(2)

ETL-0542

AD-A212 621

The Image Understanding Architecture Project
Second Annual Report

Charles C. Weems
Steven P. Levitan
Allen R. Hanson
Edward M. Riseman
David B. Shu
J. Gregory Nash
James Burrill
Michael Rudenko

University of Massachusetts
Computer & Information Science Department
Amherst, Massachusetts 01003

March 1989

Prepared for:

Defense Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, Virginia 22209-2308

U.S. Army Corps of Engineers
Engineer Topographic Laboratories
Fort Belvoir, Virginia 22060-5546

THE IMAGE UNDERSTANDING ARCHITECTURE (IUA) PROJECT SECOND ANNUAL REPORT

Contract Number: DACA76-86-C-0015

For the period
September 23, 1987 to September 22, 1988

Charles C. Weems, Steven P. Levitan, Allen R. Hanson,
Edward M. Riseman, David Shu,
J. Gregory Nash, James Burrill, and Michael Rudenko

Prepared for:

Defense Advanced Research Projects Agency
Arlington, VA 22209-2308

US Army Engineer Topographic Laboratories
Fort Belvoir, VA 22060-5546

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for Public Release; Distribution is Unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S) ETL-0542
6a. NAME OF PERFORMING ORGANIZATION University of Massachusetts 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION U.S. Army Engineer Topographic Laboratories
6c. ADDRESS (City, State, and ZIP Code) Computer & Information Science Dept. Amherst, MA 01003	7b. ADDRESS (City, State, and ZIP Code) Fort Belvoir, Virginia 22060-5546
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Research Projects Agency 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER DACA76-86-C-0015 (Arpa Order No. 5683)
8c. ADDRESS (City, State, and ZIP Code) 1400 Wilson Boulevard Arlington, VA 2209-2308	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
The Image Understanding Architecture (IUA) Project - Second Annual Report

12. PERSONAL AUTHOR(S) Charles C. Weems, Steven P. Levitan, Allen R. Hanson, Edward M. Riseman, David Shu, J. Gregory Nash, James Burrill, and Michael Rudenko

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Annual	FROM 9/22/87 TO 9/22/88	1989 March	76

16. SUPPLEMENTARY NOTATION
Previous report in series: ETL-0499 The Image Understanding Architecture (IUA) Project First Annual Report (April 1988) - Weems, et al.

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Image Understanding Architecture, Knowledge-Based Vision, Real-time Computer Vision, Software Simulator, Parallel Processor

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The primary goal of the Image Understanding Architecture (IUA) project is to build a proof-of-concept prototype of a 1/64th slice of a next generation vision architecture, and develop the software support environment that will be needed to utilize the hardware. The majority of the hardware effort is taking place at Hughes Research Laboratories, Malibu, California although UMass has principal 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 is taking place at UMass, although Hughes is 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.

During the second year of this program, has focussed on extensions to the IUA software simulator programming environment, the development of library routines and demonstration software for the IUA, construction of the custom chips for the architecture, circuit board

OVER---->

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code)	22c. OFFICE SYMBOL
Linda H. Graff	(202)355-2818	CEETL-RI

DD Form 1473, JUN 86 *Previous editions are obsolete.* SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

design, and the design and implementation of an integrated image understanding benchmark for DARPA.

This report presents the results of the IUA project for the second year of its original two-year contract period. The purpose of the IUA project is to design and construct a next-generation parallel processor that specifically addresses the needs of real-time computer vision applications.

Included in this report is a summary of accomplishments during the second year, an overview of the IUA design, a description of the new DARPA Integrated IU Benchmark Exercise, a summary of the performance figures for the IUA on the exercise, and test reports and photos of chips developed through MOSIS under this program in an appendix.

Abstract

The primary goal of the Image Understanding Architecture (IUA) project is to build a proof-of-concept prototype of a 1/64th slice of a next generation vision architecture, and develop the software support environment that will be needed to utilize the hardware. The majority of the hardware effort is taking place at Hughes Research Laboratories, Malibu, California although UMass has principal 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 is taking place at UMass, although Hughes is 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.

During the second year of this program, we have focussed on extensions to the IUA software simulator programming environment, the development of library routines and demonstration software for the IUA, construction of the custom chips for the architecture, circuit board design, and the design and implementation of an integrated image understanding benchmark for DARPA.

This report presents the results of the IUA project for the second year of its original two-year contract period. The purpose of the IUA project is to design and construct a next-generation parallel processor that specifically addresses the needs of real-time computer vision applications.

Included in this report is a summary of accomplishments during the second year, an overview of the IUA design, a description of the new DARPA Integrated IU Benchmark Exercise, a summary of the performance figures for the IUA on the exercise, and test reports and photos of chips developed through MOSIS under this program in an appendix.

Table of Contents

List of Figures

Preface

This document reports the results of efforts at the University of Massachusetts and Hughes Research Laboratories during the second year of the DARPA sponsored Contract DACA76-86-C-0015, "Image Understanding Architecture". These efforts include both hardware and software development.

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 from those documents. The report begins with an executive summary of the major accomplishments in the second year. The remainder of the report presents an overview of the project, and of our efforts related to the DARPA Image Understanding Benchmark Exercise this year. The reader is thus referred to the executive summary for a status report on the project and to the later sections for more detailed technical information.

Executive Summary

The major accomplishments and activities at UMass during the second year of the project, including supervision of the Hughes and the University of Pittsburgh subcontracts, are as follows:

1. The VAX simulator for the CAAPP was transported to the Sun-3 and integrated with the SUN windowing environment. Many enhancements were made to the simulator including graphical displays of CAAPP registers and memory, grayscale image displays, point-and-click editing of register and memory planes, memory utilization monitor display, code window, variable rate snapshot screen update, enhanced Coterie Network display with visible signal propagation, and an instruction cycle monitor. A full simulator for the ICAP was also added, and is discussed below.

2. The TI Explorer-based simulator was enhanced with additional graphics capabilities, including grayscale images and point-and-click editing.

3. Additional tools for the environment were developed, including a screen hardcopy utility, a convolution language construct that acts like a loop which spirals out through the cells in a square mask, and a memory management package that allocates space for variables in the CAAPP memory.

4. The group under the direction of Dr. Steven Levitan at Pittsburgh developed a simulator for the TMS320C25 processor. This simulator was then modified into a 64 parallel processor ICAP simulation that was integrated into the Sun CAAPP simulator. The combined simulator provides a separate ICAP code window and ICAP register monitors.

5. The combined CAAPP-ICAP simulator was transported to a Sequent Symmetry multiprocessor and enlarged to a full-size (512×512) CAAPP and a 16 processor ICAP. The simulator can be configured to simulate a 4096 processor ICAP, but our Symmetry does not have enough disk space to swap such a large image. The Symmetry also greatly limits the amount of shared memory (CISM and ISSM) that can be simulated.

6. An assembler was written for the TMS320C25 because Texas Instruments delayed release of the Sun version of its cross assembler and C compiler. It is expected that our assembler will be replaced by these tools once they are delivered.

7. Numerous library routines and demonstration algorithms were written for the IUA. These include: floating point arithmetic, floating point I/O, floating point square root, integer arithmetic and square root, integer and floating point conversion, Sobel operator, Gaussian convolution, K-curvature, median filter, connected component

labeling, border tracing, convex hull, right angle finder, windowed probes for classification of pixels and Hough transform, and graph matching. Many of these routines were used in the DARPA Integrated Image Understanding Benchmark exercise discussed below.

8. The feedback concentrator chip for the CAAPP and ICAP was redesigned and submitted for fabrication. Five of these chips are used in the IUA prototype to provide the Some/None, ICAP Done (0,1, and 2) and Backing Store Done signals from the array to the controller. This fabrication run resulted in fully functional chips with a 75 nS delay from input to output. There were a sufficient number of working parts for construction of the IUA prototype. See the appendix for a more detailed test report.

9. The ICAP communication chips were resubmitted following the previous year's failed fabrication. On this run there were no fabrication or bonding faults; however, we discovered three design errors. A replication error in the multiplexer tree resulted in several I/O paths being unintentionally wired together. The use of a pad frame that was not designed to be scaled to 2 microns resulted in a high drive current for the inputs. Lastly, a timing error in the precharge circuitry for the memory resulted in unreliable storage of connection patterns. Debugging of this chip was aided by separate submission of a test-structure chip that contained all of the major components of the communications chip divided into isolated units with a large number of external test points. The chip was redesigned to correct these errors and one additional feature was included that allows a pattern to be active while the connection pattern memory is being changed. In the previous design, communication could not take place while stored patterns were being altered. The new design, with over 60,000 devices, was submitted along with another test-structures chip, but various vendor problems delayed fabrication for a little over six months. The chips that were eventually returned were salvaged by MOSIS from what was essentially a bad run. Only six packaged parts were supplied, and a backup run was scheduled with another vendor to supply additional parts. Nonetheless, four of the parts tested were fully functional with the ability to carry data at a rate of up to 30 MHz (three times the requirement of the prototype). Another four working parts will be required to construct the IUA prototype, but these should be forthcoming under the backup fabrication run. This chip has the potential to be used in other applications, because it implements a general 32 input, 32 output bit-serial communication switch that can store and instantly recall up to 32 commonly used I/O connection patterns. A connection pattern can be any one-to-one mapping, or any set of non-overlapping one-to-many mappings (disjoint subset broadcasts). The chip can also be easily cascaded to provide large switch networks (up to 128 × 128 with reasonable physical size and component counts). We are working on a 64 I/O channel version of the chip that will enable even larger networks to be built, and that will provide distributed

control of routing in addition to central control. More details of the fabrication and testing of these chips are given in accomplishments 21 through 24 below.

10. Hughes received the 64-processor CAAPP test chips that were submitted for fabrication during the first year. These had several design errors, but large portions of the chips were functional. Hughes redesigned the chips to correct those errors, but fabrication was delayed by vendor problems for over six months. This delay resulted in a significant slip in the development schedule, and required Hughes to complete the CAAPP chip design (adding the backing-store control and glue logic) and submit it without having a chance to test their previous design. In order to compensate for this lack of feedback, Hughes spent several months simulating the new chip. Because the chip has roughly 130,000 devices, Hughes was unable to perform an exhaustive simulation due to insufficient computing resources (even on a large Apollo DN-4000 system, a simple simulation took several days). They were able to simulate an extracted circuit CAAPP chip performing a connected components labeling routine, at the electrical device level, which exercises most of the components of the chip.

11. Hughes designed and built the two-chip test and demonstration jig for the 64-processor CAAPP chips. This jig will allow us to begin to move our software environment to the hardware prior to the actual delivery of the IUA prototype. Of course, this depends upon the success of the delayed fabrication run.

12. Hughes designed the processor daughterboards and the prototype motherboard, although neither has been fabricated since the processor chips have not been built.

13. Hughes informed us that their new CAAPP chip design includes an enhanced Coterie Network. The new version provides the ability for signals to cross without interacting, for signals to bypass a processor on diagonal links, and for very high speed row and column broadcasts and OR operations.

14. It was determined that an integrated circuit test system from a commercial vendor could be used to provide much of the functionality of the custom array control unit that we originally proposed to build under a modification to our contract. The advantages of using a test system include greater flexibility than a hard-wired system, the ability to also test our custom IC's and to diagnose a wider range of faults in the prototype IUA, software compatibility with the test/demonstration system at Hughes, greater reliability than custom hardware, and immediate availability which will allow us to begin to transport our simulator environment to the hardware in advance of delivery of the prototype. A commercial test system is also somewhat less expensive than development of a custom controller. The main disadvantages are a smaller instruction memory, a command language and interface that are not tuned for our application, less capability for manipulating feedback from the array, and a less-than-optimum execution speed for large or complicated tasks. It was

therefore decided that, for development of the prototype, a commercial test system's advantages outweigh its disadvantages. However, because it may be desirable to build copies of the IUA in the future, the group at the University of Pittsburgh has undertaken to design a low cost replacement for the test system. If funding is approved, such an alternative controller would be built and delivered between one and two years after the IUA prototype.

15. In a meeting with Hughes, we assembled a complete, standardized, two volume documentation set for the IUA prototype. The manuals at all sites are maintained identically to facilitate interactions between UMass, Hughes and the University of Pittsburgh.

16. Hughes has designed an I/O subsystem for the IUA prototype using Datacube image processing boards. This system is similar to another that Hughes has developed for a different project, and so they are confident that it will work well. The system will allow input of up to 512×512 images into a frame buffer. The IUA prototype will then be able to load data from the buffer in several different ways to facilitate both 64×64 sub-image operations, or full image operations.

17. UMass, together with the University of Maryland, undertook the development of a new image understanding benchmark for DARPA. The new benchmark addresses a major deficiency in the first DARPA benchmark by simulating an entire image interpretation scenario. The first benchmark was simply a set of isolated algorithms that failed to test communication between different tasks. Also, the new benchmark provides specific test data sets whereas the earlier benchmark left many aspects of the input data unspecified. UMass and Maryland designed the basic problem together, and UMass then worked out the details, programmed a sequential and a parallel solution to the problem, distributed the software, data, and documentation, collected the results, and organized a benchmark workshop for the Image Understanding community. The sequential solution was intended to reduce the coding effort required for exercise participants and to demonstrate a valid solution to the problem. An effort was made to incorporate as much of the first benchmark as possible into the new benchmark in order to further reduce the efforts of the participants. The parallel solution was coded for a Sequent Symmetry multiprocessor in order to demonstrate that a parallel implementation could be developed. UMass publicized the benchmark exercise through many direct contacts, and articles published in the proceedings of the DARPA Workshop, International Supercomputing Conference, and the Computer Vision and Pattern Recognition Conference. Documentation was mailed to over fifty sites and about half of those obtained the software and test data. A dozen machines were tested, although only seven ran the complete benchmark, and all of the participants indicated that their performance was sub-optimal. UMass developed a complete benchmark implementation for the IUA, although it was still giving

results that were only partially correct at the time of the workshop. Debugging will, however, only reduce the times for the IUA implementation.

18. Dr. Weems spent a significant amount of time presenting reports and giving briefings on both the IUA and the DARPA Benchmark. These included a panel session at the Workshop on Computer Architecture for Pattern Analysis and Machine Intelligence, the British Informatics Society's Conference on Parallel Processing, the DARPA IU Workshop, the Frontiers of Massively Parallel Computing Conference, the International Conference on Parallel Processing, the International Supercomputing Conference, the Conference on Computer Vision and Pattern Recognition, the UMass Industrial Affiliates Meeting, the Interagency AI Steering Committee, the Defense Mapping Agency, General Motors, Texas Instruments, the Air Force Office of Scientific Research, Digital Equipment Corporation the Office of Naval Research, the Supercomputing Research Center, Sequent Computer Corp., the University of Warwick, University College London, and of course, DARPA.

19. UMass developed and submitted two proposals this year to supplement this effort. One was to the Defense University Research Instrumentation Program for equipment to set up a VLSI design laboratory capable of supporting very large custom designs, such as the CAAPP processor chip. This proposal was rejected. The second was to DARPA for continuation of our current effort for evaluation of and further development on the IUA prototype, with the goal of designing the next generation IUA. This proposal is awaiting further action at DARPA.

1. Introduction

The primary goal of the Image Understanding Architecture project is to build a proof-of-concept prototype of a 1/64th slice of a next generation vision architecture, and develop the software support environment that will be needed to utilize the hardware. The majority of the hardware effort is taking place at Hughes Research Laboratories, Malibu, California although UMass has principal 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 is taking place at UMass, although Hughes is also involved in some software devlopment, both in support of their hardware efforts, and in the form of algorithm development for specific applications on the IUA.

During the second year of this program, we have focussed on extensions to the IUA software simulator programming environment, the development of library routines and demonstration software for the IUA, construction of the custom chips for the architecture, circuit board design, and the design and implementation of an integrated image understanding benchmark for DARPA.

2. Overview of the Image Understanding Architecture (IUA)

The Image Understanding Architecture represents a hardware implementation of the three levels of abstraction inherent in our view of computer vision. It consists of three different, tightly coupled parallel processors. These are the Content Addressable Array Parallel Processor (CAAPP) at the low level, the Intermediate Communications Associative Processor (ICAP) at the intermediate level, and the Symbolic Processing Array (SPA) at the high level (Figure 1).[1] The CAAPP and ICAP levels are controlled by a dedicated Array Control Unit (ACU) that takes its directions from the SPA level. In each layer of the IUA, the processing elements are tuned to the computational granularity and algorithms required by that particular level of abstraction. For example, it is inappropriate to try to run concurrent LISP at the lowest level, because processing there is primarily concerned with fast pixel operations, and should be tuned for real-time image processing. At the highest level, on the other hand, symbolic AI processing will be the main objective, so the high-level processors are selected for their ability to run LISP code efficiently. Figure 2 is a tabular summary of the architectural requirements of each level of abstraction in vision processing, in terms of computation, communication, and control.

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

We are currently building 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 specifically noted that a feature pertains only to the prototype.

At the high level, the IUA is purely a MIMD parallel processor. Additionally, the intermediate and low levels of the IUA may be treated in a variety of modes of parallelism to allow multiple hypotheses from the SPA to be evaluated in parallel at the lower levels. These include the CAAPP operating in pure SIMD or local-SIMD mode, and the ICAP operating in synchronous-MIMD or pure MIMD mode.

A brief explanation of how the local-SIMD and synchronous-MIMD modes differ from the familiar SIMD and MIMD modes is required. In local-SIMD mode, the CAAPP cells execute in disjoint SIMD groups, with each group able to operate on locally broadcast values, and to locally compute its own summary values in parallel with all other groups. This allows different parameters to be employed in processing disjoint portions of the image, with all portions of an image being processed simultaneously. Local-SIMD differs from Multi-SIMD processing in that all processors receive the same instruction stream, whereas with Multi-SIMD the disjoint groups receive separate instruction streams. The prototype IUA does not support Multi-SIMD processing, but we are exploring several ways of adding that capability to the full scale IUA. In synchronous-MIMD mode, the programming paradigm is more like SIMD than MIMD: the ICAP processors execute the same program, but have their own instruction pointers so that they can branch independently, and globally synchronize for each stage of processing. Synchronous-MIMD has the advantage of being as simple to program as a SIMD system, but without the time penalty usually encountered in SIMD systems associated with sequential execution of all the paths in a branching control structure.

2.1 The CAAPP (Low) Level

The CAAPP is a 512×512 square grid array of custom 1-bit serial processors intended to perform low-level image processing tasks. The CAAPP is similar in many ways to CLIP-4 [Duff, 1978], MPP [Batcher, 1980], DAP [Hunt, 1981], GRID [Arvind, 1983], GAPP [Davis and Thomas 1984], and the Connection Machine [Hillis, 1986]. However, its architecture is especially oriented towards *associative processing* with an emphasis on fast global summary feedback mechanisms supported in hardware. The CAAPP is also specifically designed to interact with the ICAP in a tightly coupled fashion for both bottom-up and top-down processing. Thus, the CAAPP has been tailored to permit flexible control, to provide rapid feedback to the controlling processes so that they may exercise control in response to actual image properties, and to integrate fully into a hierarchically organized vision architecture.

The CAAPP processing elements are linked through a four way (S,E,W,N) communications grid that is augmented with circuitry that allows certain types of long distance communication to take place quickly. Each processor can execute an instruction in 100

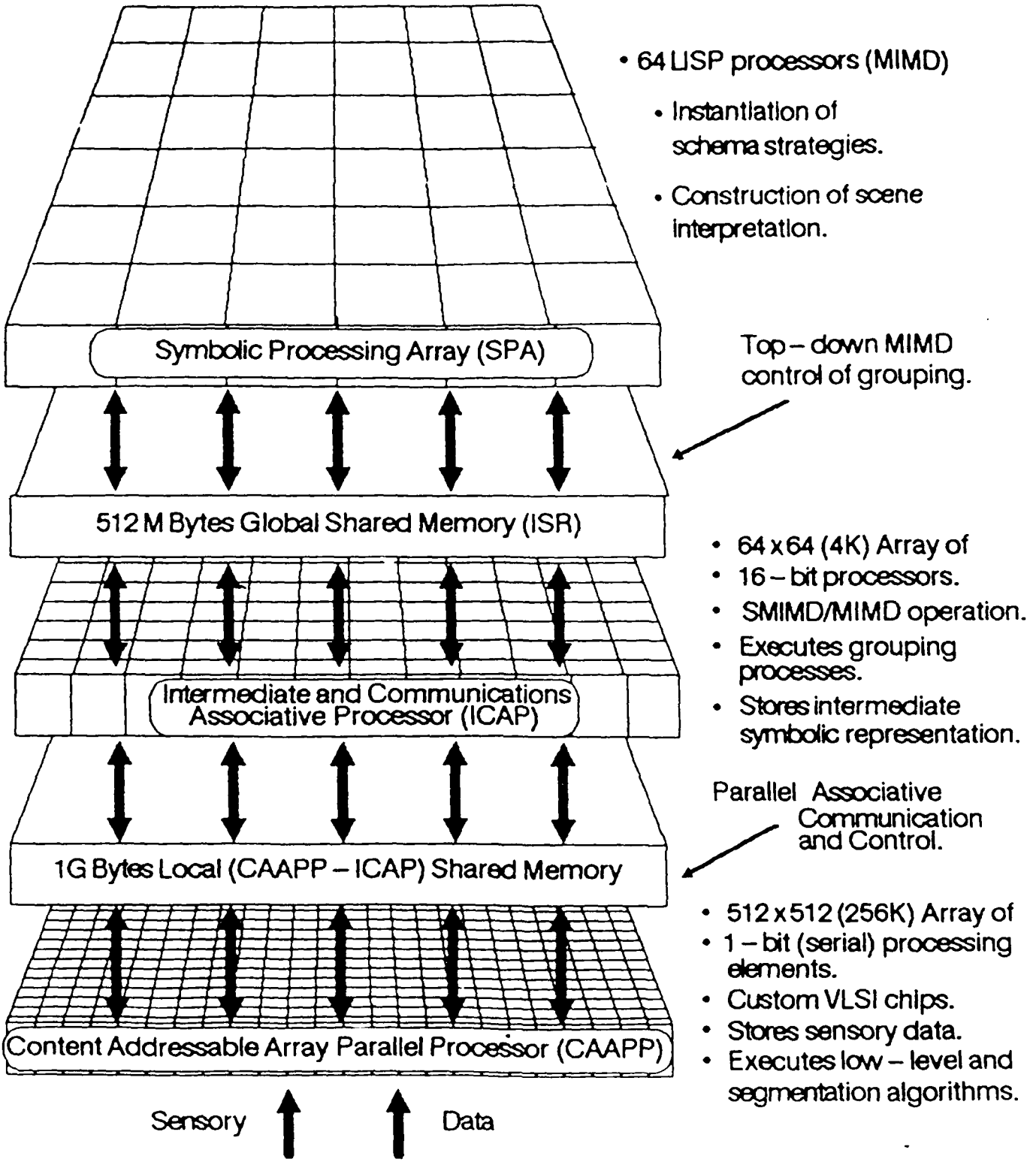

Figure 1: IUA Overview

	Low Level	Intermediate Level	High Level
Computation	• Fine grained • 256K 8-bit pixels • 8-bit integer arithmetic • Limited real arithmetic • Comparisons	• Medium grained • Thousands of "tokens" • 16-bit integer arithmetic • 32-bit real arithmetic • Building "token" records • Maintaining lists of token relationships	• Coarse grained • Hundreds of "schemas" • 32 bit real and integer arithmetic • List traversals • Symbolic processing
Communication	• Local neighborhood • Across connected components • Structured patterns • Broadcast • Up • Summary feedback • High-speed I/O • Fine grained messages	• Local neighborhood • Long distance • Broadcast • Down and up • Summary feedback • Medium length messages	• Blackboard access • Control info to lower levels • Queries to lower levels • Data up from lower levels • Coarse grained messages
Control	• SIMD-Associative • Multi-SIMD • Locally associative SIMD • Central control • Local activity control	• SIMD-Associative • Synchronous-MIMD • MIMD directed by higher level • Central and local control	• MIMD • Distributed control • Attention focusing mechanisms • Coordination with central control of lower levels

Figure 2: Computation, Communication and Control Requirements of Each Level of Vision Processing

nanoseconds and contains 5 one-bit registers, an ALU, data routing circuitry, and 320 bits of RAM that acts as an explicitly managed cache memory. Each element has access to a 32K-bit backing store memory that is dual-ported with the ICAP. The backing store is also referred to as the CAAPP-ICAP shared memory (CISM). The architecture of a CAAPP cell is shown in Figure 3, and a table of CAAPP instructions is shown in Figure 4. The custom VLSI chip layout containing 64 CAAPP Processing Elements (PE's) is shown in Figure 5. The CAAPP chip is being built in 2-micron CMOS via the DARPA MOSIS facility and will contain roughly 120,000 transistors.

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

A count of all responding cells is also available at the global controller. The counting operation is used to gather statistics about an image and the results of processing. For example, through counting we may quickly determine the mean and standard deviation of an attribute value for a given set of processors (see section 3.4). Each ICAP processor receives the count for the 8×8 subarray of the CAAPP associated with it.

Communication among CAAPP cells may take place in four different ways. One way is through global feedback and rebroadcast. This method is used when all or most of the CAAPP processors must be told the value of one of the processors (e.g. broadcasting the maximum value so that all cells can normalize their values). A second way is via the ICAP; in some cases it is more efficient to transfer CAAPP data to the backing store and let the ICAP move it across the array and place it in the backing store of the appropriate CAAPP cell. The third way uses the nearest neighborhood (S,E,W,N) mesh, which allows a CAAPP processor to read a bit from up to two of its neighbors at once. This is similar to the network employed in other mesh-connected SIMD parallel processors. The remaining communication mechanism is described in the next section.

2.2 The Coterie Network

The fourth means of communication among CAAPP processors involves a new and powerful variation on the nearest-neighbor mesh called the Coterie network. This is similar to the reconfigurable buses proposed by Kumar [Kumar, 1985], Miller and Stout [Miller, 1987] and the polymorphic torus proposed by Li [Li and Moresca 1987], but differs in that it allows general reconfiguration of the mesh, and multiple processors to write to the mesh at the same time. By adding the simple switch network shown in figure 6, it is possible, under program control, to create independent groups of processors that share a local associative Some/None feedback circuit. The isolated groups of processors can then respond

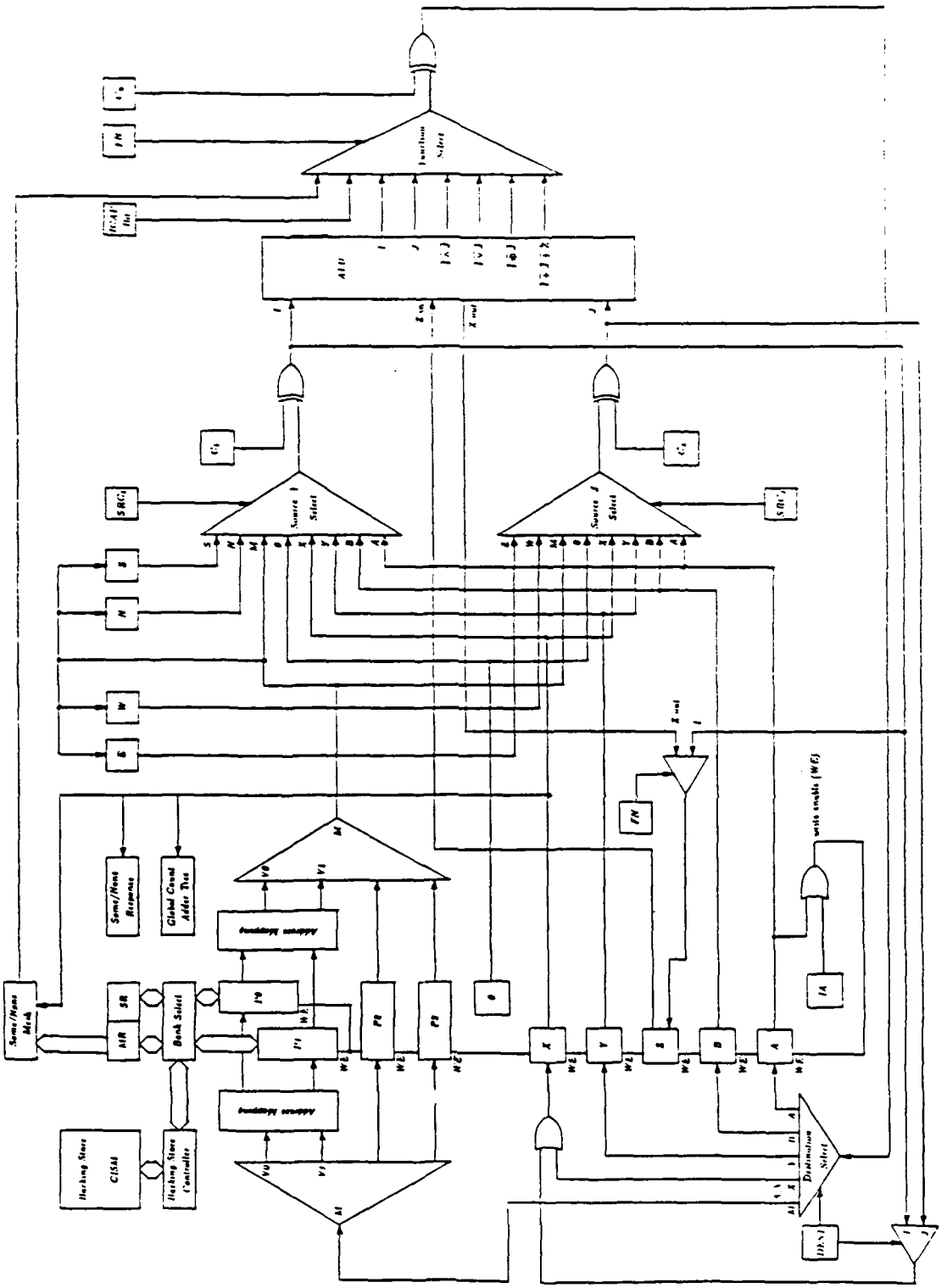

Figure 3: CAAPP Cell Architecture

31	27	23	18	13	9	8	0
0 | INH | C_r	Ftn	C_i	S_i	C_j	S_j	Dest	0	Address

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

$$I \Leftarrow C_i \oplus S_i$$
$$J \Leftarrow C_j \oplus S_j$$

	S_i	S_j	Ftn	Dest	
0	ZERO	ZERO	$Coterie \Rightarrow R$	X_{pr}	0
1	C	C	$I \Rightarrow R$	A, X	1
2	S	E	$J \Rightarrow R$	$A, X \Leftarrow I$	2
3	N	W	$\overline{I \wedge J} \Rightarrow R$	$A, X \Leftarrow J$	3
4	Y	Y	$\overline{I \vee J} \Rightarrow R$	Y	4
5	X	X	$\overline{I \oplus J} \Rightarrow R$	X	5
6	B	B	$I + J + Z \Rightarrow R$	B	6
7	A	A	$ICAP\ C \Rightarrow R$	A	7
8	memory	memory	$I \Rightarrow Z$	memory	8
9	—	—	$memory \Rightarrow MR$	—	9
10	—	—	$memory \Rightarrow MR, SB$	—	10
11	—	—	$MR \Rightarrow memory$	—	11
12	—	—	$MR, SB \Rightarrow memory$	—	12
13	—	—	—	—	13
14	—	—	—	—	14
15	—	—	—	—	15

$$Dest \Leftarrow C_r \oplus R$$

Figure 4: CAAPP Instructions

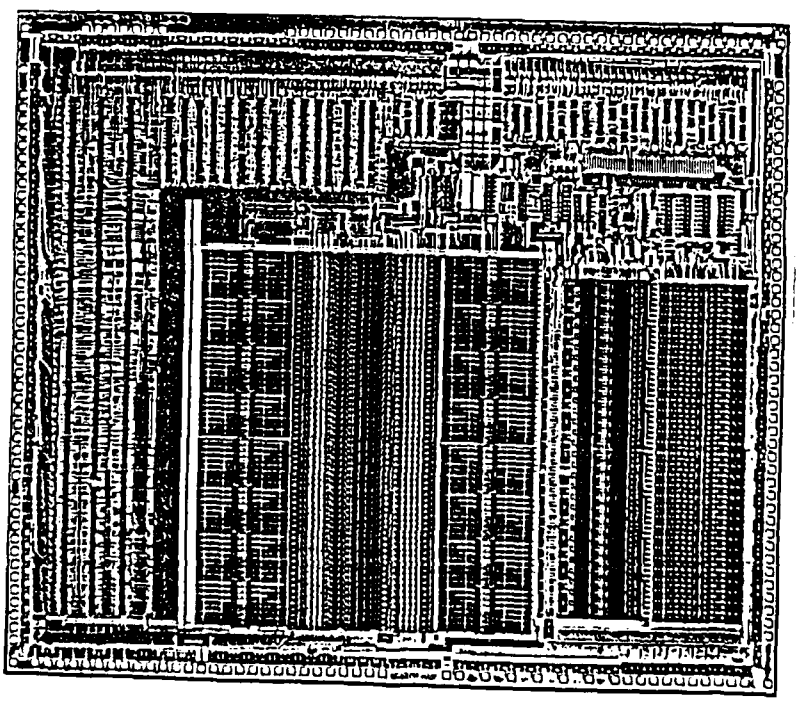

Figure 5: 64 Element CAAPP Test Chip

to globally broadcast instructions in a locally data-dependent fashion (i.e. local SIMD) which permits parallelism to be employed with more flexibility. For example, suppose that an image is divided into a large number of regions and that we wish to determine some attribute for each of the regions. In a typical SIMD architecture, this would be done by sequentially selecting each region for analysis or in parallel by complex communication between neighbors where the attribute is computed via a propagating wave that checks region labels at each step. However, using the coterie network in the CAAPP, in many cases, all regions can perform their own local evaluation in parallel without having to check region labels after one initial step of neighbor comparison.

The name Coterie Network is based upon the similarity of the isolated processor groups to a "coterie": that is, a group of people who associate closely because of common purposes, interests, etc. [Random, 1987]. The isolated groups of processors are thus referred to as coteries. Note that the Coterie Network is separate from the nearest neighbor-mesh, which we refer to as the SEWN Mesh.

Creation of a set of coteries typically begins with opening all of the switches that link processors. Using the SEWN Mesh, the processors compare their own values with the values of their neighbors. They then close the switches that connect them to neighbors with similar properties, leaving open the switches that would connect them to dissimilar neighbors. Of course similarity can be defined by an operation such as a global broadcast of a threshold and a local comparison. In this way, processors with similar properties establish independent coteries. It should be fairly obvious to the reader that, among other things, each region of a segmentation could be a coterie of cells. Because the CAAPP processors can save and restore the switch settings that make up a set of coteries, it is possible to reconfigure the Coterie Network from one processor interconnection pattern to another by broadcasting a single instruction.

Within a coterie, there is a network of wire to which all of the processors are connected. Each active CAAPP processor may be instructed to output a bit onto its coterie's network and then read whatever bit value is currently on the network within its coterie. When more than one processor in a coterie tries to output a bit onto the network the value that appears on the wire is the logical OR of the output bits of all of the processors in the coterie. The shared network is thus functionally equivalent to the global Some/None feedback circuit except that its output is locally formed and only available within a coterie. In addition to its associative feedback function, the Coterie network can be used to broadcast a value from a single processor to every member of the coterie as in a Broadcast Protocol Multiprocessor [Levitan, 1984]. This is accomplished by first selecting a single processor within each coterie, using an associative search operation in parallel within all coteries. Subsequent instructions for placing a value onto the network will only be performed by these selected cells. However, all of the cells will perform the operations for reading the value that is on the network. In this way, the Coterie Network can be used for local broadcasting of data values. The local feedback and broadcast processes can occur in every coterie in parallel.

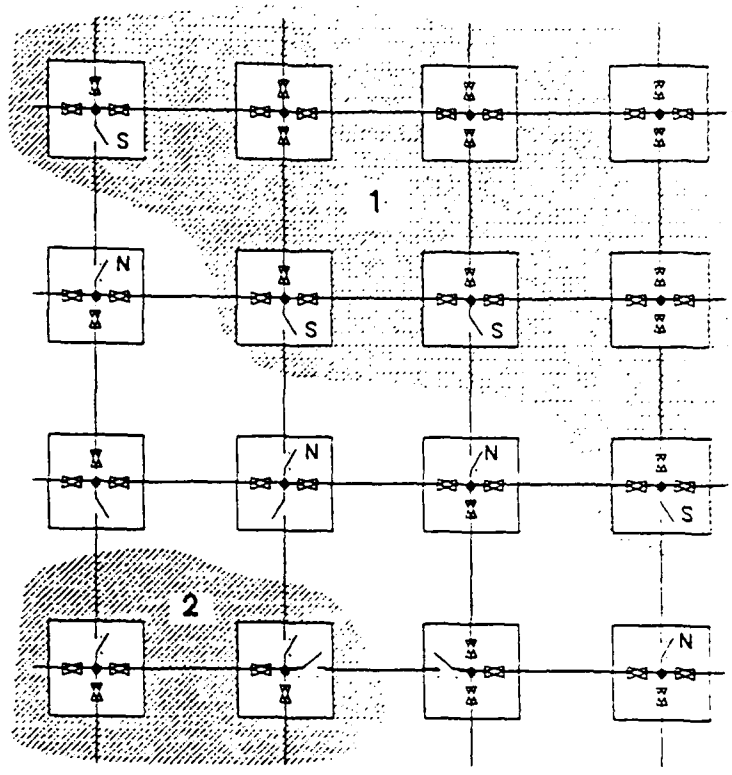

Figure 6: The Coterie Network

2.3 Inter-level Communication Between the CAAPP and ICAP

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

2.4 The ICAP (Intermediate) Level

The ICAP is designed to manipulate tokens (symbolic descriptions of extracted image events and their associated attributes) at the intermediate level and to support data base functions that allow access to these tokens by grouping processes running on the ICAP, and by symbolic interpretation processes, running on the SPA processors. For example, the recognition of a house roof in an image may require the ICAP to group together long, straight, parallel lines, and then to extract parallelograms that are candidate roof outlines. Should the need arise, the results of further processing in the CAAPP can be integrated with the representation in the ICAP because the ICAP representation is in approximate

registration with the original image events in the CAAPP.

The ICAP is a square grid (64 × 64) array of Texas Instruments TMS320C25 16-bit digital signal processor chips. Each of the 4096 ICAP processors consists of a CPU, 256K bytes of local RAM, 384K bytes of dual-ported memory for interacting with the CAAPP and SPA, and network communications hardware. The ICAP processors operate at 5 million instructions per second and can perform a 16-bit multiply-and-accumulate operation in a single instruction time. In addition to its speed, a digital signal processor was chosen at the ICAP level because its instruction set and arithmetic capabilities are well suited for performing computations in spatial geometry. Three-dimensional geometric projections, computing distances, and matching operations are common operations that may be needed at the intermediate level of vision processing.

Control of the ICAP is provided by the ACU (in Synchronous-MIMD mode) and by the SPA (in MIMD mode). Once sensory events have been extracted and represented symbolically at the intermediate level in the ICAP (and continue to evolve as grouping operations take place), each of the SPA processors may then query the ICAP in parallel to establish and verify hypotheses. The ICAP provides three different global OR outputs available to the controller that can be used to determine the status of processing in the ICAP array. The choice of meaning for each signal is left up to the programmer. For example, the programmer may choose to have them indicate completion of a task in the ICAP array with or without exceptions. Another use is as an associative Some/None mechanism. Also provided, is a global summation mechanism is also provided that uses the global count hardware in the CAAPP to form a sum of an 8-bit value from each ICAP processor.

The horizontal links between the ICAP processors provide the intra-level communications necessary for grouping and merging processes to operate on token attributes and token relations within the intermediate symbolic representation. In the IUA prototype, which has only 64 ICAP processors, the bit-serial I/O links between the processors are connected through a centrally controlled 64 × 64 bit-serial crossbar switch. Thus it is possible to establish any point-to-point network topology in the prototype ICAP. Various methods of extending the ICAP communications network to the full-size ICAP array are under consideration.

2.4.1 Interconnection Network

The ICAP connection network is used to set up a connection pattern between 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. The Parallel Communication Switch (PARCOS) chip is capable of broadcasting, allowing the connection network to realize any of the possible 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.

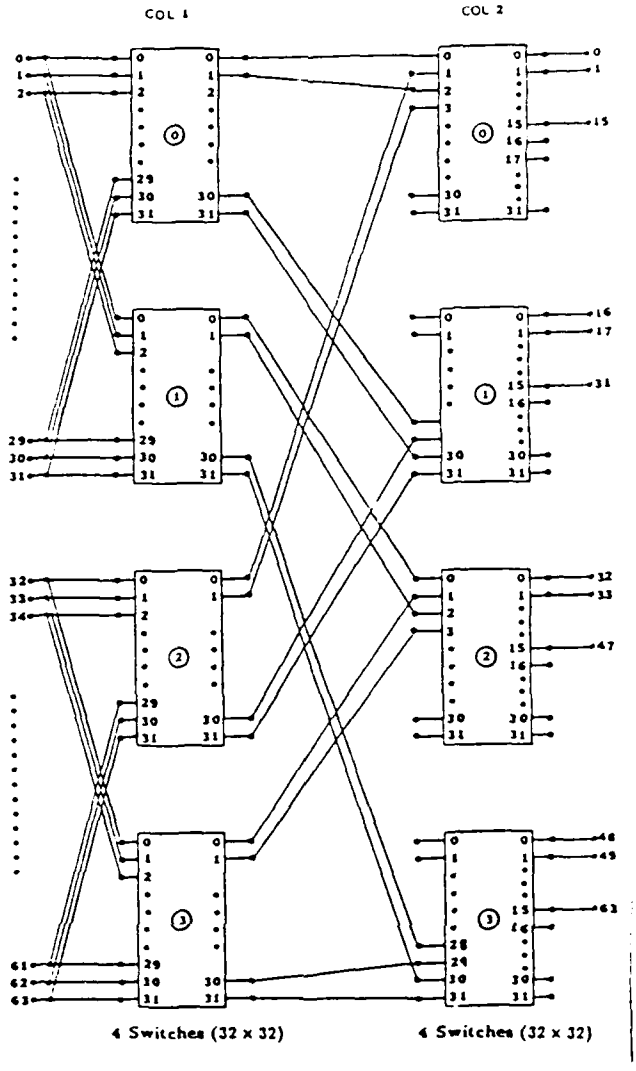

Figure 7: A 64-Input 64-Output Connection Network

The 64-input 64-output connection network for the IUA prototype uses 2 stages of 32×32 PARCOS chips. The PARCOS chips are connected to make a 64×64 cross-bar switch with broadcast capability as shown in Figure 7. A detailed discussion of the network can be found in [Rana 88].

2.4.2 The Parallel Communications Switch (PARCOS)

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 8. Multiple PARCOS chips can be used to build larger connection networks, such as the 64×64 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

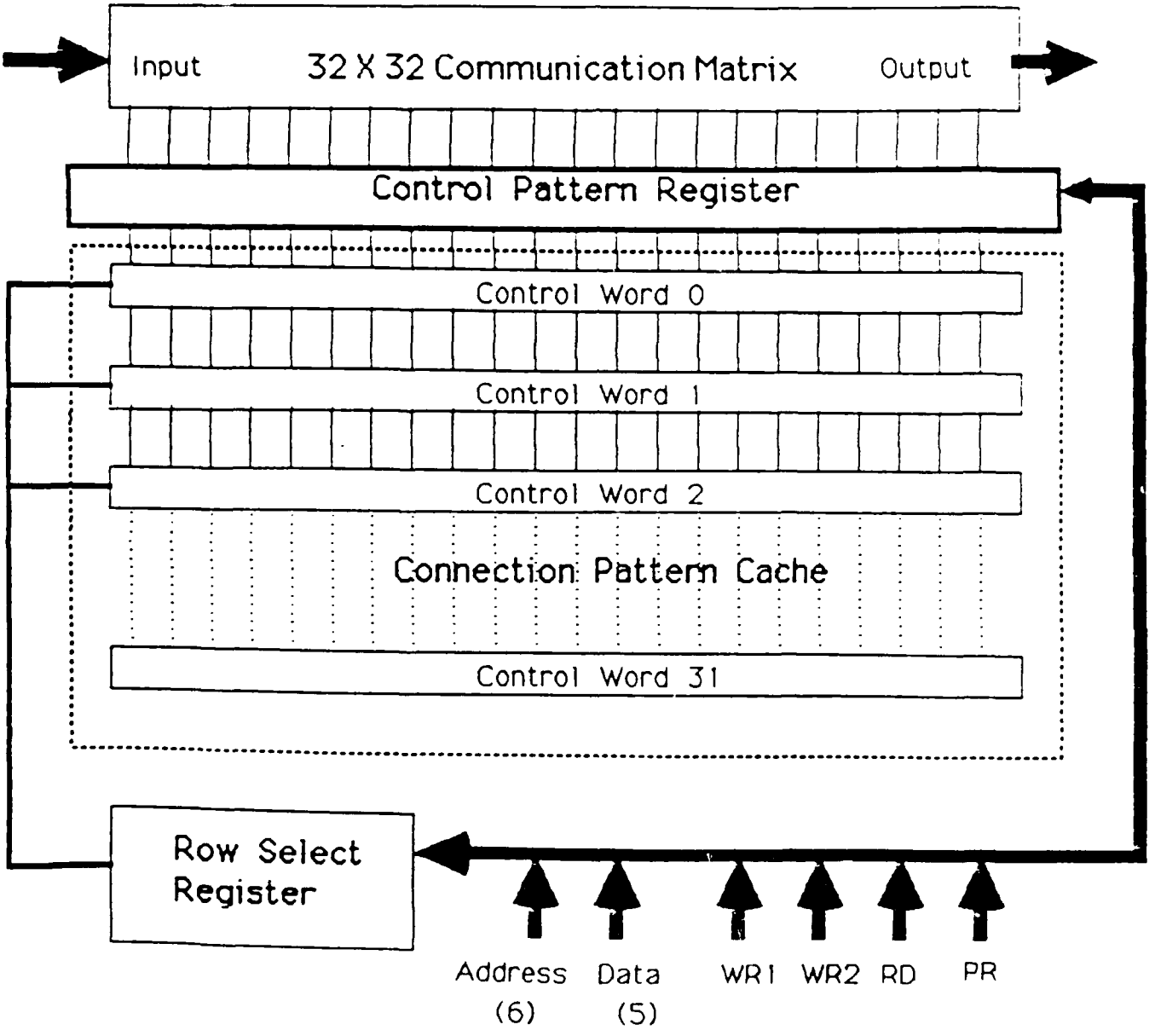

Figure 8: Parallel Communications Switch (PARCOS) Organization

same input, providing *broadcast* mode capability. Figure 9 illustrates one multiplexer tree. It will be noted that there are two multiplexer trees, one made out of n-channel transistors and the other made out of p-channel transistors, with their outputs connected together. By properly sizing the two types of transistors, we have achieved near equal delays for both low-to-high and high-to-low transitions at the output. 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 one of 32 inputs for each multiplexer, or $32 \times 5 = 160$ 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, a row number is first set in the Row Select Register (RSR). The 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.

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, scalable, CMOS technology available through 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. A microphotograph of the chip is shown in Figure 10.

2.4.3 Future Modifications

The design of the PARCOS chip was limited by the number of pins and not by the silicon area. A study for redesigning the PARCOS chip is underway with the goal of providing a 64×64 communication matrix with more than one hundred control words. Also, mecha-

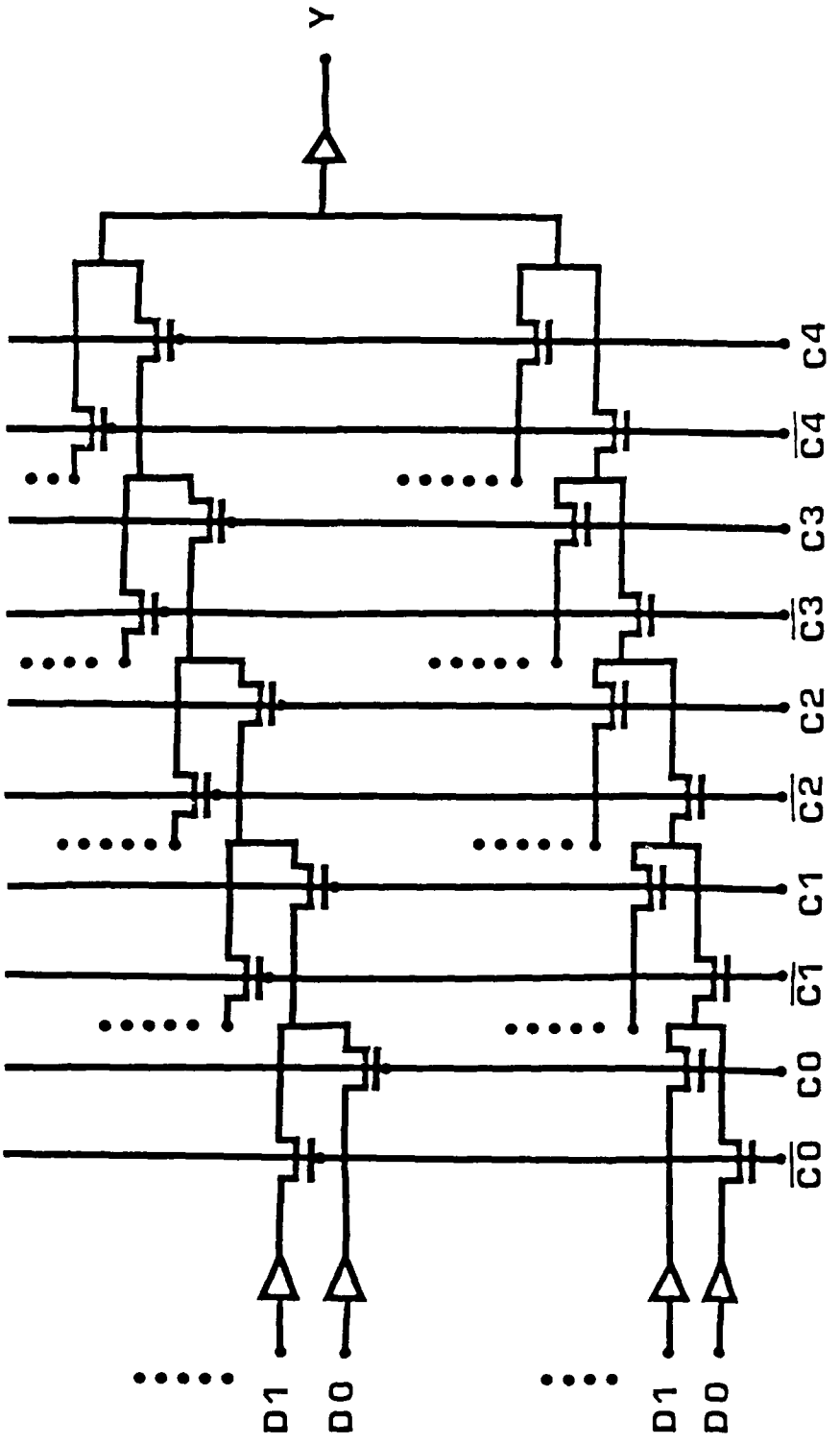

Figure 9: Multiplexer Tree

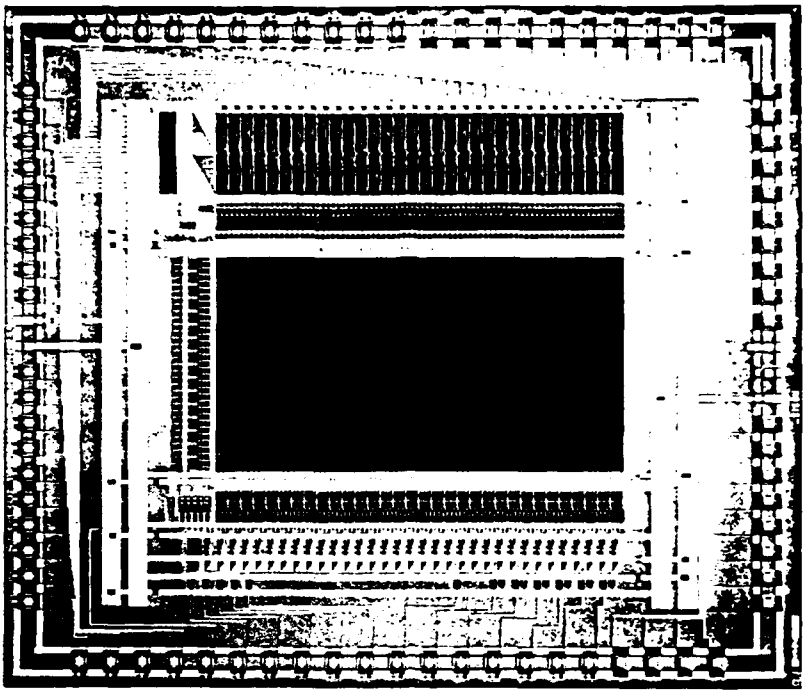

Figure 10: Microphotograph of the PARCOS Chip

nisms will be provided for self-routing in these chips in $\ominus$ (1) time. A 64 $\times$ 64 single chip implementation will allow us to build a 4096-input, 4096-output network by connecting 512 of these chips in a modified 4-stage, strictly non-blocking Clos' [Clos 53] topology or by connecting only 192 of these chips in a 3-stage, rearrangeably non-blocking Benes' [Benes 62] topology.

Currently, it is not possible to directly copy or combine connection patterns in the CPC. Adding the ability to copy one CPC connection pattern into another under the control of a mask register will allow us to build new connection patterns from old ones, which will reduce the time required to create a new pattern in certain cases.

2.5 Inter-level Communication Between the ICAP and SPA

The ICAP-SPA Shared Memory (ISSM) provides the principal communication path between the top two levels of the IUA. It is viewed as an I/O device by each ICAP processor. A given ICAP processor can write (or read) values to (from) an I/O buffer in the ISSM. The ICAP then initiates a block transfer between the I/O buffer and a page of its choosing in the ISSM RAM. An ICAP processor may only access the 128 K-byte segment of ISSM that is associated with it. However, each SPA processor has global access to the entire ISSM for all of the ICAP processors. This structure allows processes in the SPA to access the results of ICAP processing regardless of their spatial locations in the array.

2.6 The SPA: High Level Processing

The SPA processors will run a LISP-based blackboard system [Erman et al. 1980; Nii, 1986, Draper et al. 1987, 1989], through which the various knowledge-based processes can communicate while cooperatively constructing an interpretation of an image and determining the relationships of the various image components to stored knowledge. From the point of view of the blackboard system, the CAAPP and ICAP will appear as knowledge sources at different levels of abstraction. Knowledge-based processes in the system can activate different processes in the CAAPP and ICAP either for the full array or for independent sub-arrays. Thus, the SPA processors operate in MIMD mode with communication through the blackboard. The detailed architecture of the SPA has not yet been fully defined. In the first prototype of the IUA, which is a 1/64th vertical slice of the full IUA, the SPA will be a single Motorola M68020 class processor, augmented with a symbolic co-processor. A separate research investigation within the UMass VISIONS project is currently exploring the implementation of cooperative algorithms and data structures using a commercially available shared-memory multiprocessor at the SPA level [Draper et al. 1988]. This experience is providing additional direction to the future scaling up of the IUA at the SPA level.

Currently, the full SPA is envisioned as consisting of 64 or more processors, each capable of running LISP. Each processor will have some local memory and will have access to a global shared memory that will include the ISSM and the blackboard. The shared memory decouples the SPA processors from the locality of information in the image.

2.7 The ACU: Controlling the CAAPP and ICAP

One major design goal for the Array Control Unit (ACU) was to maximize the rate at which instructions are issued to the CAAPP. This meant that the overhead for controlling loops, branches, and subroutine calls in the ACU had to be minimized. A second major design goal for the ACU was to minimize the cost of implementing a complete development environment for it. Preferably, the ACU would execute a commonly used instruction set so that software could be transported from an existing machine.

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

The Macro-controller is a standard, off-the-shelf processor that brings with it the wide range of software tools that are available for such a processor. It can issue instructions to the CAAPP in two ways. The simplest way is to take direct control of the instruction bus and write out data values that will be interpreted as instructions by the processor arrays. Even at its maximum rate, however, most standard processors can only issue instructions

at about one-tenth of the rate that the CAAPP can execute them. The second method for the Macro-controller to issue instructions is to issue subroutine calls to the Micro-controller.

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

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

Although the only source of instructions for the CAAPP is the ACU, the ICAP processors each have their own program memory. The ICAP program memory is loaded with a large library of service routines upon system initialization. The way in which the ACU issues instructions to the ICAP is by storing a user program into ICAP program memory and then issuing an interrupt to the ICAP that causes it to jump to the user program. (The program is broadcast to all of the ICAP program memories in parallel.) An ICAP user program is typically just an execution script (written in C, Forth, or assembly language) of calls to the ICAP library. Thus, the ACU and ICAP interact very little when a program is running in the ICAP; the exception is when the ICAP program reaches a global synchronization point – this must be mediated by the ACU. The ACU can also set the ICAP to operate in MIMD mode, by turning control over to a task queuing program in the ICAP processors. The queuing program reads execution scripts from the ISSM according to a predefined protocol. When the ICAP is executing in MIMD mode, it depends upon the SPA to provide coordination of any required synchronization between ICAP processors.

The ACU thus supports the close interaction between the CAAPP and ICAP during the initial phases of interpreting an image. However, the ACU also permits the CAAPP and ICAP to work independently, with the ICAP taking directions from the SPA as the high level interpretation processes come into play. This allows the CAAPP to concurrently perform additional low-level processing, such as integrating information from other sensors or starting to process the next image.

3. IUA Programmer's Model

The IUA application programmer will view the system as a shared memory multiprocessor (the SPA) with a pair of tightly coupled array processors (the CAAPP and ICAP). Programs will be written primarily for the SPA level, with the ACU treated as an additional SPA processor. Thus, standard high level language mechanisms will be used to communicate and control concurrency in the SPA and between the SPA and the lower levels.

There will be two types of SPA processes: those that run purely at the high level, and those that service requests for low and intermediate level processing. The former run in the actual SPA processors, while the latter are executed by the ACU. The SPA operating system will automatically recognize ACU programs as having special resource requirements and schedule them for execution only by the ACU.

Programs that run on the ACU will make use of additional language extensions that support attached array processors. Libraries of macros and subroutines will also be available for common low and intermediate level operations. The interface to the CAAPP will be tightly coupled, with the ACU program actually issuing all of the commands on which the CAAPP operates. The ICAP interface is more loosely coupled, with the ACU program issuing processing requests to the ICAP array (in the form of execution scripts), and controlling synchronization of the ICAP processors with each other and with the CAAPP.

New library routines for the CAAPP are developed in a two stage process. First, a routine is rapidly prototyped using the high level language features provided for application programs. If maximum speed is not required, then this will be sufficient. However, if speed is a primary issue, then the rapid prototype algorithm will be migrated into horizontal microcode for the microcontroller portion of the ACU. Because the microcontroller is not a standard processor, it is reasonable to expect that the set of tools for the standard architecture macrocontroller will always be more complete and convenient than those for the microcontroller. Thus, it is unlikely that new library routines will ever be commonly developed directly for the microcontroller.

New routines for the ICAP processors will be cross-compiled or assembled on the host, and downloaded to the ICAP program memories at system initialization. Each ICAP is programmed as a uniprocessor with two serial I/O ports and access to a special data memory. The I/O ports allow communication with other ICAP processors, and the memory provides the interface to the CAAPP. Communication with the ACU is via interrupts from the ACU, a set of three Done response bits, and a global Sum of 8-bit values output from the ICAP processors. The ACU can also halt the ICAP processors, write a script into program memory, and then issue an interrupt that causes the ICAP processors to jump into the script. When an ICAP processor reaches a synchronization point in the script, it issues a Done signal and then waits for the ACU to send an interrupt that allows it to continue. A script consists mostly of calls to the ICAP routines that were downloaded at system initialization.

Rather than writing new library routines for the ICAP, most applications programmers will write ICAP scripts. A script will consist of an ICAP portion and an ACU portion. The ACU downloads the ICAP portion at the start of script execution. The ACU portion of the script is then executed, which begins by activating the ICAP portion. The two portions then interact as necessary for operations such as synchronization of the ICAP processors, or broadcasting of parameters. Note that the ACU portion of an ICAP script can contain CAAPP commands, so that it can control interactions between the two arrays.

The script mechanism will also be used in writing SPA programs that control the ICAP in MIMD mode. When used with the SPA, a script has an ICAP portion and an SPA portion. Because of differences in the hardware interfaces between the ICAP and ACU, and the ICAP and SPA, the formats of the two types of scripts will not be identical. However, they will follow the same general form and be as similar as possible. Most importantly, the script mechanism will provide a common data format interface so that the same software can be used with ICAP output data on either the ACU or the SPA. This is in keeping with the philosophy that the ACU should appear, as much as possible, to be just another SPA processor.

4. Sample Algorithms

The purpose of this section is to provide a sense of the types and range of operations that can take place on the IUA. It is by no means a complete discussion of all of the system's capabilities. The algorithms presented here are specifically intended to demonstrate the various forms of communication that occur within and between the CAAPP and ICAP levels. A processing scenario that involves the SPA level is outlined in Section 4.

This section will begin with several fairly simple, but detailed, algorithms in order to show how the IUA is programmed. The algorithmic notation used is very close to one of the programming languages employed with the IUA software simulators. However, the notation is simplified to improve the clarity of the presentation. Macro operations are also used where their machine language implementation is not an important element of the algorithmic method. For example, adding two 8-bit quantities in the CAAPP is actually performed bit-serially by a sequence of instructions, but a typical program will make use of the standard macro for addition to perform this operation. Following the detailed algorithms, several more algorithms will be sketched whose complexity makes them too lengthy to be presented here in detail. The concepts behind the algorithms are worth considering, however, because they demonstrate additional capabilities of the IUA.

4.1 Select Greatest Responding Value

The algorithm to find a maximum value in the CAAPP (Figure 11) demonstrates the use of the associative Some/None feedback from the CAAPP array. The goal is to select, from among all active cells, the cell or cells that have the greatest value in a given field of their memory. In addition, that value is to be made available in the ACU for subsequent processing.

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

4.2 Label Connected Components

The local associative Some/None operation provided by the Coterie Network is demonstrated by the label connected components algorithm. Because finding a maximum uses

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

Figure 11: Finding a Maximum Value in the CAAPP

only broadcast and Some/None feedback, it can be performed locally within a coterie and in parallel with every other coterie. This leads to the simple algorithm for computing a connected component labeling of an image shown in Figure 12.

The algorithm begins by loading each processor with its address in the array from the backing store memory, which serves to give each processor a unique number. Next, the Coterie Network switches are opened between processors that are on region boundaries (i.e. between pairs of processors that have different values), establishing a coterie for each image region. Lastly, all regions in parallel determine their local maximum address value. Note that this is the same algorithm as for finding a maximum value in the entire array except that the coterie Some/None response is used in place of the global Some/None response to control the setting of activity. As part of finding the maximum, every processor in a coterie stores the maximum address value for all cells in its coterie in its own memory. Because this value is different for every region, the result is that each connected group of processors is assigned a unique label that is common to every processor within a group. From our electrical simulations of the Coterie Network, we calculate that this algorithm will take approximately 50 microseconds to execute.

4.3 Histogram

The preceding algorithms use only the Some/None Response form of feedback. The response count is of equal importance in many of our algorithms. For example, we can form a histogram, of any numerical feature in the CAAPP using the response count. This is quite simple to do: For each bucket in the histogram, we associatively select those cells whose values fall within the range of the bucket by broadcasting the minimum and maximum value of the range, comparing with the cell's value, and appropriately setting the response register to 0 or 1; then a count of the responding cells gives the histogram bucket value. Thus, the time to form the histogram is proportional to the number of buckets in the histogram (typically about 1.6 microseconds per bucket), and is independent of the number of values in the array.

```
Load_Processor_Addresses

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

Figure 12: Connected Component Labelling using the Coterie Network

```
Sum:=0                                                        {Initialize sum}
FOR Bit_Num:=High_Order DOWN TO Low_Order DO   {Count each bit in field and add to sum, scaling appropriately}
      Response:=Field[Bit_Num]
      Sum:=Sum*2+Response_Count
Response:=Activity                                            {Count number of active cells}
Mean:=Sum/Response_Count                                      {and compute mean}
```

Figure 13: Computing the Mean of Values in Selected CAAPP Cells using the Response Count Operation

4.4 Compute Average Value

Figure 13 gives a CAAPP algorithm that uses the response count to compute the mean of the values stored in selected cells. The algorithm begins by summing the values in the selected cells. Starting with the high order bit position of the values to be summed, each bit of the values in the selected cells is separately counted. The counts are each added to the overall sum after being appropriately scaled by a power of two. The algorithm concludes by setting each cell's response bit equal to its activity bit so that the response count will be the number of active cells, and dividing the sum by that count to get the mean of the values.

```
{Compute X Magnitude}

   Double_Own := Own_Value + Own_Value
   Int_Result := Double_Own + North(Own_Value)
   Int_Result := Int_Result + South(Own Value)
   X_Magnitude := East(Int_Result)
   X_Magnitude := X_Magnitude - West(Int Result)

{Compute Y Magnitude}

   Int_Result := Double_Own + East(Own_Value)
   Int_Result := Int_Result + West(Own_Value)
   Y_Magnitude := North(Int_Result)
   Y_Magnitude := Y_Magnitude - South(Int_Result)
```

Figure 14: Sobel Algorithm for the CAAPP

4.5 The Sobel Edge Operator

Of course, in addition to processing that is oriented around associative feedback, the CAAPP is able to perform the usual image processing and low level vision algorithms that do not depend upon feedback to the controller. For example, smoothing operators such as Gaussian convolution, edge detectors such as the Sobel and Canny operators, local pixel comparisons, line curvature, border following, etc. all use the mesh connected operations that are typical of this class of machines.

To demonstrate communication between neighboring CAAPP PEs, the algorithm for performing a Sobel operation is shown in Figure 14. The Sobel computes the local X and Y gradient magnitude at each pixel in an image. These X and Y magnitude vectors subsequently can be combined to form the orientation and magnitude of the local gradient at each pixel. Pixels with large gradient magnitudes are frequently associated with strong edges or lines in an image, and therefore are likely to be of use in interpreting an image.

The Sobel operator requires that the image be convolved with two different 3×3 masks. One mask computes the gradient magnitude in the X direction while the other computes the magnitude in the Y direction. The masks are:

X magnitude:	Y magnitude:
-1 0 1	1 2 1
-2 0 2	0 0 0
-1 0 1	-1 2 -1

In the CAAPP, the X magnitude is computed by first having each cell double its own

value, and then add the values of its North and South neighbors. This intermediate result is then used to compute the actual X magnitude by subtracting the intermediate value of each cell's West neighbor from the intermediate value of its East neighbor. A similar sequence of operations is used to compute the Y magnitude.

Assuming that the operation is being applied to 8-bit integer pixels, it would require 100 CAAPP instruction cycles (10 microseconds) to compute the components of the Sobel operator for the image. The gradient magnitude and orientation can be computed from these components by applying the standard formulas as a sequence of CAAPP arithmetic operations.

4.6 Create Border Corner Lists

In addition to performing associative Some/None operations, the Coterie Network may be used to pass messages across the mesh by providing direct links between non-adjacent processors. In the algorithm shown in Figure 15, it is assumed that some corner detection operation has been performed on the borders of regions in the image. The result is a sparse set of processors that are labelled as corners (ie. those processors whose Corner_Tag field is set to true). One useful feature that can be extracted for each region is a list of the positions of its corners. The following algorithm forms the corner lists for all regions in parallel. It begins by making each region an independent coterie using the connected components labelling algorithm presented earlier. A single cell in each coterie is selected as the coterie leader. In this case, the chosen cell is the member of the coterie whose cell address equals the region's component label. The leader is responsible for collecting the corners for its region, and passing them to the ICAP which stores them in a list.

The first corner is determined by selecting the corner cell in each coterie with the maximum address. As part of this process, the coterie leader learns that cell's address and passes it to the ICAP processor associated with the CAAPP chip containing the leader. The selected corner cell is then shut off and the process is repeated so that the next corner is selected. The loop ends when there are no more corners to select, at which point every corner will have been passed to the ICAP by its coterie leader.

This algorithm causes the corner lists to be created in reverse raster-scan order, which is adequate if the regions are simple convex figures. However, for more complex regions, it may be difficult to reconstruct the shape of a region given a corner list in this order. A better arrangement is to list the corners in clockwise (or counterclockwise) boundary traversal order (i.e. the order in which corners would be encountered as the cells at the boundary of the region are traversed, starting from some arbitrary boundary point). The coterie network can also be used to accomplish this task. Although the basic concept for doing this is quite simple, in practice it is complicated by considerations of regions that are one-pixel wide and regions that completely enclose other regions. Because the algorithm is more complex, it is only discussed here in general terms; the detailed discussion will be left to a future paper.

```
Label_Connected_Components        {Each component is given a unique label that is equal to the maximum cell address
                                  within the component. The label is stored in a field called Component_Label in
                                  each cell                                                                    }
Start_ICAP(Corner_List_Builder)   {A process is started in the ICAP that will respond to each Signal_ICAP operation
                                  (except the first) by picking up a corner from the backing store, and adding it to
                                  the list for the appropriate component region                                }
Activity := 1!                                                          {Turn on all cells}
Leader := Address = Component_Label                                     {Identify the leader for each coterie}
Backing_Store_Write(Leader)                                             {Copy leader tags to backing store}
Response := Leader
Latch_Local_Count                                                       {Count of leaders in each CAAPP chip}
Signal_ICAP                       {The ICAP processor associated with each CAAPP chip reads the local count to
                                  determine the number of coteries for which it will collect corners. Each ICAP
                                  processor also scans its portion of the backing store to determine the addresses of
                                  the leaders in the coteries associated with it                               }
Activity & Response := Corner_Tag                                       {Activate all corner cells}
WHILE Some DO                                                           {Select corner with greatest address in each coterie}
    FOR Bit_Num := Address_Length - 1 DOWNTO 0 DO
        Response := Address[Bit_Num]
        Next_Corner[Bit_Num] := Coterie_Some!   {Store each bit of the greatest address in all members of the
                                                coterie (including the leader) by ignoring the activity bit  }
        IF Next_Corner[Bit_Num] THEN
            Activity := Response                                        {Turn off cells that are less than the max}
        Corner_Tag := False                                             {Disable the greatest corner once it's found}
        Backing_Store_Write(Next_Corner)                                {Copy the address to the backing store}
    Signal_ICAP                   {The ICAP processor associated with each CAAPP chip picks
                                  up the address stored in each coterie leader location in its por-
                                  tion of the backing store and saves it in a list             }
    Activity & Response := Corner_Tag!                                  {Activate remaining corners}
END WHILE
```

Figure 15: Creating Lists of Border Corners

As in the preceding algorithm, the first step is to label connected components. After connected component labelling has been performed, each member of a component examines its neighborhood to determine whether any neighboring cell has a different component label. Any cell that has a neighbor belonging to a different region is at the boundary of its own region. In the simplest case, the cells that are at a region's boundary form a chain that is a one-pixel wide closed loop. Each cell in the chain will have two neighbors; one in the clockwise direction and the other in the counterclockwise direction around the loop. The cells that make up a boundary chain can then set their coterie switches so that the chain becomes a separate coterie. (Some of the complexity of the actual algorithm stems from the situations in which a boundary chain is not a simple closed loop and how the different cases are handled.)

A leader is selected for each of the boundary-chain coteries. Each leader opens the coterie switch connecting it to its counterclockwise neighbor in the loop. The loop is thus transformed into an open figure with the leader at one end. Every cell in the chain that is also tagged as a corner now opens the switch connecting it to its clockwise neighbor in the chain. Next, each leader broadcasts a bit to its coterie. Because the corner cells have broken the coterie, the bit will only reach the first corner clockwise along the boundary from the leader. That corner is then activated and transmits its address back along the coterie to the leader which subsequently passes the address to the ICAP. The active corner then closes the switch to its clockwise neighbor and deactivates itself so that further broadcasts from the leader will pass through it. The process is repeated until all corner cells have been read out to the ICAP. Note that all region boundaries are being processed simultaneously through their coterie chains.

4.7 Region Adjacency Graph

A similar algorithm involves collecting a list of adjacent region labels for each region in an image. This algorithm begins by having every boundary cell get the label of its neighbor that is in another region, using the SEWN mesh. Each region then performs a coterie-select-greatest operation on these region labels, and a region label is output to the ICAP via the coterie leader. All boundary cells that have the selected label are then shut off and the test is repeated to obtain the next region label. The process is complete when there are no more labels to output. Because a region label is the address of the coterie leader for a region, and the ICAP processors are spatially collocated with respect to the CAAPP cells, each ICAP processor can directly compute the ID number of the other ICAP processors that contain descriptions of adjacent regions.

4.8 Rule-Based Region Merging

Once the ICAP processors have collected the information required to describe a specific type of image event, for example lines or regions, the ACU can broadcast rules (or con-

straints) to the ICAP that cause it to take some action. Given that the ICAP contains an attribute list for each region consisting of its size, average intensity, list of border corners, and list of adjacent regions, the ACU could broadcast a rule to the ICAP that is equivalent to the statement: "If a region is below size X, and is adjacent to one or more regions that exceed size Y, it should be merged with the adjacent region whose intensity differs least from its own, but only if the intensity difference is less than threshold Z."

Such a compound rule will actually take the form of a processing script that is downloaded to the ICAP processors from the ACU via a broadcast to the ICAP program memories. The script will actually be a series of calls to library routines that have been pre-stored in the ICAP. In this case, the script would select regions larger in size than threshold Y; and have their associated processors transmit their size, intensity, and ID to all ICAP processors that are associated with an adjacent region. The ICAP processors then compare each region that is smaller than X to the size and intensity of each region for which information was received. If the condition for a merge is met, then an ICAP processor transmits all of the information for the smaller region to the processor that is responsible for the larger region. The processor that contains the larger region adds the smaller region's information to its database. The processor that contains the smaller region transmits the new region label to the CAAPP by writing the label into the backing store for the region's coterie leader. The ACU then instructs all coterie leaders that have received new labels to broadcast the new label to their coterie and then resign as coterie leader. The cells that are on the common boundary between the two regions then close their coterie switches so that the two coteries are merged into one. The ICAP processor that was responsible for the smaller region then deletes the region's information from its database.

A variety of interprocessor communication topologies can be supported in the ICAP, due to the flexibility of the centrally controlled network switch. For example, topologies such as the mesh, ring, hypercube, or shuffle can be built with the ICAP switch. In this algorithm, the mesh topology would be used because the communication tends to be local in nature. (Information that is to be sent between non-adjacent ICAP processors is relayed by those processors that are between them.) In order for an ICAP processor to communicate with its four neighbors, it sends to one neighbor while receiving from the neighbor in the opposite direction. Each time the ICAP is ready to switch directions, it signals the ACU which changes the ICAP connection pattern and resynchronizes the serial ports on all of the ICAP processors. Thus, all ICAP processors might initially transmit to the North and receive from the South. When all transmissions to the North are finished, the ACU changes the connection pattern so that all messages will be transmitted to the West and received from the East, and then signals the ICAP processors to start transmitting again.

5. A Vision Processing Scenario for the IUA

The discussion that follows describes one possible sequence of operations on the IUA that could be used to form an interpretation from an image. This is actually a gross oversimplification of how the UMass VISIONS system is used to interpret an image [Draper 1987]. It would be impossible to provide a complete discussion of the full interpretation process on the IUA within the space available in this paper. Our purpose here is merely to show the types of processing and interactions that can take place in the IUA in a context that is larger than a single algorithm.

The processing is initiated with a region segmentation of the image. The first step is to apply an edge-preserving smoothing operator. We use an algorithm which involves a few iterations of a 3×3 window convolution on the CAAPP. The next step is a region segmentation [Beveridge 1987], which uses local histograms within 16×16 windows. In brief, each ICAP computes a histogram for the 8×8 tile of CAAPP cells associated with it. This is done by broadcasting a series of value range comparisons and performing local count operations in the CAAPP. The ICAP simply records the local count for each range corresponding to buckets in the histogram. The ICAPs then merge their histograms through communication with their horizontal neighbors in the same 16×16 window. (Actually, the algorithm utilizes windows that overlap by 4 pixels in each direction, forming 24×24 pixel histograms and requiring a bit more complex communication at the CAAPP and ICAP levels than we have room to discuss here.) Next, the ICAPs search their histograms for peaks and valleys, applying various criteria for defining clusters of values. The ICAPs communicate with their neighbors to consistently extract labels of peaks to be associated with pixels, and generate a cluster label plane that is returned to the CAAPP through the backing store. The CAAPPs form connected components within 16×16 windows, and then perform region merging to remove the artificial seams of the 16×16 windows in order to produce the final segmentation.

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

ICAP array links collinear segments that may have resulted from excessive fragmentation of longer lines in the original image. The result is a set of tokens that describe straight lines of various lengths which correspond to events in the image.

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

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

Next, the ICAP applies sets of object constraints retrieved from the knowledge base in the SPA shared memory to the tokens in the ISR that are resident in ICAP memory in order to form initial object hypotheses. Constraints are minimum and maximum values on token attributes that form a range on accepting tokens as object hypotheses [Hanson and Riseman 1986, 1987; Lehrer et al. 1987]. For each hypothesis, a score and threshold are generated within each ICAP for its token set. During this phase the ICAP processors are essentially running in MIMD mode.

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

The last phase uses the results of the previous two phases to extend hypotheses, detect conflicts between them, and resolve those conflicts. It is at this point that the SPA takes a greater role. The ICAP processors transfer selected token labels to the memory that is shared with the SPA. The different object schemas [Draper 1987] in the SPA apply various grouping strategies to the tokens by issuing commands to the ICAP processors that refer to the token labels and object verification strategies. Through a global blackboard the schemas incrementally attempt to resolve conflicts and find a consistent set of hypotheses with proper spatial and spectral relationships. Algorithms from the previous phases may be selectively repeated with different parameters and for different goals as different strategies for object verification are applied in different areas of the image to arrive at a consistent interpretation of the scene.

6. Performance on the DARPA Integrated Image Understanding Benchmark

In 1987 and 1988, the University of Massachusetts and the University of Maryland developed, at DARPA's request, a new benchmark to test the performance of parallel processors on an image understanding task. In this section we describe the benchmark task and present performance results for the IUA on the benchmark. The IUA simulation showed the capability for the system to solve a vision task roughly as much as 10,000 times faster than a Sun workstation.

6.1 Image Understanding Benchmark Description

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 extracted from a previous image. However, the

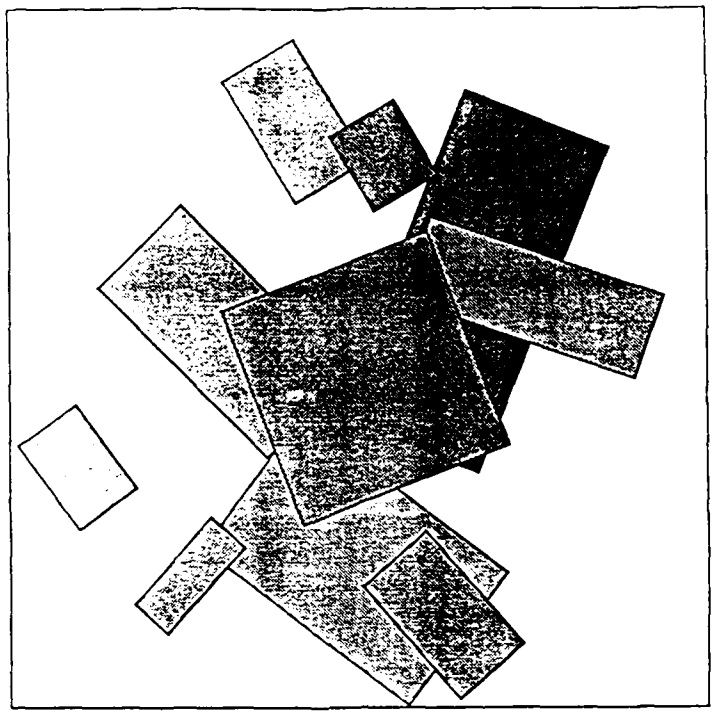

Figure 16: Intensity Image of Model Alone

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 512×512 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 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 16 shows an intensity image of a sculpture alone, and Figure 17 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.

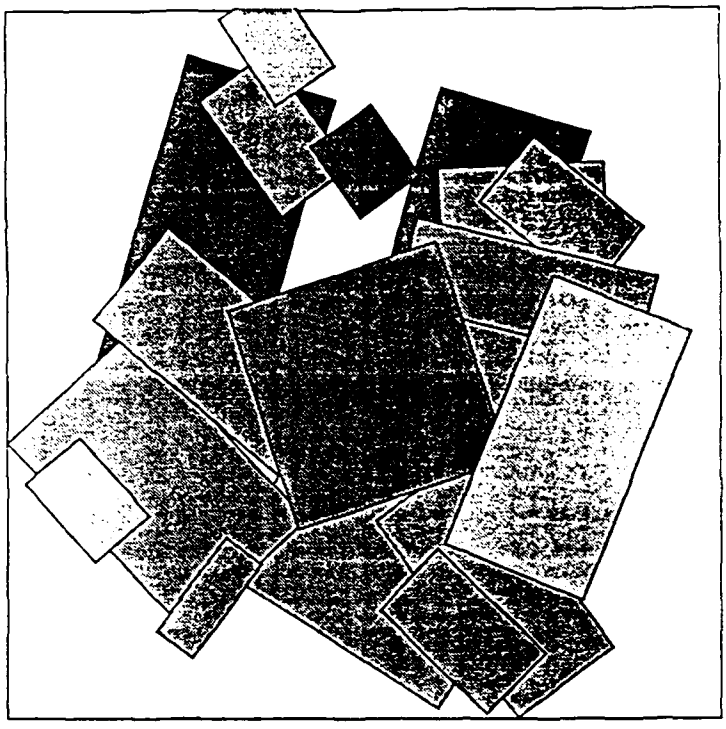

Figure 17: Image of Model with Clutter

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 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. Figure 18 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.

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	

Figure 18: Steps that Compose the Integrated Image Understanding Benchmark

6.2 Image Understanding Architecture Benchmark Results

Because the IUA is still under construction, an instruction-simulator was used to develop the benchmark implementation. The simulator is programmed in a combination of Forth and an assembly language which has a syntax that is similar to Ada or Pascal. 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. Figure 19 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-level processor was also idle roughly 50% of the time while waiting for top-down probes from the intermediate level.

Data Set	Sample	Test	Test 2	Test 3	Test 4
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	.0004095	0.0543250	0.0071766
Window selection	0.0006000	0.0000300	0.0000900	.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
Rectangle 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
Rectangles detected	31	23	19	60	55
Hough probes	44	5	8	84	100
Initial match strength probes	21	20	15	81	80
Extension mat. str. probes	20	1	3	41	54
Models remaining	3	1	1	2	1
Model selected	10	1	5	7	8
Average match strength	0.45	0.86	0.84	0.81	0.84
Translated to	151,240	256,256	257,255	257,255	257,255
Rotated by	85	359	113	23	23

Figure 19: Image Understanding Architecture Results

7. Image Understanding Architecture Program Second Year Summary of Efforts At Hughes

This section is a summary of the implementation/test activities at Hughes over the past year as part of DARPA/ETL funded program to build a feasibility prototype of the Image Understanding Architecture (IUA). The basic IUA includes 4096 CAAPP cells, 64 ICAP cells, a single SPA processor, and an array control unit. The entire prototype will plug into a single-user VME based workstation that will serve as a host.

This prototype consists of one motherboard containing 64 daughterboards, a concentrator board, an interconnect board and four driver boards. The daughterboard contains the custom CAAPP/glue chip, the TI TMS320C25 Digital Signal Processing (DSP) chip for the ICAP level, plus 256K bytes of static (SRAM) and 448K bytes of dynamic video memory (VRAM). Each of the 64 bit-serial processing elements (PEs) on the CAAPP chip contain 320 bits of local data storage. This custom VLSI chip was designed in 2- micron CMOS with 2 metal layers and contains approximately 130,000 devices.

This report summarizes our test results of the custom CAAPP/glue chip and its operations in two different breadboard environments: full daughterboard and two CAAPP custom chips. Due to limited testing time, only first-order debugging has been done.

7.1 Test Report for the CAAPP/Glue Chip

The breadboard shown in Figure 20 is used to test the CAAPP chip. The tests were based on two "good" dies out of 30 received from MOSIS. These chips had only CAAPP circuitry without the glue logic. The processing results of the 64 CAAPP PEs were observed at the edge of the mesh network at the chip boundary through eight consecutive shifting operations. Loading different data for each CAAPP PE also occurs through shifting on the mesh network. We have successfully tested all of the registers including the zero-reg (setting every PE to 1 or 0), page 1 and 2 of memory, and most of the function (ftn) fields. Byte transfer using the mesh registers works correctly for page 1, and validates the CAAPP PE's 8-bit data path. Most of the corner turning circuitry works, but PE 12 is stuck at 1 for both of the good dies. Because the chip timing is very important for successful operation, and because we have only two dies to test, we aren't sure whether this represents a design error or a fabrication error. We've repeated our simulations and they show that this PE should work. We will continue our tests by adjusting the separation interval among the three clocks, and adding extra de- coupling capacitors, without modifying the chip design. We are continuing our debugging process.

7.2 The Full Daughterboard Breadboard

The breadboard, shown in Figure 21, consists of one CAAPP/glue custom chip, a backing store video RAM block, one TI TMS320C25 DSP chip, and an SRAM block for its program

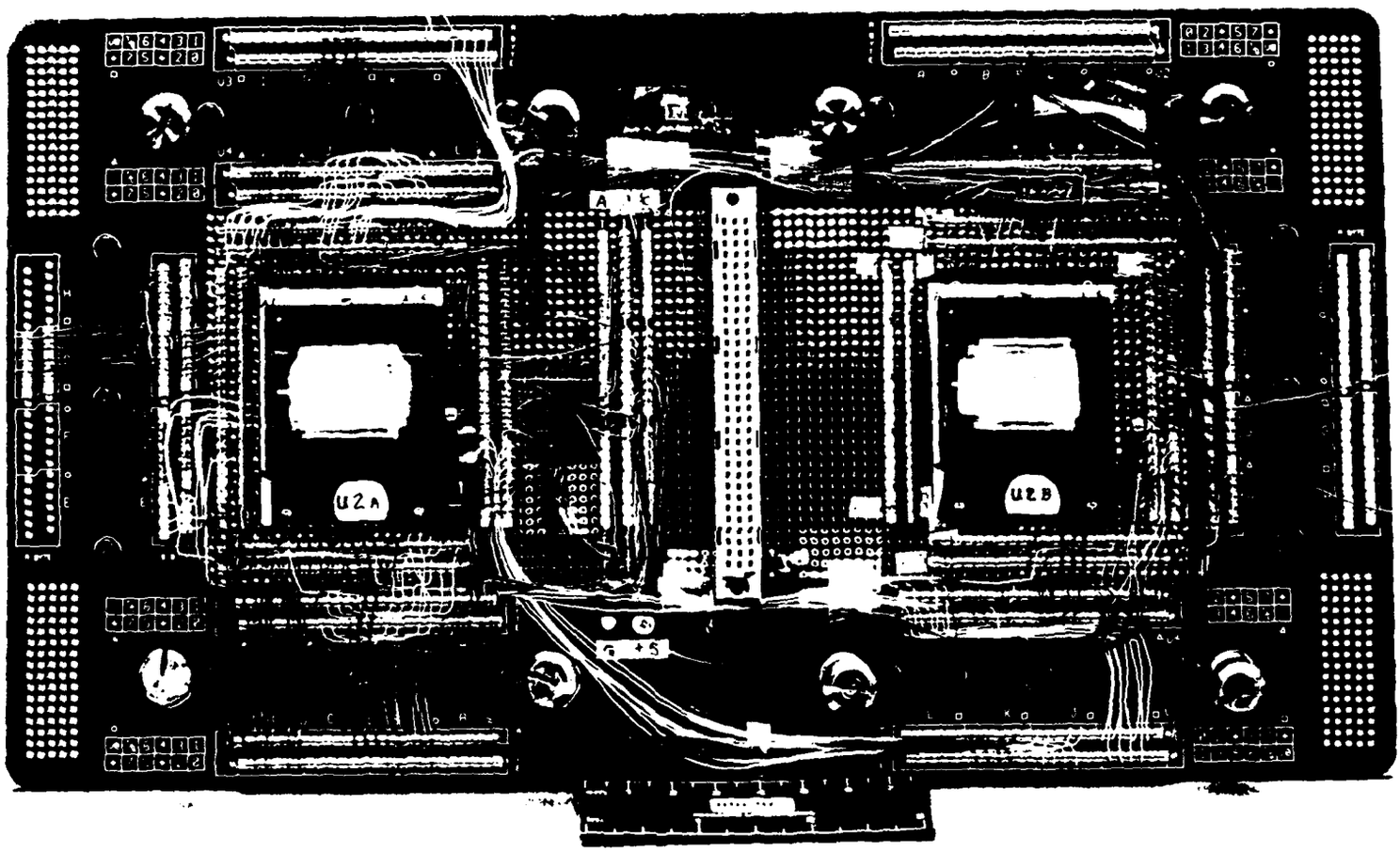

Figure 20: Two-chip Breadboard

and data memory. The daughterboard is intended to be a glue-free cost-effective building block for the full IUA system. To efficiently integrate the five basic components of the daughterboard, all the necessary "glue" circuitry is included in the same full-custom chip containing 64 CAAPP PEs. No external "glue" logic except drivers is required on the daughterboard. The glue logic will interpret CAAPP Chip instructions from the controller, while the CAAPP array will execute only CAAPP PE instructions. The testing we have done so far is based on 50 parts received from MOSIS containing the CAAPP and glue logic. The yield on the glue logic appears to be very high.

The purpose of the daughterboard breadboarding is to verify that on-chip custom glue logic can efficiently integrate the three different processor types and their associated memories in the IUA. Due to limited testing time, only the following functions have been tested: reset the IUA, ICAP/ACU interface, the ICAP memory interface including the response count, and the backing store read or write for the CAAPP PE.

7.2.1 ICAP/ACU Interface

The ICAP/ACU interface circuitry provides the interface between ACU and ICAP along with its program and data memory space. It allows the ACU to place the ICAP into a hold mode, featuring relinquished ICAP bus control, so that the ICAP's memory can be directly accessible to the ACU instruction bus. The interface includes one auto-incrementable 16-bit Address Register (AR) which points to the current ICAP memory location where the data will be written by the ACU. Every time the ACU is finished writing one data item into a specified location, the AR register is auto- incremented. The information written by the ACU can include ICAP program subroutines, routine parameters or data, etc. Similarly, the interface allows the ACU to release the ICAP from the hold mode. It also provides the ACU a means to interrupt the ICAP and force it to start to execute the routines just loaded.

We successfully observed the control signals to hold/release the ICAP by monitoring the contents of the address bus and were able to connect the ACU instruction bus to either ICAP program or data memory space. By monitoring the contents of the address bus, we verified that the AR register did auto-increment after each write. Using the test mode of the connect-to instruction, we observed correct contents on the data bus from both the program and data memory space. In other words, the ACU correctly loaded patterns into the static RAM block in our daughterboard breadboard via the custom chip.

7.2.2 ICAP Memory Interface

ICAP memory interface circuitry provides the interface between the ICAP and its program, data, and I/O memory space. It will decode memory access signals from the ICAP and generate enable signals to one of the four destinations: static RAM (SRAM) chips, CISM or ISSM VRAM chips, or I/O ports in the CAAPP chip; different sets of control signals

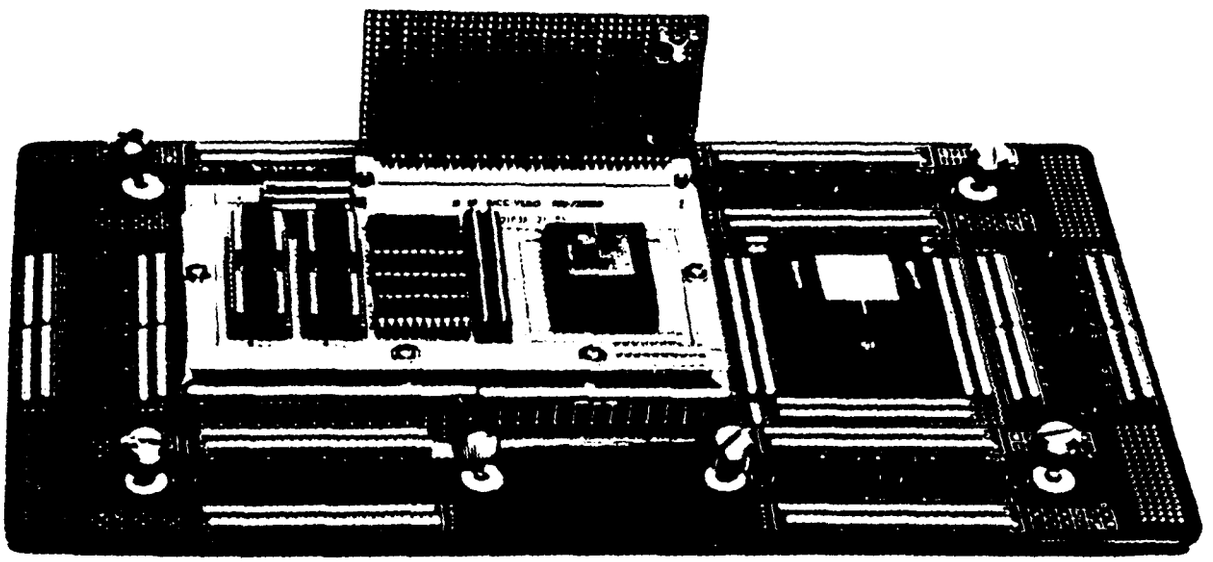

Figure 21: Daughterboard Breadboard

such as write enable and read/output enable signals will be generated for each of the 4 destinations because each has its unique timing requirements. Since SRAM, VRAMs and the CAAPP chip are different physical devices sharing the same ICAP system bus, the interface circuitry has to provide the necessary arbitration by allowing only one active master. The others must have their drivers tri-stated. At times, the interface circuitry will also insert required ICAP wait states for some situations such as access by the ICAP of a new page in CISM or access of an old page, while a transfer from the CISM's random port to the serial port is in progress or refresh, etc.

The ICAP memory interface also contains some I/O ports such as the ICAP count register (ICR), and the ICAP page register (IPR). The ICR allows the ACU to use global response count circuitry to accumulate the total count sent from all the ICAPs instead of those counts from CAAPP PEs. IPR contains the page number that specifies which page of the CISM is currently accessible by ICAP through the zero-wait-state page-mode operation. If the ICAP wants to access a new page in CISM, then one wait state has to be inserted to load a new row address into VRAM. This can be accomplished by an ICAP access to the IPR port.

Except for the ISSM, we have tested every ICAP memory interface mentioned above. We found out that our commercial SRAM does not meet the manufacturers specification. When chip select (CS) is disabled, the output enable (OE) signal should not affect the system data bus. However, we found out some patterns of data from CISM are not correct due to the influence of the SRAM with its CS disabled. If we forced the OE of the SRAM to be disabled, the problem went away. More extensive tests are required to guarantee that the four different physical devices sharing the same ICAP system bus do not affect each other inappropriately. All the other interface tests were successful. For example, we were able to access new pages by writing to the IPR port, loading one pattern into the ICR port, transferring to an eight bit shift register, and successfully shifting out the response count bit serially to the future global count circuitry for accumulation.

7.2.3 Backing Store Controller

The video RAM controller arbitrates the requests from various masters and generates the appropriate VRAM control signals such as row address strobe (RAS), column address strobe (CAS), data transmission (DT), read or write enable. It also inserts ICAP wait states by driving the "READY" line low whenever it requires use of the ICAP address bus to control the VRAM.

The transfer controller also contains two 64-bit 4-way time- multiplexed buffers. For example, in the case of the backing store read operation, while one buffer is receiving the next four 16-bit words (i.e. 8 bytes) via 16 CAAPP pins from the Serial Port, another buffer is using a cycle stealing technique to download the current 8 bytes to a group of 8 CAAPP PEs via an 8-bit data path for each involved PE. Then the two buffers will switch roles for receiving and downloading.

Before we tested the backing store controller, we used the ICAP memory interface to load 64 bytes of CAAPP PE coordinates into the random access port of the CISM, and verified it by reading it out from the same port. After issuing the backing store read instruction, we successfully observed the correct data coming out from the serial port of the CISM after the strobing of each serial clock (SC). This demonstrates that the page number, starting column address within the serial port, and VRAM control signals are all correctly generated by the backing store controller. It also completed the operation as expected.

To verify whether the data has been stored in page zero of the CAAPP PEs, the backing store write instruction was issued. The data was also verified by reading the random access port of the CISM. Furthermore, it demonstrated that when data is moved between the CAAPP and the CISM it goes through an automatic corner turning mechanism that provides bit-serial data access to the CAAPP and byte-parallel access to the ICAP. From the backing store controller's point of view, it was a complete success. However, we had to increase the delay between the negative going edge of the phi 2 clock to the positive going edge of the phi 1 clock from 20 ns to 50 ns (the spec is 5ns). The operations were verified for bytes 0, 1, 14 and 15 of page 0 of every CAAPP PE. We did have a problem getting out the first 8 bytes of the 64 bytes of data, but think for various reasons that this is a timing issue. One other problem we had is that the data in the CAAPP was destroyed after we repeatedly issued the backing store write instruction. We re-ran simulations in the APOLLO, and found that we can write as many times as we wish and the data is still retained in the CAAPP. Further debugging is required. Since the CAAPP PE has dynamic memory, which is very sensitive to clock timing, the final system clock speed depends heavily on the foundry's processing parameters. The VRAM is also very sensitive to the clock timing during serial write operations; sometimes the first data was written into the location specified by the starting column address (SCA), other times it was written into SCA + 1. Obviously, more effort is required to maintain the data integrity.

The refresh circuits did cooperate with the VRAM controller to generate the appropriate VRAM refresh signals. When the refresh signal from the ACU is high, it will repeat the VRAM's CAS before RAS refresh cycle to refresh all the memory cells.

7.2.4 Two CAAPP Custom Chip Breadboard

The breadboard, shown in Figure 22, consists of a socket card accommodating two neighboring CAAPP custom chips, a vertical Video RAM (VRAM) card, and a VME chassis containing VME interface and VRAM controllers for HCSM and ISSM. The serial port side of the VRAM card is connected to the two CAAPP chips via sharing the mesh network; the random access port side is connected to the SUN workstation and appears in the VME space so that a small image can be loaded from disk onto the CAAPP to perform an associative memory demonstration. The interface between the IMS tester and the SUN host is via control registers located in the vertical VRAM card.

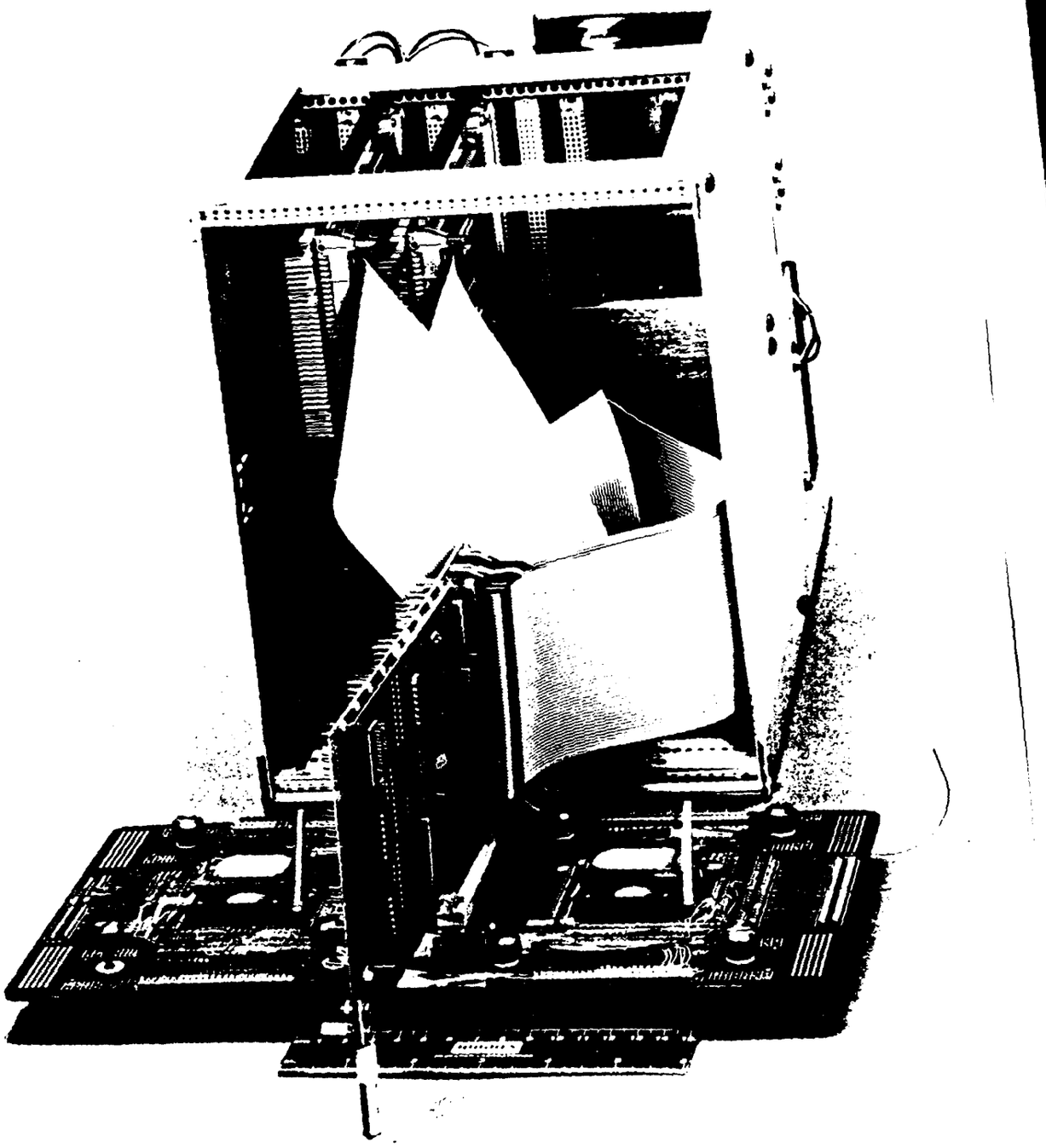

Figure 22: Associative Memory Breadboard

In the first step of two-chip breadboarding, we verified that two CAAPP chips can talk to each other. Chips are arranged in north- south fashion. We can send arbitrary byte patterns from the south side of the south chip to appear on the south side of the north chip after 8 clock cycles (5MHz) of shifting from south to north, and after another 8 clock cycles the same data began to appear on the north side of the north chip. The reverse direction from north to south has also been tested. To test east-west connection, we wired the east side mesh net of the south chip to the west side of the north chip. Similarly, for west to east shifting, we sent patterns from the west side of the south chip and they appeared on the west side of the north chip after 8 clock cycles, and after another 8 clock cycles the same data began to appear on the east side of the north chip. Our future tests will involve transferring data between the SUN workstation via HCSM and two CAAPP chips from their south sides to verify that we can time-share the mesh network between neighborhood communication and the distributed I/O which supports the associative memory.

8. Changes in Project Schedule and Goals

The hardware side of our project schedule has now slipped by almost a year. Much of this can be attributed to delays in fabrication, or faulty fabrication runs of our custom VLSI chips. Our first fabrication runs (both at UMass and Hughes) ran into a problem of excessive metal spreading with one foundry. This resulted in roughly a three month delay. One of our recent runs (again, both at UMass and Hughes) ran into other vendor problems that, at the time of this report have resulted in a five-month delay with at least one more month of delay expected. However, we must also accept some blame for having been overly optimistic in our initial schedule estimates.

In contrast, our software efforts have kept pace with the original schedule with the exceptions that the ICAP kernal is not as fully developed as we would like, and we have not been able to transport the environment to the actual hardware (since the latter does not yet exist). Instead of these activities, we have undertaken the development of sequential and parallel solutions to the Integrated Image Understanding Benchmark. The parallel solutions have been written for both the IUA and a Sequent Symmetry multiprocessor. The former has resulted in three types of contribution to our primary activity. First, it has lead to the development of many IU algorithms and supporting library routines for the IUA; second, it has helped us to more fully test and debug our development environment and has also resulted in many valuable enhancements to our software tools; third, it has given us more insight into the strengths and weaknesses of the IUA design, applied to a larger IU task, which will result in several small but significant changes to be implemented in the next version of the architecture.

It should also be noted that the goals of our project have changed somewhat from our original proposal. These changes are reflected in a modification to our contract that was approved during this year. The change basically provides for UMass and Hughes to perform sufficient additional integration work to turn the IUA prototype into a reliable, usable

system instead of being simply a proof-of-concept demonstrator. The change requires several more months of development effort on the part of Hughes and additional software integration work at UMass. These efforts will take place after the end of the initial contract period.

9. Conclusions

Two out of three of our custom chips have been verified to be fully functional and major portions of the third chip were also shown to operate correctly. Extensive simulations of the third chip, including connections to its associated componentry, have given us confidence that the newest version will work if only it can be fabricated properly. All of the circuit boards for the prototype have been designed, and the physical assembly and I/O subsystems have been fully specified.

Most aspects of the software development effort are on schedule, and we have managed to go beyond the original project goals in several areas. In particular, our simulator environment has become a very polished and integrated development tool that has greatly facilitated the creation of many vision algorithms and support routines. The simulator now mimics the entire IUA prototype, and a full-scale IUA version of the simulator is running in parallel on a Sequent Symmetry multiprocessor (with some limitations due to insufficient memory on the Sequent). We have also implemented the complete DARPA Integrated Image Understanding Benchmark on the IUA simulator (in addition to our versions for the Sun and Sequent). The IUA version of the benchmark demonstrated that the machine will be capable of performing an integrated vision task at speeds that approach frame rate and are certainly within the realm of real-time vision.

The IUA hardware construction effort has been substantially delayed by VLSI fabrication problems with the vendors under contract to MOSIS. Having to wait over six months for fabrication of our two largest custom chips is the major factor in the construction delay. We have worked around the fabrication delays as much as possible, through extensive simulation and rescheduling of certain tasks, but it has nonetheless been necessary to push back the prototype delivery date. Other factors contributing to the delay are the difficulty in obtaining video-RAM parts, and the change in the goals of the project to make the prototype more robust.

In anticipation of delivery of the working prototype, we have submitted a continuation proposal to fund the use of, experimentation with, and analysis of the IUA prototype. The goal of this second phase effort is the development of the specification and design for the next generation of the IUA. We expect this next generation design to be an order of magnitude more powerful than the current design, and to be substantially more sophisticated at the intermediate and high levels.

10. References

[Arvind, 1983] Arvind, D.K., I.N. Robinson, and I.N. Parker, "A VLSI chip for real-time image processing," *Proc. IEEE Inst. Symp, Circuits Syst.*, pp. 405-408, 1983.

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

[Benes 62] Benes, V.E., "On Rearrangeable Three-Stage Connecting Networks," *Bell Systems Tech. Journal*, vol. XLI, no. 5, pp. 1481-1492, September 1962.

[Beveridge, 1989] Beveridge, J.R., Griffith, J., Kohler, R.R., Hanson, A.R., Riseman, E.M., "Segmenting Images Using Localized Histograms and Region Merging," COINS Technical Report 87-88, University of Massachusetts.

[Burns, 1986] Burns, J.B., Hanson, A.R., Riseman, E.M., "Extracting Straight Lines," *IEEE Trans. PAMI*,Vol. 8, pp. 425-455, 1986.

[Clos 53] Clos, C., "A Study of Non-Blocking Switching Networks," *Bell Systems Tech. Journal*, vol. 32, no. 2, pp. 406-424, March 1953.

[Davis, 1984] Davis, R., Thomas, D., "Geometric Arithmetic Parallel Processor-Systolic Array Chip Meets the Demands of Heavy-Duty Processing," *Electronic Design*, pp. 207-218, October 31, 1984.

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

[Draper, 1988] Draper, B.A., Collins, R.T., Brolio, J., Griffith, J., Hanson, and Riseman, E.M., "The scheme system", COINS Technical Report 88-76, University of Massachusetts.

[Duff, 1978] Duff, M.J.B., "Review of the CLIP Image Processing System," *Proc. Nat. Comput. Conf.*, AFIPS, pp. 1055-1060, 1978.

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

[Foster, 1976] Foster, C.C., "*Content Addressable Parallel Processors*," Van Nostrand Rein-

hold, NY, 1976.

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

[Hanson, 1987] Hanson, A.R., Riseman, E.M., "From Image Measurements to Object Hypotheses," COINS Tech. Rept. 87-129, University of Massachusetts at Amherst, December 1987.

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

[Hunt, 1981] Hunt, D.J., "The ICL DAP and Its Application to Image Processing," In *Languages and Architectures for Image Processors* (M.J.B. Duff, S. Levialdi eds.), Academic Press, London, 1981.

[Kumar, 1985] Kumar, V.K.P and C.S. Raghavendra, "Array processor with multiple broadcasting," *Proc. 12th Ann. Inst. Symp. Comput. Architecture*, June 1985.

[Lehrer, 1987] Lehrer, N.B., G. Reynolds, and J. Griffith, "A Method for Initial Hypothesis Formation in Image Understanding," *Proc. Int. Conf. Computer Vision*, Computer Society Press, London, England. pp. 578-585, June 1987.

[Levitan, 1984] Levitan, S.P., "Parallel algorithms and architectures: a programmers perspective." PhD dissertation, Computer and Information Science Department; also COINS Tech. Rept. 84-11, University of Massachusetts at Amherst, May 1984.

[Li, 1987] Li, H., and M. Moresca, "Polymorphic Torus Network," *Proc. Intl. Conf. Parallel Processing*, Pennsylvania State University Press: State College, Pennsylvania, 1987.

[Miller, 1987] Miller, R., and V.K.P. Kumar, Dionisios Reisis, and Q.F. Stout, "Parallel Computations on Reconfigurable Meshes," USC Tech. Rept. IRIS #229, University of Southern California, Los Angeles, CA, March 1987.

[Nii, 1986] Nii, H.P., "The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures," *AI Magazine*, vol 7, no. 2, pp.38-53, 1986.

[Rana 88] Rana, D., Weems, C.C., and Levitan, S.P., "An Easily Reconfigurable Circuit Switched Connection Network", *Proc 1988 IEEE Int Symp on Circuits and Syst*, pp. 247-250, June 1988.

[Random 1987] *The Random House Unabridged Dictionary of the English Language* 2nd ed., S.B. Flexner, and L.C. Hauck (eds), Random House, N.Y., 1967.

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

11. Appendix: Test Reports for Custom VLSI Chip Fabrication Efforts at the University of Massachusetts

A.1 Test report for the feedback concentrator chip

P-Name: CONCEN
Fab-ID: M7ACKB01
Source: UMass

This chip implements a feedback data concentrator.

A total of 24 parts were received. They were all in good physical condition.

Ten of the parts were tested, and all were found to be fully functional. Because we require only five of these parts to build the prototype IUA, we did not bother to test more than ten of the parts. The remaining parts have been placed in storage, and will be tested if they are ever needed. Figure 23 is a microphotograph of the fabricated chip.

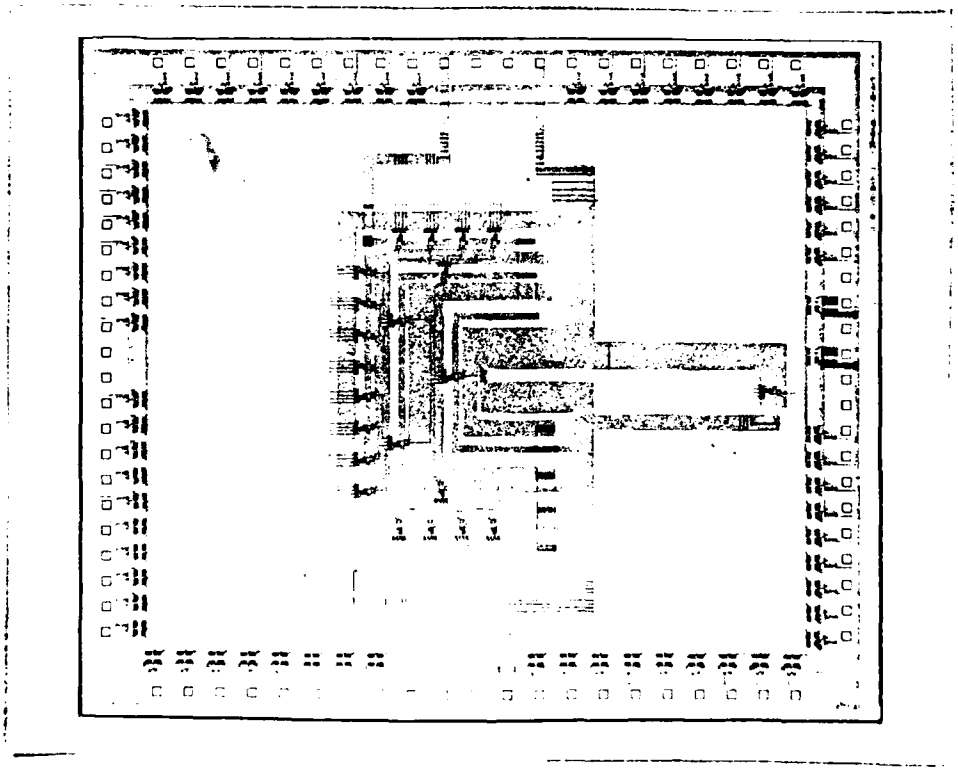

Figure 23: Feedback Concentrator Chip

A.2 Test report for the ICAP router test-structures chip

P-Name: Test_Chip
Fab-ID: M79XCE01
Source: UMass

This chip implements the various leaf-cells and assemblies that make up the ICAP communication link chip, with a large number of test points brought out to external pins to enable more extensive testing, analysis, and debugging.

A total of 30 packaged parts were received. They were all in good physical condition.

We tested 19 of the parts and found that seven were non-functional. The remaining twelve required high drive currents, and exhibited problems in the memory cell. The high drive current problem was traced to the use of a scaled pad design that was not directly scalable. The memory problem was traced to a design error in the memory precharge timing. Figure 24 shows the ICAP router test-strcutures chip.

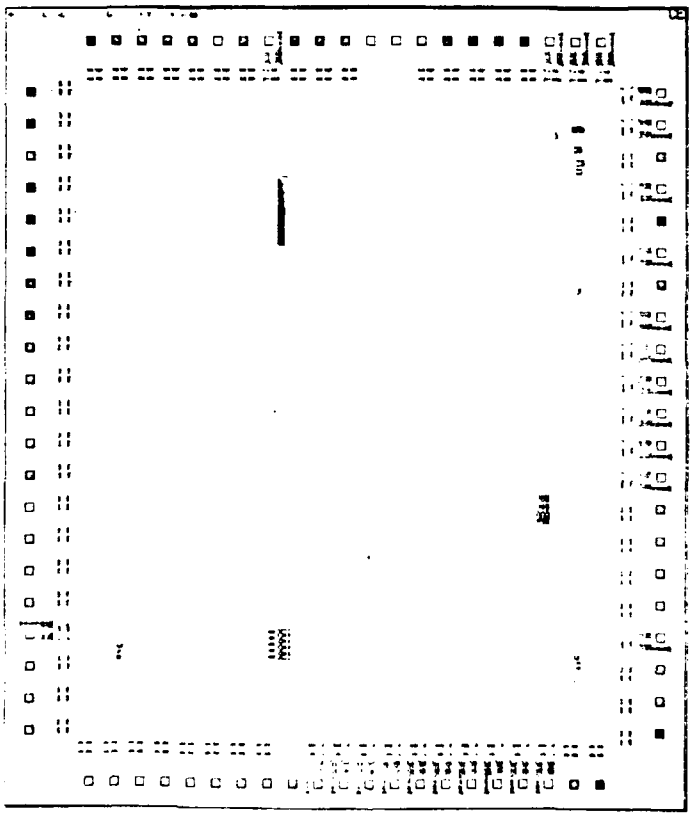

Figure 24: ICAP Router Test Structures Chip

A.3 Test report for the ICAP router chip

P-NAME: ROUTER
ID: 23665
Fab-ID(1): M77PCE1 (VTI) $\lambda = 1.0\mu$ (SCN)
Fab-ID(2): M77NCJ1 (UTMC) $\lambda = 1.0\mu$ (SCP)
Source: UMass

OVERVIEW:

This chip was assigned to two different runs by MOSIS. The circuit implements a 32-input 32-output crossbar switch with a control memory.

The crossbar switch portion of the chip is comprised of 32 MUX bits, each with 32 inputs and 1 output. The inputs of all the MUX bits are bussed together, which make the 32 inputs to the crossbar switch in the chip. The 32 outputs from the MUX bits form the outputs from the chip.

Each of the 32 MUX bits is designed as a tree structure in which inputs are connected to the leaves and the output is taken from the root. Path selection at any level of the tree can be done with a single bit of the control word from the control memory. We need 5 control bits to select any of the 32 inputs to the output in a MUX. Therefore $32 \times 5 = 160$ bits, which we call a control word, are required to set up a 32-input to 32-output connection pattern in the chip.

The control memory consists of 32 control words. Each of the 32 5-bit bytes in any of the 32 control words can be written into, first by writing a control word number in a row-select register, which is mapped to address 32, then writing in appropriate sequence of 5-bit bytes from address 0 through 31. The design incorporates fully static RAM and registers, therefore, no refresh is required and the timing is very flexible. The only requirement for writing is that the data and address should be stable before WRbar line, which is brought low for at least 100nS.

Our chip design is such that the control word number stored in the row-select register, selects the control word that determines the input-output connection pattern of the chip.

In a previous report for these chips, we reported suspected bridging faults between Metal2 - Metal2. Further investigation of the problem led to the discovery of a fault in our test system.

UTMC RUN:

A total of 24 parts were received. They were in good physical condition. Bonding of the parts was good.

About 15 parts were tested and none of them worked. We cannot give much feedback about this run because all of the outputs in all of the chips were logically stuck-at-1.

By visual inspection with a stereo microscope, we found that the geometries on the die were much better defined than the VTI run, but still metal wires separated by minimum distance appeared dangerously close in many places.

In consultation with MOSIS, these problems have been traced to the fact that VTI and UTMC do not planarize their wafers, which leads to excessive spreading in the Metal2 layer, which might have resulted in shorts between Metal2-Metal2 paths.

VTI RUN:

A total of 24 parts were received. They all were in good physical condition. Bonding was acceptable in most of the cases, but in some cases, the solder bonds on the dies were dangerously close to the outer power supply ring of the standard frame.

About 20 parts were tested and none of them worked. There were logical bridging faults between the outputs, which were at different places in different chips.

We found that the problem we initially detected with the VTI run was not due to the bridging faults reported by MOSIS. In retesting the chips, we discovered that an unusually high current was required to pull the input pads high ($Ii > 1.5mA$ for $Vi = 4.0V$). Since the drivers of our test system were not capable of supplying this high current, proper logic levels were not input to the chips and consequently our earlier testing showed a behavior in which various outputs tracked each other in a manner consistent with what we would expect if there were metal shorts in our multiplexer array.

NOTE 1:

There was a design fault in the address buffers in the chip such that address cannot be changed. This, however, still allowed us to test the chip to some degree. When power is turned on, a unique control word number from 0 through 31 appears in the row-select register. Of course we cannot write in the RAM, but an arbitrary connection pattern also appears in the control word selected by the row-select register. Therefore we will have an arbitrary, conflict free connection pattern set up in the chip at power on. This lets us check the functionality of the MUX and any bridging faults.

NOTE 2:

Some mistakes were done in the layout of the MUX bit because of that there were shorts at four places in the MUX bit. The n-type transistors in the MUX bit are twice as strong as the p-type transistors, therefore, the effect of these shorts on the output is different for shorts in the n-type structure than the p-type structure.

One of the shorts on the n-side, shorts input 9 to the output from its ancestor transistor. The result is that if any of the inputs 8 through 11 is selected, the connecting path has a bridging fault with input 9. Since input 9 path is stronger, it gets selected to the output but the correct path interferes with it. Notice that the correct path is further made stronger by fault free p-side structure. This interference cannot be represented as any boolean function

as it varies for different shorts. Similar observation holds for input 24 through 27 on the n-side.

On the p-side, however, for the bridging shorts, the fault free n-side structure is strong enough most of the time to select the correct input to the output. In some cases, however, similar interference is observed. Figure 25 shows the ICAP router chip.

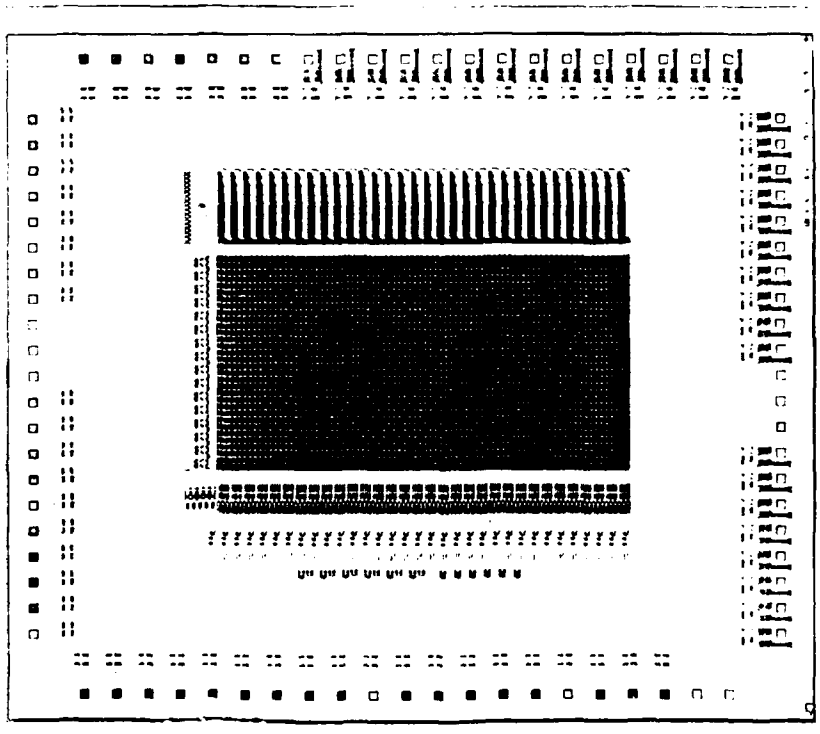

Figure 25: ICAP Router Chip

A.4 Test report for the second ICAP router chip

P-Name: TOP_CONNECTOR
Fab-ID: M84KPB1
M88KPB1
Source: UMass

This chip provides a 32-input, 32-output, bit-serial crossbar switch, with a connection pattern cache that can store up to 32 I/O connection patterns. In the IUA prototype, eight of these chips are combined to construct a 64×64 connection network that links the 64 ICAP processors to each other.

A total of 6 parts were received from the M84KPB1 run, and 56 parts were received from the M88KPB1 run, with 26 from one vendor and 30 from another. All of the parts were in good physical condition.

All six of the M84KPB1 chips were tested, and three were found to be fully functional. The remainder had random problems that were not repeated on any other devices, and are probably due to individual defects in fabrication. Only two of the thirty parts from one of the M88KPB1 vendors were found to be fully functional. The remainder had a random pattern of stuck-at faults and other problems. There were eleven fully functional parts, out of 26, from the other M88KPB1 vendor, with random faults in the non-functional parts. All of the M88KPB1 parts were significantly slower than the M84KPB1 devices. Figure 26 shows the second ICAP router chip.

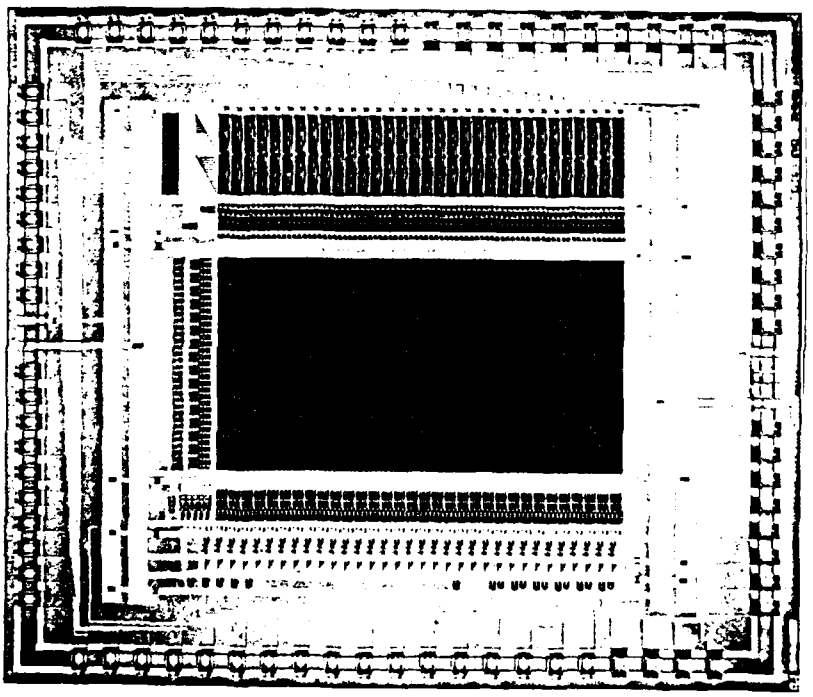

Figure 26: Second ICAP Router Chip

A.4 Test report for the second ICAP router test-structures chip

P-Name: TEST_CHIP
Fab ID: M84KMB1
M88KMB1
Source: UMass

The purpose of this chip is to facilitate testing of the individual leaf cells and assemblies that make up the ICAP Router chip. It provides direct access to various test points that are not available in the complete router chip, and thus allows more extensive timing analysis, fault diagnosis, and debugging of structures in the router chips.

A total of 48 parts were received from the two runs. They were all in good physical condition.

Only two of the chips were tested for timing analysis purposes because the actual router chips had a high rate of functionality. The two tested parts were fully functional and helped to verify that the components from the M88KPB1/M88KMB1 vendors were slower than those from the M84KPB1/M84KMB1 vendor. Figure 27 shows the second ICAP router test structures chip.

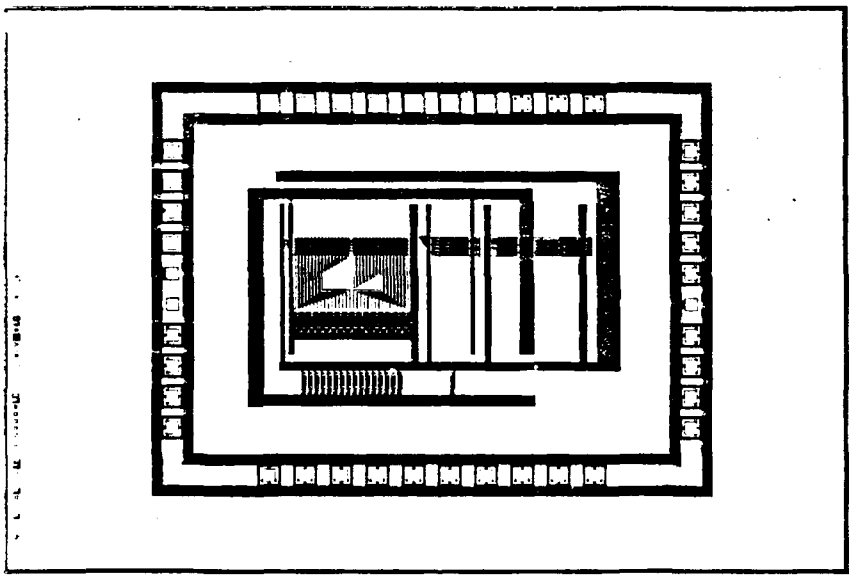

Figure 27: Second ICAP Router Test Chip