

AD-A238 046



IC
CTE

JUL 10 1991

S

C

D

RL-TR-91-76, Vol I (of four)
Final Technical Report
June 1991



2

EXPERT SYSTEMS ON MULTIPROCESSOR ARCHITECTURES Summary

Stanford University

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. 5291

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

91-04335



The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

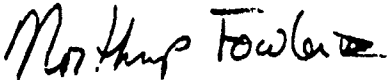
Rome Laboratory
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

91 7 8 026

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-91-76, Volume I (of four) has been reviewed and is approved for publication.

APPROVED:



NORTHROP FOWLER III
Project Engineer

APPROVED:



RAYMOND P. URTZ, JR.
Technical Director
Directorate of Command & Control

FOR THE COMMANDER:



RONALD RAPOSO
Directorate of Plans & Programs

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL(COE) Griffiss AFB, NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

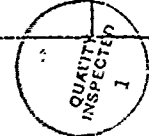
EXPERT SYSTEMS ON MULTIPROCESSOR ARCHITECTURES,
Summary

Edward A. Feigenbaum
Robert Engelmores
H. Penny Nii
James P. Rice

Contractor: Stanford University
Contract Number: F30602-85-C-0012
Effective Date of Contract: 14 March 1985
Contract Expiration Date: 31 March 1990
Short Title of Work: Concurrent Expert Systems
Architecture
Program Code Number: OE20
Period of Work Covered: Mar 85 - Mar 90
Principal Investigator: Edward A. Feigenbaum
Phone: (415) 723-4878
RL Project Engineer: Northrup Fowler III
Phone: (315) 330-7794

Accession For	
DTIC GRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special

A-1



Approved for public release; distribution unlimited.

This research was supported by the Defense Advanced Research Projects Agency of the Department of Defense and was monitored by Northrup Fowler III, RL (COE), Griffiss AFB NY 13441-5700 under Contract F30602-85-C-0012.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.

1. AGENCY USE ONLY (Leave Blank)		2. REPORT DATE June 1991		3. REPORT TYPE AND DATES COVERED Final Mar 85 - Oct 90	
4. TITLE AND SUBTITLE EXPERT SYSTEMS ON MULTIPROCESSOR ARCHITECTURES, Summary				5. FUNDING NUMBERS C - F30602-85-C-0012 PE - 62301E PR - E291 TA - 00 WU - 01	
6. AUTHOR(S) Edward A. Feigenbaum, Robert Engelmores, H. Penny Nii and James P. Rice					
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Knowledge Systems Laboratory Stanford University 701 Welch Rd, Bldg C Palo Alto CA 94304				8. PERFORMING ORGANIZATION REPORT NUMBER N/A	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Defense Advanced Research Projects Agency 1400 Wilson Blvd Arlington VA 22209-2308				10. SPONSORING/MONITORING AGENCY REPORT NUMBER RL-TR-91-76, Vol I (of four)	
11. SUPPLEMENTARY NOTES Rome Laboratory Project Engineer: Northrup Fowler III/COE/(315)330-7794					
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited.				12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) This final report documents the results of a five-year investigation of methods for achieving higher performance for knowledge-based systems through the design of innovative software and hardware systems architectures. Volume I summarizes the work performed and lessons learned, and serves as an annotated index to the set of over 50 project technical reports. Volumes II through IV contain the project technical reports. NOTE: Rome Laboratory/RL (formerly Rome Air Development Center/RADC)					
14. SUBJECT TERMS Multiprocessor Architectures, Artificial Intelligence, Blackboard Systems				15. NUMBER OF PAGES 60	
				16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL		

1.....	Introduction	1-1
1.1.....	Project Goals	1-1
1.2.....	Personnel	1-3
2.....	Hardware-Level Systems Studies	1-3
2.1.....	Simple and Helios	1-4
2.2.....	CARE	1-5
3.....	Operating Systems and Languages	1-8
3.1.....	CAREL	1-8
3.2.....	CAOS	1-9
3.3.....	LAMINA	1-9
3.4.....	Inter-Processor and Inter-Process Communication	1-10
3.5.....	Load-Balancing	1-10
3.6.....	Concurrent and High Performance Lisp	1-11
3.7.....	Distributed Cache Coherence	1-11
4.....	Problem-Solving Frameworks	1-12
4.1.....	Cage	1-12
4.2.....	Poligon	1-14
5.....	Applications	1-17
5.1.....	Elint	1-17
5.2.....	AirTrac	1-18
5.3.....	ParAble	1-19
5.4.....	Numerical and Semi-numerical programs	1-20
6.....	Conclusions, Observations Results	1-20
6.1.....	Speedup over serial computation	1-21
6.2.....	Pipelines	1-22
6.3.....	Basic computational metaphor	1-22
6.4.....	Communication	1-22
6.5.....	Problem Solving Methods	1-23
6.6.....	Development strategy	1-24
6.7.....	Analysis of the application	1-24
6.8.....	Load balancing	1-24
7.....	Bibliography of Expert Systems on Multiprocessor Architectures: Project	
.....	Publications	1-25
8.....	Bibliography of Referenced Work Not Performed on the Project	1-30

Abstract

This final report documents the results of a five-year investigation of methods for achieving higher performance for knowledge-based systems through the design of innovative software and hardware systems architectures. Volume 1 summarizes the work performed and lessons learned, and serves as an annotated index to the set of over 50 project technical reports. Volumes 2 through 4 contain the project technical reports.

1. Introduction

The Expert Systems on Multiprocessor Architectures (ESMA) project was initiated in March 1985, and technical work was completed in 1990. The research was conducted at Stanford University's Knowledge Systems Laboratory. The results and findings of the project were published in a series of technical reports, which comprise Volume 2 of this Final Report. Volume 1 sets forth the basic concepts that underlie the research, and provides a road map to guide the reader through that technical literature. Volume 1 ends with a project bibliography, which serves as a table of contents for Volumes 2 through 4. ESMA builds upon and straddles a number of areas of research in computer science, including artificial intelligence, programming languages, operating systems, communication protocols and hardware. Prior to this project, some work was done on analyzing the performance of rule-based systems on parallel architectures, most notably by Gupta [Gupta 86]. On the hardware side, there are commercially available machines that are similar in some respects to the architectures considered here, notably the Ametek machine. The relationship of ESMA to work in other fields is documented copiously in the papers cited below.

On the other hand, this investigation is unique: the project focussed on applications characterized by symbolic (largely non-numerical) computation; took an end-to-end multi-level approach toward identifying and exploiting concurrency; and used highly instrumented simulation to permit careful analysis of experimental results.

In the remainder of this chapter we set forth the goals of the project and list the personnel who contributed toward achieving those goals. Chapters 2 through 5 describe and summarize each of the four levels of analysis in our multi-level, vertical-slice strategy. Chapter 6 draws together the principal conclusions and lessons that were learned from this research. Chapter 7 is a full bibliography of the technical reports that were produced by project staff. Chapter 8 lists other referenced works.

1.1. Project Goals

The project's primary goal is to find ways to increase the performance of expert systems through the use of the new, emergent, parallel hardware designs.

The number of possible implementation strategies for such a project is huge. One has only to look at the large number of different hardware designs that are emerging and at the number of different problem-solving methods to see how combinatorial the problem would be if we endeavored to investigate all of the reasonable and plausible combinations of architectures. It was decided, therefore, that we could learn a great deal simply from making a commitment to one, or at least a small number of different options at each point in the system's make-up. We thus decided to take a "vertical slice" through the space of possible solutions. Clearly we did not intend to investigate any options that seemed non-useful, so we knew from the outset that, although we could not prove that we had the best design to

meet our goals, our design would nevertheless be at least a plausible architecture for a future computational environment.

We viewed the task of implementing concurrent expert systems as being one which was split into a number of implementation layers. If we could achieve speed-up at each one of these layers, then we could hope for a substantial overall performance improvement compared to existing AI systems. Our model of the layers into which the project could be split is shown in Figure 1.

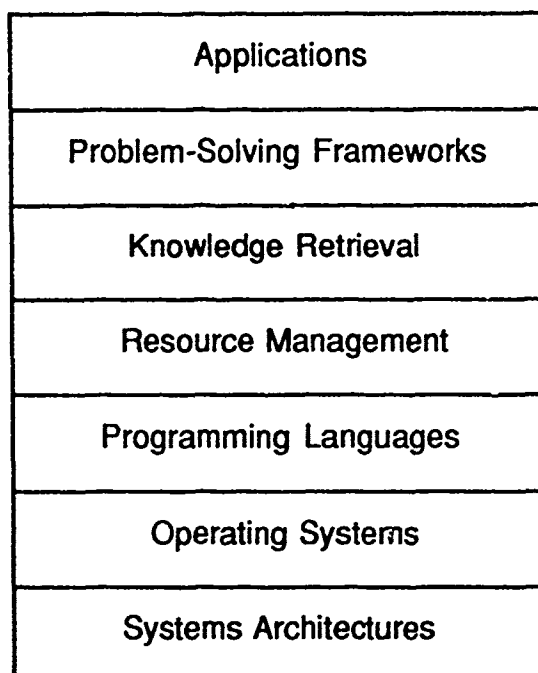


Figure 1. *The layers of system implementation through which we hoped to achieve computational speed-up in the project.*

It was originally anticipated that the needs of the applications would drive the development of the problem-solving frameworks and so on down through the implementation hierarchy shown in Figure 1 until eventually the hardware would be designed under the constraints passed down from above. In practice, however, this did not happen. Because of the difficulty of finding and mounting an application suitable to our needs and the early availability of personnel interested in the hardware design aspect, the hardware design went ahead more rapidly than the other layers. This resulted in our designs being more hardware driven than application driven. This approach has its advantages, for example, an entirely top-down design process could easily have resulted in low-level system requirements which were not implementable.

As well as the thrust of the project coming from the bottom rather than the top, the levels of abstraction actually implemented differed significantly from those shown in Figure 1. Figure 2 gives a more realistic representation of the layers that were actually investigated, as opposed to what we intended to do.

Applications
Problem-Solving Frameworks
Resource Management
Programming Languages
Hardware Systems Architectures, Topologies and Protocols

Figure 2. The layers that were actually implemented in the project. Resource Management is shown in small type because it was a recent addition and most of our work was done without the help of this layer.

The Knowledge Systems Laboratory has considerably more expertise in software than in hardware. We thus decided early on not to build any hardware - there are many other research groups that could do this better than we. We decided, therefore, to simulate our hardware. This would allow us to modify our software and hardware designs easily and allow us to extract the maximum insight with the minimum effort.

The rest of this paper is split into sections which reflect the major layers shown in Figure 2. In each of these sections the work of the relevant sub-projects will be discussed. Because of the bottom-up thrust of the project the project's components will be discussed in a bottom-up order. This will also reduce the number of forward references made, since discussion of the higher layers will inevitably have to refer to the substrates on which they are implemented.

1.2. Personnel

This project has employed a large number of people over the years and it seems appropriate to name them all here:

Ed Feigenbaum, Bob Engelmores, Penny Nii, Bruce Delagi, Harold Brown, Hiroshi Okuno, John Delaney, Byron Davies, Hirotoshi Maegawa, Nelleke Aiello, James Rice, Nakul Sarziya, Sayuri Nishimura, Eric Schoen, Greg Byrd, Max Hailperin, Russell Nakano, Masafumi Minami, Chris Rogers, Alan Noble, Jean-Christophe Bandini, Manu Thapar, Djuki Muliawan, Pandu Nayak, Jerry Yan and Sam Hahn.

2. Hardware-Level Systems Studies

As was mentioned above, hardware system design led the way in the project. In this section we discuss a little bit of the motivation for the hardware designs and briefly describe both the current generation of hardware designs on which we are working and the simulator we are using.

2.1. Simple and Helios

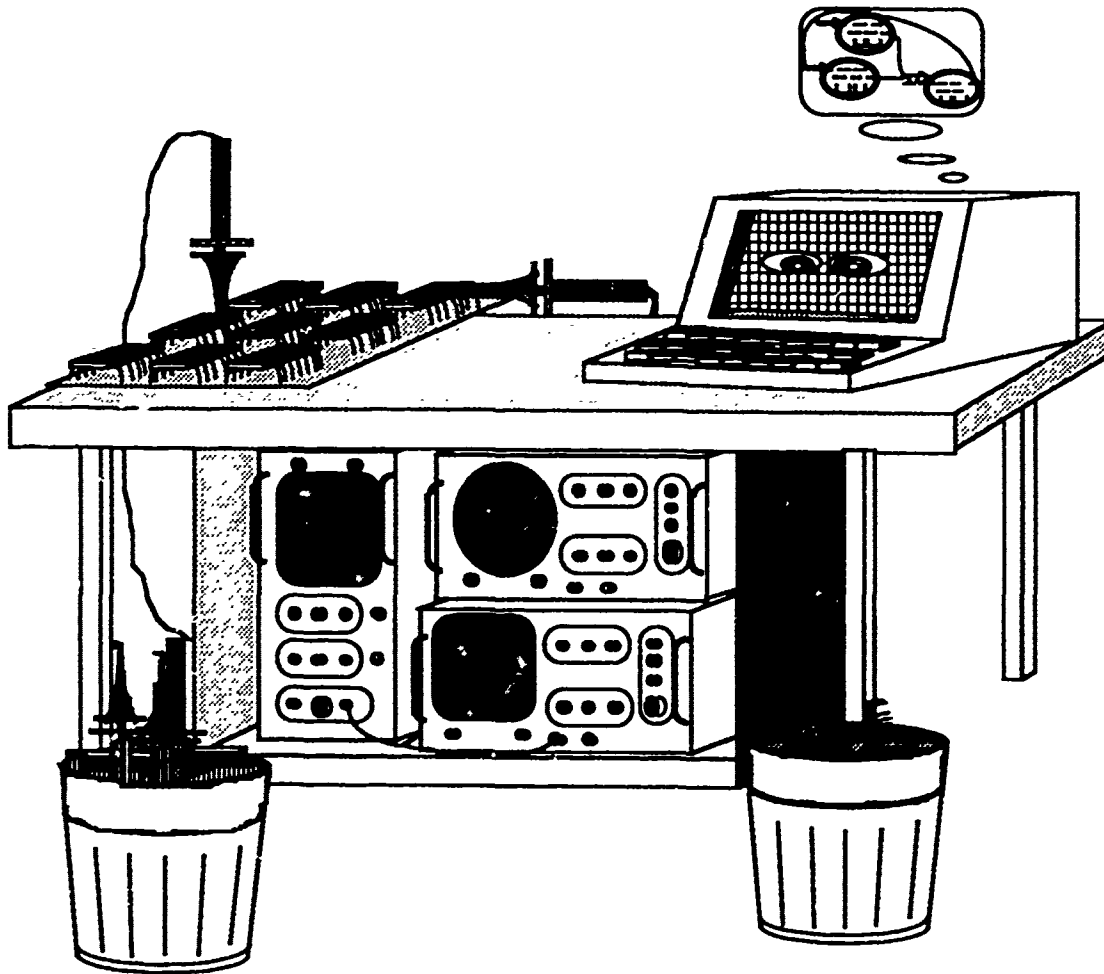


Figure 3. The Simple system provides a toolkit from which to build circuits to be simulated, a collection of probes to connect to the circuit and a set of instruments to connect to the probes.

The hub of all of the work done on the project has been the digital circuit simulator,¹ upon which everything else is built. This simulator is called *Simple*. It is an event-driven simulator, designed to allow the user to design and specialize digital circuits in a simple and modular way, using a circuit design tool called *Helios*. A sophisticated set of instrument tools allow the user to design and specialize simulated probes which can be connected to the circuit while it is running. This allows the connection of a number of instruments to the probes that permit the user to see the behavior of the circuit as it operates without interfering with the behavior of the system. We like to view this model as one of a laboratory workbench equipped with collections of instruments, probes and circuit building compo-

¹Note: This simulator could be used to simulate events down to the gate level, but one of its powerful attributes is its ability to allow the programmer to define the behavior of composite objects in terms of methods that make these devices appear to be atomic black boxes. This ability obviates the need to do gate level simulation of those aspects of the system whose behavior is well understood. This has enormous benefits in terms of simulation time.

nents from which the user can build systems and on which the user can perform quantitative experiments (see Figure 3).

The key factors that make the Simple simulator so powerful are detailed in [Delagi 86b, 2-272]¹, [Delagi 87, 2-294] and [Saraiya 90a, 4-360]. In effect, the simulator focuses on the critical design aspects of multiprocessor design, namely interprocessor communication and topology. The simulation is less detailed in other areas. This allows the user to simulate the execution of sophisticated problems, rather than the toy problems or small code fragments possible with other simulators. The instrumentation in the simulator is powerful and flexible, not only allowing the user to observe events in the simulated system at multiple levels of abstraction, but also readily allowing the user to modify and specialize instrumentation so as to focus the simulator more sharply on interesting application-specific behavior. This allows the user to gain substantial insight from simulator runs, while still allowing the user to reconfigure the system easily and quickly in the event of an unexpected result prompting unplanned experiments.

It was found early on that simulations of the sort we wanted to do would be computationally very expensive. An experiment was performed, therefore, to parallelize the simulator itself in an attempt to bring down the times taken for the simulations, which often exceeded one day in duration. This resulted in *AIDE*, a distributed version of Simple [Saraiya 86, 4-297]. Unfortunately, we were unable to achieve any speed-up at all for our simulations, largely because of the communication bandwidth and latency associated with communicating between the multiple Symbolics machines we were using via an Ethernet and because the simulator, being event-driven, required frequent synchronization on the event queue, which serialized the processing. Although this experiment yielded a negative result, it was valuable in demonstrating the importance of process grain size and synchronization effects.

2.2. CARE

The Simple simulator mentioned above was used to design and build what we refer to as the CARE² machine and simulation system [Delagi 88a, 2-301] (see Figure 4). The CARE machine is that simulated machine on which all of the experiments mentioned below have been performed. The machine's design has a few key features which are worthy of note:

- Dynamic cut-through routing with local flow control, in order to optimize network throughput [Byrd 87c, 2-155]. This protocol uses special packet terminators and selective buffering to avoid deadlock during multicasts.
- Toroidal topology. Topology can be motivated by high-level, application domain considerations, but it is also motivated by such low-level concerns as packaging and communication protocols. Cost models were developed to characterize several topologies and these topologies were tested under simulation. On balance, we believe that toroidally connected networks have the best overall cost/benefit tradeoff [Byrd 87b, 2-148].
- Non-blocking message sending, so as to encourage pipe-line processing.
- Communications network with alternative paths between points, so as to reduce communications problems due to busy communication paths.

¹ Citations for project reports point to the bibliography at the end of this volume and also to the page number where the report can be found in volumes 2 through 4.

² The expansion for this acronym seems to have been lost somewhere in the wash. We think that it has something to do with the words *Concurrent* and *Array*.

- A separate communications controller, in order to support operating system functions and to implement the non-blocking send functionality mentioned above. This communications controller is referred to as the "Operator". The processor in each processing element that executes user code is called the "Evaluator".

A simplified model of a CARE machine processing element (site) is shown in Figure 4.

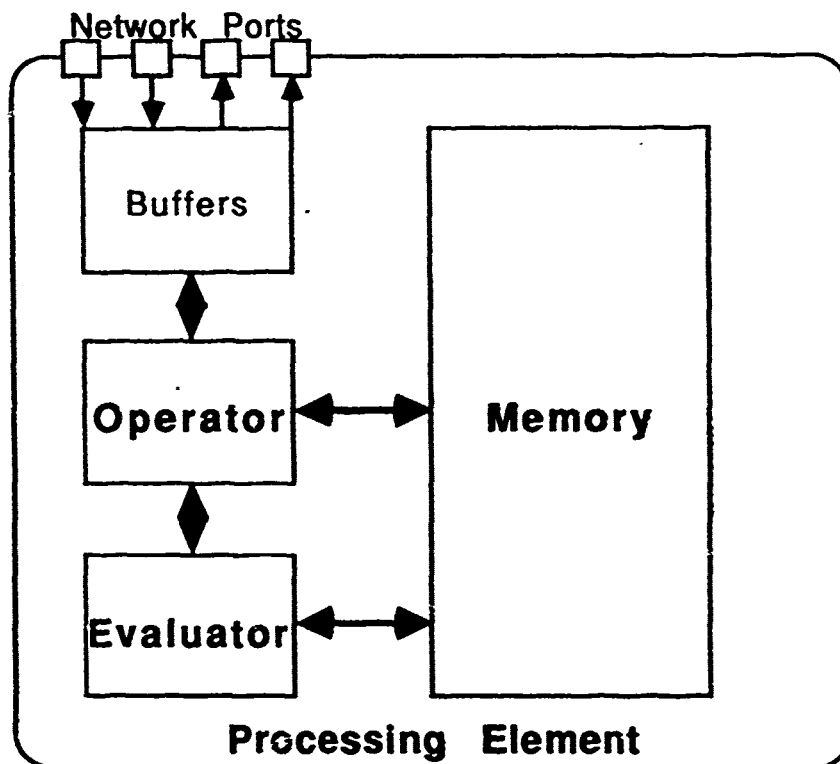


Figure 4. A CARE machine processing element (site).

These processing elements can be connected together in a number of ways, such as into grids and bus-based networks as it shown in Figure 5. When a CARE site is used simply as a memory controller its evaluator processor is not used. Similarly, when a site is used just as a processor in a bus-based shared-memory machine, only the evaluator is used.

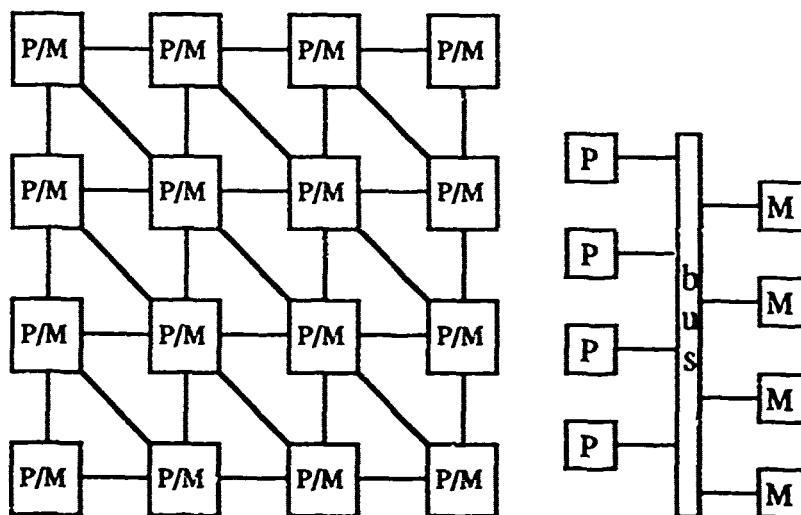


Figure 5. CARE sites can be connected together into a variety of distributed and shared memory of topologies. In this example we show a six-way connected grid and a bus based machine.

The work on the CARE sub-project has focussed mainly on the design of inter-processor communication networks, as is appropriate. This has meant that we have been able to ignore the instruction level behavior of the processors themselves. The application programs that we run are merely timed as they run between the points at which code fragments cause communication between processors. Being able to avoid doing register level simulation of the processors themselves has allowed us to execute much more complex and realistic programs on our simulated machines. We have therefore traded accuracy in our processor simulation - assuming that the processing elements will behave much like existing Lisp Machine processors - in favor of greater realism in terms of the system's performance under the load of real programs.

A number of aspects of system design have not been addressed in detail and the simulations do not take these into account. Most significant among these, perhaps, are the fact that memory usage, code distribution and garbage collection are not simulated, i.e. the CARE machine was assumed to have unbounded local memory and code was assumed to have been distributed uniformly to all processors at load time. Thus, although the CARE architecture was designed with garbage collection in mind, this was not simulated at all. In fact, all possible extraneous impediments to accurate and reproducible run-time measurement were eliminated. Such simulation machine system overheads as garbage collection, paging, I/O and page creation were carefully factored out of the timing. This resulted in timings that were not "realistic" in the sense that they did not account for certain necessary system behavior, but these timings were nevertheless far more useful in general because these system services are generally non-deterministic with respect to the simulation and their behavior is a function of the performance of the *simulation* machine, not the *simulated* machine.

The CARE/Simple simulator system is perhaps the most valuable tangible product of the project. It is now being used in a number of research departments, both corporate and academic, outside Stanford. Like all project software, it is in the public domain. CARE/Simple will soon be available running under Common Lisp, CLUE and X11 on a number of different platforms.

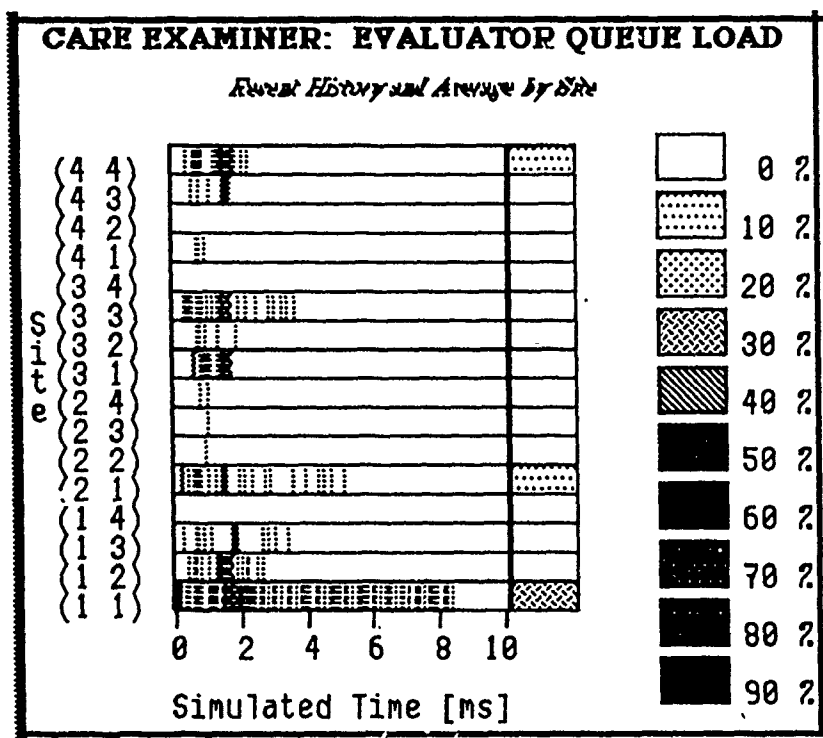


Figure 4. An example of instrumentation from the CARE system.

3. Operating Systems and Languages

A considerable amount of effort has been spent on the project in working at the operating system level of abstraction. Because our experiments dealt with a single task, and file system issues were not considered, it was not necessary to build an operating system *per se*. The CARE machine itself features a dual processor for each processing element. This allows much of the work of an operating system, particularly inter-processor communication, to be done by a dedicated processor in parallel with the execution of user code. The behavior of this communication processor is coded directly into the simulated hardware.

Amongst the work that has been done in this area has been work on concurrent object-oriented systems, concurrent Lisp dialects, programming models and resource allocation.

3.1. CAREL

CAREL [Davies 86, 2-226] was one of the first programs written to run on the CARE simulated machine. It was an early attempt to find a Lisp language interface to the distributed-memory hardware provided by CARE. It took as its basis Scheme [Abelson 83] and QLisp [Gabriel 84] and included primitives to allow remote function calls and remote consing. It was quickly found that, because of the cost of process creation, it was desirable to make the best use of any processes that were spawned. This efficiency was accomplished by storing application dependent data in non-ephemeral spawned processes. State of this type was implemented in CAREL as writable closure variables. These process closures could be used as elements in pipe-line computations or to represent mutable communicating program objects, for instance to represent real-world objects with state. *State, as encapsulated in communicating objects, and the idea of pipe-line parallelism have been pivotal in the design of the other systems developed on the project.*

The CAREL project was used mostly as a feasibility study and was soon discontinued.

3.2. CAOS

The first implementation of the Elint application, described further in Section 5.1, was made without the benefit of any problem-solving framework, *per se*, but rather using an object-oriented programming architecture. It was anticipated that the application could be easily mounted almost directly on the CARE machine and some experiments could be run quickly, which would allow us to learn some important lessons early in the project.

In order to mount the application, a distributed object-oriented system was implemented. This was done because the CARE system did not, at the time, come with its own "preferred" object system. The system that was implemented was called CAOS [Schoen 86, 4-433], a Concurrent Asynchronous Object-oriented System. It was implemented using the Flavors system supported by the Lisp machines used by the project. It had a number of key features:

- CAOS objects were dynamically instantiable and potentially multiprocess objects, though each would execute on a single processor, having at least one stack group associated with each CAOS object.
- CAOS objects were intentionally large grained. This was because it was anticipated that the communications network would be the resource most competed for, thus encouraging the programmer to perform a lot of computation in order to reduce the number or size of messages sent.
- Packet-based message-passing was used as the metaphor for communication between processes through streams in the language extensions to Lisp provided by CAOS.
- A large number of different message sending primitives were defined, including non-blocking sends that did not require a reply from the target of the message, sends that returned futures to the values returned by the targets and send operations which blocked immediately in order to wait for a reply from their targets.

Contrary to our intuition, the communications network proved to be the least loaded of the CARE machine's resources during our experiments on CAOS. The computational expense of supporting its complex object model caused the granularity of the resultant computations to be too large. However, the real-time signal interpretation application developed in CAOS focussed our attention on such key factors as decomposition grain size and the use of replicated pipelines of processes. Because of the computational expense for each process, the CAOS model was inconsistent with a large number of processors executing tightly coupled subproblems typical of reasoning systems.

3.3. LAMINA

Lamina is the object system that was designed after the lessons were learned from the CAOS experiments [Saraiya 90b, 4-394]. It was originally intended to provide a very small, light-weight layer on top of the CARE machine so that distributed object-oriented programs could be implemented efficiently. A significant part of the motivation for the design of Lamina was the desire to reduce the overhead suffered by the CAOS system in terms of associating large stack groups with each of the CAOS objects. Lamina introduced the idea of objects with restartable, rather than resumable code segments, which do not require stacks to preserve their state when they are not running. Since its first appearance Lamina has been developed extensively and, although still small and light-weight, provides a platform for the development of computational models for functional and shared-variable as well as object-oriented programming.

In [Delagi 86a, 2-243] not only are the three programming models - object-oriented, shared-variable and functional - shown all to be implementable using Lamina's unifying stream mechanism, but it also shows, by example, how these programming models can be used to create pipelines, how to manage these software pipelines and how structures can be dynamically created and relocated using the Lamina model. This model also allows the substantial localization of storage reclamation, which is a crucial factor in the development of efficient, concurrent garbage collection mechanisms.

Lamina has been used to implement a number of programs, both for direct implementations of the two real-time expert systems being investigated (see Section 5), AirTrac and Elint, and also a number of numerical programs. Lamina is now the preferred core programming system for the CARE machine and applications in Lamina have consistently shown the highest performance of all programs running on the CARE machine.

3.4. Inter-Processor and Inter-Process Communication

A considerable amount of work has been performed on the investigation of different mechanisms for inter-processor and inter-process communication. For distributed-memory machines we believe that the efficient distribution of work for large applications is crucially linked to the efficient implementation of multicast communication [Byrd 87a, 2-116]. In particular we have concentrated on the development of efficient cut-through routing methods that allow the effective use of multicast. In [Byrd 88b, 2-196] several alternative cut-through multicast protocols are described and compared experimentally. One particular adaptive scheme is found to be superior to the others investigated both in performance and in the fact that the protocol provides cut-through multicast without requiring dedicated storage in the communication architecture for a full packet.

Although the principal thrust of the project has been towards the development of distributed-memory hardware, the fact that the CARE simulator can also simulate shared-memory machines has allowed the investigation of the relative performance of these two distinct classes of machines and the relative performance and appropriateness of shared-variable and message-passing/object-oriented programming models. In [Byrd 88a, 2-181] a particular parallel application is implemented in both object-oriented and shared-variable styles. Using these examples it was possible to show how the differences in programming model affected performance and what the costs associated with each model were. This, the allowed the identification of strategies for minimizing data communication costs in each of these programming models.

Work late in the project focussed on the design of hardware that might provide efficient support for both the shared-variable and the message-passing programming models, particularly through the use of cut-through multicast protocols [Byrd 89, 2-205].

3.5. Load-Balancing

We examined load-balancing problems within the context of the "vertical slice." (recall Figure 2) [Hailperin 88, 3-1] In particular, this work is focussed on a load-balancing method which migrates Lamina objects in a large network (thousands of processing elements) of CARE processing elements in order to improve the performance of soft-real-time signal-interpretation systems such as Elint and AirTrac (see Section 5).

Experiments showed that without special attention to load balancing, performance was seriously degraded. Without load balancing, only a lightly-loaded multicomputer, which has cause to create processes dynamically, can in general achieve real-time performance. The studies focussed on how to achieve global load balancing, which would be an

attractive solution to this problem, as it would allow the effective use of massively-parallel ensemble architectures for larger soft-real-time problems.

The challenge is to replace quick global communication, which is impractical in a massively-parallel system, with statistical techniques. In this vein, a novel approach to decentralized load balancing was investigated based on statistical time-series analysis. Each processing element estimates the system-wide average load using information about past loads of individual sites and attempts to equal that average. This estimation process is practical because the soft-real-time systems in which we are interested naturally exhibit loads that are periodic, in a statistical sense akin to seasonality in econometrics.

A load-balancing system for Lamina/CARE was designed using this load-characterization technique, and its implementation and experimentation with it in the context of the ELINT and AIRTRAC applications are the subject of a Ph.D. thesis in progress..

3.6. Concurrent and High Performance Lisp

In an attempt to understand the behavior of the Lisp language on shared memory machines, work was done on the QLisp system [Okuno 87, 3-443]. Although this work was not used directly by other parts of the project, it investigated some of the constraints on parallelizing production systems by studying the OPS5 language. This was the first large application implemented in QLisp and it was found that QLisp was able to encode all of the previously found sources of parallelism in OPS5, which amounted to a proof of concept for QLisp.

3.7. Distributed Cache Coherence

A significant aspect of our research into shared memory architectures was that of caching schemes and cache coherence. During our research we have designed and developed a new scalable cache coherence protocol for large scale shared memory architectures. This protocol has lower cost and more robust performance than previous solutions.

Cache coherence is an important and well known problem in shared memory multiprocessors. In such systems, each processor has an associated cache. The same data may be shared by different processors and thus copies of the data may be present in different caches. A cache coherence mechanism must exist in order to keep these multiple copies consistent with each other.

Bus-based shared multiprocessors usually provide some form of "snoopy" cache coherence protocol. The term "snoopy" arises from the fact that on a write, each cache watches the addresses transmitted on the bus. In the case that the cache has a copy of the data, it is either invalidated or updated. Snoopy cache coherence protocols rely on the bus to provide a global broadcast and such systems are limited by the number of processors a bus can support before it saturates.

In order to overcome the requirement of a broadcast medium, directory based protocols may be used. Earlier centralized directory based protocols maintain information about the caches that have copies of the line in a directory that is an extension of the main memory. This can potentially cause the directory to become a bottleneck. We have developed a new distributed directory protocol that is based on a singly-linked list of caches. Such a system is more scalable than earlier solutions in terms of both cost and performance. This research is detailed extensively in [Thapar 89a, 4-502], [Thapar 89b, 4-527] and [Thapar 90, 4-542].

4. Problem-Solving Frameworks

One of the key layers of the vertical slice strategy was the organization of problem-solving activity according to existing AI concepts. AI provided us with a number of different problem-solving frameworks as candidates for this study. In fact, the project committed itself at an early point to the Blackboard problem-solving model [Engelmore 88]. This was not an entirely arbitrary choice. The blackboard metaphor had already been applied successfully in the area of real-time signal processing [Nii 82], the selected problem domain for the project. It was also anticipated that the blackboard metaphor would help us to extract parallelism from the application in the way that the problems were formulated because the metaphor has a model of asynchrony built into it. For reasons detailed in [Rice 88a] the blackboard model turned out not to be as parallel as we might have hoped, but we still know of no better one for concurrent execution.

The development of problem-solving frameworks for parallel computation took two distinct courses. First was the development of a fairly conservative, concurrent implementation of an existing blackboard system to run on shared-memory machines. This was the Cage system, based on AGE [Nii 79] described in Section 4.1. The second course was to rethink the blackboard metaphor from scratch in the hope of achieving really high performance on distributed-memory multiprocessors, such as the CARE machine. This resulted in the Polygon system described in Section 4.2.

Three generations of papers have been produced describing the strategy of the project, the Cage and Polygon systems as they evolved, and the experimental results produced by these systems. The early motivation for the designs of these systems is outlined in [Nii 86, 3-196], while [Nii 88a, 3-205] and [Nii 88b, 3-233] show the evolution of these concepts and detail the experiments performed on the two systems, dwelling in particular on the factors that motivate and constrain the design and performance of parallel systems in general and of parallel problem-solving systems in particular. Numerous lessons were learned in the process of this research, which are listed in the above reports and in [Rice 88a, 4-139] and [Rice 89b, 4-219].

4.1. Cage

Cage (Concurrent AGE) [Aiello 86, 2-1], [Aiello 89, 2-26] is a reimplementaion of the AGE [Nii 79] blackboard system framework also developed at the Heuristic Programming Project at Stanford. The central idea behind Cage is that the blackboard model provides a certain amount of parallelism by its very nature. It should therefore be possible to exploit this parallelism without any major redesign or rethink for the problem-solving model. Cage is, therefore, an implementation, which is designed to allow the concurrent execution of a blackboard system through the concurrent execution of the knowledge sources and rules in the application (see Figure 5). A key factor in the design of Cage was that control of which rules and knowledge sources were to be run in parallel was left entirely to the user. This allowed the user to develop an application serially, debug it and then gradually increase the amount of parallelism exhibited by the application. This allowed the easy identification of bugs that were a function of the concurrent execution of small components of the application. It also allows the developer to experiment with different configurations of parallel execution so as to maximize the performance of the application, which might not be maximized by enabling all possible concurrency options because of contention problems.

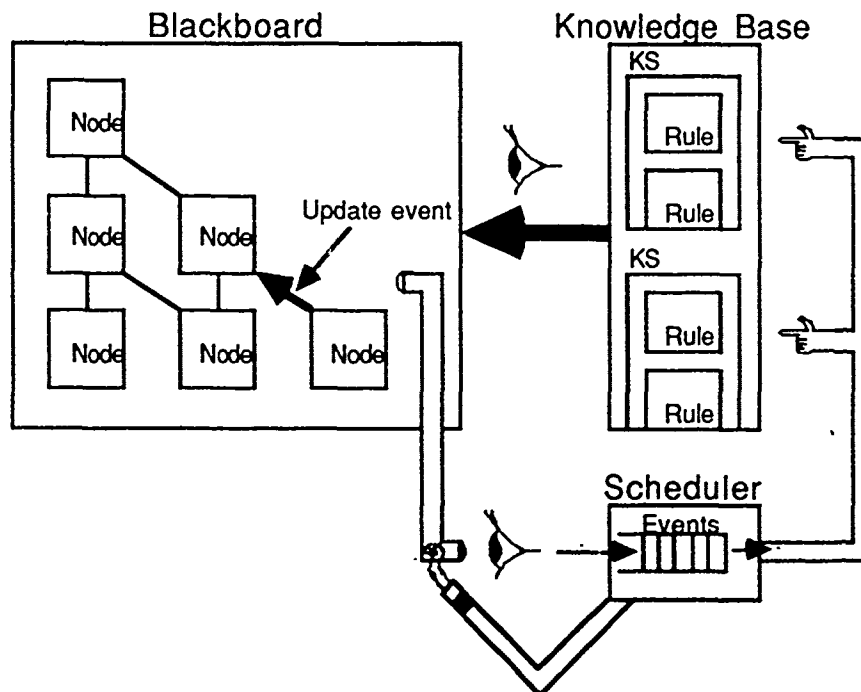


Figure 5. *The Cage Architecture. Update events are perceived by the scheduling component and collected in a global event queue. The scheduler selects the knowledge sources that are interested in any given event and can execute them in parallel. These knowledge sources in turn inspect the blackboard and perform updates that are seen by the scheduler.*

At the outset it was not known how difficult it would be to program such a system and how much performance could be expected, but it was thought that such an architecture might well be suitable for the current generation of multiprocessors, which mostly have a shared-memory design. Blackboard systems are typically implemented using a central, shared database to represent the blackboard. The match between the shared blackboard and the shared memory resource seemed to be worth investigating.

The Cage system was implemented first on a simple emulator, which emulated the functionality of a QLisp implementation without paying the costs of detailed simulation. It was later ported to run on the CARE simulator, using QL an implementation of QLisp and Multilisp language primitives built on top of the Lamina shared-variable programming interface [Saraiya 88, 4-324].

The Elint application, described in Section 5.1 was mounted on the Cage system and experiments to measure its speed-up and throughput were performed on it. These are detailed in [Aiello 88, 2-15] and [Rice 89a, 4-198]. The Cage system has shown that blackboard programs can, indeed, be run in parallel in a relatively simplistic manner. The performance of Cage, however, is restricted by a number of factors [Nii 88b, 3-233]:

- its implementation, which was not highly tuned;
- its architecture, which exhibits significant contention for global shared resources such as the event queue;
- the QLisp substrate, on which it is built. and
- the shared-memory hardware upon which it runs.

Thus, although the Cage architecture is a viable architecture for existing shared-memory hardware systems, because of the close link between the Cage programming model and its underlying hardware, we do not anticipate that future concurrent expert system tools will be built much like Cage. We believe that the trend of multiprocessor design is broadly away from shared-memory machines and towards distributed-memory designs because of their greater ability to scale.

4.2. Poligon

The expectation that the next generation of multiprocessors, for reasons of simplicity, performance and cost, are likely to be distributed memory machines required a reexamination of the blackboard model before it could be mounted on such a machine in a manner likely to deliver good performance. Poligon [Rice 86a, 4-1] and [Rice 86b, 4-19], a domain independent blackboard-like programming language and concurrent programming environment was developed in an attempt to address these needs. Poligon adopted the view that processors are going to be cheap and plentiful. Thus, it would be quite acceptable if necessary to allocate one processor or more to each node on the blackboard.

First the serializing, centralized control mechanism of conventional blackboard systems was discarded. Distributing the nodes of the blackboard over the processor network allowed the knowledge base to be spread over the blackboard as well, so as to eliminate any performance bottleneck due to the communication costs between the knowledge base and the blackboard. The simplest available rule invocation mechanism was selected, so as to maximize performance; rules were directly attached to slots of the nodes on the blackboard. A modification to a slot, to which a rule was attached, resulted in that rule being invoked. Rule invocations were spun off into different processes on different processors for execution, thus minimizing the length of the critical sections on the processors holding blackboard nodes and allowing multiple, simultaneous rule invocations for the same modified blackboard object (see Figure 6).

In practice, these mechanisms did indeed result in good performance, but they also resulted in significant problems. Many uncontrolled asynchronous processes, all reading and writing things in a shared database, are unlikely to reach a coherent or correct answer. Extra mechanisms had to be implemented, which allowed the blackboard nodes to have "goals" and the ability to evaluate their own performance with respect to the overall goal of the system. This allowed the blackboard nodes to make local decisions about whether to perform any modification operation attempted by a rule. The result was a sort of distributed hill-climbing behavior. Nodes iterated towards a good solution.

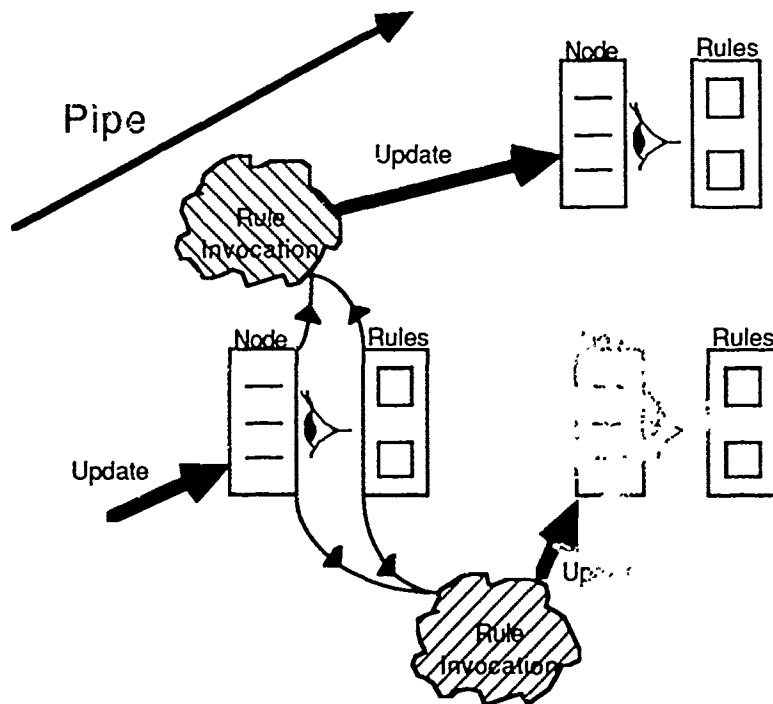


Figure 6. The Polygon Architecture. Updates on the blackboard are observed by rules which watch specific slots of blackboard nodes. These rules can fire in parallel causing further updates to the same or other nodes. This flow of updates from one node to another implicitly forms pipes, which increase the parallelism realizable by the system.

These mechanisms did not come without associated costs in terms of granularity. Although the Polygon system delivers very high performance when compared to other blackboard systems such as AGE, it nevertheless significantly lacks the performance provided by an application written directly in Lamina. However, an appropriate conceptualization and decomposition of a problem is the most difficult task for a programmer, and the most critical for obtaining speed-up. Polygon is a relatively high-level language compared to Lamina, and as such gives the programmer an edge in conceptualization. Polygon, therefore, provides a fairly general concurrent implementation of the blackboard problem-solving model with all of the advantages of abstraction and modularity that this confers. It does so, however at a price. A detailed rationale and description of Polygon's design and implementation can be found in [Rice 89b, 4-219]. This paper, through a detailed discussion of the factors that limit the performance of blackboard systems in general and concurrent blackboard systems in particular, shows the motivation for the design of different aspects of Polygon, detailing the evolution of numerous different aspects of the Polygon system, and highlighting the deficiencies of each design that was attempted and then superseded. It also describes a number of means by which the performance of Polygon could be improved by superior compilation if it were to be turned into a production quality system.

The Elint application, described in Section 5.1, was implemented in the Cage, Polygon and Lamina systems. The results of these experiments are reported in [Rice 88b, 4-165], [Rice 89a, 4-198] and [Nii 88b, 3-233]. These reports also describe both the motivation for and architecture of Polygon as well as highlighting numerous experimental results, which are analyzed with a view to the lessons that can be learned from Polygon's performance. In [Nii 89, 3-298] a discussion is given on the way in which the serial Elint application was recoded to as to run on the concurrent Polygon framework. This has numerous implications for the development of concurrent real-time signal understanding problems.

Another application called ParAble, implemented using the Poligon framework, is described in Section 5.3.

5. Applications

Our research strategy called for the project work to be application driven. The search for an application was guided primarily by two considerations (a) practical versions of the application would demand significant speed-up in execution, and (b) methods for approaching the application held a certain obvious potential for concurrent execution. Based on these considerations, the application area chosen was real-time signal understanding. Existing blackboard systems, such as HASP/SIAP [Nii 82] and Tricero [Williams 84] had shown both that the blackboard problem-solving model was appropriate for this domain and that the performance deliverable using existing blackboard tools was inadequate to field such systems.

What we needed, therefore, was a problem which was complex enough to give us a reasonable model of a real system, and yet simple enough that we would not spend too much effort on the mechanics of its implementation. We decided initially to focus on a problem called Elint, a system for the understanding of passive radar signals. This application, derived from Tricero, is described in Section 5.1

After much experimentation it was determined that our ability to exploit parallelism was being constrained by the problem we were using - it was not sufficiently complex in terms of the amount of knowledge and the amount of data available. In the search for a more knowledge-rich and computationally intensive application we developed the AirTrac application, a system for interpreting active radar signals, which is described in Section 5.2.

Experiments were also performed in application domains other than that of real-time signal understanding; ParAble, a system for fault-finding in particle accelerator beam lines has been developed using the Poligon framework. This work is described in Section 5.3. A number of numerical or semi-numerical programs have also been developed during our more hardware-related experiments. These investigations are mentioned in Section 5.4.

5.1. Elint

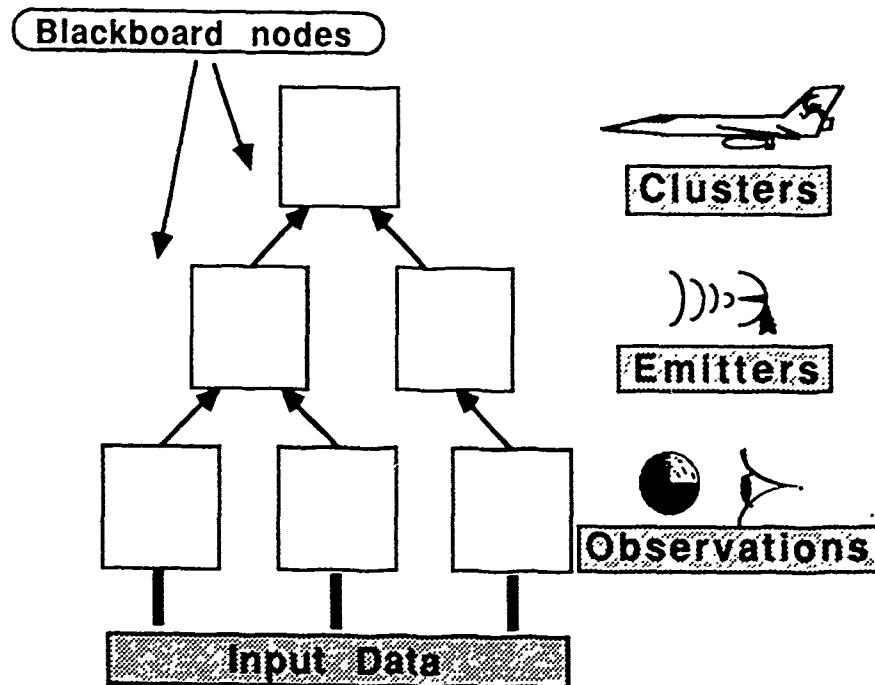


Figure 7. The Elint Application. Sensor data is abstracted into hypothetical radar emitters, which are tracked as clusters of emitters.

Elint is a soft real-time system for the interpretation of passive radar signals. Data are collected from a number of receiving stations and are integrated so as to allow the system to track radar emitting aircraft as they pass through the monitored airspace. The data are abstracted into hypothetical radar emitting platforms. These emitters are in turn collected into clusters of emitters, which might represent a number of planes or a single plane using multiple radar systems, as is often the case with modern military aircraft (see Figure 7).

Elint was first implemented using the CAOS system. It was originally thought that this work would take only a couple of months to do. In fact, the complete task — implementation, experimentation and analysis of results — took 18 months. We learned early on that it is by no means a trivial matter to reimplement an existing, serial application in a parallel environment. These initial experiments, which are detailed in [Brown 86, 2-78], delivered both qualitative and quantitative results concerning the performance of a concurrent system such as we were envisaging, over a variety of different numbers of processors, and investigated such critical areas as overall speed-up and "solution quality." The concept of solution quality arises in many knowledge-based systems, where there is no such thing as *the* correct problem solution, but only *satisficing* (i.e., acceptable) problem solutions. A primary objective of the experiments was to investigate the trade-offs between the imposition of various synchronizations (and the resulting loss of concurrency) and the quality of the problem solution.

Since the CAOS implementation, Elint has been implemented three times; using Lamina [Delagi 88b, 2-446] [Saraiya 89, 4-337] and the Cage [Aiello 88, 2-15] and Poligon [Rice 88b, 4-165] [Nii 88b, 3-233] frameworks and a number of experiments have been performed on them. In order to perform any of these experiments we found it necessary to develop a technique for performance measurement that actually measured the sustainable data-rate that the system under experimentation could maintain for a given number of

processors without being swamped by the incoming data, i.e. while still giving non-increasing latency in its outputs. This technique is discussed in detail in [Nii 88b, 3-233].

Each of these reports details not only the underlying architecture of the solution, for example an object-based, pipelined decomposition in the case of the Lamina experiments, but also covers extra areas for experimentation appropriate to the framework being studied and the intended level of abstraction of the framework. These areas included: multiple grain sizes [Nii 88b, 3-233], [Aiello 88, 2-15], speed-up as a function of only pipeline parallelism [Nii 88b], [Rice 88b, 4-165], [Rice 89a, 4-198], scaling with respect to knowledge base size (number of rules) [Nii 88b], [Rice 88b], [Rice 89a] and load balancing [Saraiya 89, 4-337].

5.2. AirTrac

The development of the Elint application showed that the amount of parallelism that could be demonstrated was much more dependent on the application than we had anticipated. Following the analysis of Reddy and Newell [Reddy 77], we hypothesized that by extracting parallelism at the different levels of the system's implementation hierarchy we could gain multiplicative speed-up. The analysis of our experiments showed that the speed-up was disappointing, largely because the application itself did not have enough potential for the exploitation of parallelism.

Our response to this was to develop an application which would really stretch the hardware and software we were developing in a realistic manner, the AirTrac application [Delaney 86, 2-459].

The AirTrac problem domain sounds superficially like that of Elint. It was a system for the interpretation of radar data, though in this case the radar systems modeled were active, not passive. Unlike Elint, AirTrac was designed to go beyond simply tracking aircraft and identifying likely threats. The scenario for AirTrac was the detection of "smugglers" flying across a border. The problem faced by existing radar users is that a large number of legitimate aircraft travel in the same airspace as smugglers. Smugglers may take advantage of variations in terrain in order to find areas of poor or no radar reception. They also resort to other evasive tactics. Thus to identify and track smugglers, the AirTrac application had to interpret the behavior of the aircraft it was tracking over time.

The system was designed in a number of layers so that different implementation efforts could be decoupled. The first subsystem implemented was called the Data Association component [Nakano 87, 3-149], and is the subsystem, which most closely matches the Elint application. It was initially intended that this component would be implemented using the Poligon framework. It was found, however, that the simulation of the Poligon system for a problem as complex as AirTrac would take prohibitively long. Consequently AirTrac was implemented directly in Lamina. Substantial speed-up was shown (of the order of one hundredfold with the use of one hundred processors), which seemed to increase linearly with the number of processors. This encouraging result was achieved by the use of replicated pipelined sequences of objects processing the input data. It was further found that the degree of correctness of the solution was not compromised by the decomposition of the problem so as to make it execute concurrently, nor was it affected by highly overloaded input data conditions [Nakano 87].

The second component of AirTrac, Path Association, was significantly more knowledge intensive than the first. The task of the Path Association module was to group together tracks produced by the Data Association component into plausible aircraft flight paths. This subsystem was also implemented directly in Lamina initially. However, programming

in the raw Lamina framework was too complex and time-consuming, so a layer was built on top of Lamina, called ELMA [Noble 88b, 3-409], which provided the abstraction model needed for the implementation. The experiments described in [Noble 88a, 3-309] provided confirmation of the earlier results obtained with the Elint application.

The project leaders decided to continue to focus resources on the Path Association component, where there was still much to learn and where we believed further speed-up and insight could be obtained. In [Muliawan 89, 3-48], further experiments in the AirTrac application's Path Association component are described. The effect of high-level control strategies on system performance is discussed, as is the effect of varying the frequency and width of the input data, for various numbers of processors. System performance was measured both in terms of sustainable data rate and in terms of latency, "excess ratio" and capacity. The relationship between the quantitative and qualitative performance of the system is also discussed.

The final, most abstract, component of AirTrac — Platform Interpretation — was intended to classify the aircraft being tracked by the Data Association and Path Association modules and to predict their behavior, based on these classifications and their past actions. A platform classification module was implemented, using a general, forward-chaining, concurrent classification system [Clancey 84]. These experiments demonstrated speedup and are described in [Maegawa 90, 3-20]. The key idea was to view the classification system as a network of nodes representing classifications and subclassifications. Speedup was achieved through the concurrent execution of multiple instances of the classification network. Because the input track information was continuously acquired over time, the system necessarily supports periodic reevaluation of all classifications. That is, all conclusions drawn by the system may be continuously modified as new supporting evidence enters the network.

5.3. ParAble

The ParAble project [Bandini 89] was an attempt, by choosing a completely different application domain, to test the generality of the problem-solving model offered by Poligon. To do this we made a parallel implementation of the ABLE system [Selig 87], developed also at Stanford.

The objective of the ABLE project was to find a rational and fast way to diagnose particle accelerator beamlines. These large and complex machines are very prone to beam alignment problems due either to misalignment of the magnets, which steer and focus the beam, or to problems with the power supplies to those magnets, which result in the magnets not having the desired strength. These beamlines are so complex that it can take many months of knob-twiddling in order to commission them.

By the use of an analytic model of the transfer functions of the beam-line components, and a number of heuristics that use successive runs of the model, comparing the results with the real data to locate the faults, the ABLE system was able to find faults in such systems in about ten minutes. As particle accelerators become more complex there may well be a need to control them in real time, so although there is no immediate need for higher performance in the ABLE system, it is not unreasonable to suppose that there might be in the future.

A number of Experiments have been performed on ParAble, detailed in [Bandini 89, 2-58]. The realizable parallelism in this project was, again, found to be limited mostly by the availability of data parallelism.

5.4. Numerical and Semi-numerical programs

The expert systems mentioned above are not ideal applications for multiprocessor execution. They are irregular and very data dependent. A large body of applications already exists in the area of numerical and semi-numerical processing, which will require the speed-up associated with parallel execution. Indeed, such programs are already being run on a number of multiprocessors. It is therefore essential that any machine designed with a view to being general-purpose must also be able to execute these regular, algorithmic problems efficiently. A number of small numerical programs were developed, to be used for experiments in system architectures and topologies [Byrd 88a, 2-181] and [Byrd 88b, 2-196]. These experiments allowed us to test our hardware and software ideas in a much more controllable way than we can with any expert system application.

6. Conclusions, Observations Results

The previous sections summarized the experiments performed, the types of computations explored, the simulation engines built to conduct the experiments, and some experimental results. We tied each of these to specific technical reports of the project. In this section, we add conclusions, results, and observations of a general nature. These have been drawn from across the range of experiments performed, and we believe will be of interest to a large body of computer scientists interested in the problems of parallel computation.

We begin with words of caution. Our experiments were performed mainly in the area of symbolic problem solving by computer—that is, the traditional mainstream area of artificial intelligence research. The kinds of entity typically manipulated were symbolic objects and rules, not algebraic formulas or matrices of numbers. The computations were largely symbolic computations (as, for, example typically performed in the LISP language).

Low-level representational choices, constituting the focus of our experiments, and therefore influencing our conclusions, include object-orientation with message passing, on a MIMD-type machine. Most experiments were performed using distributed-memory system architecture. One final caveat: all experiments were performed on our instrumented simulator. Though we are confident of the quality and veracity of the simulated computations, a simulator is only a model of reality.

Finally, one must always keep in mind the simple algebraic relationship (often called "Amdahl's law"). The ultimate limit to speedup of computation on a parallel machine, the "Amdahl limit" is determined by the residual amount of "serial processing" remaining in the computation after the programmer has extracted and used all the parallel computation schemes possible. Thus, for example, if the intrinsic serial component of the computation is no less than 1%, then the overall speedup can not exceed two orders of magnitude ($\times 100$).

To repeat, in reading what follows, the reader should have in mind the general picture of a two-dimensional network of LISP computers (each with a communications subprocessor) of size $N \times N$ (typically 10×10 or 16×16). These processors are receiving, as input, streams of encoded sensor data, and, with some latency (e.g., milliseconds or seconds of sensor time), computing hypotheses of platform track segments, platform identity, etc. Computational work is distributed over the multiprocessor but many of the nodes of the $N \times N$ network are not necessarily busy all the time.

6.1. Speedup over serial computation

An early result by Gupta [Gupta 86] for parallel rule-based computation indicated that a speedup of approximately one order of magnitude ($OM = \times 10$) was achievable. Our experiments confirmed that speedups of approximately one OM were readily achieved, but not without significant programming work and ingenuity.

Speedups of 2 OM were very difficult to achieve for individual problem solving efforts but were achievable for groups of these efforts (e.g., one aircraft versus many aircraft). We refer to such application circumstances as being characterized by "data parallelism." In data-parallel situations (which may be quite normal in the world of computing applications), the overall intrinsic parallelism can be made sufficiently high relative to the intrinsic serialization that 2 OM ($\times 100$) is achievable.

Speedups of 3 OM ($\times 1000$) were well beyond the reach of any techniques, or any problem size, explored in this study.

Because of the limits imposed by the inherent serialization, even when the application is augmented by favorable data parallelism, speedup will reach a ceiling, beyond which one cannot push the speedup by simply increasing the number of computing nodes.

When working in a problem environment in which data enter in continuous streams, determining how to measure effective speedup is an important issue. Our observation is that stable latency in delivering a computational hypothesis (i.e., answer), after the corresponding data are presented in the input stream, is the appropriate measure. Thus, for each multiprocessor configuration many input data rates must be tried before a stable latency can be found. The determination of speedup is thus a lengthy process. This technique is in contrast to the more common method of simply dividing the application's run-time into the run-time on a single processor. We found this latter method to be highly deceptive for real-time and data-reactive applications.

Our observation is that the most significant sources of "parallelization" of a problem are to be found in the application itself, and therefore by the application programmer. Of course, system-level language and architectural features must be there to support this human programmer creativity. Because the programmer plays such a vital role in realizing the speedup from parallelization, programmer language tools for conceptualizing and writing concurrent programs are of great importance. Equally important are well-instrumented debugging tools to aid the programmer since the complexity of parallel, run-time environments is well beyond anything that even the best programmers have been trained to cope with.

We can not emphasize too much the importance of a variety of software instruments in tuning parallel computations. This must be provided, either by the manufacturer (in the form of software instruments responding to hardware measurements) or by a fully instrumented, carefully designed simulator of the parallel hardware. Today, neither of these is routinely done. We found the feedback provided by the instrumentation in our simulator essential in refining designs: to break bottlenecks, balance pipelines, evaluate load balancing schemes, and so forth.

Careful thought must be given to instrumentation at the application level (not just the machine level) because execution behavior is just too complex for programmers to think through. Experimentation is needed to decide how to instrument at the application level. These decisions are not clear and straightforward.

6.2. Pipelines

The architectural approach to concurrency that most consistently proved effective in our experiments was building pipelines for computations, and replicating them when necessary.

- For our data-streaming application environment, pipelines were a natural fit for exploiting the intrinsic parallelism and the data parallelism in the problem. (By intrinsic parallelism we mean that the stages in the pipeline correspond in a natural way to steps in the problem solving process: inference steps; subproblem pipes; multiple hypotheses and goals, etc.)
- Pipelines must be carefully balanced during execution. For example, if a pipeline consists of a series of invoked knowledge sources, these must have similar "grain size" and data density.
- Some computations branch in a fan-out manner. The pipeline approach to fan-out computations can be made to yield good speedup.
- Conversely, fan-in of computations is disruptive of pipelining, and seriously impacts the speedup that can be realized. A particular type of fan-in occurs when symbols are passed "up" an abstraction hierarchy, e.g., when special cases are recognized as instances of a more general case. In an abstraction hierarchy pipeline, it is important that the communication up the hierarchy should be designed to decrease, proportionately to the amount of "branchiness" at the lower levels, as the symbolic data flows "up" the hierarchy.

Our experiments with pipelines showed that resumable processes are rarely needed (hence, the architecture need not treat this issue as one of high priority). Most computational grains can be realized in such a way that they, and therefore the processes in which they run, can run to completion. As a corollary, our experiments showed that the significant costs associated with process instantiation and process switching can often be avoided by the use of this run-to-completion programming model.

6.3. Basic computational metaphor

The object-oriented message-passing paradigm, was found to be a natural and comfortable metaphor in conceptualizing concurrent programming and in thinking through the issues of distributed memory and communications. This model was found to be highly compatible with the underlying message-passing, distributed-memory system architecture used in our experiments. It is not clear that all object-oriented models will have this property. For example, multimethods in CLOS may be incompatible with distributed memory architectures.

6.4. Communication

Our experiments showed that communication patterns among processes were surprisingly static. The implication for interprocess communication is to prefer "streams" to "futures," i.e. to amortize the cost of initiating communication between processes by maintaining connections and passing more than one value between connected processes.

An architecture for hardware-supported multicast was designed that provided for adaptive cut-through routing. Our experiments proved its effectiveness for deadlock avoidance. The scheme provides cut-through multicast without requiring dedicated storage in the communication facilities for a full packet.

6.5. Problem Solving Methods

A relatively straightforward "parallelization" of the blackboard problem solving framework effective in serial environments, i.e., Cage (Concurrent AGE) versus AGE, proved to be ineffective, i.e., did not deliver much speedup. The key issue was centralized control, and the consequent low "Amdahl limit", mentioned earlier, that resulted from the synchronization at this central control point.

A radically reorganized blackboard framework (Poligon) using decentralized control, enabled significant speedup.

- The Poligon experiments showed that although problems can be solved without global control, some limited, non-local control was needed (for example, to manage node creation).
- For the minimal control regime, each solution node should have its own goals and evaluation functions (to enable local "hill climbing"). The use of local "hill-climbing" resulted, in our experiments, in globally valid, mutually consistent results, in spite of the lack of global coordination. It was observed that this local hill-climbing reduces the latency in getting the plausible answer (i.e., performance improves).

We must caution, however, that not all problems can be solved (effectively or at all) with a control regime that enforces this "local" view. Here, as before, the choice of approach is application-dependent, or at least dependent on the way a problem is formulated. One must consider the tradeoff between global control versus local knowledge (in the form of evaluation functions and local control).

Concerning rule processing in problem solving, our experiments showed that the rules within sets of rules (i.e., knowledge sources) can be run in parallel, and that this contributes to speedup. However, running rules in parallel requires encapsulation of the data used by the rules (i.e., the rule context). Because the context can become obsolete by the time the rule is processed, this technique needs to be used in conjunction with local hill climbing to prevent outdated and invalid conclusions from being recorded.

We observe that the quality of a solution is an issue in parallel problem solving. AI problem solving methods usually "satisfice," i.e., there is no one "right answer." But in real applications some answers are better than others. With decentralized control, it can be difficult for the programmer to orient the program's behavior always in the direction of the "better" answers. Here is an obvious tradeoff—the more centralized the control, the more programmer guidance toward the "better" answers, the less the speedup (too much centralized control, synchronization, i.e. a low Amdahl limit). In our experiments we were able to preserve solution quality by keeping data consistent and controlling order-critical tasks. However, we did not extract a general technique or even a general engineering understanding of this fascinating issue.

A related issue is the tradeoff between problem decomposition and degree of coupling among the decomposed subproblems. We observed that as problems are decomposed into smaller grains, the subproblems became more interdependent (more data sharing, more communication), nullifying the potential parallelism. Thus, optimal grain size is highly dependent on the application as well as on the processing overhead.

6.6. Development strategy

We observed a three-part strategy to be important.

- a. Do a serial execution of the parallel program. This removes the obvious elementary bugs.
- b. Proceed to a relatively crude parallel simulation, using a purely functional simulator with randomized scheduling of resources. This will catch the first level of "parallel processing" bugs.
- c. Proceed to the fine-grained instrumented simulator for the experiments and the performance tuning.

6.7. Analysis of the application

Not enough can be said in motivating a careful study of the application to understand its intrinsic parallelism. Previously employed serial processing schemes used to handle the application can be seriously misleading and ineffective.

In one of our experiments (ParAble), the domain expert, a accelerator physicist, was able to reconceptualize the computation in a way that was not only "highly parallel" (enabling a good experimental result for us) but also "highly natural," enabling him to understand his problem with great clarity.

Applications analysts should not try to "force fit" their application to the parallel computing metaphor, but rather should seek a natural, intrinsic parallel structure of the computation.

6.8. Load balancing

Balancing the computational load among the nodes of a multicomputer is a serious issue of performance and economics. It takes both computing and communication to perform effective load balancing, so an obvious performance tradeoff is involved.

Our experiments on adaptive global load balancing used an explicit stochastic-process model to estimate the time evolution of processor loading and to model the dissemination of load-information. This model allows improved estimates to be made of system-wide loading, which allows a given level of load balance to be achieved with far fewer object migrations. This in turn improves the system's performance (in terms of consistently meeting latency deadlines), provided that migration costs are sufficiently high (remember the tradeoff mentioned above). The performance improvement achieved and the circumstances under which it can be achieved, however, were found to be seriously limited by the unexpectedly poor correlation between load-estimate quality and load balance.

7. Bibliography of Expert Systems on Multiprocessor Architectures: Project Publications

- [Aiello 86] Nelleke Aiello. *User-Directed Control of Parallelism; The CAGE System*. Technical Report KSL-86-31, Heuristic Programming Project, Computer Science Department, Stanford University, 1986. Also in Proceedings of DARPA Expert Systems Workshop, April 1986, Science Applications International Corp., Arlington, VA. Report SAIC-86/1701.
- [Aiello 88] Nelleke Aiello. *Cage: The Performance of a Concurrent Blackboard Environment*. Technical Report KSL-88-80, Heuristic Programming Project, Computer Science Department, Stanford University, December 1988.
- [Aiello 89] Nelleke Aiello. *The Cage System User's Manual*. Technical Report KSL-89-86, Heuristic Programming Project, Computer Science Department, Stanford University, December 1989.
- [Bandini 89] Jean-Christophe Bandini and James Rice. *An Application in Polygon*. Technical Report KSL-89-43, Heuristic Programming Project, Computer Science Department, Stanford University, 1989.
- [Brown 86] Harold Brown, Eric Schoen, Bruce A. Delagi. *An Experiment in Knowledge-Base Signal Understanding Using Parallel Architectures*. Technical Report STAN-CS-86-1136 (KSL-86-69), Heuristic Programming Project, Computer Science Department, Stanford University, October 1986.
- [Byrd 87a] Gregory T. Byrd, Russell T. Nakano and Bruce A. Delagi. *A Point-to-Point Multicast Communications Protocol*. Technical Report KSL-87-02, Heuristic Programming Project, Computer Science Department, Stanford University, January 1987.
- [Byrd 87b] Gregory T. Byrd, Bruce A. Delagi. *Considerations for Multiprocessor Topologies*. Technical Report KSL-87-07, Heuristic Programming Project, Computer Science Department, Stanford University, January 1987. Also in Proceedings of DARPA Knowledge Based Systems Workshop, April 1987.
- [Byrd 87c] Gregory T. Byrd, Russell T. Nakano, Bruce A. Delagi. *A Dynamic, Cut-Through Communications Protocol with Multicast*. Technical Report STAN-CS-87-1178 (KSL-87-44), Heuristic Programming Project, Computer Science Department, Stanford University, August 1987.
- [Byrd 88a] Gregory T. Byrd, Bruce A. Delagi. *A Performance Comparison of Shared Variables vs. Message Passing*. Technical Report KSL-88-10, Heuristic Programming Project, Computer Science Department, Stanford University, January 1988. Also in Proceedings of Third International Conference on Supercomputing, pages 1-7, Boston, MA, March 1988 International Supercomputing Institute.

- [Byrd 88b] Gregory T. Byrd, Nakul P. Saraiya and Bruce A. Delagi. *Multicast Communication in Multiprocessor Systems*. Technical Report KSL-88-81, Heuristic Programming Project, Computer Science Department, Stanford University, June 1989.
- [Byrd 89] Gregory T. Byrd. *Support for Fine-Grained Message Passing in Shared Memory Multiprocessors*. Technical Report KSL-89-15, Heuristic Programming Project, Computer Science Department, Stanford University, March 1989. Also to appear in Proceedings of the 5th Annual Computer Science Symposium, University of South Carolina, April 7-8, 1989
- [Davies 86] Davies, Byron. *CAREL: A Visible Distributed Lisp*. Technical Report KSL-86-14, Heuristic Programming Project, Computer Science Department, Stanford University, March 1986. Also in Proceedings of DARPA Expert Systems Workshop, April 1986, Science Applications International Corp., Arlington, VA. Report SAIC-86/1701.
- [Delagi 86a] Bruce A. Delagi, Nakul P. Saraiya, Gregory T. Byrd. *LAMINA: CARE Applications Interface*. Technical Report KSL-86-67, Heuristic Programming Project, Computer Science Department, Stanford University, November 1987. Also in Proceedings of Third International Conference on Supercomputing, pages 12-21, Boston, MA, March 1988 International Supercomputing Institute.
- [Delagi 86b] Bruce A. Delagi, Nakul P. Saraiya, Sayuri Nishimura, Gregory T. Byrd. *An Instrumented Architectural Simulation System*. Technical Report KSL-86-36, Heuristic Programming Project, Computer Science Department, Stanford University, January 1987. Also in [Stanford 88] and *Artificial Intelligence and Simulation: The diversity of Application*. The Society for Computer Simulation International, February 1988.
- [Delagi 87] Bruce A. Delagi, Nakul P. Saraiya, Sayuri Nishimura, Gregory T. Byrd. *Instrumented Architectural Simulation*. Technical Report STAN-CS-87-1189 (KSL-87-65), Heuristic Programming Project, Computer Science Department, Stanford University, November 1987. Also in Proceedings of Third International Conference on Supercomputing, pages 8-11, Boston, MA, March 1988 International Supercomputing Institute.
- [Delagi 88a] Bruce A. Delagi, Nakul P. Saraiya, Gregory T. Byrd and Sayuri Nishimura. *CARE User's Manual*. Technical Report KSL-88-53, Heuristic Programming Project, Computer Science Department, Stanford University, September 1990.
- [Delagi 88b] Bruce A. Delagi and Nakul P. Saraiya. *ELINT in LAMINA: Application of a Concurrent Object Language*. Technical Report KSL-88-33, Heuristic Programming Project, Computer Science Department, Stanford University, July 1988. Also in SIGPLAN Notices, February 1989.

- [Delaney 86] John R. Delaney. *Multi-System Report Integration Using Blackboards*. Technical Report KSL-86-20, Heuristic Programming Project, Computer Science Department, Stanford University, March 1986. Also in Proceedings of DARPA Expert Systems Workshop, April 1986, Science Applications International Corp., Arlington, VA. Report SAIC-86/1701.
- [Hailperin 88] Max Hailperin. *Load Balancing for Massively Parallel Soft-Real-Time Systems*. Technical Report KSL-88-62, Heuristic Programming Project, Computer Science Department, Stanford University, August 1988. A condensed version of appears in the proceedings of "Frontiers '88: The Second Symposium on the Frontiers of Massively Parallel Computation."
- [Maegawa 90] Hirotoshi Maegawa. *The Parallel Solution of Classification Problems*. Technical Report KSL-89-68, Heuristic Programming Project, Computer Science Department, Stanford University, March 1990.
- [Muliawan 89] Djuki Muliawan. Performance Evaluation of a Parallel Knowledge-Based System. Technical Report KSL-89-51, Heuristic Programming Project, Computer Science Department, Stanford University, June 1989
- [Nakano 87] Russell T. Nakano, Masafumi Minami. *Experiments with a Knowledge-Based System on a Multiprocessor*. Technical Report KSL-87-61, Heuristic Programming Project, Computer Science Department, Stanford University, October 1987. Also in a shortened form in Proceedings of Third International Conference on Supercomputing, pages 22-24, Boston, MA, March 1988 International Supercomputing Institute.
- [Nii 86] H. Penny Nii. *CAGE and POLIGON: Two Frameworks for Blackboard-based Concurrent Problem Solving*. Technical Report KSL-86-41, Knowledge Systems Laboratory, Computer Science Department, Stanford University, April 1986. Also in Proceedings of DARPA Expert Systems Workshop, April 1986, Science Applications International Corp., Arlington, VA. Report SAIC-86/1701.
- [Nii 88a] H. Penny Nii, Nelleke Aiello, James Rice. *Frameworks for Concurrent Problem Solving: A Report on CAGE and Poligon*. Technical Report KSL-88-02, Heuristic Programming Project, Computer Science Department, Stanford University, March 1988. Also in [Engelmore 88], and proceedings of AAAI 88 Blackboard Workshop.
- [Nii 88b] H. Penny Nii, Nelleke Aiello, James Rice. *Experiments on Cage and Poligon: Measuring the performance of Parallel Blackboard Systems*. Technical Report KSL-88-66, Heuristic Programming Project, Computer Science Department, Stanford University, February 1989. Also in *Distributed Artificial Intelligence II*. L. Gasser and M. N. Huhns (eds.). Pitman Publishing Ltd. and Morgan Kaufmann, 1989.

- [Nii 89] H. Penny Nii, James Rice. Signal Understanding and Problem Solving: A Concurrent Approach to Soft Real-Time Systems. Technical Report KSL-89-73, Heuristic Programming Project, Computer Science Department, Stanford University, October 1989. Also in Proceedings of Twenty-third Asilomar Conference on Signals, Systems and Computers. October-November 1989, Maple Press, San Jose, CA.
- [Noble 88a] Alan C. Noble and Everett C. Rogers. *AIRTRAC Path Association: Development of a Knowledge-Based System for a Multiprocessor*. Technical Report KSL-88-41, Heuristic Programming Project, Computer Science Department, Stanford University, June 1988.
- [Noble 88b] Alan C. Noble. *ELMA Programmer's Guide*. Technical Report KSL-88-42, Heuristic Programming Project, Computer Science Department, Stanford University, August 1988.
- [Okuno 87] Hiroshi G. Okuno, Anoop Gupta. *Parallel Execution of OPS5 in QLISP*. Technical Report KSL-87-43, Heuristic Programming Project, Computer Science Department, Stanford University, June 1987.
- [Rice 86a] James Rice. *Poligon, A System for Parallel Problem Solving*. Technical Report KSL-86-19, Heuristic Programming Project, Computer Science Department, Stanford University, April 1986. Also in Proceedings of DARPA Expert Systems Workshop, April 1986, Science Applications International Corp., Arlington, VA. Report SAIC-86/1701.
- [Rice 86b] James Rice. *The Poligon User's Manual*. Technical Report KSL-86-10, Heuristic Programming Project, Computer Science Department, Stanford University, May 1989.
- [Rice 88a] James Rice. *Problems with Problem-Solving in Parallel: The Poligon System*. Technical Report KSL-88-04, Heuristic Programming Project, Computer Science Department, Stanford University, January 1988. Also in Proceedings of Third International Conference on Supercomputing, pages 25-34, Boston, MA, March 1988 International Supercomputing Institute, *Artificial Intelligence, Simulation and Modelling*, Lawrence Widman (ed.), John Wiley Publishing Company, New York 1989.
- [Rice 88b] James Rice. *The Elint Application on Poligon: The Architecture and Performance of a Concurrent Blackboard System*. Technical Report KSL-88-69, Heuristic Programming Project, Computer Science Department, Stanford University, December 1988. Also in Proceedings of IJCAI 89.
- [Rice 88c] James Rice. *The Advanced Architectures Project*. Technical Report KSL-88-71, Heuristic Programming Project, Computer Science Department, Stanford University, March 1989, also in AI Magazine Winter 1989, Vol. 11, No. 4 pages 26-39.

- [Rice 89a] James Rice and Nelleke Aiello. *See How They Run... The Architecture and Performance of Two Concurrent Blackboard Systems*. Technical Report KSL-89-08, Heuristic Programming Project, Computer Science Department, Stanford University, March 1989. Also in *Blackboard Architectures and Applications: Current Trends*, V. Jagannathan and R. Dodhiawala (eds.), Academic Press, 1989.
- [Rice 89b] James Rice. *The Design of a High Performance, Concurrent Problem Solving System...and many Lessons Learned on the Way*. Technical Report STAN-CS-89-1294 (KSL-89-37), Heuristic Programming Project, Computer Science Department, Stanford University, November 1989.
- [Saraiya 86] Nakul P. Saraiya. *AIDE: A Distributed Environment for Design and Simulation*. Technical Report KSL-86-56, Heuristic Programming Project, Computer Science Department, Stanford University, June 1986. Also in Proceedings of DARPA Expert Systems Workshop, April 1986, Science Applications International Corp., Arlington, VA. Report SAIC-86/1701 (preliminary version).
- [Saraiya 88] Nakul P. Saraiya. *A Shared Memory Lisp Package for CARE*. Technical Report KSL-88-85, Heuristic Programming Project, Computer Science Department, Stanford University, January 1989.
- [Saraiya 89] Nakul P. Saraiya. *Design and Performance Evaluation of a Parallel Report Integration System*. Technical Report KSL-89-16, Heuristic Programming Project, Computer Science Department, Stanford University, April 1989.
- [Saraiya 90a] Nakul P. Saraiya, Bruce A. Delagi and Sayuri Nishimura. *SIMPLE/CARE. An Instrumented Simulator for Multiprocessor Architectures*. Technical Report KSL-90-66, Heuristic Programming Project, Computer Science Department, Stanford University, September 1990.
- [Saraiya 90b] Nakul P. Saraiya and James P. Rice. *The LAMINA Programming Model: A Worked Example*. Technical Report KSL-90-82, Heuristic Programming Project, Computer Science Department, Stanford University, December 1990.
- [Schoen 86] Eric Schoen. *The CAOS System*. Technical Report STAN-CS-86-1125 (KSL-86-22), Heuristic Programming Project, Computer Science Department, Stanford University, March 1986. Also in Proceedings of DARPA Expert Systems Workshop, April 1986, Science Applications International Corp., Arlington, VA. Report SAIC-86/1701.
- [Thapar 89a] Manu Thapar and Bruce A. Delagi. *Design and Implementation of a Distributed Directory Cache Coherence Protocol*. Technical Report KSL-89-72, Heuristic Programming Project, Computer Science Department, Stanford University, April 1989.

- [Thapar 89b] Manu Thapar and Bruce A. Delagi. *Distributed Cache Coherence for Large Scale Shared Memory Multiprocessors*. Technical Report KSL-89-83, Heuristic Programming Project, Computer Science Department, Stanford University, December 1989.
- [Thapar 90] Manu Thapar and Bruce A. Delagi. *Cache Coherence for Large Scale Shared Memory Multiprocessors*. Technical Report KSL-90-41, Heuristic Programming Project, Computer Science Department, Stanford University, June 1990.

8. Bibliography of Referenced Work Not Performed on the Project

- [Abelson 83] Harold Abelson and Gerald Jay Sussman. *Structure and Interpretation of Computer Programs*. MIT Press, Cambridge, MA 1983.
- [Clancey 84] Clancey, W. J., *Classification Problem Solving*, Technical Report STAN-CS-84-1018 Heuristic Programming Project, Computer Science Department, Stanford University. Also in Proceedings of the AAAI-84 (1984) 49-55.
- [Engelmore 88] Robert Engelmore and Tony Morgan (eds.) *Blackboard Systems*. Addison-Wesley Publishing Company Inc., Menlo Park 1988.
- [Gabriel 84] Richard P. Gabriel and John McCarthy. *Queue-based Multi-processing Lisp*. Proceedings of the ACM Symposium on Lisp and Functional Programming: 25-44, August, 1984.
- [Gupta 86] Anoop Gupta. *Parallelism in Productions Systems*. Technical Report, Computer Science Department, Carnegie-Mellon University, March, 1986. Ph. D. dissertation.
- [Nii 79] H. Penny Nii and Nelleke Aiello. *AGE: A Knowledge-based Program for Building Knowledge-based Programs*. Technical Report HPP-79-4, Heuristic Programming Project, Computer Science Department, Stanford University, 1979. Also in Proceedings of the 6th International Joint Conference on Artificial Intelligence: 645-655, 1979.
- [Nii 82] H.P. Nii, E.A. Feigenbaum, J. J. Anton, and A. J. Rockmore. *Signal-to-Symbol Transformation: HASPISAP Case Study*. Technical Report HPP-82-6, Heuristic Programming Project, Computer Science Department, Stanford University, 1982. Also in AI Magazine. 3:2, 23-35, 1982.
- [Reddy 77] D. Raj Reddy and Allen Newell. *Multiplicative Speedup of Systems*. In *Perspectives on Computer Science*, pages 183-198, Anita Jones (ed), Academic Press, New York 1977.
- [Selig 87] Lawrence J. Selig. *An Expert System using Numerical Simulation and Optimization to find Particle Beam Line Errors*. Technical

Report KSL-87-36, Heuristic Programming Project, Computer Science Department, Stanford University, 1987.

[Williams 84]

Mark Williams, Harold Brown and Terry Barnes. *TRICERO Design Description*. ESL Inc. 1984.

DISTRIBUTION LIST

addresses	number of copies
RL/COE ATTN: Northrup Fowler III Griffiss AFB NY 13441-5700	5
Knowledge Systems Laboratory Stanford University Attn: Mr. E. Feigenbaum 701 Welch Rd, Bldg C Palo Alto CA 94304	5
PL/DOVL Technical Library Griffiss AFB NY 13441-5700	1
Administrator Defense Technical Info Center DTIC-FDAC Cameron Station Building 5 Alexandria VA 22304-6145	2
Defense Advanced Research Projects Agency 1400 Wilson Blvd Arlington VA 22209-2308	2
HQ USAF/SCTT Washington DC 20330-5190	1
SAF/AQSC Pentagon Rm 4D 262 Wash DC 20330	1
Naval Warfare Assessment Center GIDEF Operations Center/Code 303 ATTN: E Richards Corona CA 91720	1

HQ AFSC/XTH
Andrews AFB MD 20334-5000

1

HQ SAC/SCPT
OFFUTT AFB NE 68040

2

HQ TAC/DRIY
ATTN: Maj. Divine
Langley AFB VA 23605-5575

1

HQ TAC/DCA
Langley AFB VA 23665-5554

1

WRDC/AAAI-4
Wright-Patterson AFB OH 45433-6543

1

WRDC/AAAI-2
ATTN: Mr Franklin Hutson
WPAFB OH 45433-6543

1

AFIT/LDEE
Building 642, Area B
Wright-Patterson AFB OH 45433-6543

1

WRDC/MTFL
Wright-Patterson AFB OH 45433

1

44HPL/HF Wright-Patterson AFB OH 45433-6373	1
AUL/LSE Bldg 1415 Maxwell AFB AL 36112-5534	1
HQ AFSPACCOM/XPA STINFO Officer ATTN: Dr. W. P. Matoush Peterson AFB CO 80914-5001	1
HQ ATC/TTDI ATTN: Lt Col Killian Randolph AFB TX 78150-5001	1
AFLMC/LGY ATTN: Maj. Shaffer Building 215 Gunter AFS AL 36114-6693	1
US Army Strategic Def CSSD-IM-PA PO Box 1503 Huntsville AL 35807-3801	1
Ofc of the Chief of Naval Operation ATTN: William J. Cook Navy Electromagnetic Spectrum Mgt Room 5A573, Pentagon (OP-041) Wash DC 20355	1
Commanding Officer Naval Avionics Center Library 0745 Indianapolis IN 46219-3110	1
Commanding Officer Naval Ocean Systems Center Technical Library Code 7442 San Diego CA 92152-5001	1

Cntr Naval Weapons Center Technical Library/03431 China Lake CA 93555-5011	1
Superintendent Code 524 Naval Postgraduate School Monterey CA 93943-5013	1
Space & Naval Warfare Systems Comm Washington DC 20363-5110	1
CDR, U.S. Army Missile Command Redstone Scientific Info Center AMSRI-3D-CS-R/ILL Documents Redstone Arsenal AL 35899-5241	2
Advisory Group on Electron Devices 201 Varick Street, Rm 1140 New York NY 10014	2
Los Alamos National Laboratory Report Library MS 5000 Los Alamos NM 87544	1
AEDC Library Tech Files/MS-100 Arnold AFB TX 77130	1
Commander, USAG ASDH-FOA-CPL/Tech Lib Bldg 21-01 Ft Huachuca AZ 85613-6000	1
1839 SIG/217 Keesler AFB MS 39574-6342	1

AFEWC/ESRI
San Antonio TX 78243-5000

3

ESD/XRR
Hanscom AFB MA 01731-5000

1

ESD/SZM
Hanscom AFB MA 01731-5000

1

SEI JPD
ATTN: Major Charles J. Ryan
Carnegie Mellon University
Pittsburgh PA 15213-3890

1

Director NSA/CSS
T5122/TOL
ATTN: D W Marjarum
Fort Meade MD 20755-6000

1

Director NSA/CSS
W157
9800 Savage Road
Fort Meade MD 20755-6000

1

NSA
ATTN: D. Alley
Div X911
9800 Savage Road
Ft Meade MD 20755-6000

1

Director
NSA/CSS
W11 DAFSMAC
ATTN: Mr. Mark E. Clesn
Fort George G. Meade MD 20755-6000

1

000 031 9300 Savage Road Ft. Meade MD 20755-6000	1
DIRNSA 2509 9300 Savage Road Ft Meade MD 20775	1
Director NSA/CSS R38 Fort George G. Meade MD 20755-6000	1
Rutgers University Department of Computer Science Attn: Dr Saul Amarel Busch Campus New Brunswick NJ 08903	1
USC-ISI Attn: Mr Yigal Arens 4676 Admiralty Way Marina Del Ray, CA 90292	1
University of Southern California Info Sciences Institute Attn: Dr Robert M. Balzer 4676 Admiralty Way Marina del Rey CA 90292-6695	1
TI Central Research Laboratories Computer Science Lab Attn: Dr Roger Bate, Director P.O. Box 226015, MS 238 Dallas TX 75266	1
Northwestern University Institute for Learning Sciences Attn: Mr Lawrence Birnbaum 1890 Maple Avenue Evanston IL 60201	1
ISX Attn: Mr Bruce Bullock 501 Marin Street, #214 Thousand Oaks CA 91360	1

SEN Laboratories, Inc. Attn: Dr Mark Burstein 10 Mouton Street Cambridge, MA 02238	1
Rockwell Int'l Science Center Attn: Mr John Breese 444 High Street Palo Alto CA 94301	1
General Electric Company Corporate Research and Development Attn: Dr Piero P. Bonissone 1 River Road, Building 37-567 Schenectady NY 12345	1
Carnegie-Mellon University Computer Science Department Attn: Dr Jaime Carbonell Schenley Park Pittsburgh PA 15213	1
Ohio State University Dept of Computer & Info Sciences Attn: Dr B. Chandrasekaran 2036 Neil Avenue Columbus OH 43210	1
Brown University Attn: Dr Eugene Charniak Box 1910 Providence RI 02912	1
Harvard University Allen Computation Lab Attn: Mr Thomas E. Cneatham 33 Oxford Street Cambridge, MA 02138	1
Lowestern University Institute for the Learning Sciences Attn: Mr Gregory Collins 1890 Maple Avenue Evanston IL 60201	1
University of Massachusetts Computer & Info Science Dept Attn: Mr Paul Cohen Amherst Massachusetts 01003	1

Oregon State University
Dept of Computer Science
Attn: Dr Bruce D'Ambrosio
Corvallis OR 97331-4602

1

MIT AI Lab
Attn: Dr Randall Davis
Room NE43-801A
545 Technology Square
Cambridge MA 02139-1936

1

Brown University
Computer Science Department
Attn: Mr Tom Dean
Box 1010
Providence RI 02912

1

NASA Ames Research
Attn: Mr Mark Drummond
Mailstop 244-17
Moffett Field CA 94035

1

ISX
Attn: Mr Gary Edwards
501 Marin, Suite 214
Thousand Oaks CA 91360

1

Tecknowledge, Inc
Attn: Dr Lee Erman
1850 Embarcadero
Palo Alto CA 94301

1

Oakridge National Lab
Attn: Mr Robert Edwards
P.O. Box 2008
Building 4500 North, Mailstop 6207
Oakridge TN 37831-6207

1

DARPA/TTO
Attn: Mr John N. Entzminger
1400 Wilson Boulevard
Arlington VA 22209-2389

1

Stanford University
Knowledge Systems Lab
Attn: Dr Robert Engelmire
701 Welch Road, Bldg C
Palo Alto CA 94304

1

Jet Propulsion Lab 1
Attn: Mr James Firby
MS 301-440
4800 Oak Grove Drive
Pasadena CA 91109

Carnegie-Mellon University 1
The Robotics Institute
Attn: Mr Mark Fox
5000 Forbes Ave
Pittsburgh PA 15213

GTE Labs 1
Attn: Mr W. A. Frowley
40 Sylvan Road
Waltham MA 02254

SPI International 1
AI Center
Attn: Dr Thomas D. Garvey
333 Ravenswood Avenue
Menlo Park CA 94025-3493

Stanford University 1
Attn: Dr Michael R. Genesereth
Heuristic Programming Project
701 Welch Road, Building C
Palo Alto CA 94304

Stanford University 1
Computer Science Department
Attn: Dr Matt Ginsberg
Stanford CA 94305

Kestrel Institute 1
Attn: Dr. Cordell Green
1801 Page Mill Road
Palo Alto CA 94304

University of Chicago 1
Computer Science Dept/RY 155
Attn: Kris Hammond
1100 E 58th Street
Chicago Illinois 60637

Stanford University 1
Knowledge Systems Lab
Attn: Ms Barbara Hayes-Roth
701 Welch Road Building C
Palo Alto CA 94304

Teknowledge, Inc. 1
Attn: Dr Frederic Hayes-Roth
1350 Embarcadero
Palo Alto CA 94301

Cornell University 1
Computer Science Department
Attn: Dr John Hopcroft
Uson Hall
Ithaca NY 14853

Knowledge Systems Lab 1
Medical Computer Science Group
Attn: Mr Eric Horvitz
MSCB X215
Stanford CA 94305-5479

Inference Corporation 1
Attn: Dr Philip Klahr
5300 West Century Boulevard
Los Angeles CA 90045

UCLA 1
Attn: Dr Richard Korf
Computer Science Department
Los Angeles CA 90024

BBN Systems and Technologies, Corp 1
Attn: Mr Ted Kral
4015 Hancock Street, Suite 101
San Diego CA 92110

George Mason University 1
Info Technology & Engineering
Attn: Dr Paul Lehner
4400 University Drive
Fairfax VA 22033

ADS 1
Attn: Mr Ted Linden
1500 Plymouth St
Mt. View CA 94043

Duke University 1
Computer Science Department
Attn: Dr Donald W. Loveland
Durham NC 27706

University of Chicago 1
Computer Science Dept. RY 155
Attn: Mr Charles Martin
1100 E 58th Street
Chicago IL 60637

Yale University 1
Dept of Computer Science
Attn: Mr Drew McDermott
51 Prospect Street
New Haven CT 06520

Oakridge National Lab 1
Attn: Mr Bob McLaren
P.O. Box 2008
Building 6011, Mailstop 6370
Oakridge TN 37831-6370

ONR/Code 1133 IS 1
Attn: Mr Alan Meyrowitz
800 North Quincy St
Arlington VA 22217

Intellicorp 1
Attn: Mr Paul Morris
1975 El Camino Real West
Mountain View CA 94040

Stanford University 1
Attn: Dr H. Penny Nii
Heuristic Programming Project
701 Welch Road, Building C
Palo Alto CA 94304

Stanford University 1
Computer Science Department
Attn: Dr Nils J. Nilsson
Margaret Jacks Hall
Stanford CA 94305

University of Chicago 1
Computer Science Dept RY 155
Attn: Chris Owens
1100 E 58th Street
Chicago IL 60637

Babson College 1
Math Department
Attn: Dr Gordon D. Prichett
Babson Park MA 02157-0201

MIT AI Lab Attn: Dr Charles Rich Room NE43-839 545 Technology Square Cambridge MA 02139-1930	1
Rolt, Heranek, and Newman, Inc. Department of AI Attn: Mr R. Bruce Roberts 10 Moulton Street Cambridge MA 02238	1
Telens Research Attn: Mr Stanley J. Rosenthal 576 Middlefield Rd Palo Alto CA 94301	1
Honeywell Systems & Research Center Attn: Mr Robert Schrag MN 65-2100 3660 Technology Drive Minneapolis MN 55418	1
SUNY at Buffalo Computer Science Department Attn: Dr Stuart C. Shapiro 226 Bell Hall Buffalo NY 14260	1
NRL Attn: Dr Randall Shumaker Code 5510 4555 Overlook Ave, SW Washington DC 20375-5000	1
Robotics Institute CMU Attn: Mr Stephen F. Smith Schenley Park Pittsburgh PA 15213	1
FMC Corporation ATTN: N. S. Sridharan CTC 1205 Coleman Ave. Santa Clara CA 95052	1
Carnegie Mellon University Robotics Institute School of Computer Science Attn: Ms Katia Sycara Pittsburgh PA 15213	1

AI Applications Institute Attn: Mr Austin Tate 80 Southbridge Edinburgh, EH1 1HA United Kingdom	1
Oakridge National Lab Attn: Mr Bruce Tonn P.O. Box 2003 Building 4500 North, Mailstop 6207 Oakridge TN 37331-6207	1
Lockheed AI Center, 90-36/259 Lockheed 330 Division Attn: Dr Steven Vere 3251 Hanover St. Palo Alto CA 94304-1187	1
ARM Systems & Technologies Corp Attn: Mr E. Walker 10 Moulton Street Cambridge MA 02238	1
Texas Instruments, Inc. AI Lab Attn: Raj Wall P.O. Box 655474, M/S 238 Dallas TX 75265	1
WRDC/TXI Attn: Mr Michael P. Wellman Bldg 22, Room 5-109 WPAFB OH 45433	1
SPI International Attn: Mr David Wilkins 333 Ravenswood, EJ 227 Menlo Park CA 94025	1
MIT AI Lab Attn: Dr Patrick Winston Room 4347-17 545 Technology Square Cambridge, MA 02139-1766	1
DAAP4/ISTO Attn: Lt Col Stephen L. Cross 1400 Wilson Blvd Arlington VA 22202-2321	1

The MITRE Corporation 1
Attn: Mr David Day
Burlington Road
Bedford MA 01730

Mr Joseph Fiksel 1
P.O. Box 13112
1350 Embarcadero Rd
Palo Alto CA 94303

Australian AI Institute 1
Attn: Mr Michael Georgeff
1 Grattan St
Carlton, Victoria 3053
Australia

The MITRE Corporation 1
Attn: Ms Phyllis Koton
Burlington Rd, MS A045
Bedford MA 01730

The MITRE Corporation 1
Attn: Mr Mark Nadel
Burlington Rd, MS A047
Bedford MA 01730

Rutgers University 1
Dept of Computer Science
Hill Center, Busch Campus
Attn: Mr Charles Schmidt
New Brunswick NJ 08903

AFOSR/NM 1
Attn: Prof Abraham Waksman
Rolling AFB DC 20332-6443

University of Rochester 2
Chairman, Dept of Comp Science
Attn: Prof James F. Allen
Computer Studies Building
Rochester NY 14627

Clarkson University 2
Attn: Dr Susan E. Conry
Elect & Computer Eng'g Dept
Potsdam NY 13676-1401

University of Massachusetts COINS Department Attn: Dr Victor R. Lesser Lederle Graduate Research Center Amherst MA 01003-0001	2
NASA Ames Research Center Attn: Dr Peter E. Friedland MS 24 4-17 Code R14 Moffett Field CA 94035-1009	1
SPI International Attn: Ms Marie A. Hienkowski 333 Ravenswood Ave, EK377 Menlo Park CA 94025	1
Honeywell Systems & Research Center Attn: Mr Mark S. Boddy 3660 Technology Drive Minneapolis MN 55413	1
ESD/AV Brown Building Hanscom AFB MA 01731-5000	1
Laboratory for Computer Science Attn: Jon Doyle Massachusetts Institute of Tech 545 Technology Square Cambridge MA 02139	1
UNISYS Attn: Mr Timothy J. Finin Ctr for Adv Info Technology 70 East Swedesford Rd Malvern PA 19355	1
ISX Corporation Attn: Mr Scott Fouse 501 Marin Street Suite 214 Thousand Oaks CA 91320	1
University of Massachusetts Dept of Comp & Info Science Attn: Ms Edwina Hissland Amherst MA 01003	1

The Rand Corporation 1
Attn: Mr Jeff Pothenberg
1733 Main Street
Santa Monica CA 90406-2100

ISX Corporation 1
Attn: Mr Allen G. Smith
501 Marin Street, Suite 214
Thousand Oaks CA 91320

Kestrel Institute 1
Attn: Mr Douglas Smith
3160 Hillview Ave
Palo Alto CA 94304

Rockwell International 1
Attn: Mr David E. Smith
444 High Street
Palo Alto CA 94301

University of Rochester 1
Attn: Mr Josh Tenenber
Computer Science Dept
Wilson Blvd
Rochester NY 14627

Stanford University 1
Attn: Gjo Wiederhold
Dept of Computer Science
433 Margaret Jacks Hall
Stanford CA 94305-2140

Director 1
DARPA/ISTO
1400 Wilson Blvd
Arlington VA 22202-2300

WRDC/TXI 1
Attn: Mr William Baker
WPAFB OH 45333

U.S. Army Research Office 1
Attn: Dr David Hixson
P.O. Box 1211
Research Triangle NC 27709-2211

U.S. Army Ballistic Resch Lab
Attn: SLCPR-37-C (W. A. Hirschberg)
Aberdeen Proving Ground MD
21005-5366

1

Telos Research
Attn: Mr David Chapman
572 Middlefield Road
Palo Alto CA 94301

1

University of Michigan
Attn: Mr Edmund H. Durfee
Dept of Elect Eng & CS
1101 Beal Ave
Ann Arbor MI 48109

1

MITRE Corporation
Attn: Chris Elsaesser
7525 Colshire Drive
McLean VA 22102-3431

1

University of Maryland
Attn: Mr James A. Hendler
Computer Science Dept
UMCP
College Park MD 20742

1

University of Washington
Attn: Mr Steven J. Hanks
Dept of Computer Science & Eng
FR-35
Seattle WA 98195

1

Anderson Consulting
Attn: Mr Bruce Johnson
100 South Wacker Drive
Chicago IL 60606

1

Teleos Research
Attn: Leslie P. Kaelbling
576 Middlefield Road
Palo Alto CA 94301

1

NASA Ames Research Center
Attn: Pat Langley
M/S 244-17
Moffett Field CA 94035

1

Lockheed P30 Info and CS 1
Attn: Mr William S. Mark
3251 Hanover Street
95-C1 Building 259
Palo Alto CA 94304-1191

Carnegie Mellon University 1
Attn: Raj Reddy
School of Computer Science
Pittsburgh PA 15213

Copernican Group 1
Attn: Mr Earl D. Sacerdoti
737 Melville Ave
Palo Alto CA 94301

MITRE Corporation 1
Attn: Mr Allen Sears
7525 Colshire Drive
M/S 2289
McLean VA 22102

Cornell University 1
Attn: Mr Alberto M. Segre
Dept of Computer Science
Ithaca NY 14853-7501

HCP Corporation 1
Attn: Mr. Robert L. Simson, Jr.
500 Tech Parkway
Atlanta GA 30313

**MISSION
OF
ROME LABORATORY**

Rome Laboratory plans and executes an interdisciplinary program in research, development, test, and technology transition in support of Air Force Command, Control, Communications and Intelligence (C³I) activities for all Air Force platforms. It also executes selected acquisition programs in several areas of expertise. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C³I systems. In addition, Rome Laboratory's technology supports other AFSC Product Divisions, the Air Force user community, and other DOD and non-DOD agencies. Rome Laboratory maintains technical competence and research programs in areas including, but not limited to, communications, command and control, battle management, intelligence information processing, computational sciences and software producibility, wide area surveillance/sensors, signal processing, solid state sciences, photonics, electromagnetic technology, superconductivity, and electronic reliability/maintainability and testability.