The DARPA Image Understanding Benchmark For Parallel Computers

Charles Weems, Edward Riseman
Allen Hanson, Azriel Rosenfeld

COINS TR90-98

October 1990

The DARPA Image Understanding Benchmark for Parallel Computers*

Charles Weems, Edward Riseman, Allen Hanson

Computer and Information Science Department

University of Massachusetts at Amherst

Azriel Rosenfeld

Center for Automation Research

University of Maryland

* This work was supported in part by the Defense Advanced Research Projects Agency under contract number DACA76-86-C-0015, monitored by the U.S. Army Engineer Topographic Laboratories.

Running head:

Tha DARPA Image Understanding Benchmark

Point of contact:

Prof. Charles C. Weems Jr.
Computer and Information Science Department
Lederle Graduate Research Center
University of Massachusetts
Amherst, MA 01003

Phone: (413) 545-3163

Internet: Weems@CS.UMass.EDU

Short Abstract

This paper describes a new effort to evaluate parallel architectures applied to knowledge-based machine vision. The Integrated Image Understanding Benchmark consists of a model-based object recognition problem, given two sources of sensory input, intensity and range data, and a database of candidate models. The models consist of configurations of rectangular surfaces, floating in space, viewed under orthographic projection, with the presence of both noise and spurious non-model surfaces. A partially-ordered sequence of operations is specified that solves the problem, along with a recommended algorithmic method for each step. In addition to discussing the development and specification of the new benchmark, this paper presents the results from running the benchmark on the Connection Machine, Warp, Image Understanding Architecture, Associative String Processor, Alliant FX-80, and Sequent Symmetry. The results are discussed and compared through a measurement of relative effort, which factors out the effects of differing technologies.

List of Symbols

x times e.g. 512 x 512

% percent

* asterisk e.g. *LISP (not a footnote reference)

$\frac{1}{2}$ one half e.g. $2\frac{1}{2}$ dimensional

Abstract

This paper describes a new effort to evaluate parallel architectures applied to knowledge-based machine vision. Previous vision benchmarks have considered only execution times for isolated vision-related tasks, or a very simple image processing scenario. However, the performance of an image interpretation system depends upon a wide range of operations on different levels of representations, from processing arrays of pixels, through manipulation of extracted image events, to symbolic processing of stored models. Vision is also characterized by both bottom-up (image-based) and top-down (model-directed) processing. Thus, the costs of interactions between tasks, input and output, and system overhead must be taken into consideration. Therefore, this new benchmark addresses the issue of system performance on an integrated set of tasks.

The Integrated Image Understanding Benchmark consists of a model-based object recognition problem, given two sources of sensory input, intensity and range data, and a database of candidate models. The models consist of configurations of rectangular surfaces, floating in space, viewed under orthographic projection, with the presence of both noise and spurious non-model surfaces. A partially-ordered sequence of operations is specified that solves the problem, along with a recommended algorithmic method for each step. In addition to reporting the total time and the final solution, timings are requested for each component operation, and intermediate results are to be output as a check on accuracy. Other factors such as programming time, language, code size, and machine configurations are to be reported. As a result, the benchmark can be used to gain insight into processor strengths and weaknesses, and may thus help to guide the development of the next generation of parallel vision architectures.

In addition to discussing the development and specification of the new benchmark, this paper presents the results from running the benchmark on the Connection Machine, Warp, Image Understanding Architecture, Associative String Processor, Alliant FX-80, and Sequent Symmetry. The results are discussed and compared through a measurement of relative effort, which factors out the effects of differing technologies.

Introduction

Knowledge-based image understanding presents an immense computational challenge that has yet to be satisfied by any parallel architecture. The challenge is not merely to provide a greater quantity of operations per second, but also to supply the necessary forms of computation, communication, and control. Consider that a sequence of images at medium resolution (512 x 512 pixels) and standard frame rate (30 frames per second) in color (24 bits per pixel) represents a data input rate of about 23.6 million bytes per second and, in a typical interpretation scenario, many thousands of operations may be applied to each input pixel in order to enhance, and segment an image and to extract various features from it. But in addition, a vision system must organize extracted image features via perceptual grouping mechanisms, locate relevant models in a potentially vast store of knowledge and compare them to partial models derived from the input data, generate hypotheses concerning the environment of the sensor, resolve conflicting hypotheses to arrive at a consistent interpretation of the environment, manage and update stored knowledge, and so on.

While traditional supercomputing benchmarks may be useful in estimating the performance of an architecture on some types of image processing tasks, as noted by Duff [3], those benchmarks have little relevance to the majority of the processing that takes place in a vision system. Nor has there been much effort to define a vision benchmark for supercomputers, since those machines in their traditional form have usually been viewed as inappropriate vehicles for knowledge-based vision research. However, now that parallel processors are becoming readily available, and because they are viewed as being better suited to vision processing, researchers in both machine vision and parallel architecture are taking an interest in performance issues with respect to vision. The next section summarizes the prominent other work that has been done in the area of vision benchmarks to date.

Review of Previous Vision Benchmark Efforts

One of the first parallel processor benchmarks to address vision-related processing was the Abingdon Cross benchmark, defined at the 1982 Multicomputer Workshop in Abingdon, England. In that benchmark, an input image consists of a dark background with a pair of brighter rectangular bars, equal in

size, that cross at their midpoints, and are centered in the image. Gaussian noise is added to the entire image. The goal of the exercise is to determine and draw the medial axis of the cross formed by the two bars. The results obtained from solving the benchmark problem on various machines were presented by Preston [7, 8] at the 1984 Multicomputer Workshop in Tanque Verde, Arizona, and many of the participants spent a fairly lengthy session discussing problems with the benchmark and designing a new benchmark that, it was hoped, would solve those problems.

One concern of the Tanque Verde group was that the Abingdon Cross lacks breadth, requiring a reasonably small repertoire of image processing operations to construct a solution. The second concern was that the specification did not constrain the *a priori* information that could be used to solve the problem. In theory, a valid solution is to simply draw the medial lines since their true positions are known. Although this was never done, there was argument over whether it was acceptable for a solution to make use of the fact that the bars were oriented horizontally and vertically in the image. A final concern was that no method was prescribed for solving the problem, with the result that every solution was based on a different method. When a benchmark can be solved in different ways, the performance measurements are difficult to compare because they include an element of programmer cleverness. Also, the use of a consistent method would permit some comparison of the basic operations that make up a complete solution.

The Tanque Verde group specified a new benchmark, called the Tanque Verde Suite, that consisted of a large collection of individual vision-related problems (Table I). Each of the problems was to be further defined by a member of the group, who would also generate test data for their assigned problem. Unfortunately, only a few of the problems were ever developed, and none of them were widely tested on different architectures. Thus, while the simplicity of the Abingdon Cross may have been criticized, it was the respondent complexity of the Tanque Verde Suite that inhibited the latter's use.

Table I: Tanque Verde Benchmark Suite

Standard Utilities	High Level Tasks
3x3 Separable Convolution	Edge Finding
3x3 General Convolution	Line Finding
15x15 Separable Convolution	Corner Finding
15x15 General Convolution	Noise Removal
Affine Transform	Generalized Abingdon Cross
Discrete Fourier Transform	Segmentation
3x3 Median Filter	Line Parameter Extraction
256 Bin Histogram	Deblurring
Subtract Two Images	Classification
Arctangent(Image1/Image2)	Printed Circuit Inspection
Hough Transform	Stereo Image Matching
Euclidean Distance Transform	Camera Motion Estimation
	Shape Identification

In 1986, a new benchmark was developed at the request of the Defense Advanced Research Projects Agency (DARPA). Like the Tanque Verde Suite, it was a collection of vision-related problems, but was much smaller and easier to implement (Table II). A workshop was held in Washington, D.C., in November of 1986 to present the results of testing the benchmark on several machines, with those results summarized by Rosenfeld in [9]. The consensus of participants was that the results cannot be compared directly for several reasons. First, as with the Abingdon Cross, no method was specified for solving any of the problems. Thus, in many cases, the timings were more indicative of the knowledge or cleverness of the programmer, than of a machine's true capabilities. Second, no input data was provided and the specifications allowed a wide range of possible inputs. Thus, some participants chose to test a worst-case input, while others chose "average" input values that varied considerably in difficulty.

Table II: Tasks from the First DARPA Image Understanding Benchmark

11x11 Gaussian Convolution of a 512x512 8-bit Image
Detection of Zero Crossings in a Difference of Gaussians Image
Construct and Output Border Pixel List
Label Connected Components in a Binary Image
Hough Transform of a Binary Image
Convex Hull of 1000 Points in 2-D Real Space
Voronoi Diagram of 1000 Points in 2-D Real Space
Minimal Spanning Tree Across 1000 Points in 2-D Real Space
Visibility of Vertices for 1000 Triangles in 3-D 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

The workshop participants pointed out other shortcomings of the benchmark. Chief among these was that it consisted of isolated tasks, and therefore did not measure performance related to the interactions between the components of a vision system. For example, there might be a particularly fast solution to a problem on a given architecture if the input data is arranged in a special manner. However, this apparent advantage might be inconsequential if a vision system does not normally use the data in such an arrangement, and the cost of rearranging the data is high. Another shortcoming was that the problems had not been solved before they were distributed. Thus, there was no canonical solution on which the participants could rely for a definition of correctness, and there was even one problem (graph isomorphism) for which it turned out there was no practical solution.

Having a known correct solution is essential, since it is difficult to compare the performance of architectures that produce different results. For example, suppose architecture A performs a task in half the time of B, but A uses integer arithmetic while B uses floating point, and they obtain different results. Is A really twice as powerful as B? Since problems in vision are often ill-defined, it is possible to argue for the correctness of many different solutions. In a benchmark, however, the goal is not to solve a vision problem but to test the performance of different machines doing comparable work.

The conclusion from this first DARPA exercise was that a new benchmark should be developed. Specifically, the new benchmark should test system performance on a task that approximates an integrated solution to a machine vision problem. A complete solution with test data sets should be constructed and distributed with the benchmark specification. And, the benchmark should be specified to minimize opportunities for taking shortcuts in solving the problem. The task of constructing the new benchmark was assigned to the vision research groups at the University of Massachusetts at Amherst and the University of Maryland.

A preliminary specification was drawn up and circulated among the DARPA image understanding community for comment. The specification was revised, and a solution programmed on a standard sequential machine. In creating the solution, several problems were discovered that required corrections to the specification. The solution was programmed by the University of Massachusetts group and the

University of Maryland group then verified its validity, portability, and quality. Maryland also reviewed the solution for generality and neutrality with respect to underlying architectural assumptions. The Massachusetts group developed five test data sets, and a sample parallel solution for a commercial multiprocessor (the Sequent Symmetry 81).

In March of 1988, the benchmark was made available from Maryland via network access, or on tape from Massachusetts. The benchmark release consisted of the sequential and parallel solutions, the five data sets, and software for generating additional test data. The benchmark specification was presented by Weems at the DARPA Image Understanding Workshop, the International Supercomputing Conference, and the Computer Vision and Pattern Recognition conference [11, 12, 13]. Over 25 academic and industrial groups (Table III) obtained copies of the benchmark release. Nine of those groups developed either complete or partial versions of the solution for an architecture. A workshop was held in October of 1988, in Avon Old Farms, Connecticut, to present those results to members of the DARPA research community. As with the previous workshops, the participants spent a session developing a critique of the benchmark and making recommendations for the design of the next version.

Table III: Distribution List for the Second DARPA Benchmark
*** Indicates Results Presented at the Avon Workshop**

International Parallel Machines	Hughes AI Center
Mercury Computer Systems	University of Wisconsin
Stellar Computer	George Washington University
Myrias Computer	University of Massachusetts*
Active Memory Technology	SAIC
Thinking Machines*	Eastman Kodak
Aspex Ltd.*	University College London
Texas Instruments	Encore Computer
IBM	MIT
Carnegie-Mellon University*	University of Rochester
Intel Scientific Computers*	University of Illinois*
Cray Research	University of Texas at Austin*
Sequent Computer Systems*	Alliant Computer*

The remainder of this paper begins with a brief review of the benchmark task and the rationale behind its design, then it summarizes results that were based on hardware execution or on instruction-level simulation of the benchmark. Myung Sunwoo [10] from the University of Texas at Austin and Alok Coudhary [2] from the University of Illinois also presented estimated results for proposed architectures, which are not included here. Also not included are timings from Active Memory Technology on its DAP array processor that are for a set of independent image processing tasks only somewhat related to the benchmark problem. Finally, the paper presents the criticisms raised at the workshop, along with recommendations for addressing them.

Benchmark Task Overview

The overall task that is to be performed by this benchmark is the recognition of an approximately specified $2\frac{1}{2}$ dimensional "mobile" sculpture in a cluttered environment, given images from intensity and range sensors. The intention of the benchmark designers is that neither of the input images, by itself, is sufficient to complete the task.

The sculpture to be recognized is a collection of two-dimensional rectangles of various sizes, brightnesses, two-dimensional orientations, and depths. Each rectangle is oriented normal to the Z axis (the viewing axis), with constant depth across its surface, and the images are constructed under orthographic projection. Thus an individual rectangle has no intrinsic depth component, but depth is a factor in the spatial relationships between rectangles. Hence the notion that the sculpture is $2\frac{1}{2}$ dimensional.

The clutter in the scene consists of additional rectangles, with sizes, brightnesses, two-dimensional orientations, and depths that are similar to those of the sculpture. Rectangles may partially or completely occlude other rectangles. It is also possible for a rectangle to disappear when another of the same brightness or slightly greater depth (such that the difference in depth is less than the noise threshold) is located directly behind it.

A set of models is provided representing a collection of similar sculptures, and the task is to identify which model best matches the scene. The models are only approximate representations in that they permit

variations in the sizes, orientations, depths, and spatial relationships between the component rectangles. A model is a tree structure where the links represent the invisible links in the sculpture. Each node of the tree contains depth, size, orientation, and intensity information for a single rectangle. The child links of a node describe the spatial relationships between it and nodes below.

The scenario that was imagined in constructing the problem was a semi-rigid "mobile", with invisible links, viewed from above, with portions of other mobiles blowing through the scene. The initial state is that previous processing has narrowed the range of potential matches to a few similar sculptures, and has oriented them to match a previous image. However, the objects have since moved, and new images have been taken before completing the matching process. The system must choose the best match, and update the corresponding model with the positional information extracted.

The intensity and depth sensors are precisely registered with each other and both have a resolution of 512 x 512 pixels. There is no averaging or aliasing in either of the sensors. A pixel in the intensity image is an 8-bit integer grey value. In the depth image a pixel is a 32-bit floating-point range value. The intensity image is noise free, while the depth image has added Gaussian noise. The reason that only one of the images is noisy is that adding noise to the other image simply requires more of the same sorts of processing to be performed, and one goal of the benchmark designers was to maximize the variety of processing while minimizing programmer effort.

A pair of artificial test images is created by first selecting one model. The model is then rotated and translated as a whole, and its individual elements are perturbed slightly. Next, a collection of spurious rectangles is created with properties similar to those in the chosen model. All of the rectangles (both model and spurious) are then ordered by depth and drawn in the two image arrays. Lastly, an array of Gaussian-distributed noise is added to the depth image.

Figure 1 shows an intensity image of a mobile alone, and Figure 2 shows the mobile with added clutter. Depth images are not shown, because their floating point representation makes them difficult to display accurately.

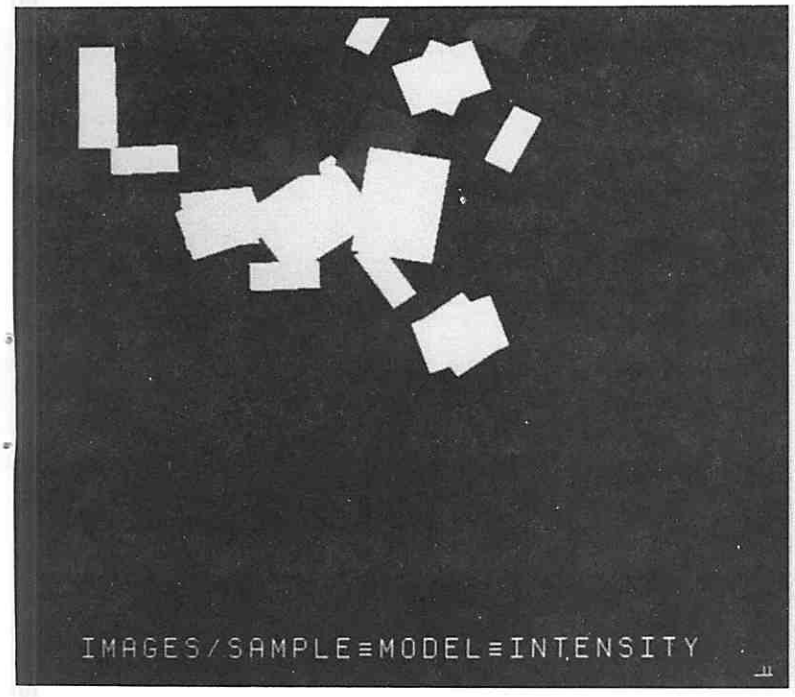

Figure 1: Intensity Image of Sample Model Alone

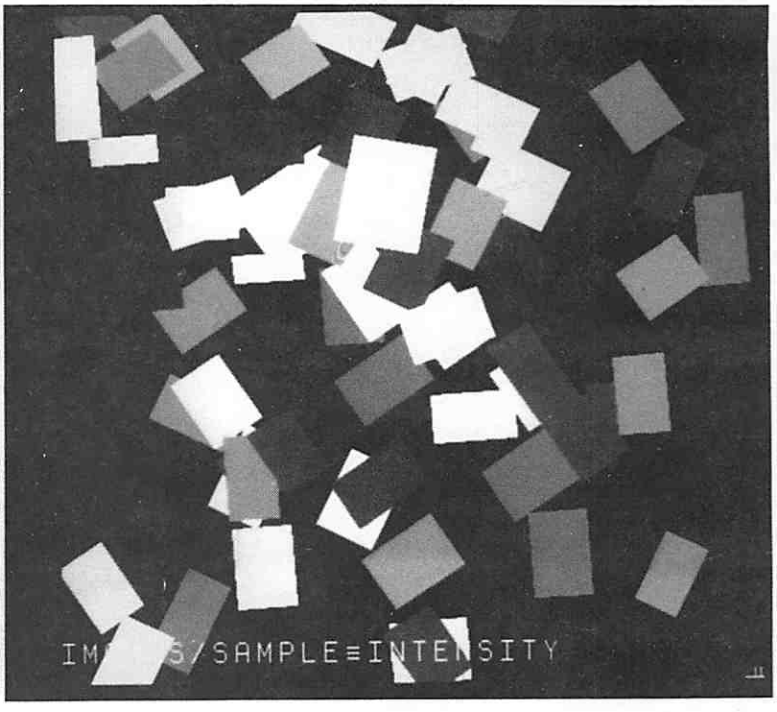

Figure 2: Image of Sample Model with Clutter

Processing begins with low-level operations on the intensity and depth images, followed by grouping operations on the intensity data to extract candidate rectangles. The candidates are used to form partial matches with the stored models. For each model, it is possible that multiple hypothetical poses will be established. For each model pose, stored information is used to probe the depth and intensity images in a top-down manner. Each probe tests a hypothesis for the existence of a rectangle in a given location in the images. Rejection of a hypothesis, which only occurs when there is strong evidence that a rectangle is actually absent, results in elimination of the corresponding model 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 pose with new size, orientation, and position information. The match strength is zero when there is no supporting evidence for the match and no evidence that the rectangle is absent, as in the case of a rectangle that is entirely occluded by another. After a probe has been performed for every unmatched rectangle in the list of model poses, an average match strength is computed for each pose that has not been eliminated. The model pose with the highest average is selected as the best match, and an image is generated that highlights the model in the intensity image. Table IV lists the steps that make up the complete benchmark task.

The benchmark specification requires these steps to be applied in implementing a solution. Furthermore, a recommended method for each step is described and should be followed if possible. However, in recognition that some methods do not work, or are extremely inefficient for a given parallel architecture, implementors are permitted to substitute other methods for individual steps. When it is necessary to differ from the specification, the implementor should supply a justification for the change. It is also urged that, if possible, a version of the implementation be written and tested with the recommended method so that the difference in performance can be determined.

Benchmark Philosophy and Rationale

In writing an integrated image understanding benchmark, the goal is to create a scenario that is an approximation of an actual image interpretation task. One must remember, however, that the benchmark problem is not an end in itself, but a framework for testing machine performance on a wide variety of

common vision operations and algorithms, both individually and in an integrated form that requires communication and control across algorithms and representations. This benchmark is not intended to be a challenging vision research exercise, and the designers feel that it should not be. Instead, it should be a balance between simplicity for the sake of implementation by participants, and the complexity that is representative of actual vision processing. At the same time, it must test machine performance in as many ways as possible. A further constraint on the design was the requirement that it re-use tasks from the first DARPA benchmark where possible, in order to take advantage of the previous programming effort. The great variety of architectures to be tested is itself a complicating factor in the design of a benchmark. It was recognized that each architecture may have its own most efficient method for computing a given function.

The job of the designers was thus to balance these conflicting goals and constraints in developing the benchmark. One result is that the solution is neither the most direct, nor the most efficient method. However, a direct solution would eliminate several algorithms that are important in testing certain aspects of machine performance. On the other hand, increasing the complexity of the problem to necessitate the use of those algorithms would require significant additional processing that is redundant in terms of performance evaluation. Thus, while the benchmark solution is not a good example of how to build an efficient vision system, it is an effective test of machine performance both on a wide variety of individual operations and on an integrated task. Participants were encouraged to develop timings for more optimal solutions, in addition to the standard solution, if they so desired.

Table IV: Steps that Compose the Integrated Image Understanding Benchmark

Low-Level, Bottom-Up Processing	
Intensity Image	Depth Image
Label Connected Components	3x3 Median Filter*
Compute K-Curvature	3x3 Sobel and Gradient Magnitude*
Extract Corners	Threshold*
Intermediate Level Processing	
Select Components with 3 or More Corners	
Convex Hull of Corners for Each Component	
Compute Angles Between Successive Corners on Convex Hulls	
Select Corners with K-Curvature and Computed Angles Indicating a Right Angle	
Label Components with 3 Contiguous Right Angles as Candidate Rectangles	
Compute Size, Orientation, Position, and Intensity for Each Candidate Rectangle	
Model-Based, Top-Down Processing	
Determine all Single Node Isomorphisms of Candidate Rectangles in Stored Models	
Create a List of all Potential Model Poses	
Perform a Match Strength Probe for all Single Node Isomorphisms (see below)*	
Link Together all Single Node Isomorphisms	
Create a List of all Probes Required to Extend Each Partial Match	
Order the Probe List According to the Match Strength of the Partial Match Being Extended	
Perform a Probe of the Depth Data for Each Probe on the List (see below)	
Perform a Match Strength Probe for Each Confirming Depth Probe (see below)*	
Update Rectangle Parameters in the Stored Model for Each Confirming Probe	
Propagate the Veto from a Rejecting Depth Probe Throughout the Corresponding Partial Match	
When No Probes Remain, Compute Average Match Strength for Each Remaining Model Pose	

Select Model with Highest Average Match Strength as the Best Match
Create the Output Intensity Image, Showing the Matching Model
Depth Probe
Select an X-Y Oriented Window in the Depth Data that will Contain the Rectangle
Perform a Hough Transform Within the Window
Search the Hough Array for Strong Edges with the Approximate Expected Orientations
If Fewer than 3 Edges are Found, Return the Original Model Data with a No-Match Flag
If 3 Edges are Found, Infer the Fourth from the Model Data
Compute new Size, Position, and Orientation Values for the Rectangle
Match-Strength Probe
Select an Oriented Window in the Depth Data that is Slightly Larger than the Rectangle
Classify Depth Pixels as Too Close, Too Far, or In Range*
If the Number of Too Far Pixels Exceeds a Threshold, Return a Veto
Otherwise, Select a Corresponding Window in the Intensity Image
Select Intensity Pixels with the Correct Value
Compute a Match Strength Based on the Number of Correct vs. Incorrect Pixels in the Images
* indicates subtasks involving floating-point operations

The designers also recognize the tendency for any benchmark to turn into a horse race. However, that is not the goal of this exercise, which is to increase the scientific insight of architects and vision researchers into the architectural requirements for knowledge-based image interpretation. To this end, the benchmark requires an extensive set of instrumentation. Participants are required to report execution time for individual tasks, for the entire task, for system overhead, input and output, system initialization and loading any precomputed data. and for different processor configurations if possible. Implementation factors to be reported include an estimate of time spent implementing the benchmark, the number of lines of source code, the programming language, and the size of the object code. Machine configuration and technology factors that are requested include the number of processors, memory capacity, data path widths, integration technology, clock and instruction rates, power consumption, physical size and weight, cost, and any limits to scaling-up the architecture. Lastly, participants are asked to comment on any changes to the architecture that they feel would contribute to an improvement in performance on the benchmark.

Results and Analysis

Due to limitations of time and resources, only a few of the original participants were able to complete the entire benchmark exercise and test it on all five of the data sets. In almost every case, there was some disclaimer to the effect that a particular architecture could have shown better performance given more implementation time or resources. It was common for participants to underestimate the effort required to implement the benchmark, and several who had said they would provide timings were unable to complete even a portion of the task prior to the workshop.

Caution on Comparison of Results

Care must be taken in comparing these results. For example, no direct comparison should be made between results obtained from actual execution and those derived from simulation as noted by Carpenter [1]. No matter how carefully a simulation is carried out, it is never as accurate as direct execution. Likewise, no comparison should be made between results from partial and complete implementations. A complete implementation includes overhead for the interactions between subtasks, and for the fact that the

program is significantly larger than a partial implementation. Consider that individual subtasks might be faster than in a complete implementation simply because less paging is required. It is also unwise to directly compare raw timings, even for similar architectures, without considering the differences in technology between systems. For example, a system that executes the benchmark faster than another is not necessarily architecturally superior if it also has a faster clock rate or more processors.

In addition to technical problems in making direct comparisons, there are other considerations to keep in mind. For example, what is impressive in many cases is not the raw speed obtained, but the speed with respect to the effort required to obtain it. While this has more to do with the software tools available for an architecture, it is still important in evaluating the overall usefulness of a system. Another consideration is the ratio of cost to performance. In addition, the size or weight or power consumption may be of greater importance than all-out speed in some applications. Finally, each vision application has a different mix of bottom-up and top-down processing, which is unlikely to match the mix used in the benchmark. Thus, readers should not focus on the total time, but may find it more useful to combine timings for the subtasks to approximate the processing mix in some familiar application. One of the purposes of this exercise is merely to assemble as much data as possible so that the performance results can be evaluated with respect to the requirements of each potential application.

The Data Sets

Five data sets were distributed with the benchmark, having the unimaginative names of Sample, Test, Test2, Test3, and Test4. The Sample data set required the greatest processing time on the sequential processors, and was in some ways the most complex. It had the greatest density of model elements (both large and numerous), and enough similarity between models to require a significant amount of top-down processing to determine the best match. Sample also required that a 5 x 5 median filter be used, rather than the 3 x 3 that was specified for the other data sets (this was intended to necessate a certain level of generality in the median filter routine, and also to see if an architecture had special hardware for 3 x 3 window operators that might not extend to larger window sizes).

While Sample was intended to represent a processing balance between the bottom-up and top-down portions, the remaining data sets were somewhat biased toward one or the other of those portions. The Test and Test2 data sets de-emphasized the top-down processing by having only one model that fit the images. It was possible to quickly reject all of the other models and simply use the top-down probes to determine the new positional information for the one remaining model. The Test3 and Test4 data sets emphasized the top-down portion by presenting several models that were nearly identical, and which had considerable symmetry so that numerous poses would be hypothesized. Thus, there were several models that could not be eliminated, and a far greater number of top-down probes were required to determine the best match.

Figures 1 and 2, above, show the intensity images for the Sample model and input image data. Figures 3 and 4 show the model and intensity images for the Test data set (which is similar to Test2), and figures 5 and 6 show the Test3 data set (which is similar to Test4).

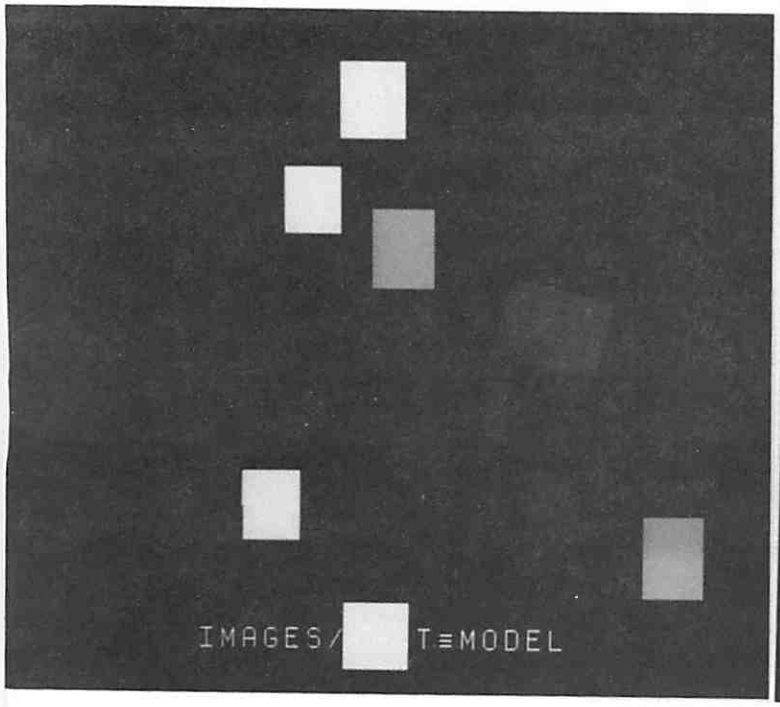

Figure 3: Intensity Image of Model Test

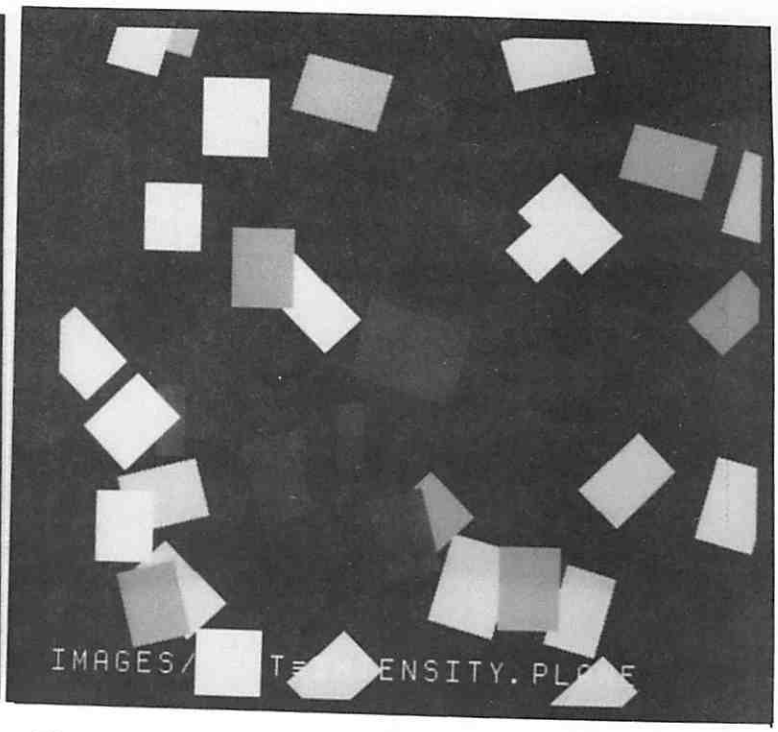

Figure 4: Image of Model Test with Clutter

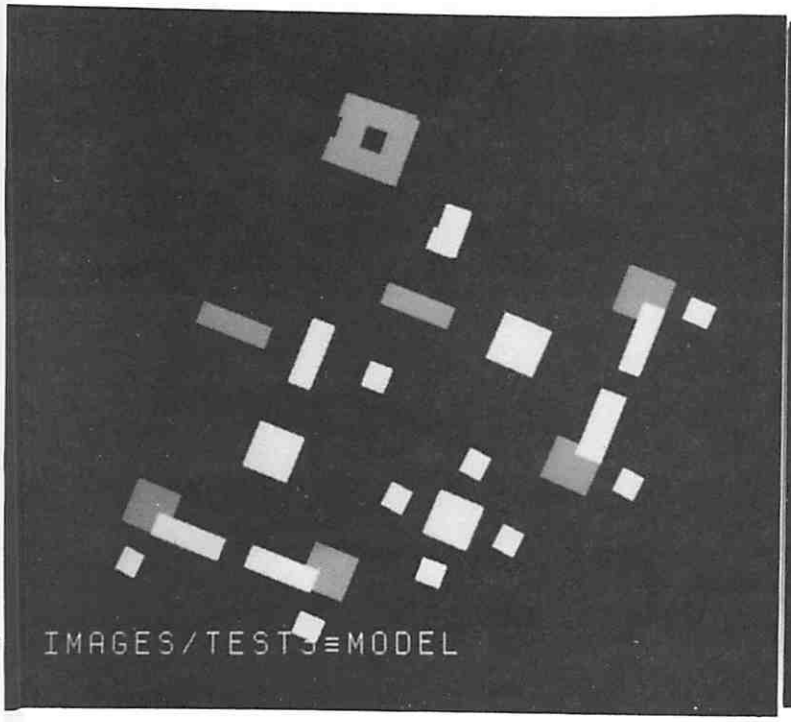

Figure 5: Intensity Image of Model Test3

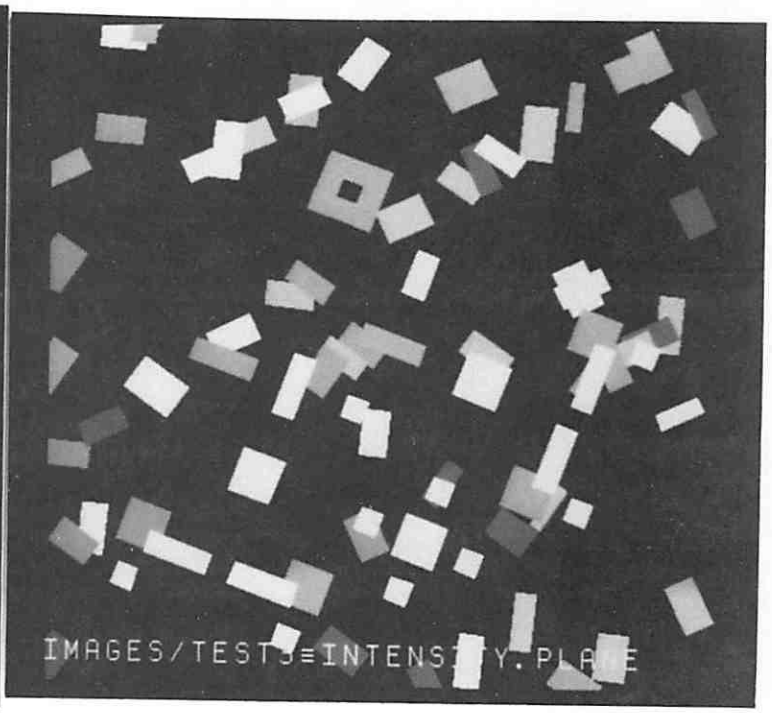

Figure 6: Image of Model Test3 with Clutter

Reporting Conventions

To set the context for the results, we first describe each of the implementations. Results based on theoretical estimations are not included here. Because of the variation in the actual timings and implementation information that was supplied, the data has been rearranged in a standard format, which specifies the timings only for the major subtasks. All timings have been scaled to seconds, even though for some of the processors they would be more readable if presented in milliseconds, or minutes. A detailed presentation of all of the data for the minor subtasks can be found in [15]. The details of the architectures can be found in the appropriate references. Physical and cost data for the commercial systems are subject to change, and should be obtained directly from the manufacturers.

In addition to the timings, the specification requested that a set of intermediate results be output to help verify that the subtasks were performing comparable operations. For example, the number of connected components in the intensity image, and the number of probes performed were among the requested validation results. It was not possible for every system to generate all of this data, but whatever validation results were provided are included here.

From an architectural point of view, one useful measure is the percentage of time devoted to each subtask, which indicates the subtask's relative difficulty for a particular architecture. It also factors out all of the technological issues, and provides one of the few measures that can be directly compared across architectures. For each implementation, we present a bar chart, showing the percentage of time spent on the major subtasks, for each data set. In several cases, where data was available for different machine configurations, the configurations are also compared. The different architectures are compared in a later section.

Sequential Solution

The sequential implementation was developed in C on a Sun-3/160 workstation, and contains roughly 4600 lines of code, including comments. It was designed for portability and has been recompiled on several different systems. The only system dependent portion is the result presentation step, which uses

the workstation's graphics display. The implementation differs from the recommended method on the Connected Component Labelling step by using a standard sequential method for this well-defined function. The sequential method minimizes array accesses and the corresponding index calculations, which incur an avoidable time penalty on a sequential machine.

Timings have been produced for all five data sets, and on three different machine configurations: a Sun-3/160 (a 16 MHz 68020 processor) with 8MB of RAM, a Sun-3/260 (a 25 MHz 68020) with 16MB of RAM, and a Sun-4/260 (a 16MHz SPARC processor) with 16MB of RAM. The extra RAM on the latter two machines did not affect performance, since the benchmark runs in 8MB without paging. The 3/260 was equipped with a Weitek floating-point co-processor, while the 3/160 and 4/260 used only their standard co-processors. Tables V, VI, and VIII show the execution times for the Sun-3/160, Sun-3/260, and Sun-4/260, respectively. Tables VII and IX show the validation data that was output by the Sun-3 systems, and the Sun-4, respectively. Notice the slight variation in the validation data, due to minor differences in the floating-point results. These variations are within the tolerances of the benchmark, and the final result is the same. The timings were obtained with the system clock utility which has a resolution of 20 milliseconds on the Sun-3 systems, and 10 milliseconds on the Sun-4.

Table V: Sun-3/160 Times for Major Subtasks

Data Set	Sample	Test	Test2	Test3	Test4
Total	797.88	338.06	329.24	551.82	553.16
Overhead	5.08	4.94	5.64	5.64	5.52
Label connected components	27.78	27.82	28.40	28.22	28.24
Rectangles from intensity	6.50	4.14	4.38	5.44	5.34
Median filter	246.66	118.88	92.86	90.92	90.90
Sobel	135.48	133.30	136.10	135.28	135.42
Initial graph match	24.46	25.00	26.04	68.44	67.62
Match strength probes	72.98	3.28	5.86	47.88	42.06
Hough probes	253.70	8.28	12.96	153.98	162.34
Result presentation	24.80	12.32	16.66	14.78	14.76

Table VI: Sun-3/260 Times for Major Subtasks

Data Set	Sample	Test	Test2	Test3	Test4
Total	299.38	132.54	119.52	194.76	195.58
Overhead	2.92	3.04	3.44	3.44	3.44
Label connected components	14.52	14.46	14.46	14.58	14.66
Rectangles from intensity	3.74	2.38	2.48	3.16	2.98
Median filter	113.70	60.28	43.10	42.98	43.26
Sobel	41.00	38.50	38.34	38.76	38.56
Initial graph match	6.16	6.08	6.58	17.32	16.94
Match strength probes	17.56	0.78	1.40	11.66	10.30
Hough probes	92.70	3.12	4.82	57.96	60.44
Result presentation	6.68	3.64	4.72	4.50	4.30

Table VII: Sun-3/160 and 3/260 Validation Output (identical results for both configurations)

Data Set	Sample	Test	Test2	Test3	Test4
Connected components	134	35	34	114	100
Right angles extracted	126	99	92	210	197
Rectangles detected	25	21	16	42	39
Depth pixels > threshold	21256	14542	12898	18584	18825
Elements on initial probe list	381	19	27	400	249
Hough probes	55	3	5	97	93
Initial match strength probes	28	20	15	142	142
Extension mat. str. probes	60	3	5	110	97
Models remaining	2	1	1	2	1
Model selected	10	1	5	7	8
Average match strength	0.64	0.96	0.94	0.84	0.88
Translated to	151,240	256,256	257,255	257,255	257.255
Rotated by (degrees)	85	359	114	22	22

Table VIII: Sun-4/260 Times for Major Subtasks

Data Set	Sample	Test	Test2	Test3	Test4
Total	121.01	42.64	40.94	81.05	82.84
Overhead	4.34	3.92	3.79	4.08	4.11
Label connected components	4.74	4.56	4.54	4.62	4.61
Rectangles from intensity	1.10	0.68	0.71	0.96	0.91
Median filter	30.53	14.64	11.30	11.30	11.34
Sobel	12.16	11.43	11.27	11.41	11.45
Initial graph match	3.42	3.46	3.54	10.10	9.94
Match strength probes	9.91	0.45	0.79	6.65	6.08
Hough probes	51.20	1.73	2.64	29.52	31.99
Result presentation	3.38	1.67	2.24	2.07	2.02

Table IX: Sun-4/260 Validation Output

Data Set	Sample	Test	Test2	Test3	Test4
Connected components	134	35	34	114	100
Right angles extracted	126	99	92	210	197
Rectangles detected	25	21	16	42	39
Depth pixels > threshold	21254	14531	12892	18579	18822
Elements on initial probe list	381	19	27	389	248
Hough probes	55	3	5	93	92
Initial match strength probes	28	20	15	142	142
Extension mat. str. probes	60	3	5	105	97
Models remaining	2	1	1	2	1
Model selected	10	1	5	7	8
Average match strength	0.64	0.96	0.94	0.84	0.88
Translated to	151,240	256,256	257,255	257,255	257,255
Rotated by (degrees)	85	359	114	22	22

Figure 7 compares the three configurations on each of the data sets, with regard to the percentage of time spent on each major subtask. The key identifies the pattern associated with each major subtask. Note that the bottom-up portions are represented by shading, while the top-down portions are shown by cross-hatch patterns. The figure shows that Test and Test2 require far less top-down processing than the other three data sets. Closer examination reveals several interesting points. For example, despite the faster Weitek co-processor, the Sun-3/260 spends proportionately more time than the Sun-3/160 in the median filter, which involves floating point data. It is also interesting to note that the Sun-4 spends a larger percentage of the time on the top-down tasks (especially the Hough probes) and overhead. Since the overhead depends mostly on disk access time, it should be expected to increase in percentage as the total time decreases.

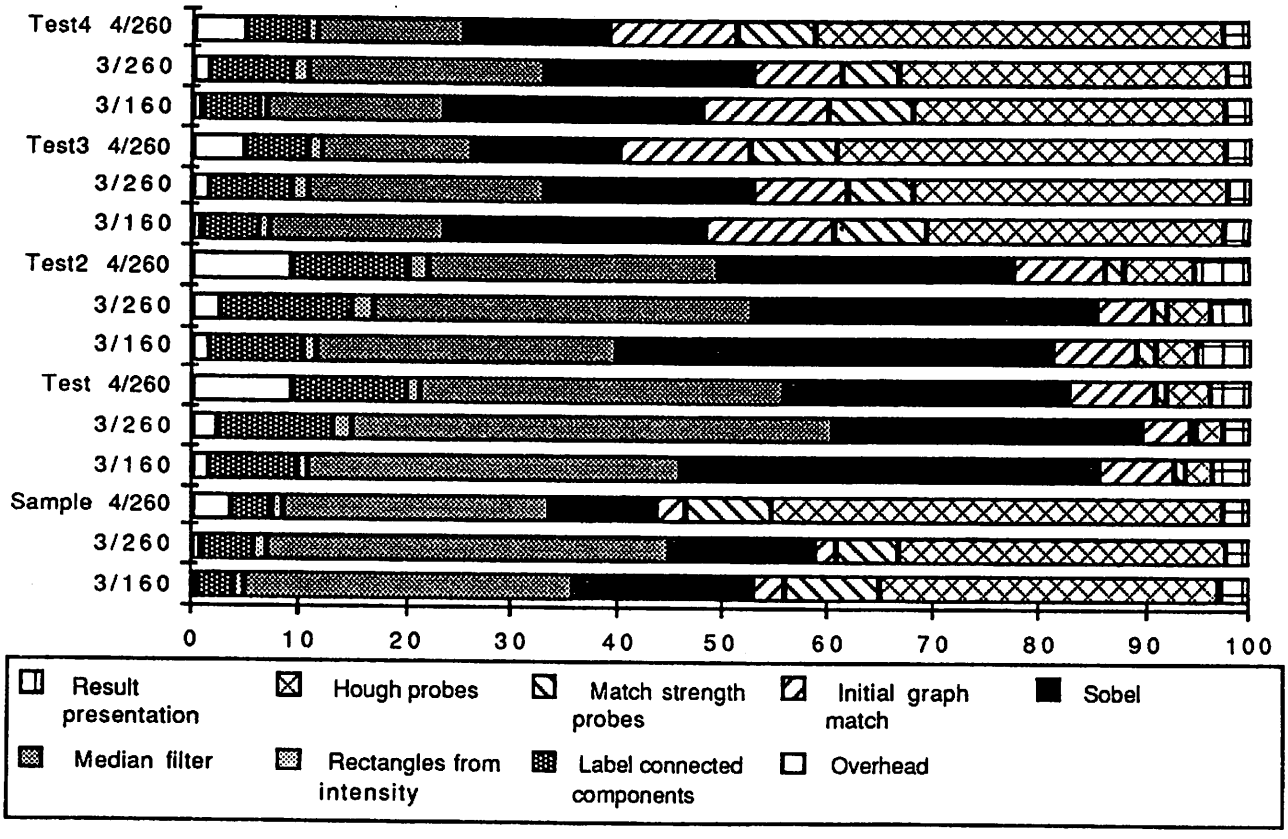

Figure 7: Sun Workstation Percent of Effort for Each Major Subtask

Alliant FX-80 Solution

The Alliant FX-80 consists of up to eight computational elements and up to twelve I/O processors that share a physical memory through a sophisticated combination of caches, busses and an interconnection network. The computational elements communicate with the shared memory via the interconnection network which links them to a pair of special purpose caches that in turn access the memory over a bus that is shared with the I/O processor caches.

Alliant was able to implement the benchmark on the FX-80 in roughly one programmer-week. The programmer had no experience in vision and, in many cases, did not bother to learn how the benchmark code works. The implementation was done by rewriting the system dependent section to use the available graphics hardware, compiling the code with Alliant's vectorizing and globally optimizing C compiler, using a profiling tool to determine the portions of the code that used the greatest percentage of CPU time, inserting compiler directives in the form of comments to break implicit dependencies in four sections of the benchmark, and recompiling. Alliant provided results for five configurations of the FX-80, with 1, 2, 4, 6, and 8 computational elements. To save space, only two of the configurations are presented here. Table X shows the results for a single FX-80 computational element, and Table XI shows an FX-80 with eight elements. Table XII shows the validation output from the Alliant, which was identical for all configurations. The validation output shows a small variation from the Sun-3, but this is due to differences in the floating-point calculations that are within acceptable limits, and produce the same final result. Alliant pointed out that the C compiler was a new product at that time and did not yet provide as much optimization as their FORTRAN compiler (a difference of up to 50% in some cases).

Table X: Alliant FX-80 Single Processor Times for Major Subtasks

Data Set	Sample	Test	Test2	Test3	Test4
Total	207.89	104.561	95.139	139.808	142.162
Overhead	8.744	8.702	8.672	8.664	8.658
Label connected components	17.185	17.088	17.053	17.195	17.189
Rectangles from intensity	3.350	2.058	2.126	2.993	2.929
Median filter	77.464	43.812	32.049	32.073	32.046
Sobel	26.148	26.080	26.064	26.129	26.130
Initial graph match	2.546	2.460	2.624	7.485	7.384
Match strength probes	7.235	0.316	0.576	4.768	4.371
Hough probes	60.956	1.901	3.312	37.631	40.632
Result presentation	3.271	1.862	2.390	2.179	2.176

Table XI: Alliant FX-80 Eight Processor Times for Major Subtasks

Data Set	Sample	Test	Test2	Test3	Test4
Total	60.112	33.138	32.915	53.952	53.620
Overhead	8.787	8.728	8.710	8.711	8.721
Label connected components	7.225	7.136	7.119	7.252	7.264
Rectangles from intensity	3.462	2.113	2.177	3.114	3.060
Median filter	10.113	5.857	4.323	4.324	4.318
Sobel	3.799	3.790	3.788	3.796	3.796
Initial graph match	2.578	2.493	2.655	7.615	7.473
Match strength probes	7.232	0.317	0.576	4.804	4.404
Hough probes	13.090	0.554	0.898	11.374	11.716
Result presentation	3.267	1.861	2.384	2.180	2.175

Table XII: Alliant FX-80 Validation Output

Statistics	Sample	Test	Test2	Test3	Test4
Connected components	134	35	34	114	100
Right angles extracted	126	99	92	210	197
Rectangles detected	25	21	16	42	39
Depth pixels > threshold	21266	14542	12888	18572	18813
Elements on initial probe list	374	19	27	389	248
Hough probes	55	3	5	93	92
Initial match strength probes	28	20	15	142	142
Extension mat. str. probes	60	3	5	105	97
Models remaining	2	1	1	2	.
Model selected	10	1	5	7	8
Average match strength	0.65	0.96	0.94	0.84	0.88
Translated to	151,240	256,256	257,255	257,255	257,255
Rotated by	85	359	114	22	22

Figure 8 compares relative time for the two FX-80 configurations. The parallel configuration achieves the greatest improvement on the floating-point operations (median filter and Sobel), the Hough probes, and labelling the connected components. These are the four subtasks that received special attention in optimizing the implementation. The proportional effort thus grew for overhead and the other tasks.

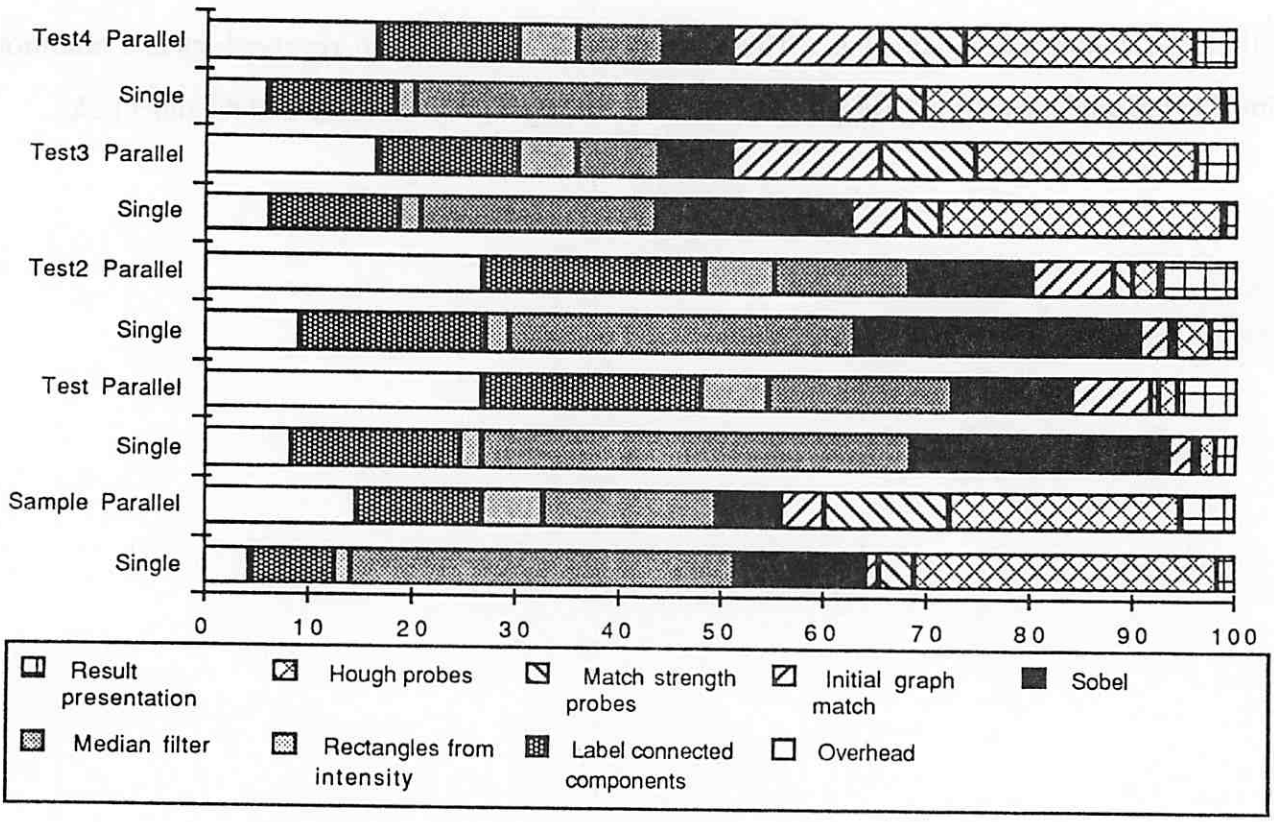

Figure 8: Alliant FX-80 Percent of Effort for Each Major Subtask

Image Understanding Architecture

The Image Understanding Architecture (IUA) is being built by the University of Massachusetts and Hughes Research Laboratories specifically to address the problem of supporting real-time, knowledge-based vision. The architecture consists of three different parallel processors, arranged in a hierarchy that is tightly coupled by layers of dual-ported memory between the processors. The low-level processor is a bit-serial, processor-per-pixel, SIMD, associative array. The intermediate-level processor is an SIMD/MIMD array of 4096 16-bit digital signal processors that communicate via an interconnection network. Each intermediate-level processor shares a dual-ported memory segment with 64 low-level processors. The high level is a multiprocessor intended to support AI processing and a blackboard model of communication through a global shared memory, which is dual-ported with a segment of the intermediate-level processor's memory. A detailed description of the architecture can be found in [14].

Because the architecture is under construction, an instruction-level simulator was used to develop the benchmark implementation. The simulator is programmed in a combination of Forth and an assembly language which has a syntax similar to Ada assignment statements. The benchmark was developed over a period of about six months, but much of that time was spent in building basic library routines and additional tools that were generally required for any large programming task. A 1/64th scale version of the simulator (4096 low-level, 64 intermediate-level, and one high-level processor) runs on a Sun workstation, and was used to develop the initial implementation. The implementation was then transported to a full-scale IUA simulator running on a Sequent Symmetry multiprocessor.

Table XIII presents the IUA results with a resolution of one instruction time (0.1 microsecond). There are several points to note. Because the processing of different steps can be overlapped in the different levels, the sum of the individual step timings does not equal the total time. Some of the individual timings are averages, since intermediate level processing takes place asynchronously and individual processes vary in their execution time. For example, the time for all of the match-strength probes is difficult to estimate since probes are created asynchronously and their processing is overlapped with each other and with other steps. However, the time for match extension includes the time to complete all of the subsidiary match-

strength probes. Thus, where the table would usually break the match extension step into separate times for match-strength and Hough probes, it shows the total time for match extension, and the average time for an individual probe.

Table XIV shows the validation output for the IUA. The number of elements on the initial probe list is not given because parallel tasks were used, and thus there is no single initial probe list. The number of probes varies from the sequential version because a somewhat more robust variation of the probe algorithm was used, which vetoed poses at different points in the matching process. Also, the separate processes shared their probe results so that a few duplicate probes were eliminated. The added robustness in the probe algorithm also lead to a slightly lower average match-strength.

Lastly, it should be mentioned that the intermediate-level processor was greatly underutilized by the benchmark (only 0.2% of its processors were activated), and the high-level processor was not used at all. The low-level processor was also idle roughly 50% of the time while awaiting requests for top-down probes from the intermediate level.

Table XIII: Image Understanding Architecture Simulator Times for Major Subtasks

Data Set	Sample	Test	Test2	Test3	Test4
Total	0.0844445	0.0455559	0.0455088	0.4180890	0.3978859
Overhead	0.0139435	0.0139435	0.0139435	0.0139435	0.0139435
Label connected components	0.0000596	0.0000596	0.0000596	0.0000596	0.0000596
Rectangles from intensity	0.0161694	0.0125489	0.0134704	0.0131378	0.0129635
Median filter	0.0005625	0.0005625	0.0005625	0.0005625	0.0005625
Sobel	0.0026919	0.0026919	0.0026919	0.0026919	0.0026919
Initial graph match	0.0155662	0.0153462	0.0135538	0.2953212	0.2356714
Match extension	0.0300650	0.0017674	0.0024856	0.0899214	0.1277396
Match strength probe (average)	0.0026500	0.0001146	0.0004095	0.0543250	0.0071766
Hough probe (average)	0.0068430	0.0003251	0.0005092	0.0084591	0.0109868
Result presentation	0.0022826	0.0009452	0.0011944	0.0029768	0.0029766

Table XIV: Image Understanding Architecture Validation Output

Statistics	Sample	Test	Test2	Test3	Test4
Connected components	134	35	34	114	100
Right angles extracted	163	106	100	262	250
Rectangles detected	31	23	19	60	55
Depth pixels > threshold	23185	13598	14065	19730	19753
Elements on initial probe list					
Hough probes	44	5	8	84	100
Initial match strength probes	24	20	15	81	80
Extension mat. str. probes	20	1	3	41	54
Models remaining	3	1	1	2	1
Model selected	10	1	5	7	8
Average match strength	0.45	0.86	0.84	0.81	0.84
Translated to	151,240	256,256	257,255	257,255	257,255
Rotated by	85	359	113	23	23

Figure 9 shows the relative time spent by the IUA on each major subtask, for each data set. The graph matching and match extension processes are clearly a dominant factor. Because task parallelism was used to match each model separately, the maximum obtainable parallelism was a factor of ten, versus a factor of over two hundred thousand for the bottom-up subtasks, which were done with data parallelism. In practise, the improvement due to task parallelism only equalled the number of models not vetoed during the matching process. However, the low utilization of the intermediate level processors permits the task parallel solution to operate on a set of several thousand models with about the same performance. The overhead for the IUA includes generating tables that are used in multiple places in the processing. Note that the time for labelling connected components is so small that it is invisible. For Test3 and Test4, the median filter is also too small to be visible.

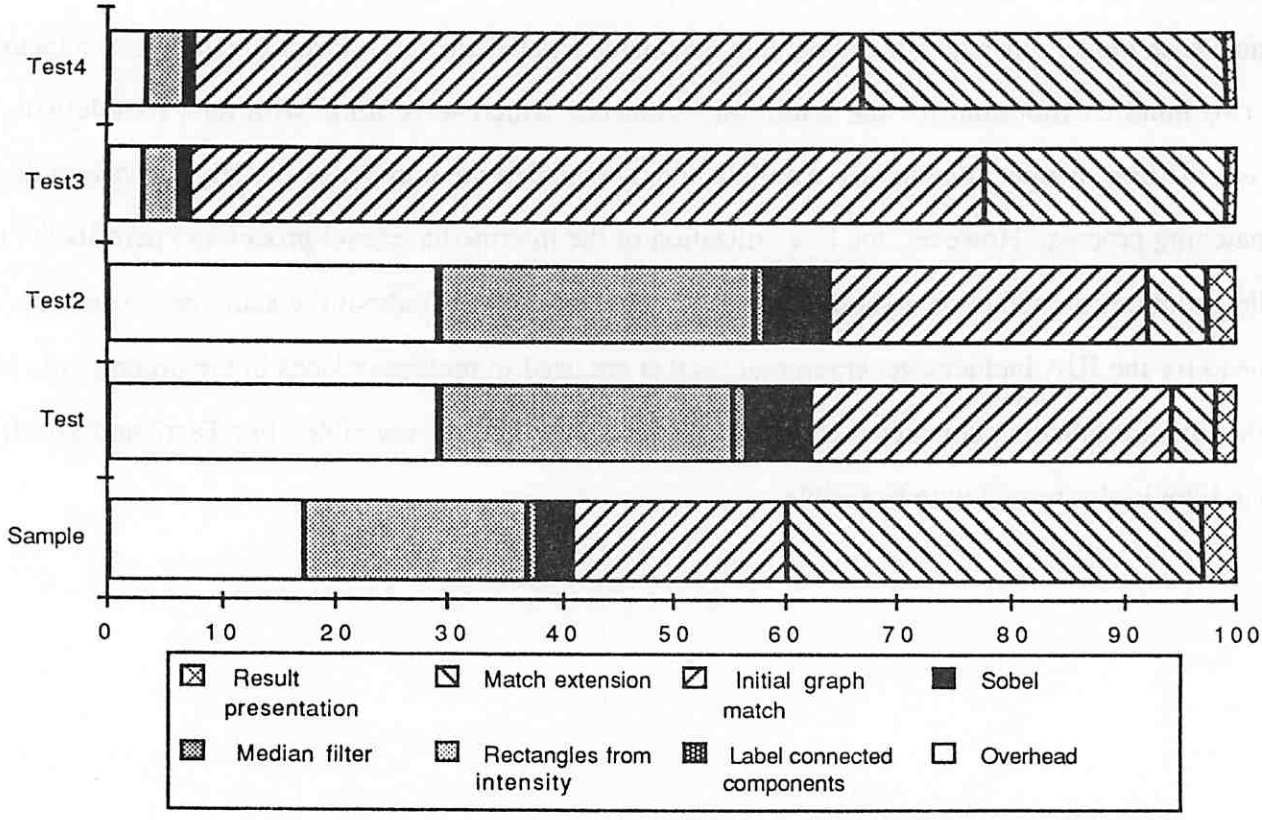

Figure 9: IUA Percent of Effort for Each Major Subtask

Aspex ASP

The Associative String Processor (ASP) is being built by R.M. Lea at the University of Brunel and Aspex Ltd. in England [6]. It is designed as a general purpose processing array for implementation in wafer-scale technology. The processor consists of 262,144 processors arranged as 512 strings of 512 processors each. Each processor contains a 96-bit data register and a 5-bit activity register. A string consists of 512 processors linked by a communication network that is also tied to a data exchanger and a vector data buffer. The vector data buffers of the strings are linked through another data exchanger and data buffer to another communication network. One of the advantages of this arrangement is a high degree of fault tolerance. The system can be built with 1024 VLSI devices, or 128 ULSI devices, or 32 WSI devices. Estimated power consumption is 650 watts. The processor clock and instruction rate is projected to be 20 MHz. Architectural changes that would improve the benchmark performance include increasing the number of processors (improves performance on K-curvature, median filter, and Sobel), increasing the speed of the processors and communication links (linear speedup on all tasks), and adding a separate controller to each ASP substring; resulting in approximately an 18% increase overall.

Because the system is under construction, a software simulator implementation was used. The benchmark was programmed in an extended version of Modula-2 over a period of three months by two programmers, following a three month period of initial study of the requirements and development of a solution strategy. A Jarvis' March algorithm was substituted for the recommended Graham Scan method on the convex hull. Table XV lists the major subtask times for the ASP. Timings were not provided for several of the minor steps in the model matching portion of the benchmark, because a different method was used. The time under overhead accounts for the input and output of several intermediate images. The time under the section that extracts rectangles from the intensity image accounts for the output and subsequent input of data records for corners and rectangles. The output and input of intermediate data was done to take advantage of the vector data buffers in the ASP which allow strings to be quickly transferred out and then be rebroadcast to the array. The implementors were thus able to cleverly recast the task parallel orientation of the model-matching process into a data parallel form by creating all of the different matching combinations, so that only a simple comparison was required to determine a match. However, for large

sets of models, this technique is likely to result in an excessive number of combinations. The use of a data parallel technique also makes it harder to compare the ASP with other systems which could have benefitted from that method. Table XVI shows the validation output, which is similar to the sequential output except for some variation in the floating-point results, and the absence of data for the number of elements on the initial probe list.

Table XV: ASP Simulator Times for Major Subtasks

Data Set	Sample	Test	Test2	Test3	Test4
Total	0.130720	0.035960	0.039810	0.113070	0.118820
Overhead	0.000820	0.000820	0.000800	0.000800	0.000800
Label connected components	0.039200	0.022800	0.022800	0.034800	0.031300
Rectangles from intensity	0.003310	0.002920	0.002880	0.003190	0.003350
Median filter	0.000720	0.000720	0.000510	0.000610	0.000510
Sobel	0.000624	0.000624	0.000624	0.000680	0.000624
Initial graph match	0.000009	0.000009	0.000009	0.000009	0.000008
Match strength probes	0.003300	0.000429	0.000388	0.005640	0.006430
Hough probes	0.080274	0.005497	0.010545	0.063354	0.071726
Result presentation	0.000850	0.000440	0.000470	0.000470	0.001030

Table XVI: Aspex ASP Validation Output

Statistics	Sample	Test	Test2	Test3	Test4
Connected components	133	34	33	113	99
Right angles extracted	126	99	92	210	197
Rectangles detected	25	21	16	42	39
Depth pixels > threshold	21255	14533	12891	18582	18817
Elements on initial probe list					
Hough probes	55	3	5	97	93
Initial match strength probes	28	20	15	142	142
Extension mat. str. probes	60	3	5	110	97
Models remaining	2	1	1	2	1
Model selected	10	1	5	7	8
Average match strength	0.64	0.96	0.93	0.84	0.87
Translated to	151,240	256,256	257,255	257,255	257,255
Rotated by	85	359	114	22	22

Figure 10 shows the percentage of time spent by the ASP on each major subtask. The time is dominated by labelling connected components, and performing Hough probes, which require significant amounts of communication between strings, which must pass through the local data exchangers and then through the data exchanger that connects the ends of the strings together. Because of the data parallel method that was used, the time for the initial graph match step is invisible in the figure.

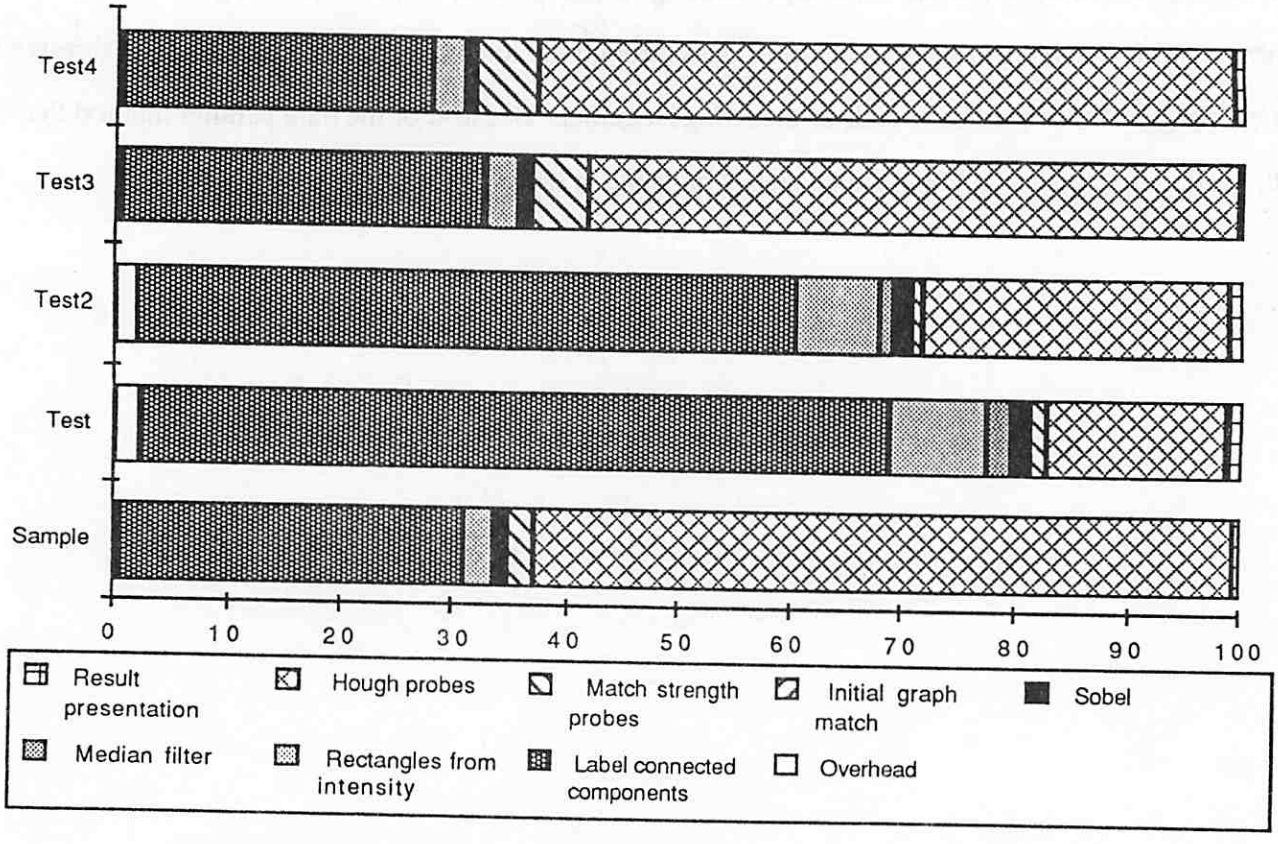

Figure 10: ASP Percent of Effort for Each Major Subtask

Sequent Symmetry 81

The Sequent Computer Systems Symmetry 81 multiprocessor consists of Intel 80386 processors, running at 16.5 MHz, connected via a shared bus to a shared memory. The configuration used to obtain these results included 12 processors (one of which is reserved by the system), each with an 80387 math coprocessor, and 96 MB of shared memory. The system also contained the older A-model caches, which induce a more traffic on the bus than the newer caches. The timings in Table XVII were obtained by the benchmark developers as part of the effort to ensure the portability of the benchmark.

About a month was spent developing the parallel implementation for the Sequent. The programmer was familiar with the benchmark, but had no previous experience with the Sequent system. Part of the development period was spent modifying the sequential version to enhance its portability. The low-level tasks were directly converted by dividing the data among the processors in a manner that avoided write-contention. About half of the development time was spent adding data locking mechanisms to the model-matching portion of the benchmark, and resolving problems with timing and race conditions. It was only possible to obtain timings for the major steps in the benchmark, because the Sequent operating system does not provide facilities for accurately timing individual child processes. The benchmark was run on configurations of from one to eleven processors, with the optimum time being obtained with eight or nine processors. Additional processors resulted in a reduction in performance, due to a combination of factors. As the data were divided among more processors, the ratio of processing time to task creation overhead decreased so that the latter came to dominate the time on some tasks. We also believe that some of the tasks reached the saturation point of the bus, since one run that was observed on a B-model cache system showed performance to improve with more processors. The table shows the performance for a single processor running the sequential version, to provide a comparison baseline, and the performance on the optimum number of processors for each data set.

Table XVII: Sequent Symmetry 81 Times for Major Subtasks

Data Set	Sample		Test		Test2		Test3		Test4	
Processors	Single	Eight	Single	Eight	Single	Nine	Single	Eight	Single	Nine
Total	889.66	251.33	300.34	73.88	282.71	77.87	562.15	174.96	578.14	139.72
Overhead	5.84	6.00	5.57	5.93	5.62	5.87	5.75	5.86	5.65	5.90
System time	3.60	9.40	2.00	5.40	2.10	6.40	2.80	7.60	2.90	8.80
Label conn. components	19.27	12.68	19.34	15.83	19.29	16.01	19.60	16.84	19.58	16.89
Rectangles from intensity	4.18	1.45	2.62	0.92	2.74	1.92	3.42	1.42	3.38	1.89
Median filter	239.24	31.00	114.12	15.25	85.81	11.08	85.83	11.45	85.79	11.11
Sobel	110.89	15.00	113.21	15.46	110.80	14.83	110.84	15.20	110.81	14.73
Initial graph match	18.52	3.08	18.53	3.76	19.90	4.35	52.53	7.21	51.63	7.17
Match extension	470.90	161.34	16.16	5.97	24.08	9.38	271.07	103.99	288.21	69.10
Result presentation	20.82	20.78	10.80	10.76	14.47	14.43	13.11	12.99	13.09	12.93

Figure 11 compares the relative time for a single processor versus the optimum number of processors, for each data set. As with the Alliant, it can be seen that the fixed overhead cost grows in proportion as the total time decreases. The best performance increase was obtained for the Sobel and median filter portions, because they involve sufficient processing to keep all of the processors busy. Labelling connected components, on the other hand, is less computationally intensive, and thus the process creation overhead is a significant portion of the time. The Test and Test2 data sets also show less than the average performance increase, since the models are assigned to separate processors for the matching process, and all but one of the models quickly drops out for those data sets. In the other sets, where matching can continue on multiple models in parallel, the improvement in performance is closer to the average. Result presentation suffers from the same problem as the overhead component, being dependent on sending an image to an external device.

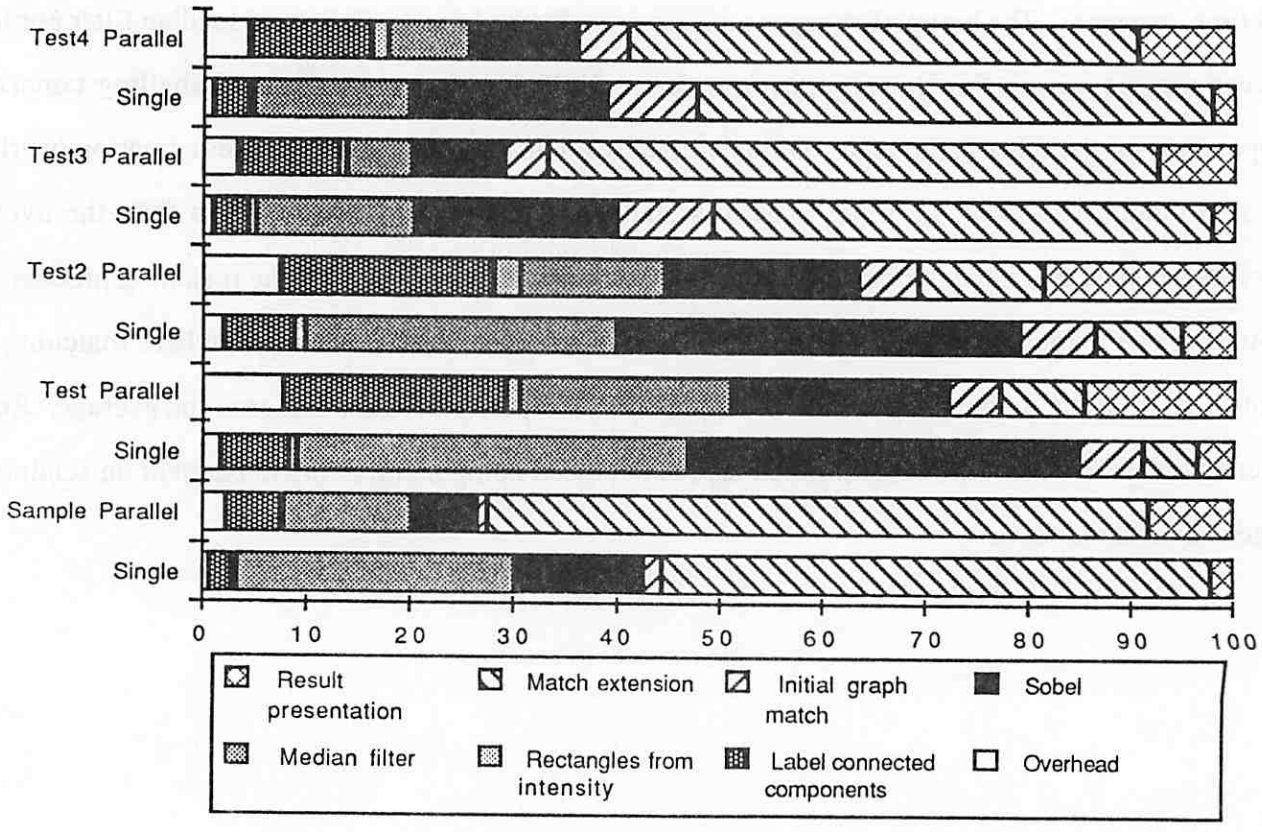

Figure 11: Sequent Symmetry 81 Percent of Effort for Each Major Subtask

Warp

The CMU Warp is a systolic array consisting of ten high speed floating point cells in a linear configuration [5]. Processing in the Warp is directed by a host processor, such as the Sun-3/60 workstation used in executing the benchmark. The implementation was programmed by one person in two weeks, using a combination of the original C implementation and subroutines written in Apply and W2. The objective was to obtain the best overall time, rather than the best time for each task. While it would seem that the latter guarantees the former, consider that the Warp and its host can work in parallel. Even though the Warp could perform a step in one second that requires four seconds on the host, it is better to let the host do the processing if it would otherwise sit idle while the Warp is computing. Thus the Warp implementation exploits both the tightly-coupled parallelism of the Warp array, and loosely-coupled task parallelism present in the benchmark.

Table XVIII lists the major subtask times for the Warp. Note that sums of the times for the individual steps will not equal the Total time because of the task parallelism. Table XIX presents the validation data supplied for the Warp implementation.

Table XVIII: Warp Times for Major Subtasks

Data set	Sample	Test	Test2	Test3	Test4
Total	43.60	20.30	22.30	58.10	55.30
Overhead	14.14	13.18	14.68	17.82	19.70
Label connected components	3.98	4.04	4.60	4.54	4.56
Rectangles from intensity	5.50	3.30	3.60	4.13	4.44
Median filter	10.70	8.70	1.38	1.40	2.00
Sobel	0.48	0.48	0.72	0.94	0.92
Initial graph match	0.42	0.24	0.22	1.22	1.38
Match strength probes	9.10	2.64	2.86	13.60	13.50
Hough probes	15.30	0.96	1.68	23.30	25.80
Result presentation	2.60	2.26	2.52	2.24	2.26

Table XIX: Warp Validation Output

Statistics	Sample	Test	Test2	Test3	Test4
Total match strength probes	91	23	20	247	239
Hough probes	58	3	5	97	95

Figure 12 shows the relative effort of the Warp on each major subtask, for each data set. Because all parallelism was used, the sum of the individual times exceeds 100 percent of the total wall-clock time. Thus, figure 12 shows percentages of the sum of the major subtask timings. The overhead for the Warp includes initialization and downloading of code for the Warp array, and the overhead for the Sun-3/60 host, which performed I/O, extracted the strong cues, and controlled all of the top-down portions of the benchmark. It is clear that the least amount of time was required for the Sobel and median filter operations, which took advantage of Warp's floating-point capabilities. The connected components labelling operation was also performed by the array, as was the K-curvature portion of extracting the rectangles from the intensity image. The host performed all of the model matching, but called on the Warp to do the match-strength and Hough probes.

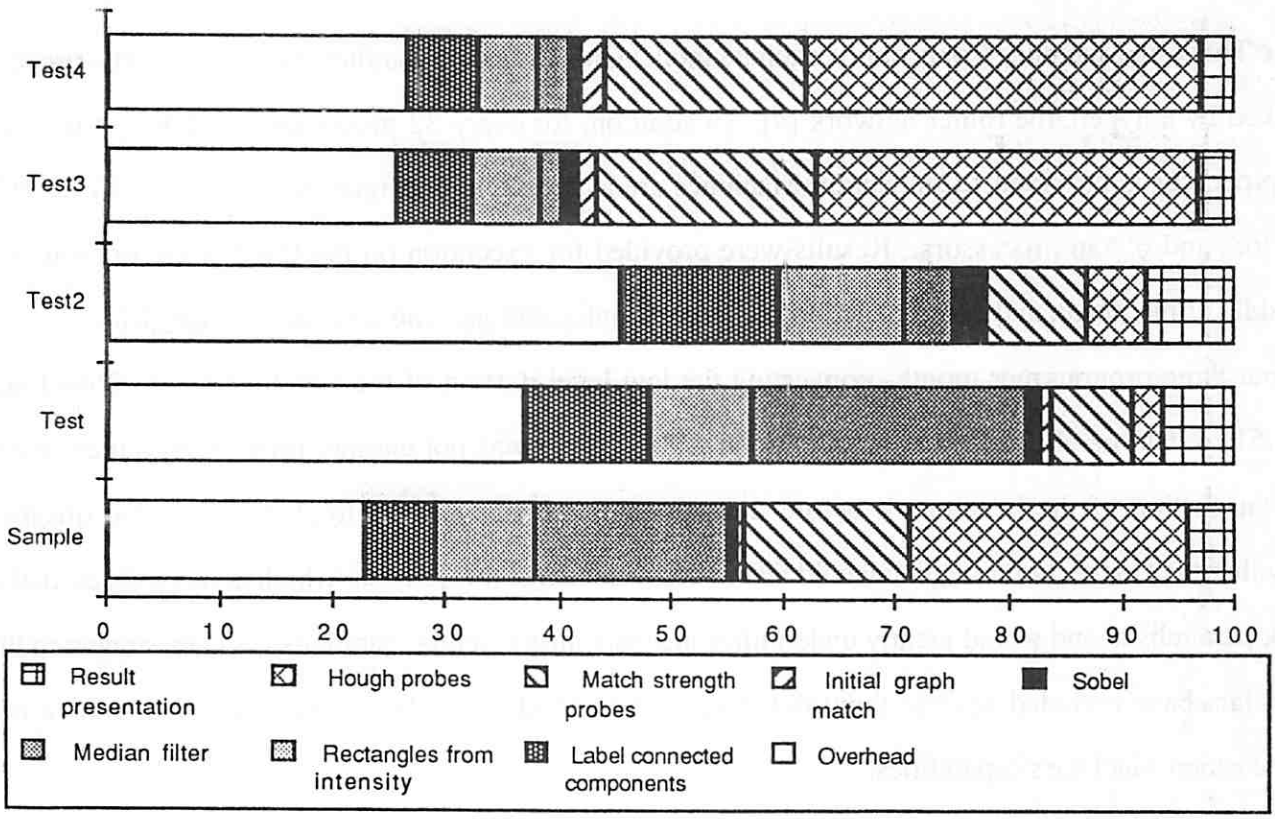

Figure 12: Warp Percent of Effort for Each Major Subtask

Connection Machine

The Thinking Machines Connection Machine model CM-2 is a data-parallel array of bit-serial processors linked by a hypercube router network [4]. In addition, for every 32 processors, a 32-bit floating-point coprocessor is provided. Connection Machines are available in configurations of 4096, 8192, 16384, 32768, and 65536 processors. Results were provided for execution on the three configurations in the middle of the range, and extrapolated to the largest configuration. The team at Thinking Machines spent about three programmer months converting the low-level portion of the benchmark into 2600 lines of *LISP, a data-parallel extension to Common LISP. There was not enough time to implement the top-down portion of the benchmark before the workshop. However, the implementors also questioned whether the Connection Machine would be the best vehicle for this portion, which is more concerned with task parallelism, and would greatly underutilize the machine's potential parallelism. They suggested that if the data base included several thousand models, a method might be found to take advantage of the Connection Machine's capabilities.

Table XX summarizes the results for the Connection Machine, with times rounded to two significant digits (as provided by Thinking Machines). A 32K-processor CM-2 with a Data Vault disk system and a Sun-4 host processor was used to obtain the results. Results were supplied for only one data set, and did not indicate which one was used. It is interesting to note that several tasks saw little speedup with the larger configurations of the Connection Machine. Those tasks involved a collection of contour values that are mapped into 16K virtual processors, which are enough to operate on all values in parallel, and so there was no advantage in using more physical processors. It was suggested that the Connection Machine might thus be used to process contours for several images at once to make use of the larger number of processors. For those tasks that are pixel oriented, 256K virtual processors were used and therefore a proportional speedup can be observed as the number of physical processors increases.

Table 20: Results for the Connection Machine on the Low-Level Portion

Configuration	8K	16K	32K	64K
Total (low level tasks only)	1.26	0.91	0.71	0.63
Overhead	0.255	0.255	0.255	0.255
Label connected components	0.34	0.21	0.14	0.10
Rectangles from intensity	0.52999	0.38437	0.31506	0.25336
Median filter	0.082	0.041	0.025	0.015
Sobel	0.052	0.026	0.014	0.008

Intel iPSC-2

The Intel Scientific Computers iPSC-2 is a distributed memory multiprocessor that consists of up to 128 Intel 80386 processors linked by a virtual cut-through routing network which simulates point-to-point communication. Each processor can have up to 8 MB of local memory, and an 80387 math coprocessor. The implementation for the iPSC-2 was developed by the University of Illinois at Urbana-Champaign using C with a library that supports multiprocessing. The group had only enough time to implement the median filter and Sobel steps. However, they did run those portions on five different machine configurations, with 1, 2, 4, 8, and 16 processors, and on four of the five data sets. Table XXI presents the results, which are divided into user time and system time (including data and program load time, and output time). Note that no system overhead was counted in the single-processor configuration because uniprocessing is done on the cube server, so no time was required for downloading code and data to the cube processors.

Table XXI: iPSC-2 Results for Median Filter and Sobel Steps

Configuration	1		2		4		8		16	
	User	System	User	System	User	System	User	System	User	System
Median Filter										
Sample	176.47	0.00	87.93	11.52	43.46	11.23	22.27	3.1	11.14	3.82
Test	75.45	0.00	37.72	10.88	18.99	10.84	9.66	3.15	4.84	3.87
Test2	60.84	0.00	30.36	11.48	15.25	11.45	7.63	3.73	3.81	4.19
Test3	60.83	0.00	30.36	11.12	15.25	11.23	7.63	3.49	3.82	4.03

Sobel	1		2		4		8		16	
Sample	78.63	0.00	39.32	3.53	19.68	3.00	9.84	2.37	4.92	2.91
Test	80.82	0.00	40.42	3.47	20.25	2.89	10.15	2.43	5.10	2.82
Test2	80.82	0.00	40.42	1.46	20.25	1.99	10.15	1.87	5.10	2.50
Test3	78.63	0.00	39.31	2.62	19.68	2.51	9.84	2.17	4.92	2.69

Comparative Performance Summary

As mentioned above, direct comparison of raw timings is not especially useful. We leave it to the reader to make informed and intelligent comparisons of the results in the context of his or her own applications. For example, a valid comparison of architectural features should take into account the technology, instruction rate, and scalability of the processors used to obtain the results. An example of such a comparison is given below. The authors hope to develop a reasonably broad set of scaling functions for future versions of the benchmark, so that it will be possible to directly compare architectures that are not too dissimilar.

In the meantime, figures 9 through 13 compare the percentage graphs for the different architectures, for the five data sets. Only the complete implementations are shown, since a total time is required to compute the charts. It should again be noted that the results for the IUA and the ASP are from simulations, and the other results are based on actual execution.

The figures make it possible to compare the architectures on the basis of their individual strengths and weaknesses, with technology factored out. Some general observations can be made from these figures. First, some dramatic speed increases are possible in the bottom-up portions of the benchmark, where the communication and control requirements are simple enough to allow data parallel processing. Second, the multiprocessors show a performance pattern similar to the uniprocessors; and in contrast, each of the data parallel arrays manages to nearly eliminate the cost of one or more subtasks, so that the remaining subtasks are greatly emphasized. Lastly, fixed overhead costs, such as I/O, grow to dominate the processing time, especially when the task itself is small.

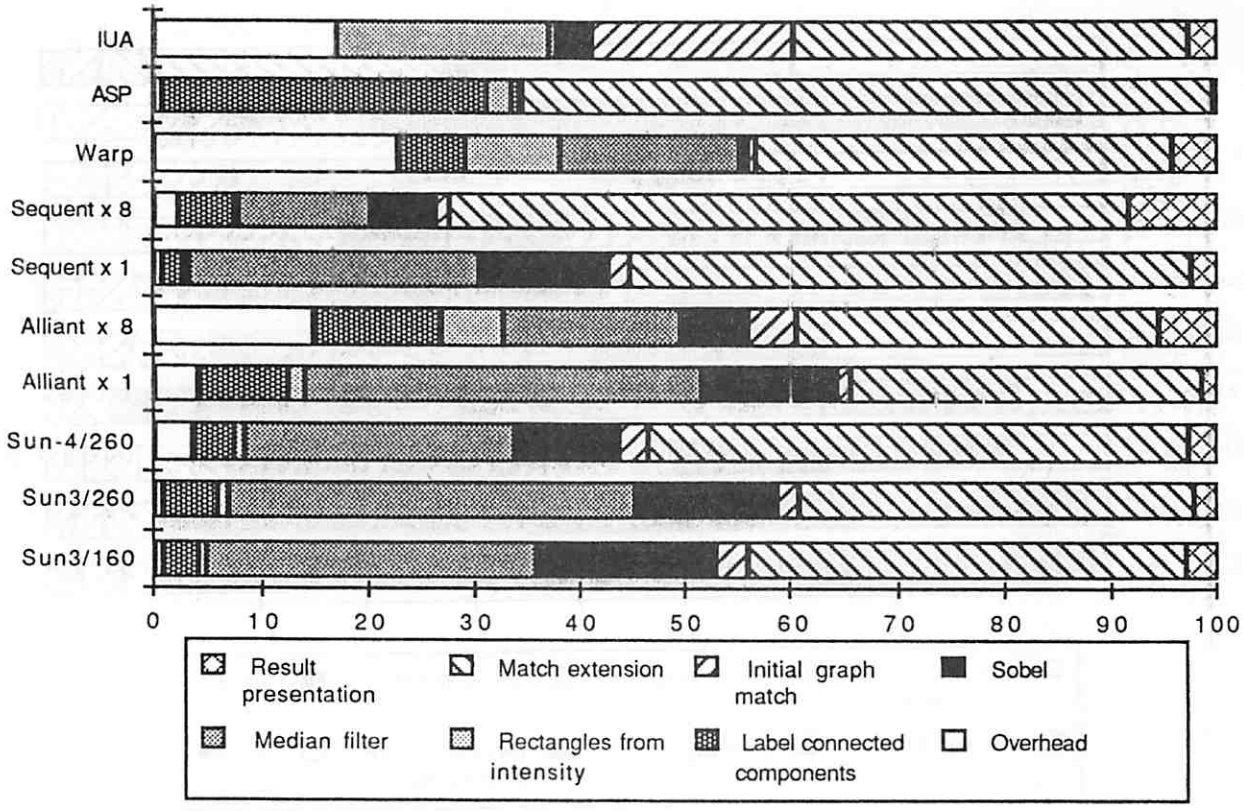

Figure 13: Distribution of Processing Time for Data Set Sample

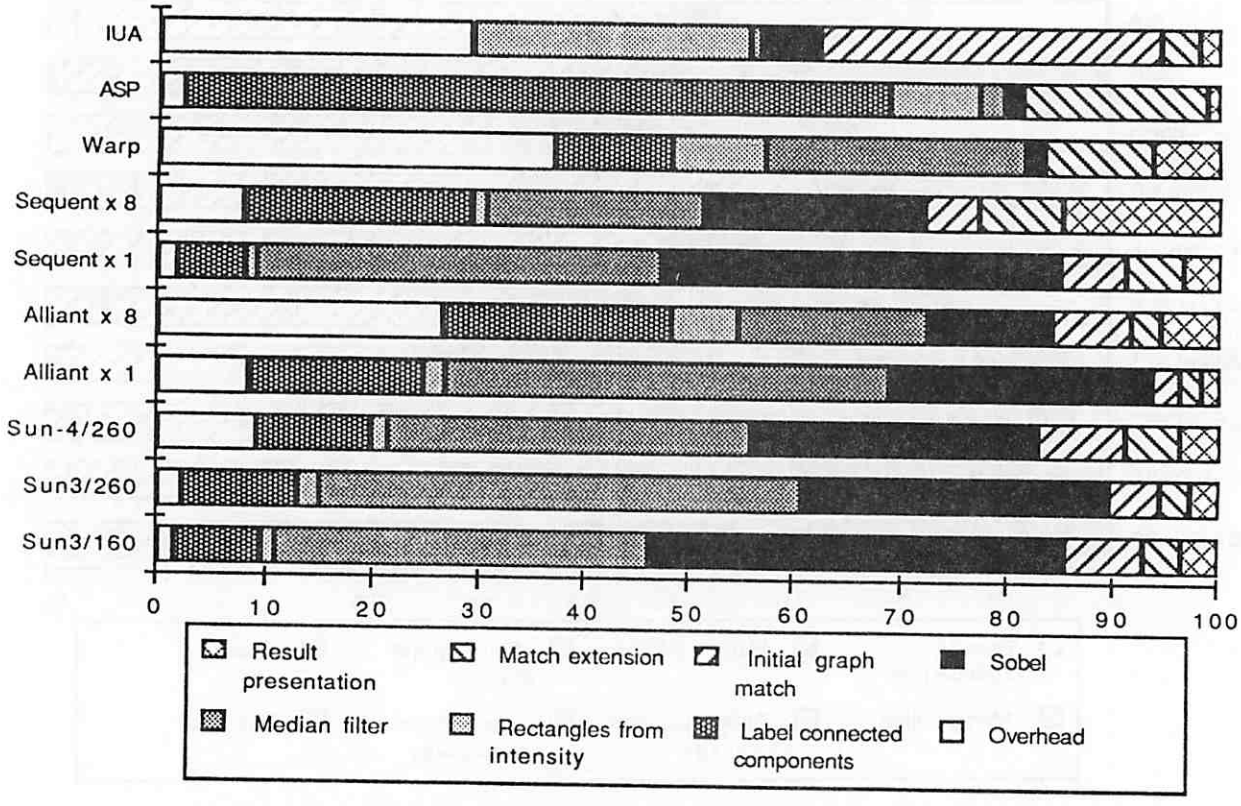

Figure 14: Distribution of Processing Time for Data Set Test

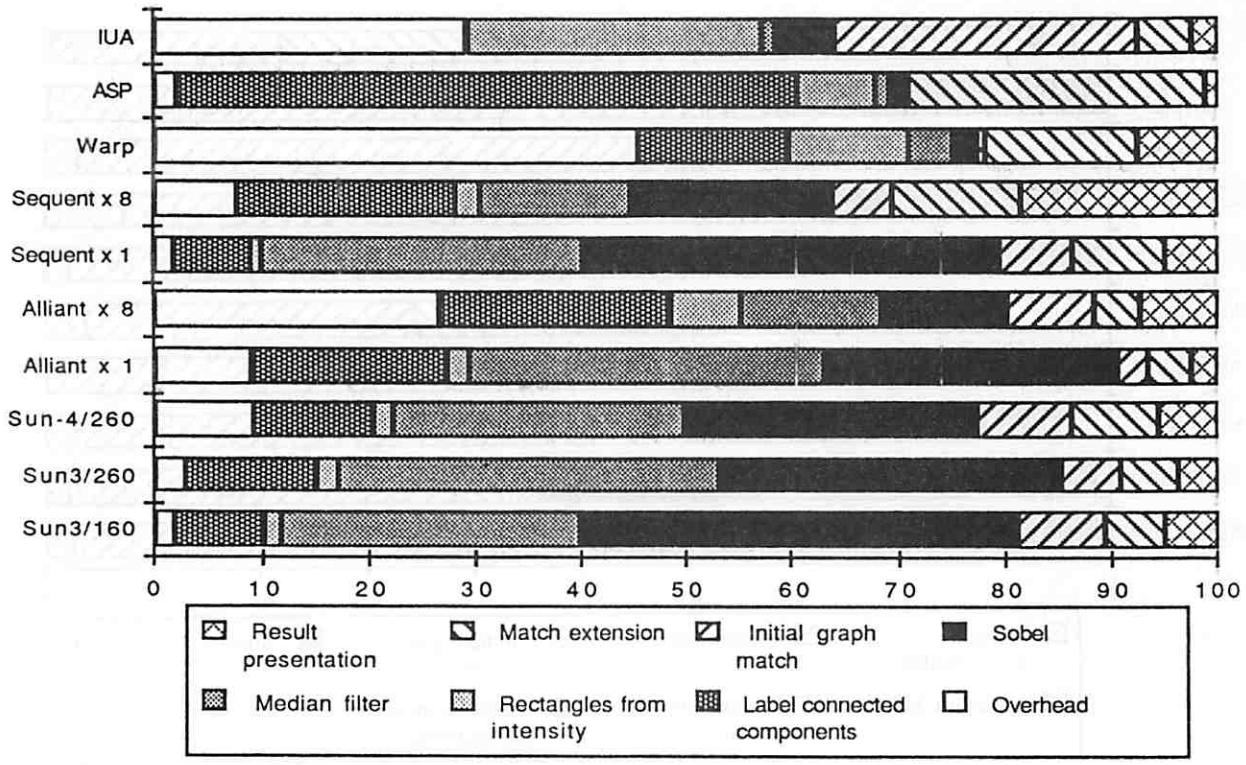

Figure 15: Distribution of Processing Time for Data Set Test2

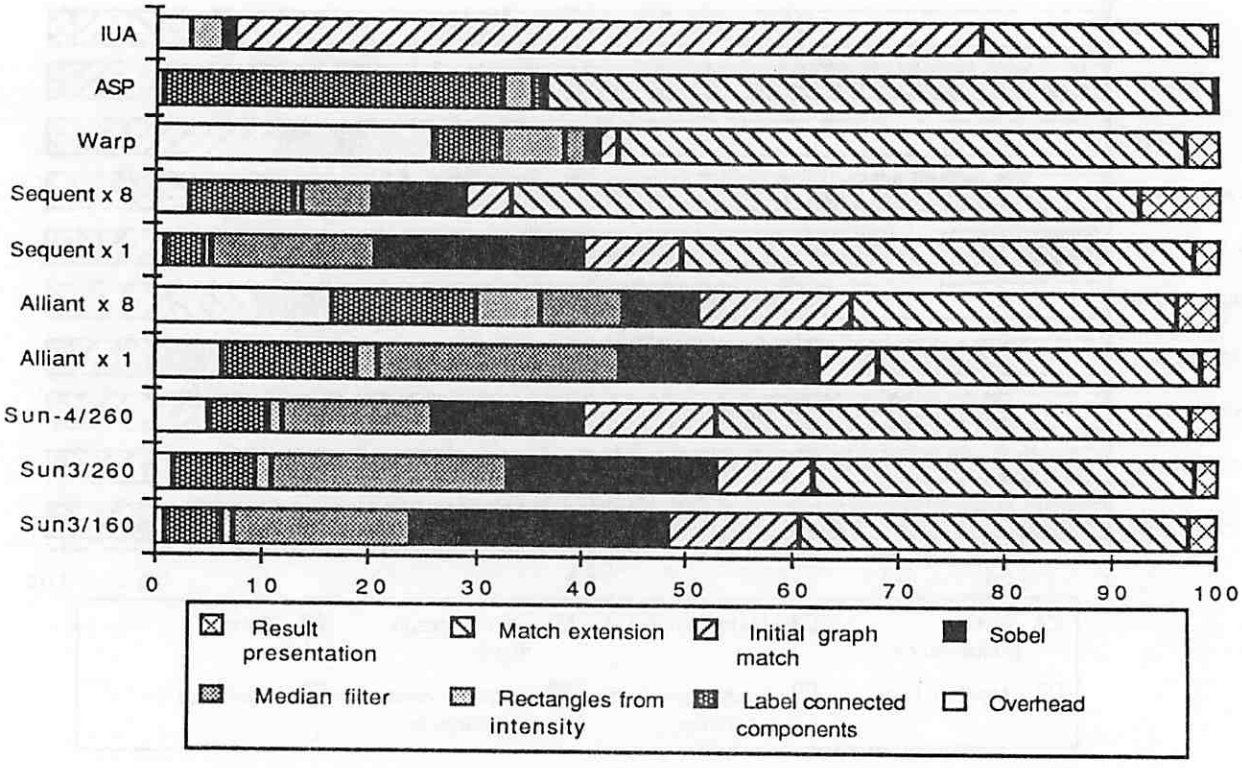

Figure 16: Distribution of Processing Time for Data Set Test3

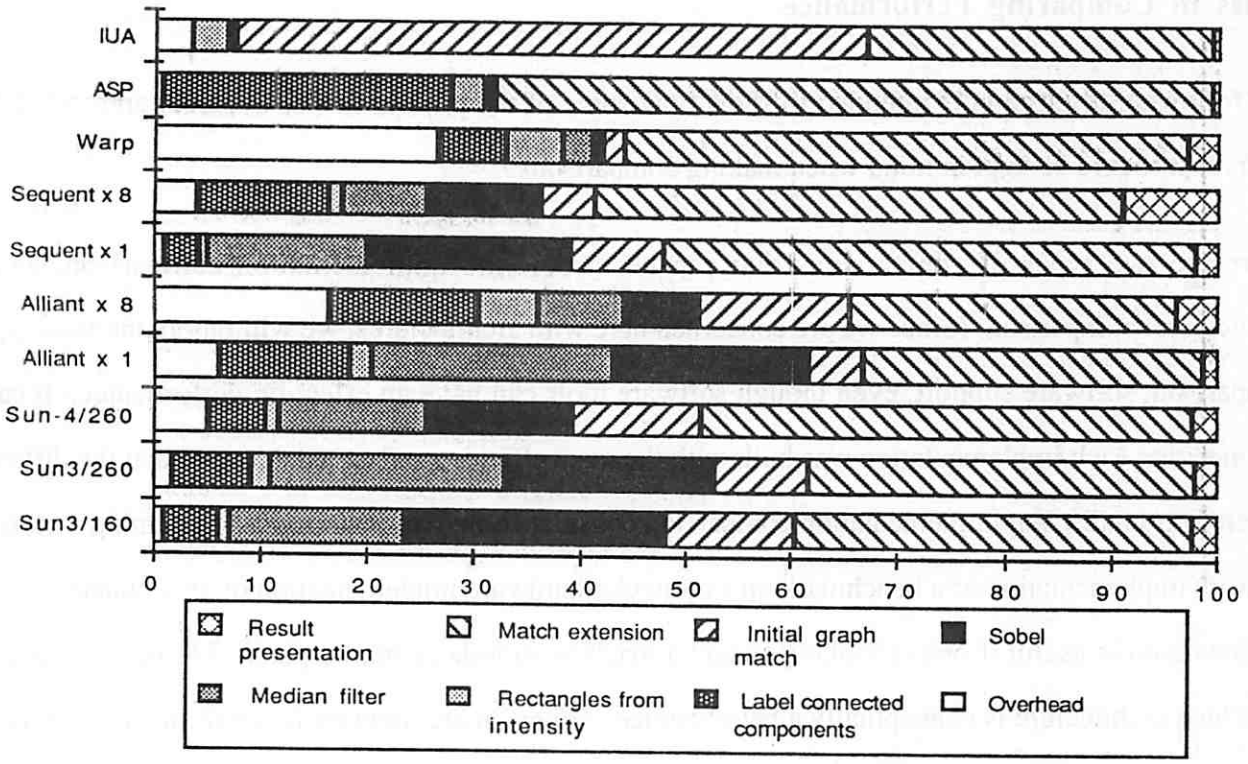

Figure 17: Distribution of Processing Time for Data Set Test4

Issues in Comparing Performance

The following discussion is intended to show how some of the pitfalls can be avoided, and to point out others that should be kept in mind when making comparisons.

There are two types of comparisons that can be made: the implementation comparison, and the architectural comparison. (Since we are concerned here with architectures, we will ignore the third type of comparison, software support, even though software tools can have an effect on performance. It can be assumed that each implementation was built with the most efficient tools available, and that the difference is therefore small.) An implementation comparison considers only the raw timings for running a particular software implementation of a benchmark on a particular hardware implementation of an architecture. Such a comparison is useful if one is looking to buy a machine in today's marketplace. However, it does not tell which architecture is conceptually a better choice. When an architecture is constrained to a hardware implementation, numerous design decisions are made that affect its performance. Any change in any of those constraints, such as improvements in technology, increase in budget, different size, weight, and power restrictions, will impact performance. Thus, one architecture may perform better than another simply because it has been committed to hardware at a later date, with more modern technology. Given equal technology, the other architecture might actually be superior.

An architectural comparison seeks to adjust benchmark timings for differences in implementation constraints. However, it is not fair to simply divide by the clock rate, list price, volume, shipping weight, and power consumption. When constraints change, there are many design choices that can be made. For example, moving to a denser integration technology makes it possible to add more processors, or to make the existing processors more powerful. Furthermore, performance does not always scale directly with changes. Consider that adding processors may increase performance only on tasks with the greatest parallelism, so that performance increases just in proportion to the processor utilization of an application. This is precisely why simulation results and actual execution results should not be compared. An architecture that is still under simulation has yet to be fully constrained.

When a constraint is changed to equal that of another system, the many possible outcomes cannot be captured by a single formula. In making any such adjustment, one is assuming that the system designers would make a specific design choice given a change in constraints. In some cases the assumption is unjustified. Even if the assumption is justified, one must be careful in estimating the effects. The process of adjusting a constraint and making an assumption about the resulting effects on a real architecture essentially turns it into a new, hypothetical architecture. Thus, without detailed knowledge of the hardware, a simple formulaic adjustment can reduce the validity of benchmark timings to just slightly better than back-of-the-envelope estimates.

As an example, we consider an architectural comparison of the Connection Machine, ASP, and IUA running the benchmark on data set Sample (it will be assumed that this is the data set used to obtain the Connection Machine timings). For the sake of this example, we will also make the invalid assumption that the simulated timings are as accurate as those for actual execution. These three architectures were chosen because, for the low-level portions of the benchmark, they have enough architectural similarity to be compared without too many distracting complications.

Table 22 compares the total times for the three architectures. The line labelled "Raw Total" would be an implementation comparison. If all three machines actually existed, the Connection machine would be the slowest. However, each of these machines has a different clock rate. For the Connection Machine the rate is 4 MHz; for the ASP, 20MHz is assumed; and for the IUA 10MHz is assumed. The second line of the table shows the times normalized to a 10MHz clock rate. This is perhaps the simplest adjustment that can be made on this data. It assumes only that, with improved technology, there is no reason that the Connection Machine clock cannot speed up by a factor of 2.5. The time shown for the Connection Machine is for 64K processors, while the other two machines have four times that number. We could simply multiply by four, but from table 20, it is clear that the total time for the Connection Machine is not scaling up linearly with the number of processors. The third line of table XXII extrapolates the 256K processor performance from the data in table 20, again with a normalized clock.

Table XXII: Example of an Architectural Comparison
(Note: based on invalid assumptions, not for actual comparison purposes)

Architecture	CM-2	ASP	IUA
(low level tasks only)			
Raw Total	0.630	0.04489	0.03343
Normalized Clock	0.252	0.08977	0.03343
Normalized Clock and Processor Count	0.228	0.08977	0.03343

The last row of figures in the table may seem like a reasonable comparison, but consider that the Connection Machine has 32-bit floating-point coprocessors, while the ASP and IUA do not. What would the performance of the ASP and IUA be if coprocessors were added to them? Also, the ASP is to be built in custom wafer-scale integration, the IUA in custom VLSI, and the Connection Machine is in gate array technology. What design changes would occur if their technologies were made equivalent?

Architectural comparisons can not be taken very far, even for similar architectures, before the hypothetical questions begin to dominate the analysis. Consider the difficulty in comparing radically different architectures such as the Alliant FX-80 and the Connection Machine. However, the more data there is available, the further the analysis can be taken. For example, running the benchmark on a CM-2 without the floating-point option would answer one of the preceding questions. The example considers only the total time for the low-level processing, but a detailed examination of subtask times can reveal patterns that would be more informative. Thus, the greater the variety of processing in a benchmark, and the more extensive its instrumentation requirements, the further architectural comparisons can be taken.

Recommendations for Future Benchmarks

At the conclusion of the Avon workshop, a panel session was held to discuss the benchmark, ways it could be improved, and future efforts. The general conclusion of the participants was that the benchmark is a significant improvement over past efforts, but there is still work to be done.

One complaint was the size and complexity of the benchmark solution. The sample solutions help, but much work is still required to transport them to parallel architectures. Several people felt that a FORTRAN version should be made available so that the benchmark would be taken up by the traditional supercomputing community. Another comment was that most groups don't have the resources to implement such a complex benchmark, and it would be almost impossible to tune its performance as is done with smaller benchmarks. A counter-argument was voiced that most vision applications are not highly tuned, and that the benchmark might therefore give a more realistic indication of the performance that could be expected. Suggestions for reducing the size of the benchmark included removing either the

match-strength or the Hough probe (although there was no consensus on which one to remove), and simplification of the graph matching code through increased generality.

Several people complained that the benchmark data sets were too small. The groups that had benchmarked data-parallel systems all indicated that they would like to see data sets involving thousands of models so that they could exploit more data parallelism, rather than being forced into a task parallel model. Of course, those who had benchmarked multi-tasking systems took the opposite view. It was suggested that the benchmark should provide a range of data sets with model-bases ranging through several orders of magnitude. Such data sets would provide another dimension to the performance analysis, and thus some insight into the range of applications for which an architecture is appropriate. Beyond simply increasing the size of the model-base, several of the vision researchers expressed a desire to see a broader range of vision tasks in the benchmark. For example, motion analysis over a succession of frames would test an architecture's ability to deal with real-time image input and help to identify those with a special ability to pipeline the stages of an interpretation. However, there was an immediate outcry from the implementors that the benchmark is already too complex. It was then suggested that an optional second level of the benchmark could be specified that would be based on the basic task, but extended to include image sequences and motion processing.

An important observation was that the complexity of the benchmark was not the issue, but the cost of implementation. It was suggested that the benchmark might be more palatable if it was reorganized into a standard set of general purpose vision subroutines. Even though a group might have to implement all of those routines, they would then have a library that could be used for other applications, over which they could amortize the cost. The benchmark specification would then be a framework for applying the library to solve a problem, and could involve separate tests for evaluating the performance and accuracy of the individual subroutines.

Part of the discussion focussed on the fact that the benchmark does not truly address high-level processing. However, as the benchmark designers were quick to point out, there is no consensus among the vision research community as to what constitutes high-level processing. Until agreement can be reached on what types of processing are essential at that level, it will be pointless to try to design a

benchmark that includes it. The current benchmark has a well-defined task of model-directed processing as its high-level component. It was also noted that the current top-down direction of low-level processing by the benchmark has some of the flavor of the high-level control of intermediate- and low-level processing which many researchers feel is necessary. It was decided that the community is not yet ready to define high-level processing to the degree necessary to build a benchmark around it.

Another point was that a standard reporting form should be developed, and that the sequential solution should output its results to match that form. Although the benchmark specification included a section on reporting requirements, the sequential solution did not precisely conform to it (partly because the reporting requirements included aspects of the implementation beyond the timings and statistics that were to be output). In fact, most of the groups followed the example of the reporting format for the sequential solution, rather than what was requested in the specification. It was also noted that because the benchmark allows alternate methods to be used whenever dictated by architectural considerations, the reporting format can not be completely rigid.

The conclusion of the panel session was to let the benchmark stand as specified for some period of time, to allow more groups to complete their implementations. Then a new version should be developed with the following features: It should be a reorganization of the current problem into a library of useful subroutines and an application framework. A set of individual problems should be developed to test each of the subroutines. A broader range of data sets should be provided, with the size of the model-base scaling over several orders of magnitude, and perhaps a set of images of different sizes. The graph matching code should be simplified and made more general purpose. A standard reporting format should be provided, with the sample solutions generating as much of the information as possible. Lastly a second level of the benchmark might be specified that extends the current problem to a sequence of images with motion analysis. The second level would be an optional exercise that could be built on top of the current problem to demonstrate specific real-time capabilities of certain architectures.

Conclusions

The DARPA Integrated Image Understanding Benchmark is another step in the direction of providing a standard exercise for testing and demonstrating the performance of parallel architectures on a vision-like task. While not perfect, it is a significant improvement over previous efforts in that it tests performance on a wide variety of operations within the unifying framework of an overall task. The benchmark also helps in eliminating programmer knowledge and cleverness as a factor in the performance results, while providing sufficient flexibility to allow implementors to take advantage of special architectural features.

Complete implementations have only been developed for a handful of architectures. In the meantime, it is possible to draw a few general conclusions from the data that has been gathered. Tremendous speedup is possible for the data parallel portions of the interpretation task. However, almost every architecture in this sample devoted the majority of its overall time to the model matching portion of the benchmark on data sets involving complex models. One conclusion might be that this portion of the task simply doesn't permit the exploitation of much parallelism. However, when the model matching step is viewed at an abstract level, it appears to be quite rich with potential parallelism, but in the form of task parallel direction of limited data parallel processing. While this style of processing can be sidestepped by increasing the size of the model-base so that the entire task becomes data parallel in nature, the inclusion of more complex and realistic high-level processing brings us back to dealing with this processing model. Thus, one potential area for research that the benchmark points out is the development of architectures, hardware and programming models to support task parallelism which can direct data parallel processing in a tightly coupled manner.

A benchmark can be used to make either implementation or architectural comparisons. Implementation comparisons can be made for any benchmark data, and are primarily useful for making purchasing decisions regarding contemporary machines. Architectural comparisons require that a benchmark include a wide variety of processing, and a rich set of instrumentation. Even then, architectural comparisons must be made with great care and an understanding of the potential for misleading results.

Acknowledgements

We would like to thank Claire Bono, Chris Brown, C.H. Chien, Larry Davis, Todd Kushner, Ram Nevatia, Keith Price, George Reynolds, Lew Tucker, and Jon Webb for their many helpful comments and suggestions in response to the draft benchmark specification.

For their efforts in designing, programming, and debugging the sequential and Sequent Symmetry solutions to the benchmark, we would like to thank Poornima Balasubramaniam, Sunit Bhalla, Chris Connolly, John Dolan, Martin Herbordt, Michael Scudder, Lance Williams, and especially Jim Burrill,

For providing benchmark results on other architectures we also thank Alok Choudhary, R. M. Lea, A Krikelis, I. Kossioris, Lew Brown, Dan Mezynski, Jon Webb, Lew Tucker, Steven Levitan, Mary Jane Irwin, Sunit Bhalla, Martin Herbordt, Michael Scudder, Michael Rudenko, and Jim Burrill.

And, for their many suggestions for improvements to the benchmark we again thank all of the above, plus Jake Aggrawal, Thomas Binford, Martin Fischler, Prasanna Kumar, Daryl Lawton, Randall Nelson, Karen Olin, Dennis Parkinson, Tomaso Poggio, Tony Reeves, Arnold Rosenberg, and Myung Sunwoo.

Bibliography

1. Carpenter, R.J., Performance Measurement Instrumentation for Multiprocessor Computers, *Report NBSIR 87-3627*, U.S. Department of Commerce, National Bureau of Standards, Institute for Computer Sciences and Technology, Gaithersburg, MD, August, 1987, 26pp.

2. Choudhary, A.N., Parallel Architectures and Parallel Algorithms for Integrated Vision Systems, *Ph.D. Thesis*, Univ. of Illinois, Urbana-Champaign, August, 1989.

3. Duff, M.J.B., How Not to Benchmark Image Processors, in *Evaluation of Multicomputers for Image Processing*, L. Uhr, K. Preston, S. Levialdi, and M.J.B. Duff, Eds., Academic Press, Orlando, FL, 1986, pp. 3-12

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

5. Kung, H.T., and Menzilcioglu, O., Warp: A Programmable Systolic Array Processor, *Proc. SPIE Symp.*, Vol. 495, Real-Time Signal Processing VII, Aug. 1984

6. Lea, R.M., ASP: A Cost-effective Parallel Microcomputer, *IEEE Micro*, October, 1988, pp. 10-29

7. Preston, K., Benchmark Results: The Abingdon Cross, in *Evaluation of Multicomputers for Image Processing*, L. Uhr, K. Preston, S. Levialdi, and M.J.B. Duff, Eds., Academic Press, Orlando, FL, 1986, pp. 23-54

8. Preston, K., The Abingdon Cross, *IEEE Computer*, Vol. 22, No. 7, July, 1989, pp. 9 -18

9. Rosenfeld, A.R., A Report on the DARPA Image Understanding Architectures Workshop, *Proceedings of the 1987 DARPA Image Understanding Workshop*, Morgan Kaufmann Pub., Los Altos, CA, 1987, pp. 298-302

10. Sunwoo, M.H., and J.K. Aggarwal, VisTA: An Image Understanding Architecture, in *Parallel Architectures and Algorithms for Image Understanding*, V.K. Prassana Kumar, Ed., Academic Press, 1990.

11. Weems, C.C., E.M. Riseman, A.R. Hanson, and A. Rosenfeld, An Integrated Image Understanding Benchmark, Recognition of a 2-1/2 D "Mobile," *Proceedings of the 1988 DARPA Image Understanding Workshop*, Morgan Kaufmann Pub., Los Altos, CA, 1988, pp. 111-126

12. Weems, C.C., E.M. Riseman, A.R. Hanson, and A. Rosenfeld, A Computer Vision Benchmark for Parallel Processing Systems, *Proceedings of the Third International Conference on Supercomputing*, Volume III, International Supercomputing Institute, Pub., St. Petersburg, FL, 1988, pp. 79-94

13. Weems, C.C., A.R. Hanson, E.M. Riseman, and A.R. Rosenfeld, A Parallel Processing Benchmark for Vision, *Proceedings of the 1988 IEEE Conference on Computer Vision and Pattern Recognition*, 1988, pp. 673-688

14. Weems, C.C., S.P. Levitan, A.R. Hanson, E.M. Riseman, D.B. Shu, and J.G. Nash, The Image Understanding Architecture, *International Journal of Computer Vision*, Vol. 2, Kluwer Academic Publishing, Boston, MA, 1989, pp. 251-282.

15. Weems, C.C., A.R. Hanson, E.M. Riseman, and A. Rosenfeld, A Report on the DARPA Integrated Image Understanding Benchmark Exercise, *Proceedings of the 1989 DARPA Image Understanding Workshop*, Morgan Kaufmann Pub., Los Altos, CA, 1989, pp. 165-192