Performance Analysis of a Strategy to Distribute And-work and Or-work in Parallel Logic Programming Systems

Inês de Castro Dutra*

COPPE/Sistemas
Universidade Federal do Rio de Janeiro
Rio de Janeiro, Brasil
e-mail: ines@cos.ufrj.br

Abstract

This paper studies the performance of Andorra-I, a parallel logic programming system that exploits and-parallelism and or-parallelism with a novel strategy to distribute and-work and or-work among processors. The strategy, *work-guided* guides its decisions by looking at the amount of current and-work and or-work available in an application during execution. The scheduler decision strategy moves workers from one parallel task to another according to the tasks sizes. Results show that the work-guided strategy works quite well and produces better results than the ones produced with a version of Andorra-I that does not allow dynamic migration of workers during execution. We believe that this strategy can be applied to other parallel logic programming systems that aim to exploit both and- and or-parallelism in a single framework.
Keywords: and-or scheduling, and-or parallelism, logic programming, parallelism, Andorra-I, performance evaluation

1 Introduction

There are two main sources of parallelism in logic programming, namely or-parallelism and and-parallelism. Or-parallelism is exploited when several alternative clauses to a goal are executed in parallel. And-parallelism is exploited when two or more goals in the body of a clause are executed simultaneously. Exploitation of full or- and and-parallelism is limited by the number of physical processors available in a system. And-parallelism is also limited by the interdependence among goals.

When allowing and-goals to proceed in parallel, the logic programming system needs to choose what goal to select next. This job is usually done by an *and-scheduler*. When allowing several clauses to proceed in parallel, the system needs to choose what alternative from which branch to select next. This task is usually performed by an *or-scheduler*. There are several and-scheduling techniques [14, 6, 20] implemented for scheduling and-parallel work in systems that exploit and-parallelism only, such as &-Prolog [13] and commited-choice languages systems [7, 4]. There are also several or-scheduling techniques [12, 2, 19] implemented in systems that exploit only or-parallelism such as the Aurora [15], Muse [1], Delphi [5], and Opera [3] systems.

So far, only the problem of and-scheduling alone and or-scheduling alone have been tackled by the parallel logic programming researchers. Scheduling both and-work and or-work is a new and hard problem to be solved in parallel logic programming systems. When allowing both kinds of parallelism to be exploited in a single framework, the system needs to deal with an extra problem that is what kind of work to choose from: and- or or-.

*Research supported by CNPq, Brasilian Research Foundation under grant 202270/89.0. This work was developed in the Department of Computer Science, University of Bristol, England

In figure 1 we show a tree that contains different kinds of parallel work. In this tree we have several branches with choicepoints that contain alternatives left open, and each branch can produce some amount of and-work through the execution of parallel and-goals. The problem that arises is how to distribute the two different kinds of work among the processors. For example, if the and-work in the left arc of the tree is fine grained, it can be a good strategy to allocate only one processor to this arc and to allocate more processors to expand the open alternatives located on the third arc of the tree. Moreover, only one processor should be allocated to the task in the second arc. As all solutions are to be found, all arcs need to be expanded anyway, but the way and the order they are expanded will lead to an earlier or longer time to produce a solution.

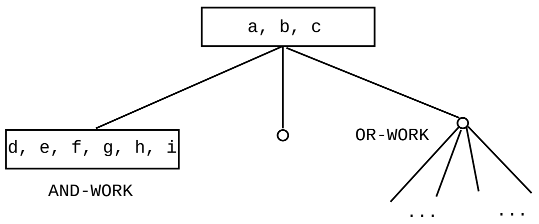


Figure 1: Different kinds of work in an and-or tree

This paper studies a dynamic strategy that solves this problem in the context of the Andorra-I parallel logic programming system [23, 16, 22].

Most scheduling strategies reported in the literature are applied to scientific applications that have a very regular task structure. Therefore most of the solutions are static scheduler strategies. Our strategy is dynamic and applied to irregular computations. We implemented the strategy in the Andorra-I parallel logic programming system and evaluated it by showing results for a wide range of applications.

In an earlier work [8] we described briefly the first work-based strategy and showed some results for a very small number of processors and different applications. In another work [9], the work-guided strategy is compared with another scheduling strategy. In this work we discuss the benefit of adding dynamic distribution of workers in Andorra-I and compare the performance of the old version of Andorra-I that does not implement dynamic distribution of workers with the performance achieved with the work-guided strategy.

The paper is organised as follows. Section 2 describes briefly the first version of Andorra-I that did not allow dynamic migration of workers between and-work and or-work. Section 4 describes the work-guided strategy heuristics used to guide the scheduler in the new version of Andorra-I. Section 3 describes the applications used to test the strategy. Section 4.2 evaluates the work-guided strategy and shows results produced for a wide range of applications. Finally, section 5 draws some important conclusions of this work and presents alternative solutions and future work.

2 Andorra-I in the Past

Andorra-I is a system that exploits multiple forms of parallelism. Systems that exploit several forms of parallelism, such as Andorra-I, face a new problem: *how to transparently and efficiently distribute and-work and or-work that arise dynamically and irregularly during the computation of a program*? By *transparent* we mean that the system should exploit parallelism without any user's intervention. By *efficient* we mean to distribute both kinds of work and achieve speedups close to a possible optimal. By *dynamic* and *irregular* we mean that the sizes and forms of parallelism vary with time.

Andorra-I is based on the Basic Andorra Model [21]. The Basic Andorra Model assumes that goals that match at most one solution are to be executed in parallel, and that alternatives left open in the execution tree can be executed in parallel. Goals that match at most one solution are called *determinate* goals. In the Basic Andorra Model, these goals are executed eagerly and in parallel. Andorra-I implements and-parallelism in the same way as the JAM abstract machine [7], and it expoits or-parallelism in much the same way as the Aurora or-parallel system [15].

The system is designed in a way that workers are grouped into teams. Each team is composed of a master and some slaves. Workers in a team cooperate to share available and-parallel work, while teams cooperate with each other to exploit or-parallelism. Before our work, the user had to choose a fixed configuration of masters and slaves to exploit the parallelism available. This brought three main drawbacks to the system:

- It was very inconvenient and difficult to use, and it was inconsistent with the aim of exploiting parallelism implicitly.
- For most cases, the user would choose a configuration of workers that would produce results far from being optimal.
- Because of the varying nature of parallelism in some programs, the system would still produce performance below the best achievable, even if the user could (somehow) choose the best fixed configuration.

The main objective of dynamic dsitribution of workers is to distribute workers automatically between the two kinds of work available, and thus release the user from the burden of having to adjust manually fixed configurations of masters and slaves into teams.

In order to achieve this goal, and hence make Andorra-I a practical system for the user, there are at least three possible approaches. In the first approach we can use compile-time analysis of the programs for a certain set of queries (probably the most common) to generate information about the best fixed configuration of masters and slaves that should be used for that particular computation. In the second approach the work distribution is decided at runtime. In the third approach the work distribution is decided at runtime, but guided by information generated through compile-time analysis.

The first approach has three main disadvantages:

- The fixed configuration is set forever in the beginning of the computation and *never* changes. This may lead to loss of parallelism, since the parallelism varies along the execution time.
- The process of collecting information about the amount of parallelism in a program in order to find a suitable fixed configuration of workers into teams is very complicated and sometimes does not produce the precise and expected results. First because the computation of an application varies greatly with different queries and different sizes of data. Second because some variable dependencies in the program are only solved at runtime, which makes the task of generating *precise* or even *useful* information more difficult.
- Usually, the process of obtaining useful information through compile-time analysis is very slow, therefore the overall gain in running the application in a parallel system may not be justified.

The second approach seems to be better than the first one. Dynamic reconfiguration allows the workers to react to the runtime system, and therefore gives a chance to the workers to change its position according to events that happen at runtime. It has also some disadvantages. The first one is that dynamic strategies, in general, do not find global optimums. However a fixed configuration would not achieve a global optimum either. The second disadvantage is that we have the overhead of re-scheduling at several stages of the computation. But as long as we keep scheduling overhead costs and frequency of task switching low, this disadvantage is manageable.

The third approach, which is to combine a static schedule with dynamic scheduling seems to be the best choice, as long as the static information can guide the dynamic scheduling decisions whenever possible, and overheads are kept low.

Although we think that the combination of dynamic plus static information is a better choice to guide scheduling decisions, we concentrated only on the dynamic aspect of the problem (leaving scope to static information to be used), as the issue of generating static information for the programs is a whole subject by itself. In any case, as we believe that static information alone is not sufficient to do a good schedule, but totally dynamic scheduling will suffice, we concentrated on the main problem, that is to find a *dynamic* solution for distributing work that varies *dynamically*.

A dynamic strategy not only solves the problem of making Andorra-I a practical system, but can also allow the system to exploit more parallelism from programs where and- and or-parallelism vary with time. A very simple example is a program whose search tree produces and-parallelism in the beginning of the execution, say to set up a set of constraints, and later produces or-parallelism to search for a solution. No single fixed configuration of reasonable number of workers (25 workers arranged in 5 teams would do the job!) into teams solves this problem optimally, because the system should be able to configure workers in order that they exploit and-parallelism in the beginning, and later, work independently in each or-parallel branch.

3 About the Benchmarks

All programs used as the benchmark set were selected according to their degree of parallelism. One group of programs has predominantly and-parallelism, another has predominantly or-parallelism, another has both kinds of parallelism in different phases of the computation, and another has both kinds of parallelism appearing at the same computational phase.

Some of the benchmarks are specially written to test the reconfigurer. Others are real applications used by companies or by academic people. The idea behind writing special programs is to predict the behaviour of the reconfigurer and evaluate closely the scheduling strategies.

A detailed decription of the benchmarks can be found in [10].

4 Andorra-I with the Reconfigurer

4.1 Showing the Benefits of Dynamic Reconfiguring

The work-guided strategy bases its decisions on a set of parameters: cost to redeploy a worker to and-work, cost to redeploy a worker to or-work, different priorities to give work, correction factor between and-work and or-work, size of goals and size of alternatives. We made experiments for a wide range of combined parameter values. In this chapter we present results obtained with a single parameter setting that produced the best results over all benchmark set. Initial values for some of the parameters were taken from a performance analysis of the Andorra-I system. A range of integer values greater than and lesser than those values were tried. Not all combination of parameter values were tried as this would lead to an infinite set of experiments. Our main conclusion for the parameter settings tried is that although, for some benchmarks, some variation of parameter values can affect performance, we could choose overall one single set of parameters that produced good performance on all benchmarks. Some experiments with different parameter settings can be found in [10]. More detailed study of variation of parameters for ten processors can also be found in [10].

Another important parameter is the starting configuration to run Andorra-I with the reconfigurer. The starting configuration could have been obtained through compile-time analysis of the programs. But sometimes compile-time analysis would not be able to infer any starting configuration due to the complexity of the problem. In that case we would be back to the same problem of having to choose different configurations for each computation. The reconfigurer, therefore, uses a standard initial configuration, which is chosen to be a single team (i.e., a configuration that supports and-parallelism, but not or-parallelism). This was an arbitrary choice, but as we will see later, this choice for the starting configuration does not cause any impact on the performance, because the reconfiguring overheads are very low.

In this section we intend to show the benefits of adding a reconfigurer to the Andorra-I system. Our objective is to show that Andorra-I with a reconfigurer is much more user friendly, while achieving very good performance. In order to do that we compare the performance of our reconfigurer with the performance achieved by different versions of the old Andorra-I corresponding to different fixed configurations chosen. This method of comparing Andorra-I with the reconfigurer with the old version of Andorra-I seems reasonable, since as we take real results produced in practice with plausible fixed configurations.

There is an infinite number of fixed configurations possible. We will limit our comparison to five plausible fixed configurations. As explained before, in Andorra-I, users can specify the number of teams and number of slaves to run a computation, where the slaves are evenly allocated to each team. In order to allow the user to enter a more convenient fixed configuration for all numbers of processors we assume that the system provides a formal way of choosing the configurations. In that case for a number of processors n we will use the following formula, where we only have plausible user choices, assuming a particular weight between and-parallelism and or-parallelism, and with teams having approximately the same size.

$$n^p \textit{ teams with } n^{1-p} \textit{ workers each}$$

The number p corresponds to the proportion of or-parallelism, i.e. the relative weight we want to give to or-parallelism, and it is in the range 0 to 1. We take the least integer approximation for n^p and n^{1-p}, with n being the number of processors. If there are any remaining workers from the approximation to n^{1-p}, they are allocated evenly as slaves to the existing teams, until there are no more remaining workers. This formula, allows the user to specify weights for and-parallelism and or-parallelism, and to use a single formula for any numbers of processors.

We consider the following five plausible configurations:

1) give weight 0 to or-parallelism. In this case $p = 0$. We have 1 team of n workers.

2) give weight 0 to and-parallelism. In this case $1 - p = 0$, and $p = 1$. We have n teams of 1 worker each.

3) give equal weights to both and- and or-parallelism. In this case $p = \frac{1}{2}$ and $1 - p = \frac{1}{2}$. In that case we can have $\sqrt[2]{n}$ teams with $\sqrt[2]{n}$ workers each.

4) give weight 2 to and-work and weight 1 to or-work. In this case $p = \frac{1}{3}$ and $1 - p = \frac{2}{3}$. In that case we can have $\sqrt[3]{n}$ teams with $\sqrt[3]{n^2}$ workers each.

5) give weight 1 to and-work and weight 2 to or-work. In this case $p = \frac{2}{3}$ and $1 - p = \frac{1}{3}$. In that case we can have $\sqrt[3]{n^2}$ teams with $\sqrt[3]{n}$ workers each.

Benchmarks	Speedup with weighting, p =					Speedup with Reconfigurer
	0	$\frac{1}{3}$	$\frac{1}{2}$	$\frac{2}{3}$	1	
ndetdet	2.037	4.673	5.913	7.971	8.487	9.831
mixed	0.980	1.920	2.642	3.819	3.584	4.004
detndet	1.044	1.622	2.052	2.387	2.032	3.157
bqu8	1.527	2.858	3.937	5.487	7.516	4.474
chat	0.666	1.578	2.502	3.866	7.334	7.034
cypher	1.147	2.164	2.875	4.046	5.922	6.116
mutest	0.584	1.386	2.071	3.450	5.959	6.025
bqu6	1.499	2.562	2.940	3.132	3.137	2.920
flypan2	6.203	4.105	3.507	1.964	1.007	6.307
bt_cluster	6.002	4.353	3.947	1.995	1.011	5.793
flypan4	5.079	3.725	3.305	1.947	1.062	4.961
scanner	5.009	5.460	4.992	4.087	3.226	5.414
road_markings	4.887	3.452	2.479	1.790	0.965	4.269
bcnet	3.226	3.763	3.336	2.767	2.033	3.942
cross11	2.666	1.647	1.615	1.669	1.112	2.917
flypan5	2.629	2.878	2.506	2.139	1.335	3.382
cross6	2.529	2.260	2.095	1.342	0.964	2.260
H.mean	1.656	2.505	2.787	2.574	1.836	4.258

Table 1: ANDORRA-I IN THE PAST X ANDORRA-I WITH THE RECONFIGURER, AT 10 WORKERS

As an example, for 10 workers, the system would set a configuration of a team with 10 masters for weighting 1, 1 team of 1 master and 9 slaves for weighting 2, 3 teams of three masters with the first master having three slaves and the two remaining masters having two slaves each for weighting 3, two teams with 2 masters with 4 slaves each for weighting 4, and 5 teams with 5 masters with 1 slave each for weighting 5. The size of the teams for different choices gives the desired balance to exploit and-parallelism and or-parallelism.

Table 1 shows speedups achieved at 10 processors with the plausible fixed configurations, and the speedups achieved with the reconfigurer. The five middle columns of the table give the speedups of the benchmark set for the five plausible fixed configurations, in increasing order of p. The column `Reconfig` gives the speedups achieved by the reconfigurer. Shaded rectangles show the best speedups achieved at different fixed configurations. The benchmarks are presented in the following order: artificial programs, programs that contain predominantly or-parallelism in decreasing order of amount of or-parallelism, and programs that contain predominantly and-parallelism, in decreasing order of amount of and-parallelism. The last row of the table shows the harmonic mean over all speedups. The reason for using the harmonic mean in this context is to find an overall mean performance for all benchmarks, giving equal weight to each individual benchmark.

For our benchmark set, despite the fact that we have reconfiguring overheads with the work-guided strategy, the work-guided strategy is consistently close to or better than the best of the five fixed configurations. The reconfigurer reaches an overall result that is around 55% better (≈ 1.5 times faster) than the best result produced with a single fixed configuration (which is given by the choice equal weights to both and- and or-parallelism). It is interesting to note that $p = \frac{1}{2}$ gives the best performance overall for the old Andorra-I, but is not the best individually for any of the benchmarks.

From the figures shown in table 1, we can summarise the following:

- For computations that contain or-parallelism only, `chat` and `mutest`, the reconfigurer performs similarly to a fixed configuration of n teams with one worker each (choice of user 1), which means that the overhead of reconfiguring slaves into masters is negligible.

- For computations that contain and-parallelism only, `bt_cluster`, the reconfigurer performs similarly to a fixed configuration of workers in a single team (choice of user 2), which means that there was no overhead for reconfiguring during the computation.

- For computations with high degree of parallelism and distinct phases of computation, and-parallel and or-parallel phase, despite the reconfiguring overheads, the work-guided strategy performs similarly or better than the fixed configurations. This is shown for computations `ndetdet`, `detndet`, `flypan2`, and `flypan5`.

- For computations that contain mainly one form of parallelism, but with small amounts of the other form, sometimes the work-guided strategy does not perform so well as one of the fixed configuration. This is shown for computations `bqu8` and `road_markings`. The difference is very significant for `bqu8`. In other cases, as for `mixed`, `cypher`, `flypan4` and `scanner`, the performance obtained with the reconfigurer is comparable with Andorra-I without the reconfigurer.

- For computations with a low degree of parallelism, e.g. `bcnet` and `cross11`, the reconfigurer performs slightly better than any of the fixed configurations. For others like `bqu6`, one of the fixed configurations performs slightly better.

In summary, we can conclude that Andorra-I with dynamic reconfiguration is overall far better than any single fixed configuration, and even on individual benchmarks is generally better than any fixed configuration that the user might plausibly choose. Moreover, this was achieved *automatically* without any user intervention.

4.2 Performance Evaluation of the Work-Guided Strategy

After showing that Andorra-I with the reconfigurer performs much better than Andorra-I without the reconfigurer, in this section we intend to evaluate how good is the performance of the work-guided strategy compared with the best performance we might hope to achieve. This study is very important, since we are not only interested in showing the benefits of using dynamic reconfiguration in Andorra-I, but also we are interested in obtaining the best possible performance.

There are several ways that can be used to evaluate the performance of the work-guided strategy. One of them is to use an analytical model to evaluate if the performance of the system corresponds to the optimal model of each computation. Another method is to find the optimal performance of a computation, for limited and unlimited number of processors by simulating the parallel model. Yet another method is to evaluate the performance through pure measurement, i.e., we have two different systems, Andorra-I without the reconfigurer and Andorra-I with the reconfigurer, we collect results from these two different systems and compare both performances.

The first method consists of devising a mathematical model for the computations, where a mathematical formula would give an estimate of a possible optimal speedup for a computation. This method is not suitable for logic programs, because the computations are very irregular, therefore it would be difficult to find a good mathematical model to describe the computational behaviour of a program.

The second method is a whole subject by itself, and not very trivial. The idea is to simulate the parallel model and estimate optimal speedups for the computations, with limited and unlimited numbers of processors. The simulation can be done at a high level, i.e., only simulate the conceptual model, or it can be done at a low level. Low level simulation is more realistic because it takes into account implementation issues of the system. Work done by Shen [18] provides such a tool, but for studying the exploitation of or-parallelism combined with non-determinate independent and-parallelism, which is a very different model from what we are using. Also Fernandez [11] studies ideal speedups for the same model. Both of them predict performance of a model that combines or-parallelism with independent and-parallelism at unlimited number of processors, and at a limited number of processors by finding a quasi optimal schedule to run the applications. Sehr [17] uses a yet different method to achieve the same goal, by annotating the Prolog program with special predicates and extra argument variables to pass around goal and clause time information.

The third approach consists of evaluating the results by pure measurement by comparing our new Andorra-I system with the old Andorra-I system. Although this method does not evaluate the system with respect to the best optimal achievable speedups, it is still useful as an evaluation method. As we do not have a tool to estimate the performance of the Andorra model or of the Andorra-I system, we limited our study of performance to measurements of performance for different fixed configurations of workers, or in other words, different versions of the Andorra-I system. Our ambitious goal is to achieve at least the best performance achievable with any fixed configuration, which is a difficult target given that the reconfigurer is dynamic (incurs overheads), and sometimes does not make the right choices due to the instant measures taken for the amounts of and-work and or-work. We will call it the *target performance*. We define the target performance as the one produced when we choose the best fixed configuration of workers into teams for each computation at each number of processors. As most logic programs of our benchmark set can run reasonably well with the best configuration of workers into teams, if we can produce results similar or better than the ones achieved by this best fixed configuration, then we are well enough able to say that our objectives are completely satisfied.

Therefore, we compare the performance of the work-guided strategy with what we defined as our target performance. This target performance was determined by running all programs with all possible fixed teams configurations, and getting the best performance for each configuration. As an example, we show in table 2 speedups achieved for the best fixed teams configuration of 10 workers for each of our benchmarks. The second column of the table (`Target/Cfg`) shows the speedups achieved by the best fixed configuration. The best fixed configuration is shown between parentheses. For example, program `detndet` achieves its best speedup, at 10 processors, with a fixed configuration of eight masters and two slaves, i.e., eight teams, with two of them with two workers and the remaining six with only one worker. The third column of the table (`Reconfig`) shows the speedups achieved with the work-guided strategy. The results shown for the reconfigurer were produced with a single version of Andorra-I, where the reconfigurer starts from a configuration of only one team of workers. The last row of the table (`H.mean`) shows the harmonic mean over all the benchmarks.

The overall result shown by row `H.mean` confirms that the work-guided strategy performs similarly to the target performance. For each benchmark individually, the work-guided strategy performs better than the target performance for 10 benchmarks. For the remaining 7 benchmarks, the work-guided strategy does not reach our target performance, although the results are still good, given that it is not an easy task to allocate dynamically the right numbers of workers to the varying parallel work available.

	Target/Cfg		Reconfig
ndetdet	8.487	*(10M)*	9.831
mixed	3.997	*(6M4S)*	4.004
detndet	3.012	*(8M2S)*	3.157
bqu8	7.516	*(10M)*	4.474
chat	7.334	*(10M)*	7.034
cypher	5.922	*(10M)*	6.116
mutest	5.959	*(10M)*	6.025
bqu6	3.950	*(9M1S)*	2.920
flypan2	6.203	*(9S)*	6.307
bt_cluster	6.002	*(9S)*	5.793
flypan4	5.079	*(9S)*	4.961
scanner	5.485	*(2M8S)*	5.414
road_markings	4.887	*(9S)*	4.269
bcnet	3.746	*(2M8S)*	3.942
cross11	2.666	*(9S)*	2.917
flypan5	2.862	*(2M8S)*	3.382
cross6	2.529	*(9S)*	2.260
H.mean	4.407		4.258

Table 2: BEST FIXED CONFIGURATION X RECONFIGURER, AT 10 WORKERS

Next we evaluate the work-guided strategy up to 15 processors. In an earlier paper [8] we described briefly the initial work-based strategy and showed some results for a very small number of processors and different benchmarks, for the interpreted-based Andorra-I [8].

In order to reduce the complexity of this section we present graphs for some representative classes of programs of our benchmark set. They fall into six subgroups depending on whether the parallelism is high or low, and whether it is mainly and-parallelism or or-parallelism. For brevity, we call these groups: (1) *only or*, (2) *only and*, (3) *high or, low and*, (4) *high and, low or*, (5) *low or, lower and*, and (6) *low and, lower or*. Group (1) has only or-parallelism. Group (2) has only and-parallelism. Group (3) has a high degree of or-parallelism with low degree of and-parallelism. Group (4) has a high degree of and-parallelism, and low degree of or-parallelism. Group (5) has a low degree of or-parallelism and low degree of and-parallelism, but the amount of or-parallelism is bigger than the amount of and-parallelism. Group (6) has a low degree of and- and or-parallelism, but the amount of and-parallelism is bigger than the amount of or-parallelism.

Figure 2: Work-guided strategy: only or benchmarks

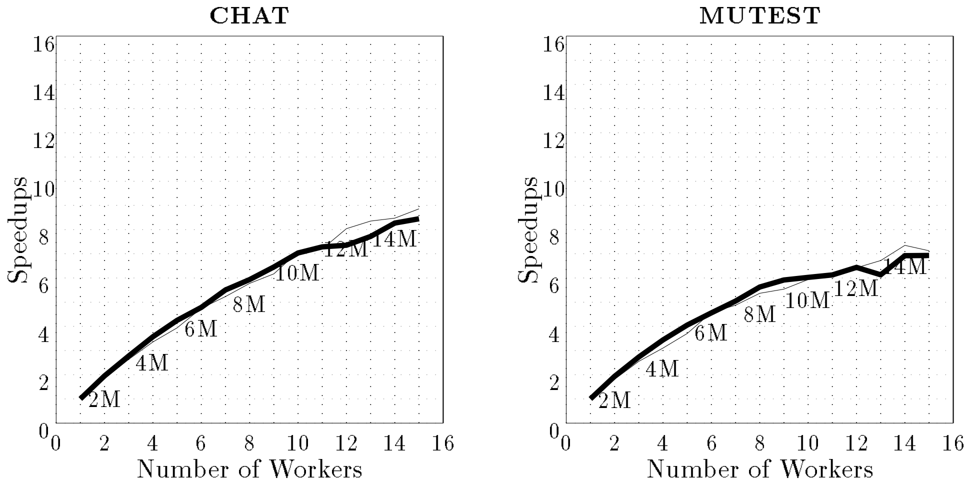
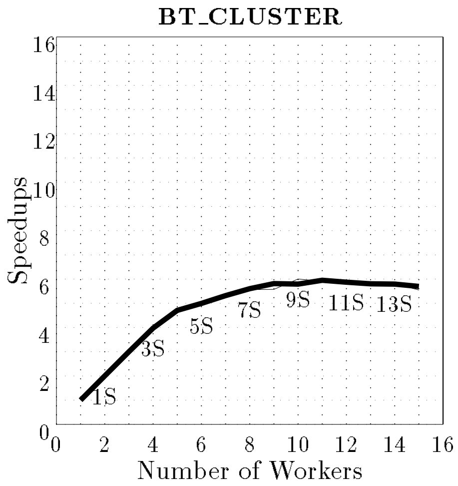


Figure 3: Work-guided strategy: only and benchmark



The graphs in figures 2, 3, 4, 5, 6 and 7 show the comparison between the work-guided strategy and our target performance. The thick line in the graphs represents the performance of the work-guided strategy, while the thin line represents our target performance. Note that for some of the benchmarks, we need different weights for and- and or-, which means that we may need different configurations of masters and slaves at different numbers of processors in order to achieve best speedups with a fixed configuration. The fixed configuration that produced the best speedup is attached to the fixed configuration curve (these are shown as small numbers in the graph. For example, `11M3S` in figure 4 shows that, at 14 processors, the best fixed configuration for the benchmark `ndetdet` is the one with 11 teams, three of them with 2 workers and the remaining with only one worker).

Computations that contain or-parallelism only achieves best performance with a fixed configuration tuned to exploit or-parallelism only, i.e., several teams with only one master (choice of weighting 1). Computations that contain and-parallelism only achieve best performance with a fixed configuration tuned to exploit and-parallelism only, i.e., one team of one master and several slaves (choice of weighting 2). The work-guided strategy performs similarly to these different versions of Andorra-I, as is shown by benchmarks `chat` and `mutest` in figure 2, and benchmark `bt_cluster` in figure 3. This means that the overhead of rearranging workers into teams is very low.

For programs that contain a mixture of both kinds of parallelism, we would like the work-guided strategy to perform similarly or better than the target performance, although this is a difficult target. Indeed the reconfigurer reaches the target performance for most of the benchmark programs that contain a mixture of both forms of parallelism (e.g., `ndetdet` in figure 4, `flypan2` in 5, and `bcnet` in figure 7).

Figure 4: Work-guided strategy: high or, low and benchmarks

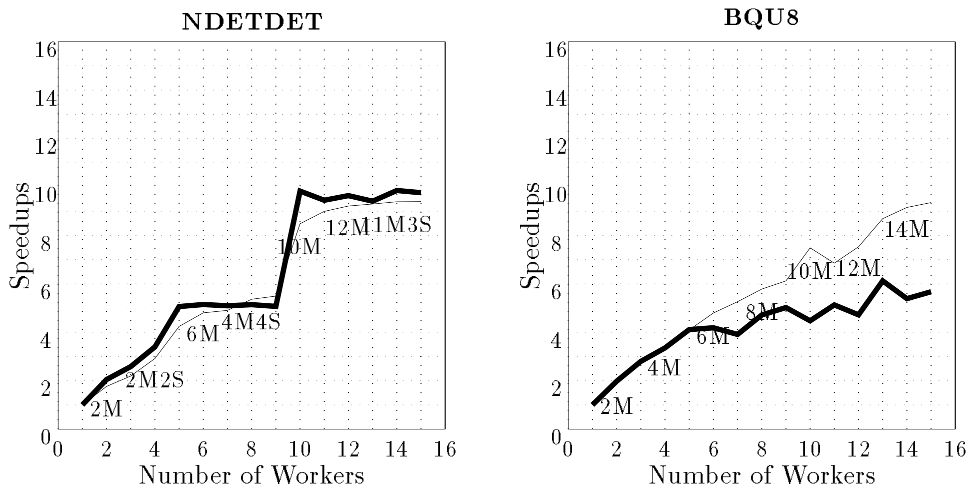
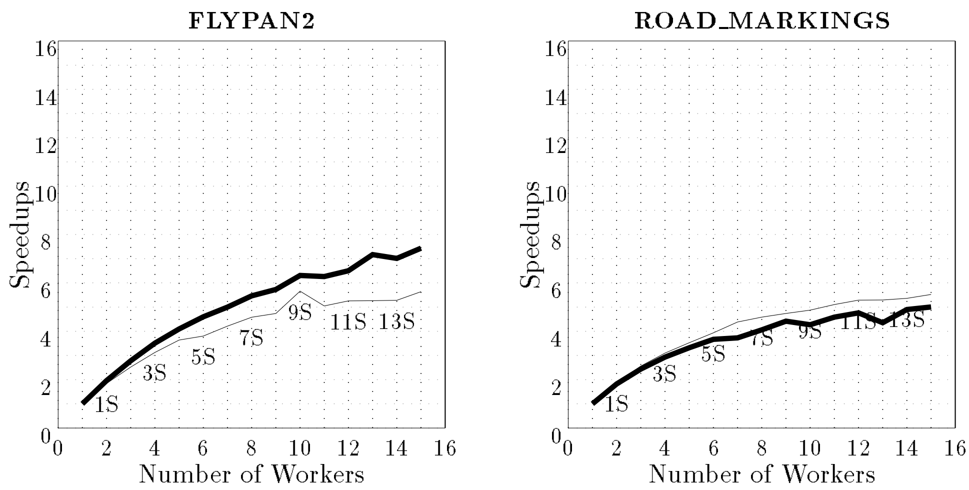


Figure 5: Work-guided strategy: high and, low or benchmarks



The computation `flypan2` (figure 5) is an example of a program where no fixed configuration would achieve best performance, because it contains two distinct phases of computation, an and-parallel phase followed by an or-parallel phase. In practice, with this particular class of computations (that includes all computations that contain distinct phases), we showed that using dynamic reconfiguration of workers into teams we achieve better speedups than setting the best fixed configuration. This is also true for the benchmark `detndet` (figure 6).

Figure 6: WORK-GUIDED STRATEGY: LOW OR, LOWER AND BENCHMARKS

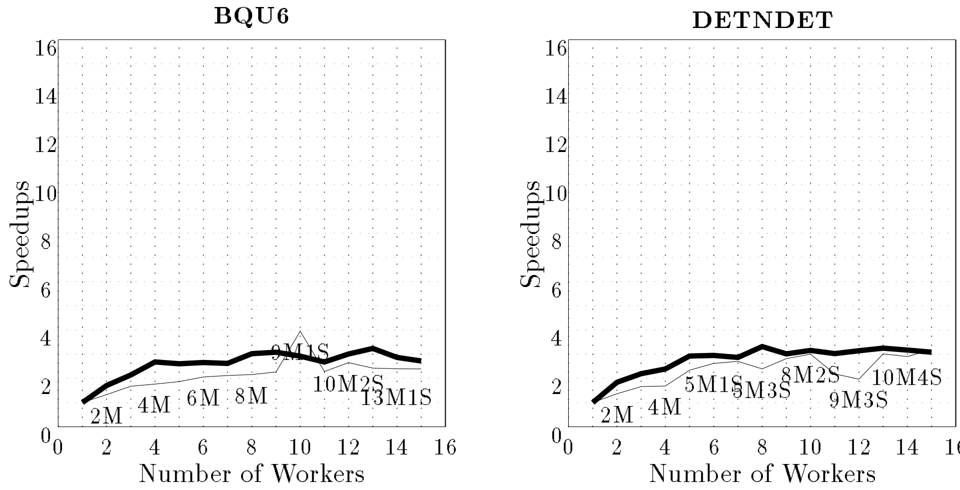
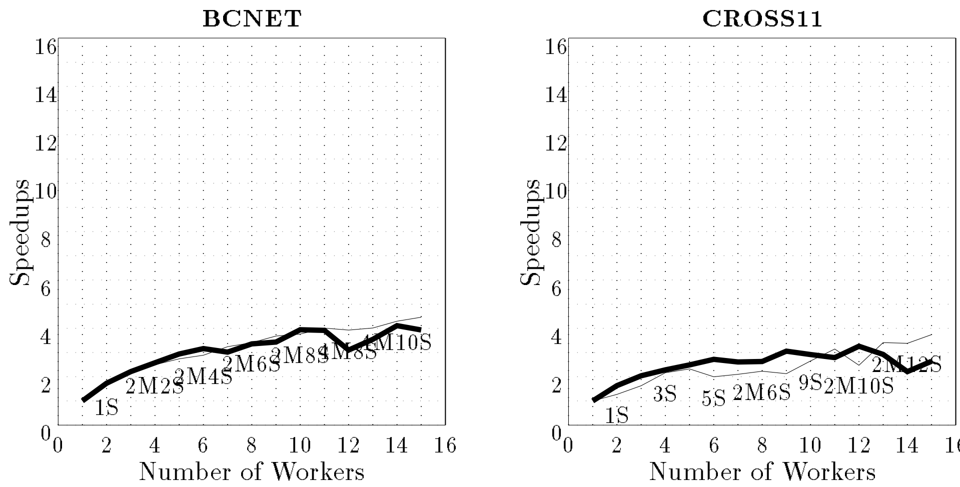


Figure 7: WORK-GUIDED STRATEGY: LOW AND, LOWER OR BENCHMARKS



For the program `flypan2`, we can show by simple analysis that our strategy obtains the best speedup possible with the Andorra-I implementation. As program `flypan2` has two distinct phases of computation, we can say the program is executed optimally in parallel by Andorra-I within the following time:

$$D_p + N_p$$

where D_p is the time taken to execute the and-parallel phase in parallel, and N_p is the time taken to execute the or-parallel phase in parallel, i.e., the determinate phase taking D_p time units to finish and the non-determinate phase taking N_p time units to finish. Let D_s be the time to execute the and-parallel phase sequentially, and N_s be the time to execute the or-parallel phase sequentially. From our experiments with Andorra-I, taking as an example runs for 10 processors, we have the following time information for this program:

(1) $D_s + N_s = 13.727 secs$, this corresponds to the average time taken to execute the program sequentially;

(2) $D_p + N_s = 2.422 secs$, this is the average time taken to execute the program with only and-parallelism, at 10 processors;

(3) $D_s + N_p = 13.646 secs$, this is the average time taken to execute the program with only or-parallelism, at 10 processors.

Adding (2) and (3), we get:

$$D_s + N_s + D_p + N_p = 16.068$$

Subtracting (1), we get,

$$D_p + N_p = 2.341$$

i.e., the time to execute the program optimally with and-parallelism and or-parallelism with 10 processors should be around 2.341 secs. The best fixed configuration of workers in Andorra-I executes this program at 10 processors in time 2.422 secs, while the work-guided strategy produces execution time of 2.172 secs. This means not only that the reconfigurer produces a better speedup than the target speedup, but it manages to achieve slightly better speedups than the best predicted by the preceding analysis. This is explained by the fact that, if there is little and-parallelism available, and there are more workers in a team than necessary to exploit the parallelism, Andorra-I behaves very inefficiently. Idle slaves in a team can steal the work of any other worker, and the execution may be swapped from one worker to another very frequently. The immediate disadvantage of this phenomenon is that processors need to reload their caches frequently. While using fixed configuration of workers, Andorra-I incurs more overheads with workers in a team competing to execute the work available. While using dynamic reconfiguring, workers instead of competing to obtain more work in its team, have a chance of being redeployed to elsewhere, and do not disturb the execution of other workers. As the reconfigurer prevents slaves of disturbing each other by redeploying them to elsewhere, and the total cost of reconfiguring is much lower than the costs of sharing parallel and-work or parallel or-work in a fixed configuration, the reconfigurer achieves slightly better speedups than the best possible computed from real runtime units.

If we apply the same reasoning to program `detndet`, the result is similar, with the reconfigurer performing around 10% better in average than the best possible, predicted analytically.

For some of the computations (e.g., `bqu8`, in figure 4), the work-guided strategy achieves good performance, but it does not reach the target performance. This is explained by the fact that the or-parallel phase in these computations is dominant, but there is a very small amount of and-parallelism in the or-parallel branches. Because the reconfigurer uses instant runtime information to redeploy workers, it believes that there is more and-parallelism than the program really contains (although the run queue has plenty of goals, they are finished very shortly), and prevents slaves from becoming masters to exploit the or-parallelism available.

5 Conclusions

In this work we performed two important evaluations. First, we evaluated the benefits of using dynamic reconfiguring in the Andorra-I system. Second, we studied the performance of the work-guided reconfiguring strategy.

We showed how Andorra-I with dynamic reconfiguration compares with Andorra-I with no dynamic reconfiguration using the work-guided reconfigurer as an example of dynamic reconfiguring. We did a comparison for some plausible fixed configurations, where we give different weights to and-parallelism and to or-parallelism. We concluded that Andorra-I with dynamic reconfiguring is much better than Andorra-I without a reconfigurer, for three reasons:

- The user does not have the burden of deciding which configuration to use.
- Overall, the reconfigurer performs much better than any one of the fixed configurations.
- For individual benchmarks, the reconfigurer performs as well as or better than any of the fixed configurations.

We also attempted to evaluate whether the performance of the work-guided strategy is as good as it could be by comparing it with a target performance given by the best speedup achievable by any fixed configuration.

The comparison with the target performance depends on the kind of application. For programs that present only one form of parallelism, the work-guided strategy performed similarly to the target performance, which implies that rearrangement of workers into teams does not incur much overhead. For programs that present both kinds of parallelism with a reasonable grain size, the work-guided strategy performs around 6.5% better in average than the target performance, at 10 processors. For programs that contain both kinds of parallelism, but with fine grain size, the work-guided strategy still performs better than the target performance for some cases, is similar to the target performance in other cases, and does not perform very well for certain cases.

Although dynamic reconfiguring was applied to a particular parallel logic programming system, we believe that the same idea can be applied to other parallel logic programming system that aims to exploit both and- and or-parallelism.

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] Anthony Beaumont, S. Muthu Raman, and Péter Szeredi. Flexible Scheduling of Or-Parallelism in Aurora: The Bristol Scheduler. In Aarts, E. H. L. and van Leeuwen, J. and Rem, M., editor, *PARLE91: Conference on Parallel Architectures and Languages Europe*, volume 2, pages 403–420. Springer Verlag, June 1991. Lecture Notes in Computer Science 506.

[3] J. Briat, M. Favre, C. Geyer, and J. Chassin. Scheduling of or-parallel Prolog on a scaleable, reconfigurable, distributed-memory multiprocessor. In *Proceedings of Parallel Architecture and Languages Europe*. Springer Verlag, 1991.

[4] Keith L. Clark, F. G. McCabe, and S. Gregory. IC-PROLOG – language features. In K. L. Clark and S. Å. Tärnlund, editors, *Logic Programming*, pages 253–266. Academic Press, London, 1982.

[5] William Clocksin. Principles of the DelPhi Parallel Inference Machine. *Computer Journal*, 30(5):386–392, 1987.

[6] J. A. Crammond. Scheduling and Variable Assignment in the Parallel Parlog Implementation. In *Proceedings of the 1990 North American Conference on Logic Programming*, pages 642–657, October 1990.

[7] J. A. Crammond. The Abstract Machine and Implementation of Parallel Parlog. Technical report, Dept. of Computing, Imperial College, London, June 1990.

[8] Inês Dutra. A Flexible Scheduler for Andorra-I. In A Beaumont and G. Gupta, editors, *Lecture Notes in Computer Science 569, Parallel Execution of Logic Programs*, pages 70–82. Springer-Verlag, June 1991.

[9] 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.

[10] Inês Dutra. *Distributing And- and Or-Work in the Andorra-I Parallel Logic Programming System*. PhD thesis, University of Bristol, Department of Computer Science, February 1995. Ph.D. thesis.

[11] M. J. Fernańdez, M. Carro, and M. V. Hermenegildo. IDRA (IDeal Resource Allocation): A Tool for Computing Ideal Speedups. In *ICLP'94 Pre-Conference Workshop on Parallel and Data-Parallel Execution of Logic Languages*, Facultad de Informática, Universidad Politécnica de Madrid, June 1994.

[12] Wai-Keong Foong. Or-Parallel Prolog with Heuristic Task Distribution. In *Lecture Notes in Artificial Intelligence 592, Logic Programming Russian Conference*, pages 193–200, 1991.

[13] 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.

[14] Manuel V. Hermenegildo. Relating Goal Scheduling, Precedence, and Memory Management in AND-parallel Execution of Logic Programs. In Jean-Louis Lassez, editor, *Logic Programming: Proceedings of the Fourth International Conference, Volume 2*, pages 556–575. The MIT Press, 1987.

[15] Ewing Lusk, David H. D. Warren, Seif Haridi, et al. The Aurora Or-parallel Prolog System. In *Proceedings of the 1988 International Conference on Fifth Generation Computer Systems*, pages 819–830. ICOT, Tokyo, Japan, November 1988.

[16] 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.

[17] David C. Sehr and Laxmikant V. Kalé. Estimating the Inherent Parallelism in Prolog Programs. In *Proceedings of the 1992 International Conference on Fifth Generation Computer Systems*, pages 783–790. ICOT, 1992.

[18] Kish Shen. *Studies of And/Or Parallelism in Prolog*. PhD thesis, Computer Laboratory, University of Cambridge, 1992.

[19] Raéd Yousef Sindaha. *Branch-Level Scheduling in Aurora: The Dharma Scheduler*. PhD thesis, University of Bristol, Department of Computer Science, In preparation, 1993.

[20] Evan Tick. Compile Time Granularity Analysis for Parallel Logic Programming Systems. *New Generation Computing*, 7(2,3):325–337, 1990.

[21] David H. D. Warren. The Andorra model. Presented at Gigalips Project workshop, University of Manchester, March 1988.

[22] Rong Yang, Tony Beaumont, Inês Dutra, Vítor Santos Costa, and David H. D. Warren. Performance of the Compiler-Based Andorra-I System. In *Proceedings of the Tenth International Conference on Logic Programming*, pages 150–166. MIT Press, June 1993.

[23] 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.