The Elint Application on Poligon: The Architecture and Performance of a Concurrent Blackboard System

James Rice*
Knowledge Systems Laboratory
Stanford University
701 Welch Road, Bldg. C,
Palo Alto, California 94304

Abstract

In this paper we discuss Poligon, a skeletal system for the development of concurrent blackboard based applications, its architecture and the motivation for its design. A number of experiments have been performed in order to evaluate the performance of Poligon. Some of these are detailed and the results are shown. Lessons learned in the development of Poligon are given and conclusions about the performance of similar systems are drawn.

1. Introduction

It is often said that future AI applications will make significantly greater computational demands than the present generation. The Advanced Architectures Project of Stanford University's Heuristic Programming Project [Rice, 1988b] is investigating this issue, since it has as its objective achieving computational speed-up for expert systems through the use of parallel hardware and new, advanced software architectures. This requires the development of everything from designs for parallel hardware, which might be appropriate for the execution of future symbolic programs, through operating system and language concepts to problem-solving frameworks and eventually mounting applications on them in order to test the new designs.

Poligon [Rice, 1986] is one of the problem-solving frameworks developed as part of the Advanced Architectures Project. In Section 2, we discuss Poligon's architecture as a design for a high-performance, concurrent blackboard system aimed particularly at the problem domain of soft real-time problems, and what motivated this design. Section 3 discusses the applications mounted on the Poligon framework and experiments performed on the Poligon system to measure its performance. Section 4 presents the results of these experiments and an interpretation of them. We conclude in Section 5 with a number of the lessons we have learned in the process and pointers for future research.

The author gratefully acknowledges the support of the following funding agencies for this project; DARPA/RADC, under contract F30602-85-C-0012; NASA, under contract number NCC 2-220; Boeing Computer Services, under con tract number W-266875.

2. The Poligon Architecture

In this section we briefly discuss the architecture of the Poligon system. A more detailed description of the design rational for Poligon can be found in [Nii et al., 1988]. Because of space constraints, it will be assumed that the reader is conversant with the terminology of Blackboard Systems [Engclmore and Morgan, 1988], though no deep knowledge will be assumed.

When we started the Advanced Architectures Project we had a hunch that the Blackboard problem-solving architecture might offer a basis for the efficient exploitation of concurrent hardware. This was because the blackboard model appeared to have concurrency built into it. Why this is, in fact, not the case is explained in [Rice, 1988a]. The primary reasons why the blackboard model of a collection of simultaneously cooperating experts cannot develop the parallelism that one might expect is that the blackboard model itself assumes effectively infinite bandwidth with which the experts can see any part of the blackboard that might be of interest. It also assumes that experts do not get into one another's way whilst solving the problem. In practice a knowledge source can only see a small segment of the blackboard at any one time without degrading the performance of the system unacceptably. Similarly, the experts are dependent on one another, they must often wait for the results deduced by other agents and can be confused by updates being posted at unexpected times or in surprising orders. We are, however, unaware of a better architecture for concurrent problem-solving than that of Blackboard systems.

Although a number of other research efforts have looked at concurrent blackboard systems, these have concentrated primarily on either the aspects of distributed, concurrent problem-solving, such as [Lesser and Corkill, 1983] or on coarse grained parallel systems, such as [Fcnnell and Lesser, 1977, Aiello, 1986, Ensor and Gabbe, 1985]. Poligon is a finer grained system than these, directed particularly at gaining speed-up through parallel execution.

The normal, serial implementations of the blackboard metaphor use a scheduling mechanism to cause one rule to fire after another. In parallel systems it is crucial that the programmer eliminate serial components, since this limits speed-up.[1] The main motivation of the Poligon system was to find a way to eliminate the serializing aspects of the blackboard model. We viewed this as doing the following:

- Eliminating the scheduling mechanism and finding ways to support concurrent rule activation all across the blackboard.
- Optimizing the design for distributed-memory, message-passing hardware, which should be able to deliver the best performance for large numbers of processors (of the order of hundreds to thousands.)
- Distributing the knowledge base over the blackboard so that there would be no serialization in the access to the blackboard from the executing knowledge.
- Designing the system so as to allow it to be highly compilable. It was clear from the outset that a considerable portion of the expense of existing AI systems is due to the fact that they are optimized for easy modification and debugging, rather than high run-time performance. The resulting system, therefore, had to be designed so as to be able to be compiled efficiently yet still be intelligible and debuggable during the development cycle.

As these ideas progressed we developed the notion of a blackboard consisting of active nodes, tightly associated with the knowledge relevant to them.

A very simple scheme was developed for invoking the knowledge that had been distributed to the blackboard nodes: rules arc activated as daemons as a result of modifications to the slots of a node (see Figure 1).

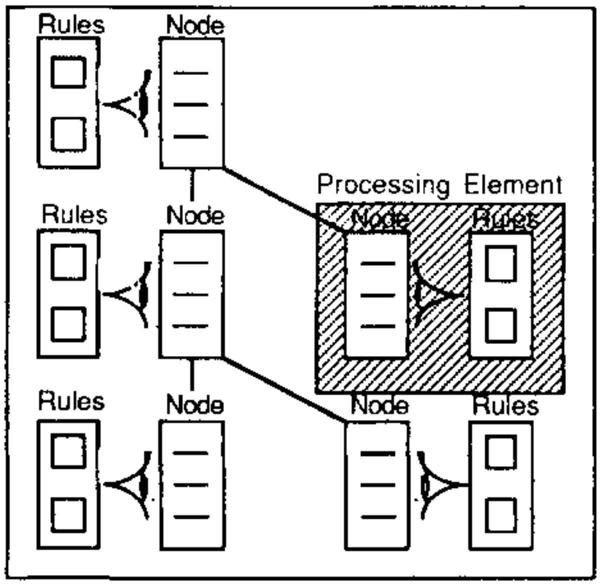

Figure 1. The Organization of the Poligon Blackboard. Rules are distributed over the network of processors and are attached to the blackboard nodes so that they can watch for modifications made to the slots in which they are interested.

The distributed-memory hardware model, on which the Poligon system was to operate had the property that each processor was effectively a uniprocessor system. This meant that if we viewed the blackboard with a *"Node as a Process/Processor"* model then we would lose potential parallelism due to being able to execute only one piece of code (rule) at a time for any given node.

What we needed, therefore, was a mechanism to allow the activation of multiple rules for any given blackboard node. This caused us to develop a model of Poligon which was as follows: *"A blackboard node is a process on a processor, surrounded by a collection of processors able to service its requests to execute rules."* It can easily be seen that this model is very close to a distributed object system model. This is by no means a coincidence. The underlying hardware system on which Poligon was implemented was a concurrent, distributed object oriented system [Delagi et al., 1986].

The model expressed above is not without problems. In order to minimize the probability of a node being locked for a long period, which would delay remote access to it, as much processing is done in the remote rule invocations as possible[2]. This means that, when the rules execute, they have to do so in the context of a snap-shot of the solution state as it was when the rule was invoked (see Figure 2). Remote reads to other nodes, even the invoking node, are expensive and one cannot guarantee that things haven't changed by the time that the result of the read has been returned.

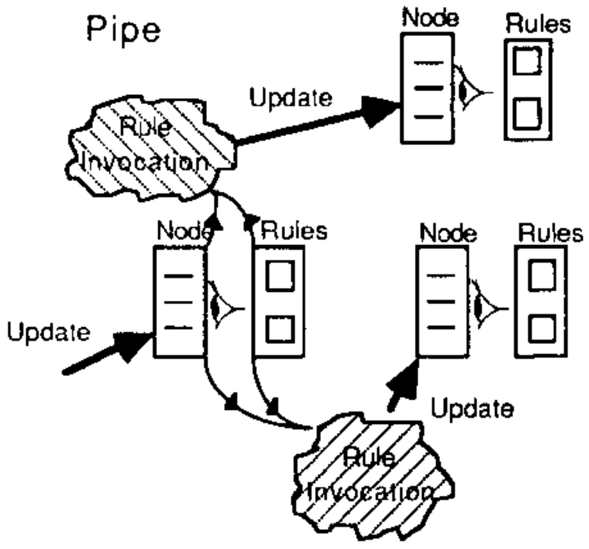

Figure 2. Updates to Poligon nodes cause concurrent rule execution, which themselves cause further updates. This implicitly forms pipes on the blackboard as data flows up or down the abstraction hierarchy.

[1] Speed-up can be viewed as the ratio of the system's speed using *N* processors to its speed using only one.

[2]There are no user accessible locks in Poligon. Poligon nodes become locked (enter critical sections) during slot reads and updates, which are cheap operations. The Poligon architecture is such that deadlock will not happen as a result of system action, though the user can still write a program that will live-lock, e.g. two nodes each waiting for one node to update the other will wait for ever.

This led to the development of the idea of a Poligon node as being an agent capable of evaluating its own performance. Mechanisms had to be included so as to allow the system to be able to assess any request to modify its local state and to decide whether to perform the update, or what else to do instead, on the basis of its own view of how it is progressing towards its goal of solving the problem.

3. Experiments on Poligon

In this section we briefly describe the experiments performed on the Poligon system to date. Two applications have been mounted on Poligon: *Elint,* a soft real-time situation assessment problem and *ParAble*, a diagnostic application for particle accelerator beam-lines [Selig, 19871. The experiments with the Elint application have now been completed, whereas those on the ParAble system are in their infancy, so only the Elint application will be considered here. A more detailed treatment of the Elint experiments can be found in [Niietal., 1988].

The Elint application encodes knowledge used to interpret the radar emissions made by planes that are received by ground-based tracking stations distributed across the country. Because these tracking sites are passive devices, they can only detect the bearing and spectral characteristics of the radar emissions. Between them, it is their responsibility to deduce a position, course, identity and intention for any aircraft traveling through the monitored airspace. The Elint application simulates a central machine that integrates reports from these detection sites in order to achieve the overall goals mentioned.

The important characteristics of the Elint problem were:
- A continuous stream of input data.
- No *a priori* knowledge of the behavior or number of the aircraft being tracked.
- The need to emit periodic reports capturing the system's evolving view of the solution.

The Elint problem was chosen both because it was nontrivial and was in a class of problems, for which blackboard systems had already been used, and also because it was hoped that parallelism would be readily available. It was anticipated that parallelism could be extracted from the concurrent execution of knowledge on any given part of the solution space and from the potentially large number of independent elements in that solution space, i.e. aircraft.

The application was taken from a serial implementation and was not restructured so as to be better suited for parallel execution. The blackboard was, however, composed of three distinct layers in the abstraction hierarchy. Data flowing from one level to the next allowed pipes to be formed that were three stages long.

Perhaps the most important lesson that we learned from performing these experiments was to find a way to measure the relative performance of concurrent real-time systems. The best way that we found to do this was to pump data into the system at a given rate, which was under the control of the user, and examine the system's output over time. There is a measurable time between the time that data comes into the system and the time that any associated reports come out of the system. If this time difference increases on average over the course of a run then the system was not able to keep up with the rate at which data was being pumped into it. The experiment was then performed again with the data rate turned down until the report latency did not increase. This gave us a measure for the system's throughput, which we took to denote its peak performance.

The experiments that were performed were intended to measure a number of different aspects of the system's performance:
- The speed-up that the Poligon system could deliver.
- The peak throughput of the system.
- The ability of the system to exploit large knowledge bases.
- The granularity of the system.

Experiments to measure these are described in Section 4.

4. Experimental Results

The space available for this paper does not allow a full explanation of the experimental results, so the interested reader is again advised to refer to [Nii et al., 1988] for more details. It is hoped that the treatment here will be sufficient to give the gist of what we have learned.

It should be noted here that wherever reference is made to absolute times, these are measured in terms of the performance of the simulated hardware on which the Poligon system runs [Delagi, 1986]. Each processing element of this machine is of about the performance of a TI Explorer™ II+ processor.

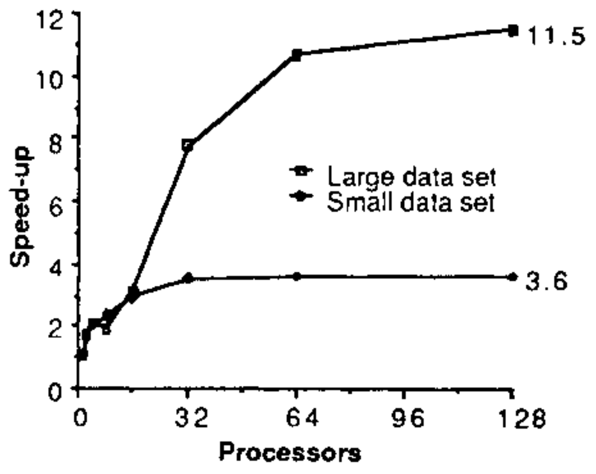

Figure 3. A graph showing the speed-ups derived from the large and small data sets plotted against the number of processors used.

4.1. Measurement of Speed-up and Throughput

In this experiment two different data sets were used. One was designed to allow the Poligon system only to create one pipe in the solution space, the second allows Poligon to

Explorer is a trade mark of Texas Instruments Corporation.

create four pipe-lines; it was four times as dense[1]. The combination of these two results allows us to do the following:

- Measure the peak throughput for the larger data set.
- Determine the contribution to speed-up due simply to pipe-line parallelism.
- Compare the results from the two data sets so as to be able to get a measure of the ability of the system to exploit parallelism in the source data, i.e. data parallelism.

The results from the two data sets are shown in Figure 3.

In this experiment we learned the following:

- The peak speed-up shown in this application due to pipe-line parallelism was 3.6. This showed that although the length of the pipe was three, speed-up was greater than three because of the concurrent execution of rules by the different stages of the pipe.
- The peak throughputs measured from the two data sets were not significantly different. This indicates that Poligon was able to achieve an almost linear increase in speed-up as the problem size of the data set increased, an important result.
- The peak throughput for the system as measured from the larger data set was about 340us per signal data record. Because of the linear increase in performance with data set complexity it is assumed that with more complex problems higher performance could be achieved. By comparison the Elint application, when coded to run in the AGE blackboard system took about 3.7 seconds to process each piece of signal data.

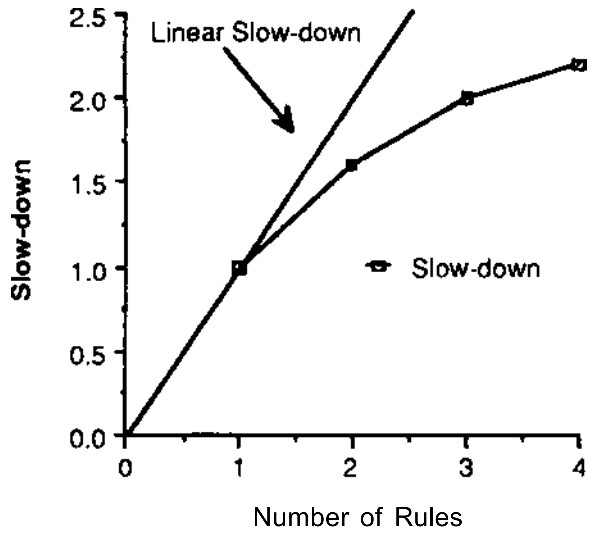

Figure 4. A graph showing application throughput slow-down plotted against the number of rules being fired for each rule-invoking event

4.2. Measurement of Poligon's Ability to Exploit Large Knowledge Bases

In this experiment the Poligon system was tested using the small data set used above. The Poligon framework was modified so that, whenever a rule was invoked, *N* rules would be invoked, rather than just one. *N -1* of these rules had the special characteristic that they performed almost all of the processing required except for any blackboard modifying updates. This gave a measure of the system load if the knowledge base was *N* times larger, whilst still giving the right behavior for this application.

The results from this experiment are shown in Figure 4.

In this experiment, if the system were able to exploit parallelism in the knowledge base to the full, one would expect that the system would not slow down at all as new knowledge was added, i.e. the line shown in Figure 4 would be horizontal. If, on the other hand, the system bogged down completely as more knowledge was added one would expect that the result would be worse than linear slow-down, that is the plot would appear above the "linear slow-down" line. As can be seen easily from the graph, Poligon's performance was better than linear. In order to perform four times as much work it took only 2.2 times as long. This means that, as long as there are sufficient computational resources, the Poligon system delivers good performance for a knowledge base whose size is at least up to four times that of the Elint applications

4.3. Measurement of the Granularity of Poligon's Rules

In this experiment some of the internal mechanisms within Poligon were timed in order to get some empirical measure of the granularity of the system.

Within a blackboard system a number of mechanisms are of crucial importance to the performance of the system. Amongst these arc slot reads, slot writes and rule invocation.[3]

In order to determine the costs of these operations they were performed repeatedly in a manner which allowed the individual costs to be measured with some precision.

The results of these experiments are as follows. It should be noted that all of these results neglect any communication overhead, so they are only representative for local operations.

- Slot reads take 1.36 + 0.94N Us, where *n* is the number of slots being read at once, Poligon supports a form of multiple slot read operation.
- Slot updates lake 18 + *53.In u*s, where *n* is the number of slots being written. Poligon allows arbitrary user code to be executed during the slot update operation, so this is a representative figure taken from the Elint application. This is for the case of no rules being associated with the slots being updated.

[1]The small data set can be thought of as representing only one aircraft, the second had four. The data was, therefore, no more *complex,* there was just more of it.

[2]In AGE, the Elint knowledge base was composed of about twenty knowledge sources, each having about three rules.

[3]Node creation is another important aspect, which was not measured in this experiment.

- The overhead cost of starting up a rule's execution is about 1ms per rule invoked.

A substantial part of the time taken performing these operations could be optimized considerably. For instance, a figure of about 500us for rule invocation could relatively easily be achieved in a real system and more than this improvement could be expected for a system which allowed specialized microcode or similar efficiency tuning. This shows that there is a lower bound to the granularity that the user can expect to achieve. For computations taking less than a few milliseconds it may not be worth starting up a rule to perform the computation, the cost of parallel execution would be in excess of the serial execution time.

5. What We have Learned

We have learned a number of lessons from this project, some of which were counter to our intuitions.

- Our intuition told us that programming a concurrent blackboard system would not be too hard because of the assumed implicit asynchrony in serial blackboard systems. We found this not to be the case. We found the programming task to be difficult and, we believe, a reconccptualization of existing problems will be required in order to port them for efficient parallel execution. The difficulty of implementation of applications is due largely to the divergence of implementations of serial blackboard systems from the pure blackboard model in order to make implementation and programming more manageable as was mentioned in Section 2 and is covered more thoroughly in [Rice 88a].
- We found that the Poligon system and architecture itself performed fairly well. Although programming the system was not trivial, the Poligon system provided a useful abstraction model that allowed the development of an application in a blackboard-like manner that still gave the correct answers and acceptable performance.
- We had thought that parallelism in the knowledge base would be crucial to the achievement of high performance. In the applications that we used, knowledge proved to be sparse and the pipe-line parallelism that resulted from it delivered only a factor of three in speed-up. The small amount of speed-up from pipe-line parallelism was due to the short length of the pipes, the lack of applicable knowledge and the difficulty in balancing the pipes. Most of the parallelism seen in the applications implemented in the Advanced.Architectures Project was derived from the data, not the code. The limit to the length of the pipes derived from the application was not one that resulted from the structure of the problem itself, but rather came from the fact that the application was reimplcmcntcd for Poligon from the AGE implementation, not reformulated.
- When we started the project our intuition told us that the significantly greater cost of communication relative to computation would bias the programmer in favor of doing as much as possible locally before a message was sent. It turned out that doing this increased the granularity of the system and restricted parallelism. We found that, although communication is expensive, as long as data keeps flowing along a pipe the price that is payed is in latency, not in speed-up. The fact that processes are not held up by communication is a result of the non-blocking message sending ability of the hardware. Thus, fine-grained systems are likely to be significant for achieving good performance from large multiprocessors but the increased latency due to distributing the work could have an adverse effect on real-time performance.
- We learned that simulation of multiprocessors is expensive. A number of projects are interested simply in the difficulties caused by the asynchronous behavior of concurrent systems. Such projects are able to use a simple model for their implementations on existing hardware. We, on the other hand, wanted to measure the performance of our software on the hardware we were developing precisely in order to refine both our hardware and software designs. This is computationally a very expensive task and has proved to be a major limiting factor on the work that we have done. Having said this, however, it should be noted that we arc confident that we have achieved better results and have gained deeper insights than we would have done if we had concentrated on building real hardware.
- Resource allocation was found to be a significant factor in delivering high performance. The fact that blackboard nodes are often very long-lived means that an even load balance can easily be disrupted by a few busy nodes. In the experiments reported here the allocation of processes to processors was done randomly. Other experiments in the Advanced Architectures project have shown that, compared to the ideal, perfect load balanced state[1], even with very careful site allocation the Elint application lost 30% in efficiency and delivered 30% less speed-up than in the perfectly load balanced case. This could not be recovered through the use of more processors |Dclagi and Saraiya, 1988].

6. Conclusions

In this paper we have described Poligon, a blackboard framework designed to operate on distributed-memory multiprocessors. We have described experiments performed on it, shown the results and discussed the conclusions that can be drawn from them and mentioned some lessons that were learned along the way.

We have shown that the Poligon system can deliver a speed-up for the Elint application of nearly twelve, with near linear speed-up gain with increasing problem complexity. We have also shown significantly better than linear slowdown as a result of increasing knowledge base complexity. We are confident, therefore, that given a larger problem Poligon could deliver significantly more speed-up than this.

From our work we can conclude that data parallelism is likely to be the most important source of parallelism in the foreseeable future, at least until truly huge knowledge bases are developed. This requires that concurrent problem-solving systems should be not only able to exploit data parallelism but be able to do so in a manner which allows the rapid development, easy maintenance and modification of knowl-

[1]'This was possible to measure because of being able to "cheat" in the processor allocation for the simulator, assuming global knowledge.

edge bases and encourages the development of software that is not brittle when knowledge is added or removed or when the system meets circumstances that were not anticipated by the programmer. Poligon is a possible first step in this direction.

References

[Aiello, 19861 Nelleke Aiello. User-Directed Control of Parallelism: The CAGE System. Technical Report KSL-86-31, Knowledge Systems Laboratory, Computer Science Department, Stanford University, April 1986.

[Delagi, 1986] Bruce Delagi. CARE Users Manual. Technical Report KSL-86-36, Knowledge Systems Laboratory, Computer Science Department, Stanford University, 1986.

[Delagi et al., 1986] Bruce A Dclagi, Nakul P. Saraiya, Gregory T. Byrd. LAMINA: CARE Applications Interface. In *Proceedings of Third International Conference on Super computing,* pages 12-21, Boston, MA, March 1988 International Supcrcomputing Institute. Also Technical Report KSL-86-76, Knowledge Systems Laboratory, Computer Science Department, Stanford University, 1986.

[Dclagi and Saraiya, 1988] Bruce A. Dclagi and Nakul P. Saraiya. ELINT in LAMINA: Application of a Concurrent Object Language. In SIGPLAN Notices, February 1989, ACM. Also Technical Report KSL-88-33, Heuristic Programming Project, Computer Science Department, Stanford University, 1988.

[Engclmore and Morgan, 19881 Robert Engclmorc and Tony Morgan (eds.) *Blackboard Systems.* Addison-Wesley Publishing Company Inc., Menlo Park 1988.

[Ensor and Gabbc, 19851 J. Robert Ensor and John D. Gabbe. Transactional Blackboards. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence,* pages 340-344, Los Angeles, California, August 1985. International Joint Committee on Artificial Intelligence

[Fcnnell and Lesser, 1977] Richard D. Fennell and Victor R. Lesser. Parallelism in Al Problem Solving: A case Study of Hearsay-!I. In *IEEE transactions on Computers,* pages 98-111, February, 1977.

[Lesser and Corkill, 1983] Victor R. Lesser and Daniel D. Corkill. The Distributed Vehicle Monitoring Testbcd: A Tools for the Investigation of Distributed Problem Solving Networks. In *The Al Magazine,* pages 15-33, Fall 1983.

[Nii et al., 19881 H. Penny Nii, Nelleke Aiello and James Rice. Experiments on Cage and Poligon: Measuring the Performance of Parallel Blackboard Systems. In *Distributed Artificial Intelligence II.* L. Gasser and M. N. Huhns (eds). Pitman Publishing Ltd. and Morgan Kaufmann, 1989. Also Technical Report KSL-88-66, Knowledge Systems Laboratory, Computer Science Department, Stanford University, October 1988.

[Rice, 1986] James Rice. The Poligon User's Manual. Technical Report KSL-86-10, Heuristic Programming Project, Computer Science Department, Stanford University, 1986.

[Rice, 1988a] James Rice. Problems with Problem-Solving in Parallel: The Poligon System. In *Proceedings of Third International Conference on Super computing,* pages 25-34, Boston, MA, March 1988 International Supcrcomputing Institute. Also Technical Report KSL-88-04, Knowledge Systems Laboratory, Computer Science Department, Stanford University, January 1988.

[Rice, 1988bl James Rice. The Advanced Architectures Project. Technical Report KSL-88-71, Knowledge Systems Laboratory, Computer Science Department, Stanford University, January 1989.

[Selig, 1987] 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.