Evaluating the Impact of Coherence Protocols on Parallel Logic Programming Systems

Vítor Santos Costa*, Ricardo Bianchini, and Inês de Castro Dutra

Department of Systems Engineering and Computer Science
Federal University of Rio de Janeiro
Rio de Janeiro, Brazil
e-mail: {vitor,ricardo,ines}@cos.ufrj.br

Abstract

In this paper we use execution-driven simulation of a scalable multiprocessor to evaluate the performance of the Andorra-I parallel logic programming system under invalidate and update-based protocols. We study a well-known invalidate protocol and two different update-based protocols. Our results show that for our sample logic programs the update-based protocols outperform their invalidate-based counterparts. The detailed analysis of these results shows that update-based protocols outperform invalidate-based protocols regardless of the type of parallelism exhibited by the benchmarks. The reasons for this behaviour are explained in detail. We conclude that parallel logic programming systems can benefit from update-based protocols and that multiprocessors designed for running these systems efficiently should adopt some form of update or hybrid protocol.

1. Introduction

One of the most important advantages of logic programming is the availability of several forms of implicit parallelism that can be naturally exploited on shared-memory multiprocessors. These forms include: or-parallelism, independent and-parallelism, dependent and-parallelism, data-parallelism, and combined and–or parallelism [4]. All the systems implementing these forms of parallelism have been able to obtain good performance on bus-based machines, such as the Sequent Symmetry multiprocessors.

As modern architectures are developed and the gap between CPU and memory speeds widens, the issue arises of whether the current parallel logic programming systems can also perform well on the new, scalable, architectures. In modern multiprocessors, performance depends heavily on the miss rates and may be limited by the communication overhead that is involved in the sharing of writable data.

Sharing in parallel logic programming systems occurs under several circumstances. The use of logical variables for communication in dependent and-parallel applications, for instance, is an example of producer–consumer sharing of data.

A second major form of sharing, migratory sharing, arises from synchronisation between processors. Synchronisation occurs in tasks such as fetching work from other processors, and on being the leftmost goal or branch to execute cuts or side-effects.

The sharing of writable data structures introduces the problem of coherence between the processors' caches. Most parallel machines have used a write invalidate (WI) protocol [5] in order to keep caches coherent. Write update (WU) protocols [8] are the main alternative to invalidate-based protocols.

WI protocols have been more popular than WU protocols because of the extra traffic updates introduce. Quite often the updates sent will not be used by their recipients. This introduces extra, useless, traffic that consumes bandwidth and can actually degrade performance. However, the nature of sharing in parallel logic programming systems suggests that update-based protocols might be more appropriate than their invalidate-based counterparts. In order to confirm this hypothesis, in this paper we evaluate the performance of update and invalidate-based protocols for parallel logic programming systems.

We use execution-driven simulation of a scalable multiprocessor running the Andorra-I system [11]. The system was developed at the University of Bristol by Beaumont, Dutra, Santos Costa, Yang, and Warren [11]. Andorra-I is an ideal subject for our experiments because it supports two rather different forms of parallelism in logic pro-

*On leave from the Universidade do Porto, Portugal. Research supported by the Brazilian Research Council, CNPq, under grant 300953/95-0

grams, and-parallelism and or-parallelism. We use a detailed on-line, execution-driven simulator that simulates a 24-node, DASH-like [7], directly-connected multiprocessor. The simulator was developed at the University of Rochester and uses the MINT front-end [14] (developed by Veenstra and Fowler) to simulate the MIPS architecture, and a back-end [1] (developed by Bianchini, Kontothanassis, and Veenstra) to simulate the memory and interconnection systems. We study a well-known invalidate protocol and two different update-based protocols. More details about the simulator and the Andorra-I system can be found elsewhere [10].

Our approach contrasts with previous studies of the performance of coherence protocols for parallel logic programming systems. Tick and Hermenegildo [13] studied caching behaviour of independent and-parallelism in bus-based multiprocessors. Other researchers have studied the performance of parallel logic programming systems on scalable architectures, such as the DDM [9], but did not evaluate the impact of different coherence protocols. Our goal in this study is to evaluate the performance implications of different protocols for parallel logic programming systems on scalable multiprocessors, while categorising cache misses and update messages in the system.

The paper is organised as follows. Section 2 presents speedup results for the or-parallel, and-parallel, and combined parallel benchmarks ran in Andorra-I under WI and WU protocols. Section 3 evaluates the performance of a hybrid protocol for the same benchmarks. Finally, Section 4 draws our conclusions and suggests future work.

2. Performance Under WI and WU Protocols

In this section we present speedup results for Andorra-I under both kinds of protocol, WI and WU. We experimented with applications representing predominantly and-parallelism, or-parallelism, and both and- and or-parallelism. The benchmarks represent real applications used by companies or in academia. More details about the applications can be found elsewhere [4]. We tried to select applications with good parallelism, whilst avoiding applications with too much parallelism that would scale well regardless of architecture.

2.1. And-Parallel Applications

As an example and-parallel application, we used the clustering algorithm for network management from British Telecom.

Figure 1 shows the speedups for both protocols. The application has excellent and-parallelism, resulting in almost linear speedups for the `perfect` curve. This curve is obtained for an ideal shared-memory machine, where data

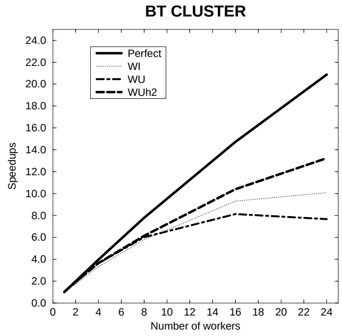

Figure 1. Speedups for `bt-cluster`

items can always be found in cache, and gives an idea of the maximum available parallelism in the application. The Wuh2 curve will be discussed later.

Performance for a realistic machine is acceptable. The invalidate protocol (WI curve) starts with an efficiency of 87% for 2 processors. Efficiency smoothly decreases as the number of processors increases, but is still over 50% for 16 processors. The update protocol (WU curve) has a very interesting behaviour. It starts with excellent efficiency of about 90% up to four processors. Then performance starts degrading, and after 16 processors there is a smooth slowdown. We discuss this problem in detail in the next section.

2.2. Or-Parallel Applications

As our or-parallel application we use an example from the natural language question-answering system `chat-80`.

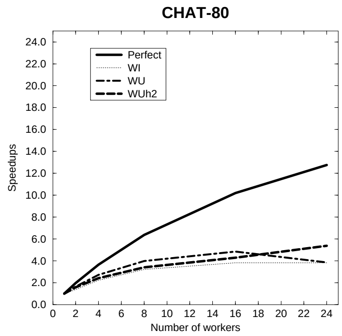

Figure 2. Speedups for `chat-80`

Figure 2 shows the speedups for both protocols, from 1 to 24 processors. The `perfect` curve gives almost linear speedup up to 4 processors, after which the speedup starts to level off. These speedups are very similar to the ones obtained under the Sequent Symmetry architecture [4]. The main difference is that the speedup on the Symmetry levelsoff at a maximum speedup of 10. This comparison shows that the Symmetry behaves almost as if memory accesses were free of cost, which for state-of-the-art processor and memory speeds is unrealistic.

A comparison between the `perfect` and realistic curves proves this point. On this small-scale benchmark, the invalidate protocol manages to obtain a maximum speedup of 4. Up to 16 workers, performance for the update protocol is from 20% to 27% better for this benchmark, obtaining a maximum speedup of 4.9 for 16 workers. It is very interesting to notice that, as the number of workers increases up to 24, performance for the update protocol quickly deteriorates, whereas the invalidate protocol manages to sustain performance.

2.3. And/Or-Parallel Applications

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.

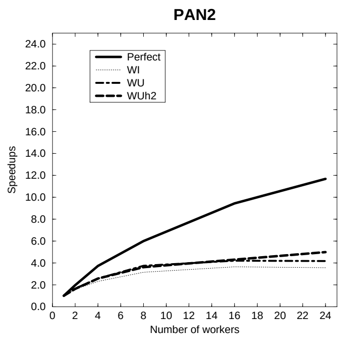

Figure 3. Speedups for `pan2`

Figure 3 shows the speedups for `pan2`. The `perfect` speedups are acceptable. The application performs well up to 8 workers, and then parallelism starts to level off. The performance of `pan2` on the Symmetry is slightly worse than `perfect` [4].

The performance on a realistic machine demonstrates effects similar to the previous applications. Andorra-I on a modern machine cannot match speedups it would obtain under a machine with one-cycle memory access time. The invalidate protocol starts with an efficiency of about 80% for two workers. Efficiency then decreases quickly, but the machine is able to improve speedups up to 16 workers. The update protocol starts with an advantage of 5% for two workers over invalidate. This advantage improves up to 8 workers, and then starts decreasing.

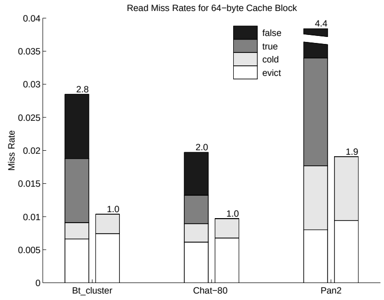

Figure 4. Read miss rate under WI (left) vs. WU (right), 16 processors.

2.4. Analysis of Results

According to the results shown in the previous section, the WU protocol invariably performs better than the WI protocol for small numbers of processors. As we increase the number of processors, performance of the WU protocol starts to degrade. Figures 4, 5, and 6 help to understand this phenomenon. Figure 4 shows the shared read miss rates for the three applications, with WI as the left bar and WU as the right one. The percentage at the top of each column represents the percent of all shared reads that result in a miss. For WI, within a column, misses are classified according to the algorithm described in [3], as extended in [1]. A **cold start miss** happens on the first reference to a block by a processor. A **true sharing miss** happens when a processor references a word belonging in a block it had previously cached but has been invalidated, due to a write by some other processor to the same word. A **false sharing miss** occurs in roughly the same circumstances as a true sharing miss, except that the word written by the other processor is not the same as the word missed on. An **eviction (replacement) miss** happens when a processor replaces one of its cache blocks with another one mapping to the same cache line and later needs to reload the block replaced. Note that the WU protocol does not have sharing misses.

As expected, WU always results in lower read miss rates, even though WI usually exhibits lower replacement miss rates as invalidations effectively free up cache space. The reason is that for WI most misses are sharing misses (from 55%, in `chat-80`, to 70%, in `bt-cluster`). This is sufficient to compensate for the increase in eviction misses from WU, which increases up to 17% for `bt-cluster`. Ultimately, by avoiding sharing misses, WU more than halves the total miss rate for the benchmark set.

Note that for all applications Andorra-I demonstrates a very high percentage of false sharing misses, ranging from 24% of total misses in `pan2` to 34% in `bt-cluster`. False sharing mainly hurts performance under the WI pro-

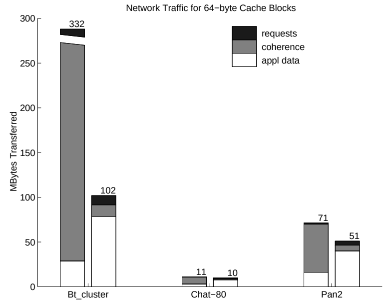

Figure 5. MBytes transferred under WU (left) vs. WI (right), 16 processors.

tocol, by increasing the miss rate. False sharing may arise from shared variables in the schedulers, or from two logical variables that were created in sequence, say, within the same compound term.

Although WU produces lower miss rates than WI, it is at the cost of a large increase in network traffic. This increase can cause several forms of performance degradation due to an increase in network and memory congestion. Figure 5 shows this difference for 16 processors. Basically, WU causes an increase in network traffic with respect to WI that ranges from a factor of 3 in `bt-cluster` to 40% in `pan2` and 11% in `chat-80`.

It is important to note that most of the traffic generated by the WU protocol is coherence-related, i.e. update messages. However, as previous studies have shown [2], most of these updates are useless, at least for scientific applications. In order to determine whether the same is true for Andorra-I, we classify updates as either *useful*, which are needed for correct execution of the program, or *useless*. The latter category of updates is classified according to the algorithm described in [1].

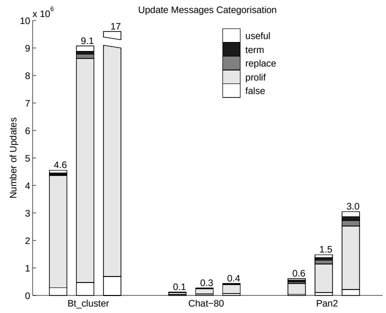

Figure 6. Categorisation of updates, 4 (left), 8 (center), and 16 (right) processors.

True sharing updates are updates where the receiving processor references the word modified by the update message before another update message to the same word is received. **False sharing updates** are updates where the receiving processor does not reference the word modified by the update message before it is overwritten by a subsequent update, but references some other word in the same cache block. **Proliferation updates** are updates where the receiving processor does not reference the word modified by the update message before it is overwritten, and it does not reference any other word in that cache block either. **Replacement updates** are updates where the receiving processor does not reference the updated word until the block is replaced in its cache. A **termination update** is a proliferation update happening at the end of the program.

We can observe from figure 6 that for 16 processors more than 90% of the update messages are indeed useless, with the actual percentage of useful updates varying from only 2.5% in the `bt-cluster` application, to 4.4% in `chat-80`, and 6% in `pan2`. The main culprits are by far the proliferation updates, which grow quickly with the number of processors and eventually become 77% of the useless updates in `chat-80`, 81% in `pan2`, and 93% in `bt-cluster` for 16 processors. A high rate of proliferation updates is typical of migratory sharing of data such as in shared counters, or shared work queues: data structures that are updated very often during execution, and which are indeed very common in both and- and or-schedulers and in memory allocation within a team.

The second most important category of useless updates in all applications is false updates. For 16 processors they are 4% of the total number of useless updates in `bt-cluster`, 7.5% in `pan2`, and 16% in `chat-80`. These updates are the direct consequence of the widespread false sharing in the system. `bt-cluster` is an interesting example where processors do falsely share data, but not simultaneously.

This excessive increase in useless update traffic for large numbers of processors degrades WU performance significantly by increasing memory access times due to network and memory congestion. These results suggest that reducing the number of useless updates should improve WU performance and therefore improve its scalability.

3. A Hybrid Protocol for Andorra-I

In order to reduce the number of update messages of the WU protocol, we experimented with protocols that aim at reducing update traffic. The best performance was under a dynamic hybrid protocol (WUh2) [6] based on the coherence protocols of the bus-based multiprocessors using the DEC Alpha AXP21064 [12]. In these multiprocessors, each node makes a local decision to invalidate or update a cache block when it sees an update transaction on the bus. We associate a counter with each cache block and invalidate the block when the counter reaches the threshold. References to a cache block reset the counter to zero. We used counters with a threshold of 2 updates.

Figures 1, 2, and 3 show speedup curves for the WUh2 protocol. The application `bt-cluster` that contains only dependent and-parallelism (figure 1) has its speedups enormously improved with the hybrid protocol. Speedups for 24 processors increased from 8 with the WU protocol to 13.4 with the WUh2 protocol. Moreover, the knee of the speedup curve, which is reached for 16 workers with WU, is not reached with WUh2 up to 24 processors.

The `chat-80` benchmark (figure 2) performs well under the hybrid protocol. This protocol achieves the best performance overall, being comparable to the WU protocol for smaller numbers of processors, while maintaining rising speedups for the largest number of processors.

Finally, we would expect the `pan2` application (figure 3) to also improve speedups with a change of protocol, as it contains a reasonable amount of and-parallelism. This is indeed the case, although the difference is not as impressive as for `bt-cluster`. WUh2 performs closely to WU up to 16 processors, and then obtains slightly better performance for 24 processors. WUh2 also has better performance than WI for all numbers of processors.

3.1. Analysis of Results

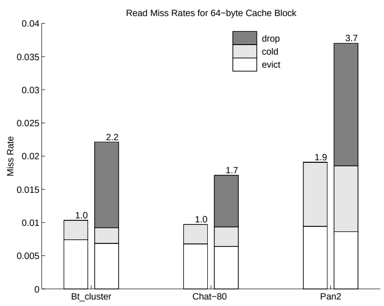

Figure 7. Read miss rate under WU (left) vs. WUh2 (right), 16 processors.

Figure 7 compares the shared read miss rate for WU (left) and WUh2 (right). The cache miss categorisation now includes an extra class, `drop` misses, to account for cache misses resulting from excessively eager self-invalidations. The results show that the price to pay for WUh2 is a significant number of drop misses: 46% of total misses with `chat-80`, 50% for `pan2`, and 58% of total misses for `bt-cluster`. As a result, the read miss rate about doubles in comparison to WU for all applications. In the case of `bt-cluster` the total shared read miss rate is still significantly lower than for WI, 2.2% versus 2.8%. The difference is smaller in `chat-80` and `pan2`. For `chat-80` it is 1.7% versus 2.0%, and for `pan2` it is 3.7% versus 4.4%.

Figure 8 shows the network traffic for all applications under the WU (right) and WUh2 (left) protocols for 16 processors. In all cases there is an increase in application data transfer. The increase is most substantial in `bt-cluster`, where application data transfer more than doubles.

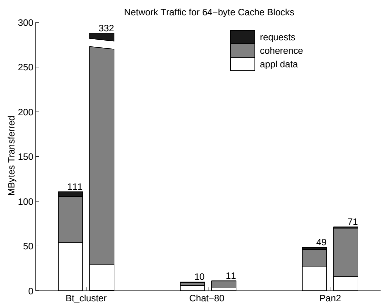

Figure 8. MBytes transferred under WUh2 (left) vs. WU (right), 16 processors.

However, in `bt-cluster` the major effect is the decrease in coherence traffic. The `pan2` benchmark also benefits from the same effect, but in `chat-80` the reduction in coherence traffic is less significant as a percentage of the total traffic. This behaviour of `chat-80` is due to the fact that data sharing is not widespread in or-parallel execution; the vast majority of update operations have a single recipient for all numbers of processors.

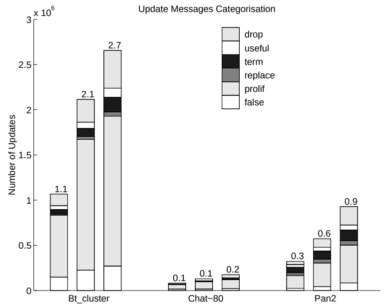

Figure 9. Categorisation of updates, WUh2, 4 (left), 8 (center), and 16 (right) processors.

In figure 9 we present the categorisation of update messages for the WUh2 protocol. The categorisation of updates shown in the figure includes an additional category, `drop` updates, to account for the updates that cause blocks to be invalidated. A comparison between figures 9 and 6 shows that the total number of update messages in WUh2 versus WU at 16 processors has been reduced by a factor of 2 for `chat-80`, 3 for `pan2`, and 6 for `bt-cluster`. This is a direct result of the fact that WUh2 has been very effective at reducing the number of proliferation messages: for 16 processors, the proliferation updates dropped by 90% in `bt-cluster`, by 82% in `pan2`, and by 69% in `chat-80`.

In summary, WUh2 seems to be the most appropriate coherence protocol, as it consistently achieves better scalability than the other protocols, and performs acceptably for small numbers of processors.

4. Conclusions and Future Work

The main conclusion of our work is that parallel logic programming systems that aim at good performance on scalable machines will benefit from some form of update-based protocol. In our benchmarks this was clear both for applications with dependent and-parallelism and or-parallelism. The trend towards faster CPUs, larger caches, and more available bandwidth also favours update-based protocols.

Our results have given a better perspective into the performance of the current version of Andorra-I and how it can be improved. A major result from our analysis is that migratory sharing in the system, not data sharing in the applications, is the most significant form of sharing in Andorra-I. We have also noticed that a substantial part of this sharing is false, and it must come from accesses to static scheduler data structures. We confirmed these results by performing an in-depth analysis of the caching behaviour of the system's shared data structures [10]. This result suggests that to obtain better scalability in Andorra-I the key effort should be on improving scheduling and synchronisation. This is good news, as reducing sharing in the applications would require work from Andorra-I users. We have noticed that the system can be easily improved by reducing false sharing in the schedulers' data structures. More distributed algorithms for fetching work and synchronisation within and between teams will require substantial changes to the Andorra-I's schedulers and to the system's memory management. We are currently investigating new scheduling algorithms for Andorra-I.

In the near future, we will be performing a similar analysis for other parallel logic programming systems. Besides confirming the generality of our claims, such an analysis will give us further insight into the current performance and scalability of parallel logic programming systems.

Acknowledgements The authors would like to thank Leonidas Kontothanassis and Jack Veenstra for their help with the simulation infrastructure used in this paper. The authors would also like to thank Rong Yang, Tony Beaumont, D. H. D. Warren for their work in Andorra-I. Vítor Santos Costa would like to thank the University of Porto and the PROLOPPE project for their support.

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] R. Bianchini, T. J. LeBlanc, and J. E. Veenstra. Categorizing network traffic in update-based protocols on scalable multiprocessors. In *Proceedings of the IPPS'96*, April 1996.

[3] M. Dubois, J. Skeppstedt, L. Ricciulli, K. Ramamurthy, and P. Stenstrom. The detection and elimination of useless misses in multiprocessors. In *Proceedings of the 20th ISCA*, pages 88–97, May 1993.

[4] I. 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.

[5] J. R. Goodman. Using cache memory to reduce processor-memory traffic. In *Proceedings of the 10th International Symposium on Computer Architecture*, pages 124–131, 1983.

[6] A. R. Karlin, M. S. Manasse, L. Rudolph, and D. D. Sleator. Competitive snoopy caching. *Algorithmica*, 3:79–119, 1988.

[7] 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 ISCA*, pages 148–159, May 1990.

[8] E. M. McCreight. The Dragon Computer System, an Early Overview. In *NATO Advanced Study Institute on Microarchitecture of VLSI Computers*, July 1984.

[9] S. Raina, D. H. D. Warren, and J. Cownie. Parallel Prolog on a Scalable Multiprocessor. In P. Kacsuk and M. J. Wise, editors, *Implementations of Distributed Prolog*, pages 27–44. Wiley, 1992.

[10] V. Santos Costa, R. Bianchini, and I. Dutra. Analysing the Caching Behaviour of Parallel Logic Programming Systems. ES-390/96, COPPE/Sistemas, Universidade Federal do Rio de Janeiro, May 1996.

[11] 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 Symposium on Principles & Practice of Parallel Programming*, pages 83–93. ACM press, April 1991. SIGPLAN Notices vol 26(7), July 1991.

[12] C. P. Thacker, D. G. Conroy, and L. C. Stewart. The alpha demonstration unit: A high-performance multiprocessor for software and chip development. *Digital Technical Journal*, 4(4):51–65, 1992.

[13] E. Tick. *Memory Performance of Prolog Architectures*. Kluwer Academic Publishers, Norwell, MA 02061, 1987.

[14] J. E. Veenstra and R. J. Fowler. Mint: A front end for efficient simulation of shared-memory multiprocessors. In *Proceedings of the 2nd International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS '94)*, 1994.