Hughes Research Laboratories developed this noncommercial, experimental system for the Strategic Computing program. The IUA represents a hardware implementation of the low, intermediate, and high levels of representation and processing that are encountered in many knowledge-based vision systems.[4] It consists of three different, tightly coupled parallel processors. The low- and intermediate-levels are controlled by a dedicated array control unit, which takes its directions from the high level. As Figure 5 indicates, each of these parallel processors provides different granularities and modes of parallelism. The low level is a fine-grained, processor-per-pixel array of bit-serial processing elements. The intermediate level is a medium-grained multicomputer consisting of 16-bit processors, and operates in both multiple-instruction multiple-data and single-program multiple-data modes. The high level is a coarse-grained multiprocessor oriented toward manipulating models, inferencing, running Lisp, and so on.

As a proof-of-concept demonstration, IUA researchers have built and are testing a 1/64th-scale vertical slice of the system. However, a detailed software simulator has been in use since 1987 and has allowed researchers to construct software before completing the hardware and to evaluate many design aspects. A second generation of the IUA is under development that will incorporate straightforward but significant enhancements, and a third generation is already being planned.

New software tools

To use parallel-processing hardware, vision researchers must have effective software tools beyond the basic tool set of operating systems, languages, and debuggers. For example, researchers need

- interfaces for nonstandard image acquisition and display,
- controllers that direct robots to point and focus cameras,
- libraries of basic image-processing and vision operations, and
- tools for examining and analyzing imagery and processing results.

Each site employing a parallel architecture developed a different set of tools. This was partly due to differences in the architectures themselves, the particular applications in which machines were being used, and the differences between researchers' approaches at each site. The result is a diverse collection of tools, leading to significant cross-fertilization of ideas between the sites.

The MIT Vision Machine. As mentioned earlier, the Connection Machine at MIT is closely integrated with a custom eye and head sensory platform on a robot arm. MIT researchers developed interface software for this specialized hardware that precisely controls the viewpoint and camera parameters (for example, focus) for laboratory experimentation. Images from the cameras are digitized and sent to the Connection Machine for processing. Working with researchers at Thinking Machines, MIT also developed an extensive set of vision subroutines in *Lisp (a data-parallel version of Lisp) for the Connection Machine as well as tools for examining system output.

The overall organization of the MIT Vision Machine is shown in Figure 6. Independent modules corresponding to different visual cues have been developed to process images in parallel: they extract edges, compute disparity from stereo images, estimate optical flow due to motion, compute true texture attributes such as density and orientation of texels, and estimate the spectral albedo of surfaces.

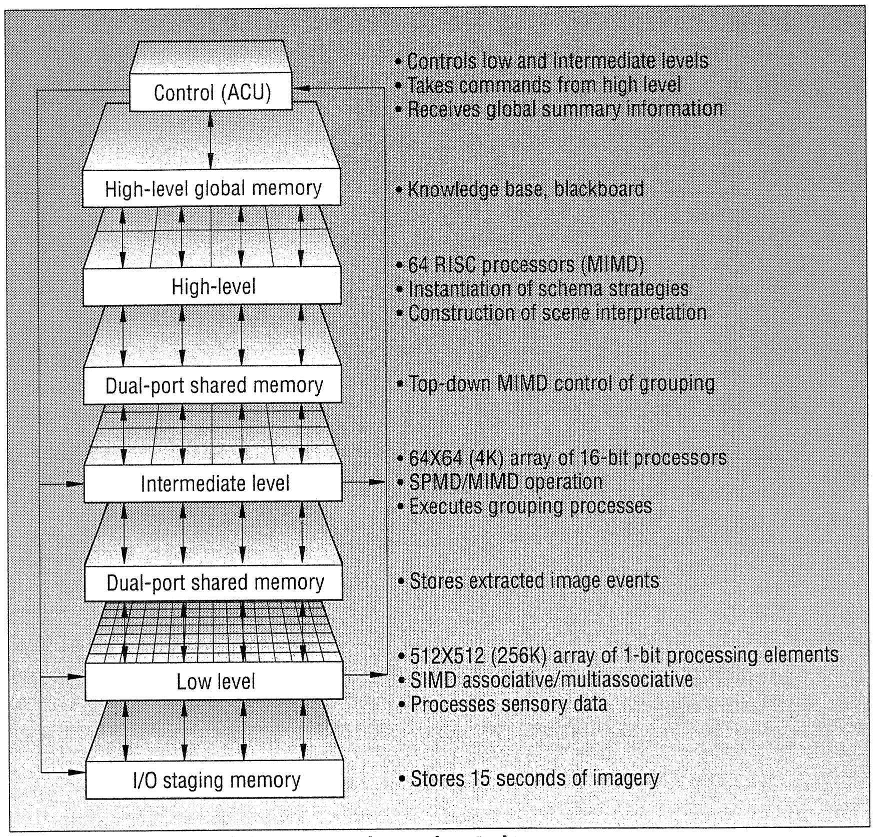

Control (ACU)
• Controls low and intermediate levels
• Takes commands from high level
• Receives global summary information
High-level global memory
• Knowledge base, blackboard
High-level
• 64 RISC processors (MIMD)
• Instantiation of schema strategies
• Construction of scene interpretation
Dual-port shared memory
• Top-down MIMD control of grouping
Intermediate level
• 64X64 (4K) array of 16-bit processors
• SPMD/MIMD operation
• Executes grouping processes
Dual-port shared memory
• Stores extracted image events
Low level
• 512X512 (256K) array of 1-bit processing elements
• SIMD associative/multiassociative
• Processes sensory data
I/O staging memory
• Stores 15 seconds of imagery

Figure 5. An overview of the Image-Understanding Architecture.

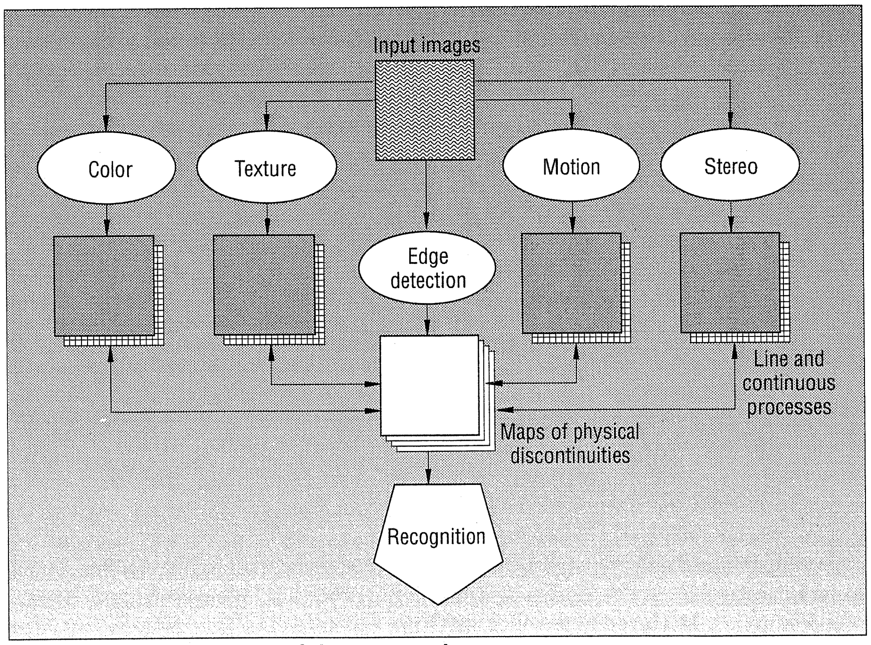

Input images
Color
Texture
Motion
Stereo
Edge detection
Line and continuous processes
Maps of physical discontinuities
Recognition

Figure 6. Overall organization of the Vision Machine.

A parallel-vision environment. Columbia University has developed several programming environments that support research on stereo and texture algorithms in a parallel image-pyramid style.[5] The first environment, consisting of simulators of various grain sizes, supported multiresolution algorithms for the Columbia Non-Von supercomputer (since discontinued). The foundation for the final programming environment is a highly efficient pyramid machine emulator that executes on a Connection Machine.

The programming environment reduces communication contention by optimally mapping the pyramid onto the hypercube network, assisted by pipelining. Mesh operations take only a small, fixed amount of overhead proportional to the hypercube's size; parent-child operations run in a smaller fixed time independent of hypercube size.

Image functions let the user create pyramid data structures, load and unload various pyramid levels, move data up and down, and perform standard low-level and intermediate-level vision operations on data structures, such as convolution and other multiresolution operations. Implemented and tested parallel-vision algorithms include pyramid-based edge extraction, binary search correlation, hierarchical stereo matching, multiresolution gray-level statistical texture segmentation, generalized Hough algorithms, and a novel autocorrelation-based shape-from-texture algorithm.

```
procedure sobel (inimg : in array (–1...1, –1...1) of byte
                    border 0, mag  : out real) is
   horiz, vert : integer;
begin
   horiz := inimg (–1, –1) + 2 * inimg (–1, 0) + inimg (–1, 1)
            – inimg (1, –1) – 2 * inimg (1, 0) – inimg (1, 1);
   vert :=  inimg (–1, –1) + 2 * inimg (0, –1) + inimg (1, –1)
            – inimg (–1, 1) – 2 * inimg (0, 1) – inimg (1, 1);
   mag := sqrt (horiz * horiz + vert * vert);
end sobel;
```

Figure 7. An Apply implementation of Sobel edge detection.

The Warp and I-Warp programming model. A critical step in developing a new parallel machine is developing programming models that help define and restrict the programmer's view. Warp's developers tried both systolic and data-partitioned models. In systolic models they divided the algorithm among processors; for example, convolution is broken down into a series of add/multiply steps, done on a pipeline of cells. Warp's high I/O rate and low latency made it possible to do this efficiently with very little memory use. However, it was difficult to apply this technique uniformly to a wide class of algorithms. As a result, data-partitioned methods were used for the bulk of computer vision programming on Warp.

The Apply programming language implements the data-partitioned model for image processing in a stable, easy-to-use, architecture-independent manner.[6] An Apply program is an Ada-like procedure that represents the inner loop of an image-to-image operation.

An Apply implementation of the Sobel edge detector is shown in Figure 7. The first line defines the input, output, and constant parameters to the function. The input parameter Inimg is an input image window, a 3×3-pixel window centered about the current pixel-processing position, which is filled with the value 0 when the window lies outside the image. This same line declares the constant and output parameters to be floating-point scalar variables. The third line defines Horiz and Vert, which are internal variables used to hold the results of the horizontal and vertical Sobel edge operator. The Sobel convolution is computed in the straightforward expressions in lines 5-7.

Apply has been implemented on Warp and I-Warp (Intel's custom VLSI version)[7] as well as on a number of other machines. Apply programs are shorter than and as fast as hand-written code on Warp and Sun machines, and often faster.

To provide a stable base of software for image processing and a set of examples of image-processing programs, Carnegie Mellon developed the Web library. Web consists of about 140 routines, 80 percent of which are written in Apply, and the rest in W2. The Apply programs include almost all the local image-to-image operations. Web routines include basic image operations, convolutions, edge detectors, image gray-value operations, smoothing operations, image feature computations, binary image operators, type and coordinate conversions, color conversion, orthogonal transformations, warping, pattern generation, and multilevel image processing. Web has also been used as a source of test programs for the I-Warp simulator and hardware.

Psyche, Zebra, and Moviola. University of Rochester researchers have developed various software tools and environments to analyze and debug parallel programs.

Vision applications need many parallel operations of different sorts to run simultaneously and cooperate intimately. In analyzing parallel programs, the focus of concern is no longer simply the internal state of a single process, but must include the internal states of potentially many different processes and the interactions among those processes. We can still use a cyclic methodology, but four issues complicate analysis:

- Parallel programs often exhibit nonrepeatable behavior.
- Interactive analysis can distort a parallel program's execution.
- To analyze large-scale parallel programs, we must collect, manage, and present an enormous amount of data.
- The execution environment (exhibiting extensive parallelism) and the analysis environment (a single, comprehensive user interface) differ dramatically.

Psyche. The Psyche operating system provides both an interface for implementing multiple models as well as conventions for interaction across models.[8] Rather than providing a fixed set of parallel-programming abstractions to which programming environments must be adapted, Psyche provides an abstraction mechanism from which many different user-level abstractions can be built in user space. Through a low-level kernel interface, developers can write new pack-ages on demand and use underlying mechanisms to communicate between models.

Psyche structures memory as a collection of passive data abstractions called realms, which include both code and data. With appropriate protocols, realms can provide shared data structures, monitors, communication channels, message or I/O buffers, and a host of other abstractions for process interaction. Depending on the degree of protection desired, invoking a realm operation can be as fast as an ordinary procedure call or as safe as a remote procedure call between heavyweight processes. The kernel creates virtual processors to handle computation within a protection domain. These domains maintain the boundaries between distinct models of parallelism. Psyche has run two user applications, a five-processor robotic balloon-bouncing demonstration and a 12-processor program that uses vision and manipulation to play checkers with a person.

Zebra. Rochester also developed Zebra, an object-oriented programming interface to the Max-Video image-processing board. Zed, built on Zebra, is a menu-driven system that lets programmers create new instruction words or modify existing ones directly from the keyboard.

Moviola. The core of the Moviola toolkit[9] consists of facilities for recording execution histories, a common user interface for interactively and graphically manipulating those histories, and tools for examining and manipulating the program state during the replay of a previously recorded execution.

One form of Moviola output is a diagram

in which time flows from top to bottom. Events that occur within a process are aligned vertically, forming a time line for that process. Edges joining events in different processes reflect temporal relationships resulting from synchronization. Event placement is determined by global logical time computed from the partial order of events collected during execution. Each event is displayed as a shaded box whose height is proportional to the event's duration (see Figure 8).

Moviola's user interface provides a rich set of interactive operations to control graphical displays. A programmable interface lets the user write Lisp code to traverse the execution graph and gather detailed, application-specific performance statistics.

The IUA simulator, libraries, and Apply. The University of Massachusetts has developed a parallel programming environment[10] as part of the IUA project. The environment consists of the IUA simulator, a graphical user's console, extensions to the C and Forth languages, an Apply compiler, and a C++ compiler.

The IUA simulator executes object code for both the low- and intermediate-level processors on an instruction-by-instruction basis. It can be configured to simulate any size of IUA within the limits of the host's memory. (A full-scale IUA contains more than three gigabytes of RAM.)

The user's console provides a graphical interface to the simulator (and eventually to the hardware). The console is a window-based application that lets the user simultaneously monitor the status of all the processors in the low-level array. As shown in Figure 9, the left-hand side of the display shows various registers and memory locations in the machine as small images, and the right-hand side provides an enlargement of any one of the small images. The enlarged display lets the user directly interact with the processors in the low-level array by using the mouse. The right-hand window can also display the status of any intermediate-level processor.

The C and Forth languages have been extended to permit the user to declare variables in the low-level processors, and to manipulate them with calls to a library of subroutines, in a manner similar to the C-Paris facility on the Connection Machine. The libraries support standard arithmetic operations on both variable-precision integers and 32-bit floating-point values, and various common vision routines. The Apply compiler for the low level of the IUA produces C code with these same extensions, allowing functions written in Apply to be fully integrated with C programs.

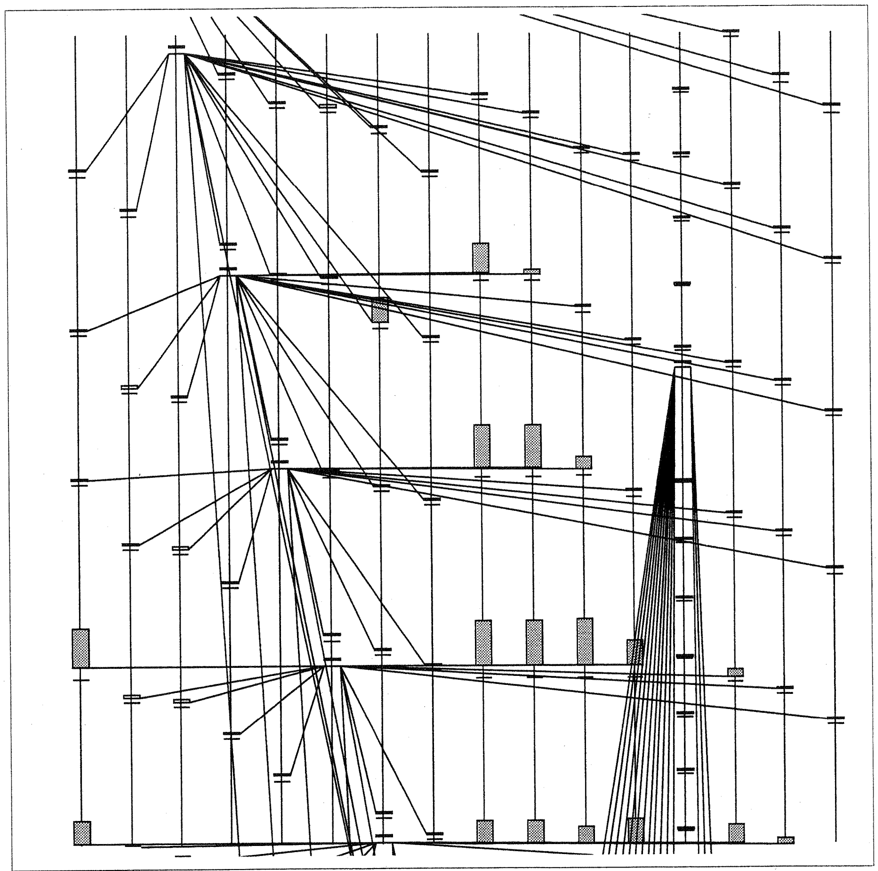
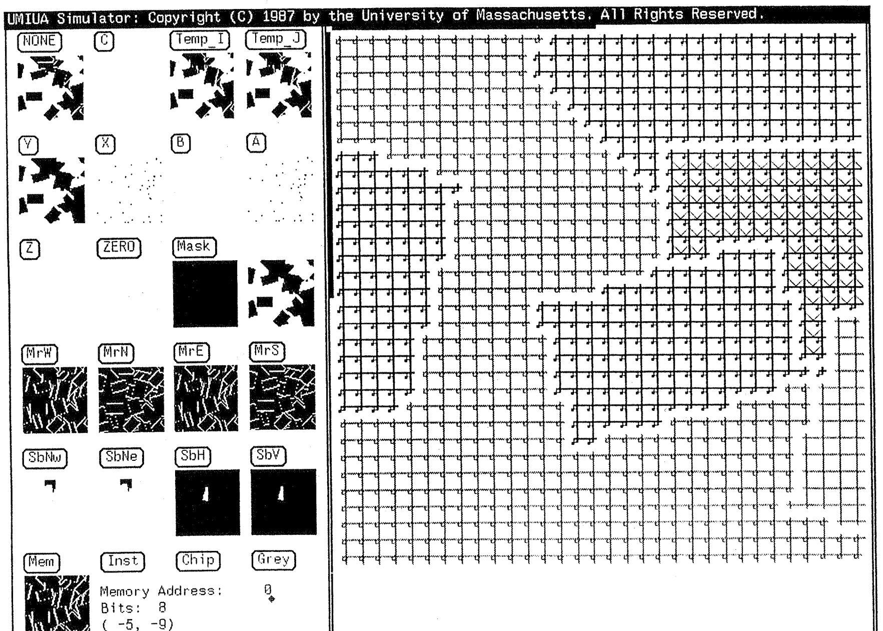
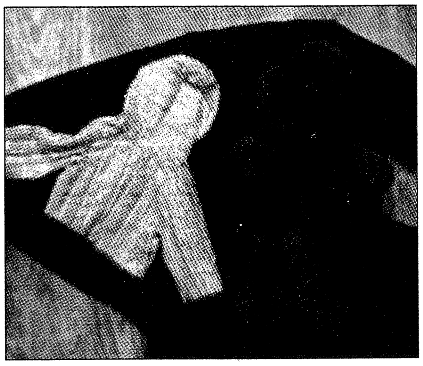


Figure 8. Rounds of an erroneous triangulation shown in physical time.


UMIUA Simulator: Copyright (C) 1987 by the University of Massachusetts. All Rights Reserved.

Memory Address: 0
Bits: 8
(-5, -9)

Figure 9. IUA simulator display.



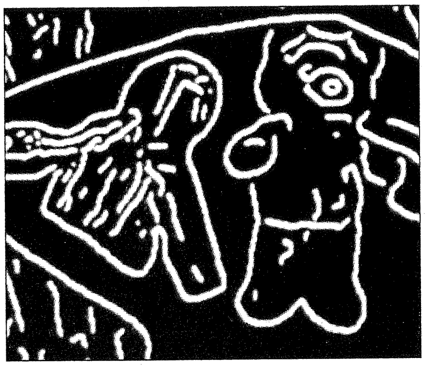
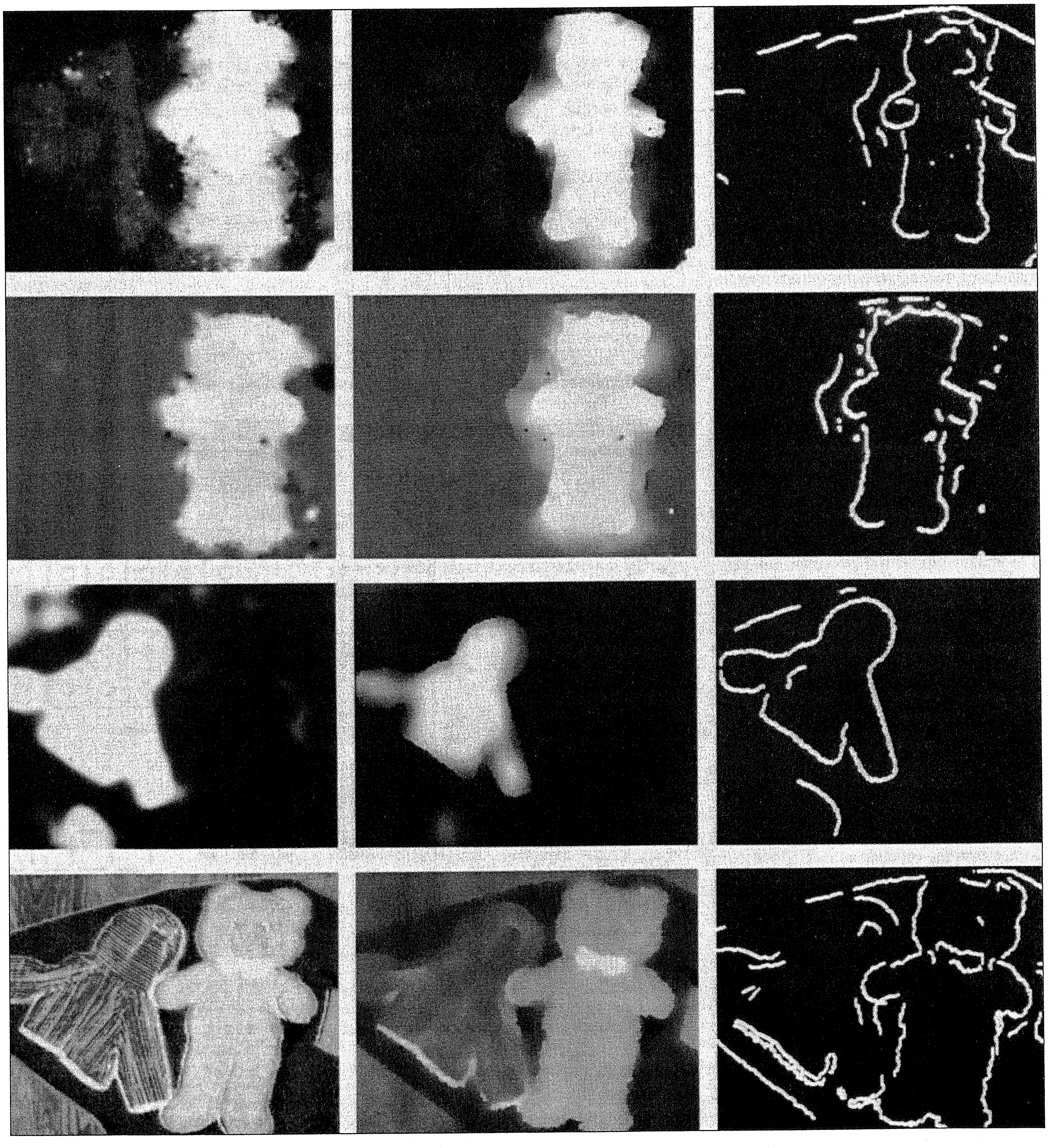


Figure 10 (far left). One frame from the motion sequence input to the MIT Vision Machine.

Figure 11 (left). Brightness edges computed by the Canny algorithm.

Figure 12 (below). Features extracted and discontinuities detected.



The Web library has been successfully compiled with the IUA Apply compiler and is also available for use. The language of choice for programming the low level of the IUA is a set of standard C++ class libraries that have been developed to support an image-plane data type and operators on that type. Programs written on sequential machines with these class libraries can then be recompiled for execution on the IUA merely by selecting a different runtime library.

New algorithms

Research under the Strategic Computing vision program involves experimentation in the laboratory and in the context of real applications. Laboratory experimentation is necessary so that the many factors affecting the performance of a new vision algorithm can be carefully controlled, thereby permitting the algorithm to be fully characterized. In the final analysis, however, the proof of many an algorithm's value is its performance under the stress of a real-world application.

Low-level vision algorithms. The MIT Vision Machine tries to integrate several vision cues to perform recognition and navigation tasks in unstructured environments.[11] It is also a testbed for progress in low-level vision algorithms, their parallel implementation, and integration. Parallel algorithms that execute in close to real time have been implemented for edge detection, stereo, motion, texture, and surface color. Their integration leads to a cartoon-like map of discontinuities in the scene. The output is interfaced to a parallel model-based recognition algorithm. The current implementation of the MIT Vision Machine uses two different kinds of edges. The first consists of zero-crossings in the Laplacian of the image, filtered through an appropriate Gaussian. The second consists of the edges found by Canny's edge detector.

The Drumheller-Poggio stereo algorithm[12] produces data that comprise one of the inputs to the integration stage of the Vision Machine. MIT researchers are exploring various extensions to this algorithm as well as the use of feedback from the integration stage. The stereo algorithm runs on the Connection Machine with good results on natural scenes at an average speed of one second.

The motion algorithm computes the optical-flow field, producing sparse or dense output depending on whether it uses edge features or intensities. This algorithm assumes that image displacements are small and that the optical flow is locally constant in a small region surrounding a point. It uses the sum of the pointwise squared differences between a patch in the first image centered at one point and a patch in the second image centered at a point corresponding to a displacement of the first point. This summation operation is efficiently implemented on the Connection Machine using scan computations. Each processor collects a vote indicating support that a patch of surface exists at that displacement. The algorithm iterates over all candidate displacements, recording each sum. Using nonmaximum suppression, the algorithm finally chooses the displacement that maximizes support. The corresponding displacement is the velocity of the surface patch.

Results from the low-level vision modules can be noisy and sparse. They are smoothed and made dense by exploiting known constraints within each process. This is the stage of data approximation, restoration, and fusion, performed using a Markov random-field model. The system output is a set of labeled discontinuities of the surfaces around the viewer. These surface properties are attributes of the physical world rather than of the images. The system has been implemented on the Connection Machine using multiple communication modes, including routing operations, scanning, and distance doubling.

Figures 11 and 12 show the results of the Vision Machine applied to the scene in Figure 10 and some of the intermediate steps. Figure 11 shows the brightness edges computed by the Canny algorithm at two different spatial scales. In the motion sequence, the teddy bear was rolling slightly on its back from one frame to the next. The first column of Figure 12 gives the results of the stereo, motion, texture, and color algorithms after initial smoothing to make them dense.[11] They represent the input to the Markov random-field process that integrates them with brightness edges. The central column of Figure 12 shows the reconstructed depth (from stereo), motion flow, texture, and color. The last column shows the discontinuities found by the Markov random-field process in each of the cues. Figure 13 shows a "cartoon" reconstruction, resulting from the union of the discontinuities in depth, motion, and texture, overlaid on the original image.

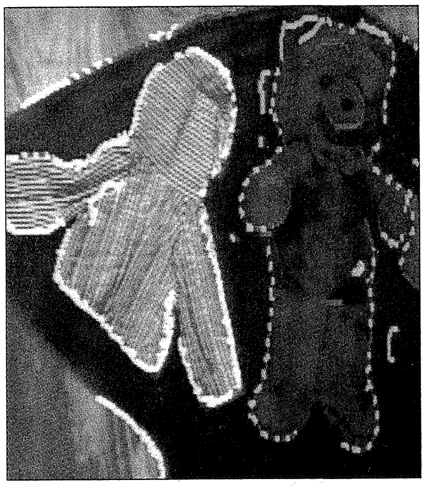


Figure 13. A cartoon representaton of the scene resulting from the union of depth and motion discontinuities.

The output of the integration stage provides a set of edges labeled in terms of discontinuities of surface properties, and represents a usable input to a model-based recognition algorithm.[13] Thus far, only discontinuities are used for recognition in the MIT Vision Machine; future work at MIT will also incorporate the properties of surfaces between discontinuities into the recognition process. Tucker[14] and Voorhees[15] have provided other recognition algorithms for the Connection Machine.

Parallel texture algorithms. Columbia University researchers implemented and tested parallel versions of Andrew Witkin's shape-from-texture algorithm, and improved the method by developing a new algorithm for determining local surface orientation, from rotationally invariant textures, based on the two-dimensional two-point autocorrelation of an image.[16]

This new method is computationally simple and easily made parallel. It uses information from all parts of the image, assumes only texture isotropy, and requires neither texels nor edges in the texture. When applied to locally planar patches of real textures such as roads, dirt, and grass, the resulting slant and tilt are highly accurate, even in cases where human perception is so difficult that people must be assisted by the presence of an

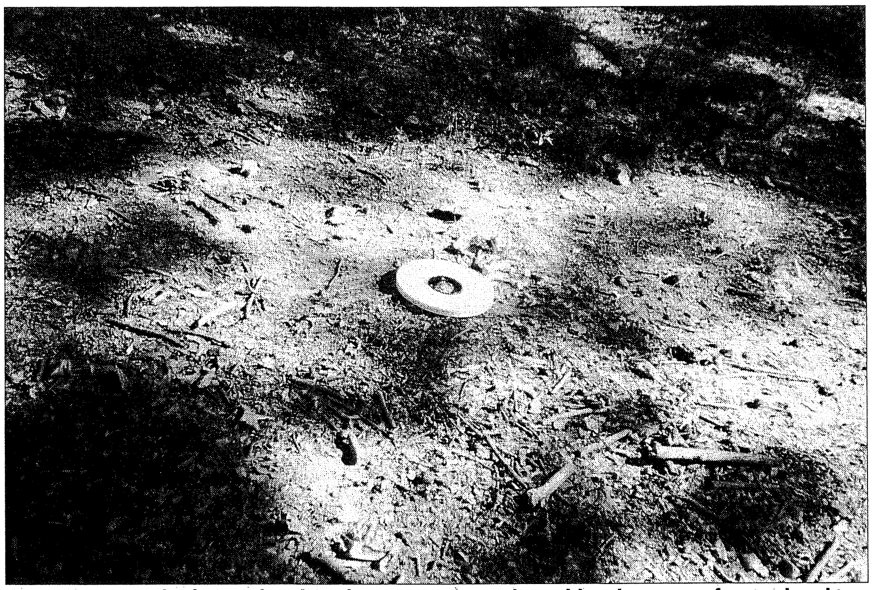


Figure 14. A patch of ground with surface orientation indicated by placement of a circular object.

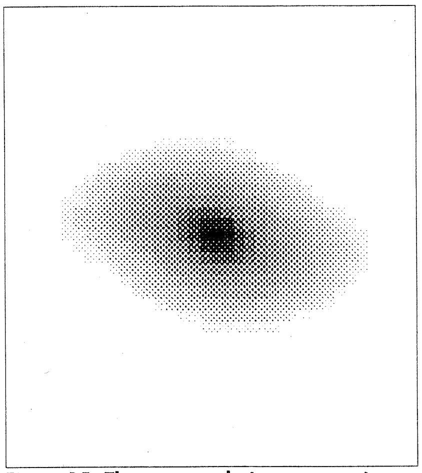


Figure 15. The autocorrelation computation, indicating surface slant and tilt.

artificially embedded circular object, as in Figure 14.

The algorithm has several mathematical elegancies, some of which aid its parallelization, and others make it robust to noise. For example, one of these makes the extraction of slant and tilt information equivalent to determining the parameters of an ellipse, and also reduces the algorithm's sensitivity to noise generated by charge-coupled-device cameras, such as the horizontal smearing of pixels.

Intrinsically straightforward, the parallel method was implemented on the Connection Machine, where it became even more elegant. Using image shifts to compute a finite window of the autocorrelation, we can compute surface orientation for surface patches in constant time. The Columbia team successfully demonstrated the algorithm on various natural textures. However, the assumption of isotropy appeared critical. Follow-up attempts to use the method for nonisotropic textures, even with preset heuristic biases, gave disappointing results.

Fusing stereo and texture. Columbia's main objective was to develop, implement, and integrate parallel multiresolution stereo and texture algorithms for determining local surface orientation and depth, to be used by the navigation systems of an autonomous land vehicle. The work on fusing texture and stereo had three components. The first was a new method for fusing multiple textural cues within a shape-from-texture component. The second was a novel method for handling stereo correspondence, surface interpolation, and surface segmentation all in one step, based on a new, unifying mathematical foundation. The third was a system for controlling the two systems' interaction. The full system was tested on real imagery, and parts of the system were parallelized.[17]

The first system assumed a coarse degree of parallelism to fuse five textural cues, such as shape-from-eccentricity and shape-from-spacing, into one machine perception of surface orientation. The system was demonstrated on both synthetic and real surface imagery (see Figure 14), in which some of the surfaces were curved and some even transparent (the first system to do so), with robust results: slant and tilt were usually recovered within five degrees (see Figure 15). An unanticipated further result was that the system also performed limited texture classification, based on the differential strengths with which the individual methods responded to a textured stimulus.

The second system was one-step stereo, based on applying the theory of reproducing kernel-based splines. Quantitatively, this results in a significantly higher percentage of correct left-right stereo matches. The system dealt naturally with occluded objects and with sharply slanted surfaces, such as roads as seen from a vehicle. It even handled transparency.

Columbia combined the stereo and texture processes in a third system. This system combines information in two fundamentally different ways — by intraprocess and interprocess integration — and therefore admits both fine and coarse parallelization. Communication is standardized via a blackboard. In operation, the stereo process uses the relative accuracy and sparseness of the centroid of texels to begin feature localization, later switching to traditional zero-crossings. The work is further characterized by the choice of a smoothness measure; roughly, it minimizes variation in the 1.5 (not the second) derivative, a value supported by recent human psychophysical measurements. To properly balance stereo, which is dense, with the sparser texture processing, the system achieves final integration by weighted average. Applying this fusion technology to a real-world problem, Columbia researchers built an oceanographical system now being used by the university's Lamont-Doherty Geological Observatory to infer from gravitational information the geological structures below the ocean floor.

Parallel surface interpolation. The output of middle-level vision algorithms is often a sparse depth map, or a sparse surface-orientation map, or both. Researchers often need to interpolate full surfaces (such as roads) through such points. Simulating a fine-grained pyramidal architecture, Columbia quantitatively evaluated the efficiency and accuracy of known and novel depth-interpolation methods.[18] The optimal method can be mapped onto a pyramid machine in a way that requires

Table 1. Tasks from the first DARPA image-understanding benchmark.

11×11 Gaussian convolution of a 512×512 8-bit image
Detection of zero-crossings in a difference-of-Gaussians image
Construct and output a border pixel list
Label connected components in a binary image
Hough transform of a binary image
Convex hull of 1000 points in 2D real space
Voronoi diagram of 1000 points in 2D real space
Minimal spanning tree across 1000 points in 2D real space
Visibility of vertices for 1000 triangles in 3D real space
Minimum-cost path through a weighted graph of 1000 nodes of order 100
Find all isomorphisms of a 100-node graph in a 1000-node graph

local computations only; other methods, though more robust, intrinsically require global computations.

Robot navigation applications. Warp has been applied to several robot navigation problems. It is often used as a processor for sensory data; however, it performs more than low-level vision functions. In several cases it has been used for all processing up to the detection of the primary object of interest, either obstacles or the road in front of the vehicle, depending on the system.

Carnegie Mellon researchers applied Warp to road following and navigation in several different approaches. Warp processing included image pyramid generation, feature point detection, pyramid-based correlation, and thresholding. Later systems used color and texture classes to classify pixels according to whether they resembled the color and texture of the road or the background. Using Warp, the color classes were updated at each step. In the most advanced system, SCARF (for "supervised classification applied to road following"), all visual processing up to road detection was implemented on the Warp array.[3] Images were fed to the Warp pipeline, and the predicted road positions were output after processing (the color classes were maintained internally in the pipeline). The total time for execution on Warp was one second, with two seconds for some limited displays and Navlab control.

Carnegie Mellon applied a conventional back-propagation neural-network training algorithm on Warp to road following. In ALVINN ("autonomous land vehicle in a neural net"), the bottom layer of the three-layer neural network was supplied with a reduced-size image, and the top layer gave the predicted steering angle for the vehicle to stay on the road. A person drove the vehicle during training, and the road image and steering angle were fed to the back-propagation algorithm on Warp. The result was that after just a few minutes of training, the neural network could successfully drive the vehicle.

Apply and W2 applications. Apply has been used in various ways at Warp sites. For example, it was used to implement the G-State algorithm as part of the SCORPIUS project, which involved constructing a system to aid in the interpretation of aerial reconnaissance photos. G-State includes several steps, namely edge mask convolution, thinning, linking, and removing headerless links.

Apply was also used in synthetic-aperture-radar image analysis as part of the Adries automatic target recognition project. Adries used several Web routines that are written in Apply, including convolution. Adries also used W2 programs from Web, such as connected components, and developed new W2 programs.

The Mosaic project was a hardware effort at ESL intended to support threat analysis as part of the Strategic Defense Initiative. Mosaic used Apply programs from Web to develop a stereo image-processing system based on image pyramid generation followed by pyramid-based image correlation to calculate disparity. ESL also created new W2 code to perform correlations.

The most extensive application of new Apply and W2 programming outside of Carnegie Mellon was at Martin Marietta in the DARPA-sponsored Autonomous Land Vehicle project.[19] Warp was used for the off-road vehicle navigation, which involved transforming a laser scanner range image into an obstacle position map, and then calculating a path through the map. Criteria, such as maximum obstacle height and maximum road tilt, were used to determine whether a particular vehicle orientation at a particular point was acceptable. An Apply program calculated acceptable orientations at every point, and then linked them together in a path that took the vehicle to the goal.

Pipelined parallel image-analysis experiments. Pipelined image processors can produce histograms and Fourier transforms many times per second, report the locations and characteristics of blobs in an image, and do template matching and other image-processing applications at frame rates. Using the Max-Video system, Rochester has demonstrated fast object recognition with color histograms, tracking moving objects, binocular vergence, binocular stereo, full-frame kinetic depth calculations, foveal-peripheral processing for learning skilled camera-motion sequences, segmentation by sensor motion, disparity-filtered segmentation, and the detection and localization of moving objects by a moving observer.

Performance analysis

Applications demonstrate the utility of parallel hardware and software tools. However, comparing architectural performance based on different applications is quite difficult. Each application is developed with different requirements, constraints, and goals, some of which depend on the architecture itself. To better understand the performance of different architectures, we must program them to execute the same task — typically a benchmark. Thus, to analyze the performance of the parallel architectures developed under Strategic Computing, DARPA sponsored a series of image-understanding benchmark exercises.

In 1986, the University of Maryland developed the first benchmark for DARPA, consisting of the tasks listed in Table 1. The results of testing the benchmark on several machines have been published elsewhere.[20] However, it is difficult to compare the results because no methods were specified for the tasks, so that the timings were often more indicative of the programmer's cleverness than of a machine's capabilities. In addition, input data were not provided, so that the test data developed by different groups varied considerably in difficulty.

Because the benchmark consisted of isolated tasks, it did not measure performance related to interactions between

Table 2. Execution times in seconds for the data set sample.

	Sun 3	CM-2	Warp	IUA
Total executiion time	797.88		43.60	0.0844445
Overhead	5.08	0.25500	14.14	0.0139435
Labeling connected components	27.78	0.10000	3.98	0.0000596
Computing rectangles from intensities	6.50	0.25336	5.50	0.0161694
Median filter (floating point)	246.66	0.01500	10.70	0.0005625
Sobel (floating point)	135.48	0.00800	0.48	0.0026919
Total time in low-level processing	421.50	0.63000	34.80	0.0334269
Initial graph match (probe list)	24.46		0.42	0.0155662
Match-strength probes	72.98		9.10	0.0327150
Hough probe	253.70		15.30	0.0068430
Presentation of results	24.80		2.60	0.0022826

vision system components. Also, no definition of a correct solution was provided. Having a known correct solution is important, since problems in vision are often ill-defined, and it is possible to argue for the correctness of different solutions involving different amounts of computation. In a benchmark, however, the goal is to test the performance of different machines doing comparable work.

The second benchmark. The major result of the first DARPA benchmark exercise was further insight into how to construct an image-understanding benchmark.[21] A new benchmark was needed to test system performance on a task that approximates an integrated solution to a vision problem. DARPA assigned the task of constructing the Integrated Image-Understanding Benchmark to the University of Massachusetts at Amherst and the University of Maryland. A complete solution with test data was distributed. The specification was designed to minimize shortcutting in solving the problem, yet be flexible enough to permit implementation on very different types of parallel architectures.

Task description. The new benchmark involves recognizing an approximately specified 2.5-dimensional mobile sculpture in a cluttered environment, given images from intensity and range sensors. A set of models is provided that represent a collection of similar sculptures, and the task is to pick the model that best matches the scene. The intention is that neither of the input images, by itself, is sufficient to complete the task.

The sculpture is a collection of 2D rectangles of various sizes, brightnesses, orientations, and depths. Each rectangle's orientation is normal to the viewing axis, with constant depth across its surface. The images are constructed under orthographic projection. While the rectangles have no intrinsic depth component, depth is a factor in the spatial relationships between rectangles; hence the notion that the sculpture is 2.5-dimensional.

The clutter consists of additional rectangles, similar to those in the sculpture. Rectangles may partially or completely occlude other rectangles. A rectangle can also disappear when another of the same brightness or slightly greater depth is directly behind it.

Test images are created by selecting one model from a set, then rotating and translating it as a whole. Its individual elements are then perturbed slightly. Next, a collection of spurious rectangles is created. All of the rectangles are then ordered by depth and drawn in the two image arrays. Finally, Gaussian distribution noise is added to the depth image.

The processing scenario. Processing begins with low-level operations on the intensity and depth images, followed by grouping operations on the intensity data that extract candidate rectangles. These are used to form partial matches with the models. Multiple hypothetical poses can be established for each model. For each pose the depth and intensity images are probed from the top down. Each probe tests a hypothesis for the existence of a rectangle in a given location in the images. Rejection of a hypothesis, which requires strong evidence that a rectangle is absent, eliminates the pose. Confirmation of the hypothesis results in the computation of a match strength for the rectangle, and an update of its representation in the model. After all the probes have been performed, an average match strength is computed for each pose and the greatest of these is selected as the best match.

The benchmark specification lists the steps to be applied in implementing a solution. These steps include labeling connected components, K-curvature, extracting corners with geometric constraints, median filter, Sobel, convex hull, rectangle parameter computation, graph matching, probe list generation, probing of image data (windowed Hough transform and pixel classification), model match evaluation, and display of selected and updated models. Furthermore, for each step, a recommended method is described. However, since some methods might not work for a given architecture, implementers can substitute other methods for individual steps. When an implementation must differ from the specification, the implementer must supply a justification.

Results. Table 2 shows the results of running the benchmark on a Sun 3/160, a CM-2, a Warp, and an IUA. The times for the CM-2 were extrapolated from execution on a machine with 32,768 processors to one with 65,536 processors. Only the bottom-up portion of the benchmark was implemented on the Connection Machine. The Warp implementation used the systolic array to perform the majority of the bottom-up processing and all probes of the image data, while the Sun 3 host machine performed all the model matching as well as some of the image processing in parallel with the Warp. The IUA implementation was run on the IUA simulator, configured to represent a full-scale system with 262,144 low-level processors and 4,096 intermediate-level processors; however, even a detailed simulation will not be as accurate as execution on actual hardware.

The table shows that all the parallel architectures provided a significant speedup over a conventional uniprocessor built with comparable technology. The massively parallel single-instruction multiple-data arrays in the Connection Machine and the IUA are obviously faster than the Warp for low-level image processing. However, raw speed is only one element of machine performance. The Warp is also considerably smaller and less expensive than these arrays. The goals of strategic computing included developing architectures to address a wide range of applications, and the Warp fills an important niche in the application spectrum.

Current issues

Research that began with the Strategic Computing vision program has continued to unfold, and the experience gained from this research is forming the basis for new efforts in parallel hardware and software. For example, as a result of the increased processing speed gained from parallelism, we now know that vision is much easier in an active rather than a passive context: The added technical complications of an active agent are offset by increased information flow and reduced uncertainties. This change in the basic approach to vision complements nicely the Strategic Computing vision goal of developing large reactive systems.

Some of the most ambitious computing applications are reactive systems characterized by extensive real-time interaction with a largely unpredictable environment that is too complex to be modeled, monitored, or controlled completely within the system's resources. These systems are of great importance to such diverse fields as industrial manufacturing, command and control applications, undersea monitoring and exploration, and automated navigation and reconnaissance.

The University of Rochester is currently exploring the following parallel-vision software architecture issues:

- Active vision. Using pipelined and multiple-instruction multiple-data stream parallel architectures, active vision should interact with its environment at a low level and with planning and uncertain inference at a high level. The potential power of this approach has barely been tapped.[22]
- Multimodel programming. Different notions of process, communication, sharing, and protection are needed at different levels of the vision problem.
- Real-time issues. Computing and physical resources are generally inadequate to achieve optimal reactive behavior. System architectures and scheduling disciplines should be designed to achieve suboptimal but acceptable results in real time.
- Debugging and performance analysis. Existing tools do not provide debugging within and between different models of parallelism, or on- and off-line performance measures for feedback to applications and for tuning based on long-term behavior.

Analog and hybrid VLSI implementations of vision machine components. Once algorithms and systems are perfected, it makes sense to compile them in silicon to make them faster, cheaper, and smaller. Rather than directly hardwiring algorithms, "algorithmic hardware" designs must be considered that utilize the local, symmetric nature of low-level vision problems. This requires an iterative process, as the algorithm influences the hardware design, and hardware constraints influence the algorithm.

PIPELINED PROCESSING OR OTHER METHODS OF TIME-SHARING COMPUTING POWER MIGHT BE ABLE TO COMPENSATE FOR THE LOWER DEGREE OF CONNECTIVITY.

Within the integrated circuit, we can represent image data as a digital word or an analog value. While digital computation is accurate and fast, digital circuits do not have as high a degree of functionality per device as analog circuits. Therefore, analog circuits should allow much denser computing networks. This is particularly important when integrating computational circuitry and photosensors, which will help alleviate the I/O bottleneck of transferring data between components. However, analog circuits are limited in accuracy and are difficult to characterize and design. Specific VLSI implementations must therefore trade some computational flexibility for faster performance and a higher degree of integration.

When implementing algorithms in integrated circuits, it is not clear what level of parallelism is needed. While biology uses three dimensions to construct highly interconnected networks, VLSI is limited to 2.5 dimensions, making such networks difficult and costly to implement. However, integrated circuits are approximately four orders of magnitude faster than the electrochemical components of biology. This suggests that pipelined processing or other methods of time-sharing computing power might be able to compensate for the lower degree of connectivity. Clearly, a VLSI vision system might not resemble biological vision systems.

MIT researchers have fabricated an analog chip (6.5×5.9 millimeters, made with a 4-micron double-polysilicon, double-metal charge-coupled-device process), which integrates an analog signal processor with a full fill-factor CCD imager. The device computes edges faster than 1,000 frames per second. This represents an example for more complex chips performing algorithms such as motion detection, for which preliminary design concepts already exist.

Machine-independent global image-processing operations. Apply has shown us that it is possible to treat local image-processing operations from a machine-independent point of view. Using this approach, we can

- implement the same program efficiently on various machines simply by recompiling,
- write programs with less effort, even as compared with sequential code,
- create software for new machines with little effort,
- explore models of parallelism, and
- more easily test, develop, and compare parallel computers.

A topic of current research is to extend this approach to global image-processing operations, such as histograms, Hough transforms, connected components, and image warping. Research at Carnegie Mellon has led to two significant developments. The first is the reversible-operations theorem: Any operation that can be computed forward or backward on a data structure can be computed in parallel using a simple split-and-merge model, in which the data is partitioned among processors, processed separately, and then combined with a merging function. The split-and-merge programming model is applicable to a wide range of image-processing algorithms and includes any reversible operation. Only error-diffusion halftoning and some types of region-merging algorithms are not reversible.

The second result is a programming language called Adapt, which implements the split-and-merge algorithm. In Adapt the programmer describes the function to be applied locally at each pixel, using sequential semantics; results from a previous pixel can be reused. The programmer also supplies a function that initializes the state at the beginning of a row. These two functions are then applied at different processors to produce results for different regions of the image, and the results are combined with a third function also written by the programmer. The programmer can also supply a "last" function that is executed once at the end of all other processing to clean up results and discard intermediate variables. For example, in a histogram, the function local to a pixel increments a particular histogram element, and the function applied at the beginning of a row zeroes the histogram. The combining function adds the histograms together, and the last function divides each element of the histogram by the total number of histogram elements to compute a frequency table.

Adapt has been implemented for Unix/C, Warp, I-Warp, the Ektron Image-Boss, and Nectar, a crossbar-connected network computer developed at Carnegie Mellon. Several different programs for global image processing have been developed; some of these, including connected components and image halftoning, exhibit better speed or higher quality results than previous algorithms.

An integrated programming model for the IUA. With an architecture as complex as the IUA, one of the first questions that comes to mind is how to program it. Currently it is programmed in a different dialect of C at each level, and programs at the different levels interact through various simple protocols. As larger systems are developed that cross levels more frequently, a programming model must be provided that integrates all three levels.

One area currently under exploration is extending Common Lisp with both data-parallel and control-parallel constructs. This would most likely be built on the existing Parallel Implementation of Common Lisp compiler, developed at the University of Massachusetts. PICL currently provides futures-based control parallelism at four granularities: node, process, task, and thread. Extending this with data-parallel constructs would let users program the entire IUA using a single language and environment.

COLLABORATION BETWEEN the DARPA Strategic Computing architecture and vision communities has resulted in new parallel hardware, software tools, algorithms, and metrics, which have in turn led to a better understanding of both the computational requirements and potential sources of parallelism in vision. Perhaps even more significant, parallel processing has changed the way vision researchers work and has led to wholly new avenues of exploration.

Before parallel processors were available, experiments often took days to execute and the number of trials of an algorithm was necessarily limited. Using parallel processors, vision researchers can conduct an empirical study much more quickly and thoroughly. New approaches, such as active vision, become feasible. When these new approaches are analyzed, they lead to new sets of requirements for parallel architectures, and the cycle repeats. Of course, the priming of just such a cycle is what Strategic Computing was all about.

Acknowledgments

This work was supported in part by the Defense Advanced Research Projects Agency under contracts DACA76-85-C-0001, DACA76-85-C-0002, DACA76-85-C-0003, DACA76-85-C-0008, DACA76-85-C-0010, DACA76-86-C-0015, DACA76-86-C-0019, and DACA76-86-C-0024, all monitored by the US Army Engineer Topographic Laboratory.

References

1. D. Hillis, *The Connection Machine*, MIT Press, Cambridge, Mass., 1985.

2. M. Annaratone et al., "The Warp Computer: Architecture, Implementation, and Performance," *IEEE Trans. Computers*, Vol. 36, No. 12, Dec. 1987, pp. 1,523-1,538.

3. C. Thorpe, "Toward Autonomous Driving: The CMU Navlab," *IEEE Expert*, Vol. 6, No. 4, Aug. 1991, Part 1, pp. 31-42, and Part 2, pp. 44-52.

4. C.C. Weems et al., "The Image-Understanding Architecture," *Int'l J. Computer Vision*, Kluwer Academic Publishers, Norwell, Mass., Vol. 2, pp. 251-282.

5. H. Ibrahim, "On the Implementation of Pyramid Algorithms on the Connection Machine," *Proc. 1988 DARPA Image-Understanding Workshop*, Morgan Kaufmann, San Mateo, Calif., 1988, pp. 634-639.

6. L.G.C. Hamey, J.A. Webb, and I-C. Wu, "An Architecture-Independent Programming Language for Low-Level Vision," *Computer Vision, Graphics, and Image Processing*, Vol. 48, No. 6, Nov. 1989, pp. 246-264.

7. S. Borkar et al., "I-Warp: An Integrated Solution to High-Speed Parallel Computing," *Proc. Supercomputing 88*, CS Press, Los Alamitos, Calif., 1988, pp. 330-339.

8. M.L. Scott, T.J. LeBlanc, and B.D. Marsh, "Multi-Model Parallel Programming in Psyche," *Proc. Second ACM SIGPlan Symp. on Principles and Practice of Parallel Programming*, ACM Press, New York, 1990, pp. 70-78.

9. T.J. LeBlanc, J.M. Mellor-Crummey, and R.J. Fowler, "Analyzing Parallel Program Executions Using Multiple Views," *J. Parallel and Distributed Computing*, Vol. 9, June 1990, pp. 203-217.

10. C.C. Weems and J.R. Burrill, "The Image-Understanding Architecture and Its Programming Environment," in *Parallel Architectures and Algorithms for Image Understanding*, V.K. Prassana-Kumar, ed., Academic Press, Orlando, Fla., 1991, pp. 525-562.

11. T. Poggio et al., "The MIT Vision Machine," *Proc. 1988 DARPA Image-Understanding Workshop*, Morgan Kaufmann, San Mateo, Calif., 1988, pp. 177-198.

12. M. Drumheller and P. Poggio, "On Parallel Stereo," *Proc. IEEE Conf. Robotics and Automation*, CS Press, Los Alamitos, Calif., 1986, pp. 1,439-1,488.

13. T. Cass, "Robust Parallel Computation of 2D Model-Based Recognition," *Proc. 1988 DARPA Image-Understanding Workshop*, Morgan Kaufmann, San Mateo, Calif., 1988, pp. 640-650.

14. L.W. Tucker, C.R. Feynman, and D.M. Fritzsche, "Object Recognition Using the Connection Machine," *Proc. IEEE Computer Vision and Pattern Recognition Conf.*, CS Press, Los Alamitos, Calif., 1988, pp. 871-878.

15. H.L. Voorhees, D.M. Fritzsche, and L.W. Tucker, "Exploiting Data Parallelism in Vision on the Connection Machine,"

Glossary

Blackboard: A globally accessible data structure on which hypotheses and evidence can be posted for use by multiple processing agents, called knowledge sources.

Convolution: A basic image-processing operation. The convolution of two functions f and g is a function h of a displacement y, as defined by the following:

$$h(y) = f \times g = \int_{-\infty}^{\infty} f(x)g(y-x)dx$$

This can be thought of as sweeping a function in one dimension past the other function in the other dimension. In practice, discrete versions of the convolution function are implemented.

Digitizer: A device for taking the analog signal output by an imaging sensor and converting it into an array of values that can be processed by a digital computer.

Disparity: The displacement of points or features between two images of a scene, obtained from different viewpoints. The two viewpoints may correspond to a stereoscopic view or to points along the trajectory of a moving sensor.

Distance doubling: A parallel-processing primitive in which messages are transmitted repeatedly, but over distances that increase by successive powers of two. Thus, a message is first passed to an adjacent processor, which returns the address of its neighbor. A message can then be sent directly to that processor, which is a distance of two away. After that processor returns the address of its two-away neighbor, a message can be sent directly to a four-away processor, and so on.

Foveal-peripheral processing: Processing that emulates the combination of foveal and peripheral vision found in the human eye. Typically uses a pair of sensors, one with a wide-angle lens to provide low-resolution coverage of a large area, and another with a telephoto lens to give a high-resolution view of a small area. Specialized sensors can also be used to achieve this effect.

Frame buffer: A device for storing one or more digitized images, often for I/O with a parallel processor. Sometimes also used to store intermediate processing results.

Futures: A multiple-input multiple-data parallel-processing primitive in which an operation (usually a function call) is initiated, but its results are not needed until they are actually used in a later computation. Thus, the initiating instruction stream can proceed in parallel with the spawned operation until the results are needed.

Gaussian: A mask that, when convolved with an image, has the effect of smoothing the image, thereby reducing certain types of noise. The discrete form of the convolution computes a weighted average of each pixel with those surrounding it, out to some radius. The weights correspond to the Gaussian distribution function e^{-x^2}. Thus, the weights diminish with the distance away from any given pixel under consideration.

Hough transform: An operator that can be used to detect lines in an image. Each member of a subset of the pixels in an image (chosen by some criteria) votes for all lines that might pass through it. The votes are accumulated in a two-dimensional histogram array, indexed by angle and distance from the origin. Peaks in the histogram correspond to lines that were "voted for" by many pixels. The Hough transform has the advantage that it can detect lines made up of fragments that are widely separated in an image. The transform can also be generalized for detecting similarities in other types of features.

Hypercube: An n-dimensional cube. Typically refers to a graph of communication links in a parallel processor. The corners of the hypercube are occupied by processors, and the edges are communication links between the processors.

Image pyramid: A set of images at successively coarser resolutions. The input image is at the base of the pyramid and each level above is typically a factor of four smaller. Also refers to parallel-processing architectures with a similar organization. An image pyramid is often the basis for multiresolution processing.

Isotropic texture: A texture whose properties are independent of orientation.

K-curvature: A technique for measuring curvature along a line. For each point on the line, find the two neighboring points on the line that are a distance K away. The angle formed by the three points is then computed as an approximate measure of local curvature.

Laplacian: A convolution for extracting edges from an image. It is a discrete approximation to the Laplacian differential operator

$$\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

where the image is cast as a continuous function f.

Markov random field: A Markov random process is one whose next state in time is a random function of its current state, and independent of previous states. A Markov random field is a generalization of this concept to a spatial field, such that the next state of an element in the field depends on the current state of itself and its neighbors.

Massively parallel: A parallel processor with more than 1,064 processing elements.

Multiple-instruction multiple-data: Multiple instruction stream multiple data stream parallelism is one of the classes in the widely recognized processor taxonomy defined by Michael Flynn. A processor of this type has multiple instruction fetch and execution units. It is usually composed of multiple standard uniprocessors linked by a communication mechanism.

Multiresolution: Processing that takes place at multiple scales of image resolution. For example, it might be easier to detect large features in an image using a coarser version of the data, and then use this information to guide refinement of feature detection at successively higher resolutions.

Optical flow: In successive images taken from a moving sensor, features appear to flow away from the point toward which the sensor is moving (called the focus of expansion). This effect is known as optical flow. The field of vectors representing the apparent motions of points in the image sequence is called the optical-flow field.

Pipelined parallel: A form of parallelism in which multiple computational elements are configured into stages. Data enter the elements in a given stage, are processed, and the results are passed on to the elements in the next stage. For example, an image might be passed through a pipeline consisting of several convolution processors and a classification processor.

Scan computation: A single-instruction multiple-data parallel-processing primitive in which an operation is applied so that it appears that a set of processors have been operated on in a sequential (scan) ordering. For example, enumerating a set of processors would appear to be a sequential task (each processor must learn the number of the processor preceding it). In a processor such as the Connection Machine, however, such operations can be performed in logarithmic time.

Shape-from-*X*: Refers to a wide range of methods for extracting surface information from image data. For example, shape-from-shading, shape-from-texture, etc.

Single-instruction multiple-data: Single instruction stream multiple data stream parallelism is one of the classes in the processor taxonomy defined by Michael Flynn. Such a processor has a single instruction fetch unit that broadcasts each instruction to an array of processing elements. These elements typically consist of an arithmetic logic unit, local data memory, and an interprocessor communication mechanism.

Single-program multiple-data: A variation of the MIMD mode of parallelism in which, like SIMD processing, each processor executes the same program. However, each processor may branch independently within the code.

Sobel: A convolution operator whose results are the image gradient in the x and y directions.

Spectral albedo: The albedo is the ratio of reflected light to incident light. Spectral albedo is the albedo measured within a specified set of wavelength bands in the visual (or near-visual) spectrum. In vision, this often refers to the albedo measured in the primary color bands (red, green, and blue).

Spline: As used here, short for B-Spline; a piecewise polynomial curve that is related to a guiding polygon, defined by a set of points in a plane.

Systolic processing: A form of parallelism in which multiple processors are chained together such that the results of one processor's computations are passed to the next processor in the (typically linear) chain. Similar to pipelined parallelism, except that the processors are usually general-purpose programmable devices.

Texel: A texture element. A primitive component of a texture that is independent of position, size, and orientation.

Proc. 10th Int'l Conf. Pattern Recognition, CS Press, Los Alamitos, Calif., 1990, pp. 617-622.

16. L. Brown and H. Shvaytser, "Surface Orientation from Projective Foreshortening of Isotropic Texture Autocorrelation," *Proc. IEEE Computer Vision and Pattern Recognition Conf.*, CS Press, Los Alamitos, Calif., 1988, pp. 510-514.

17. M. Moerdler and T. Boult, "The Integration of Information from Stereo and Multiple Shape-from-Texture," *Proc. IEEE Computer Vision and Pattern Recognition Conf.*, CS Press, Los Alamitos, Calif., 1988, pp. 524-529.

18. D. Choi and J.R. Kender, "Solving the Depth Interpolation Problem on a Parallel Architecture with a Multigrid Approach," *Proc. IEEE Computer Vision and Pattern Recognition Conf.*, CS Press, Los Alamitos, Calif., 1988, pp. 189-194.

19. K.E. Olin and D.Y. Tseng, "Autonomous Cross-Country Navigation," *IEEE Expert*, Vol. 6, No. 4, Aug. 1991, pp. 16-30.

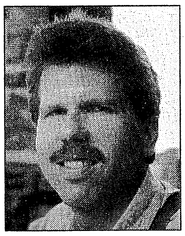
20. A. Rosenfeld, "A Report on the DARPA Image-Understanding Architectures Workshop," *Proc. 1987 DARPA Image-Understanding Workshop*, Morgan Kaufmann, San Mateo, Calif., 1987, pp. 298-302.

21. C.C. Weems et al., "The DARPA Image-Understanding Benchmark for Parallel Computers," *J. Parallel and Distributed Computing*, Vol. 11, No. 1, 1991, pp. 1-24.

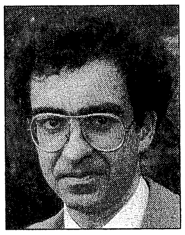
22. R.D. Rimey and C.M. Brown, "Selective Attention as Sequential Behavior: Modeling Eye Movements with an Augmented Hidden Markov Model," *Proc. 1990 DARPA Image-Understanding Workshop*, Morgan Kaufmann, San Mateo, Calif., 1990, pp. 840-849.

Charles C. Weems is a research assistant professor and director of the Parallel Image-Understanding Architectures research group at the University of Massachusetts. His research interests include parallel architectures to support low-, intermediate-, and high-level computer vision, benchmarks for vision, parallel-programming languages, and parallel-vision algorithms. He received his BS and MA degrees from Oregon State University in 1977 and 1979, respectively, and his PhD from the University of Massachusetts at Amherst in 1984, all in computer science. He is a member of the IEEE Computer Society, IEEE, and ACM.

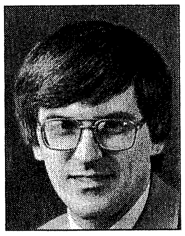
Christopher Brown is a professor of computer science at the University of Rochester. His current research interests are computer vision and robotics, especially integrated parallel systems performing animate vision (the interaction of visual capabilities and motor behavior). This interest leads to involvement in real-time operating and control systems, real-time vision algorithms, and the basic connections between planning, learning, and control. He received his BA from Oberlin in 1967 and his PhD from the University of Chicago in 1972, and he completed a postdoctoral fellowship at the School of Artificial Intelligence at the University of Edinburgh in 1974. He is an editorial-board member of *Computer Vision, Graphics, and Image Processing* and *The International Journal of Computer Vision*.



Jon A. Webb is a research computer scientist in the School of Computer Science at Carnegie Mellon University. He received his BA in mathematics from the University of South Florida in 1975, his MS in computer science from Ohio State University in 1976, and his PhD in computer science from the University of Texas at Austin in 1981. His research focuses on parallel computer vision. He developed the Apply language and helped to develop the Warp and I-Warp computers. He is now studying architecture-independent programming of parallel computers and is developing the Adapt programming language.



Tomaso Poggio is the Uncas and Helen Whitaker Professor in the Department of Brain and Cognitive Sciences at MIT. He directs research in computational vision at MIT's AI Lab and is codirector of the Center for Biological Information Processing. A corporate fellow of Thinking Machines Corp., he helped develop applications for the Connection Machine. He received his doctorate in theoretical physics from the University of Genoa in 1970. He is on the editorial boards of several journals, including *Biological Cybernetics*, *Neural Computation*, and *Synapse*. He is an AAAI fellow and a member of IEEE, the Society for Neuroscience, and the Neurosciences Research Program.



John R. Kender is one of the founding faculty of the Department of Computer Science of Columbia University and was one of the first National Science Foundation Presidential Young Investigators. He was coeditor of the Proceedings of the IEEE special issue on computer vision (August 1988), and program cochair for the 1989 IEEE Conference on Computer Vision and Pattern Recognition. His primary research interests are the perception of objects and surfaces from middle-level image cues such as texturings and shadows. He holds a patent on a technique for noncontact inspection of industial parts. He received his PhD in computer science from Carnegie Mellon University in 1980.

Readers can contact the authors in care of Charles C. Weems, University of Massachusetts at Amherst, Lederle Graduate Research Center, Amherst, MA 01003.