An OR Parallel Prolog Model for Distributed Memory Systems [§]

(Extended Abstract)

V. Benjumea, J.M. Troya

Dpto. de Lenguajes y Ciencias de la Computación
Universidad de Málaga
Spain
{benjumea,troya}@ctima.uma.es

Introduction and Brief Description

We present a multisequential model to exploit OR parallelism on distributed memory systems. It presents an implementation of the incremental copy mechanism oriented to distributed systems and a novel distributed scheduler is also proposed. WAM modifications to support the proposed model are very simple.

Multisequential models seem to be the best approach to exploit OR parallelism since they can take advantage of the very efficient sequential Prolog compilers based on WAM [3]. These models are based on having multiple Prolog engines that work like sequential ones combined with a method to keep the multiple variable bindings that different Prolog engines could achieve, and with a method to share the work between the system workers.

Several systems based on multisequential models have been proposed. The most known are the Aurora Prolog system [2] on shared memory multiprocessors, and the Muse Prolog system [1] on a multiprocessor machine with distributed and shared memory. Both of them have very good performance results. But in the scope of *pure* distributed memory multiprocessor machines there are very few multisequential systems. However multisequential models seem very suitable for this kind of architecture since they can exploit coarse grain parallelism, which implies large parallel tasks with very low interaction between them.

We have implemented a multisequential Prolog system running on distributed memory multiprocessor machines (a 16 Transputer network) and obtained very good results. The extra overhead introduced by the multisequential implementation with respect to the sequential one is very low, around 6%, and there is an almost linear speed-up for problems with coarse grain parallelism. It is based on a *distributed scheduler* that allows the work to be shared between the workers in the network, and on an *incremental copy mechanism* oriented to distributed systems. The distributed scheduler is particularly important in a distributed implementation, since the communication cost is very dependent on the strategy of distributing the work between the workers. The distributed scheduler proposed defines levels of neighbourhood. It favours the cooperation between the nearest processors, establishing different task granularity depending on the distance between processors. The incremental copy idea was introduced by the Muse model in order to reduce the overhead generated by copying the worker's state to an idle one in a non-shared stack scheme during the sharing of the work. The idea is based on the fact that the idle worker could have already traversed a part of the path to the work, thus it does not need to copy this part of the stacks. This idea has been oriented, in our model, to pure distributed memory machines, differing in some aspects from that proposed by Muse. The current implementation is based on a simple

[§]This work was supported by the project CICYT-TIC 340/90.

WAM as it was defined by D.H.D. Warren [3] without optimizations and without side-effect predicates being supported.

Performance Results and Discussion

This section shows some results after running our system with a set of benchmarks on a distributed memory multiprocessor machine (Parsys Supernode) with 16 Transputers connected to one another following a mesh topology. Each processor has a Prolog machine with its own local memory, and a small router.

The set of benchmarks consists in the following programs divided in three groups: the first group is made up of coarse grain parallelism programs. *8-queens-1* and *8-queens-2* are two well known programs from ECRC. *9-queens-b* is a naive generate&test n-queens program attributed to M. Bruynooghe. *9-queens-p* is a fused generate&test n-queens program attributed to L. Pereira. The second group is made up of medium grain parallelism programs. *house* is the "who owns the zebra" puzzle from ECRC. The last group is made up of programs that exploit small grain parallelism. *farmer* is the "farmer, wolf, goat, cabbage" puzzle from ECRC. Table 1 shows the run times (in milliseconds) for these benchmarks. All of them look for all solutions of the problem. The speed-up relative to the 1 worker implementation is given in parentheses. The last column shows the run times for the sequential WAM emulator on which our implementation is based, and its speed-up relative to the 1 worker one.

	1 Worker	4 Workers	8 Workers	16 Workers	Seq. WAM
9-queens-b	2252240.0	569987.0 (3.95)	287953.0 (7.82)	146255.0 (15.39)	2113510.0 (1.06)
9-queens-p	66886.3	16830.6 (3.97)	8643.2 (7.73)	4673.3 (14.31)	63920.6 (1.04)
8-queens-1	17378.5	4424.1 (3.92)	2404.6 (7.22)	1489.3 (11.66)	16686.2 (1.04)
8-queens-2	42651.6	10753.3 (3.96)	5793.5 (7.36)	3400.7 (12.54)	41419.3 (1.02)
house*20	13391.9	4155.5 (3.22)	2679.3 (4.99)	2026.1 (6.60)	11094.5 (1.20)
farmer*100	7187.7	2822.7 (2.54)	2675.7 (2.68)	2886.9 (2.48)	6437.3 (1.11)

Table 1:

These results are very encouraging. Our system presents almost linear speed-up for programs that exploit coarse grain parallelism, not very bad speed-up for programs with medium grain parallelism, and very little speed-up for programs with small grain parallelism. The overhead introduced to support the multisequential model is very low, around 6%. These results are comparable to the most known multisequential models for shared memory systems such as Muse [1] and Aurora [2].

Due to the early stage of development of our system, these results must be considered only as a point of reference about the proposed model.

References

[1] Ali, K., Karlsson, R. The Muse Or-Parallel Prolog Model and its Performance. *North American Conference on Logic Programming*, pages 757–776. The MIT Press, 1990.

[2] Lusk, E., Warren, D., Haridi, S. The Aurora Or-Parallel Prolog System. Technical Report, TR-90-07. University of Bristol, 1990.

[3] Warren, D.H.D. An Abstract Prolog Instruction Set. Technical Note 309. SRI International, 1983.