N88-17259

IMPLEMENTING CLIPS ON A PARALLEL COMPUTER

Gary Riley
NASA/Johnson Space Center
Artificial Intelligence Section
Mail Code FM7
Houston, Texas 77058

ABSTRACT

The 'C' language integrated production system (CLIPS) is a forward chaining rule-based language developed by the Artificial Intelligence Section (AIS) of the Mission Planning and Analysis Division (MPAD) at the Johnson Space Center (JSC) to provide training and delivery for expert systems. Conceptually, rule-based languages have great potential for benefiting from the inherent parallelism of the algorithms that they employ. During each cycle of execution, a knowledge base of information is compared against a set of rules to determine if any rules are applicable. Parallelism can be employed to speed up this comparison during each cycle of execution. Parallelism also can be employed for use with multiple cooperating expert systems. To investigate the potential benefits of using a parallel computer to speed up the comparison of facts to rules in expert systems, a parallel version of CLIPS was developed for the FLEX/32, a large-grain parallel computer. The FLEX implementation takes a macroscopic (or high-level) approach in achieving parallelism by splitting whole sets of rules among several processors rather than by splitting the components of an individual rule among processors. The parallel CLIPS prototype demonstrates the potential advantages of integrating expert system tools with parallel computers.

INTRODUCTION

Expert system building tools have shown a great deal of utility in solving knowledge intensive tasks that would often daunt conventional approaches using procedural languages. These tools provide languages that allow solutions to be expressed in paradigms that "closely" resemble the human solution process. Knowledge engineers can express heuristics using rule paradigms as opposed to coding nested if/then statements in a procedural language. The inference engine of the expert system is used to determinine which information has satisfied the conditions of the appropriate rules. The control routines for matching information (facts) against rules are provided by the tool, not the programmer. In addition, many expert system building tools are provided on computers hosting extremely powerful development environments that promote the interactive and incremental development of programs.

The use of high-order languages, however, does not come without cost. Typically, expert systems written in high-order languages run one to two orders of magnitude slower than expert systems directly coded in procedural languages. Speed is very often traded for increased productivity during development and easier maintenance. Sometimes, this tradeoff is acceptable, but many applications requiring real-time speed that could benefit from expert system technologies might not be able to accept this tradeoff.

The AIS of JSC's MPAD has been active in both the design of expert system building tools and the use of parallel computers. Several expert systems have been developed which require real-time or near real-time speed, including NAVEX[1] and MCCSSES[2]. Parallel processing is one of the ways in which expert system speed performance can be increased[3]. This background presented the opportunity and motivation to investigate the use of parallel processing in expert system building tools.

CLIPS

CLIPS is a forward chaining, rule-based language developed by the AIS at JSC to solve both training and delivery problems not fully addressed by most commercially available expert system shells[4]. A forward chaining, rule-based language such as CLIPS has three primary components: a set of rules, a knowledge base consisting of facts, and an inference engine. Facts represent chunks of information such as the altitude of the Space Shuttle or the temperature reading of a particular sensor. Rules basically are if/then statements of heuristic knowledge. The if portion of a rule is a series of patterns which must have appropriate matches with facts in the knowledge base for the rule to be activated. The then portion of a rule is a series of actions to be taken when the rule is executed. Two possible actions (among many) could be to add new facts to the knowledge base or to remove existing facts from the knowledge base. The inference engine is the mechanism that determines which rules apply and also compares the facts in the knowledge base to the rules and determines which rules are applicable given the current state of the knowledge base. It then selects one of the applicable rules and applies the actions found in the then portion of the rule. For a more complete description of CLIPS, see references [5] and [6].

FLEX/32 PARALLEL COMPUTER

The FLEX/32[7] is a large-grain parallel computer capable of housing up to 20 computer modules and 10 shared memory modules in 1 cabinet. Cabinets also can be connected together. Computer modules available are based on the Motorola 68020 and the National 32032 and may be used in any combination. The FLEX/32 is a multiple instruction stream/multiple data stream (MIMD) computer. Each processor can run independent of the others and can access either shared or local memory. The FLEX/32 (used by the AIS at JSC) has six National 32032 processor modules and two shared memory modules. The processor supporting UNIX has 4 megabytes of local memory, while the other five processors have 1 megabyte of local memory. Each common memory module has 256 kilobytes of memory.

The operating system used on the FLEX/32 is the UNIX System V Operating System. This provides all of the language support normally associated with this operating system. In addition, Flexible Computer offers two languages for parallel programming: Concurrent C [8] and Concurrent FORTRAN. These two languages have been extended to allow parallel processing constructs.

APPROACH TO PARALLELISM

Two levels of incorporating parallelism into CLIPS were considered: macroscopic and microscopic parallelism. A macroscopic approach would attempt to preserve the low-level implementation of the CLIPS inference engine and to incorporate parallelism on a "high" conceptual level. A microscopic approach, by contrast, would attempt to incorporate parallelism in the low-level implementation of the CLIPS inference engine.

A macroscopic approach would provide the quickest means of incorporating parallelism into CLIPS. Source code changes using this approach could be kept to a minimum by utilizing most of the code used for the sequential version of CLIPS. This was desirable because a sequential version of CLIPS was being maintained on a VAX 11/780 for use on that and other sequential computers. This sequential version experiences frequent change for both maintenance and improvement. A macroscopic approach would allow easier integration of changes made in the sequential version with the parallel version. Use of a macroscopic approach also would allow the final product to be a fully developed expert system tool and not a research prototype. A "start from scratch" approach inevitably would not contain of all of the features the sequential version of CLIPS provides. Finally, the source code for CLIPS already was well understood and available.

A microscopic approach would allow the investigation of the best possible techniques for incorporating parallelism. Converting a program developed on a sequential computer to take advantage of the architecture of a parallel computer would not be able to take advantage of other algorithms that may better exploit the architecture of the parallel computer. Recoding the inference engine to take advantage of parallelism at a very low level would allow the very best techniques to be applied.

PARALLEL CLIPS OVERVIEW

A macroscopic approach incorporated parallelism into CLIPS. This approach limited the number of changes to the CLIPS source code and allowed the ongoing changes to the sequential version of CLIPS to be integrated more easily with the parallel version of CLIPS.

The steps taken for the assertion of a new fact are shown for the sequential version of CLIPS in figure 1. First, the fact is filtered through the pattern matcher. The pattern matcher determines which patterns in the if portion of the rules have been matched by the fact that is being asserted. Rules with matched patterns then are given the information that a fact has matched one of their patterns. If this additional fact causes all of the conditions of the rule to be satisfied, the rule is placed on the agenda (in this case, local to a single processor). After all new assertions have taken place, a rule will be selected from the local agenda and its actions will be applied.

An assertion in the parallel version adds an additional level above local fact assertions. The FLEX/32 implementation of CLIPS splits the set of rules among several processors to achieve parallelism. Figure 2 shows the steps taken to assert a fact in the parallel version. A master processor provides the user interface capabilities and acts as a driver for the other processors. The master processor picks a single rule to be applied from the global agenda (the set of all applicable rules). The master processor then informs the processors containing groups of rules of the actions of the rule that are to be applied. A given action of the rule then is performed by all of the rule group processors in parallel before the next action of a rule is undertaken.

In the case of a fact assertion, the processor containing the rule is informed that it may begin executing its actions. The fact to be asserted is posted to global memory, and the other processors are notified that a fact has been posted to global memory. Each individual processor then asserts the fact exactly as if it were running on a sequential computer using the steps shown in Figure 1. After the assertion takes place on the processor, the local agenda selects one applicable rule (if it has any available) to be posted to the global agenda.

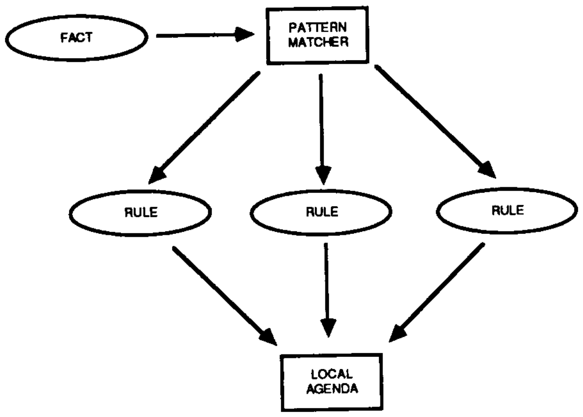

Figure 1: Local Fact Assertion

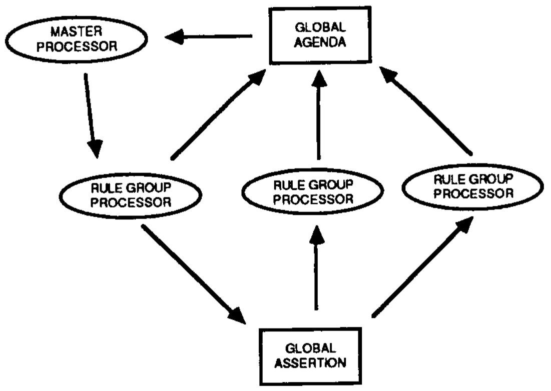

Figure 2: Global Fact Assertion

Retractions are handled similarly to assertions, with the fact to be retracted being posted to global memory and other processors being informed of the task by the rule group processor that contains the rules and is executing the actions. The rule group processor containing the rule waits for all of the other processors to finish before beginning the next action. Other actions that take place in the then portion of a rule (variable bindings, function calls, etc.) are handled only by the processor with the executing rule. Once the rule has finished executing, control is returned to the master processor where another rule is selected from the global agenda to be executed, repeating the basic cycle until no rules remain on the global agenda.

IMPLEMENTATION

The main problem in the implementation of parallel CLIPS was the communication between the master and slave processors. Initial attempts used processes to create and control the slave processors and their tasks. For example, if the main processor wanted the slave processors to assert a fact, it would create a process running on each of the slave processors to handle this task. The main processor then would wait for slave processors to finish. This method turned out to be relatively easy to code using the high-level, parallel constructs of Concurrent C; however, it also was quite inefficient. Sample problems actually ran slower as the number of additional processors was increased. Process creation is expensive, especially when the task to be performed is of a small time duration. Further, multitasking on a single processor also does not seem to work as well as one might expect. Running a slave process on the same processor as the master process caused inefficiency in multitasking. The FLEX/32 arbitration for multitasking does not appear to be very efficient. This conclusion was bolstered further by the results of other parallel programs.

The second implementation attempt corrected two of the errors experienced in the first attempt. Process creation and multitasking were avoided during run time of the expert system. The processor with the master process was not given a slave process. All other processors had a slave process. This slave process ran constantly, waiting for a "message" which informed it that it had a task to perform. When it received the "message" and processed it, the slave process then would send a "message" back to the master process, informing it that the task had been completed.

An attempt was made to use the message passing facility of exceptions provided by Concurrent C; however, it proved too difficult to configure the channels in the appropriate manner for message passing. The final implementation used a set of flags in shared memory. Each processor had an active flag and, in addition, all processors shared a task flag. A processor requiring other processors to perform an action would set the task flag to the appropriate task to be performed. It then would set the active flag of the other processors to active to signal them to begin execution of the task. The controlling processor then would monitor the active flag. When the active flag was set to inactive, the controlling processor knew that the subordinate processor has completed its task. Information passing was controlled by copying information to global memory and by having each processor copy the information down to its local memory.

The problems encountered during implementation showed that many ways exist to implement a problem given a concurrent language on a parallel computer. Unfortunately, the best way to implement a problem often has to be determined empirically.

RESULTS

Two problems were used to demonstrate potential speed benefits of the parallel algorithm used in parallel CLIPS. The first of these problems was a "goal" problem. This problem was a 30-rule version of the monkeys and bananas problem described in *reference [9] modified to handle more goals and* situations. Eighty-six rules fire to solve the problem for the initial conditions used. The other problem used was a "data" problem. This problem has 13 rules: 1 startup up rule and 12 data-intensive rules. The data-intensive rules were combinatorial in nature in that each rule potentially could add tens to thousands of rule activations to the agenda with the addition of each new fact (depending upon the number of facts already in the knowledge base). To prevent all of these rule activations from actually occurring, a pattern was added to the end of each of the if portions of the rules which had no corresponding matches among facts in the knowledge base. Although this pattern prevented the rules from being activated, it still allowed the computational work in computing the partial matches to be finished. The startup rule asserted nine facts and was the only rule that fired.

The problems were run on CLIPS V3.11 on a VAX 11/780 using VMS, CLIPS V3.11 on the FLEX using UNIX, and parallel CLIPS (based on V3.11) using one to four processors under the multitasking multiprocessing operating system (MMOS). The results are shown in table I.

Table I: Timing Test Results

Version	Data Problem	Goal Problem
CLIPS VAX	15.2	3.3
CLIPS FLEX	21.9	6.4
Parallel CLIPS (1P)	18.1	5.7
Parallel CLIPS (2P)	9.3	5.3
Parallel CLIPS (3P)	7.3	4.4
Parallel CLIPS (4P)	5.5	4.2

The "goal" problem demonstrated only modest speedup as more processors were added. This demonstrates that speedup will occur only for problems in which the problem is divided evenly among the processors. That is, for each fact assertion and retraction, each set of rules on a processor has approximately the same amount of work to perform. This could best be achieved with a set of rules that numbers in the hundreds rather than in the tens.

The "data" problem specifically was tailored to demonstrate a "best case" situation for parallel CLIPS. Only one rule is fired and this rule asserts several facts. For each of the facts asserted, a great deal of work has to be done and this work is very evenly divided among the processors. Two processors ran the problem 1.9 times faster, and four processors ran the problem 3.3 times faster than a single processor. These numbers represent 95 percent and 80 percent, respectively, of maximum possible speedup.

Although rule sets run slightly faster for most examples and much faster for some examples, it is important to remember that the inference engine is not actually working faster. Parallel CLIPS speeds up the system by making the set of rules appear smaller by distributing them among several processors.

AREAS FOR IMPROVEMENT

CLIPS uses the Rete pattern matching algorithm which provides an efficient method for finding all of the facts that match the patterns in the if portions of the rules[10],[11]. It is important to remember that optimizations used in the Rete algorithm may be affected by splitting up rules among processors. Common elements of both patterns and rules can be shared, making the system more efficient. To split the rules among several processors will remove some of the efficiency that is gained by sharing. The version of CLIPS used for parallel CLIPS (version 3.11) uses the Rete algorithm. However, it does not take advantage of common sets of patterns shared between rules (join sharing). Starting with version 4.0, versions of CLIPS incorporate this optimization. The "data" problem used cannot take advantage of join sharing; however, most problems can take advantage of join sharing to a greater or lesser extent. For example, the "goal" problem has 7 of its 30 rules which can benefit from join sharing. Join sharing especially benefits large expert systems with many sets of similar rules. A version of parallel CLIPS based on version 4.0 would allow investigation of the tradeoffs encountered between sharing commonality among rules on a single processor and splitting rules among several processors.

The next logical step in testing the benefits of parallel CLIPS is to develop a suitable problem for testing large expert systems. This problem should consist of at least 100 rules and should not be dependent on extensive input/output (I/O) or external functions. The initial state or condition should be hardwired so the problem can just execute without human intervention.

The parallel implementation could make use of an action queue to store a list of assertions and retractions to be performed by the rule groups. Each processor could retrieve the next action to be performed from this queue when it has completed its current action. This would ease some of the strict synchronization of rule execution and also would allow processors to proceed at their own pace rather than at the pace of the slowest processor out of all the groups.

Programming constructs should be provided which allow rules to be specifically assigned to certain processors by the programmer. In the current implementation, CLIPS distributes rules among processors with a round robin distribution scheme. The ability to assign rules specifically to processors would be useful when attempting to fine tune a parallel expert system for speed.

CONCLUSIONS

The early results from parallel CLIPS are very encouraging. Parallel CLIPS could be used not only as a program for investigating parallel inference engines, but as a program for the actual delivery of an expert system. Parallel CLIPS is still a prototype, and more development work is required to remove the remaining rough edges. In addition, more suitable problems need to be found to investigate the speed improvements possible with parallel CLIPS.

ACKNOWLEDGEMENTS

The author would like to thank Joe Giarratano and Chris Culbert for their comments and suggestions on this paper.

ACRONYMS

AIS	Artificial Intelligence Section
CLIPS	C language integrated production system
I/O	input/output
JSC	Johnson Space Center
MIMD	multiple instruction stream/multiple data stream
MMOS	multitasking multiprocessing operating system
MPAD	Mission Planning and Analysis Division

REFERENCES

[1] Maletz, M. and C. Culbert, "Monitoring Real-Time Navigation Processes Using the Automated Reasoning Tool (ART)." In Proceedings of the First Annual Aerospace Applications of Artificial Intelligence Conference, AAAIC-85, Dayton, OH, September 1985.

[2] Clemons, P., C. Culbert, and L. Wang, "Development of an Expert System to Assist Monitoring Mission Control Center Software Status." In Proceedings of the First Annual Conference on Robotics and Expert Systems, ROBEXS'85, Houston, TX, June 1985.

[3] Boarnet, M., "Requirements for the Next Generation of Expert System Builders." NASA Technical Memo FM7(86-27), NASA/Johnson Space Center, Houston, TX, February 1986.

[4] Giarratano, J., C. Culbert, G. Riley, and R. Savely, "A Solution to the Expert System Delivery Problem." Submitted for publication.

[5] Culbert, C., "CLIPS Reference Manual." NASA Technical Memo FM7(87-131), NASA/Johnson Space Center, Houston, Texas, July 1986.

[6] Giarratano, J., "CLIPS User's Guide." NASA Internal Note 86-FM-25 (JSC-22308), Mission Planning and Analysis Division, NASA/Johnson Space Center, Houston, TX, October 1986.

[7] Flexible Computer Corporation, "FLEX/32 MultiComputer System Overview." Flexible Computer Corporation, Dallas, TX, June 1985.

[8] Flexible Computer Corporation, "ConCurrent C Reference Manual." Flexible Computer Corporation, Dallas, TX, March 1986.

[9] Brownston, L., R. Farrell, E. Kant, and N. Martin, Programming Expert Systems in OPS5: An Introduction to Rule-Based Programming. Addison-Wesley Publishing Company, Inc., Reading MA, 1985.

[10] Forgy, C. L., "On the Efficient Implementation of Production Systems." Ph.D. Dissertation, Carnegie-Mellon University, Pittsburgh, PA, 1979. (Available from University Microfilms International, Ann Arbor, MI)

[11] Forgy, C. L., "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem." Artificial Intelligence 19, 1982, pp. 17-37.