Or-Parallel Execution Models for Non-deterministic Programming Languages

Ravikanth Ganesan Kannan Govindarajan
Department of Computer Science
State University of New York at Buffalo
Amherst NY 14260
E-mail : govin-k@cs.buffalo.edu

January 31, 1994

1 Or-Parallel Execution Models

A natural way to represent the state of parallel execution of a non-deterministic program is by means of an or-parallel tree. [GJ90] and [GJ93] identify three important criteria that underlie the design of or-parallel implementations based upon the or-parallel tree. These are:

1. Constant-time access to variables.
2. Constant time task creation.
3. Constant-time task switching.

These three criteria correspond to three fundamental operations of any non-deterministic language, which are:

1. Procedure Call which involves environment creation.
2. Assignment and Parameter transmission which involves variable access and binding.
3. Resumption after success or failure which involves task switching.

The main result of [GJ93] is that all the three criteria cannot be satisfied by any or-parallel execution model based on a finite number of processors and unbounded memory.

1.1 Or-Parallel Execution of Logic Programs

Non-determinism in logic programming arises because many clause heads may unify with some procedure call (or goal). Logic programming languages are *single assignment languages*, and the main negative result of [GJ93] holds for single assignment languages. Since multiple assignment languages are atleast as complicated as single assignment languages, the result holds true for any non-deterministic language whose execution model depends on searching some or-parallel tree. The following are some of the models that have been proposed in the literature.

1. The Binding Arrays Method [W84] [W87a] and [W87b].
2. The Randomised Parallel Backtracking Method [JAM88].
3. The Directory Tree Method [HC83] and [HC86].
4. The Hashing Window Method [B84].
5. The Time Stamping Method [T89].

The main difference between the various or-parallel execution models for logic programs is in the way that the *binding nodes* for variables is identified.

2 What we propose to do

[GJ93] also discusses the suitability of the various or-parallel execution strategies for different architectures. For example, for shared-memory multiprocessors, methods that sacrifice constant time task-switching tend to be superior to those that sacrifice constant time task-creation and variable access, but on the other hand for distributed memory machines, methods with constant time task creation are likely to perform better. To our knowledge, there have been no experimental studies to confirm these theoretical results. We hope to use the DIMACS challenge as a forum to conduct experimental studies to confirm the theory. General-purpose parallel machines are invariably SIMD or MIMD machines. In the literature we have surveyed, there seems to be no mention of which methods are better

suited for implementation on a SIMD or MIMD machine. This is the other aspect of our experimental study. We want to understand the relation between the architecture of parallel machines and the execution method which is most suitable for that architecture. We want to compare the performances of the methods on shared memory and distributed memory machines. We also want to compare the performance of these methods on a distributed memory SIMD machine versus that on a distributed memory MIMD machine.

2.1 Input Specifications, Benchmark Instances and Input Generators

Our implementation will take a query (goal) to a prolog program as input. The implementation will generate the or-parallel tree and search it. We will take up a few of the methods (say 2 or 3) and generate the or-parallel tree the way the methods dictate, and obtain performance measurements. We will generate the or-parallel tree for a few prolog programs. We want to study the relationship between the critetion that the methods satisfy and the architecture of the machine. A typical program that will be used as input is the quicksort program:

```
qsort([],[]).
qsort([H|T],L) :- split(H,T,A,B), qsort(A,A1), qsort(B,B1),append(A1,[H|B1],L).

split(H,[A|X],[A|Y],Z) :- A =< H, split(H,X,Y,Z).
split(H,[A|X],Y,[A|Z]) :- A > H, split(H,X,Y,Z).
split(H,[],[],[]).

append([],X,X).
append([H|T],U,[H|V]) :- append(T,U,V).
```

Given a query

```
| ?  - qsort([2,5,2,54,678,23,234,8,908,21,6,987,432,56],C).
```

to this program, the implementation would generate the or-parallel tree and search the tree for possible solutions. Notice that the size of the tree generated can be controlled by varying the size of the arguments in the query.

2.2 Computing Platform

Since we want to write programs on distributed-memory SIMD and MIMD machines, the natural choice is the CM-5. Using CM-Fortran, we can write programs in the SIMD style and using f77 with the cmmd library, we can write programs in the MIMD style. We also want to be able to write programs on a shared memory machine. We have access to an *ancient* Encore Multimax at our institute. We would welcome any other suggestions.

3 Conclusion

We have briefly outlined what we propose to do in the challenge. We expect researchers implementing parallel logic programming systems to be interested in the results of this work.

References

[B84] P. Borgwardt, "Parallel Prolog using stack segments on shared memory multiprocessors", in *Proc. Int. Symp. on Logic Prog.*, pp. 2-11, Atlantic City NJ, 1984.

[GJ90] Gopal Gupta and Bharat Jayaraman, "Criteria for Or-Parallel Execution Models for Logic Programs", in *Proc. North American Conf. on Logic Programming* Austin TX, November 1990.

[GJ93] Gopal Gupta and Bharat Jayaraman, "Analysis of Or-Parallel Execution Models", in *ACM Trans. on Prog. Lang. and Systems*, Vol. 15, No. 4, pp. 659-680, Sept. 1993.

[HC83] B. Hausman and A. Ciepielewski, "A Formal Model for Or-parallel Execution of Logic Programs", in *IFIP 83*, P. C. Mason (ed.), North Holland.

[HC86] B. Hausman and A. Ciepielewski, "Performance Evaluation of an Or-Parallel Execution Model for Logic Programs", in *Symposium on Logic Programming*, pp. 246-255, 1986.

[JAM88] V. Janakiram, D. Agarwal and R. Malhotra, et. al., "A Randomised Parallel Backtracking Algorithm", in *IEEE Trans. on Computers*, Vol. 37, No. 12, Dec. 1988.

[T89] P. Tinker, "Performance of an Or-Parallel Logic Programming System", in *Int. Journal of Parallel Programming*, Vol. 17, No. 1, pp. 59-92, 1989.

[W84] D. S. Warren, "Efficient Prolog Memory Management for Flexible Control Strategies", in *Int. Symp. on Logic Prog.*, Atlantic City, pp. 198-202, 1984.

[W87a] D. H. D. Warren, "The SRI-model for Or-Parallel execution of Prolog - Abstract Design and Implementation Issues", in *IEEE Int. Symp. on Logic Prog.*, San Francisco, 1987.

[W87b] D. H. D. Warren, "Or-Parallel Execution Models of Prolog", in TAPSOFT'87, Springer Verlag, LNCS 250.