PROCEEDINGS OF THE INTERNATIONAL CONFERENCE
ON FIFTH GENERATION COMPUTER SYSTEMS 1992,
edited by ICOT.

Concurrent Logic Programming as a Basis for Large-scale Knowledge Information Processing

Koichi Furukawa

Institute for New Generation Computer Technology
4-28, Mita 1-chome, Minato-ku, Tokyo 108, Japan
furukawa@icot.or.jp

1 The Future of Information Processing

As the Fifth Generation Computer Systems project claims, the information processing field is pursuing knowledge information processing.

Since the amount of information being produced is increasing rapidly, there is a growing need to extract useful information from this information. The most important and promising technologies for information extraction are knowledge acquisition and machine learning. They include such activities as classification of information, rule acquisition from law data and summary generation from documents. For such activities, heavy symbolic computation and parallel symbolic processing are essential.

Combinatorial problems are another source of applications requiring heavy symbolic computation. Human genome analysis and the inversion problems are examples of these problems. For example, in diagnosis, it is quite easy to forecast the symptoms given the disease. However, to identify the disease from the given symptoms is usually not so easy. We need to guess the disease from the symptom and to verify the truth of that guess by further observation of the system. If the system is linear, then the inversion problem is simply to compute the matrix inverse. But, in general, there is no straightforward way to solve the inversion problem. There may be many candidates for any guess and this becomes even worse when we take multiple faults into account. Note that abductive reasoning, one of the most important reasoning processes for open-ended problems, is also characterized as a general inversion formalism against deduction.

Cooperative problem solving (or distributed AI) is another important direction for future information processing. Like human society, one feasible way of dealing with large-scale problems is for a number of experts to cooperate. To exchange ideas between experts, mutual understanding is essential, for which we need complicated hypothetical reasoning to fill the gaps in terminology between them.

These three examples show the need for heavy concurrent information processing in the field of knowledge information processing in the future.

2 The Role of Logic Programming

Logic programming provides a basic tool for representing and solving many non-trivial artificial intelligence problems.

1. As a knowledge representation tool, it can express situations without being limited to a closed world, as was believed until recently. The negation by failure rule makes it possible to express an open-ended world, which is essential for representing common sense and dealing with non-monotonic reasoning. Recently, a model theory for dealing with general logic programs which contains negation-by-failure literals in the body of clauses has been studied. The theory, called stable model semantics, associates a set of feasible models, natural extensions of least models, to each general logic program.

2. As an inference engine, logic programming provides a natural mechanism for computing search problems by automatic backtracking or by an OR-parallel search mechanism. Recent research results show the possibility of combining top-down and bottom-up strategies for searching.

3. As a syntactic tool for non-deductive inference, logic programming provides a formal and elegant formalism. Abduction, induction and analogy can be naturally formalized in terms of logic and logic programming. Inoue et al. [Inoue *et al.* 92] showed that abductive reasoning problems can be compiled into proof problems of first order logic. This means that non-deductive inference problems can be translated into deductive inference problems. Since abduction is a formalization of a kind of inversion problems,

this provides a straightforward way to solve such problems.

There was a common belief that logic and logic programming had severe restrictions as tools for complex AI problems that require open endedness. However, recent research results shows they are expressive enough to represent and solve such problems.

3 The Role of Concurrent Logic Programming

Concurrent logic programming is a derivative of logic programming and is good for expressing concurrency and executing in parallel. From a computational viewpoint, concurrent logic programming only supports AND-parallelism, which is essential for describing concurrent and cooperative activities.

The reason why we adopted concurrent logic programming as our kernel language in the FGCS project is that we wanted simplicity in the design for our *machine language* for parallel processors. Since concurrent logic programming languages support only AND-parallelism, they are simpler than those languages which support both AND- and OR-parallelism.

We succeeded in writing many useful and complex application programs in KL1, the extension of our concurrent logic programming language, FGHC, for practical parallel programming. These include a logic simulator and a router for VLSI-CAD, and a sequence alignment program in genome analysis. These experimental studies show the potential of our language and its parallel execution technology.

The missing computational scheme in concurrent logic programming is OR-parallelism. It comes from the very fundamental nature of concurrent logic programming language, that is, the committed choice mechanism. OR-parallelism plays an essential role in many AI problems because of the requirement for searching. A great deal of effort has been made to achieve OR-parallel searching in concurrent logic programming by devising programming techniques. We developed three methods for different applications: a continuation-based method for algorithmic problems, a layered stream method for parallel parsing, and a query compilation method for database problems. These three methods cover many realistic applications. Therefore, we have almost developed OR-parallelism successfully. This means that there is a possibility of building parallel deductive databases in concurrent logic programming.

One of the most significant achievements using the query compilation method is a bottom-up theorem prover, MGTP [FujitaHasegawa 91]. This is based on the SATCHMO prover by [Manthey 88]. MGTP is a very efficient theorem prover which utilizes the full power of KL1 in a natural way by performing only one way unification. SATCHMO has a restriction in the problems it can efficiently solve: range-restrictedness [Furukawa 92]. However, most real life problems satisfy this condition and, therefore, it is very practical.

We succeeded in computing abduction, which was translated to a theorem proving problem in first order logic, by using MGTP. We succeeded in solving a very important class of inversion problems in parallel on our parallel inference machine, PIM.

4 Conclusion

Concurrrent logic programming gained its expressive power for concurrency at the sacrifice of Prolog's search capability. By devising programming techniques we have finally almost recovered the lost search capability. This means that we now have a very expressive parallel programming language for a wide range of applications.

As an example, we have shown that the technique enabled realization of an efficient parallel theorem prover, MGTP. We have also shown success in deductively solving an important class of inversion problems, formulated by abduction, by the theorem prover.

Our research results indicate that our concurrent logic programming and parallel processing based technologies have great potential for solving many complex future AI problems.

References

[FujitaHasegawa 91] H. Fujita and R. Hasegawa, *A Model Generation Theorem Prover in KL1 Using a Ramified-Stack Algorithm.* In Proc. of the Eighth International Conference on Logic Programming, Paris, 1991.

[Furukawa 92] K. Furukawa, Summary of Basic Research Activities of the FGCS Project. In *Proc. of FGCS92*, Tokyo, 1992.

[Inoue *et al.* 92] K. Inoue, M. Koshimura and R. Hasegawa, Embedding Negation as Failure into a Model Generation Theorem Prover. To appear in *CAD-11: The Eleventh International Conference on Automated Deduction,* Saratoga Springs, NY, June 1992.

[Manthey 88] R. Manthey and F. Bry, *SATCHMO: A Theorem Prover Implemented in Prolog.* In Proc. of CADE-88, Argonne, Illinois, 1988.