STIF

1/90

ORIGINAL PAGE IS
OF POOR QUALITY

Final Report on NASA-Ames grant NAG-2-487 Architectures for Reasoning in Parallel

AMES
GRANT
IN-62-CR
256759
37P.

Lawrence O. Hall
Intelligent Systems Laboratory
Department of Computer Science and Engineering
University of South Florida
Tampa, Fl. 33620

1 Summary

The research conducted has dealt with rule-based expert systems. The algorithms that may lead to effective parallelization of them have been investigated. Both the forward and backward chained control paradigms have been investigated in the course of this work. The *best* computer architecture for the developed and investigated algorithms has been researched.

In the Intelligent Systems Lab at the University of South Florida, two experimental vehicles have been developed to facilitate this research. They are Backpac, a parallel backward chained rule-based reasoning system and Datapac, a parallel forward chained rule-based reasoning system. Both systems have been written in Multilisp [26], a version of Lisp which contains the parallel construct, **future**. Applying the **future** function to a function cause the function to become a task parallel to the spawning task. Backpac has also been ported to MUL-T [32], Common Lisp [59], and C. These systems originally run under a simulator developed at MCC [38].

Additionally, Backpac and Datapac have been run on several disparate parallel processors. The machines are an Encore Multimax with 10 processors (and one with 8 processors), the Concert Multiprocessor with 64 processors, and a 32 processor BBN

(NASA-CR-186237) ARCHITECTURES FOR REASONING IN PARALLEL Final Report (University of South Florida) 37 p CSCL 09B

N90-14836

Unclas
G3/62 0256759

GP1000. Both the Concert and the GP1000 are switch-based machines. The Multimax has all its processors hung off a common bus. All are shared memory machines, but have different schemes for sharing the memory and different locales for the shared memory. The main results of our investigations come from experiments on the 10 processor Encore and the Concert with partitions of 32 or less processors. We have some initial results from the BBN, but they are not definitive at this time.

Additionally, experiments have been run with a stripped down version of EMYCIN [57]. The original parallel system called EMY [38], has had its performance in simulations described in the literature. Our work with it has been on true parallel processors. The need for load balancing rules and being judicious in the use of and/or parallelism has become apparent from experiments on the Encore and Multimax.

The real-world knowledge bases that were used were a 64-rule knowledge base which could diagnose the cause of fevers and a 99 rule knowledge base on gem identification. In addition, our group generated a number of *fictional* knowledge bases which contained no real knowledge. Instead they contained large numbers of rules structure in desired formats. The rationale for this was to empirically examine the limits of parallelism in the paradigms.

1.1 Backpac

This system is conceptually very much like EMYCIN. The advantage of it is that the same functionality and spirit does not have to be maintained, as it is not the same. It is also a somewhat simpler model. Questioning the user has been ignored in our studies thus far.

Parallelism can be used in evaluating rules and in evaluating clauses. We found no benefit in evaluating clauses of rules in parallel in general. The biggest benefits from parallelism were in grouping goal rules together and evaluating groups of them as separate tasks. This is especially effective on the Concert, where speed increases as high as 17 on 20 processors were observed. It is not as effective on the Multimax architecture. In fact, clustering goal rules has a negligible effect for a number of generated knowledge bases on the Multimax.

1.2 Datapac

Any forward chained system must be put into the context of the OPS5, etc. [9] systems. These systems make use of the highly efficient RETE [8] pattern matching algorithm and are widely used. They follow the match, recognize and act control cycle. They are truly data-driven, as changes to working memory drive the system. Parallelism in these systems has been investigated by a number of researchers [41, 14, 11, 13, 14, 35]. The initial results were not encouraging showing less than an order of magnitude speed-up to be possible from the use of parallelism. Later results have both empirically and theoretically shown that more than one order of magnitude, but less than approximately 64 times speed-up may be achieved.

It turns out that in these systems match will take up to 90% of the time. The match component becomes the most important to parallelize and it has shown some definite limitations in that respect. Hence, our examination has taken a somewhat different direction. We have concentrated on the structuring of the rules in attempting to parallelize such a system. In a system which chains there is a hierarchy of rules. That is, some rules must be fired before others can be. Datapac makes use of the hierarchy information in pruning the set of possible rules at any given time. There is no explicit synchronization of the rules in the system, as in the OPS match, recognize and act cycle. It is not as efficient at matching patterns as the RETE algorithm and does not currently allow rules to fire multiple times. Working memory is distributed among processors in this model with a global working memory containing all the information derived. Individual rule processors will only have some needed subset of the overall working knowledge.

It is functionally limited in comparison to OPS systems, which are truly rule-based programming languages. Datapac is simply an expert systems *tool* (though it lacks most features that a tool would have). The system has been investigated less deeply than the backward chained systems. Order of magnitude speed increases have been observed on the Concert with approximately 100 rules on family relationships.

1.3 The outlook

Some issues that affect research in this area are the availability of real knowledge bases. Many of them are proprietary. The parallel Lisps are in general not very fast. This is certainly true of Multilisp, which does not have a true compiler and BBN's Lisp which does (yet can take 1/2 hour to load a 50 rule knowledge base, which can be loaded in seconds in Common Lisp on SUNS). All the features one would like do not exist in current parallel Lisps, such as spin locks, threads as opposed to tasks, separate memory locations, etc. However, this is changing and will get better in the future. Top Level Lisp and Lucid both have recently come out with parallel Lisps which have good performance and some nice tools.

The parallel machines in existence are not mature and may not provide the kind of performance that they will in the future. Tools to determine what is happening in a parallel program and where the bottlenecks exist are largely non-existent, but extremely important. The C language provides access to most of the existing features of parallel machines and is much faster than most of the parallel Lisps. For this reason it makes a viable vehicle for this type of research [14]. It has proven possible to build expert systems in languages such as C and of course in C, which is why we have begun using a C based system to examine the possibilities with the Backpac system. It is clearly not in the class of Lisp for developing intelligent systems. The parallel Lisps should catch up to C as the sequential ones have.

It is clear that parallelization of expert systems (and one might also claim intelligent systems) can provide speed increases. In forward chained systems it is important to get around the limitation of so much time being spent in the match phase. The Datapac system does not spend so much time on match. However, it has other characteristics which must be further investigated before one can make a good comparison to OPS based systems. The incorporation of hierarchy information into OPS based systems (such as is done in Datapac) may provide a way of decreasing the percentage of time spent on match. Recent results from IJCAI-89 [41] suggest that saving less state than is done by the RETE algorithm may allow less percentage time to be spent in the match phase, also. Hence, more progress in speed increases in forward chained systems seems possible. At the current time, 64 processors would be the maximum needed in a machine to be used for parallel forward chained expert systems.

For backward-chained systems the picture is maybe less clear than the forward chained systems. The major bottleneck here is working memory, again. Currently, there are no definable limits on the speed-up that may be achieved. We have not made an extensive study of actual knowledge bases, but it is likely that their general structure will provide limits. Instead, we have concentrate on trying to determine general limits based on the structure of knowledge bases. The results have been mixed. It seems that careful attention to load balancing and the structure of knowledge base will allow the tuning of parallelism to provide better performance than we have currently obtained. Also, it is clear that spin locks on working memory are needed. This is an area of great contention, but the time spent modifying it is small. We have not had the capability to effectively implement spin locks in our Multilisp systems. It is also important to partition working memory, when it is possible. However, this is difficult to do in Multilisp (and we believe *non-packaged* Lisps in general). Lisp will provide different cons cells, but end up pointing them to the same memory location, which of course is where contention can occur.

It seems certain that order of magnitude speed-ups can be observed for some knowledge base structures, if one discounts the fact that most backward chained systems ask the user for information. This is clearly a sequential bottleneck. It also brings the question of how questions will be ordered, as they may come in a jumbled order due to the parallelism. This issue must be addressed for actual systems.

Whether or not two orders of magnitude of speed increase can be achieved with backward chaining is an open question. If we were to speculate, it would be that it can be achieved with large knowledge bases that require little, if any interaction with the user. In general parallelism can provide some benefits to the speed of expert systems and in the future may provide truly indispensable benefits.

It is too early in the game to definitively specify the *best* architecture for parallel expert systems. However, one can make some experience based speculations, which follow. The processors should each have some significant amount of local memory. There needs to be some shared memory also. This might be in one location or distributed, it is very unclear which is best. The processors we have been concerned with would be MIMD, capable of doing independent work, which is clearly different from a neural model. Because of the shared working memory, it would seem that some sort of hypercube or switch connected system would be better than a bus-based system. On the Multimax, it was hypothesized (but never definitively shown) that an unexpected amount of bus contention was causing unexpected performance degradation. At this time it would seem that the system need not be concerned with thousands of processors, but hundreds at the maximum. The ability to efficiently implement spin locks will be important.

2 Publications and programs

This section contains a summary of our current publications under the grant and the programs developed. Submitted papers are not included.

Parallel Rule-based Algorithms for Reasoning Systems, AAAI Spring Symposium Series on Parallel Models of Intelligence, Stanford, Ca. March 1988.

Parallel Rule-Based Algorithms for Reasoning Systems, 1st Florida Artificial Intelligence Research Symposium, Orlando, Fl. May 1988.

Parallelism Applied to Fuzzy Rule-Based Reasoning, North American Fuzzy Information Processing Society Conference, San Francisco, June, 1988.

(With O. Kim)

Datapac: A Parallel Reasoning Forward Chained System, Proceedings of the 2nd annual Florida A.I. Research Symposium, Orlando. April 1989.

(With T. Higgins and C. Eggert)

Backpac: A Parallel Goal-Driven Reasoning System, IJCAI-89 Workshop on Parallel Algorithms for Machine Intelligence, Detroit, Aug. 1989.

Parallel Rule-Based Algorithms for Reasoning Systems, Advances in Artificial Intelligence (Mark Fishman, Ed.), Jai Press, pp. 179-186, 1989, Greenwich, Conneticut.

(With O. Kim)

Results from a Parallel Reasoning Forward Chained System, Advances in Artificial Intelligence, Vol. 2, Mark Fishman (ed.), Jai Press, To appear.

Parallelism in Backward-chained Expert Systems: Experimental Results, Applications of Artificial Intelligence VIII, Orlando, Fl. April 1990.

Higgins, Timothy A. (1989), Issues in Parallel Expert Systems, M.S. Theses, Department of Computer Science and Engineering, University of South Florida, Tampa, December.

Kim, O. (1989), A Distributed Parallel Hierarchical Forward Chaining Inference System, Master's Thesis, Department of Computer Science and Engineering, University of South Florida, Tampa, April.

Programs:

Datapac (Multilisp version), Backpac (C, Multilisp, Mul-T, and Common Lisp versions), Parallel EMYCIN (Common Lisp and Multilisp versions).

3 Details of the results and issues in the USF ISL's parallel expert systems research

3.1 Introduction

Intelligent systems make inferences about a situation based on their internal knowledge. When they contain a lot of knowledge or are working on a *big* problem, many inferences and/or much searching of knowledge must be done. This can cause a knowledge-based system to be slow. Large knowledge bases currently might be classified as being on the order of about 5000 pieces of knowledge. In the near future NASA, for example, envisions knowledge bases of 20,000, and more, pieces of knowledge for the space station! At the 1988 AAAI spring symposium, the usefulness of medical knowledge bases with 100,000 or more pieces of knowledge was discussed. It is known that the current limits of computing on one processor are being approached and even if the limit wasn't being approached, the desired size of a knowledge base is probably growing faster than raw compute speed. Parallel inference may provide a significant speed increase and allow larger problems to be attacked effectively. This work has been targeted to determine the opportunities and limits for the incorporation of parallelism into rule-based intelligent systems.

NASA is in the process of developing a multiprocessor system to use in the Space Station for intelligent systems tasks. The spaceborne very large scale integrated circuit multiprocessor system (SVMS) will likely be made up of Lisp chips. It will be designed to support Artificial Intelligence tasks. Algorithms for AI and expert systems must be developed to support this and other parallel architectures. More specifically, expert systems for real time applications may be developed on such a machine and systems that would, currently, be too large to run in reasonable time on a uniprocessor may be investigated.

Initially, we will give a summary of our current progress. Then we will discuss our

plan for the continuation of the research.

3.2 Background

On the gross level of detail, independent inferences may be performed in parallel and much of an individual inference may also be performed in parallel (i.e. pattern matching). If the communication overhead of the parallel breakup of the reasoning system is not overwhelming some excellent increases in system speed may be hypothesized. Fine-grained parallelism may also enable speed-ups, if the overhead is kept to a minimum[14].

There are various ways to represent knowledge and perform inference [46]. Here, our work is concerned with a rule-based knowledge representation scheme, which allows fuzzy certainties or beliefs in both the rules and the working set of information that the system contains [17]. The use of fuzzy logic allows the modeling of many intelligent/expert tasks in domains which contain imprecision or uncertainty. The only real effect this has is to require continued work, if we are evaluating clauses in parallel. Even if a value for an *or'ed* clause is found (unless the value is one) processing must continue until they are all known, since a greater value may be found. A rule-based system may operate in either forward chained (data-driven) or backward chained (goal-driven) mode. Here, we discuss them in the both modes. We discuss the current results of a backward chained expert system, Backpac. Also, experience with Datapac and parallel EMYCIN is expounded upon. Each of the paradigms discussed here has been tried on at least one parallel processor machine.

There is much work going on in parallel Prolog [47]. Certainly, it can provide for effective parallel rule-based systems. Our approach is concerned with paradigms other than those available under Prolog. The Lisp paradigm is reported on here, but we are also looking at systems in C because of the efficiency and low-level features that it offers. In fact, the paradigm which makes use of the **future** construct as its primary parallel operator is the actual one used. It will be explained in the later discussion on the languages used.

Our systems have been implemented in Multilisp [26], which is a parallel version of the Lisp programming language. Multilisp has the flavor of the Scheme dialect of Lisp. Multilisp takes Lisp expressions and compiles them into an intermediate representation called Mcodes. At run time it interprets these Mcodes. It also has a run-time library of support routines which are written in Multilisp. Hence, the system is rather slow, especially when compared to a good compiled Lisp. It is very portable code, though. Code compiled on a SUN will (and has) run without change on a multiprocessor (the Concert), since the Mcodes across the two systems are the same. The implementation of the Mcode instructions is, of course, different between the two machines. In fact the Mcode versions of a program will run on any machine which has the same version of the Multilisp system.

Multilisp [27] has one primary parallel operator, which we will briefly describe. A function may be future'd which means to declare it to run in parallel. An example use of the construct is

```
(setq ex (future (+ 2 3))).
```

The future construct immediately returns a *future* value for ex and spawns a task to evaluate (+ 2 3). Now ex may be put in lists or manipulated in any manner which does not require an actual value. When an actual value is needed the task requiring it is suspended until the value is determined, unless the future has previously resolved to the necessary value, in which case operation continues normally. This powerful construct is the basis used to provide parallelism in the system discussed here. Semaphores are available to protect shared variables.

MUL-T [32] is a compiled Lisp which runs on the Encore Multimax. It has an environment which makes it look very much like Multilisp. In this environment, most Multilisp constructs are available. Most importantly for this work, the *future* construct for parallelism is available, as are semaphores. Since code in MUL-T can be compiled, it provides significantly faster execution times than does Multilisp.

Backpac and Datapac have been tested on the Concert multiprocessor system and the Encore Multimax [29, 23]. The Concert is a multiple instruction, multiple data machine (MIMD), which may be configured for up to 64 processors in the version of it we have used. The most processors actually used in this study are 31. The Concert consists of eight clusters each cluster having eight MC68000 processors, which have a half a megabyte of memory on board. The clusters are connected by a crossbar switch. There are eight megabytes of 16 way interleaved memory shared memory. All memory may be accessed by all the processors. However, we do not explicitly have non-local processors access another's local memory.

The Multimax had 10 processors in the configuration at USF. Five processor boards are in it, each with 2 processors. The machine may have up to 20 processors (10 boards). It is a shared memory machine with 32 megabytes of shared memory in this configuration. In addition each processor has a 64K cache local to it, which only it may access. The processors share a common bus which has 100 megabit peak throughput. Both Multilisp and MUL-T are available on the Multimax.

The basis for all speed-up comparisons on the Concert and Multimax is the following. The best sequential version of the developed system is run on one processor and its performance is compared with the best parallel version running on some specified number of processors. We do not time the initial loading and set up of the knowledge base in either the sequential or parallel version. Only the process of inferencing is timed.

3.3 Backpac

Backpac is a rule-based expert system which can diagnose diseases which cause fevers to be evident, or identify rocks or diagnose problems with four-stroke piston-driven car engines. These problems are attacked with separate knowledge bases. They make up our suite of real-world knowledge bases. While the domains are quite different, they can be loosely described as diagnostic in nature. The system chains backward from goals in its reasoning process. Backpac allows the user to enter fuzzy certainties to its queries. The system paradigm is that of a classification or diagnostic expert system. It is implemented in Multilisp and has been re-implemented in MUL-T. Backpac is implemented in such a manner that it can run in Common Lisp [59] with parallelism functions removed. It in fact has run on a Symbolics Lisp machine in common Lisp. This is important with the advent of parallel common Lisps from Lucid, Franz, and Top Level among others.

This effort at parallelizing rule-based expert systems differs from others [45, 38] in the following manner. Rather than parallelizing a match algorithm such as Rete [8] in an existing (in this case forward-chained system) or parallelizing an existing system, Backpac has been built from the start with parallel inferencing as a goal of its operation. One important issue this work has been aimed at, is to discover how much speed-up might be available in a rule-based backward-chained parallel reasoning system and what the limiting constraints are.

The system itself is simpler and more limited than one which uses Rete (or some other match algorithm in a match, recognize and act cycle type of production system). There are two versions of Backpac, one which makes some use of variables and the current model that we report on which does not. In this system a rule's premise is examined only once and pattern matched at that time. It is intended that under the variable version of the system several instantiations of the rule may be fired, but only at the time it is examined. Hence, minimal state information is saved. Backpac is not as general a tool as an OPS5-based system, but it is still capable of being used to solve interesting problems at the cost of more burden on the knowledge engineer.

Rules in the system are defined by Lisp functions. They may contain any level of the nested conjunctives *and* and *or*. Negation is also available. Special purpose functions can be added and some comparators exist. The rules come in two types, **goal** and others. Each clause in the rule is defined in a function in which it is associated with a string to be used to ask the system user questions. A clause can be a function such as **(greaterp age 15)**, where the value of age being larger than 15 is determined to be true or false. This is the only place that any variable facility is currently allowed in the version of Backpac (V.3).

Originally, Backpac was implemented in a simulator which ran on a VAX [19]. Unfortunately, it did not take into account memory contention or bus contention. Small granularity tasks tended to provide speed increases in the system. Also, until we made some optimizations the granularity of some tasks was larger than it should have been. The fallout of this is that we originally did much more in parallel than is done currently.

There is only one level of parallelism in Backpac. Every goal rule, a rule which has no successor and therefore provides a conclusion for the system, can be run in parallel. Alternatively, they may be broken into groups of goal rules and these may be run as one process. This high-level is the only one in which parallelism is used. If we attempt to determine the values of clauses in parallel for example, the system will slow down because the task granularity is not large enough in general.

Our algorithm for breaking goal rules up into groups is naive at this time. We simply partition them into the first available group of the desired size as they are encountered. This can lead to uneven task size.

There is a question ordering issue when the user is queried interactively. The system user will be processing questions sequentially, but they can arrive in an unordered manner. The questioning process must be able to order them so that the user perceives lines of questioning. In the parallel tests reported here values for the clauses are pre-set in working memory.

Backpac uses fuzzy modus ponens, as defined in [1], to reason with uncertain information. Conjunction is represented by *max* and disjunction is represented by *min*. Multiple pieces of evidence are combined as in the Fess expert system [17]. The evidence accrues into a stronger belief. In this area the only effect of fuzziness on parallelizing the system is that a non-zero predicate in an *or* does not necessarily mean do not evaluate the other ones. However, this only occurs if we examine clauses in parallel, which we are not currently doing.

3.4 Parallel reasoning

The following is a description of how Backpac's flow of control proceeds. Initially, all the bottom level goals are partitioned into groups. The user may set the size of the groups. The group size may be as small as one, which means each individual goal rule is fired by a separate process. These processes will also be responsible for doing any chaining which may be necessary.

At the largest level all the goal rules may be in one group. This will provide sequential execution of the rules with some added overhead, as one task to do them (separate from the control thread) is spawned. Alternatively, any number of intermediate group sizes may be created, each of which will correspond to a process.

Working memory is a shared data structure among processes. Values of clauses are represented on property lists. There is no difficulty with multiple readers and there is only one case in which multiple writers might collide. It is described in the following.

It may be the case that several rules from different process groups will require the same rule to be fired in order to determine a clause value. A semaphore is associated with clauses. When a rule is fired to determine a clause value the semaphore is set. This prevents other rules from firing. Those suspended can get the value from working memory after the rule has been fired and the semaphore released.

After all goal rules have provided a belief value for their consequents, the system will present all goals which have a belief value associated with them which is greater than a pre-set threshold. These values are returned by the goal rule and kept in a list.

3.5 Experimental results

We will now present the results from our tests of the Backpac system with high-level parallelism implemented. The results from Concert and the Multimax will be discussed separately. First the results from Concert will be discussed. The results are summarized in Table 1.

The best results from the four knowledge bases run on the Concert are reported here. In Backpac, a 17-fold increase in speed was observed with the use of 20 processors. This occurred on a knowledge base of 800 rules with them all on the goal level. This knowledge base could be viewed as a set of control rules, where periodic actions are taken based on some time clock. Using the knowledge base on fevers with the use of 12 processors by assigning goal rule clusters to each processor, the speed-up was just over 10 times. In a 400 rule knowledge base with 50 goal rules and 8 levels a speed increase of just over 9 was observed with the use of 10 processors.

From the table, it is clear without clustering goal rules onto processors, there is very little speed improvement. There are a set of lights on the Concert which indicate switch contention. In the case of no rule clustering they show a lot of contention. It is clear that as we use use more clusters of rules, performance will increase to a point and then adding clusters will provide a degradation of performance. This is due to the fact that adding another cluster of rules also adds a process to the system. The process then will make memory and switch requests, which cause contention. The contention becomes a more important issue than the task granularity. Also, as more clusters of goal rules are assigned to a processor the size of the cluster will go down, since the number of goal rules remains constant. Hence, task granularity is reduced in addition to the increase in contention.

On the Encore Multimax our speed-ups are less impressive. However, we do have less processors which is one factor. None of our efficiencies on the Concert is above 90%[1].

[1]Efficiency is defined as the amount of speed increase over the number of processors used.

Knowledge Base	*Speed up*	*Number of goal processes*
fevers	10.4	12
fevers	8.8	10
fevers	6.4	7
fevers	9.7	20
fevers	3.5	24
fevers	1.9	31
fevers	1.4	60
gems	5.16	10
gems	4.2	7
gems	2.9	11
400 rules, 8 levels	9.1	10
400 rules, 8 levels	8.1	12
400 rules, 8 levels	6.0	15
400 rules, 8 levels	2.3	30
400 rules, 8 levels	1.9	2
800 rules, 1 level	17.0	20
800 rules, 1 level	16.2	30

Table 1: Results from the Concert multiprocessor with 31 available processors.

Hence, 8+ times speed increase is about the maximum that we would expect from the Multimax.

The best results will be highlighted here and then we will discuss the overall flavor of these results. With the fevers knowledge base, a speed-up of 5.5 times with 8 processors was observed. This was the best result with a real-world knowledge base.

Some better results were observed with the use of generated knowledge bases. All of the generated knowledge bases have regularity in their structure. An 800 rule knowledge base made up of 8 levels of chaining and 100 rules per level was observed to have a little bit over a 7 times speed-up. This is with the use of 10 processors. Another knowledge base that had a good speed-up was the 400 rule knowledge base, which consisted of 8 levels with 50 rules per level. It provided a speed-up of slightly less than 6 1/2 times, when run on 10 processors. The rest of the results are summarized in Table 2.

It can be seen that chunking rules into groups on the goal level was done. The idea of increasing granularity and decreasing contention for memory and the shared bus seems reasonable. However, it did not provide the benefits hoped for. Apparently, having more available small tasks works better in some cases. This is despite the clearly increased overhead of having more tasks with a lower granularity size competing for resources.

3.6 Backpac in Mul-T

The Mul-T version of Backpac makes use of arrays for the types of knowledge representation that originally used property lists. This and the compiled nature of it makes the system very fast. Larger knowledge bases can and have been used under Mul-T. The performance of the fevers knowledge base is less since the granularity of the tasks has gotten quite small. To add processors causes a distinct degradation of performance. The multi-level knowledge bases prove to have little contention and get quite good performance. Table 3 contains the results of tests in MUL-T. All tests have just goal level parallelism implemented.

Knowledge Base	*Max. speed up*	*Number of processors*	*Rule clusters*
fevers	3.8	7	N/A
fevers	5.13	8	8
gems	3.095	9	N/A
gems	3.7	7	7
200 rules, 8 levels	5.24	9	N/A
200 rules, 8 levels	3.29	8	8
800 rules, 8 levels	5.14	10	10
800 rules, 8 levels	7.27	10	N/A
200 rules, 4 levels	5.53	9	N/A
200 rules, 4 levels	5.11	10	10
200 rules, 4 levels	3.59	8	7
400 rules, 8 levels	6.47	10	N/A
400 rules, 8 levels	5.09	10	10

Table 2: Results from the Encore Multimax.

Knowledge Base	*Max. speed up*	*Number of processors*	*Rule clusters*
fevers	3.43	6	N/A
1200 rules	5.24	9	N/A
1280 rules (2 levels)	6.31	10	N/A
400 rules (8 levels)	7.25	9	N/A
2560 rules (2 levels)	7.25	9	N/A

Table 3: Results from the Encore Multimax under MulT.

It is our intent to add more complex variable handling and pattern matching features to Backpac. This will up the task granularity and provide a more powerful system for experiments. One thing has become clear. The language, machine and operating system currently have a strong effect on results and all results must be looked at in the overall context.

3.7 Backpac in C

The Backpac system has also been written in C [4]. The C language is available on most parallel machines. Hence, this effort provides us a straightforward method to examine the performance of the system on different architectural platforms. The C language implementations are also, generally, faster in terms of compile and run time speed.

Briefly, we discuss some results from the Encore version. In this version the speed-ups were less than in the Lisp version for comparable knowledge bases. As the knowledge bases got larger, approximately the same speed-ups were observed as shown above. For an example, a generated knowledge base of 12,800 rules all on one level gave a speed-up of about 7 times with 9 processors. With the Fevers knowledge base using busy waits for access to working memory, we got about 6 times speed increase. These results do not include the initial overhead of copying shared information. One issue that shows up in this arena is that of memory copying and process overhead. The overhead of initially copying the shared information, such as working memory and sets of rules, is quite high. It is easier to measure than in the Lisp systems, where the initial start-up of processes and pointer structures is folded into the Lisp system initialization. However, over many runs of an expert system this overhead will become a negligible portion of each one.

3.8 Datapac

The Datapac system is a parallel hierarchical forward chaining inference system designed to maximize the inferencing speed in a multiprocessor environment by employing the parallelisms in a full range, which includes parallelism between different rule inferences as well as parallelism within a rule inference. The hierarchical forward chaining inference system is the rule based system where the rules are arranged in hierarchy according to the precedence relationship so that each rule is examined only once for firing. Rule level parallelism allows every rule to try to start its inferencing process as soon as it satisfies the precedence conditions. Thus, control of the inference flows from the top of the rule system to the bottom. This processing system, which could be modeled (and has been run) on a multiple instruction multiple data (MIMD) machines, has been simulated by Multilisp in this research. Performance evaluations demonstrate that this hierarchical forward chaining system (Datapac) provides a significant speed increase in a simulated environment and real gains in multiprocessor environment.

The basic idea is simple and very much aimed at expert problem solving. In order to come to some conclusion, a path of reasoning will be followed. That is, some chain of inferences will occur. Datapac breaks the various rules in a knowledge base, which may participate in a reasoning chain, into levels of rules. On each level the rules are independent of one another. On higher levels the rules depend upon the inference made by lower level rules. The bottom level leads nowhere and the top-level has no conclusions of other rules in its rules' antecedent.

This scheme is clearly limited. It turns a knowledge base into a sort of decision tree. However, it does have applicability for at least some types of expert knowledge bases.

Initially, an unlimited number of processors is assumed to be available for our system. All the rules in the system are given one processor respectively and start inferencing at once. What controls the reasonable transition of inferencing is the preceder-successor relationships between processors. A rule processor won't try matching its premise to the working memory for rule firing until its every preceding rule's firing is completed.

The working memory(WMP) contains all the facts which are either initially given or derived from previous inferencing. Each currently active rule has its antecedent matched against working memory, if it is to be ever fired.

After a rule is fired it sends the result to its succeeding rule processors immediately, enabling them to start inferencing.

Through the testings, this hierarchical forward chaining system (Datapac) shows a considerable improvement in speedup. For example, in a rule base where 15 non-variable rules on average reside in each level, the speedup ratio of parallel run to sequential run was over 9. For a variable rule system of 100 rules, where 20 variable or constant rules on average reside in each level, the speed-up ratio was over 10.

3.9 Results from simulation

This processing system run has been simulated by CSIM Multilisp [38] for performance testing. In our simulations, it is assumed that an unlimited number of processors are available. The results are reported in terms of speed-up and the maximum number of tasks, which ran in the parallel trials. The speed-up is measured by the best sequential version versus the best parallel version. The times are divided thereby providing the reported number.

Some artificial and real domains have been applied to the system. The results show that this hierarchical forward chaining system (Datapac) can provide a considerable improvement in terms of speedup. For example, for a 90 rule set without variables (constant) where 16 rules on average are in each level, the speedup ratio of parallel run to sequential run was over 9 with a maximum of 22 processors used. For a 100 variable rule system inferencing where 20 variable or constant rules on average reside in each level, the speed-up ratio of parallel run to sequential run was over 10 with a maximum of 180 processors used. A rule base consisting of 25 variable rules gave a speed-up of about 5 times with a maximum of 30 processors.

A knowledge base which consists of 64 rules and can diagnose fevers was a real example which has been used in our system. It showed a simulated speed-up of about

11 times with a maximum of 43 processors.

3.10 Results From The Encore Multimax

The system has also been run on a 10 processor Encore Multimax in a version of Multilisp which runs on the Multimax. With a knowledge base about fevers which consists of 64 rules, the system speeded up by a factor of about 6.3 times with 9 processors. This particular knowledge base has been implemented without variables. This is consistent with our quest to determine the limits of parallelism in reasoning systems. However, a generally useful system will need the use of variables. Note, one processor is left to do system's sorts of tasks, such as run the login shell and handle ethernet traffic.

On eight processors a knowledge base of one hundred rules provides us with a speed-up of almost linear proportions (a factor of 7.3). This rule base contains variables and the variable pattern matching, which has been parallelized, makes the task granularity larger than it would otherwise be. With a twenty-five variable rule knowledge base with nine processors a speed-up of about five and a half times was observed. These knowledge bases have to do with relationships in a family. They are really based on tests of logical structure and have no real depth of knowledge in them. The 100 rule knowledge base contains 25 rules per level. The 25 rule knowledge base has five rules per level. Basically, a Model can not be a substitute for a reality. Though the results from the real machine are the easiest justification of (usefulness of) a proposed scheme, the results from the model simulation could be a better suggestion for an idealistic machine structure itself.

3.11 Summary of Datapac issues

Fundamentally, a forward chaining system is a kind of exhaustive search system and it is commonly known that parallel architectures are very effective for that kind of search system. Significant speed-up improvement in our system demonstrates that the system is a good parallel scheme for forward chaining rule based inference systems. In this parallel inferencing scheme, the central control part becomes relatively small by distributing most of its decision to each individual rule processor. That is another indication that the system is really close to the one of the idealistic parallel processing concepts in that controls are distributed over individual processors. We see that as a knowledge base grows larger, further parallelism and speed-up are possible, almost linearly proportional to the number of independent sets of rules which means that there's no clear bottleneck with any big knowledge base (at least in the range of CSIM simulations and the inputs we used). The superiority of parallelism to the sequential execution tells about the suitability of the algorithm in parallel processing. But the superority of our hierarchical inferencing algorithm to the *random* inferencing can not be overlooked. Building a robust construct for more speedy and efficient parallel logical and/or subprocesses will be one of the main breakthroughs for an upgraded parallel inferencing scheme which, with current Multilisp features, is not easily achieved [31]. We observe that the size and the structure of the knowledge base affects the performance of the system.

There is also the question of how often such a model as this may be useful. This is one of the areas for ongoing investigation.

4 Experiments with parallel Emycin

EMY is the name given to a parallel version of EMYCIN by two researchers at the Microelectronics Computer Technology Consortium [38]. It runs under Multilisp, which is a parallel version of Lisp.

There are results from simulations, which are reported in [38] and [16]. In the first case contention for memory and possibly a bus are not taken into account. In the second simulation of EMY under Common Lisp, an attempt is made to incorporate reasonable hardware contention in the results. In this work, we compare the simulation results to the actual results from running the EMY system on an Encore Multimax, which is a bus-based architecture and the 64 processor switch-based architecture Concert machine. Further, we have made changes to the system to (hopefully) improve its performance. These results will also be presented with an analysis of the possibilities for providing any future speed-ups.

The parallelism used by Krall and McGehearty was on three levels. Each rule was applied in parallel. The hypotheses were traced in parallel. Each predicate of a premise was determined in parallel with the use of parallel *and* and *or* operators. An examination of the operation of EMYCIN shows that these are logical places for the insertion of parallel operators.

In actual operation on the machines we used, which are early versions of parallel architectures and thus have reasonably high overheads, there is some question on whether the granularity of determining predicates is large enough to justify the parallelism used. In fact in the Backpac system [20], we have found, that it is not in general. This same fact was true of EMY as is shown in the following.

Also, in other parallel systems with which we have worked, Backpac and Datapac, [19, 23] contention for memory and the bus have been observed to be problems in some cases. The hypothesis that EMY might also suffer from these problems was tested in EMY with inconclusive results.

On the Concert, all results reported here are with 23 processors available. Most of the time we were working with four megabytes of memory, although a couple of runs were done with 8 megs. There can be a difference of times based on less garbage collections. However, we were careful about filling up our garbage space and restarted the system when it began to be a problem (after multiple runs).

On the Multimax, results will be provided with the number of processors that they were obtained on. In general, it has been found that under Multilisp on the Multimax, performance is better with less than 10 processors. This apparently has to do with system tasks mixing in with user tasks and some scheduling overhead.

In initial results with EMY, and a knowledge base on fevers, it was predicted that about a 12 times speed-up would be the upper limit with 24 processors (comparable to our 23) [38]. With 8 processors the upper limit was 5.6 times [38]. In the later study EMY with the same knowledge base was shown to peak at a 9 times speed-up with 64 processors. In fact it came very close to this peak by 16 processors, although exact figures were not given [16]. The model was of a shared memory machine with a hierarchical interconnect and clustered processors (8 processors per cluster). It is somewhat like the Concert architecture, but rather different than the Multimax architecture.

Below, we just discuss on a high level the results from running parallel EMYCIN on the Concert. We have made some adjustments to the original EMY, but retained the semantics. A complete description of this work is in [25].

4.1 Results from parallel EMYCIN on the Concert

The results were lower than one would expect from 23 processors. With the fevers knowledge base a speed-up of five times was observed. In this case no *and/or* parallelism was used. Both the rules and hypothesis lists were broken into chunks. There were 20 rule chunks and 9 hypothesis chunks in the best case, given above. Not all possible combinations were tried, but enough to narrow down the best to within one or two chunks and the time to no more than about two tenths of a second difference. Note, that the speed-up reported belongs to the fastest parallel and sequential times over several runs.

Without using any chunking, the fevers KB provided a five times speed-up, also. Chunking was of essentially no help. If it was not done to maximize the speed with the *best* chunks, worse performance occurred! This was a surprise, so an investigation was made as to why it happened. It turns out that while the time to determine a hypothesis may vary by a factor of 2 it is small compared to the average rule evaluation time. This is not surprising since a hypothesis is evaluated by evaluating the appropriate premise of a rule which has already been fired. That is all the clauses or predicates have a stored value. Rule firing times on the other hand can vary widely. By an order of magnitude in fact. If the rules are not chunked or clustered very cleverly, it is likely that several long rules will end up in the same chunk. This process will then take a long time to finish, because it has much work to do. Now the work which may be done in parallel is reduced, so while contention may be less, it is offset by the lack of parallel work (or the increase in the sequentiality of the system).

The parallel *and/or* code was also tried on the Concert. However, with the chunked system it significantly increased the time to determine a conclusion (by up to a factor of 2). It spawns a lot of small tasks, whose overhead costs outweigh the benefits of more parallel work.

Knowing that semaphores are expensive, we tried one scheme to minimize the necessary locking on a semaphore. If a value has already been determined, it will be in the appropriate clause. Currently, whoever wants to look at it must check a semaphore. The code was changed so that a flag could be examined instead (after the value is initially determined). Unfortunately, the constant evaluation of the extra code cost more than was saved by the skip of the semaphore. It turned out that the time was only about four to eight percent greater. Hence, it was almost a wash.

The **gems** knowledge base was also examined under the EMY paradigm on the Concert. Its characteristics are distinctly different than the fevers KB. It has more rules and more levels of chaining (up to 4). With gems, just as in [38], the speed-up was less than with the fevers knowledge base. It was about 2.8 times. This was with chunking of 8 rule chunks and 8 hypothesis chunks. With no chunking, the speed-up was slightly slower. The use of the parallel and/or mechanism slowed the system down slightly, also. The method of skipping a semaphore did nothing for gems, either.

The results discussed are less than the simulation would lead one to expect. However, it is possible that they may be improved.

5 Directions from parallel tests

We, initially, used a simulator [19] to develop the systems which are reported upon. This simulator did not effectively take into account memory contention and bus contention delays. This lead us to great results, which we realized were inflated, and some higher expectations. After actual experimentation, we would not place too much faith in a simulation, except for a rather general idea of possible performance. The exception would be a simulator strongly tied to a specific machine architecture and using the measured or expected delay times, and contention (bus and memory) times, etc.

Another thing we learned is that creating a parallel task wherever there is a reasonable opportunity to perform some computation in parallel is not necessarily the correct action. Consideration of the amount of processors available must be done in addition to the usual granularity considerations. Many more tasks than processors caused a scheduling slowdown that was noticeable under our paradigm for some knowledge bases. Further, even if there are processors available, more tasks will create more bus and memory access requests. If they do create significant bus and memory access requests, contention will slow the system down. This is, of course, architecture dependent. The results of bus (switch) contention were quite evident on the Concert. On the Multimax, we suspected bus contention, but could not show it. We came to conclude that it was really memory contention, caused by the fact that we could not generate copies of objects in a reasonable fashion. Certainly, separate *cons* cells (pointers) were no problem, but in the end they would point to the same location where contention could occur. This was again a function of our Lisp's paradigm.

Good tools to determine what is happening in a parallel computing system are a necessity. These tools did not exist for us to measure performance under Lisp on either the Concert[2] or the Multimax.

[2]Actually, there were some parameter recording utilities in Multilisp. There was not documentation however and the decoding programs were targeted for a Symbolics machine to which we had no access.

6 Summary of current status

In our simplified model of a backward chained expert system, there have been some reasonable speed increases with the use of parallel processing. The limiting factors are not fully clear at this time. Certainly contention for working memory is one problem. Task granularity is another issue, but it is not unique to expert systems.

The Concert implementation of Backpac responds much better to clustering of goal rules into tasks than the Multimax implementation. On the Multimax clustering goal rules was helpful with the complicated predicates in the fevers and gems knowledge bases, but not elsewhere. Both machines use shared memory, but the architecture of the processors makes their characteristics markedly different. The current bottlenecks are contention based and better tools may help mitigate this.

The particular dialects of Lisp that we have been using have quirks of their own which have prevented some trials from being made. The lack of a way to get true copies of values and arrays has kept us from trying some methods to minimize memory contention. It has also prevented the effective partioning of working memory to reduce contention. This would show whether it is different from bus contention in preventing exploitation of parallelism. On the Multimax the lack of a way to keep other processes (such as the ethernet controller) from mixing in with ours has had an effect in obscuring some possible exploitation of parallelism.

Datapac shows promise in increasing inferencing speed with the use of parallel processors. The main issue here is how many problems can it effectively be used to solve. It is certainly much less functional than OPS5 with Rete or some other pattern matching algorithm. However, it has less bottlenecks for parallel processing and may be used to solve a reasonable set of expert emulation (system) problems.

Our initial test with the EMYCIN system show that the speed increase from parallelism is less than simulations tend to show. It also shows that little is gained from using

and/or parallelism in general. It seems that one might be able to use it in pre-identified places, however. Naive, clustering of rules is not helpful for the rules in the example knowledge bases that we have been using. This is due to the fact that the time to fire a rule can differ by an order of magnitude.

There is a benefit to parallelizing expert systems on current architectures. For backward chained systems, there are limits to the speed increase that will be available. These limits need to be better quantified. As the architectures become better able to handle finer grained tasks and the languages evolve, parallelism should be more of a factor in improving the performance of expert/intelligent systems.

6.1 Technological issues

There are several technologic issues which affect this work. Using C in the Backpac effort comes under this category. For it allows us access to low-level parallel features such as spin locks, which haven't been available under Multilisp. Spin locks in particular are a very useful concept for protecting working memory. A process that is writing to it will not take a large amount of time. However, the use of semaphores is time-consuming, because a process gets suspended (involving state saving) and put on a task queue waiting for an event. In this case a busy wait (which will normally be quite short) is more effective, embodying less overhead.

In addition there can be different classes of parallel tasks from full processes with all the information they embody to lightweight tasks sometimes called threads [42], which don't have a control or binding stack of their own. These entities can allow for finer grained parallelism than a full-fledged *future* process. The use of such entities will better allow for at least some predicate/clause processing to be done in parallel. These features are often available in C and are available in some of the new parallel common Lisp's such as Top Level's, PICCL[3]. We will use the Common Lisp features, when a machine

[3]PICCL is a trademark of the Top Level Co.

with the language on it comes available.

Knowledge-based systems may become large for a number of reasons. Among them are that the domain embodies a lot of information (often called commonsense notions), the domain is relatively broad or multiple areas of expertise are needed to solve the problem. These are not comprehensive or mutually exclusive, but they serve to introduce the final topic to be examined near the end of this proposal's work. Blackboards [5] have proven to be effective in allowing for multiple cooperating experts. In the blackboard concept several independent knowledge sources, which are expert systems in their own right cooperate to solve a problem. Intermediate solutions, solutions to portions of a problem and modifications of proposed solutions are posted to a global data structure called a blackboard. This has lead such systems to be called blackboard systems.

There are issues such as control and how the blackboard is used that make for significant differences between blackboard systems [44]. Clearly, the independent knowledge sources may operate asynchronously, or in parallel, under some control schemes. Indeed, one of the early expert system called HEARSAY [39], a speech understanding system, used the knowledge source, blackboard paradigm. It has further been investigated [44] in parallel simulations to look at the possible benefits in speed from parallelization. Not all the results at this time are very promising, but we believe it to be a domain dependent situation.

In the context of our work, the use of blackboards would fall under the broader heading of partitioning knowledge bases. It could provide a very clean partitioning with each knowledge source having its own working memory, thus reducing contention for that. Additionally, given a *good* partition of the problem the granularity of the knowledge sources could be kept high. This would be an artificial use of blackboard architectures getting away from the original idea of purely cooperating experts. However, it holds out the possibility of some clear advantages. Additionally, parallelism could be used within the knowledge base in the vein of Backpac and Datapac, providing that the knowledge sources had a large enough number of rules to warrant such activity. The investigation of the utility of blackboards for parallel expert systems would provide both a contrast and possible enhancement to the effort of parallelizing individual knowledge bases.

6.2 Summary

This work has involved the development and measurement of both goal-driven and data-driven expert system paradigms for parallel machines. Currently, we have a 64 processor Concert machine, 32 processor Butterfly machine, a 12 processor Ardent Titan (which is a collection of four networked machines in our College of Engineering) and an 8 processor Encore Multimax as available research platforms. A 16 processor Hypercube will become available in February or soon thereafter of 1990. Each of these machines runs Common Lisp, C, Multilisp or some combination of them. Promise has been shown in the use of parallelism in expert systems work. Many questions have been raised and a few of them answered.

References

[1] Bandler, W. and Kohout, L.J. (1984), The Four Modes of Inference in Fuzzy Expert Systems, *Cybernetics and Systems Research 2*, R. Trappl (ed.), North-Holland.

[2] Boborow, D.G., et.al. (1988), Common Lisp Object Specification, Chpts. 1,2, Xerox PARC, Palo Alto, Ca.

[3] Clark, K. and Gregory, S. (1986), PARLOG: Parallel Programming in Logic, ACM Transactions on Programming Languages and Systems, V. 8, No. 1, pp. 1-49.

[4] Eggert, C.V. (1989), Backpac Results under C, Technical Report ISL-3-89, Dept. of Computer Science, Univ. of South Florida, Tampa.

[5] Englemore, R. and Morgan, T. (eds.) (1988), Blackboard Systems, Addison-Wesley, Reading, Ma.

[6] Fergurson, C. and Korf, R.E. (1988), Distributed Tree Search and its Application to Alpha-Beta Pruning, AAAI-88, pg. 128-132.

[7] Finkel, R. and Fishburn, J. (1982), Parallelism in Alpha-Beta Search, *Artificial Intelligence*, Vol. 19, No. 1, Sept.

[8] Forgy, C.L. (1982), Rete: A fast algorithm for the many pattern/many object pattern match problem, *Artificial Intelligence*, 19, 17-37.

[9] Forgy, C.L. (1981), OPS5 User's Manual. Tech Report CMU-CS-81-135, Department of Computer Science, Carnegie-Mellon University.

[10] Greenberg, M. and Cuny, J.E. (1988), Parallelism in Knowledge-Based Systems with Inheritance, International Conference on Parallel Processing, August 1988, pp. 141-145.

[11] Gupta, A. (1987), Parallelism in Production Systems, Morgan Kaufmann Publishers Inc., Los Altos, Ca.

[12] Gupta, A., Tambe, M., et. al. (1988), Parallel Implementation of OPS5 on the Encore Multiprocessor: Results and Analysis, *International Journal of Parallel Programming*, V. 17, No. 2, April.

[13] Gupta, A., Forgy, C.L., Kirp, D., et. al., Parallel OPS5 on the Encore Multimax, International Conference on Parallel Processing, August 1988., pp. 271-280.

[14] Gupta, A. and Tambe, M. (1988), Suitability of Message Passing Computers for Implementing Production Systems, AAAI-88, pg.687-692.

[15] Gupta, A., Forgy, C. and Newell, A. (1989), High-Speed Implementations of Rule-Based Systems, *ACM Transactions on Computer Systems*, Vol. 7, No. 2, pp. 119-146.

[16] Guzman, A., Krall, E.J., McGehearty P.F. and Bagherzadeh, N. (1987), Performance of Symbolic Applications on a Parallel Architecture, *International Journal of Parallel Programming*, Vol. 16, No.3, pp. 183-214.

[17] Hall, L.O. and Kandel, A. (1986), Designing Fuzzy Expert Systems, Verlag TUV Rheinland, Germany.

[18] Hall, L.O. (1987), Architectures for Reasoning in Parallel, NASA-Ames Research Center, RCR branch report 2018, Moffett Field, CA.

[19] Hall, L.O. (1988), Parallelism in Fuzzy Rule-Based Reasoning, University of South Florida, Dept. of C.S., Report CSE-88-00003, Tampa, Fl.

[20] Hall, L.O. (1988), *Parallel Rule-based Algorithms for Reasoning Systems*, AAAI Spring Symposium Series on Parallel Models of Intelligence, Stanford, Ca. March 1988.

[21] Hall, L.O. (1988), *Parallel Rule-Based Algorithms for Reasoning Systems*, 1st Florida Artificial Intelligence Research Symposium, Orlando, Fl. May 1988. Also a version will appear in *Recent Advances in Artificial Intelligence*, JAI press, 1989.

[22] Hall, L.O. (1988), *Parallelism Applied to Fuzzy Rule-Based Reasoning*, North American Fuzzy Information Processing Society Conference, San Francisco, June, 1988.

[23] Kim, O. and Hall, L.O. (1989), *Datapac: A Parallel Reasoning Forward Chained System*, Proceedings of the 2nd annual Florida A.I. Research Symposium, Orlando. April 1989.

[24] Hall, L.O., Higgins, T. and Eggert, C. (1989), *Backpac: A Parallel Goal-Driven Reasoning System*, IJCAI-89 Workshop on Parallel Algorithms for Machine Intelligence, Detroit, Aug. 1989.

[25] Hall, L.O. (1989), Parallel EMYCIN on an Encore Multimax and the Concert, Technical report ISL-1-89, Dept. of Computer Science, University of South Florida, Tampa.

[26] Halstead, R.H. (1985), Multilisp: A Language for Concurrent Symbolic Computation, *ACM Transactions on Programming Languages and Systems*, Vol. 7, No. 4, October.

[27] Halstead, R.H. (1986), Parallel Symbolic Computing, *IEEE Computer*, August, pp. 35-43.

[28] Halstead, R.H. (1986), Concurrent Lisp Machines, M.I.T. Lab for Computer Science, To appear.

[29] Halstead, R.H. and Thaker, G. (1988), The Concert 1.0 Execution Environment, M.I.T. Laboratory of Computer Science, Cambridge Ma.

[30] Higgins, T.A. (1989), Issues in Parallel Expert Systems, Master's Thesis, Department of Computer Science and Engineering, University of South Florida, Tampa.

[31] Kim, O. (1989), A Distributed Parallel Hierarchical Forward Chaining Inference System, Master's Thesis, Department of Computer Science and Engineering, University of South Florida, Tampa.

[32] Kranz, D. (1988), MUL-T manual, Department of Computer Science, Tech. Report, Yale University.

[33] Kumar, V., Ramesh, K., and Rao, V.N. (1988), Parallel Best-First Search of State-Space Graphs: A Summary of Results, AAAI-88, pg. 122-127.

[34] Kumar, V. and Kanal, L.N. (1984), Parallel branch and bound formulations for and/or tree search, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6:768-778.

[35] Kelly, M.A. and Seviora, R.E. (1987), A Multiprocessor Architecture for Production System Matching, AAAI-87, pp.36-41.

[36] Kelly, M.A. and Seviora, R.E. (1989), An Evaluation of DRete on Cupid for OPS5 Matching, IJCAI-89, pp.84-89.

[37] Kowalik, J.S. (ed.) (1988), Parallel Computation and Computers for Artificial Intelligence, Kluwer Academic Press, Norwell, Ma.

[38] Krall, E.J. and McGehearty, P.F. (1986), A Case Study of Parallel Execution of a Rule-Based Expert System, *International Journal of Parallel Programming*, Vol. 15, No.1.

[39] Lesser, V.R., Fennell, R.D., Erman, L.D. and Reddy, D.R. (1975), Organization of teh Hearsay-II speech understanding system, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-23, 11-24.

[40] Miranker, D.P. (1987), Treat: A Better Match Algorithm for AI Production Systems, AAAI-87, The National Conference on Artificial Intelligence.

[41] Miranker, D.P., Kuo, C-M, Browne, J.C. (1989), Compiling Parallelism Among Rules, *IJCAI89 Workshop on Parallel Algorithms for Machine Intelligence and Pattern Recognition*, Detroit, August.

[42] Murray, K.E. (1989), Umass Concurrent Common Lisp: An Extensible and Efficient Concurrent Programming Language, COINS Technical Reprot 89-90, Dept. of Computer and Information Science, University of Massachusetts, Amherst, Ma.

[43] Moon, D., Stallman, R.M. and Weinreb, D. (1983), Lisp Machine Manual, A.I. Laboratory, Massachusetts Institute of Technology.

[44] Nii, H.P. (1986), CAGE and POLIGON: Two Frameworks for Blackboard-Based Concurrent Problem Solving, Technical Reprot KSL-86-41, Stanford University, Knowledge Systems Laboratory, Stanford, Ca.

[45] Oflazer, K. (1984), Partitioning in Parallel Processing of Production Systems, *Proceedings 1984 International Conference Parallel Processing*, IEEE Computer Society Press, pp. 92-100.

[46] Rich, E. (1983), Artificial Intelligence, McGraw-Hill, N.Y.

[47] Shapiro, E. (1986), Concurrent Prolog: A Progress Report, IEEE Computer, August, pp. 44-58.

[48] Schmolze, J.G. (1988), An Asynchronous Parallel Production System with Distributed Facts and Rules, *Proceedings of AAAI-88 Workshop on Parallel Algorithms for Machine Intelligence and Pattern Recognition*, St. Paul, Minn.

[49] Schwetman, H. (1986), CSIM Reference Manual, Microelectronics and Computer Technology Corporation, Austin, TX.

[50] Siler, W. and Tucker, D. (1989), Patterns of Inductive Reasoning in a Parallel Expert System, *International Journal of Man-Machine Studies*, V. 30, pp. 113-120.

[51] Smith, B.T. and Middleton, D. (1988), Exploiting fine-grained parallelism in Production Systems, NASA-Langley Research Center Report.

[52] Sobell, M.G. (1984), A Practical Guide to the Unix System, Benjamin-Cummings, Reading, Ma.

[53] Stolofo, S.J. (1984), Five Parallel Algorithms for Production System Execution on the DADO Machine, AAAI-84, The National Conference on Artificial Intelligence.

[54] Tinker, P.A. (1988), Performance of an OR-Parallel Logic Programming System, *International Journal of Parallel Programming*, Vol. 17, No. 1., pp. 59,92.

[55] Turksen, I.B. (1984), Production Control with a Linguistic Rule-Based System, *First International Conference on Fuzzy Information Processing*, Kauai, Hawaii.

[56] Umeyama, S. and Tamura, K. (1983), A Parallel Execution Model of Logic Programs, In the 10^{th} Annual International Symposium on Computer Architecture, IEEE and ACM, June.

[57] Van Melle, W., Scott, A., Bennet, J., and Peairs, M. (1981), The Emycin Manual, Tech. Rep STAN-CS-81-885, Stanford University, Stanford, Ca.

[58] Walmer, L.R. and Thompson, M.R. (1988), A Programmer's Guide to the Mach User Environment, Department of Computer Science, Carnegie-Mellon University, Pittsburg.

[59] Wilensky, R. (1986), Common LISPcraft, W.W. Norton, N.Y.