A SYSTEM FOR DISTRIBUTED INFERENCING

Bradford J. Rodriguez
Department of Electrical and Computer Engineering
and
W. F. S. Poehlman
Department of Computer Science and Systems

Applied Computersystems Group [1]
Department of Computer Science and Systems
McMaster University
Hamilton, Ontario L8S 4K1 Canada

ABSTRACT

The Applied Computersystems Group at McMaster University has been researching the use of inference-driven microprocessors for distributed real-time control. Prior work has involved a network of single-board-computers (SBCs) with a single inference engine. Now we are exploring the use of multiple inference-driven "agents" cooperating over a network.

Rather than adopt the prevailing "blackboard" model, we have developed an advisory/consulting model in which each agent is free to share information directly with its peers. System knowledge is composed of "facts" (the current state) and "rules" for deducing the facts. Facts are represented as "distributed objects"; an extension of the inference engine allows facts to be cached, obsoleted, and automatically updated.

We have also devised a representation scheme for the rules which allows them to freely migrate around the network. "Mobile" rules are written in a processor-independent token-threaded language, ITL. Our eventual goal is to automatically redistribute the inferencing tasks among the networked processors, to compensate for short- or long-term changes in load.

THE PROJECT

For several years, the Applied ComputerSystems Group of McMaster University has been researching the application of artificial intelligence techniques -- expert systems, fuzzy logic, and neural networks -- to real-time control problems. Recently our attention has turned to the use of distributed networks of processors.

The objective of this research project has been to develop a distributed expert system for embedded microcontrollers. Such processors are characterized by their small word size (8 or 16 bit), small address space, and relatively slow speed. Commercial expert system shells, requiring as they do a conventional operating system such as Unix or Windows, cannot be used with single-chip CPUs.

The target of this control system is the Tandem Accelerator at McMaster University (Figure 1). This is a model FN particle accelerator, used to accelerate positive ions to moderate energies for physics research. The accelerator system has approximately 70 observable quantities and 30 controllable quantities, and is normally run by a human operator with no automated assistance.

A similar but much less complicated machine, the model KN Van de Graaff accelerator, was recently automated by the ACsG [Lind, 1992]; this control system employed an inference engine on an IBM PC, using single-board computers strictly as I/O controllers.

The work described herein has expanded objectives.

First, use inference-based control for a larger and more complex problem (the FN accelerator). Some indication of the complexity is revealed by just one subtask of the control system: charging the accelerator terminal to a desired voltage. Figure 2 shows a simplified model of the terminal charging system. This is one of the few subsystems which is susceptible to an analytical solution; much of the accelerator cannot be reduced to any mathematical form.

Second, move the inference engine from the PC out to the single-board computers, sharing the load among several processors. Each microprocessor should be capable of making immediate expert decisions at the control point, rather than relaying information to a central computer for decision.

Third, having distributed the inferencing tasks, allow multiple single-board computers to cooperate on solving problems. Our model for this is a network of independent, cooperating agents, often able to act autonomously, but also able to interact directly with "peers" when unable to act alone. In short, we intend to implement distributed inference processing on a network of microcontrollers.

We have previously reported on a microprocessor inference engine, **TexMex**, which achieves very high inferencing speeds, and has only small memory requirements [Rodriguez, 1990; Rodriguez, 1993]. This report describes the adaptation of that engine to a networked multiprocessor environment. The two key innovations in the distributed inference engine are Distributed Facts, and Mobile Rules.

DISTRIBUTED FACTS

Prior work on cooperative expert systems has focused on the use of a "blackboard," a central store where multiple experts can write their results and read others' results. This model is reminiscent of the multicomputing model of the 1960's, in which a few mainframe computers shared a common memory. Then, as now, this central "hub" eventually becomes a bottleneck; while suited to a small number of expensive computers, it does not scale up gracefully to large numbers of inexpensive processors.

There is, however, another alternative. Modern multicomputers are distributed over a network, and computers on the ideal network can cooperate without competing for a limited resource.[2] We sought a comparable model for distributed inferencing, in which peers cooperate directly rather than through a central clearinghouse. This has led to the "advisory/consultant" model. Each processor works on its own set of problems. A processor can ask another processor directly for information needed to solve a problem; this "consultation" need not distract the second processor from its tasks. Furthermore, if a processor is aware that some knowledge in its possession is needed by others, it can "advise" others -- one processor, a group of processors, or all processors -- of new or changed facts.

The difference between the two distributed connection schemes is illustrated in Figure 3.

Fact Storage in TexMex 4

The single-processor expert system **TexMex** constrained all facts to be integer values.[3] Each fact has an evaluator function -- its "rule" -- which describes how that fact is determined when it is unknown. **TexMex 2** added the concept of "expiration time" for facts, and a temporal algebra for operating on time-valued facts. Fast forward chaining was achieved through the introduction of a "dependency list."

The distributed expert system is known as **TexMex 4**. An earlier version, **TexMex 3**, revealed the value of late-binding object-oriented programming for a distributed expert. A survey [Rodriguez & Poehlman, 1996] revealed that the existing object-oriented extensions to the Forth language had inadequate late-binding performance, or were otherwise unsuited; we therefore developed a new "fast binding" objects package.

To extend the **TexMex** inferencing model to multiple processors, facts are deemed to have an "owner." Only the owner of a fact knows how to evaluate that fact when it is unknown; that is, only the owner posesses the rule for that fact. Associated with each fact is an "export list," the list of all processors (other than the owner) which use the value of that fact. This is the network equivalent of the dependency list.[4]

Backward Chaining

The process of backward chaining requires all processors to know the "owner" of every desired fact. A fact remains valid until its expiration time; therefore, the procedure to obtain the value of a fact is as follows:

a. If the fact has not expired, use the truth value "remembered" from the last evaluation. (Each processor independently caches the facts that it uses.)

b. If the fact has expired, and I am the owner, perform the evaluator function for the truth value and update the cache.

c. If the fact has expired, and I am NOT the owner, send a message to the owner asking for current truth value of the fact. This value, and the associated expiration time, is used to update the cache. (If the fact has expired, this may in turn prompt its owner to perform the evaluator function.)

Note that the concept of expiration time is crucial to the operation of the cache, since this is how cache consistency is enforced. The cache, in turn, is essential to reduce the amount of network traffic, and to speed the inferencing process.

There remains the problem of invalidating a rule before its expiration time. This is addressed by the forward chaining mechanism.

Forward Chaining

The process of forward chaining uses the "export list" information. Evaluation of a rule, to produce a new value for a fact, is performed by the owner of that fact, usually in response to some stimulus. The new fact is forward-propagated normally within that processor. If the fact is "exported," the following also takes place:

a. A network message is sent to the export list (currently one processor or all processors) to advise of the fact's new value and expiration time.

b. The cache in each receiving processor is updated.

c. This may, in turn, trigger a forward-propagation process within any or all of the receiving processors.

Note that forward chaining does not depend on expiration time for its correct function. A chain of inferencing may be initiated by an external event or a periodic trigger, and this inferencing will proceed across distributed processors to a conclusion or conclusions.

Both chaining techniques are combined in a single inference engine, whose logic is shown in Figure 4. Only two network messages are required for distributed inferencing:

ASK fact#
is used only in backward chaining; it is sent by one processor to request the current value of a fact.

TELL fact#, value, expiration time
is the reply to an ASK message; it informs the recepient(s) of the new value of a fact and its expiration time. This is the only message required to perform forward chaining.

MOBILE RULES

Facts -- representing the currently known "truths" -- can, through the mechanisms just described, be shared among distributed processors. But facts are only half of the knowledge base. The rules by which these facts are determined are also knowledge. A truly distributed system would also be able to share rules, i.e., allow rules to be transferred from one processor to another. This is particularly challenging in a network of dissimilar processors, such as ours.

The expert system **TexMex 2** represented rules as functions using direct-threaded code [Kogge, 1982]. This representation, employing a "thread" of function addresses is efficient but processor-specific. (Even identical CPUs may locate identical functions at different addresses.) For mobile rules a a machine-independent and address-independent representation is required.

Our distributed expert, **TexMex 4**, represents rules as token-threaded code. Tokenized representations are common when processor-independence is required; for example, UCSD Pascal's P-code, the IEEE Open Firmware Standard [IEEE, 1994], or the Java language.

Rules are written in a new language, ITL (Inferencing Token Language), which is compiled to tokenized form. ITL is essentially a token-threaded subset of Forth: it is a stack-oriented, postfix language. Unlike many tokenized languages, user-defined functions (i.e., rules) expand the token set. The kernel tokens are encoded as 8-bit values for economy, but the token set also support a 16-bit expansion, for a total of 4096 possible functions. Unlike Forth, the arithmetic and logical operators of ITL operate on time-valued data, using the temporal algebra devised for **TexMex 2** [Rodriguez, 1993].

The price of using a token interpreter, rather than a direct-threaded Forth interpreter, is speed. Our tests have shown that the token-threaded inference engine of **TexMex 4** achieves about half the inferencing speed of the direct-threaded **TexMex 2**.

Rule Mobility

Rules are characterized in two ways (Figure 5): private vs. public, and bound vs. unbound.

Private rules are known only to one processor, and are used exclusively for that processor's internal inferencing. The facts produced by this rule cannot be exported; other processors are unaware of the rule's existence. Public rules, on the other hand,

are known to all processors on the network. Public rules may be the object of ASK and TELL messages. All distributed inferencing involves public facts.

A public rule may be bound or unbound. A bound rule is one that must execute on a specific processor. This might be because the rule is written in processor-specific form (e.g., machine code); the rule might use an input which is only available on one particular processor (e.g. an A/D converter); or the rule might use a private fact. Although the result of the rule (the fact) is publicly known, the rule itself cannot be relocated to another processor.

Rules which are not so restricted are called unbound rules. The essential characteristics of unbound rules are that a) they use only public facts in their evaluation, and b) they are written in the processor-independent ITL. Unbound rules can be executed by any processor, and are free to be relocated over the network.

Three network messages support rule transfer across the network.

DEFINE-RULE rule#, definition
is sent by the current owner of a rule, to transfer that rule to a different processor. The message includes the complete ITL definition of the rule.

I-OWN rule#
is sent by the new owner of a rule, to acknowledge that the rule was successfully received and installed. This acknowledges a DEFINE-RULE message. It also notifies all other processors on the network that the rule has changed ownership.

WHO-OWNS rule#
is an optional message, which can be sent by any processor to determine the current owner of a rule.

Additional network messages are being considered; for example, a message to solicit ownership of a rule may be useful, since it would allow a lightly loaded processor to "adopt" a rule it frequently needs.

PERFORMANCE

Inferencing

To measure logical inferences per second, a test problem must be posed which requires a large number of rule firings, without requiring external inputs. We have used the Towers of Hanoi problem as a standard benchmark. For a baseline, the Towers of Hanoi test was run on a single processor, on both a desktop PC and the 68HC16 microprocessor chosen for the FN accelerator project. Using the **TexMex 2** inference engine, we achieved

2000 LIPS on a 33 MHz 80386
370 LIPS on a 16 MHz 68HC16

For the distributed test, we used a "worst case" distribution of the rules across three processors. This distribution ensured that no processor could fire more than one rule, without needing a fact from some other processor. Using the **TexMex 4** inference engine, we achieved

31 LIPS on three 16 MHz 68HC16s

A study of the message traffic during the three-processor solution reveals that -- as expected -- the limiting factor in solution time is not inferencing speed, but network speed. Our "low cost" network is a token passing ring operating at 28.8 Kbaud. Solving an 8-disk problem requires 2040 packets of 15 bytes each to be sent over the network. Including best-case network overhead, this accounts for 12 seconds: nearly three-quarters of the measured solution time of 16.6 seconds.

Process Control

To evaluate the performance of the expert system as a process controller, we looked at the subproblem of terminal voltage control. The expert system was compared with the performance of a human operator. We examined three figures of merit for terminal voltage control:

1. Response time, i.e., how quickly the terminal can be charged (or discharged) to a new voltage. When charging to a higher voltage, the human operator would occasionally outperform the expert system. One reason is that the human operator could disregard one of the safety rules built into the expert system, a limit on positive slew rate, which is intended to limit sparking. Even with this "handicap," however, the expert system could often equal or exceed the performance of the human operator.

When discharging to a lower voltage, the expert system was clearly superior: perhaps because this operation is rarely required, and the human operator has had few occasions to practice it.

2. Overshoot. When charging to a higher voltage, the operator (human or computer) may overshoot the desired voltage and then have to correct. This is potentially hazardous when operating near the maximum voltage of the accelerator, and so reducing overshoot is a priority. By this measurement, the expert system is clearly superior to the human operator. We normally achived overshoot of less than .05 MV, which is nearly zero, given the measurment resolution of .01 MV.

3. Regulation. Even in the "steady state," Tandem accelerators exhibit a substantial drift. This is also a concern when operating near the maximum voltage, since an upward drift of .25 MV when unattended is by no means uncommon. The human operator must frequently direct his attention elsewhere; for this reason, special regulator devices were needed on the unautomated FN accelerator. The expert system, able to maintain constant vigilance, was able to maintain terminal voltage within .03 MV of the desired voltage, in long-term unattended operation with the external regulator disabled.

CONCLUSION

We have devised a novel mechanism for distributing the knowledge and the inferencing load of an expert system across a network of small microprocessors. This network has been shown to be effective in a complex real-time control task, namely, control of an FN Tandem accelerator.

At present, the act of relocating a rule to a different processor is strictly manual (under operator control). It would be desirable to automate this, allowing the expert system itself to decide when a rule should be transferred. One approach could be to optimize placement of all rules so as to minimize network traffic. Another approach would be to transfer rules from heavily-loaded processors to lightly-loaded processors. These represent antagonistic objectives; e.g., except for the presence of "bound" rules, network traffic is minimized by putting all rules into one processor.

As an initial step, the inference engine has been instrumented to record frequency of use and evaluation time for each rule, and total time spent by each processor in the inference engine (i.e., total inferencing load of each processor). Future extensions can use this data to implement automatic load-balancing mechanisms.

ACKNOWLEDGMENTS

The authors would like to acknowledge the staff of the McMaster Accelerator Laboratory, without whose assistance, this study would not have been so successful. Thanks also go to Dr. David Chettle, Laboratory Director, who allowed the FN to remain operable (before decommissioning) several more months than planned, which enabled the timely completion of this work.

REFERENCES

Institute of Electrical and Electronics Engineers (IEEE). Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices. IEEE Standard 1275-1994. New York: IEEE, 1994.

Kogge, Peter M. "An Architectural Trail to Threaded-Code Systems." IEEE Computer (March 1982): 22-32.

Lind, P. C. and W. F. S. Poehlman. "Design of an Expert System- based Real-time Control System for a Van de Graaff Particle Accelerator." In Proceedings of AIENG '92 - Applications of Artificial Intelligence in Engineering, July 1992.

Rodriguez, Bradford J. "Rules Evaluation Through Program Execution." In Proceedings of the 1990 Rochester Forth Conference on Embedded Systems, by the Institute for Applied Forth Research. Rochester, NY: Institute for Applied Forth Research, 1990, 123-125.

Rodriguez, Bradford J. "Fast Inferencing of Static Rule Sets." In Proceedings of the 1993 Rochester Forth Conference on Process Control, by the Institute for Applied Forth Research. Rochester, NY: Institute for Applied Forth Research, 1993, 102-106.

Rodriguez, Bradford J. "The Asynchronous Token Ring." In Proceedings of the 1995 FORML Conference, by the Forth Interest Group. Oakland, CA: Forth Interest Group, 1995.

Rodriguez, Bradford J. and W. F. S. Poehlman. "A Survey of Object-Oriented Forths." ACM SIGPLAN Notices 31, no. 4 (April 1996): 39-42.

NOTES

(1) Under partial support from the Natural Sciences and Engineering Research Council of Canada.

(2) A single physical network, common to all processors, resembles a blackboard. Competition is reduced when the network is partitioned into subnetworks.

(3) This constraint was retained in **TexMex 3**, but **TexMex 4** has the potential to handle different fact types.

(4) The token ring network [Rodriguez, 1995] developed for the FN control system allows broadcast messages; it is therefore more efficient to restrict the export list to be either a single processor or "all processors."

FIGURES