Optimising Parallel Logic Programming Systems for Scalable Machines

Vítor Santos Costa[1] and Ricardo Bianchini[2]

[1] LIACC and DCC-FCUP, 4150 Porto, Portugal
[2] COPPE Systems Engineering, Federal University of Rio de Janeiro, Brazil

Abstract. Parallel logic programming (PLP) systems have obtained good performance on traditional bus-based shared-memory architectures. However, the scalable multiprocessors being developed today pose new challenges. Our experience with a sophisticated PLP system, Andorra-I, demonstrates that indeed performance suffers greatly on modern architectures. In order to improve performance, we perform a detailed analysis of the cache behaviour of all Andorra-I data structures via execution-driven simulation of a DASH-like multiprocessor. Based on this analysis we optimise the Andorra-I code using 5 different techniques. Our results show that the techniques provide significant performance improvements, leading to the conclusion that PLP systems can and should perform well on modern scalable multiprocessors.

1 Introduction

Parallel computers can improve performance of both numerical and symbolic applications. Logic programs are good examples of symbolic applications that often exhibit large amounts of implicit parallelism and that can greatly benefit from parallel computers. Several PLP systems have been developed so far and have obtained good performance for traditional bus-based shared-memory architectures. However, the scalable multiprocessors being developed today pose new challenges, such as the high latency of memory accesses and the demand for scalability.

The complexity of PLP systems and the large amount of data they process raise the issue of whether PLP systems can obtain good performance on these new parallel architectures. In order to address this issue, we experiment with a sophisticated PLP system, Andorra-I [8], that exploits both dependent and-parallelism and or-parallelism. Andorra-I is a particularly interesting example of PLP system, since most of its data structures are similar to the ones of several other PLP systems. Andorra-I has obtained good performance on the Sequent Symmetry, but our experience with it running on modern multiprocessors demonstrates indeed that scalability suffers greatly on these architectures [7].

This paper addresses the question of whether the poor scalability of Andorra-I is inherent to the complex structure of PLP systems or can be improved through careful analysis and tuning. In order to answer this question, we analyse the cache behaviour of all Andorra-I data areas when applied to several different

logic programs. The analysis pinpoints the areas that are responsible for most misses and the main sources of the misses. Based on this analysis we remove the main performance limiting factors in Andorra-I through a small set of optimisations that did not require a redesign of the system. More specifically, we optimise Andorra-I using 5 different techniques: trimming of shared variables, data layout modification, privatisation of shared data structures, lock distribution, and elimination of locking in scheduling.

We present the isolated and combined performance improvements provided by the optimisations on a simulated DASH-like multiprocessor with up to 24 processors. In isolation, shared variable trimming and the modification of the data layout produced the greatest improvements. The improvements achieved when all optimisation techniques are combined are substantial. A few of our programs approach linear speedups as a consequence of our modifications. In fact, for one program the speedup of the modified Andorra-I is a factor of 3 higher than that of the original version of the system on 24 processors. Our main conclusion is then that, even though PLP systems are indeed complex and sometimes irregular, these systems can and should scale well on modern scalable multiprocessors.

2 Methodology

In this section we detail the methodology used in our experiments. The experiments consisted of the simulation of the parallel execution of Andorra-I [8, 10]. The Andorra-I parallel logic programming system employs a very interesting method for exploiting and-parallelism, namely to execute *determinate* goals first and concurrently, where determinate goals are goals that match at most one clause in a program. The Andorra-I system also exploits or-parallelism that arises from the non-determinate goals.

Andorra-I requires access to shared memory for both execution and scheduling of work. In order to study the Andorra-I execution more fully, we divided its shared memory into ten different areas. Andorra-I implements the standard Prolog work areas. The *Code Space* includes the compiled code for every procedure and is read-only during execution of our benchmarks. The *Heap Space* stores structured terms and variables. The *Goal Frame Space* keeps the goals yet to be executed. The *Choicepoint Stack* maintains alternatives for open goals. The *Trail Stack* records any conditional bindings of variables.

Andorra-I also requires several new areas for and/or parallelism. The *Or-Scheduler Data Structures* are used to manage or-parallelism. The data structures for and-parallelism are in the *Worker* area. The *Binding Arrays* are used to implement the SRI model [5] for or-parallelism, by storing conditional bindings. A *Lock Array* was needed in our port to establish a mapping between a shared memory position (such as a variable in the heap) and a lock. Finally, the *Miscellaneous Shared Variables* include the remaining data structures.

To simulate Andorra-I we ported the system to a detailed on-line, execution-driven simulator. The simulator uses the MINT front-end [9], and a back-end

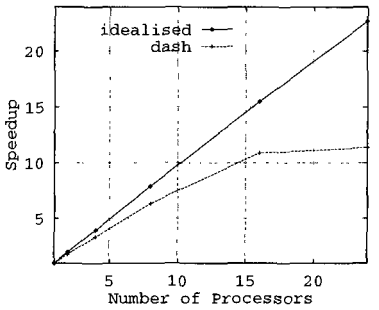

Fig. 1. Speedups for `bt-cluster`

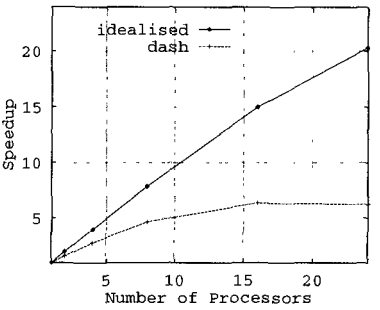

Fig. 2. Speedups for `tsp`

that simulates the memory and interconnection systems. We simulate a 24-node, DASH-like [4], directly-connected multiprocessor. Each node of the simulated machine contains a single scalar processor, a write buffer, a 128-KB direct-mapped data cache with 64-byte cache blocks, local memory, a full-map directory, and a network interface. We use the DASH write-invalidate protocol with release consistency [3] in order to keep caches coherent. We classify cache misses under this protocol using the algorithm presented in [1].

3 Workload and Original Performance

The benchmarks we used in this work are applications representing predominantly and-parallelism, predominantly or-parallelism, and both and- and or-parallelism. We next discuss application performance for the original Andorra-I (more detailed information on the benchmarks can be found in the extended version of this paper [6] and in Dutra's thesis [2]). Note that our results correspond to the *first* run of an application; results would be somewhat better for other runs.

We use two example *And-parallel* applications, the clustering algorithm for network management from British Telecom, `bt-cluster`, and a program to calculate approximate solutions to the traveling salesman problem, `tsp`. To obtain best performance, we rewrote the original applications to make them determinate-only computations.

Figure 1 shows the `bt-cluster` speedups for the simulated architecture as compared to an idealised shared-memory machine, where data items can always be found in cache. The `idealised` curve shows that the application has excellent and-parallelism and can achieve almost linear speedups up to twenty four processors. Unfortunately, performance for the DASH-like machine is barely acceptable. Figure 2 shows that the `tsp` application achieves worse speedups than `bt-cluster` on a modern multiprocessor. The maximum speedup actually decreases for 24 processors, whereas the ideal machine would achieve a speedup of 20 for 24 processors. Figure 3 illustrates the number and sources of cache misses

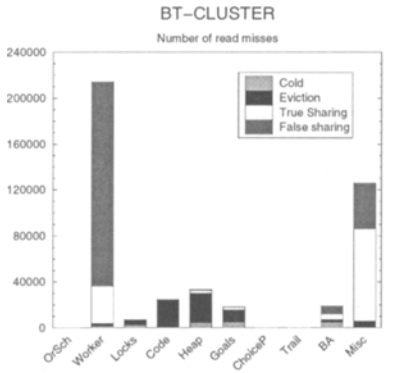

Fig. 3. Misses by data area for `bt-cluster`

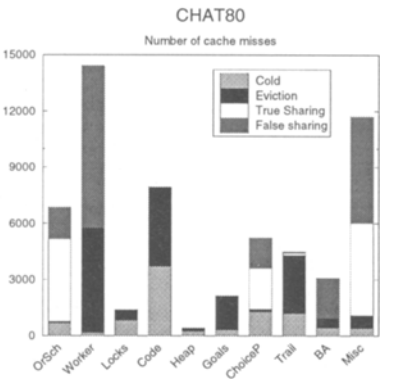

Fig. 4. Misses by data area for `chat80`

per data area in the `bt_cluster` application running on 16 processors as a representative example. The Figure shows that the overall miss rate of `bt-cluster` is dominated by true and false sharing misses from the `Worker` and `Misc` areas. This suggests that the system could be much improved by reducing false sharing and studying activity in the `Worker` and `Misc` areas.

We use two *Or-parallel* applications. Our first application, `chat80`, is an example from the well-known natural language question-answering system `chat-80`, written at the University of Edinburgh by Pereira and Warren. The second application, `fp`, is an example query for a knowledge-based system for the automatic generation of floor plans. This application should at least in theory have significant or-parallelism. Figure 5 shows the speedups for the `chat80` application from 1 to 24 processors. These speedups are very similar to those obtained by Andorra-I on the Sequent Symmetry architecture. In contrast, the DASH curve reaches a maximum speedup of 4.2 for 16 processors. Figure 6 shows the speedups for the `fp` application. The theoretical speedup is very good, in fact quite close to linear, in sharp contrast to the actual speedup for the DASH-like machine.

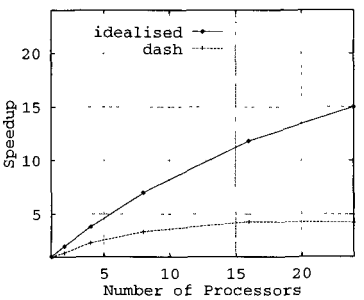

Fig. 5. Speedups for `chat80`

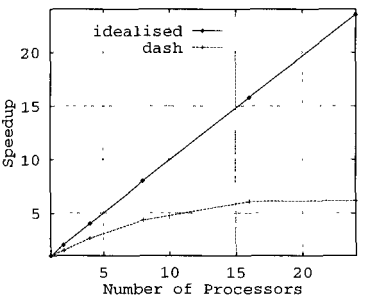

Fig. 6. Speedups for `fp`

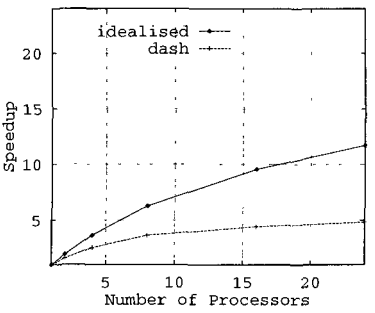

Fig. 7. Speedups for pan2

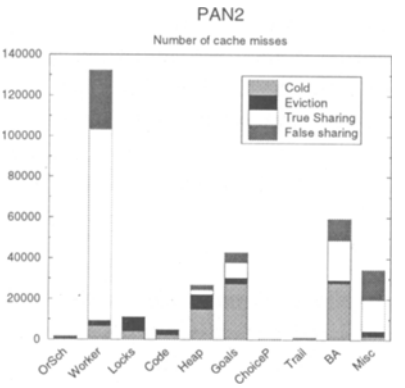

Fig. 8. Misses by data area for pan2

Figure 4 shows the number and source of misses for `chat80` running on 16 processors, again as an example of this type of application. Note that `chat80` does not have enough parallelism to feed 16 processors, suggesting that most sharing misses should result from or-parallel scheduling areas, `OrSch` and `ChoiceP`. Indeed, the figure shows that these areas are responsible for a large number of sharing misses, but the areas causing the most misses are `Worker` and `Misc` as in the and-parallel applications, indicating again that these two areas should be optimised.

As an example of *And/Or* application we used a program to generate naval flight allocations, based on a system developed by Software Sciences and the University of Leeds for the Royal Navy. Figure 7 shows the speedups for `pan2`. The `idealised` curve shows that the application has less parallelism than all other applications; the ideal speedup does not even reach 12 on 24 processors. When run on the DASH simulator, `pan2` exhibits unacceptable speedups for all numbers of processors; speedup starts out at about 1.8 for 2 processors and slowly improves to a maximum of only 4.8 for 24 processors. Figure 8 shows the distribution of cache misses by the different Andorra-I data areas for 16 processors. In this case, the `Worker` area clearly dominates, since the contribution from the `Misc` area is not as significant as in the and-parallel benchmarks. Note that there is more true than false sharing activity in `Worker`. The true sharing probably results from idle processors looking for work.

4 Optimisation Techniques and Performance

The previous analysis suggests that relatively high miss rates may be causing the poor scalability of Andorra-I. It is interesting to note that most misses come from fixed layout areas, such as `Worker` and `Misc`, and not from the execution stacks, as one would assume.

We next discuss how several optimisations can be applied to the system, particularly in order to improve the utilisation of the `Worker` and `Misc` areas.

The first two optimisations were prompted by our simulation-based analysis of caching behaviour, and they are the ones that give the best improvement. The other three were based on our original intuitions regarding the system, and were the ones we would have performed first without simulation data. We studied performance for three applications, `bt-cluster`, `chat80` and `pan2`. A detailed discussion of our experiments and results can be found in [6]. In the remainder of this section, we simply summarise the impact of each of the techniques studied when applied in isolation.

Variable Trimming. In this technique we investigate the areas that have unexepected sharing, and try to eliminate this sharing if possible. For two of our applications, the `Misc` area gave a surprisingly significant contribution to the number of misses. The area is mostly written at the beginning of the execution to set up execution parameters. During execution it is used by the reconfigurer and to keep reduction and failure counters. By investigating each component in the area, we detected that the counters were the major source of misses. As they are only used for research and debugging purposes, we were able to eliminate them from the Andorra-I code.

The results in [6] show that `chat80` benefits the most from this optimisation. This is because the failure counter is never updated by and-parallel applications, but often updated by this or-parallel application. The optimisation does not impact the and-parallel benchmarks as much, leading to less than 10% speedup improvements.

Data Layout Modification. All benchmarks but `pan2` exhibit a high false sharing rate, showing a need for this technique. On 16 processors, 15% of the misses in `pan2` are false sharing misses, whereas in the other applications false sharing causes between 40% (`chat80`) and 51% (`bt-cluster`) of all misses. These results suggest that improving false sharing is of paramount importance. According to our detailed analysis of caching behaviour, false sharing misses are concentrated in the `Worker`, `OrSch`, `ChoiceP` and `BA` areas, besides the `Misc` area optimised by the previous technique.

The `Worker` and `OrSch` data areas are allocated statically. This indicates that we can effectively reduce false sharing. We applied two common techniques to tackle false sharing, *padding* between fields that belonged to different workers or that were logically independent, and *field reordering* to separate fields that were physically close but logically distinct. Although these are well-known techniques, padding required careful analysis, as it increases eviction misses significantly. The field reordering technique was not easily applied either, as the relationships between fields are quite complex.

Padding may lead to serious performance degradation for the dynamic data areas, such as `ChoiceP` and `BA`. This restricted our options for layout modification to just field reordering for these areas. The `BA` area was the target of one final data layout modification, since the analysis of cache behaviour surprised us with a high number of false sharing misses in this area for `chat80`. Further investigation showed that this was a memory allocation problem. The engines'

top of stacks were being *shmalloc'ed* separately and appeared as part of the `BA` area in the analysis. This increased sharing misses in the area and was especially bad for the or-parallel applications, as different processor's top of stacks would end up in the same cache line. We addressed the problem by moving these pointers into the `Worker` area, where they logically belong. Our results show that the `bt-cluster` and `chat80` applications benefit the most from this optimisation; speedup improvements can be as significant as 60%. In contrast, the `pan2` application achieves improvements of less than 10% from this optimisation.

Privatisation of Shared Variables. This technique reduces the number of shared memory accesses by making local copies in each node of the machine. In the best case, the shared variables are read-only and hence local copies can actually be allocated in private memory. The high number of references to `Worker` suggested that privatisation could be applied there. In fact, Andorra-I did already use private copies of the variables in `Worker` and there was little room for improvement. The `Locks` and `Code` data areas are the major candidates to privatisation in Andorra-I. The `Locks` area only includes pointers to the actual locks, is thus read-only during execution, and can be easily privatised. Another area that is also read-only during parallel execution of our benchmarks is `Code`. Unfortunately, logic programs in general can change the database and, therefore, update `Code`, making privatisation complex. Our results show that privatisation improves speedups by up to 10% at most and that the impact of this optimisation decreases as the number of processors increases.

Lock Distribution. This technique was considered to reduce contention on accesses to logical variables, and-scheduling, or-scheduling, and stack management. The original implementation used a single array of locks to implement these operations. In the worst case, several workers would contend for the same lock causing contention. To improve scalability, we implemented different lock data structures for different purposes. We expected best results for or-parallel applications, as the optimisation prevents different teams from contending on accesses to logical variables. The cost of this optimisation is that, if the arrays of locks are shared, there will be more expensive remote cache misses. Our results show that the `bt-cluster` and `chat80` applications benefit somewhat from this optimisation, but that the `pan2` application already exhibited a significant number of misses in the `Locks` area and suffers a slowdown.

Elimination of Locking in Scheduling. This technique improves performance in benchmarks with significant and-parallelism by testing whether there is available work, before actually locking the work queue. This modification is equivalent to replacing a test_and_set lock with a test_and_test_and_set lock. This optimisation provides a small speedup improvement for `pan2`, as it avoids locking when there is no and-work. For `bt-cluster` the technique does not improve speedups as this application exhibits enough and-work to keep processors busy.

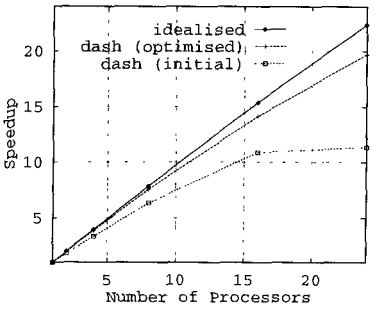

Fig. 9. Speedups for `bt-cluster`

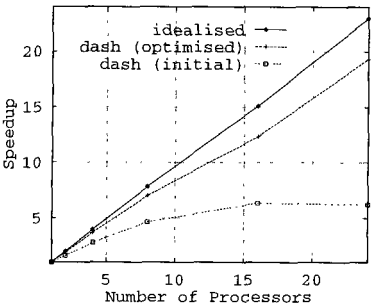

Fig. 10. Speedups for `tsp`

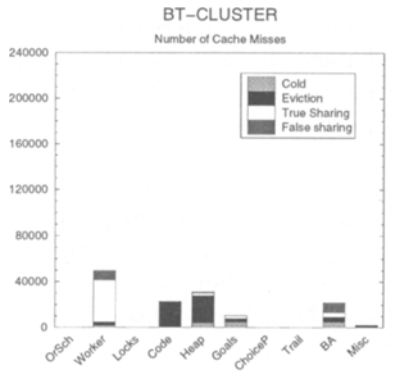

Fig. 11. Misses by data area for `bt-cluster`

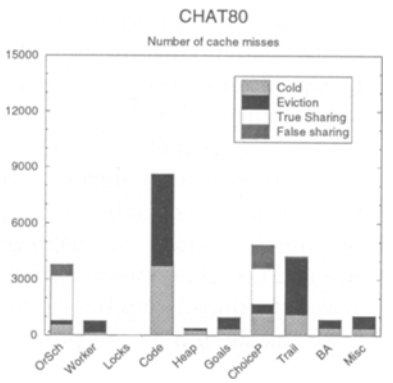

Fig. 12. Misses by data area for `chat80`

5 Combined Performance of Optimisation Techniques

We next discuss the overall system performance with all optimisations combined. We compare speedups against the **`idealised`** and **`original`** results. The **`idealised`** speedups were recalculated for the new version of Andorra-I, but, as it is shown in the figures, the optimisations did not have any significant impact for the **`idealised`** machine. Figures 9 and 10 show the speedups for the two and-parallel applications running on top of the modified Andorra-I system. The maximum speedup for **`bt-cluster`** jumped from 12 to 20, whereas the maximum speedup for **`tsp`** jumped from 6.3 to 19. This indicates that the realistic machine is now able to exploit the available parallelism more fully. The explanation for the better speedups is a significant decrease in miss rates. For **`bt-cluster`**, the new version of Andorra-I exhibits a miss rate of only 0.6% for 16 processors, versus the 1.6% of the previous version. In the case of **`tsp`**, the optimisations decreased the miss rate from 3% to 1.2% again on 16 processors.

Figure 11 shows the number and source of misses for **`bt-cluster`** on 16 processors. Note that the figure keeps the same Y-axis as in Figure 3 to simplify

comparisons against the cache behaviour of the original version of Andorra-I. The figure shows that the number of misses in the `Worker` area was reduced by a factor of 4, while the number of misses in the `Misc` area was reduced by an order of magnitude. The figure also shows that there is still significant true sharing in `Worker`, but false sharing is much less significant. The number of misses from `Misc` is now almost irrelevant.

The or-parallel benchmarks also show remarkable improvements due to the combination of the optimisation techniques we applied. Figure 13 shows the speedups for `chat80` and Figure 14 shows the speedups for `fp`. The maximum speedup for `chat80` almost doubles from one version of the system to the other. Note that speedups for the optimised system still flatten out on 16 processors, but at a much better efficiency. The other benchmark, `fp`, displays our most impressive result. The speedup for 24 processors jumps from 6.2 with the original Andorra-I system to 20 when all our optimisations are applied. This result represents more than a three-fold improvement. Figure 12 shows the distribution of misses for `chat80` with 16 processors. The figure demonstrates that the number of misses in the `Worker` and `Misc` areas was reduced by an order of magnitude. The large number of eviction and cold start misses in the `Code` area remains however. Sharing misses are now concentrated in the `OrSch` and `ChoiceP` areas, as they should.

Figure 15 shows the speedups of the new version of Andorra-I for the `pan2` benchmark. In this case, the improvement resulting from our optimisations was quite small. Figure 16 shows the cache miss distribution for the optimised Andorra-I. The main source of misses was true sharing originating in the `Worker` region. A more detailed analysis proved that these misses originate from lack of work. Workers are searching each other's queues and generating misses. We are investigating more sophisticated scheduling strategies to address this problem.

6 Conclusions and Future Work

Andorra-I is an example of an and/or-parallel system originally designed for traditional bus-based shared-memory architectures. We have demonstrated that

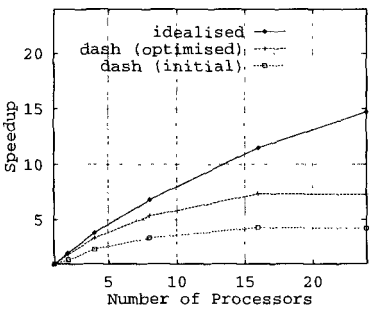

Fig. 13. Speedups for `chat80`

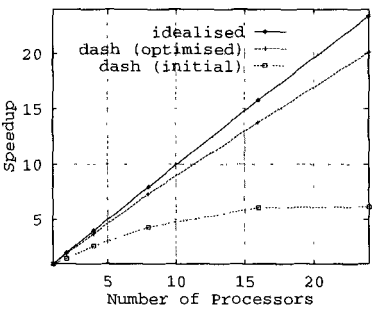

Fig. 14. Speedups for `fp`

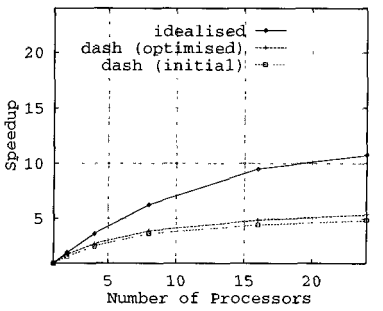

Fig. 15. Speedups for pan2

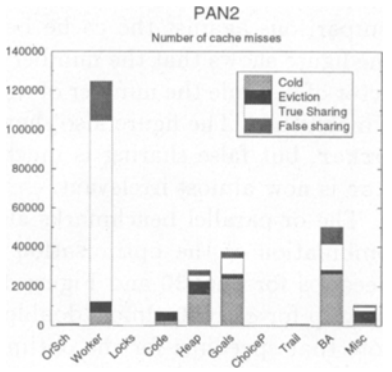

Fig. 16. Misses by data area for pan2

the system can also achieve good performance on scalable shared-memory systems. The key to these results was the extensive data available from detailed simulations of Andorra-I. This information showed that there was no need to restructure the system or its schedulers. Instead, performance could be dramatically improved by focusing on accesses to shared data.

We believe there is potential for improving the performance of PLP systems even further. To prove so will require more radical changes to data structures within Andorra-I itself, as the system was simply not designed for such large numbers of processors. Last, but not least, we are interested in studying performance of other parallel logic programming systems, such as the systems that exploit independent and-parallelism.

Acknowledgements The authors would like to thank Leonidas Kontothanassis and Jack Veenstra for their help with the simulation infrastructure, and Rong Yang, Tony Beaumont, D. H. D. Warren for their work in Andorra-I. This paper results from collaboration work with Inês Dutra, and has benefited from Márcio da Silva's studies. Vítor Santos Costa would like to thank support from the Praxis PROLOPPE and FCT MELODIA projects. Ricardo Bianchini would like to thank the support of the Brazilian CNPq.

References

1. R. Bianchini and L. I. Kontothanassis. Algorithms for categorizing multiprocessor communication under invalidate and update-based coherence protocols. In *Proceedings of the 28th Annual Simulation Symposium*, April 1995.
2. 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.
3. D. Lenoski, J. Laudon, K. Gharachorloo, A. Gupta, and J. Hennessy. The directory-based cache coherence protocol for the DASH multiprocessor. *Proceedings of the*

17th International Symposium on Computer Architecture, pages 148–159, May 1990.

4. D. Lenoski, J. Laudon, T. Joe, D. Nakahira, L. Stevens, A. Gupta, and J. Hennessy. The dash prototype: Logic overhead and performance. *IEEE Transactions on Parallel and Distributed Systems*, 4(1):41–61, Jan 1993.
5. 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.
6. V. Santos Costa and R. Bianchini. Optimising Parallel Logic Programming Systems for Scalable Machines. Technical Report DCC-97-7, DCC - FC & LIACC, UP, October 1997.
7. V. Santos Costa, R. Bianchini, and I. C. Dutra. Evaluating the impact of coherence protocols on parallel logic programming systems. In *Proceedings of the 5th EUROMICRO Workshop on Parallel and Distributed Processing*, pages 376–381, 1997.
8. V. Santos Costa, D. H. D. Warren, and R. Yang. Andorra-I: A Parallel Prolog System that Transparently Exploits both And- and Or-Parallelism. In *Third ACM SIGPLAN PPOPP*, pages 83–93. ACM press, April 1991.
9. J. E. Veenstra and R. J. Fowler. Mint: A front end for efficient simulation of shared-memory multiprocessors. In *Proceedings of MASCOTS '94)*, 1994.
.0. 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.