Programming Language and Computer Architecture Research at ICOT

John S. Conery

University of Oregon
Eugene, Oregon, USA

June 22, 1987

I spent the first three weeks of June, 1987, as a visiting researcher at ICOT. I met with members of the Fourth Lab, and discussed their projects on machines and languages for fifth generation computer systems. I was also able to share with members of ICOT some ideas on a project I started at the University of Oregon. This report is a summary of these meetings and presentations, with a few comments on ICOT and the direction of fifth generation computer research.

My first day at ICOT, June 1, was the fifth anniversary of the research institute. My wife and mother-in-law and I were all invited to attend the celebration held that night in the annex of the Mita-Kokusai Building. We were impressed by how outgoing and friendly everyone was; we talked about everything, from technical subjects, to gossip, to comparing life in the United States to life in Japan. This gathering seemed to set the tone for the following three weeks, which were filled with many more friendly and informative discussions.

Presentations by ICOT Members

PSI-II

The first presentation I attended, on Tuesday morning, was given by Katsuto Nakajima. He described the architecture of the PSI (personal sequential inference machine) and the improvements that were made in the design of PSI-II. The goals of the PSI project are to do for Prolog what the Symbolics LISP machines did for LISP: provide fast execution, in an advanced programming environment tailored to support this one particular language, in a single-user machine. The two main goals for PSI-II are, one, to take advantage of advances in Prolog implementation technology, and two, to use newer semiconductor technology in order to build a smaller, less expensive machine. In addition, the processor and memory of the PSI-II are expected to be used as a PE in the Multi-PSI. The PSI-II is from three to ten times as fast as the PSI-I, depending on the type of program.

To meet the first of these goals, the new machine is based on the Warren Abstract Machine (WAM), a proven technique for implementing sequential Prolog.

The PSI-II instruction set is a microprogrammed implementation of the WAM instruction set, whereas the PSI-I used a more complicated microprogram that interpreted programs that were close to their original source form. In an experiment to gauge the effectiveness of the WAM instruction set *vs.* the original PSI-I instruction set, a WAM interpreter was microprogrammed on the PSI-I hardware. The WAM instruction set ran selected programs twice as fast, on the average. Performance figures given in the paper by Nakashima and Nakajima show that over half of the improvement of PSI-II over PSI-I is accounted for by the use of the WAM instructions. The relative efficiency of the WAM is attributed to compilation techniques that were not possible with the older instruction set. Other implementation techniques (also used in PSI-I) include the use of tagged data, a 20KB (4KW) write-back cache, and specialized stack manipulation operations designed for Prolog stacks (more than two stacks must grow in the same address space).

To meet the second goal, the PSI-II uses a combination of new, commercially available circuits (such as 1Mb RAM chips), along with custom LSI gate-array circuits used in the processor. The resulting processor fits on three PC boards, and the entire machine, which includes up to 160MB RAM, fits in a small cabinet roughly the size of a Symbolics 3645. The cycle time of the PSI-II processor is 166.7ns, a rather modest increase of 20% over the cycle time of the PSI-I.

The PSI-II has certainly met most of the original goals of the PSI project. The execution speed is orders of magnitude faster than the Prolog systems most of us learned the language on, and the system software provides a rich environment for program development (described in another section, below). The goal of working as a building block of the Multi-PSI is also achieved. When the PSI-II is used as a node in the Multi-PSI, the microprogrammed interpreter will be changed (it is in a writable control store) to an interpreter for the parallel language KL1.

The PSI-II represents one point in an interesting set of implementation strategies:

- Implement the WAM via software on a host architecture; an example is Quintus Prolog on the Vax.
- Implement the WAM in microcode on a microprogrammable host, without changing the hardware. A paper by Gee, Melvin, and Patt at the last ICLP described an implementation where the Vax 8600 instruction set was extended to include WAM instructions.
- Microprogram the WAM on hardware designed to support WAM operations; the Berkeley PLM probably fits in this category.
- Microprogram the WAM on a host machine that has very little extra hardware support. The PSI-II is in this category; the extra hardware is for extracting and comparing tag fields.
- Compile Prolog directly to a lower level host instruction set, either for a machine like the 68000 or a RISC machine (the LOW-RISC project is an example of the latter).

Making meaningful comparisons based on experience from these projects will be very difficult, given the wide range of goals and execution environments, but there should be a wealth of experience to draw on for the next designer of a Prolog machine. The designers of the PSI-II feel that their approach shows that a modest investment in hardware can lead to a significant improvement in speed. In this case the just over half of the improvement of PSI-II over PSI-I is due to a better instruction set, and the remainder is due to the addition of a small bit of hardware for tag checking.

PIM and Multi-PSI

The PIM (for parallel inference machine) is one of the long range goals of the fifth generation computer project. The PIM project is now in the middle of its intermediate phase. The main goal for this phase is the construction of a prototype machine, with approximately 100 processors, capable of 5 MLIPS processing speed.[1] At the beginning of the intermediate phase, a new project was started. Recognizing the need for stable hardware to test the operating system and application software that will eventually run on PIM, plans were made to develop a testbed for parallel software. This is the role of the Multi-PSI, which is a network of 64 PSI-II processing elements.

Dr. Atsuhiro Goto gave a presentation on the history of PIM and the plans for development in the intermediate phase. In the initial phase, several alternative machine styles were investigated, and some were implemented in hardware. These projects were PIM-R (a reduction machine), PIM-D (a dataflow machine), and Kabu-Wake (a multi-sequential machine, with a novel task assignment method). In the next phase, the intermediate PIM will consist of up to 100 processors, each slightly more powerful than the PSI-II, connected in a network of local clusters. Each machine will have its own local memory, and machines within a cluster will also have access to a shared global memory. Clusters will be connected via a high speed network. Each processor will execute KL1-b as its instruction set (KL1-b, the "base language" of KL1, is an abstract machine instruction set; I will describe it in more detail, below).

[1] To get a feel for what this number represents, the PSI-II runs about 150 KLIPS on its small benchmarks; the designers of the Berkeley PLM hope to achieve about 350 KLIPS, and the fastest "mainframe" execution I am aware of is just under 1 MLIP.

Dr. Kazuo Taki gave the presentation on Multi-PSI. The goal of this system is not to be a prototype PIM, in the sense that it is to test PIM hardware design ideas, but to be a stable environment for running and evaluating parallel software. The important attributes of PIM that must be supported by a software evaluation workbench are: it must be a large-scale MIMD machine, without shared memory; it should have dynamic load balancing, and a large amount of local memory at each node. The Multi-PSI clearly meets these requirements. The first version of Multi-PSI, which is now operational, consists of 6 PSI-I machines connected via local network controllers. Each machine is a complete PSI-I, with I/O devices, keyboard, and monitor. The second version of Multi-PSI will be running later this summer. It will have 64 nodes, each of which is a PSI-II processor with local memory. This machine will not be a shared-memory

machine; instead, messages will be passed via network controllers in each node. The interconnection network will be an 8×8 grid, with each PE connected to four neighbors. The microprograms in the PSI-II nodes of the Multi-PSI will be changed, so the instruction set is KL1-b, instead of the WAM instruction set of the sequential PSI-II.

The software projects that will be investigated using Multi-PSI are:

- KL1, the application language for the machine. This "second generation" parallel language is basically flat GHC, with constructs for object oriented programming and pragmas for process allocation.
- PIMOS, the operating system for PIM (described in a later section).
- A dynamic load balancing mechanism.

Initially, program development will be on a host PSI-II, used as a front-end to Multi-PSI.

I think the plan to use Multi-PSI as a testbed for PIM software is an excellent idea. Aside from the obvious benefits of giving programmers a chance to run their programs before the final system is ready, it will give members of each project valuable feedback on their projects. A while ago, on a visit to Berkeley, I talked with some of the people who had developed some VLSI design tools. They said that one thing that helped them tremendously was that another research project at Berkeley (I think it was the RISC project) used the tools as they were being developed, and provided feedback on how they worked. This sort of symbiotic relationship should occur here as well: the software group will have a machine to run their programs on, and the hardware group will have a large body of real software to use for performance analysis. The existence of two large groups that can interact in this manner is one of the greatest strengths of ICOT, I feel.

A word of caution is in order here: the feedback can be negative, as well as positive. By this I mean the feedback from one project to another could lead to a development path that settles into a static set of concepts. At the risk of mixing metaphors, the process might be like a hill-climbing algorithm that is drawn toward a local maximum instead of the best long-term solution. Is it possible that software developers will create programs that run well on the hardware they have to work with, and then the hardware developers will react by building machines that run those programs better? Instead of a "positive feedback system" that continues to move toward better and more advanced concepts, there might be a "negative feedback system" where input from one group acts as a damper on the other.

GHC

I had a chance to meet with Kazunori Ueda, to talk about his Guarded Horn Clause (GHC) language. By the time we met, I had had time to experiment with the GHC compiler on the DEC-20, and I had written a couple of small programs.

The current implementation strategy is to use what is called "flat" GHC. In a procedure such as

```
p(X,Y) :- X > 10 | q(X,Y).

p(X,Y) :- X <= 10 | Y = 10.
```

the guards of the two clauses (the goals before the commit operator |) are evaluated in parallel. In a flat language, the guards can contain only calls to system predicates, not to user-defined goals. This restriction allows the system to avoid maintaining OR-parallel environments for guard evaluation.

The people at ICOT now regard FGHC as an intermediate level language, not the high level language that users will eventually write applications in. A good example of the use of FGHC as a base language is seen in Ueda-san's technique for compiling nondeterministic search into FGHC programs. One of the drawbacks of writing programs in FGHC is that there is no "don't know" nondeterminism in the evaluation of the clauses; clauses that generate multiple results in Prolog must commit to one value in FGHC. However, it may be possible to compile the clauses into an equivalent GHC program that gathers all results, similar to Prolog's `bagof`, where each result is computed in parallel.

There are two drawbacks to this approach. The first (described in Ueda's paper) is that not all nondeterministic search programs can be transformed with the techniques developed so far. In order to aviod the problems of shared variables in OR-parallel environments, the programs must generate ground structures at each step. This class of problems is larger than it appears at first, however. By rearranging the order of the body, the "output unification" can be placed where it will do the most good (the binding of `Y` to `10` in the second clause in the example above is output unification; in GHC, the head cannot be `p(X,10)` since unification of the head is not allowed to return bindings to the caller of the clause). Instead of generating partial structures at each step, which is a common Prolog technique, the compiled clauses can generate complete structures from the bottom up.

The second drawback is that the transformed clauses are now AND-sequential. This is at first reminiscent of the OR-parallel search component of Parlog. However, it should be more efficient, since the need for multiple binding environments has been compiled away. There is another problematic aspect of pure OR-parallel search: in some circumstances, the same problem will be solved many times. For example, if the goal is `p(X), q(Y)`, where both variables are unbound, and there are n solutions to `p(X)`, an OR-parallel system may end up solving `q(Y)` n different times, each time performing the exact same computation. These are the situations where the AND/OR Process Model and Restricted AND-Parallel models would exploit AND parallelism, so that each goal is solved once and the results combined.

I think the idea of compiling higher level languages into GHC is very promising, and worth pursuing further. Two of the attributes of logic programming that distinguish it from functional programming are nondeterminism and the logical variable. The committed choice AND-parallel systems and the AND/OR Process Model each do a good job exploiting one of these features, while ignoring or doing a poor job of handling the other. Compiling a higher level language into GHC may be a good way to unify the two abstract models so that both features of logic are handled well.

KL1-b

Yasunori Kimura gave a presentation on KL1-b, the base language of the KL1 programming language. KL1-b is an abstract machine, in the style of the Warren Abstract Machine, that can be used for the family of flat parallel committed choice languages. KL1-b will be implemented in microcode on the PSI-II processors of the Multi-PSI, and will also be the base language of the nodes of the intermediate stage PIM machines.

The principle execution unit at this level is the goal. The guards of the clauses (which include the heads) in a procedure are compiled into instructions that are executed when the goal is selected for execution. A scheduler on each processor maintains a queue of goals, and decides which goal will be selected at any time.

When a goal is made the current goal, the processor branches to the code for the guards of the goal, and executes these instructions sequentially. If a guard tries to bind a global variable, or fails in unifying the head of a clause, or the execution of a guard goal fails, the processor branches to the next alternative in the goal. If a guard is successfully executed, however, the output unifications in the body (after the commit operator) can be executed, and goals for the other procedures can be scheduled. If no guard in the procedure succeeds, the goal is suspended.

An important feature of this machine is that it avoids busy-waiting. When a goal is suspended because it must wait for another goal to bind a shared variable, the variable is bound to a "hook" to the blocked process. The process will not be placed in the ready queue until another process binds the variable.

Since I am also currently working on an abstract machine (in my case, it will be for the AND/OR Process Model), I found this work very interesting. I was able to offer some detailed suggestions to Kimura-san, which I hope will be constructive:

- The code size of compiled programs might be reduced if the clause indexing is fully implemented. If it is, the `Label` field of the `wait` and built-in predicate instructions can probably be eliminated.

- An example in the paper shows a goal that is "hooked" on two variables. Currently, such a goal is rescheduled when just one of the variables is bound, only to immediately suspend again when it is restarted. Some data presented in a paper by Kishimoto *et al* at Fujitsu shows a large number of goals suspend immediately like this. Perhaps the overhead of scheduling and then suspending these goals can be eliminated by not scheduling a goal until all of the variables it is waiting on are bound.

- When a goal is resumed, processing starts over again with the code for the first clause. I think this could be made more efficient, as well, so that only clauses that could possibly succeed are retried when the goal is resumed. As things are described now, even clause heads that fail to unify are tried again when the goal is rescheduled, since there is no way to separate the clauses within a procedure.

SIMPOS and PIMOS

SIMPOS and PIMOS are, respectively, the operating systems for the sequential inference machine and parallel inference machine. I was given a presentation on SIMPOS by Hiroyoshi Ishibashi. He also gave me a demonstration of the current version of the system on the PSI.

SIMPOS is written in ESP, a sequential logic programming language that includes object oriented programming constructs. Version 2.5, released last summer, has over 1400 class definitions, 18,000 procedures, and 230,000 lines of code. Version 3.0, which will be larger yet, will be released later this summer. The major goals for SIMPOS were to provide a rich programming environment of logic programmers. A great deal of work has gone into the user interface (window system and other graphics), editors, debuggers, and so on. Little or no work seems to have been done on memory management (paging) or task scheduling, the "performance" issues that are the concerns of more traditional operating systems.

I had hoped to get a chance to experiment with pemacs, the editor for SIMPOS, but I didn't have enough time. I was intereseted in seeing how edit macros were written in ESP, instead of the LISP used in various emacs implementations. I also wanted to get a chance to evaluate the user interface. One comment I have, based only on the short demonstration I saw, is that it would be a good idea to give more feedback to the user about what is happening in the system. Whenever the mouse moves, the cursor position should be updated; whenever a time consuming operation is taking place, a message telling the user what is happening should be displayed on the screen.

The PIMOS presentation was by Hiroyuki Sato. PIMOS will be one of the major parallel software projects that will be developed with Multi-PSI. It will be written in KL1.[2] Two goals for PIMOS that I agree with are, one, to give the user a single system image, hiding the fact that the underlying machine is a multiprocessor; and, two, to try to write the system in a "more logical style" than was used for SIMPOS, which relied heavily on the metalogical extensions of ESP.

KAPPA

The final presentation, by Kazumasa Yokota, was on KAPPA, a knowledge representation and database query system based on an extension to the relational model. This project is also aimed at solving large applications efficiently. In this case, there are two existing applications that KAPPA is being designed to support efficiently. Although there is clearly a danger in designing what is intended to be a new, general purpose, language based on the requirements presented by just two applications, Yokota-san and his group feel that the two applications are large enough and sufficiently general that this approach is warranted. In addition, one of the applications (a massive dictionary, for both Japanese and English, with over a million words and nearly a million semantic relationships) will be the heart of many other knowledge-based applications, so it is worthwhile in itself to optimize this part of the system.

The formal model underlying KAPPA is the nested relational model, in

which values in domains may be structures as well as atomic items. An example used by Yokota-san is:

```
parent^{jack,betty} * child^{john,mary,cathy}
```

This is a tuple from a relation with two domains, named `parent` and `child`. The values in these domains are sets; the intended meaning is that `jack` and `betty` are the parents of the people named in the `child` domain. Representing this information in Prolog or a flat relational database would require six tuples

[2]KL1 has four components: KL1-u is the "user" language, which will be used for applications like PIMOS. KL1-u will be compiled into KL1-b, the instruction set of the abstract machine.

instead of one, and rules based on this information (such as `sibling`) would be correspondingly more complex.

The KAPPA group has developed a computational model based on nested relations, as a means for expressing queries. Unfortunately, I am not very familiar with this area of research, so I cannot offer any constructive criticisms of the project. Yokota-san and the group are familiar with other projects, in the U.S. and Europe, that have computational formalisms for extensions to the relational model. I found the examples of the queries to be quite clear and easy to understand, and my guess is that the extra information provided by grouping items in a set could lead to more efficient processing. For example, the rule for `sibling` in a Prolog database shows, at the source level, two accesses to the `parent` relation, but in the nested relational model all the information is present in just one access.

My Presentations

I gave two talks while I was at ICOT. The first was to an audience of ICOT members, many of whom work in other labs. This talk was mostly on my research on binding environments for the AND/OR Process Model (the paper this talk was based on will be presented at the 1987 Symposium on Logic Programming). Most of the questions from the audience were on the subject of OR parallelism. The emphasis at ICOT has been, so far, on committed choice AND parallelism, so I gave some of my views on why OR parallelism was important.

The second talk was presented to the PIM working group. This group consists of researchers in the Fourth Lab who are working on the PIM and associated projects, plus researchers from industry and universities. This talk was more general. I talked a little bit about Oregon and the University, gave a brief history of my research, and finally talked about some of the projects I want to work on in the future.

One of the things I intend to work on is a system that combines logic programming and object oriented programming. Before I left Oregon, I had written a Prolog meta-interpreter that implemented my ideas for classes and objects, and in my second week at ICOT I wrote a version for DEC-10 Prolog. For lack of a better name, the system is called OOPS (for object oriented prolog system).

The system is based on the notion of an *object clause*, which is a non-Horn clause. Future work will have to define a semantics for programs written with

these clauses, and a formal definition of the inference rule used when calling an object clause (the rule is similar to, but not identical to, Monteiro's rule for distributed clauses). I wrote a brief paper describing object clauses and distributed it to various people around ICOT who are interested in object-oriented programming.

I also wrote a program that illustrates the style of programming that is possible. This program is (yet another) implementation of the eight queens problem, where each queen is an object. When a queen is placed in a square, it sends messages to other queens telling them to stay out of certain squares. This style of solution will lead to fewer backtracking choices than a Prolog solution.

Visit to Fujitsu Labs

On June 10 I went to Fujitsu Labs in Kawasaki with Yasunori Kimura. I gave a short presentation on my work, listened to a presentation by Mitsuhiro Kishimoto on a paper he will present at the Symposium on Logic Programming, and took a tour of the labs. The work by Kishimoto-san and his colleagues is an evaluation of large application programs written in FGHC, and executed by the KL1-b virtual machine. The data they collected was very interesting, and they made some insightful comments on the nature of the programming language and strategies for implementing it.

General Comments on ICOT

The working environment at ICOT seemed to me to be very conducive to research. Each person seems to have a large chunk of a project to take responsibility for, and at the same time there are many people working on projects that are closely enough related that meaningful discussions can take place. As I mentioned above, I think the large number of people working on similar projects is a major strength of the lab.

ICOT is a unique combination of different styles of organizations. It has elements of an advanced product development department of a corporation, an applied research lab, and a university research department. Somehow, these components work together quite well. One can find projects with varying degrees of immediate "real world" application. Examples at the two extremes taken only from the projects I have described here are the PSI-II project and the development of GHC. PSI-II will be a commercial product of the Mitsubishi corporation, and the goals for the development of PSI-II were basically to improve (and lower the cost of) the PSI-I. The development of GHC as a base language for future languages, and techniques for compiling future higher level languages into GHC, is a very long range project, and the kind of thing one could find in a university research lab.

It was interesting to observe the interactions between people. Even though I don't understand Japanese, so I couldn't tell exactly what was being said, the small groups that would gather to discuss things were just what I would expect to find at any lab in the U.S. At any time, there will be some people hunched in front of terminals, a group having a serious (and maybe animated)

technical discussion, and another group having a far less serious talk. The physical environment is also conducive to work. I did not find the open room to be distracting at all when I was trying to get work done.

I knew before I got here that even though my research and the projects at ICOT had the common theme of parallel execution of logic programs, we had very different ideas about how this should be done. In spite of the differences, it was very good for me to come here. In explaining my views, listening to the presentations, and asking questions, I gained a new perspective on my own projects. If the researchers at ICOT benefitted in a similar fashion, even a fraction as much as I did, I will be pleased.

Thank You

I would like to thank Dr. Kazuhiro Fuchi, the director of research at ICOT, and Dr. Shun-ichi Uchida, head of the Fourth Lab, for inviting me to visit ICOT. This was a tremendous opportunity for me, not only to learn about the current status of research at ICOT, but also to have a forum to discuss my own projects. The members of the Fourth Lab went out of their way to make me feel at home. In particular, Kazuaki Rokusawa, who was assigned to be my host, went to great lengths to make sure I had everything I needed, from the necessary computer accounts to plans for lunch. For the sake of future visitors to ICOT, I hope Rokusawa-san misses a few more research meetings!

During my three weeks at ICOT, nearly everybody, at one time or another, helped me in some way. In addition to those who took time out from their schedules to make presentations on their research, I would especially like to thank the following people:

Dr. Kazuhide Iwata, of the research planning department, reserved our hotel and made other financial arrangements. Tokyo, in June of 1987, was a pretty intimidating place for people whose salaries were paid in American dollars. The arrangements made by Iwata-san set our minds at ease, and we were able to enjoy our three weeks in Japan without worry.

Noboyuki Ichiyoshi, when he returned from Australia half-way through my visit, was also assigned to be my host. Together, he and Rokusawa-san made sure I kept all my appointments, and in general made sure I had everything I needed during my visit. Thank you also for the introduction to *shochu*.

Yasunori Kimura, whose parent company is Fujitsu, arranged my visit to Fujitsu Labs in Kawasaki. In the past few years, many people from Fujitsu have visited me in the United States, and I was glad to be able to renew acquaintances with them and get updates on their projects.

Akira Matsumoto, whose desk was next to mine, and Shigeyuki Takagi, researcher and local Unix wizard, were most helpful in setting up my working environment on the DEC-20 and Balance 21000 machines, and showing me the ins and outs of running programs at ICOT.

References

Goto, A. and Uchida, S. *Toward a High Performance Parallel Inference Machine – The Intermediate Stage Plan of PIM.* ICOT TR-201, Sept. 1986.

Kimura, Y. and Chikayama, T. An abstract KL1 machine and its instruction set. To appear in *Proceedings of SLP '87*, Palo Alto, Sept. 1987.

Kishimoto, M., Hosoi, A., Kumon, K., and Hattori, A. An evaluation of the FGHC via practical application programs. To appear in *Proceedings of SLP '87*, Palo Alto, Sept. 1987.

Nakashima, H. and Nakajima, K. Hardware architecture of the Sequential Inference Machine: PSI-II. To appear in *Proceedings of SLP '87*, Palo Alto, Sept. 1987.

Sato, M., Shimizu, H., Matsumoto, A., Rokusawa, K., and Goto, A. KL1 execution model for PIM cluster with shared memory. *Proceedings of the Fourth International Conference on Logic Programming*, Melbourne, May 1987.

Taki, K. The parallel software research and development tool: Multi-PSI System. *France-Japan Artificial Intelligence and Computer Science Symposium 86.*

Uchida, S. *Toward the Parallel Inference Machine.* ICOT TR-196, Aug. 1986.

Ueda, K. *Guarded Horn Clauses: A Parallel Logic Programming Language with the Concept of a Guard.* ICOT TR-208, Oct. 1986.

Ueda, K. Making exhaustive search programs deterministic: Part II. *Proceedings of the Fourth International Conference on Logic Programming*, Melbourne, May 1987.

Research Resume

John S. Conery
Department of Computer and Information Science
University of Oregon
Eugene, Oregon, USA

Ph.D. in Computer Science,
from University of California at Irvine, 1983.
M.S. in Computer Science,
from University of California at Irvine, 1979.
B.S. in Psychology,
from University of California at San Diego, 1976.

Current Research Interests:
Logic programming, parallel computer architecture,
implementing parallel logic programming languages.

Selected Publications:

J. S. Conery and D. F. Kibler. "Parallel Interpretation of Logic Programs", in Proceedings of the Conference on Functional Programming Languages and Computer Architecture, Wenthworth-by-the-Sea, NH, 1981. (introduced AND and OR parallelism, and an abstract model for executing OR-parallel programs)

J. S. Conery. "The AND/OR Process Model for Parallel Execution of Logic Programs." Ph.D. Thesis, UC Irvine, 1983. (extended the model of the Wentworth paper, and described a method for AND parallelism in nondeterministic systems)

J. S. Conery and D. F. Kibler. "AND Parallelism and Nondeterminism", New Generation Computing, 1985.

J. S. Conery. "Parallel Execution of Logic Programs." Kluwer Academic Publishers, Boston, MA. 1987. (A book containing most of the work from my Ph.D. thesis, plus an updated survey of recent work and implementation techniques.)

J. S. Conery. "Binding Environments for Parallel Logic Programs on Non-Shared Memory Multiprocessors", to appear in the Proceedings of the 1987 IEEE Symposium on Logic Programming, San Francisco, CA, 1987.

Professional Activities:

Chairman of the Technical Committee, 1985 IEEE Symposium on Logic Programming, Boston, MA.

Technical Committee, 1986 IEEE Symposium on Logic Programming, Salt Lake City, UT.