AM2: An AND/OR Parallel Prolog Simulator

Luiz R. R. Monnerat* Inês Dutra

{*monnerat,ines*}*@cos.ufrj.br*
COPPE/Sistemas
Federal University of Rio de Janeiro

Abstract. Prolog programs have two main sources of parallelism: OR-parallelism and AND-parallelism. The combined exploitation of both sources can lead to high degrees of parallelism, but it is hard to implement. The severe costs of the implementations make it necessary to investigate the potential gains of a combined strategy and to carefully evaluate the overheads of the model devised. The availability of a general but flexible simulator would be useful, not only for the first studies of a new model, but also for the evaluation of the potential sources of parallelism in real applications. In this paper we present a system that addresses this need by performing simulations of the combination of OR-parallelism and determinate AND-parallelism. This system, called AM2, allows the user to change the cost of several tasks and overheads involved in a parallel Prolog execution. By using different parameters values, it is possible to estimate the behavior of a particular parallel model. Different selections criteria and models of execution are also available for simulation. Detailed timing information is shown for each execution, allowing the identification of the performance problems of the model under study. In its current version, which does not implement cut, the system is an interesting tool for research and teaching purposes, being available by e-mail. The implementations of cut and a new AND-scheduler are the next steps towards the development of more complete simulator.

Keywords: and-or parallelism, simulation, Andorra-I, performance evaluation, logic programming.

1 Introduction

Being different from the conventional Von Neumann languages, Prolog allows the implicit exploitation of parallelism. Two main sources of parallelism can be exploited in Prolog: OR-parallelism and AND-parallelism. Various systems that exploit one of these sources have already been implemented [10, 1, 9]. The combined exploitation of both sources can lead to higher degrees of parallelism but it is hard to implement [13, 5]. The severe costs of the implementation make it necessary to investigate the potential gains of the combined strategy and to carefully evaluate the overheads of the model devised.

These evaluations are usually made with a simulator of the model in study. The availability of a general but flexible simulator would be useful, not only for the first studies of a new model, but also for the evaluation of the potential sources of parallelism in real applications. In this paper a general AND-OR Parallel simulator, the AM2 system, is presented. This system, which was firstly implemented by a wider team [11], allows the simulation of systems that exploit both determinate AND-parallelism and OR-parallelism.

AM2 allows the user to play with a number of parameters. By using different parameters values, it is possible to estimate the behavior of a particular parallel model. AM2 is flexible enough to allow the simulator to evaluate models that would run in different hardware architectures (e.g. shared vs. distributed memory). Different goal selection criteria (e.g. rightmost and leftmost) and execution models (e.g. Prolog and Andorra) are supported.

The remainder of this paper is organized as follows. Section 2 presents the basic AM2 model. Section 3 describes the strategies used. In section 4 we discuss the completeness of the simulator. Section 5 presents and analyzes the results of the evaluations of three small programs. Section 6 summarizes the future work on the AM2 system. In the last section we draw our conclusions.

* Supported by PETROBRAS

2 The AM2 Model

The model is based on an algorithm similar to Andorra [16], that exploits determinate AND-parallelism and OR-parallelism. In Andorra, determinate goals, i.e., goals which match at most one clause in the program, are executed eagerly and in parallel. The basic algorithm of the simulator is described bellow:

1- Read initial query;
2- Adds the query to a public pool of goal lists
3- For all goal lists in the public pool do:
3.1- For all determinate goals in the list do:
  3.1.1- Substitute the determinate goal by the corresponding clause body.
3.2- When there is no determinate goal left in the list, select one of the non-determinate goals and make a choicepoint. Make the choicepoint public by putting its alternatives in the public pool of goal lists.

This algorithm is valid for the Andorra model of execution. The Prolog execution model is also available, for which the step 3.1 is not performed and no AND-parallelism is exploited. The selection of goals can be made in leftmost or rightmost order.

3 Strategies

To support both AND-parallelism and OR-parallelism the workers[2]are divided into teams. Each worker has its own Prolog sequential engine. The teams exploit the different OR-branches in parallel. Workers in a team evaluate determinate goals in AND-parallel. In the current version all teams have the same number of workers. The numbers of teams and workers are defined by the user.

3.1 AND-Parallelism

The exploitation of AND-parallelism is made by performing the reductions of the determinate goals in parallel by different workers, as in Andorra-I [18]. This strategy is called determinate AND-parallelism and usually leads to fine grained parallelism, but it is easier to implement than dependent or independent AND-parallelism, making it suitable for combining with OR-parallelism. The AND-scheduler implemented is quite simple. There is a master worker responsible for distributing the goals in the active list among all the workers, including itself. The master tries to balance the work by giving the same number of goals to each worker. Upon receiving the list of goals a worker performs the reduction of the determinate goals or fails if there is a goal with no matching clause. When there is no determinate goal left (or after failing) the worker returns the new list to the master.

The master builds a new list of goals from the sublists returned by the workers and checks if there was any reduction. If a reduction took place it distributes the goals to the workers again, trying to find a new determinate goal[3]. If no reduction is performed the master selects a goal and makes a choicepoint.

This scheduling strategy is not efficient for load balancing. An idle worker cannot select available work from the list of goals of a busy worker of the same team. A new AND-scheduler that addresses this problem is being designed.

[2] In an actual system each worker is usually associated to an operating system process or to a hardware processor. AM2 simulates parallel execution of workers

[3] A reduction made by a worker can turn another goal determinate

3.2 OR-parallelism

The AM2 base model exploits OR-parallelism by using stack-copying with bottom-most dispatching strategy (like MUSE [1]). The system is flexible enough to allow the user to estimate the performance of models other than stack-copying, with the use of the cost functions, which will be presented later. Top-most dispatching simulations would require some changes in the structure of the simulator.

The OR-scheduler implemented tries to keep the idle time to a minimum. During the simulation, the OR-scheduler keeps track of the parallel time of creation of each choicepoint, matching idle teams with available OR-branches as soon as possible. To accomplish this, when a worker becomes idle, the OR-scheduler matches it with the next OR-branch available from the last[4] choicepoint created.

As in Andorra, for each query all the results are shown. The OR-scheduling mechanism can lead to the results being shown out of order. A suspension mechanism to force the Prolog order is being implemented.

3.3 Execution Models and Selection Criteria

AM2 can evaluate the query using different execution models and goal selection criteria. At the moment two selection criteria are available: leftmost and rightmost.

The execution models implemented by the simulator are Prolog (depth first search) and Andorra (determinate goals first). With the Andorra model the determinate goals are executed first and in parallel. No AND-parallelism is exploited with the Prolog model of execution.

The traditional execution models are Prolog and Andorra with leftmost goal selection. With AM2 we can also evaluate these models with rightmost selection of goals.

3.4 Execution Time

The system is instrumented to evaluate the execution time for each query. The time is calculated using cost functions defined by the user. There are several tasks and overheads defined, and for each one there is an associated cost function initialized with a default. The cost function has the form:

cost(s(C-, S-, W-, R-), Task+, Parm+), where

Task	: Task/overhead performed
Parm	: Parameter for cost determination
C	: Parallel execution time
S	: Sequential execution time
W	: Number of workers used
R	: Number of reductions performed

The units of parallel and sequential execution time are user defined but must be the same, e.g. microseconds or processor cycles.

Through the cost functions the user can define the costs of reductions, failures, AND-scheduler overhead, OR-scheduler overhead, choicepoint creation and backtracking. Using these elementary costs as input, AM2 evaluates the execution time of a given query. For any query, AM2 evaluates the execution times for a sequential system, for a parallel system with just one worker, and for a parallel system with the number of workers specified by the user, showing the parallel overhead and speedups obtained.

Default cost functions are provided, but they are not intended to be representative of any particular system. The default costs were chosen from observation of actual parallel systems. For example, the default reduction cost function assumes that the speeds of the parallel and sequential

[4] Last in terms of parallel execution time

engines are the same, like in MUSE [1], which is not valid for systems using Binding Arrays [15]. The user is responsible for the definition of the costs for his/her particular system.

As an example, we show in figure 1 the definition of the costs of reductions for a hypothetical system. The first clause of figure 1 defines the parallel engine of this hypothetical system as being 10% slower than the sequential one. Besides, all reductions in this system take a time that is proportional to the number of non instantiated variables of the goal, with the exception of the reductions performed for the goal *fix/1*, which has a fixed cost and always takes 20 time units (e.g. microseconds).

```
cost(s(C,S,1,1),reduction,P) :-
        cost_r(S,P),
        C is 1.1*S.
cost_r(20,fix(_)).
cost_r(S,Pred) :-
        num_vars(Pred,N),
        S is 5*N.
```

Fig. 1. Definition of the costs of reductions for a hypothetical system.

3.5 Consulting Environment

AM2 provides the standard Prolog prompt with some extra control commands:

{T,W}: Sets the number of teams (T), and the number of workers per team (W). Default: {1,1}.
{left},**{right}**: Sets the selection criteria.
{andorra},**{prolog}**: Sets the model of execution.
loop/noloop: Enters/Exits loop mode. In this mode, identified by an "L" in the prompt, any query is run with all combinations {Teams,Workers} ranging from {1,1} to {T,W}.

3.6 Execution Time Information

The information shown by AM2 is detailed for each team and task/overhead type. For the teams it is shown the parallel and sequential execution times, the number of reductions, and the number of workers available and used. For each task/overhead type it is shown the number of reductions and the sequential and parallel times spent. It is also shown the idle parallel execution time of the workers. The sequential time is valid only for executions with only one worker.

Figure 2 illustrates a sample output with the statistics provided by the simulator. This output is from the execution of the query *won(X)* of a small program representing the possibilities of nought winning the tic-tac-toe game. This computation has very little AND-parallelism and some OR-parallelism. For this simulation we used Andorra execution model, rightmost goal selection and two teams of one worker each.

The first table of the output (figure 2) shows for each team the parallel execution time (PT), the number of reductions (Red), the number of workers actually used during the computation (WU), and the total number of workers available (WT). The last line of this table shows the same statistics for the whole system. The parallel execution time for the whole computation is the maximum value among the execution times of the teams. From this table we can see that for this computation the work was well balanced among the teams.

The second table of figure 2 shows the simulated execution time spent on different activities. The first column of this table shows the type of activity:

and-sched: AND-scheduler overhead;
built-in: Execution built-in predicates;
cp: Choicepoint creation;
or-sched: OR-scheduler overhead;
backtrack: Backtracking;
reduction: Reductions;
failure: Failures;
idle: Total time the workers remained idle.

For each activity, the second table of figure 2 shows the number of reductions and the parallel and sequential execution times. From this table, it can be seen that for this simulation, 7.9% of the total parallel time was due to OR-scheduler overhead, 74.1% was spent performing reductions, and 13.9% was due to failures. The workers remained idle during 2.3% of the total parallel execution time.

The last three lines of the output show the simulated sequential execution time, the parallel overhead and the speedup obtained. The parallel overhead is calculated using the sequential execution time and the parallel time for an execution with one team of one worker. The speedup is calculated in relation to the parallel time for an execution with one team of one worker.

```
query   won(_483)

teams 2; workers 1;
model andorra; order rightmost

Answer: nought

Statistics:

Team    PT   Red   WU   WT
---------------------------
  1    104    11    1    1
  2    108    11    1    1
---------------------------
Total  108    22    2    2

Type      red   red  ST    ST%  PT   PT%
-------------------------------------
and_sched  0    0.0   0    0.0   0   0.0
built_in   0    0.0   0    0.0   0   0.0
cp         0    0.0   0    0.0   4   1.9
or_sched   0    0.0   0    0.0  17   7.9
backtrack  0    0.0   0    0.0   0   0.0
reduction 16   72.7 160   84.2 160  74.1
failure    6   27.3  30   15.8  30  13.9
idle       0    0.0   0    0.0   5   2.3
-------------------------------------
total     22 100.0 190 100.0 216 100.0

sequential time    : 190
parallel overhead  : 11.1 %
speedup            : 1.95
```

Fig. 2. Timing information for the query *won(X)*.

4 Completeness and Robustness

In this first version we did not implement cut and synchronous side-effects. The system may also show the solutions out of the Prolog order. Cut is being implemented now and will probably be available by January/1996. An efficient AND-scheduler is also being designed. The system evaluated a selection of small programs without problems. At this stage AM2 would be useful for students to evaluate issues and potentials of the parallel execution of Prolog programs. With the implementation of cut, the system will be ready for evaluating real applications.

5 Results

As an example of the capabilities of the AM2 system, three small programs were evaluated. The first program is an instance of the tic-tac-toe game (query *won(X)*) with some OR-parallelism and very little AND-parallelism. The output of one execution of this program is presented in figure 2. The second program is a small database adapted from Conery [2]. The query run for this database (*paper(A,1978,I)*) has no AND-parallelism. The third benchmark is a conditional permutation of a list with three elements with a good degree of OR-parallelism and some AND-parallelism (query *cond_perm([3,5,2],X)*). This program could not be run with rightmost selection.

All the simulations were run with the default costs. These simulations are not representative of any particular system, we just wanted to show which kind of information AM2 provided.

Query	Order	Prolog	Andorra		
		1	1	2	3
won	left	24	24	28	28
won	right	22	22	27	28
paper	left	19	19	19	19
paper	right	17	17	17	17
cond_perm	left	138	137	146	170

Table 1. Size of the computation (number of reductions and failures). The second line of the table indicates the number of workers per team

Table 1 shows the size of the computation (number of reductions and failures) for each query. The first column of the table shows the query chosen to run the program, the second column (order) shows the goal selection criteria, the third column shows the size of the computation for the Prolog model with one worker. The last three columns show the size of the computation for the Andorra model with one, two and three workers. For the queries with AND-parallelism, *won* and *cond_perm*, the size of the computation increased with the number of workers per team, because of the speculative AND-parallel work. As cut is not implemented, there was no speculative OR-parallel work. With respect to the sequential executions, we can observe that the number of reductions varied according to the goal selection criteria.

The second table shows the simulated execution times with sequential Andorra and parallel Andorra with just one worker. We assumed the same speed for the parallel and sequential engines (default costs). The simulated parallel overhead for the three simulations was around 10%. The output of the simulation gives enough information for identifying the overhead sources, as can be seen from the example shown in figure 2.

Tables 3, 4 and 5 show the speedups for the queries *won*, *paper* and *cond_perm*. All speedups were calculated in relation to the simulated parallel Andorra with one worker and the standard leftmost goal selection criteria.

Query	Order	Andorra		Parallel
		seq.	par.	Overhead
won	left	210	231	10.0
paper	left	175	193	10.3
cond_perm	left	1245	1385	11.2

Table 2. Execution times in sequential Andorra and parallel Andorra with just one worker.

No. of	workers per team		
Teams	1	2	3
1	1.00	1.00	1.03
2	1.86	1.82	1.94
3	2.57	2.75	2.72
4	3.21	3.40	3.40
5	3.21	3.55	3.50
6	3.30	3.61	3.55

Table 3. Speedups for the query *won(X)* for the Andorra model with leftmost selection.

Figure 3 plots the information from table 3 and shows that the exploitation of OR-parallelism for the query *won* led to good speedups up to four teams. The gains from the exploitation of the few sources of AND-parallelism in this query were very poor.

For the query *paper* the results were a little worse. As this query has no AND-parallelism, the use of more than one worker per team implied in loss of performance due to overheads of the AND-scheduler. These overheads could be avoided or minimized with a new AND-scheduler design, or with different cost functions. Table 4 shows also that rightmost selection performed better for this query. From figure 4, which plots the information from table 4, it is easy to see that six teams are enough to exploit all the available OR-parallelism from this query.

Table 5 and figure 5 show that for the query *cond_perm* the speedup steadily improves with the number of teams. This shows that this program has a good degree of parallelism. As we increase the size of the teams, the speedups did not improve significantly. The AND-scheduler was not capable of exploiting the AND-parallelism sources efficiently.

No. of	Left	Left	Right
Teams	1	2	1
1	1.00	0.97	1.13
2	1.97	1.89	2.12
3	2.76	2.64	3.11
4	3.06	2.97	3.57
5	3.71	3.64	4.11
6	4.29	4.11	4.29
7	4.29	4.11	4.29
8	4.29	4.11	4.29

Table 4. Speedups for the query *paper(A,1978,L)*, with the Andorra model. The first line of the table indicates the goal selection criteria. The second line of the table indicates the number of workers per team.

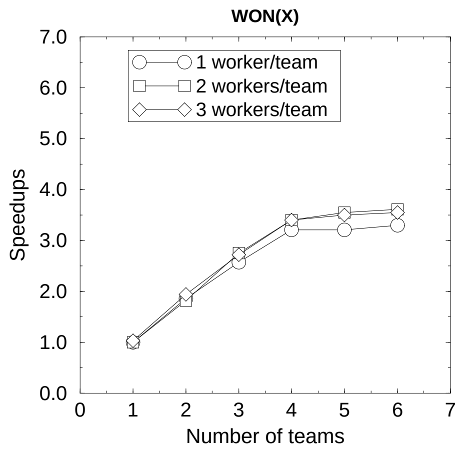

Fig. 3. Speedups for the query *won(X)*, from table 3.

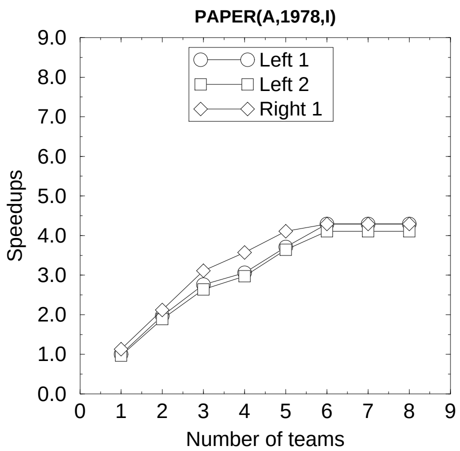

Fig. 4. Speedups for the query *paper(A,1978,I)*, from table 4.

6 Related and Future Work

AM2 is an experimental simulator that was developed to study the nature and characteristics of parallelism of classes of applications, and to allow evaluations of parallel execution models. Other simulators [14, 8, 3] have the purpose of simulating a particular model of parallelism. AM2 purpose is more general because it is capable of simulating various parallel models, besides being able to evaluate different goal selection criteria.

Currently, AM2 does not support the executions of cuts and synchronous side-effects. The next steps towards the developing of a more realistic simulator will be:

1. To support cut and commits.
2. To improve the AND-scheduler.
3. To implement a more flexible scheduler to allow workers to migrate freely between teams [4].
4. To implement other execution models and goal rule selection, in order to simulate new models such as PBA [7], ParAKL [12], IDIOM [6] and EAM [17].
5. To exploit independent AND-parallelism.

No. of Teams	Prolog 1	Andorra 1	Andorra 2	Andorra 3
1	1.06	1.00	1.08	1.11
2	1.97	1.88	2.06	2.12
3	2.79	2.64	2.88	3.00
4	3.49	3.31	3.64	3.75
5	4.06	3.89	4.31	4.46
6	4.60	4.44	4.90	5.07
7	5.14	4.73	5.52	5.57
8	5.50	5.19	5.90	6.12
9	5.63	5.59	6.33	6.64
10	6.07	5.69	6.58	6.83
11	6.30	6.02	6.92	7.06
12	6.74	6.56	7.34	7.38

Table 5. Speedups for the query *cond_perm([3,5,2],X)*, with leftmost goal selection. The second line of the table indicates the number of workers per team.

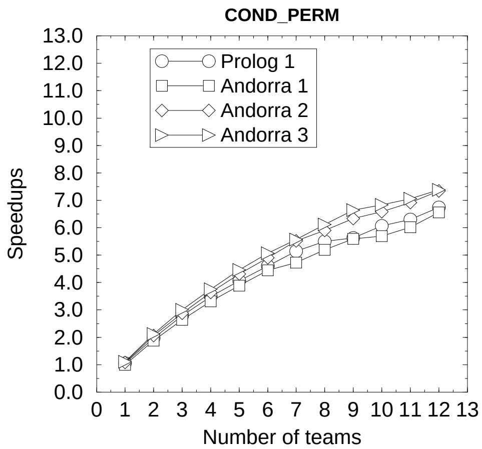

Fig. 5. Speedups for the query *cond_perm*, from table 5.

A further step on improving AM2 will be to inform the user what kind of model is more suitable to a given class of applications, with the ambitious goal of rewriting a given application to adapt to a particular model.

Another step forward would be to develop compilation techniques, that guided by the information provided by the simulator, could be used by the parallel runtime system to exploit more parallelism from the applications.

7 Conclusions

The system implemented provides a tool for studying parallelism issues. It is flexible enough to be tuned to become similar to a specific model, allowing first level overhead evaluations. The detailed timing information provides an inside view of the potential sources of parallelism and overhead of an application.

Results of the simulation of three small programs were shown, illustrating the level of information the AM2 system provides. At the moment the simulator is not robust enough to run real applications. By the beginning of the next year, when cut and a new AND-scheduler will have been implemented, evaluations of bigger programs will be made. In the meantime AM2 will basically serve as a didactic tool. It is available by e-mailing monnerat@cos.ufrj.br with AM2 as subject. Requesters are kindly asked to inform in the body of the message the proposed use of the system.

References

1. Khayri A. M. Ali and Roland Karlsson. The Muse or-parallel Prolog Model and its Performance. In *Proceedings of the 1990 North American Conference on Logic Programming*, pages 757–776. MIT Press, October 1990.
2. J. S. Conery. *Parallel Execution of Logic Programs*. Kluwer Academic Publishers, 1987.
3. Ruy Marinho da Costa. Simulacao de Modelos Paralelos de Programacao em Logica. Master's thesis, COPPE/Sistemas, UFRJ, 1991.
4. Inês Dutra. Strategies for Scheduling And- and Or-Work in Parallel Logic Programming Systems. In *Proceedings of the 1994 International Logic Programming Symposium*, pages 289–304. MIT Press, 1994. Also available as technical report CSTR-94-09, from the Department of Computer Science, University of Bristol, England.
5. Gopal Gupta, Enrico Pontelli, and Manuel Hermenegildo. &ACE: A High Performance Parallel Prolog System. In *Proceedings of the First International Symposium on Parallel Symbolic Computation, PASCO'94*, 1994.
6. Gopal Gupta, V. Santos Costa, R. Yang, and M. V. Hermenegildo. IDIOM: Integrating Dependent and-, Independent and-, Or-parallelism. In *Proceedings of the 1991 International Logic Programming Symposium*, pages 152–166. MIT Press, October 1991.
7. Gopal Gupta and Vítor Santos Costa. And-Or Parallelism in Full Prolog with Paged Binding Arrays. In *LNCS 605, PARLE'92 Parallel Architectures and Languages Europe*, pages 617–632. Springer-Verlag, June 1992.
8. Gopal Gupta and David H. D. Warren. An Interpreter for the Extended Andorra Model. Technical Report, University of Bristol, 1991.
9. Manuel Hermenegildo. An Abstract Machine for Restricted And-Parallel Execution of Logic Programs. In Ehud Shapiro, editor, *Proceedings of the Third International Conference on Logic Programming*, pages 25–39. Springer-Verlag, 1986.
10. Ewing Lusk, David H. D. Warren, Seif Haridi, et al. The Aurora Or-parallel Prolog System. *New Generation Computing*, 7(2,3):243–271, 1990.
11. Luiz R. R. Monnerat, M. R. Dia, and A. F. Tavares. AM2: Um simulador Prolog Paralelo. September 1995.
12. Johan Montelius. Penny, A Parallel Implementation of AKL. In *ILPS'94 Post-Conference Workshop in Design and Implementation of Parallel Logic Programming Systems, Ithaca, NY, USA*, November 1994.
13. Vítor Santos Costa, David H. D. Warren, and Rong Yang. The Andorra-I Preprocessor: Supporting full Prolog on the Basic Andorra model. In *Proceedings of the Eighth International Conference on Logic Programming*, pages 443–456. MIT Press, 1991.
14. Kish Shen. *Studies of And/Or Parallelism in Prolog*. PhD thesis, Computer Laboratory, University of Cambridge, 1992.
15. David H. D. Warren. The SRI Model for Or-Parallel Execution of Prolog—Abstract Design and Implementation Issues. In *Proceedings of the 1987 International Logic Programming Symposium*, pages 92–102, 1987.
16. David H. D. Warren. The Andorra model. Presented at Gigalips Project workshop, University of Manchester, March 1988.
17. David H. D. Warren. The Extended Andorra Model with Implicit Control. Presented at ICLP'90 Workshop on Parallel Logic Programming, Eilat, Israel, June 1990.
18. Rong Yang, Vítor Santos Costa, and David H. D. Warren. The Andorra-I Engine: A parallel implementation of the Basic Andorra model. In *Proceedings of the Eighth International Conference on Logic Programming*, pages 825–839. MIT Press, 1991.

This article was processed using the LaTeX macro package with LLNCS style