TEC-0029

(2)

US Army Corps of Engineers
Topographic Engineering Center

Image Understanding Architecture Final Report

Charles C. Weems
Edward M. Riseman
Michael Scudder
Deepak Rana
James Burrill
Martin Herbordt
Allen R. Hanson

University of Massachusetts
Computer Science Department
Amherst, MA 01003

September 1991

Approved for public release; distribution is unlimited.

Prepared for:
Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, VA 22209-2308

Monitored by:
U.S. Army Corps of Engineers
Topographic Engineering Center
Fort Belvoir, Virginia 22060-5546

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE September 1991	3. REPORT TYPE AND DATES COVERED Final Technical Report Sep. 1986 - Sep. 1991

4. TITLE AND SUBTITLE Image Understanding Architecture Final Report	5. FUNDING NUMBERS DACA76-86-C-0015
6. AUTHOR(S) Charles C. Weems Edward M. Riseman Allen R. Hanson James Burrill Martin Herbordt Michael Scudder Deepak Rana	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) University of Massachusetts (UMass) Computer Science Department Amherst, MA 01003	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Defense Advanced Research Projects Agency 1400 Wilson Blvd., Arlington, VA 22209-2308 U.S. Army Topographic Engineering Center Fort Belvoir, VA 22060-5546	10. SPONSORING/MONITORING AGENCY REPORT NUMBER TEC-0029

11. SUPPLEMENTARY NOTES

Effective 1 October 1991, the U.S. Army Engineer Topographic Laboratories (ETL) became the U.S. Army Topographic Engineering Center (TEC).

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The primary goal of the Image Understanding Architecture (IUA) project was to build a proof-of-concept prototype of a 1/64th slice of a parallel architecture to support real-time, knowledge-based image understanding, and develop the software support environment that will be needed to utilize the hardware. The unique feature of the IUA is that it tightly couples three distinctly different parallel architectures whose capabilities are matched to the computational requirements of the three primary levels of abstraction in knowledge-based computer vision. The low-level processor (CAAPP) is a SIMD cellular array designed to perform sensory preprocessing. The intermediate-level processor (ICAP) is intended to support computation on symbolic data that has been extracted from the images or instantiated from stored models. The high-level processor (SPA) supports the general knowledge-based processing which employs multiple strategies to form an interpretation of a scene. The majority of the hardware effort has taken place at Hughes Research Laboratories, Malibu, California, although UMass has principle responsibility for the design of the IUA architecture. UMass has also undertaken some smaller portions of the hardware development (the feedback concentrator for the low and intermediate-level arrays, and the communications router for the intermediate-level array). The majority of the software effort took place at UMass, although Hughes was also involved in some software development, both in support of their hardware efforts, and in the form of algorithm development for specific applications on the IUA.

14. SUBJECT TERMS Image Understanding Architecture, Knowledge-Based Vision, Real-Time Computer Vision, Software Simulator, Parallel Processor	15. NUMBER OF PAGES 61
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UNLIMITED

Table of Contents

List of Figures

List of Tables

Preface

This research is spnsored by the Advanced Research Projects Agency (ARPA) and monitored by the U.S. Army Topographic Engineering Center (TEC) under Contract DACA76-86-C-0015, titled "Image Understanding Architecture". The ARPA Program Manager is Col. Erik Mettala, and the TEC Contracting Officer's Representative is Ms. Lauretta E. Williams.

These efforts encompass both software development and hardware design, including some discussion of the hardware fabrication effort at Hughes Research Laboratories.

Because a great deal of documentation has already been written as part of this effort, we have chosen to assemble the majority of this report by editing existing documents. The report begins with an executive summary of the major accomplishments in the contract period. The remainder of the report presents an overview of the project, and of our efforts under this contract. The reader is thus referred to the executive summary for a status report on the project and to the later sections for more detailed information.

Executive Summary

The goals of the Image Understanding Architecture (IUA) contract were to construct a proof of concept prototype of a heterogeneous parallel processor to support machine vision, and to develop software support and demonstration applications for the machine. These goals have been essentially achieved, although the schedule for the program was delayed for various reasons. Primarily, the delays were caused by problems with the MOSIS integrated circuit brokerage service. Ours was one of the first major research efforts to try building large chips via MOSIS. As with any pioneering effort, we encountered difficulties which have since been resolved. The chip fabrication delays also caused slippage in other portions of the project schedule, which in turn required the timetable to be stretched, which then resulted in some additional costs. Delays in receiving funding and legal issues relating to nondisclosure agreements also incurred some delays.

In the area of software, as required by the contract, simulators were developed for the Content Addressable Array Parallel Processor (CAAPP) and Intermediate and Communication Processor (ICP) levels of the architecture, and these were used to evaluate the design. Some revisions were made to the chip architecture and then incorporated in the simulators. A parallel programming language (code generator) was developed for the CAAPP level of the architecture, based on the FORTH programming language. A programming system for the ICP level using a standard C compiler together with a library of routines for interprocessor communication was developed. The functional simulator was extended to the General Purpose Processor Array (GPPA) level in that the prototype uses the host machine as its GPPA level, and the host supports a high-level vision system that can be used with the simulators for the other two levels. Software development tools for the system include a debugger for the CAAPP and ICP levels, a memory management system for the CAAPP level, various image-file conversion utilities, and a graphics-based programmer's interface that allows the machine's status to be displayed as images and animated or static diagrams. As part of our work on the DARPA Integrated Image Understanding Benchmark, we examined and developed vision applications in low-level image event extraction, intermediate-level grouping, matching models to grouped primitives, and feedback mechanisms. We examined but did not implement any algorithms specifically for the system in the areas of planning, inference, and control. A planning system has been implemented separately for the host processor, and we have a high-level vision system that uses inference, but neither of these has been integrated with our simulation environment, primarily because they and the simulations are too large to run together on the same machine without being impractically slow. As can be seen from the list of activities below, a significant portion of our effort has been spent on cooperating with DARPA and strategic computing contractors to share our results and explore avenues for transferring technology.

In the area of hardware, Hughes essentially completed all of the objectives of the project. A 1/64th scale proof-of-concept prototype of the Image Understanding Architecture has been constructed. It contains 4096 CAAPP processing elements and 64 ICP processors. A simple array control unit has been built that allows the machine to be tested and demonstrated with small but representative applications. The machine is fully functional in that all of the memory, communication channels, and processing elements operate correctly, although at a slower speed than was our original target.

Two significant problems remain in the prototype hardware that cause it to run at 2 MHz instead of 10 MHz, as was originally specified. The first problem involves a subtle interaction between two adjacent blocks of logic in the CAAPP chip, one of which is memory. Although Hughes simulated each of these blocks separately, they did not have the computational resources to simulate them together. It turned out that there was an overlap between the two blocks in which a line ran across some of the memory cells. Even though there was no contact between the line and the cells, it induced additional leakage in the cells such that their refresh time could become impractically short (i.e. they lose their contents in only a few hundred cycles). The problem has been addressed by lowering voltages and increasing cycle times in the processor to reduce the leakage effect, and the processors are now functionally operational but at a reduced speed. Fully correcting the problem would require a revised run of silicon, which is not covered in the budget. The second problem is that Hughes did not use enough ground pins in the CAAPP chip design. Hughes worked around this problem by adjusting the voltages, bias, and clocking of each individual chip in their test fixture. Under these conditions, all of the chips function (except for the memory problem) at 10 MHz, but these conditions cannot be replicated at the individual chip sites on the boards. Again, lowering the voltages and clock rate has solved the problem at a cost in performance. To fully address this problem would require that the chips be recast in a different package with more ground pins, and that the processor boards be reworked to accept the larger package. Alternatively, the boards could be reworked at each chip site to obtain maximum performance from each chip, but that would make replacement of faulty parts very difficult.

The major accomplishments and activities performed during the period of this report, including the subcontract with Hughes Research Laboratories, are listed in roughly chronological order as follows:

1. The first DARPA IU Benchmark, a suite of seven IU algorithms, was manually analyzed for the IUA.

2. A functional specification for use by Hughes was developed for each of the IUA elements. This included a revised CAAPP (low-level processor) instruction set, a block structure definition of the IUA, and selection of the TMS320C25 as the processing element for the intermediate-level array (ICAP).

3. Preparation of a set of IU Benchmark times for an Encore Multimax multiprocessor.

4. Representatives of UMass attended the first DARPA IU Benchmark workshop.

5. Representatives of UMass attended a DARPA meeting on VLSI and Architectures in December of 1986.

6. UMass met with Hughes to further design the IUA hardware: designed the backing store controller and instruction set, designed the ICAP bus arbiter, designed the Array Control Unit (ACU) interface to the ICAP bus, designed the DRAM refresh system, designed the CAAPP-ICAP Shared Memory (CISM) page mapping system, defined the CAAPP-ICAP communication registers, designed the ICAP-SPA Shared Memory (ISSM) interface, developed electrical timing diagrams for each inter-level interface, developed a sample program using interlevel communication, and developed a functional specification for the ACU.

7. Hughes designed and fabricated a 32-processor CAAPP test chip.

8. UMass obtained a Texas Instruments Explorer Workstation with an Oyssey board containing four TMS320C20 processors, to be used in simulating the IUA.

9. Hughes and UMass developed a detailed logic design of the CAAPP processor.

10. UMass wrote a paper on the IUA for the 1987 DARPA IU Workshop.

11. UMass representatives attended a DARPA workshop on Ultra-large Scale Message Passing Concurrent Computers in February 1987.

12. UMass developed an IUA simulator for the TI Explorer/Odyssey system.

13. UMass wrote a document entitled "Philosophy of Benchmarks for Computer Architectures to Support Machine Vision," to be circulated in the IU community for comment. This was in preparation for developing a new DARPA IU Benchmark with the University of Maryland.

14. UMass met with DARPA and ESL Corp. to discuss construction of an Array Control Unit (ACU) for the IUA.

15. At the request of DARPA, UMass prepared an estimate of IUA processing time for performing a typical vision interpretation and MFLOPS and MIPS equivalents for the IUA.

16. UMass representatives attended the 1987 DARPA IU Workshop and gave a talk on the IUA.

17. UMass completed a software simulator for the CAAPP processor on the VAX.

18. UMass layed out a 64-way Fast-OR chip for fabrication through MOSIS, to be used as part of the feedback from the CAAPP and ICAP to the ACU.

19. Hughes developed behavioral simulations of the CAAPP chip and processor daughterboard.

20. UMass met with ESL Corp. to further discuss development of an ACU.

21. UMass gave a presentation on the IUA at a Texas Instruments Satellite Symposium in April of 1987.

22. Hughes and UMass met with DARPA to discuss problems encountered in using MOSIS to support a development effort.

23. UMass prepared transparencies for use by DARPA.

24. UMass gave a presentation on the IUA to representatives of the Defense Mapping Agency.

25. UMass gave a 6-month program review at DARPA, together with Hughes. Also met with ESL to discuss the ACU development. ESL concluded that they did not wish to participate, and UMass and Hughes were directed to prepare a new proposal regarding the ACU.

26. UMass and Hughes wrote an article on the IUA for the International Journal of Computer Vision.

27. UMass developed a floating point subroutine package for the VAX-based simulator.

28. UMass completed the design for a crossbar-based communication network for the ICAP.

29. A UMass representative attended the International Symposium on Computer Architecture.

30. UMass and U. of Maryland completed the specification for the new DARPA IU Benchmark and distributed it to the IU community for comment.

31. A supplementary proposal regarding the ACU was written and submitted to DARPA by Hughes and UMass.

32. A project summary was written for DARPA by UMass.

33. The floating point package was transported by UMass to the TI Explorer/Odyssey-based simulator.

34. An ICAP communication network chip was designed by UMass and submitted to MOSIS for fabrication.

35. Hughes received a bad lot of 32-processor CAAPP test chips from MOSIS (after considerable delays in the fabrication). Hughes was able to determine that the static structures were functional, through microprobing of the dies, but the chip defects made the dynamic structures inoperable.

36. UMass representatives attended the 1987 International Conference on Parallel Processing.

37. Enhancements were made by UMass to the debugging tools for the IUA simulator.

38. Comments from the DARPA IU community on the IU Benchmark specification were used to refine the design.

39. Hughes completed the design of a 64-processor CAAPP test chip and submitted it to MOSIS.

40. A new project summary together with a quarter chart was prepared by UMass for DARPA.

41. Representatives of UMass attended the DARPA ISTO PI meeting in Gaithersburg, MD in September of 1987.

42. The VAX-based simulator was transported to a new Sun workstation by Umass, and a FORTH interpreter was integrated with it and the SunViews windowing system to provide an interactive environment.

43. UMass developed test-data generation software for the IU Benchmark.

44. The UMass Fast-OR chips were fabricated unsuccessfully due to a mix-up in communication with MOSIS. Instead of the 2 micron process we thought we were requesting, the chips were built with a 1.5 micron process. The chips were redesigned in CMOS (formerly in NMOS) for resubmission.

45. Representatives of UMass attended the Computer Architectures for Pattern Analysis and Machine Intelligence conference in Seattle, October 1987, and participated in a panel session on vision architectures.

46. UMass added gray-scale display capabilities to the Sun-based simulator.

47. UMass implemented a memory management package for the Sun-based CAAPP simulator.

48. UMass implemented convolution, connected component labelling, and corner detection on the simulator as part of the benchmark work.

49. UMass developed code for the sequential solution to the IU Benchmark and sent it to U. Maryland for verification of portability and generality.

50. UMass developed a revised functional specification for the CAAPP processor chips, and an up-to-date documentation set, together with Hughes.

51. The UMass ICAP communication chips were returned from MOSIS and it was found that, like the 32-processor Hughes CAAPP test chips, the processing had resulted in a bad lot of chips. Excessive metal spreading in Metal-2 had resulted in widespread shorts.

52. The redesigned Fast-OR chip was resubmitted to MOSIS by UMass.

53. Laser-printer screen hardcopy functions were added to the IUA simulators by UMass.

54. Separate presentations were given to the Interagency AI Steering Committee, Defense Mapping Agency, and General Motors on the IUA project by UMass.

55. UMass and Hughes briefed representatives of DARPA on project status as of December, 1987.

56. UMass and Hughes worked with MOSIS to resolve our fabrication problems.

57. The new Fast-OR chips were received by UMass and found to be fully functional.

58. UMass met with representatives of Texas Instruments to discuss the IUA and TI applications.

59. UMass installed an LX-Unix board into the TI Explorer/Odyssey system, but found that incompatibilities between the Odyssey and LX processors prevented their use together.

60. UMass briefed a representative of AFOSR on the IUA project.

61. UMass received the ICAP communication test chips and tested them. Three design errors were found.

62. UMass wrote an article for the DARPA IU Workshop on the IUA programming environment and IUA algorithms.

63. Hughes and UMass prepared the first annual report for the contract.

64. UMass wrote an article for the IU Workshop on the new DARPA IU Benchmark.

65. UMass developed a simulator for the TMS320C25 because we were unable to obtain one from TI.

66. UMass corrected the ICAP Communication chip design.

67. UMass developed two "Grand Challenge" statements at the request of DARPA.

68. UMass implemented a "Spiral Out" operation for the CAAPP simulator on the Sun to simplify the development of convolution operations.

69. UMass briefed a representative of ONR on the IUA project, and discussed evaluation of a Xenologics X-1 processor for use in a future SPA design.

70. UMass met with representatives of IMS Inc. to explore the possibility that one of their ASIC testers could be used as an interim IUA ACU.

71. Hughes completed construction of a daughterboard emulation/chip tester board to be driven by their IMS test system.

72. UMass representatives attended the DARPA IU Workshop in April, 1988 and presented two papers.

73. UMass developed a parallel implementation of the IU Benchmark for the Sequent Symmetry multiprocesor.

74. UMass implemented the median filter and K-curvature portions of the IU Benchmark on the IUA simulator.

75. UMass revised the simulator to reflect (unauthorized) changes in the CAAPP Coterie network that had been developed by Hughes under IR&D support.

76. UMass resubmitted the ICAP communication chip to MOSIS for fabrication. The corrected design also includes some architectural enhancements.

77. Hughes tested the 64-processor CAAPP test chip and found several design errors, although major portions of the design are functional.

78. UMass representatives attended the International Conference on Supercomputers and presented two papers, one on the IUA, and the other, on the IU Benchmark.

79. UMass briefed a representative of the Supercomputing Research Center on the IUA at UMass.

80. The TMS320C25 simulator was modified by UMass to work as an ICAP processor simulator, and the ICAP level of the Sun-based simulator was implemented.

81. UMass implemented the gradient magnitude portion of the IU Benchmark on the CAAPP simulator.

82. UMass completed the timing and instrumentation report specification for the IU Benchmark.

83. UMass implemented an assembler for the ICAP processor simulator.

84. UMass implemented the Sobel portion of the IU Benchmark on the IUA simulator.

85. UMass representatives attended the 1988 Computer Vision and Pattern Recognition Conference and presented a paper on the IU Benchmark.

86. UMass wrote a paper for the 1988 CONPAR conference on the preliminary performance of the IUA and the IU Benchmark.

87. UMass representatives attended the 1988 Computer Architecture Symposium.

88. UMass developed a bit-slice design for an ACU.

89. Hughes completed the layout of the backing store memory controller for the CAAPP.

90. A UMass representative gave an invited talk on the IUA at the Supercomputing Research Center.

91. UMass implemented a parallel version of the IUA simulator on the Sequent Symmetry, which provides for simulation of a full-scale CAAPP array.

92. UMass implemented software to support communication between the CAAPP and ICAP levels of the IUA.

93. The IU Benchmark was initially distributed by UMass to 20 sites.

94. Representatives of UMass attended the 1988 International Conference on Parallel Processing.

95. UMass implemented the match-strength probe and Hough probe portions of the IU Benchmark on the IUA simulator.

96. UMass met with representatives of Sequent to discuss improving the IU Benchmark performance on the Symmetry.

97. A UMass representative gave a presentation on the IUA to representatives of Digital Equipment Corp. at an invited talk.

98. The revised UMass ICAP communication chips were received from MOSIS, which again had problems with the fabrication run. Of the very small number of usable die (6), four were fully functional.

99. Hughes submitted a complete CAAPP chip design to MOSIS for fabrication. Due to a delay at MOSIS, Hughes took the opportunity to add some features to the CAAPP chip. The complete design has 130,000 devices with 80,000 in the memory, occupying 60% of the chip area. The size of the chip is 350 x 330 mils.

100. A representative of UMass attended the 1988 CONPAR conference in England, and while there, visited labs at the University of Warwick and University College London.

101. The rectangle hypothesis generation portion of the IU Benchmark was implemented on the IUA simulator by UMass.

102. UMass briefed the DARPA/ISTO Office Director on the IUA in October of 1988.

103. A representative of UMass attended the 1988 Frontiers of Massively Parallel Processing Conference and gave a poster paper on the performance of the IUA on the DARPA benchmark.

104. UMass organized and ran a DARPA-sponsored workshop on the IU Benchmark in October of 1988.

105. UMass completed the IUA implementation of the DARPA benchmark.

106. Hughes completed simulations of the CAAPP processor chip design and released the chip to MOSIS for fabrication.

107. Dr. Weems attended the 1988 DARPA Principal Investigators Meeting in Dallas, TX, and presented the results of the IU Benchmark Workshop; he also participated on a new ideas panel session for computer architecture.

108. UMass implemented the Abingdon Cross benchmark on the IUA simulator.

109. UMass received a third run of ICAP communication chips from MOSIS and tested them. They are slower than the earlier runs and the yield is 14 working parts out of 60.

110. UMass implemented a set of trigonometric routines for the CAAPP and ICAP using CORDIC methods.

111. A UMass paper on the ICAP communication chip was accepted by the 1989 IEEE Conference on Circuits and Systems, and camera ready copy was prepared and submitted.

112. Hughes received and tested the 64-processor CAAPP test chips from MOSIS, and in spite of very low yield, managed to get two of the chips to work together in a demonstration test jig.

113. UMass implemented the Weymouth-Overton edge-preserving smoothing operator on the CAAPP simulator, using FORTH.

114. UMass briefed DARPA program managers on project status at DARPA in February of 1989.

115. Dr. Weems attended the 1989 DARPA Image Understanding Review meeting.

116. UMass received a fourth run of ICAP communication chips from MOSIS. These were tested and the yield was found to be much better (53 of 98 parts are fully functional).

117. An annual report was prepared by UMass and submitted to ETL for review.

118. An article on the IU Benchmark exercise was written by UMass for the 1989 DARPA IU Workshop.

119. Representatives of General Dynamics were briefed on the IUA in an April 1989 visit to UMass.

120. A generalized permutation routing algorithm has been developed for the CAAPP by UMass.

121. Researchers at Textron Defense Systems were briefed on the IUA by UMass at a May, 1989 meeting.

122. UMass purchased and installed an IMS ASIC test system to act as an interim controller for the IUA, following the lead of Hughes. The system will also be used to test custom chips developed at UMass.

123. Dr. Weems attended the 1989 IEEE Circuits and Systems Conference with a student who presented a paper on the ICAP communication chip. While there, they met with IMS to discuss a faster I/O link between the Sun host and the test system.

124. Dr. We[illegible]s attended the 1989 DARPA IU Workshop where he presented the results of the IU Benchmark exercise. While there, he met with Hughes to discuss project efforts.

125. The design of a new feedback concentrator chip for the IUA was completed by UMass and submitted to MOSIS for fabrication. The design will integrate the Global-OR and Response Count functions. The previous design performed only the Global-OR.

126. UMass developed an alternative design for an ACU, based on a SPARC co-processor. This design was intended to explore the performance/cost tradeoffs of such an approach. It appears that the coprocessor interface is too inefficient to control the IUA at full speed.

127. Hughes completed debugging of the CAAPP processor chip, and successfully demonstrated its functionality.

128. Hughes decided to contract the IUA integration work to an outside consultant (VillaMar Inc.) with lower overhead rates. However, this was delayed for some time while the contract modification and a six-month no-cost extension were processed to allow it to take place, which further delayed the completion of the IUA prototype.

129. Hughes completed construction of the IUA motherboard and daughterboards.

130. Hughes completed associated circuit boards for the IUA prototype.

131. Hughes tested the IUA prototype system and found numerous bugs, most of which were corrected. The remaining problems are avoided by reducing the clock rate and voltage of the system.

1. Introduction

Computer vision using color imagery requires a processor capable of accepting 23 megabytes of input per second, and interpreting it to construct a three dimensional model of the sensor's environment. An interpretation may require hundreds of objects of many different types to be identified. Vision researchers [Hanson, 1986] have shown that pattern recognition techniques, by themselves, are inadequate for this task. In fact, most of what we "see" in natural scenes is really inferred from partial information. In addition to sensory and knowledge-based processing it is useful to introduce a level of symbolic processing. Thus, vision researchers tend to classify algorithms and representations into three levels: low (sensory), intermediate (symbolic), and high (knowledge-based).

While it may be argued that a general-purpose processor can fulfill the requirements of vision, the goal of real-time performance necessitates the use of special-purpose processors. Another key to achieving real-time performance is processing at all levels simultaneously, which leads to the idea of linking together three different parallel processors. But because of the massive amount of communication between levels, and the requirement for flexible, top-down control, the different parallel processors must be designed from the start to be tightly coupled with each other. This analysis lead to the concept of the Image Understanding Architecture (IUA).

The primary goal of the IUA project (Contract DACA-76-86-C-0015) was to build a proof-of-concept prototype of a 1/64th slice of a parallel architecture to support real-time, knowledge-based image understanding, and develop the software support environment that will be needed to utilize the hardware.

The majority of the hardware effort has taken place at Hughes Research Laboratories, Malibu, California, although UMass has principle responsibility for the design of the IUA architecture. UMass has also undertaken some smaller portions of the hardware development (the feedback concentrator for the low and intermediate level arrays, and the communications router for the intermediate level array). The majority of the software effort took place at UMass, although Hughes was also involved in some software development, both in support of their hardware efforts, and in the form of algorithm development for specific applications on the IUA.

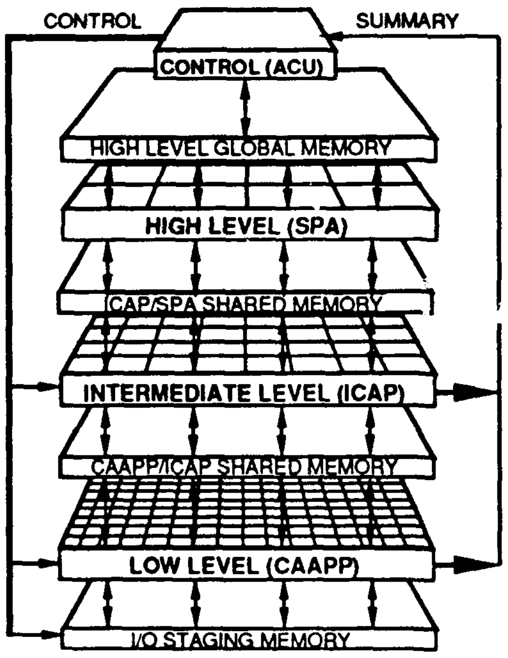
Our software efforts have included development of a detailed software simulation of the IUA and a FORTH interpreter for the low-level processor of the IUA. The remainder of the software effort has been in the development of run-time support libraries, diagnostics, and vision algorithms. Some effort was also spent on implementing the original DARPA Image Understanding Benchmark and on developing a successor to the benchmark.

2. Overview of the Image Understanding Architecture (IUA)

The Image Understanding Architecture [Weems, 1989] consists of three different, tightly coupled parallel processors. The low- and intermediate-levels are controlled by a dedicated Array Control Unit (see Figure 1) (ACU) that takes its directions from the high level. As Figure 1 indicates, each of these processors provides a different granularity and different modes of parallelism. We have built a 1/64th slice of the IUA as a proof-of-concept demonstration. The discussion that follows describes the full IUA, except where it is noted that a feature pertains only to the prototype.

At the high level, the IUA is purely a Multiple-Instruction Multipe-Data (MIMD) parallel processor. In our original proposal, the high level was called the General Purpose Processor Array (GPPA), but has since been renamed the Symbolic Processing Array (SPA) to avoid confusion with scientific parallel processors. The low level, called the Content Addressable Array Parallel Processor (CAAPP), operates in pure Single-Instruction Multiple-Data (SIMD) or multi-associative mode, and the intermediate level operates in Single Program, Multiple Data (SPMD) or MIMD mode. In the original proposal, the intermediate level was called the Intermediate and Communication Processor (ICP), but was later renamed the Intermediate Associative and Communication Processor (ICAP), to reflect the emphasis of associative processing on its design.

Briefly, the multi-associative and SPMD processing modes differ from the familiar SIMD and MIMD modes as follows. In multi-associative mode, the PE's execute a single instruction stream, but are arranged into disjoint groups, with each group able to locally broadcast values, and compute its own summary values in parallel with other groups. In SPMD the processors execute the same program with autonomous instruction pointers so that they can branch independently.



- Controls CAAPP and ICAP.
- Takes commands from SPA.
- Receives global summary info.

- Knowledge base, blackboard.

- 64 RISC processors (MIMD).
- Instantiation of schema strategies.
- Construction of scene interpretation.

- Top-down MIMD control of grouping.

- 64x64 (4K) Array of 16-bit processors.
- SPMD/MIMD operation.
- Executes grouping processes.

- Stores extracted image events.

- 512x512 (256K) Array of 1-bit processing elements.
- SIMD Associative / Multi-associative.
- Processes sensory data.

- Stores 15 seconds of imagery.

Figure 1. IUA Overview

3. Tradeoffs in the Development of the Low-Level Processor

Our analysis of low-level vision algorithms showed that the majority would best be served by a mesh-connected array, augmented with the features of an associative processor (i.e. global broadcast with local partial matches, activity control with global override, and dedicated response hardware) [Weems, 1984].

3.1 Neighbor Communication

One tradeoff is a four-way versus an eight-way mesh. We found that few algorithms take advantage of an eight-way mesh, and the increase in performance is small unless operations take place on eight inputs at once. Even then, the improvement does not justify the resultant cost of tripling the number of I/O pins on the processor chip and at circuit board boundaries.

In addition to local communication, several low-level vision algorithms require communication between processors that are spatially distant in the mesh. We chose to enhance the mesh itself, so that no additional connectivity is required between processors. This scheme is similar to those proposed by [Kumar, 1985], [Miller, 1988], and [Li, 1987], and is a generalization of the propagate operation in the CLIP-4 [Duff. 1978], and the "flash-through" mode of the ILLIAC III [McCormick, 1963]. In our scheme, any contiguous group of processors in the mesh can be connected by a bus. For example, each region in an image could be electrically isolated from its neighbors, allowing local broadcast and some/none operations to occur simultaneously in all regions (Figure 2). An important result is that maximum or minimum values can be determined within regions, which is used to label connected components. Our simulations show that this takes roughly 50 microseconds on a 512 by 512 array, assuming a 100 nanosecond cycle. This Coterie Network, as it is called, has many other uses, including matrix arithmetic, Fast Fourier Transform (FFT), convex hull computation, simulating a pyramid processor, etc.

3.2 Memory

The two obvious options for expanding processor memory are to add memory to the processor chip, or to use external memory. We anticipate chips with more than 64 (1024 is feasible) processors, and increased clock rate, which favors on-chip memory. However, we also saw the need for at least tens of thousands of bits per processor, which only off-chip memory could support. Our solution to this dilemma is to do both. Each processor contains an explicitly managed data cache on the chip. In our current implementation, this contains 320 bits, but the architecture provides for expansion to 1024 bits. Two pages of the cache also perform corner-turning (the transformation of bit-serial data into bit-parallel formats). The external memory is dual-ported with the intermediate-level processor, and is the primary data path between processing levels. The low-level, bit-serial data must therefore be "corner-turned" on its way to the backing store, so that the intermediate-level processor can work with it directly in bit-parallel formats. Each low-level processor has access to 32K bits of external memory. Backing store transfers take 16 instruction cycles per byte.

The corner-turning pages have eight-bit data paths, providing a factor of eight speed-up over the bit-serial data path for movement of data between locations in these pages. The wider data path is also useful for aligning mantissas in floating-point operations. The registers that control the Coterie Network switches are also attached to the 8-bit data path, allowing the entire network to be reconfigured with a single instruction.

3.3 Response and Activity Control

Traditional associative processor designs use a response flag to both control local activity and provide summary information to the central control. An opposite approach is to separate response from activity by providing a register for each. Both, split and combined activity, and response can be provided by allowing writes into either of the two registers, or both, simultaneously. This small change provided a 20 percent speed improvement in equality comparisons between a local value and a broadcast value. A similar change allows inequality tests to be performed in about 50% less time, by permitting the response register to be loaded with an operand at the same time that a result is stored in the activity register.

3.4 Response Count

Our analysis showed that counting processors with a specific bit set is frequently done in bursts. For example, summing a set of values in the array involves counting the ones in each bit position (each processor loads one bit at a time into its response register; a sum is developed by counting the bits in each position and scaling the counts appropriately before summing them). Another example, is creating a histogram of a set of data; which could require 256 counts for an 8-bit field (each processor compares its value to a broadcast value, the bucket number, and if the values match the processor sets its response bit, the table of counts then corresponds to all of the buckets of the histogram).

For real-time applications, a count must be developed very quickly. One technique proposed by Foster [Foster, 1971], uses a pyramid of adders. Within the processor chip, Foster's scheme is used to produce a response count at the end of each instruction cycle. A special instruction latches the count into a shift register so that it can output serially. The processors are able to overlap computation with output of the current count.

A custom VLSI chip was designed with 64 serial inputs, one serial output, and six parallel outputs. One cycle after the low order bits of a set of partial counts are input, the low order bit of the result appears at the serial output. The high order bits of the result appear in the parallel outputs. The process can be repeated to sum 64 inputs of any bit length with the low order portion of the result being shifted out serially and the high order six bits available in parallel. Two levels of the chips are cascaded to form a count for the entire array. Only 1.6 microseconds are required to count the response registers in an array of 262,144 processors, using 65 copies of a single custom chip. The same chip also provides the boolean summaries some/none, some/all, and exactly one responder. It can be thought of as a general purpose 64-input reduction unit.

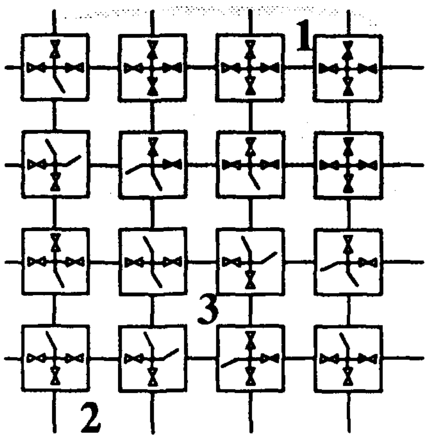


Figure 2. Coterie Network

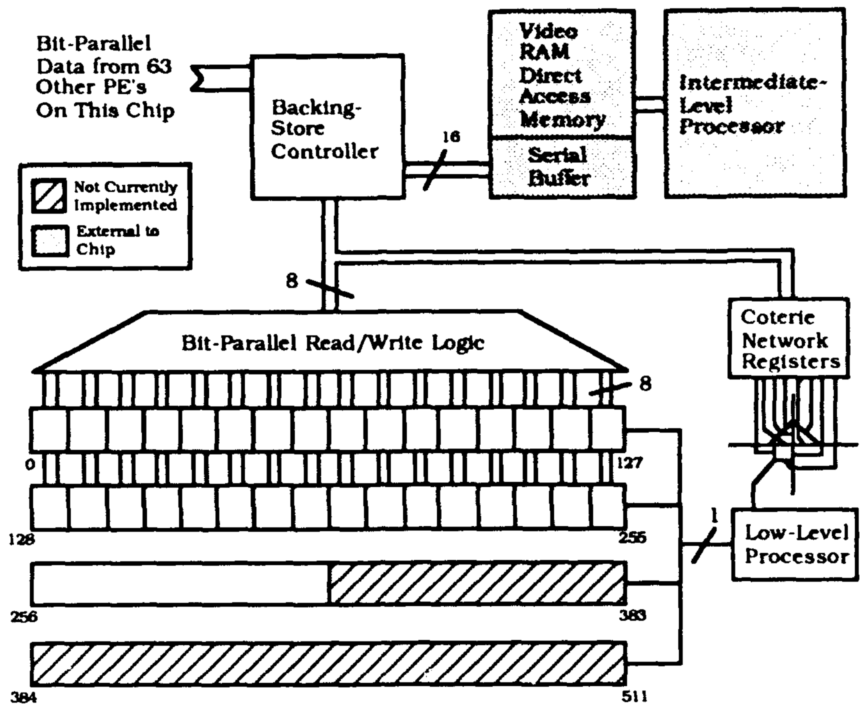

Bit-Parallel Data from 63 Other PE's On This Chip
Backing-Store Controller
Video RAM Direct Access Memory
Serial Buffer
Intermediate-Level Processor
16
Not Currently Implemented
External to Chip
8
Bit-Parallel Read/Write Logic
Coterie Network Registers
8
0
127
128
255
1
Low-Level Processor
256
383
384
511

Figure 3. Memory Architecture

3.5 Input/Output

Our original I/O scheme was similar to that used in the MPP [Batcher, 1980], with data shifted in from an array edge. With parallel mesh communication, the time required to fill a 512 by 512 array with an eight-bit image, assuming a 100 nanosecond clock is 410 microseconds. Initially, we felt that this would be fast enough, but later realized that even a pause of this length could interfere with real-time deadlines. Also, in a multisensory task, the amount of I/O is much more significant.

Another consideration is that vision systems occasionally fall behind and need to subsample the incoming stream of frames in order to catch up. It is also desirable to be able to retrieve previous frames. We were thus faced with redesigning the I/O after beginning to design the new chip, which greatly limited our options.

Our solution for the prototype is to associate an additional Video RAM (VRAM) with each processor chip, connected to the South edge of the chip's mesh network (Figure 4). A special instruction tri-states the North edge of the chip during I/O. Then the data in the serial buffer of the VRAM is shifted in from the South, and stored by the processors, using the corner-turning circuitry. To output to the VRAM, the Coterie Network is used to send data from the North edge of the chip to the South, where it is streamed into the VRAM's serial port. Transfers to and from this staging memory take only 8 microseconds for an array-sized 8-bit image. The memory is large enough to hold sixteen seconds of imagery, providing a reasonable time window.

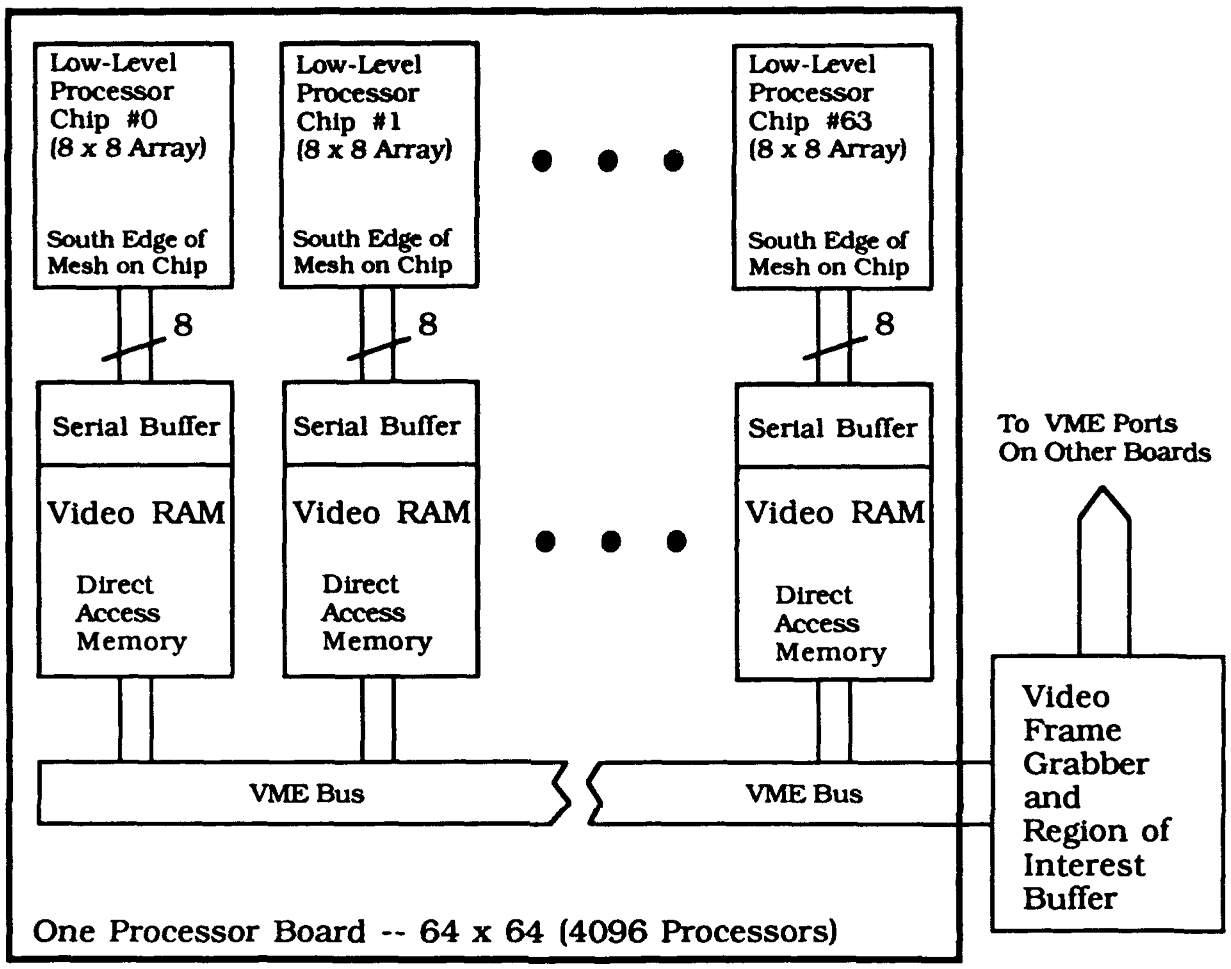

Low-Level Processor Chip #0 (8 x 8 Array)
South Edge of Mesh on Chip
Low-Level Processor Chip #1 (8 x 8 Array)
South Edge of Mesh on Chip
Low-Level Processor Chip #63 (8 x 8 Array)
South Edge of Mesh on Chip
8
8
8
Serial Buffer
Serial Buffer
Serial Buffer
To VME Ports On Other Boards
Video RAM
Direct Access Memory
Video RAM
Direct Access Memory
Video RAM
Direct Access Memory
Video Frame Grabber and Region of Interest Buffer
VME Bus
VME Bus
One Processor Board -- 64 x 64 (4096 Processors)

Figure 4. I/O Architecture.

The direct access portion of the VRAM is connected to a VME port, and appears as a block of memory in the VME address space. In the full-scale system, each processor card could have its own VME port so that parallel I/O to the staging memory can take place. In our prototype, the single VME port is connected to a smart frame grabber that moves data in and out through a region-of-interest buffer.

4. Design of the ICAP Interconnection Network

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

The ICAP connection network is used to set up a connection pattern between the N output ports of the processors and the N input ports of these same processors. The connection network can be programmed on-line, to make a direct link from the output port of any processor to the input port of one or more processors. We have built a custom VLSI chip, called the PARallel COmmunication Switch (PARCOS), which is capable of both point-to-point and broadcast communication, allowing the connection network to realize any of N^N mappings of its input ports onto its output ports. All of the processors can send and receive data on their links at the same time. These links can be changed by the ACU at any time.

The 64-input, 64-output connection network for the IUA prototype uses 2 stages of 32 x 32 PARCOS chips. The PARCOS chips are connected to make a 64 x 64 crossbar switch with broadcast capability as shown in Figure 5. A detailed discussion of the network can be found in [Rana, 1988]. While these chips have been constructed and tested, they have not yet been installed in the IUA prototype because neither the Hughes nor the UMass budgets included the cost for fabricating the circuit-boards to hold them. UMass assumed Hughes would build the boards because it was up to them to decide on the physical construction of the system, including circuit-board form factors (which are necessary to determine costs). Hughes assumed that UMass had budgetted construction of the board because UMass was designing the custom VLSI chips. Fortunately, the system was designed to also function without the communication network (communication via the host instead), so this merely affects communication performance rather than functionality. We still hope to eventually construct the boards and install them, using other funds, once the prototype is physically installed at UMass.

4.1 The Parallel Communication Switch

The PARCOS chip consists of a communication matrix of 32 bit-serial inputs and 32 bit-serial outputs, a control memory, a set of registers and associated read/write circuitry. The PARCOS chip organization is shown in Figure 6. Multiple PARCOS chips can be used to build larger connection networks, such as the 64 x 64 network in the IUA prototype.

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

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

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

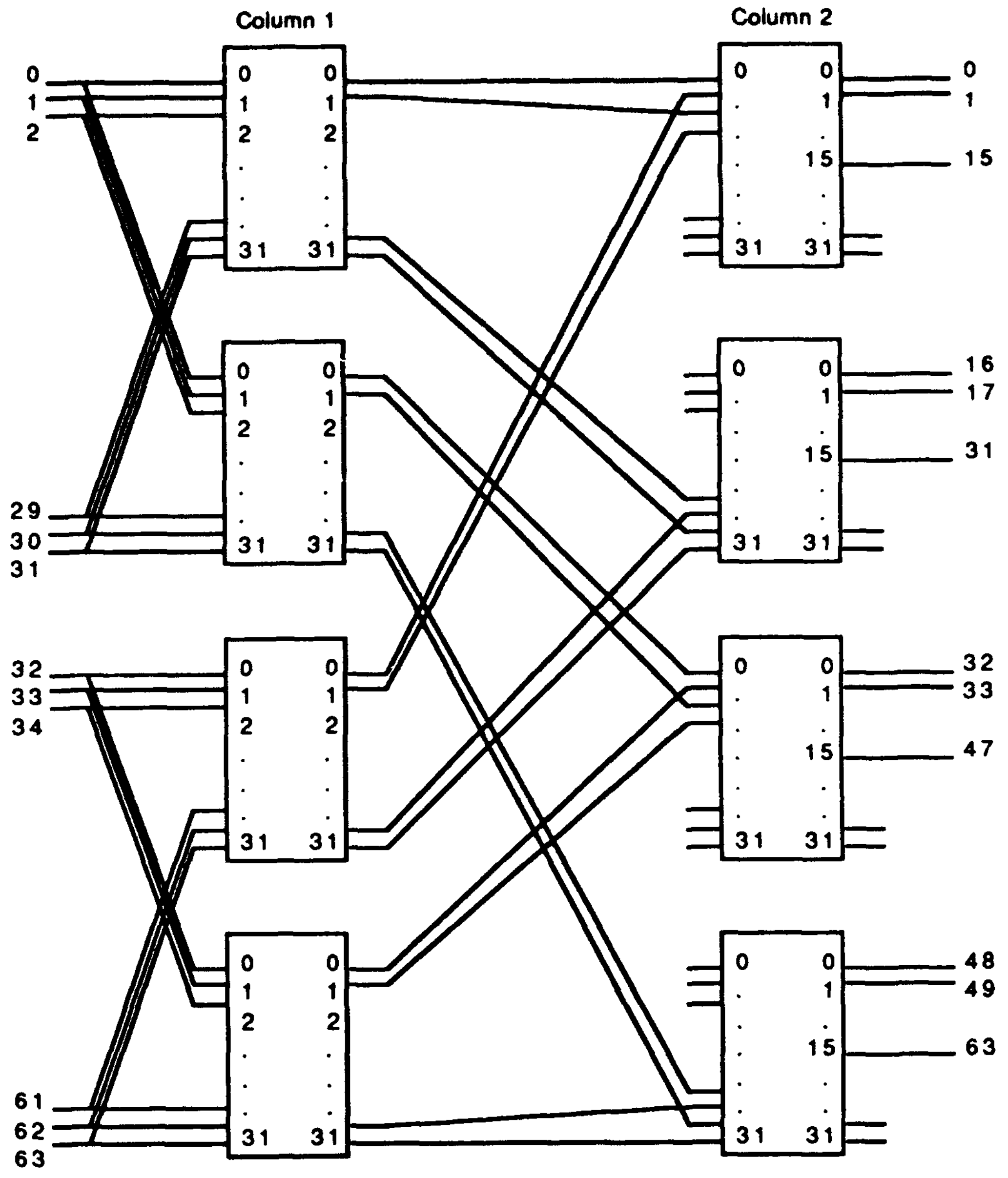


Figure 5. A 64 x 64 Network Built with PARCOS Chips.

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

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

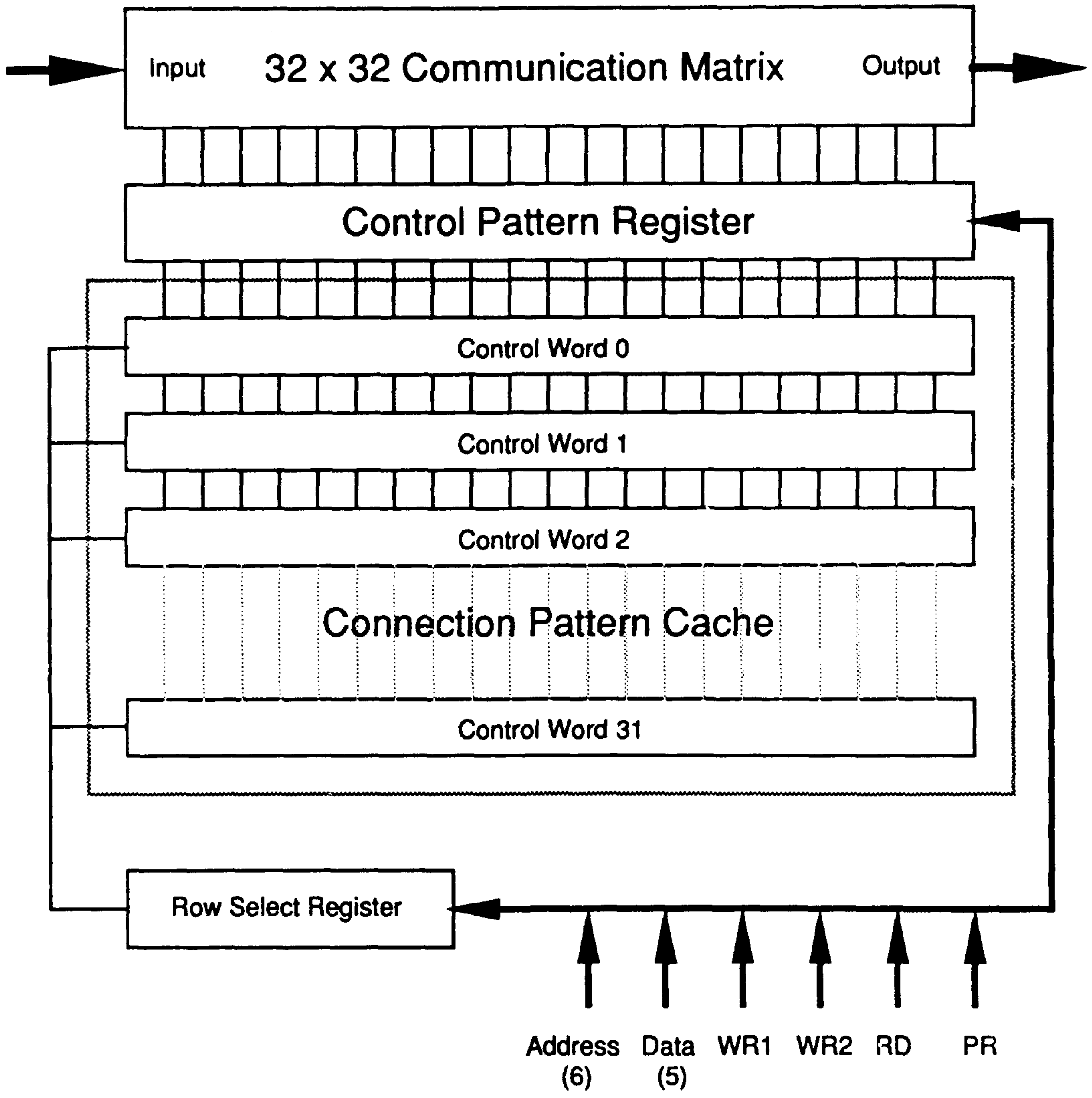


Figure 6. PARCOS Chip Organization

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

5. The IUA Simulator

The simulator for the Image Understanding Architecture provides a way of testing the design of, and developing the software for, the IUA. The simulator runs under X and is currently installed on Sun and DEC workstations and on a Sequent Symmetry multiprocessor. Several versions of the simulator exist on each machine and differ in the size of the IUA being simulated. The larger the IUA, the slower and larger is the simulation. The simulator supports IUA configurations with various numbers of Mother Boards as shown in the following table.

Mother Boards	1	4	16	64
CAAPP PEs	4096	16384	65536	262144
ICAP Processors	<= 64	<= 256	<= 1024	<= 4096

Even with the smallest complement of Mother Boards, a complete IUA is usually not simulated due to limitations on the real memory available on the host computer and the desire to avoid page faults when running simulations. (Even the smallest configuration of the IUA contains 42 MB of RAM.) The amounts of CAAPP-ICAP Shared Memory (CISM), ICAP-SPA Shared Memory (ISSM), and ICAP Data Memory (IDM) are thus limited to that needed by the problem being run on the simulator. For the same reasons, the ICAP Program Memory (IPM) is considered to be read-only so that it need not be duplicated for all the processors.

The simulator has been constructed in a modular fashion so that the various parts may be replaced easily for different needs such as allowing substitution of a 64 processor ICAP simulator module for a 16 processor ICAP simulator module when the primary simulation is at the ICAP level instead of at the CAAPP level. The user's view of the simulator is presented by the "user console," which contains separate windows such as a control panel, display window, and programming terminal.

The user console is a window which is roughly 600 rows by 800 columns in size, leaving sufficient room to view other windows (such as a command window or editor) on the screen. This display is split into a left and right side. The left side is the Control Panel and the right side is the Display Window.

The Control Panel contains displays of the 1-bit CAAPP registers arranged as blocks of 64 x 64 pixels with one pixel per processor. For the smallest simulator this is the complete set of processors. For larger versions of the simulator, this is a sub-window into the n by n array of CAAPP processors. Associated with each register display is a button. Clicking this button with the mouse causes the display to be shown enlarged in the Display Window. Other buttons in the Control Panel cause other displays to appear in the Display Window or bring up pop-up windows for special operations such as loading and saving files. Having all of the registers shown at the same time allows the programmer to see the state of the CAAPP processors arranged in correspondence to images stored in the array.

The Display Window consists of a foreground and background display. The background display is always the Programming Terminal. The foreground display may or may not be present and shows the display selected by clicking a button in the Control Panel. The foreground display leaves the bottom sixth of the Programming Terminal always visible. The Programming Terminal allows entry of commands and input to a running program. Output from a program can also be shown on the terminal.

The foreground displays include processing element (PE) registers, PE memory, a grey level display, coterie network display, ICAP display, and PE instruction display. All of the displays can be selected and manipulated by the user program running in the simulated ACU.

The PE register display presents a binary image indicating the status of one selected PE register from all PEs. The user can zoom in or out from a 2 by 2 processor display up to a display showing all the processors being simulated, even for a 512 x 512 simulation. Scroll bars on the sides indicate what portion of the complete array is being shown. Individual PE registers may be set or cleared by clicking with the mouse inside of the display window. The Control Panel simultaneously shows the current row-column processor location selected by the mouse.

The PE memory display shows one location in every PE Memory as a binary image. This display has the same functionality as the PE Register display.

The grey display shows a contiguous range of bits in PE Memory as a grey scale image. The user can zoom in or out from a 2 by 2 processor display up to a display showing all the processors being simulated. (Scroll bars on the sides indicate what portion of the complete array is being shown.) The user may select from 1 to 32 bits in the range to be displayed, although the actual screen representation may be limited by the available graphics hardware. The grey display may be changed to an inverse grey or false color mode.

The user may select a 3 by 3 pixel array with the mouse to be shown as numeric values in a pop-up window in either signed integer, unsigned integer, or IEEE floating point format. The location in PE memory or CISM memory that is sampled may be different from the location shown in the Grey Display, allowing an image to be used to guide exploration of other values that may not be visually informative when shown as an image. The Grey Display can be overlaid with red, green, and blue pixel maps. The overlay can be any of the PE registers or locations in PE memory.

The Coterie Display shows a graphic representation of the state of the switches anu the the electrical charge in the network. Currently, the Coterie Display is limited to 32 by 32 PEs. A particular PE can be dragged to the center of the display with the mouse. Scroll bars on the side indicate what portion of the complete network is being displayed. Using function keys, individual switches can be opened or closed in the Coterie Network.

The ICAP display shows the complete set of registers for one ICAP processor. Also shown are all of the registers on the same Daughter Board (except for the CAAPP PE registers). When this display is selected, a pop-up window appears which may be used to select a particular Daughter Board, and a range of locations in ICAP Data Memory (IDM), to be displayed. A range of locations in ICAP Program Memory (IPM), surrounding the current value of the program counter, is also shown. Up to four breakpoints may be set for the program running in the ICAP processors. Both the IDM range and the breakpoints may be selected using symbolic expressions.

The PE Instruction Display is a scrollable display of the last 2048 instructions sent to the CAAPP by the ACU. The instructions are shown both in hex and in symbolic form.

The Daughter Board Simulator simulates all of the PEs, CISM, ISSM, and glue logic. While the IUA is made up of multiple Daughter Boards, the Daughter Board simulator does not simulate them one at a time. Instead, for efficiency, the PEs are simulated as a vector of processors. At those places where the geometry of the IUA is apparent, the simulation applies a board-by-board approach. For example, the instruction "zero the Z registers" is done for all PEs in a tight loop while the backing-store operations are done board-by-board.

The Daughter Board simulator receives instructions in the same form that the real Daughter Boards receive instructions from the ACU. The instructions are sent on a simulated bus as 32 bit signals and are then decoded. This provides two benefits over a more tightly coupled scheme. First, the instruction stream on the bus is easily captured and can be used in exercising circuit boards under test. Second, using a bus allowed quick construction of versions of the simulator that could utilize a parallel processor. In fact, we use the same Daughter Board code for both the uniprocessor and multiprocessor implementations. A compile time switch is used to select the data partitioning parallel code which resides in only one subroutine. The parallel code is coupled with synchronizing check points that are no-ops in the non-parallel versions.

The computation of execution time is also simplified through the use of the simulated bus. Because the majority of instructions take a single clock cycle, an accumulator is used to count the instructions sent over the bus. An ACU overhead cost is also added as appropriate. (Micro-controller routines have lower per-instruction overhead.) For those operations such as backing-store read where the result is not available immediately, the code that simulates the individual instruction adds the worst case time to the accumulator. (In the multiprocessor version of the simulator, only one processor does the addition.) For the real machine, these variable length processes will be handled either by using feedback or by fixed time idle loops in the ACU for the worst case. For the simulator, we took the second approach since we did not want to simulate the backing-store finite state machine at the level needed to provide the correct timings using the feedback method.

The Coterie Network provided special problems in the simulation because it is really an analog circuit using electrical charge propagation. As the cost of an analog simulation of the network is prohibitive, we simulated the charge propagation digitally; permitting us to execute one cycle of 100 ns in approximately 18 milliseconds for a 512 by 512 PE simulator using 9 processors on the Sequent multiprocessor. For the DARPA Integrated IU benchmark, this would have otherwise required approximately 24 days of wall-clock time to run only one of the five test cases. An analysis of the problem showed that the configuration of the network was being changed infrequently with respect to the number of network operations performed. We thus modified the simulator to record a list of connected PEs whenever the network is reconfigured, and this information is used to accelerate the simulation of subsequent network operations using that configuration. The result is that the complete set of 5 IU benchmark test cases can be run in just two and one half days on the Sequent.

The Programming Terminal is an interactive interpreter that allows entry of commands and programs to directly manipulate the processing arrays. The IUA Simulator has been designed so that this module may be easily replaced by other modules. (Currently, the only module available is for interpreting the FORTH language.) A module for Lisp could be provided as well. The interface consists of input and output streams from the simulator, a procedure call for issuing instructions to the CAAPP bus, and other procedure calls for changing the displays.

We felt that it was very important to provide an interactive environment so that the edit, compile, test loop would be very fast. The programmer can rapidly prototype code interactively and then reimplement the tested algorithm as an "ACU Macro" if desired.

FORTH is a threaded language. A few simple constructs are combined into ever more powerful constructs. Each construct is called a word. FORTH was selected because its interpreter is small and fast. W: also had access to the source code for a FORTH implementation. New features such as floating point operations were built, in addition to interfaces to the IUA Simulator. FORTH provides a quick way of changing a program and re-trying it, or of just entering instructions. We provided a FORTH-based assembler for the CAAPP PE instructions so that the user is able to enter an instruction such as

```
A := B 'AND Y !!
```

and have it assembled and sent on the CAAPP bus. Each of the symbols (A, :=, B, etc) are FORTH words that place information on the FORTH stack. This information is processed by the FORTH word !! to produce the CAAPP bus instruction. The special words provided to interface with the simulator allow FORTH to control all aspects of the simulation and act as the ACU.

The low level processing portion of the DARPA Integrated IU benchmark was written mostly in FORTH with some of the simple ACU Macros such as ADD-FIELDS written in C. The major problem encountered with the use of FORTH was its flat name scoping, which prevented the FORTH code from being completely modular due to name conflicts (no vocabulary facility is available).

The following example is the definition of a FORTH word that adds two integer fields of the same length in PE memory.

```
( 1 ) : ADD-FIELDS ( length f1 f2 -- )      ( Add field f2 to field f1 )
( 2 )     Z := ZERO !!
( 3 )     2 ROLL 1 - 0 DO
( 4 )         1 PICK I + >R  X := R> A!
( 5 )         0 PICK I + DUP >R := X '+ R> A!
( 6 )     LOOP ;
```

Line 1 defines the word ADD-FIELDS. This word takes three arguments off of the FORTH stack. The top stack value is the PE Memory address of the second operand. The next stack value is the first operand/result field PE Memory address. The third value is the length of the fields in bits. This argument protocol is documented with the comment in line 1. Line 2 clears the Z (carry) register in all PEs. Line 3 is a FORTH indexed loop. The 2 ROLL picks up the length value from the FORTH stack. If this value were 8, then line 3 would be equivalent to 7 0 DO which would loop over the values 0, 1, 2, 3, 4, 5, 6, 7. The end of the loop is specified in line 6 which also ends the FORTH word. Line 4 generates the PE instruction "load the X register of the active PEs with the value from memory location f2 + I" where I is the loop index. The FORTH return stack is used as a temporary holding place for the value f2 + I. Line 5 generates the PE instruction

```
M[f1+I] := X + M[f1+I] A!
```

which causes the X register, the Z register, and memory location f1 + I to be added in all active PEs. The result is placed in memory location f1 + I and the carry goes to the Z register. For the FORTH statement

```
2 10 20 ADD-FIELDS
```

the following PE instructions would be issued to add a pair of 2-bit values at locations 10..11 and 20..21, with the result being stored back in locations 10..11:

```
Z := ZERO !!
X := M[20] A!
M[10] := X + M[] A!
X := M[21] A!
M[11] := X + M[] A!
```

The ICAP simulator is structured to simulate one instruction from the first ICAP processor, one instruction from the next and so forth. Because the ICAP processors run in MIMD mode, these instructions will probably be different. As we did not want to pay the price of a gate level simulator, we chose to implement a functional simulator for the TMS320C25.

Because the simulation is on an instruction level, code written to use timing loops is not valid as the simulator will not maintain synchronization between the various ICAP processors. A disadvantage of this approach is that the timings are not exact. Our timing model uses the average time for an instruction with data in on-chip memory and instructions in off-chip memory. Our experience shows that approximately 90 percent of the data references are to the on-chip memory. With the TMS320C25 there is a large benefit to using a small amount of contiguous data memory, which is due not only to the on-chip data memory, but also to the addressing modes supported.

The full IUA will have 4096 ICAP processors. Each one will have 128k bytes of data memory and 128k bytes of program memory. This is a gigabyte of memory and is far more than the real memory available on any of the machines we have used to run the simulator. In order to prevent page faults which would have drastically increased the elapsed time for any simulation, we reduced the number of ICAP processors and the amount of ICAP Data Memory (IDM) and ICAP Program Memory (IPM) available in the simulator. The IUA simulator can be easily re-configured as to the number of ICAPs being simulated (independently of the size of the CAAPP array) and the size of IDM. The IPM is shared as read-only memory among all the ICAP processors. Even though the ICAP processors run independently, they all have the same programs loaded in IPM; which restricts the programmer to writing code that is not self-modifying.

Because there is a single system clock, the CAAPP and ICAP instruction streams are in approximate synchronization with roughly a two to one execute rate. Therefore, the IUA simulator executes two CAAPP instructions and then one instruction for each ICAP processor. Since an ICAP instruction may take more than one cycle, a particular ICAP processor is held up if its clock shows more time than the CAAPP has used.

The ICAP code is loaded into IPM by the ACU. The ACU can write the same program in every ICAP IPM directly using the CAAPP bus. The ACU can also write directly into any off-chip IDM location. Because the bus is a one-to-many bus, all ICAPs receive the same data and programs. However it is also possible for the ACU to write a loader program into IPM that causes IPM to be loaded from ISSM by each individual ICAP processor. Since each ICAP processor sees a separate part of ISSM, each can be loaded with a different program. This mode of operating is not supported by the IUA simulator as there is only one IPM shared by every ICAP processor.

Code for the ICAP processors can be written in either C or Assembler. The IUA simulator provides a loader for either special absolute code or for the TI COFF loader text format.

While the full-scale IUA has one ACU and 64 SPA processors, the prototype hardware has one ACU and just one SPA. In the simulator, the ACU and SPA are the same computer and the simulator follows this simpler model regardless of how many PEs or ICAP processors are simulated. The ACU/SPA is the interactive module linked into the Programmers Terminal, which is currently the FORTH interpreter.

A separate module called the ACU Macros is also part of the ACU. The ACU Macros are procedures written in C for standard operations on the CAAPP such as ADD-FIELDS or FIND-GREATEST. On the real IUA, these macros will be stored in the micro-code memory of the Micro-controller that feeds the bus to the Daughter Boards. The ACU will request that a macro be executed with specified parameters, the micro-controller will execute the macro, substituting the parameters at the clock rate of the CAAPP while the ACU program sets up the next request. The approach taken in the simulator has been to identify which sequences of code should be in the micro-code memory, and what capabilities the micro-controller must have to efficiently execute those sequences. The ACU calls the macros via a special simulator interface procedure.

Also available to the user are "User macros". These macros allow the user to provide procedures written and compiled in C, which can either be special code for an application or candidates for an ACU Macro. Thus, the interactive Programmer's Terminal can still be used while taking advantage of the benefits of C. The cost to the user is that the simulator must be relinked whenever a change is made in the C code.

5.1 Libraries

We currently have an extensive library of arithmetic subroutines for the CAAPP, including byte, integer, and floating point arithmetic, and many of the standard transcendental functions. Thus, we have the basis for a compiler run-time library. We also have implemented many vision subroutines, including various convolutions, filters, edge-preserving, smoothing, convex hull, expand and contract morphological operators, connected component labelling, boundary tracing, windowed Hough transform, etc. In addition, we have implementations of the DARPA Integrated IU Benchmark, the Abingdon Cross Benchmark, and an optical ray-tracing application (as an example of a non-vision application).

For the ICAP, in addition to the compiler run-time library, we have a library that supports communication via the serial ports, and synchronization with the ACU. We also have an implementation of the Linda programming environment, originally developed at Yale (although it is very inefficient, as are most Linda implementations). From the DARPA Benchmark, we also have a model-matching algorithm that runs on the ICAP.

6. Image Understanding Benchmark

While traditional supercomputing benchmarks may be useful in estimating the performance of an architecture on some types of image processing tasks, those benchmarks have little relevance to the majority of the processing that takes place in a vision system [Duff, 1986]. 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 becau.. .hey 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. We begin by summarizing the work that has been done in the area of vision benchmarks to date, then we examine the DARPA IU Benchmark developed under this effort.

6.1 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 [Preston, 1986]. In that benchmark, an input image was specified that consisted 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, and with Gaussian noise added to the entire image. The goal of the exercise was 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 at the 1984 Multicomputer Workshop in Tanque Verde, Arizona, and many of the participants (including members of the UMass IUA group) spent a fairly lengthy session discussing problems with the benchmark and designing a new benchmark that it was hoped would solve those problems.

It was the perception of the Tanque Verde group that the major drawback of the Abingdon Cross was its lack of breadth. The problem required a reasonably small repertoire of image processing operations to construct a solution. The second concern of the group was that the specification did not constrain the a priori information that could be used to solve the problem. In theory, a valid solution would have been to simply draw the medial lines since their true positions were 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 become more 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 1 contains the list of problems that was developed. 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.

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

Table 1: Tanque Verde Benchmark Suite

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 the set of problems that made up the new benchmark was much smaller and easier to implement. Table 2 lists the problems that comprised the first DARPA Image Understanding Benchmark. 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 in [Rosenfeld, 1987]. The consensus of the workshop 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.

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

Table 2: Tasks from the First DARPA Image Understanding Benchmark

The workshop participants pointed out other shortcomings of the benchmark. Chief among these was that because it consisted of isolated tasks, the benchmark 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 for which it turned out there was no practical solution. The issue of having a ground truth, or known correct solution was considered very important, since it is difficult to compare the performance of two architectures when they produce different results. For example, is an architecture that performs a task in half the time of another really twice as powerful if the first machine's programmer used integer arithmetic, while the second machine was programmed to use floating point, and thus obtained significantly different results? 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 the first DARPA benchmark exercise was that a new benchmark should be developed that addresses the shortcomings of the preceding benchmarks. 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, every effort should be made to specify the benchmark in such a way as to minimize the opportunities for taking shortcuts in solving the problem. The task of constructing the new benchmark, to be called the Integrated Image Understanding Benchmark, was assigned to the vision research groups at the University of Massachusetts at Amherst, and the University of Maryland.

Following the 1986 meeting, a preliminary benchmark specification was drawn up and circulated among the DARPA image understanding community for comment. The benchmark specification was then revised, and a solution was programmed on a standard sequential machine. In creating the solution, several problems were discovered and the benchmark specification was modified to correct those problems. The programming of the solution was done by the group at the University of Massachusetts and the code was then sent to the group at the University of Maryland to verify its validity, portability, and quality. The group at Maryland also reviewed the solution to verify that it was general in nature and neutral with respect to any underlying architectural assumptions. The Massachusetts group developed a set of five test cases, and a sample parallel solution for a commercial multiprocessor.

In March of 1988, the benchmark was released, and made available from Maryland via network access, or by sending a blank tape to the group in Massachusetts. The benchmark release consisted of the sequential and parallel solutions, the five test cases, and software for generating additional test data. The benchmark specification was presented at the DARPA Image Understanding Workshop, the International Supercomputing Conference, and the Computer Vision and Pattern Recognition conference [Weems, 1988]. Over 25 academic and industrial groups, listed in Table 3, 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.

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*

Table 3: Distribution List for the Second DARPA Benchmark
* Indicates Results Presented at the Avon Workshop

The remainder of this section summarizes those results and recommendations, following a brief review of the benchmark task and the rationale behind its design.

6.2 Benchmark Task Overview

The overall task that is to be performed by this benchmark is the recognition of an approximately specified 2 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 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 rectangle of the same brightness or slightly greater depth is located directly behind it.

A set of models is provided that represent a collection of similar sculptures, and the recognition task involves identifying the model which best matches the object present in the scene. The models are only approximate representations of sculptures in that they allow for slight variations in component rectangle's sizes, orientations, depths, and the spatial relationships between them. A model is constructed as a tree structure where the links in the tree 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 in the tree describe the spatial relationships between that node and certain other nodes below it.

The scenario that the designers imagined in constructing the problem was a semi-rigid "mobile", with invisible links, viewed from above, with bits and pieces of other mobiles blowing through the scene. The state of the system is that previous processing has narrowed the range of potential matches to a few similar sculptures, and has oriented them to correspond with information

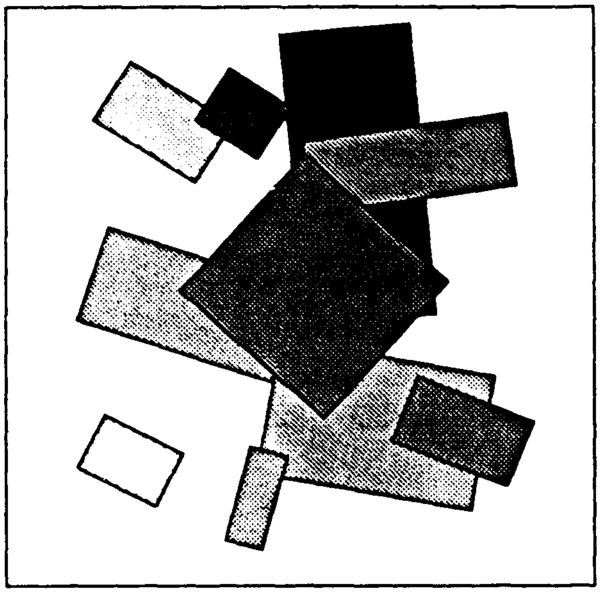


Figure 7: Intensity Image of Model Alone

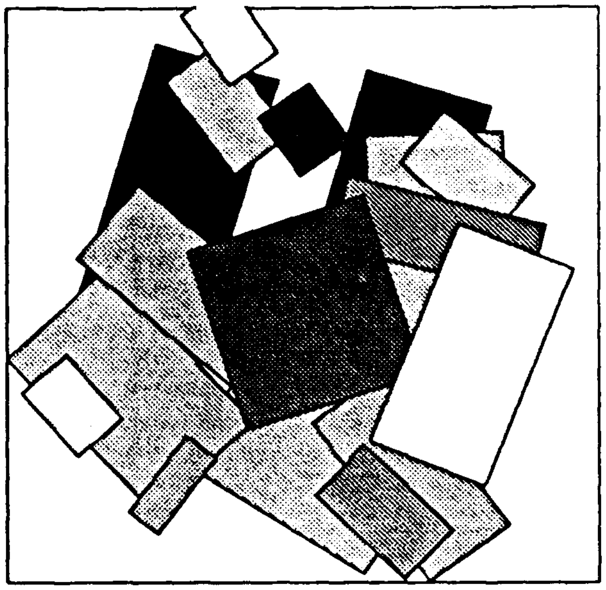


Figure 8: Image of Model with Clutter

extracted from a previous image. However, the objects in the scene have since moved, and a new set of images has been taken prior to completing the matching process. The system must make its final choice for a best match, and update the corresponding model with the positional information extracted from the latest images.

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.

A set of test images is created by first selecting one of the models in a set. The model is then rotated and translated as a whole, and its individual elements are then perturbed slightly. Next, a collection of spurious rectangles is created with properties that are 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-distribution noise is added to the depth image array.

Figure 7 shows an intensity image of a sculpture alone, and Figure 8 shows the sculpture with added clutter.

Processing in the benchmark begins with some low-level operations on the intensity and depth images, followed by some grouping operations on the intensity data that result in the extraction of candidate rectangles. The candidate rectangles are used to form partial matches with the stored models. For each model, it is possible that multiple hypothetical poses will be established. The benchmark then proceeds through the model poses, using the stored information to probe the depth and intensity images in a top-down manner. Each probe can be thought of as testing an hypothesis for the existence of a rectangle in a given location in the images. Rejection of an 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 at the hypothetical location, and an update of its representation in the model with new size, orientation, and position information. It is possible for the match strength to be as low as zero when there is no supporting evidence for the match and a lack of strong 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 match strength is selected as the best match, and an image is generated that highlights the model in the intensity image. Table 4 lists all of the steps that make up the complete benchmark task.

The benchmark specification requires that this set of steps be applied in implementing a solution. Furthermore, for each step, a recommended method is described that should be followed whenever possible. However, in recognition of the fact that some methods simply may not work, or will be extremely inefficient for a given architecture, implementors are permitted to substitute other methods for individual steps. When it is necessary for an implementation to differ from the specification, the implementor is expected to 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.

6.3 Benchmark Philosophy and Rationale

In writing an integrated image understanding benchmark, the goal is to create an interpretation 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 is 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 make use of as many of the tasks from the first DARPA benchmark as possible, in order to take advantage of the previous programming effort.

The job of the designers was thus to balance these conflicting goals and constraints in developing the benchmark scenario. One result is that the benchmark solution is neither the most direct, nor the most efficient method of solving the problem. However, making the solution more direct would have eliminated several of the 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 have required 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. Moreover, having taken a lesson from the Tanque Verde Suite, the benchmark design attempts to minimize the effort required of the participants, while maximizing the information obtained.

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. However, the purpose of the benchmark requires that the tasks and methods be well defined so that the results from different machines will be comparable. Otherwise, the results will include a significant factor that depends on the cleverness of the programmer. Thus, the benchmark specification requests that participants do not take shortcuts in the solution, and that they use the recommended methods whenever possible. It should be noted that the recommended methods are not always the most efficient techniques because they were chosen to be as widely implementable as possible. Thus, while the processing time for a given step or for the entire task may not be the best performance that a machine can muster, it will be comparable to the results from others. Participants were also encouraged to develop timings for more optimal solutions, in addition to the standard solution, if they so desired.

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 a much more extensive set of instrumentation than simple execution times. 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 that are to be reported include an estimate of the time spent implementing the benchmark, the number of lines of source code, the programming language used, 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.

6.4 Results and Analysis

Due to limitations of time and resources, only a few of the 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. Despite requests to groups that did not attend the workshop that they submit belated results to be included in this report, not one new benchmark report has been received. Thus, the results presented here are those that were provided by the workshop participants. In a few cases, the results have been updated, corrected, or amended since they were originally presented.

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	

Table 4: Steps that Compose the Integrated Image Understanding Benchmark

Care must be taken in comparing these results. For example, no direct comparison should be made between results obtained from actual execution and those that were derived from simulation [Carpenter, 1987]. No matter how carefully a simulation is carried out, it is never as accurate as direct executior. Likewise, no comparison should be made between results from a partial implementation and a complete one. The complete implementation must account for overhead involved in the interactions between subtasks, and even for the fact that the program is significantly larger than for a partial implementation. Consider that a set of subtasks might appear to be much faster than their counterparts in a complete implementation simply because less paging is required to keep the code in memory. It is also unwise to directly compare the raw timings, even for similar architectures, without considering the differences in technology between systems. For example, a system that executes a portion of the benchmark in half the time of another is not necessarily architecturally superior if it also has a clock rate that is twice as high or if it has twice as many processors.

In addition to the technical problems involved in making direct comparisons, there are other considerations that must be kept in mind. For example, every participant expressed the view that given more time to tune their implementation, the results for their architecture would improve considerably. What is impressive in many cases is not the raw speed increase obtained, but the increase with respect to the amount of effort required to obtain it. While this has more to do with the tools available for developing software for an architecture than with the architecture itself, it is still important in evaluating the overall usefulness of the system. Another major consideration is the ratio of cost to performance, since many applications can afford to sacrifice a small amount of performance in order to reduce the cost of the implementation. In other applications, the size or weight or power consumption of a system may be of greater importance than all-out speed. One of the purposes of this exercise has been merely to assemble as much of this data as possible so that the performance results can be evaluated with respect to the requirements of each potential application of an architecture.

Thus, in what follows, there is no single best architecture and there are no winners or losers. Each has its own unique merits and drawbacks, of which none are absolute. To play down the direct comparison of raw timings, the results for each architecture will be presented separately. The order of presentation is random, except that the sequential solution is presented first to provide a performance baseline, and then complete parallel implementations are presented, followed by partial implementations. Results that were based on theoretical estimations are not included in this report. The timings in all of the tables are in seconds, for the sake of consistency. Where a timing is zero, it indicates that the processing time was less than the resolution of the timing mechanism employed. Blanks in the tables indicate values that were omitted from the reports that were supplied by the implementors.

6.5 Sequential Solution

The sequential solution to the benchmark was developed in C on a Sun-3/160 workstation. The solution contains roughly 4600 lines of code, including comments. The implementation was designed for maximum portability and has been successfully recompiled on several different systems. The only portion that is system dependent is the actual result presentation step, which uses the graphics primitives provided for drawing on the workstation's screen. The implementation differs from the recommended method on the Connected Component Labelling step by using a standard sequential method for computing this well-defined function. The sequential method is designed to minimize array accesses and their corresponding index calculations, which is not a problem for array processors, but incurs a significant time penalty on a sequential machine.

Timings have been produced for the sequential code running on all five data sets, and on three different machine configurations. The configurations are 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 Timings have been produced for the sequential code running on all five data sets, and on three different of RAM. The extra RAM on the latter two machines did not affect performance, since the benchmark was able to run in 8MB without paging. The 3/260 was equipped with a Weitek floating-point co-processor, while the 3/160 used only the standard 68881 co-processor. Table 5 shows the results for the Sun-3/160, Table 6 shows the Sun-

Data Set	Sample		Test		Test2		Test3		Test4	
	User	System	User	System	User	System	User	System	User	System
Total	794.94	2.94	335.96	2.10	326.84	2.40	549.30	2.52	550.26	2.90
Overhead	4.02	1.06	4.06	0.88	4.50	1.14	4.60	1.04	4.58	0.94
Miscellaneous	2.24	0.04	2.18	0.04	2.16	0.06	2.12	0.02	2.10	0.02
Startup	0.02	0.00	0.04	0.00	0.02	0.04	0.00	0.02	0.02	0.00
Image input	0.60	0.68	-0.58	0.54	1.32	0.78	1.50	0.74	1.42	0.66
Image output	0.24	0.30	0.30	0.28	0.06	0.24	0.06	0.24	0.08	0.26
Model input	0.92	0.04	0.96	0.02	0.94	0.02	0.92	0.02	0.96	0.00
Label connected components	27.40	0.38	27.46	0.36	28.12	0.28	27.86	0.36	27.88	0.36
Rectangles from intensity	6.42	0.08	4.00	0.14	4.34	0.04	5.36	0.08	5.10	0.24
Miscellaneous	2.06	0.06	1.84	0.02	1.94	0.02	1.94	0.02	1.92	0.06
Trace region boundary	0.52	0.02	0.28	0.02	0.38	0.00	0.42	0.00	0.38	0.06
K-curvature	1.62	0.00	0.80	0.00	0.82	0.00	1.22	0.00	1.10	0.00
K-curvature smoothing	1.26	0.00	0.62	0.00	0.70	0.00	0.96	0.00	1.02	0.02
First derivative	0.46	0.00	0.22	0.02	0.24	0.00	0.28	0.02	0.22	0.02
Zero-crossing detection	0.26	0.00	0.06	0.00	0.04	0.00	0.18	0.00	0.24	0.02
Final corner detection	0.20	0.00	0.16	0.02	0.18	0.02	0.28	0.02	0.16	0.04
Count corners	0.00	0.00	0.00	0.02	0.02	0.00	0.00	0.02	0.00	0.00
Convex hull	0.02	0.00	0.00	0.02	0.00	0.00	0.02	0.00	0.04	0.00
Test for right angles	0.00	0.00	0.02	0.02	0.00	0.00	0.04	0.00	0.00	0.00
Final rectangle hypothesis	0.02	0.00	*0.00*	*0.00*	*0.02*	*0.00*	0.02	0.00	0.02	0.02
Median filter	246.06	0.60	118.62	0.26	92.58	0.28	90.70	0.22	90.66	0.24
Sobel	135.3	0.18	133.14	0.16	135.92	0.18	135.12	0.16	135.14	0.28
Initial graph match	24.4	0.06	24.94	0.06	26.02	0.02	68.30	0.14	67.48	0.14
Match data rectangles	0.14	0.00	0.10	0.02	0.08	0.02	0.26	0.04	0.24	0.00
Match links	0.22	0.00	0.06	0.00	0.08	0.00	0.74	0.00	0.58	0.02
Create probe list	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.02	0.00
Partial match	24.04	0.06	24.78	0.04	25.86	0.00	67.28	0.10	66.64	0.12
Match strength probes	24.02	0.06	24.74	0.02	25.82	0.00	66.64	0.10	65.82	0.12
Window selection	0.02	0.00	0.02	0.00	0.00	0.00	0.12	0.02	0.10	0.02
Classification and count	24.0	0.06	24.72	0.02	25.82	0.00	66.50	0.06	65.70	0.08
Match extension	326.54	0.50	11.46	0.12	18.72	0.20	202.58	0.32	204.68	0.44
Match strength probes	72.88	0.10	3.28	0.00	5.80	0.06	47.82	0.06	42.00	0.06
Window selection	0.08	0.00	0.00	0.00	0.00	0.00	0.08	0.02	0.10	0.00
Classification and count	72.80	0.10	3.28	0.00	5.80	0.06	47.72	0.02	41.88	0.06
Hough probes	253.32	0.38	8.16	0.12	12.84	0.12	153.76	0.22	161.98	0.36
Window selection	0.00	0.00	0.00	0.00	0.00	0.00	0.08	0.02	0.02	0.02
Hough transform	252.20	0.36	8.10	0.12	12.78	0.12	151.86	0.16	160.34	0.28
Edge peak detection	1.08	0.02	0.06	0.00	0.06	0.00	1.76	0.00	1.54	0.02
Rectangle parameter update	0.04	0.00	0.00	0.00	0.00	0.00	0.04	0.02	0.04	0.00
Result presentation	24.80	0.00	12.28	0.04	16.64	0.02	14.78	0.00	14.74	0.02
Best match selection	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.00	0.00
Image generation	24.80	0.00	12.28	0.04	16.64	0.02	14.76	0.00	14.74	0.02
Statistics										
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 5: Sun-3/160 Results

Data Set	Sample		Test		Test2		Test3		Test4	
	User	System	User	System	User	System	User	System	User	System
Total	293.42	5.96	130.48	2.06	116.96	2.56	191.38	3.38	192.38	3.20
Overhead	2.26	0.66	2.46	0.58	2.76	0.68	2.50	0.94	2.72	0.72
Miscellaneous	1.28	0.00	1.24	0.00	1.24	0.02	1.22	0.02	1.22	0.00
Startup	0.02	0.00	0.00	0.00	0.00	0.02	0.00	0.04	0.02	0.00
Image input	0.30	0.50	0.50	0.50	1.00	0.48	0.76	0.72	0.92	0.54
Image output	0.18	0.14	0.26	0.08	0.06	0.16	0.06	0.14	0.08	0.18
Model input	0.48	0.02	0.46	0.00	0.46	0.00	0.46	0.02	0.48	0.00
Label connected components	14.14	0.38	14.20	0.26	14.10	0.36	14.46	0.12	14.40	0.26
Rectangles from intensity	3.60	0.14	2.36	0.02	2.44	0.04	3.12	0.04	2.90	0.08
Miscellaneous	1.28	0.02	1.12	0.00	1.22	0.02	1.26	0.00	1.08	0.00
Trace region boundary	0.28	0.02	0.20	0.00	0.18	0.00	0.14	0.02	0.26	0.04
K-curvature	0.82	0.02	0.44	0.02	0.42	0.00	0.68	0.00	0.48	0.02
K-curvature smoothing	0.78	0.02	0.26	0.00	0.42	0.02	0.50	0.00	0.56	0.00
First derivative	0.20	0.02	0.16	0.00	0.10	0.00	0.18	0.00	0.26	0.00
Zero-crossing detection	0.02	0.02	0.04	0.00	0.06	0.00	0.18	0.00	0.14	0.00
Final corner detection	0.20	0.00	0.12	0.00	0.04	0.00	0.18	0.00	0.04	0.00
Count corners	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Convex hull	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.20	0.04	0.00
Test for right angles	0.00	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.04	0.00
Final rectangle hypothesis	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02
Median filter	112.50	1.20	59.86	0.42	42.64	0.46	42.64	0.34	42.72	0.54
Sobel	38.96	2.04	38.12	0.38	37.90	0.44	38.02	0.74	38.14	0.42
Initial graph match	6.10	0.06	6.06	0.02	6.38	0.20	17.02	0.30	16.80	0.14
Match data rectangles	0.08	0.00	0.06	0.00	0.04	0.00	0.14	0.02	0.12	0.00
Match links	0.10	0.00	0.04	0.00	0.04	0.00	0.30	0.00	0.26	0.00
Create probe list	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Partial match	5.92	0.06	5.96	0.02	6.30	0.20	16.58	0.28	16.42	0.14
Match strength probes	5.90	0.06	5.94	0.02	6.30	0.20	16.34	0.22	16.04	0.14
Window selection	0.00	0.00	0.00	0.00	0.00	0.00	0.10	0.02	0.02	0.00
Classification and count	5.90	0.06	5.94	0.02	6.30	0.18	16.24	0.18	16.02	0.10
Match extension	109.18	1.28	3.78	0.14	6.02	0.22	69.32	0.76	70.42	0.74
Match strength probes	17.54	0.02	0.78	0.00	1.40	0.00	11.60	0.06	10.20	0.10
Window selection	0.04	0.00	0.00	0.00	0.00	0.00	0.04	0.00	0.04	0.00
Classification and count	17.50	0.02	0.78	0.00	1.40	0.00	11.56	0.06	10.16	0.08
Hough probes	91.44	1.26	3.00	0.12	4.62	0.20	57.30	0.66	59.80	0.64
Window selection	0.04	0.00	0.00	0.00	0.00	0.00	0.04	0.02	0.02	0.00
Hough transform	90.64	1.24	2.98	0.12	4.60	0.20	56.40	0.64	59.00	0.62
Edge peak detection	0.76	0.02	0.02	0.00	0.02	0.00	0.82	0.00	0.78	0.02
Rectangle parameter update	0.00	0.00	0.00	0.00	0.00	0.00	0.04	0.00	0.00	0.00
Result presentation	6.68	0.00	3.64	0.00	4.72	0.00	4.30	0.20	4.28	0.02
Best match selection	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Image generation	6.68	0.00	3.64	0.00	4.72	0.00	4.30	0.02	4.28	0.02
Statistics										
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 6: Sun-3/260 Results

Data Set	Sample		Test		Test2		Test3		Test4	
	User	System	User	System	User	System	User	System	User	System
Total	117.21	3.80	40.19	2.45	38.88	2.06	78.41	2.64	80.15	2.69
Overhead	2.49	1.85	2.34	1.58	2.43	1.36	2.62	1.46	2.66	1.45
Miscellaneous	1.23	1.20	1.17	0.81	1.24	0.70	1.45	0.77	1.43	0.74
Startup	0.02	0.03	0.00	0.05	0.03	0.02	0.01	0.05	0.01	0.06
Image input	0.33	0.48	0.27	0.58	0.33	0.47	0.35	0.46	0.38	0.47
Image output	0.10	0.11	0.12	0.10	0.05	1.11	0.05	0.10	0.09	0.09
Model input	0.52	0.02	0.50	0.02	0.50	0.04	0.50	0.04	0.49	0.04
Label connected components	4.39	0.35	4.29	0.27	4.31	0.23	4.36	0.26	4.33	0.28
Rectangles from intensity	1.01	0.09	0.68	0.00	0.67	0.04	0.86	0.10	0.87	0.04
Miscellaneous	0.31	0.05	0.32	0.00	0.27	0.02	0.33	0.05	0.32	0.02
Trace region boundary	0.06	0.01	0.04	0.00	0.04	0.01	0.04	0.00	0.03	0.00
K-curvature	0.21	0.00	0.05	0.00	0.11	0.00	0.08	0.00	0.08	0.00
K-curvature smoothing	0.22	0.00	0.16	0.00	0.15	0.00	0.21	0.01	0.22	0.00
First derivative	0.12	0.00	0.09	0.00	0.06	0.00	0.14	0.00	0.08	0.00
Zero-crossing detection	0.04	0.01	0.01	0.00	0.00	0.01	0.02	0.00	0.04	0.00
Final corner detection	0.04	0.01	0.01	0.00	0.03	0.00	0.02	0.02	0.06	0.00
Count corners	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02
Convex hull	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.00	0.00
Test for right angles	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.01	0.00
Final rectangle hypothesis	0.01	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.00
Median filter	30.33	0.20	14.47	0.17	11.14	0.16	11.16	0.14	11.15	0.19
Sobel	11.21	0.95	11.26	0.17	11.17	0.10	11.11	0.30	11.15	0.30
Initial graph match	3.41	0.01	3.36	0.10	3.53	0.01	10.01	0.09	9.83	0.11
Match data rectangles	0.03	0.00	0.00	0.03	0.02	0.00	0.05	0.01	0.04	0.02
Match links	0.07	0.00	0.01	0.01	0.02	0.00	0.22	0.01	0.18	0.00
Create probe list	0.03	0.00	0.02	0.00	0.01	0.00	0.12	0.00	0.12	0.01
Partial match	3.28	0.01	3.33	0.06	3.48	0.01	9.62	0.07	9.49	0.08
Match strength probes	3.27	0.10	3.33	0.60	3.47	0.01	9.44	0.07	9.30	0.08
Window selection	0.00	0.00	0.01	0.00	0.00	0.00	0.01	0.00	0.04	0.01
Classification and count	3.15	0.00	3.23	0.06	3.38	0.01	8.85	0.05	8.65	0.02
Match extension	60.98	0.26	2.06	0.12	3.35	0.08	36.18	0.23	38.10	0.26
Match strength probes	9.89	0.02	0.45	0.00	0.79	0.00	6.63	0.02	6.06	0.02
Window selection	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.01	0.01	0.00
Classification and count	9.60	0.00	0.44	0.00	0.78	0.00	6.12	0.00	5.56	0.02
Hough probes	50.99	0.21	1.61	0.12	2.56	0.08	29.32	0.20	31.77	0.22
Window selection	0.03	0.00	0.00	0.00	0.01	0.00	0.09	0.01	0.07	0.00
Hough transform	50.65	0.12	1.60	0.11	2.54	0.07	28.86	0.08	31.32	0.12
Edge peak detection	0.15	0.00	0.01	0.00	0.01	0.00	0.24	0.02	0.21	0.00
Rectangle parameter update	0.03	0.01	0.00	0.00	0.00	0.00	0.03	0.01	0.06	0.00
Result presentation	3.37	0.01	1.67	0.00	2.24	0.00	2.07	0.00	2.02	0.00
Best match selection	0.06	0.00	0.02	0.00	0.02	0.00	0.10	0.00	0.04	0.00
Image generation	3.31	0.01	1.65	0.00	2.22	0.00	1.97	0.00	1.98	0.00
Statistics										
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	

Table 7: Sun-4/260 Results

3/260 results, and Table 7 gives the execution times for the Sun-4/260. The timings were obtained with the standard system clock utility which has a resolution of 20 milliseconds on the Sun-3 systems, and 10 milliseconds on the Sun-4.

6.6 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, buses 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. The FX-80 differs from the older FX-8 primarily in that the computational elements are significantly faster.

Alliant was able to implement the benchmark on the FX-80 in roughly one programmer-week. The programmer who built the implementation had no experience in vision and, in many cases, did not even 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 the new version of the code. Alliant provided results for five configurations of the FX-80, with 1, 2, 4, 6, and 8 computational elements. In order to save space, only two of the configurations are represented here. Table 8 shows the execution times for a single FX-80 computational element, and Table 9 shows the results for an FX-80 with eight elements. Another point that was noted by Alliant is that the C compiler is a new product and does not yet provide as great a degree of optimization as their FORTRAN compiler (a difference of up to 50% in some cases). They expect to see significantly better performance with later releases of the product.

6.7 Image Understanding Architecture

Because the IUA is still under construction, the simulator was used to develop the benchmark implementation. 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 benchmark implementation. The implementation was then transported to a full-scale IUA simulator running on a Sequent Symmetry multiprocessor. Table 10 presents the results from the IUA simulations with a resolution of one instruction time (0.1 microsecond). There are several points to note about these results. Because the processing of different steps can be overlapped in the different processing levels, the sum of the individual step timings does not always equal the total time for a segment of the benchmark. Some of the individual timings represent average execution times, since the intermediate level processing takes place asynchronously and individual processes can 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. However, the time for a step such as match extension takes into account the span of time required to complete all of the subsidiary match-strength probes. 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-

Data Set	Sample		Test		Test2		Test3		Test4	
	User	System	User	System	User	System	User	System	User	System
Total	204.858	2.531	102.700	1.861	93.311	1.828	136.759	3.049	139.130	3.032
Overhead	7.968	0.776	7.925	0.777	7.897	0.775	7.900	0.764	7.895	0.763
Miscellaneous	0.627	0.030	0.585	0.033	0.559	0.033	0.554	0.030	0.554	0.031
Startup	0.030	0.031	0.029	0.033	0.029	0.031	0.029	0.032	0.029	0.029
Image input	5.692	0.515	5.691	0.051	5.691	0.505	5.697	0.509	5.690	0.504
Image output	1.039	0.175	1.039	0.179	1.038	0.183	1.039	0.171	1.040	0.177
Model input	0.580	0.021	0.058	0.017	0.580	0.018	0.580	0.017	0.580	0.019
Label connected components	16.917	0.268	16.830	0.258	16.800	0.253	16.948	0.247	16.930	0.259
Rectangles from intensity	2.760	0.590	1.791	0.267	1.874	0.252	2.312	0.681	2.286	0.643
Miscellaneous	1.005	0.231	0.928	0.097	0.931	0.094	0.986	0.255	0.983	0.239
Trace region boundary	0.312	0.078	0.172	0.021	0.183	0.019	0.255	0.062	0.221	0.054
K-curvature	0.592	0.037	0.287	0.017	0.308	0.017	0.438	0.045	0.432	0.045
K-curvature smoothing	0.362	0.037	0.176	0.018	0.188	0.017	0.269	0.045	0.264	0.044
First derivative	0.158	0.037	0.077	0.017	0.082	0.016	0.119	0.045	0.117	0.043
Zero-crossing detection	0.170	0.037	0.076	0.017	0.099	0.017	0.135	0.045	0.133	0.043
Final corner detection	0.135	0.042	0.060	0.022	0.069	0.022	0.103	0.051	0.101	0.049
Count corners	0.006	0.037	0.003	0.017	0.002	0.017	0.007	0.044	0.006	0.042
Convex hull	0.013	0.026	0.006	0.017	0.006	0.017	0.015	0.042	0.015	0.040
Test for right angles	0.006	0.013	0.005	0.011	0.004	0.009	0.009	0.022	0.008	0.021
Final rectangle hypothesis	0.003	0.013	0.003	0.011	0.002	0.009	0.006	0.022	0.005	0.021
Median filter	77.294	0.170	43.652	0.160	31.886	0.163	31.919	0.154	31.880	0.166
Sobel	26.147	0.001	26.079	0.001	26.063	0.001	26.128	0.001	26.129	0.001
Initial graph match	2.458	0.088	2.397	0.063	2.569	0.055	7.117	0.368	7.011	0.373
Match data rectangles	0.067	0.023	0.051	0.012	0.046	0.014	0.129	0.047	0.111	0.041
Match links	0.067	0.002	0.024	0.004	0.022	0.004	0.262	0.013	0.214	0.023
Create probe list	0.002	0.001	0.002	0.001	0.002	0.001	0.005	0.001	0.006	0.003
Partial match	2.321	0.062	2.320	0.046	2.499	0.036	6.722	0.307	6.680	0.307
Match strength probes	2.305	0.045	2.303	0.032	2.486	0.024	6.502	0.228	6.429	0.229
Window selection	0.009	0.032	0.003	0.011	0.002	0.008	0.020	0.076	0.020	0.077
Classification and count	2.299	0.015	2.298	0.011	2.482	0.008	6.471	0.076	6.397	0.076
Match extension	68.025	0.385	2.149	0.083	3.817	0.091	42.243	0.600	44.806	0.584
Match strength probes	7.139	0.096	0.311	0.005	0.568	0.008	4.600	0.168	4.216	0.155
Window selection	0.009	0.032	0.000	0.002	0.001	0.003	0.15	0.056	0.014	0.052
Classification and count	7.125	0.032	0.310	0.002	0.566	0.003	4.576	0.056	4.193	0.052
Hough probes	60.754	0.202	1.833	0.068	3.241	0.071	37.330	0.301	40.320	0.312
Window selection	0.008	0.030	0.001	0.002	0.001	0.003	0.014	0.051	0.014	0.051
Hough transform	60.259	0.082	1.806	0.061	3.210	0.061	36.650	0.097	39.604	0.110
Edge peak detection	0.474	0.031	0.026	0.002	0.030	0.003	0.642	0.050	0.681	0.050
Rectangle parameter update	0.008	0.030	0.000	0.002	0.001	0.003	0.015	0.051	0.014	0.051
Result presentation	3.269	0.002	1.860	0.002	2.388	0.002	2.177	0.002	2.174	0.002
Best match selection	0.003	0.001	0.001	0.001	0.001	0.001	0.004	0.001	0.002	0.001
Image generation	3.266	0.001	1.859	0.001	2.387	0.001	2.174	0.001	2.172	0.001
Statistics										
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		1	
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	

Table 8: Alliant FX-80 Single Processor Results

Data Set	Sample		Test		Test2		Test3		Test4	
	User	System	User	System	User	System	User	System	User	System
Total	57.177	2.935	31.056	2.082	30.872	2.043	50.357	3.577	50.153	3.467
Overhead	7.940	0.847	7.903	0.825	7.897	0.813	7.891	0.820	7.899	0.822
Miscellaneous	0.601	0.042	0.558	0.039	0.558	0.039	0.553	0.041	0.560	0.058
Startup	0.030	0.056	0.029	0.047	0.029	0.042	0.029	0.043	0.029	0.033
Image input	5.690	0.549	5.695	0.541	5.691	0.532	5.690	0.542	5.690	0.536
Image output	1.039	0.173	1.040	0.172	1.038	0.177	1.039	0.173	1.039	0.173
Model input	0.580	0.023	0.580	0.021	0.580	0.017	0.580	0.017	0.580	0.017
Label connected components	6.930	0.295	6.864	0.272	6.849	0.270	6.979	0.273	6.992	0.272
Rectangles from intensity	2.776	0.686	1.799	0.314	1.882	0.295	2.329	0.785	2.309	0.751
Miscellaneous	1.010	0.277	0.931	0.120	0.934	0.113	0.994	0.303	0.990	0.290
Trace region boundary	0.312	0.084	0.172	0.023	0.183	0.022	0.227	0.071	0.224	0.063
K-curvature	0.594	0.042	0.287	0.020	0.308	0.019	0.438	0.051	0.433	0.049
K-curvature smoothing	0.364	0.042	0.176	0.019	0.189	0.019	0.270	0.052	0.267	0.050
First derivative	0.159	0.042	0.077	0.019	0.083	0.019	0.120	0.051	0.120	0.050
Zero-crossing detection	0.171	0.049	0.077	0.020	0.100	0.019	0.136	0.051	0.135	0.050
Final corner detection	0.136	0.048	0.060	0.028	0.070	0.025	0.103	0.057	0.130	0.055
Count corners	0.007	0.041	0.003	0.019	0.003	0.019	0.008	0.050	0.007	0.052
Convex hull	0.014	0.030	0.007	0.019	0.007	0.019	0.016	0.047	0.016	0.045
Test for right angles	0.006	0.016	0.005	0.013	0.004	0.010	0.010	0.025	0.009	0.023
Final rectangle hypothesis	0.004	0.015	0.003	0.013	0.002	0.010	0.007	0.026	0.005	0.023
Median filter	9.890	0.223	5.637	0.220	4.111	0.212	4.110	0.214	4.109	0.209
Sobel	3.798	0.001	3.789	0.001	3.787	0.001	3.795	0.001	3.795	0.001
Initial graph match	2.455	0.123	2.399	0.094	2.569	0.086	7.130	0.485	7.014	0.459
Match data rectangles	0.068	0.048	0.052	0.028	0.046	0.033	0.131	0.102	0.112	0.083
Match links	0.068	0.004	0.024	0.009	0.022	0.009	0.263	0.030	0.213	0.020
Create probe list	0.002	0.001	0.002	0.001	0.002	0.001	0.005	0.001	0.006	0.004
Partial match	2.317	0.070	2.322	0.055	2.499	0.043	6.732	0.351	6.682	0.351
Match strength probes	2.301	0.050	2.304	0.037	2.485	0.027	6.509	0.259	6.429	0.263
Window selection	0.004	0.017	0.004	0.012	0.002	0.009	0.023	0.087	0.025	0.087
Classification and count	2.294	0.017	2.298	0.012	2.482	0.009	6.473	0.085	6.390	0.087
Match extension	20.105	0.455	0.786	0.107	1.376	0.122	15.926	0.739	15.845	0.702
Match strength probes	7.121	0.111	0.311	0.006	0.567	0.009	4.609	0.195	4.219	0.185
Window selection	*0.010*	*0.037*	*0.001*	*0.002*	0.001	0.003	0.019	0.065	0.016	0.065
Classification and count	7.105	0.037	0.310	0.002	0.565	0.003	4.580	0.066	4.193	0.060
Hough probes	12.847	0.243	0.468	0.086	0.799	0.099	10.996	0.378	11.350	0.366
Window selection	0.008	0.033	0.001	0.002	0.001	0.003	0.014	0.057	0.014	0.057
Hough transform	12.353	0.110	0.441	0.078	0.767	0.086	10.315	0.151	10.629	0.140
Edge peak detection	0.472	0.034	0.026	0.002	0.030	0.003	0.645	0.057	0.682	0.057
Rectangle parameter update	0.009	0.033	0.000	0.002	0.001	0.003	0.013	0.056	0.014	0.057
Result presentation	3.265	0.002	1.859	0.002	2.382	0.002	2.178	0.002	2.173	0.002
Best match selection	0.003	0.001	0.001	0.001	0.001	0.001	0.004	0.001	0.002	0.00
Image generation	3.262	0.001	1.858	0.001	2.381	0.001	2.174	0.001	2.171	0.001
Statistics										
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		1	
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	

Table 9: Alliant FX-80 Results with Eight Processors

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
Miscellaneous	0.0092279	0.0092279	0.0092279	0.0092279	0.0092279
Startup	0.0038682	0.0038682	0.0038682	0.0038682	0.0038682
Image input	0.0000020	0.0000020	0.0000020	0.0000020	0.0000020
Image output	0.0000020	0.0000020	0.0000020	0.0000020	0.0000020
Model input	0.0008302	0.0008302	0.0008302	0.0008302	0.0008302
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
Miscellaneous	0.0003227	0.0002421	0.0002010	0.0006216	0.0002421
Trace region boundary	0.0033792	0.0015472	0.0018672	0.0010912	0.0012832
K-curvature	0.0038256	0.0019936	0.0023136	0.0015376	0.0017296
K-curvature smoothing	0.0005525	0.0005525	0.0005525	0.0005525	0.0005525
First derivative	0.0003777	0.0003777	0.0003777	0.0003777	0.0003777
Zero-crossing detection	0.0000108	0.0000108	0.0000108	0.0000108	0.0000108
Final corner detection	0.0000118	0.0000118	0.0000118	0.0000118	0.0000118
Count corners	0.0000020	0.0000020	0.0000020	0.0000020	0.0000020
Convex hull	0.0036694	0.0019109	0.0015290	0.0025947	0.0026463
Test for right angles	0.0006122	0.0006009	0.0005906	0.0006421	0.0006421
Final rectangle hypothesis	0.0067877	0.0067877	0.0078821	0.0067877	0.0064229
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	0.0121876	0.0076429	0.0066834	0.1124236	0.0822296
Match data rectangles	0.0029096	0.0015672	0.0013264	0.0134885	0.0106136
Match links	0.0088872	0.0056950	0.0049762	0.0985542	0.0712324
Create probe list	0.0000968	0.0001299	0.0001130	0.0009252	0.0008618
Partial match	0.0033786	0.0077033	0.0068704	0.1828976	0.1534418
Match strength probes	0.0009275	0.0011460	0.0012285	0.0025175	0.0212640
Window selection	0.0002100	0.0003000	0.0002700	0.0005700	0.0004800
Classification and count	0.0001043	0.0001490	0.0001341	0.0002831	0.0002384
Match extension	0.0300650	0.0017674	0.0024856	0.0899214	0.1277396
Match strength probes	0.0026500	0.0001146	0.0004095	0.0543250	0.0071766
Window selection	0.0006000	0.0000300	0.0000900	0.0012300	0.0016200
Classification and count	0.0002980	0.0000149	0.0000447	0.0006109	0.0008046
Hough probes	0.0068430	0.0003251	0.0005092	0.0084591	0.0109868
Window selection	0.0000675	0.0000045	0.0000090	0.0001755	0.0002385
Hough transform	0.0053010	0.0002223	0.0003036	0.0044499	0.0053477
Edge peak detection	0.0011745	0.0000783	0.0001566	0.0030537	0.0041499
ectangle parameter update	0.0003000	0.0000200	0.0000400	0.0007800	0.0010600
Result presentation	0.0022826	0.0009452	0.0011944	0.0029768	0.0029766
Best match selection	0.0000404	0.0000403	0.0000405	0.0000406	0.0000397
Image generation	0.0022352	0.0009185	0.0011396	0.0029464	0.0029464
Statistics					
Connected components	134	35	34	114	100
Right angles extracted					
Rectangles detected	31	23	19	60	55
Depth pixels > threshold					
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

Table 10: Image Understanding Architecture Results

level processor was also idle roughly 50% of the time while awaiting requests for top-down probes from the intermediate level.

6.8 Aspex ASP

The Associative String Processor (ASP) is being built by the University of Brunel and Aspex Ltd. in England [Lea, 1988]. 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 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 (gives approximately an 18% increase overall).

Because the system is still under construction, a software simulator was used to implement and execute the benchmark. 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 11 lists the benchmark results for the ASP. Timings were not provided for several of the steps in the model matching portion of the benchmark, possibly because a different method was used. Startup and model input times were not listed separately, perhaps because those operations are done outside of the simulation. The miscellaneous time under overhead accounts for the input and output of several intermediate images. The miscellaneous 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. No indication was given on whether any data rearrangement took place as part of these I/O operations.

6.9 Sequent Symmetry 81

The Sequent Computer Systems Symmetry 81 multiprocessor consists of multiple Intel 80386 microprocessors, running at 16.5 MHz, connected via shared bus to a large shared memory. The particular 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 test system also contained the older A-model caches, which induce a considerably greater level of traffic on the shared bus than the newer B-model caches. An improvement of 30 to 50 percent in the overall performance is possible with the new caching system. Sequent was to have provided timings for a system with the improved cache, but they have not yet done so. The timings presented in Table 12 were obtained by the benchmark developers at UMass as part of their effort to ensure the portability of the benchmark to different systems.

About a month was spent developing the parallel implementation for the Sequent. The programmer who did the work was familiar with the benchmark, but had no previous experience with the

Data Set	Sample	Test	Test2	Test3	Test4
Total	0.1307200	0.0359600	0.0398100	0.1130700	0.1188200
Overhead	0.0008200	0.0008200	0.0008000	0.0008000	0.0008000
Miscellaneous	0.0002560	0.0002560	0.0002560	0.0002560	0.0002560
Startup					
Image input	0.0000512	0.0000512	0.0000512	0.0000512	0.0000512
Image output	0.0000512	0.0000512	0.0000512	0.0000512	0.0000512
Model input					
Label connected components	0.0392000	0.0228000	0.0228000	0.0348000	0.0313000
Rectangles from intensity	0.0033100	0.0029200	0.0028800	0.0031900	0.0033500
Miscellaneous	0.0000761	0.0000860	0.0000842	0.0000795	0.0000734
Trace region boundary	0.0000047	0.0000047	0.0000047	0.0000047	0.0000047
K-curvature	0.0007800	0.0007800	0.0007800	0.0007800	0.0007800
K-curvature smoothing	0.0004500	0.0004500	0.0004500	0.0004500	0.0004500
First derivative	0.0000320	0.0000320	0.0000320	0.0000320	0.0000320
Zero-crossing detection	0.0000045	0.0000045	0.0000045	0.0000045	0.0000045
Final corner detection	0.0000018	0.0000018	0.0000018	0.0000018	0.0000018
Count corners	0.0000400	0.0000380	0.0000380	0.0000530	0.0000380
Convex hull	0.0003300	0.0002820	0.0002820	0.0003300	0.0003300
Test for right angles	0.0008800	0.0008400	0.0008400	0.0009500	0.0009200
Final rectangle hypothesis	0.0004500	0.0003800	0.0002900	0.0007600	0.0007000
Median filter	0.0007200	0.0007200	0.0005100	0.0006100	0.0005100
Sobel	0.0006240	0.0006240	0.0006240	0.0006800	0.0006240
Initial graph match	0.0000090	0.0000090	0.0000090	0.0000090	0.0000080
Match data rectangles					
Match links					
Create probe list					
Partial match					
Match strength probes					
Window selection	0.0001200	0.0001320	0.0001080	0.0005500	0.0006400
Classification and count	0.0009500	0.0008850	0.0008650	0.0015400	0.0016000
Match extension	0.0835200	0.0001470	0.0001400	0.0002650	0.0002590
Match strength probes					
Window selection	0.0003000	0.0000240	0.0000360	0.0009200	0.0009800
Classification and count	0.0030000	0.0004050	0.0003520	0.0047200	0.0054500
Hough probes					
Window selection	0.0002880	0.0000240	0.0000360	0.0005800	0.0007300
Hough transform	0.0790000	0.0054000	0.0104000	0.0610000	0.0690000
Edge peak detection	0.0007700	0.0000640	0.0000990	0.0015400	0.0017600
Rectangle parameter update	0.0002160	0.0000090	0.0000100	0.0002340	0.0002360
Result presentation	0.0008500	0.0004400	0.0004700	0.0004700	0.0010300
Best match selection	0.0000250	0.0000150	0.0000150	0.0000280	0.0000150
Image generation	0.0007200	0.0003200	0.0003500	0.0008400	0.0009100
Statistics					
Connected components		34	33	113	99
Right angles extracted		99	92	210	197
Rectangles detected		21	16	42	39
Depth pixels > threshold		14533	12891	18582	18817
Elements on initial probe list					
Hough probes		3	5	97	93
Initial match strength probes		20	15	142	142
Extension mat. str. probes		3	5	110	97
Models remaining		1	1	2	1
Model selected		1	5	7	8
Average match strength		0.96	0.93	0.84	0.87
Translated to		256,256	257,255	257,255	257,255
Rotated by		359	114	22	22

Table 11: Aspex ASP Results

Data Set	Sample		Test		Test2		Test3		Test4	
	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 data rectangles	0.17	0.04	0.11	0.03	0.09	0.03	0.26	0.13	0.22	0.06
Match links	0.19	0.24	0.06	0.20	0.06	0.65	0.74	0.29	0.59	0.78
Create probe list	0.01	0.00	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
Partial match	18.15	2.80	18.35	3.52	19.74	3.66	51.52	6.78	50.81	6.32
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

Table 12: Sequent Symmetry 81 Results

Sequent system. Part of the development period was spent back-porting modifications to the sequential version of the benchmark in order to enhance its portability. The low-level tasks were directly converted to a parallel implementation by dividing the data sets among the processors in a manner that completely avoided write-contention. About half of the development time was spent adding the appropriate 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 from one to eleven processors, with the optimum time being obtained with eight or nine processors. Additional processors resulted in an overall reduction in performance, which was due to a combination of factors. As the data sets 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 shared bus at about eight or nine processors, since the one run that was observed on a B-model cache system showed performance to improve with more processors. Table 12 shows the performance obtained for a single processor running the sequential version of the benchmark, to provide a comparison baseline, and the performance on the optimum number of processors for each data set.

6.10 Warp

The CMU Warp is a systolic array consisting of ten high speed floating point units in a linear configuration [Kung, 1984]. Processing in the Warp is directed by a host processor, such as the Sun-3/60 workstation that was used in executing the benchmark. The benchmark 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 of the implementation 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 on different portions of a problem. Thus, 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 of the benchmark exploits

Data Set	Sample	Test	Test2	Test3	Test4
Total	43.60	20.30	22.30	58.10	55.30
System	3.00	2.30	2.50	4.30	4.90
Overhead					
Miscellaneous	3.56	2.24	2.30	5.52	7.30
Startup	5.76	6.04	5.96	5.88	6.00
Image input	3.52	3.72	5.40	5.34	5.34
Image output					
Model input	1.30	1.18	1.02	1.08	1.06
Label connected components	3.98	4.04	4.60	4.54	4.56
Rectangles from intensity					
Miscellaneous					
Trace region boundary					
K-curvature	3.14	2.24	2.20	2.272	2.54
K-curvature smoothing	1.38	0.64	0.78	0.98	0.90
First derivative	0.42	0.24	0.28	0.34	0.40
Zero-crossing detection	0.32	0.06	0.12	0.14	0.22
Final corner detection	0.16	0.10	0.12	0.22	0.20
Count corners	0.02	0.02	0.04	0.06	0.06
Convex hull	0.02	0.00	0.02	0.08	0.06
Test for right angles	0.00	0.00	0.02	0.02	0.02
Final rectangle hypothesis	0.04	0.00	0.02	0.02	0.04
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 data rectangles	0.20	0.16	0.16	0.40	0.68
Match links	0.22	0.08	0.06	0.82	0.70
Create probe list					
Partial match					
Match strength probes					
Window selection					
Classification and count					
Match extension	24.80	3.64	4.58	38.60	41.20
Match strength probes	9.10	2.64	2.86	13.60	13.50
Window selection	0.02	0.02	0.02	0.24	0.18
Classification and count	9.00	2.56	2.82	13.20	13.10
Hough probes	15.30	0.96	1.68	23.30	25.80
Window selection	0.02	0.00	0.02	0.12	0.06
Hough transform	12.80	0.88	1.44	19.30	20.00
Edge peak detection	2.38	0.08	0.22	3.80	5.58
Rectangle parameter update	0.02	0.00	0.00	0.00	0.08
Result presentation	2.60	2.26	2.52	2.24	2.26
Best match selection	0.02	0.00	0.00	0.02	0.02
Image generation	2.54	2.20	2.46	2.16	2.18
Statistics					
Total match strength probes	91	23	20	247	239
Hough probes	58	3	5	97	95

Table 13: Results for the Warp

both the tightly-coupled parallelism of the Warp array, and the loosely-coupled task-level parallelism present in the benchmark.

Table 13 lists the results for the Warp. Timings were not provided for a few of the steps, but the totals include all of the processing time. The Miscellaneous category under Overhead is the time required for downloading code to the Warp array at various stages of the processing. A figure for the total system time was provided, rather than a breakdown of system time by task. The overall Total includes the system time, which is listed on the line below the Total. Note that sums of the times for the individual steps will not equal the Total time because of the task-level parallelism that was used.

6.11 Connection Machine

The Thinking Machines, Connection Machine model CM-2 is a data-parallel array of bit-serial processors that are linked by an N-dimensional hypercube router network [Hillis, 1986]. In addition, for every 32 of the bit-serial processors, a 32-bit floating-point coprocessor is provided. Connection Machines are available in configurations of 8192, 16384, 32768, and 65536 processing elements. Results were provided for direct execution on the three smaller configurations, and extrapolated to the largest configuration. The development team at Thinking Machines spent about three programmer months converting the low-level portion of the benchmark into 2600 lines of *LISP, which is a data-parallel extension to Common LISP. There was not enough time to implement the intermediate and top-down processing portions of the benchmark before the workshop, and other projects have taken priority over completing the benchmark since then. However, there was also some concern as to whether the Connection Machine would be the best vehicle for implementing the other portions, since they are more concerned with task parallelism than data parallelism. It was suggested that if the model data base included several thousand models to be matched, then an appropriate method might be found to take advantage of the Connection Machine's capabilities.

Table 14 summarizes the results for the Connection Machine on the low-level portion of the benchmark, 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. The results that were supplied were for only one data set, and did not indicate which one was used. It is interesting to note that several of the tasks saw little speedup with the larger configurations of the Connection Machine. Those tasks involved a collection of contour values that had been mapped into 16K virtual processors, which are enough to operate on all of the contour points in parallel, and so there was no advantage in using more physical processors than virtual processors. It was suggested that the Connection Machine might thus be used to process the contours for several images at once in order to make use of the larger number of processors. On the other hand, for those tasks that are pixel oriented, 256K virtual processors were used and therefore a proportional speedup can be observed as the number of processors increases.

6.12 Intel iPSC-2

The Intel Scientific Computers iPSC-2 is a distributed memory multiprocessor that consists of up to 128 Intel 80386 microprocessors that are linked by a virtual cut-through routing network which simulates point-to-point communications. Each of the microprocessors can have up to 8 MB of local memory, and an 80387 arithmetic coprocessor. The benchmark 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 of the low-level depth image processing. 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 15 presents their results, which are divided into user time and system time (including data and program load time, and output time).

Configuration	8K	16K	32K	64K
Total (low level tasks only)	1.26	0.91	0.71	0.63
Overhead				
Miscell.neous				
Startup	0.10	0.10	0.10	0.10
Image input	0.155	0.155	0.155	0.155
Image output				
Model input				
Label connected components	0.34	0.21	0.14	0.10
Rectangles from intensity				
Miscellaneous				
Trace region boundary	0.44	0.30	0.23	0.17
K-curvature	0.019	0.019	0.018	0.018
K-curvature smoothing	0.0056	0.0055	0.0062	0.0055
First derivative	0.00038	0.00037	0.00037	0.00037
Zero-crossing detection	0.00021	0.00020	0.00019	0.00019
Final corner detection	0.0058	0.0053	0.0053	0.0053
Count corners	0.018	0.016	0.016	0.016
Convex hull	0.041	0.038	0.039	0.038
Test for right angles				
Final rectangle hypothesis				
Median filter	0.082	0.041	0.025	0.015
Sobel	0.052	0.026	0.014	0.008

Table 14: Results for the Connection Machine on the Low-Level Portion

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										
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

Table 15: iPSC-2 Results for Median Filter and Sobel Steps

Architecture	Sun-3	Alliant	IUA	ASP	Sequent	Warp
Overhead	0.6	14.6	16.5	0.6	2.3	
Label connected components	3.5	12.0	0.1	30.0	4.9	9.1
Rectangles from intensity	0.8	5.8	19.1	2.5	0.6	
Median filter	30.9	16.8	0.7	0.6	11.9	24.5
Sobel	17.0	6.3	3.2	0.5	5.8	1.1
Initial graph match	3.1	4.3	14.4	0.0	1.2	1.0
Match data rectangles	0.0	0.2	3.5		0.0	0.5
Match links	0.0	0.1	10.5		0.1	0.5
Create probe list	0.0	0.0	0.1		0.0	
Partial match	3.0	4.0	4.0		1.1	
Match extension	40.9	34.2	35.6	63.9	61.9	56.9
Result presentation	3.1	5.4	2.7	0.7	8.0	6.0

Table 16: Distribution of Processing Time for Data Set Sample

Architecture	Sun-3	Alliant	IUA	ASP	Sequent	Warp
Overhead	1.5	26.4	30.9	2.3	3.1	
Label connected components	8.2	28.9	0.1	63.4	9.4	19.9
Rectangles from intensity	1.2	6.4	27.5	8.1	0.9	
Median filter	35.2	17.7	1.2	0.2	34.6	42.9
Sobel	39.4	11.4	5.9	1.7	34.4	2.4
Initial graph match	7.4	7.5	16.8	0.0	6.0	1.2
Match data rectangles	0.0	0.2	3.4		0.0	0.8
Match links	0.0	0.1	12.5		0.1	0.4
Create probe list	0.0	0.0	0.3		0.0	
Partial match	7.3	7.2	16.9		5.8	
Match extension	3.4	2.7	3.9	0.4	5.9	17.9
Result presentation	3.6	5.6	2.1	1.2	5.8	11.1

Table 17: Distribution of Processing Time for Data Set Test

Architecture	Sun-3	Alliant	IUA	ASP	Sequent	Warp
Overhead	1.7	26.5	30.6	2.0	3.2	
Label connected components	8.6	21.6	0.1	57.3	9.8	20.6
Rectangles from intensity	1.3	6.6	29.6	7.2	1.3	
Median filter	28.2	13.1	1.2	1.3	26.9	6.2
Sobel	41.3	11.5	5.9	1.6	34.8	3.2
Initial graph match	7.9	8.1	14.7	0.0	6.7	1.0
Match data rectangles	0.0	0.2	2.9		0.0	0.7
Match links	0.0	0.1	10.9		0.3	3.7
Create probe list	0.0	0.0	0.2		0.0	
Partial match	7.9	7.7	15.1		6.4	
Match extension	5.7	4.6	5.5	0.4	9.2	20.5
Result presentation	5.1	7.2	2.6	1.1	7.9	11.3

Table 18: Distribution of Processing Time for Data Set Test2

Architecture	Sun-3	Alliant	IUA	ASP	Sequent	Warp
Overhead	1.0	16.2	3.3	0.7	1.6	
Label connected components	5.1	13.4	0.0	30.8	4.9	7.8
Rectangles from intensity	1.0	5.8	3.1	2.8	0.7	
Median filter	16.5	8.0	0.1	0.5	13.2	2.4
Sobel	24.5	7.0	0.6	0.6	17.1	1.6
Initial graph match	12.4	14.1	26.9	0.0	8.1	2.1
Match data rectangles	0.1	0.4	3.2		0.1	0.7
Match links	0.1	0.5	23.6		0.1	1.4
Create probe list	0.0	0.0	0.2		0.0	
Partial match	12.2	13.1	43.7		58.3	
Match extension	36.8	30.9	21.5	0.2	50.9	66.4
Result presentation	2.7	4.0	0.7	0.4	3.5	3.9

Table 19: Distribution of Processing Time for Data Set Test3

Architecture	Sun-3	Alliant	IUA	ASP	Sequent	Warp
Overhead	1.0	16.3	3.5	0.7	1.6	
Label connected components	5.1	13.5	0.0	26.3	5.1	8.2
Rectangles from intensity	1.0	5.7	3.3	2.8	0.7	
Median filter	16.4	8.1	0.1	0.4	13.5	3.6
Sobel	24.5	7.1	0.7	0.5	17.5	1.7
Initial graph match	12.2	13.9	20.7	0.0	8.2	2.5
Match data rectangles	0.0	0.4	2.7		0.0	1.2
Match links	0.1	0.4	17.9		0.2	1.3
Create probe list	0.0	0.0	0.2		0.0	
Partial match	12.1	13.1	38.6		8.0	
Match extension	37.1	30.9	32.1	0.2	49.8	74.5
Result presentation	2.7	4.1	0.7	0.9	3.6	4.1

Table 20: Distribution of Processing Time for Data Set Test4

6.13 Comparative Performance Summary

As previously mentioned, the direct comparison of raw timings is not especially useful. We have attempted to provide as much information about each benchmark implementation as is necessary for others to make informed and intelligent comparisons of the results. For example, a valid comparison of architectural features should take into account the technology, instruction rate, and scalability of the processors that were actually used to obtain the results. On the other hand, a comparison that seeks to establish the currently available machine with the best cost to performance ratio should look at the timings with respect to both the programming effort required and the price of the hardware. The authors hope to develop and publish some direct comparisons of architectural features, once a few more implementations are added to the sample and a reasonably broad set of scaling functions is established.

In the meantime, one interesting comparison that can be immediately drawn from the data, which requires no scaling for technology, is the relative amount of processing time that each architecture expends on each portion of the benchmark. This function, which is just the percentage of the total time taken for each step, provides an indication of those tasks that each architecture excels at and those that it struggles with. Tables 16 through 20 compare the efforts for the different architectures on each of the major benchmark steps, for the five data sets. It should be noted that the data sets, Test and Test2, require very little mode! matching effort since they involve very simple models. The other three data sets involve more complex models, which is easily seen in Tables 16, 19, and 20. Only the complete implementations are listed, since a total time for the benchmark is required to compute the values in the tables. Blanks in the tables represent information that was missing from the reports by the different groups.

6.14 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 benchmark efforts. The general conclusion of the participants was that the benchmark is a significant improvement over past efforts, but that there is still work to be done.

One of the major complaints was the sheer size and complexity of the benchmark solution. The sample solutions are a considerable help in this regard, but a great deal of work is still required to transport them to parallel architectures. Several people expressed the opinion that a FORTRAN version should be made available so that the benchmark would be taken up by the traditional supercomputing community. It was pointed out that most groups don't have the time or resources to implement such a complex benchmark, and that it would be almost impossible to tune it for optimum 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 one of the top-down probes (although there was no consensus on which one should be removed), and simplification of the graph matching code through increased generality.

On the other hand, several people complained that the benchmark task was 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 then s· ;gested that an interesting variation on the benchmark would be to 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 would 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 made in 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 to be built out of a standard set of general purpose vision subroutines. Even though a group might still have to implement all of those routines, they would then at least 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 an 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 the high level. 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 people feel is necessary. In the end, 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 many of the reporting requirements were for aspects of the implementation that went 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 made completely rigid.

The conclusion of the panel session was to let the benchmark stand as specified for some period of time, in order to allow more groups to complete their implementations. Then a new version of the benchmark 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.

7. Conclusions

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

The CAAPP is optimized to perform local operations on neighborhoods of pixels and to provide feedback to the higher levels of processing about the state of the computation and statistics about low-level data. It excels at very tightly-coupled fine-grained parallelism. The mapping of one pixel onto each processor ensures that the maximum amount of parallelism available in the low-level vision tasks will be utilized. The reconfigurable nature of its Coterie network supports communication among groups of processors that correspond to image events. The global summary feedback capabilities and I/O subsystem of the CAAPP make it especially well suited for real-time applications. Of course, its unique feature is its interface to the ICAP level.

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

The high level tasks which perform knowledge-based inference and manipulation of object models are run in the SPA. To support distributed artificial intelligence processing, powerful processors are needed with large amounts of memory. The communication between processes will primarily be in terms of a blackboard system managed by the processes themselves. As these processes run and make requests via the ACU to the ICAP (and sometimes directly to the CAAPP) they will extract information about the image and post the results of their analysis on the blackboard for other processes to use. The end result will be an interpretation of the image achieved by cooperation of the set of object processes.

The IUA simulator, parallel language extensions to FORTH, debugger, subroutine libraries, and applications provide an environment for developing code and demonstrating the IUA prototype. The simulator, in particular, permits the programmer to directly observe the status of algorithms running in the machine as they are executing. The wealth of visual information its displays provide is very helpful to both experienced and novice programmers.

The IUA prototype hardware has been completed, and is fully functional, although it operates at a speed that is less than originally specified. The sources of this reduced performance have been determined, and could be corrected if funding were available. The hardware has been demonstrated running several vision algorithms.

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 goes a long way toward 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 to date, but it is hoped that others will be added to the sample. In the meantime, it is possible to draw a few general conclusions from the data that has been gathered. It is clear that a tremendous speedup is possible for the data parallel portions of the interpretation task. However, every one of the architectures in the sample devoted the greatest percentage of its overall time to the model matching portion of the benchmark on those data sets that involved 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 true high-level processing will force 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.

8. References

[Annexstein, 1990] Annexstein, F., and Baumslag, M., A Unified Approach to Offline Permutation Routing, 2nd ACM Symp. on Parallel Algorithms and Architectures, 1990.

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

[Beveridge, 1989] Beveridge, J.R., Griffith, J., Kohler, R., Hanson, A., and Riseman, E., Segmenting Images Using Localized Histograms and Region Merging, International Journal of Computer Vision, 2, 1989.

[Carpenter, 1987] Carpenter, R., 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.

[Dally, 1986] Dally, W., and Seitz, C., The Torus Routing Chip, Distributed Computing, 1 (3), 1986.

[Dally, 1987] Dally, W., and Seitz, C., Deadlock Free Routing in Multiprocessor Interconnection Networks, IEEE Trans. on Comp., 36 (5), 1987.

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

[Duff, 1986] Duff, M., 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

[Foster, 1971] Foster, C., and Stockton, F., Counting Responders in an Associative Memory, IEEE Transactions on Computers, Vol. C-31, No. 12, Dec. 1971, pp. 1580-1583.

[Graham, 1972] Graham, R., An Efficient Algorithm for Determining the Convex Hull of a Planar Set, Information Processing Letters, 1, 1972.

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

[Herbordt, 1990a] Herbordt, M., Weems, C., and Shu, D., Routing on the CAAPP, Proceedings of the 10th Int. Conf. on Pattern Recognition, 1990.

[Herbordt, 1990b] Herbordt, M., Weems, C., and Corbett, J., Message Passing Algorithms on a SIMD Torus with Coteries, Proceedings of the 2nd ACM Symposium on Parallel Algorithms and Architectures, 1990.

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

[Kermani, 1979] Kermani, P., and Kleinrock, L., Virtual Cut-Through: A New Computer Communication Switching Technique, Comp. Networks, 3, 1979.

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

[Kung, 1984] Kung, H., and Onat M., Warp: A Programmable Systolic Array Processor, Proc. SPIE Symp., Vol. 495, Real-Time Signal Processing VII, Aug. 1984

[Lea, 1988] Lea, R., ASP: A Cost-effective Parallel Microcomputer, IEEE Micro, October, 1988, pp. 10-29

[Leighton, 1989] Leighton, F., Makedon, F., and Tollis, I., A 2n - 2 Step Algorithm for Routing in an n x n Array With Constant Size Queues, 1st ACM Symp. on Parallel Algorithms and Architectures, 1989.

[Li, 1987] Li, H., and Maresca, M., Polymorphic-Torus Network, Proc. International Conference on Parallel Processing, 1987.

[Little, 1989] Little, J., Blelloch, G., and Cass, T., Algorithmic Techniques for Computer Vision on a Fine-Grained Parallel Machine, IEEE Trans. on PAMI, 11 (3), 1989.

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

[Miller, 1988] Miller, R., Prasanna, V., Kumar, D., and Reisis, Q., Meshes With Reconfigurable Buses, Proc. of the MIT Conference on Advanced Research in VLSI, 1988.

[Nassimi, 1979] Nassimi, D., and Sahni, S., Bitonic Sort on a Mesh-Connected Parallel Computer, IEEE Trans. on Comp., 28, 1979.

[Nassimi, 1980] Nassimi, D., and Sahni:, S., An Optimal Routing Algorithm for Mesh-Connected Parallel Computers, J. of the ACM, 27. 1980.

[Preston, 1986] 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

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

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

[Rosenfeld, 1987] Rosenfeld, A., A Report on the DARPA Image Understanding Architectures Workshop, Proceedings of the 1987 DARPA Image Understanding Workshop, February 1987, Los Angeles, CA, Morgan Kaufmann Publishers, Los Altos, CA, 1987, pp. 298-302

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

[Thompson, 1977] Thompson, C., and Kung, H., Sorting on a Mesh Connected Computer, Comm. of the ACM, 20 (4), 1977.

[Valiant, 1981] Valiant, L., and Brebner, G., Universal Schemes for Parallel Computation, 13th ACM Symp. on the Theory of Computing, 1981.

[Weems, 1984] Weems, C., Image Processing on a Content Addressable Array Parallel Processor}, Ph.D. Dissertation Computer and Information Science Department, University of Massachusetts at Amherst, September 1984.

[Weems, 1988] Weems, C., Hanson, A., Riseman, E., and Rosenfeld, A., An Integrated Image Understanding Benchmark: Recognition of a 2 1/2 D "Mobile", Proceedings of the 1988 DARPA Image Understanding Workshop, March, 1988, Cambridge, MA, Morgan Kaufmann Publishers, Los Altos, CA, 1988, pp.

[Weems, 1989] Weems, C., Levitan, S., Hanson, A., Riseman, E., Shu, D., and Nash, J., The Image Understanding Architecture, International Journal of Computer Vision, Vol. 2, Kluwer Acauemic Publishing, Boston, MA, 1989, pp. 251-282.

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

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

[Weems, 1990c] Weems, C., and Burrill, J., The Image Understanding Architecture and its Programming Environment, in Parallel Architectures and Algorithms for Image Understanding, V.K. Prasanna Kumar, ed. Academic Press, Orlando, Florida, 1990.

[Weems, 1991] Weems, C., Riseman, E., Hanson, A., and Rosenfeld, A., The DARPA Image Understanding Benchmark for Parallel Computers, Journal of Parallel and Distributed Computing, To Appear.