Saving cache memory using a locking cache in real-time systems

A. Martí Campoy, F. Rodríguez-Ballester, R. Ors, J.J. Serrano
Computer Engineering Dept., Technical University of Valencia,
Camino de Vera s/n, 46022, Valencia (Spain)
{amarti, prodrig, rors, jserrano}@disca.upv.es

Abstract – *Locking cache is a practical alternative to conventional caches in real-time systems. With similar performance than conventional caches, a locking cache allows a simple, accurate schedulability analysis. This work presents a new application of the locking cache. Along the modern trend to design Systems-On-a-Chip (SOCs) in which a single IC, usually a programmable device like an FPGA, is designed with one or more microprocessors and peripherals, a locking cache is used to reduce the cache size to the minimum that satisfies the system schedulability. Although results are not as good as the authors expected, the developed technique is promising, and future work may lead to very interesting cost reductions in the size of the memory hierarchy of real-time systems while maintaining their schedulability properties.*

Keywords: Cache memories, Locking cache, Cost aware, Real-Time systems, System on a Chip

1 Introduction

The choice of a processor for the design of a strict real-time system is a complex problem. The designer must estimate the performance required for the tasks to run meeting their respective deadlines. Once the architecture is selected and the system tasks developed, the designer is challenged to verify the system schedulability by any of the usual methods (static analysis, simulation, monitored execution, etc.) [1].

If the test result is negative, i.e. tasks do not meet their time constraints, the designer must modify the tasks to reduce its response time, or choose a new more powerful processor to meet the deadlines of all the tasks.

Choosing a processor with more than enough performance to start the design may seem like a good idea, but it is a way to increase the system cost that, in some cases, may not be acceptable. Using a processor with architectural enhancements such as speculative execution, cache memories, branch prediction, etc. is a good way to achieve higher performance while maintaining the system cost bounded. However, these architectural improvements present a behavior difficult to predict and extremely complex to analyze, which often represents an insurmountable complexity in the schedulability analysis of the real-time system that forces the designer to discard them.

The ability to implement custom processors in FPGAs or ASICs has partly simplified the choice of the processor for a real-time system [2]. With these tools, it is possible to build a system with its performance adjusted to the actual requirements, decreasing the cost of the resulting system. What's more, the designer can incorporate those architectural improvements that interest he most, choosing those with an affordable complexity analysis and a significant increase in performance.

This paper presents a proposal for the use of locking cache memories to ensure that the system is schedulable, but maintaining, on the one hand, the simplicity of the schedulability analysis, and on the other hand, the system cost as low as possible in terms of memory hierarchy. This proposal is most attractive to the designer of ad-hoc processing systems implemented in programmable logic devices like FPGAs.

1.1 Cache memories

Conventional cache memories are one of the most effective ways to enhance the performance of a system [3]. But three problems make their use unattractive in real-time systems:

- Their behavior is difficult to predict, and although there are a large number of publications on this area [4], the complexity of these techniques make their use extremely difficult in real applications.

- The cost of the memory used to build the cache is very high, which makes their use not recommended in systems where the final price is a determining factor.

- The cache memory has to be very close to the processor to get the best performance results, which in practice compels the designer to implement it into the same silicon die, or inside the same FPGA in which the CPU is located. This can significantly increase the cost and complexity of development process if the cache size is relatively large.

1.2 Locking caches

The basic operation of a locking cache memory is simple: a set of selected blocks from main memory are loaded and locked into the cache memory, and a locking mechanism prevents the cache content replacement. Since there are no changes in the contents of the cache, and these contents are chosen at design time, the location of every instruction — cache or main memory— is fixed and a-priori known, so it is its execution time. Thus it is possible to use well-known techniques to estimate the execution and response times of all the tasks, easing the schedulability analysis required for real-time systems.

There are multiple variants of the use of locking cache memories in real-time systems [5, 6, 7, 8, 9, 10]. The most basic form is called static locking. In this technique, a subset of the system memory is chosen to be loaded and locked into the cache memory, remaining there for the lifetime of the system. Other techniques are generically called dynamic locking, since the content locked in cache can be changed during the system runtime, provided that the replacement happens at fixed or known points to maintain the simplicity of the schedulability analysis.

Although static locking delivers worst performance results than dynamic locking techniques [11], several reasons exist that have decided the authors in favor of the static locking technique for the proposal presented here:

- Dynamic locking needs additional hardware, and in some cases an additional memory, to manage the cache contents replacement in a controlled way. Since the aim of this work is to reduce the hardware required by the resulting system it seems appropriate to use the technique which requires no additional hardware.

- With the use of the static locking technique the selected instructions have to be loaded and locked into the cache before the system starts execution. In those cases where the system is implemented in a FPGA or similar programmable device the cache contents can be "hard-coded" in the design and so the cache controller may be greatly simplified (as there is no need to replace its content). This by no means would require a hardware re-design or re-synthesis if the code of the tasks needs modification. For example, the ISE Design Suite software used to program Xilinx FPGA devices offers the possibility to modify the contents of the memory blocks inside the FPGA working directly with the bitstream (the file to download into the FPGA) at the very last step of the design process.

In any case, to achieve the best possible performance with the locking cache, two techniques are used simultaneously:

- To exploit spatial locality for those instructions not loaded into the cache (which produce cache misses at runtime) and reduce the number of memory transfers, a prefetch buffer is included into the cache controller. This buffer has a size equal to a single cache line and allows the system to transfer instructions using bursts of the same length as if a conventional cache memory were used.

- A genetic algorithm is used to perform the selection of those blocks from main memory that have to be loaded into the locking cache.

2 Searching the minimal cache size

The proposal presented in this article is based on the results obtained in previous research works. These results showed that when the size of the system, measured in terms of the memory occupied by the instructions of the tasks, was significantly larger than the size of the cache memory, the performance of the system with a locking cache was far better than using a conventional cache. This suggests that it is possible to maintain the schedulability of the real-time system (with a cache with its content locked at runtime) while reducing significantly the size of the cache memory size.

In those previous works about locking caches, the selection of the blocks to be loaded was performed using a genetic algorithm or a greedy algorithm. The goal of both algorithms was to select a set of blocks to guarantee the system schedulability and, at the same time, to improve some system performance parameter such as the system utilization or the slack of the tasks.

However, the main goal for many real-time designers is to guarantee that the system is schedulable only, with little or no interest in optimizing the performance of the resulting system. Therefore, although the use of a cache memory of minimal size produce a performance loss, savings in energy consumption and system costs compensate it.

The algorithms used to decide the set of blocks which must be loaded into the locking cache use multiple input parameters. These parameters include information about the tasks of the real-time system, about the temporal requirements of the system, and of course, information about the locking cache.

One of these input parameters is the size of the cache memory, precisely a piece of information that in this work should not be an input parameter but a result of the algorithm, as in this case the goal is the search of the minimum locking cache size that guarantee the schedulability of the real-time system. Therefore, selection algorithms must be modified to produce this value instead of requiring it as an input.

However, before facing the change of the search algorithms, the authors have preferred to conduct a study to verify that the use of a static locking cache memory allow the reduction of the cache memory size of schedulable real-time systems.

3 Experiments

3.1 Experiments description

The goal of the experiments carried out is to determine what is the minimum size of the cache memory required to guarantee a real-time system is schedulable, using both a conventional cache and static locking cache. To perform these experiments the following elements have been used:

- 13 sets of tasks have been developed. Each task set is composed by a number of tasks ranging from 3 to 8 tasks. The code for each task has been synthetically created and may contain sequential code, loops, nested loops, if-then-else conditional structures and any combinations of these.

- The processor used is a MIPS R2000 and the following cache characteristics are assumed: a cache line size of 16 bytes (4 32-bit wide instructions). Fetching an instruction from cache takes 1 cycle while fetching an instruction from main memory takes 10 cycles. The mapping functions used for the conventional cache are direct, 2-way set-associative, 4-way set-associative and full associative. The mapping function for the locking cache is only direct-mapping, because this one is the most restrictive for the locking cache.

- To estimate the response time for the tasks using a conventional cache, a modified version of the SPIM simulator (a freely available, widely used MIPS simulator) has been used. Cache effects can be analysed without interference since the simulator does not include any architectural enhancement. From the original version of SPIM, modifications include the simulation of a parametric (size, mapping) cache.

- The genetic algorithm described in has been used to estimate the response time for the tasks when using a static locking cache.

Two factors may have a decisive impact on the performance of a system using a cache memory, with or without locking. The first factor is the structure of tasks code: the use of large sequences of sequential instructions, small nested loops, etc. The second factor is the number of expulsions the tasks suffer at runtime due to higher priority tasks; this affects the tasks response times and the overall utilization of the system.

The structure of the tasks has been taken into account at the time the synthetic tasks were created, making use to different control flow mechanisms. The number of expulsions has been included when two real-time systems originate from the same task set.

Each task set produce 2 different (called type A and type B) real-time systems to schedule for a total of 26 systems. A fixed-priority preemptive scheduler is used in every case where the task priority is assigned according to a Rate Monotonic policy. Also, notice that it is assumed that the deadline for a given task, D, is equal to the task period, T.

Type A and type B systems originating from the same task set differentiate in the periods assigned to the tasks only. Tasks priorities are the same in both cases, but in the type A systems the periods are sufficiently large to achieve an overall system utilization below 50% when running on a conventional cache of 4096 lines. Whereas in the type B systems the periods of all tasks have been reduced, thus increasing the number of expulsions experienced by the lower priority tasks and therefore increasing the overall utilization of the system above 90%, again with a conventional cache of 4096 lines.

The reason to use a 4096-lines cache as a reference is that for any of the set of tasks created the sum of the sizes of the tasks is less than 64Kbytes (4096 blocks of 16 bytes, the size of a cache line) so all systems run without cache misses (except mandatory misses).

On the other hand, why use the same set of tasks with two groups of different time periods is due to the results obtained in previous works. In these studies a strong relationship was found between the performance of the locking cache and the sizes of the tasks and the cache itself.

Specifically, when the cache was very small compared to the size of the tasks, the performance of a locking cache was clearly better than the performance obtained using a conventional cache. Conversely, when the size of the cache was close to the sum of the sizes of the tasks, a locking cache provided worse performance than a conventional cache.

The goal is that a given system requires different caches —in terms of cache size— allowing the evaluation of the effect of the cache size. Achieving such a goal is possible by modifying the tasks periods and therefore the system utilization because to maintain the utilization below 100% (a requirement for the system to be schedulable) it will be

necessary to reduce the execution time of the tasks of the system, reduction that can be obtained increasing the size of the available cache.

Once the tasks for the real-time systems have been created (a total of 13 sets of tasks, each using two sets of periods for a total of 26 systems), the experiments have been carried out this way:

To find the minimum size required to schedule a system using a conventional cache (Minimum Size of Conventional cache, MSconv) these systems have been simulated using a modified version of the SPIM simulator. This version of SPIM gives as result, among other information, the execution and response times of tasks, the overall utilization of the system and an error message if any of the tasks misses its deadline. Each system is simulated with all possible sizes of a conventional cache ranging from a single line to 4096 lines, until the minimum size that guarantees schedulability of the system is found.

To find the minimum size required to schedule a system using a static locking cache (Minimum Size of Locking cache, MSlocking) a genetic algorithm has been run. The genetic algorithm provides several results from the analyzed system, including the overall system utilization, execution and response times of all the tasks, and an error message if the system is not schedulable. All these results are estimates and represent a safe upperbound for the results that would be obtained in the case the system were actually implemented and the locking cache were filled with the blocks selected by the genetic algorithm.

The number of executions, both the modified SPIM simulator and the genetic algorithm, required to obtain the results presented below have been enormous, but this have been automated and reduced through the use of scripts that perform a binary search on the size of the cache memory.

3.2 Experimental results

Table 1 and Table 2 show the results for the 13 sets of tasks and the two sets of periods with the column "Conv. 4096" containing the system utilization when a conventional cache memory of 4096 lines is used to simulate the execution of systems. These data allows comparing the effect of tasks periods and system utilization on the performance of the system using a locking cache.

Columns "Lines conv." and "Lines lock." show the minimum cache size required to make the system schedulable for a conventional and a static locking cache respectively. The absolute difference between these two columns is shown in the "Diff" column. Positive values indicate the locking cache makes the system schedulable with a lower number of lines than the conventional cache. Finally the percentage ("%") show the relative difference in percentage between "Lines conv." and "Lines lock.". Positive values indicate a smaller locking cache is required to schedule the system.

From Tables 1 and 2 the first conclusion is that for a total of 26 systems, the locking cache needs a lower number of lines than conventional cache in 12 cases (46% of total experiments). However, if results are studied as a function of the type of periods, the locking cache beats the conventional cache in 9 of 13 systems when the global utilization is below 50% (type A periods) and the conventional cache beats the locking cache in 10 of 13 systems when global utilization is over 90% (type B periods).

	Type A periods				
Task set	Conv. 4096	Lines conv.	Lines lock.	Diff	%
1	0,35	544	261	283	52,02
2	0,35	549	302	247	44,99
3	0,44	436	450	-14	-3,21
4	0,36	75	105	-30	-40,00
5	0,38	156	150	6	3,85
6	0,32	448	161	287	64,06
7	0,41	488	372	116	23,77
8	0,35	380	400	-20	-5,26
9	0,29	89	14	75	84,27
10	0,28	96	4	92	95,83
11	0,34	377	267	110	29,18
12	0,39	80	129	-49	-61,3
13	0,32	77	60	17	22,08

Table 1. Results for systems with periods Type A

	Type B periods				
Task set	Conv. 4096	Lines conv.	Lines lock.	Diff	%
1	0,96	2219	2313	-94	-4,24
2	0,98	3044	2870	174	5,72
3	0,93	954	1482	-528	-55,35
4	0,96	377	515	-138	-36,60
5	0,97	1414	1880	-466	-32,96
6	0,94	2932	2751	181	6,17
7	0,98	1727	3448	-1721	-99,65
8	0,93	1687	2300	-613	-36,34
9	0,98	572	711	-139	-24,30
10	0,92	1567	1826	-259	-16,53
11	0,92	1831	1756	75	4,10
12	0,97	1012	2975	-1963	-193,97
13	0,95	534	1401	-867	-162,36

Table 2. Results for systems with periods Type B

The effect of the type of periods applied is not definitive, however. For example, for task set number 2, the locking cache beats the conventional cache independently of the type of periods while for task set number 12, the conventional cache beats the locking cache in both cases. The reason of this behavior is the structure of the tasks that form the system. Tasks in set 12 include nested loops up to three levels while tasks in set 2 have only one level of loops.

Figure 1 shows the distribution of reduction/increase (percentage column) of lines needed to make the systems schedulable. Again, positive values (columns on the right side) indicate that the locking cache mechanism demands smaller caches than conventional cache. Figure 2 and 3 show the same frequency histogram, but grouping systems using the type of periods.

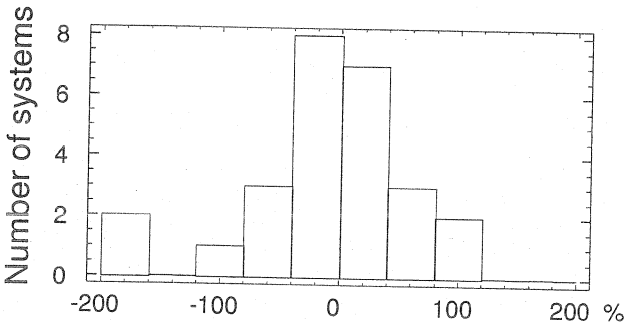

Figure 1. Frequency histogram for all 26 systems.

From Figure 1, we can observe that the reduction or increase or lines needed by locking cache is similar (except for two extreme systems). That is, systems designer may save cache size or may spent more cache size, but in similar quantities.

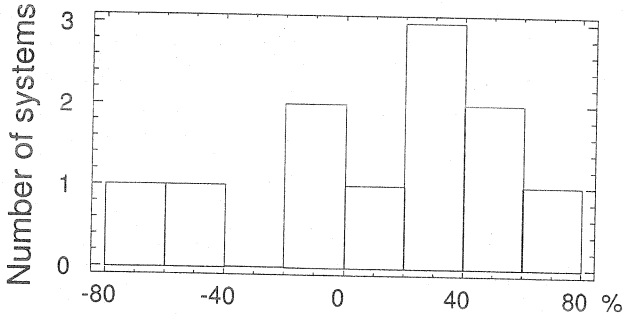

Figure 2. Frequency histogram for systems with type A periods.

Figure 2 shows that the system designer will save more cache memory and in more cases. When the locking cache needs more lines than the conventional cache, this extra amount of cache memory is usually small, lower than the saving for those systems with cache reduction.

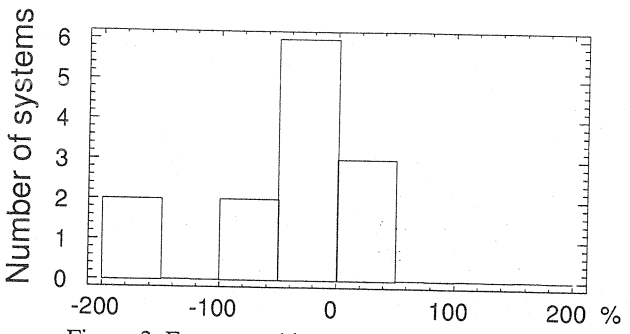

Figure 3. Frequency histogram for systems with type B periods.

Figure 3 shows the opposite results for cases to Figure 2. When system utilization is very high, the locking cache will need greater caches than conventional, and in more cases. This increase is significantly larger than the cache saving in those cases where locking cache behaves better than the conventional cache.

Results are not excellent, but authors think the locking cache may reduce the size of cache memory needed to make the systems schedulable. The conclusion from a detailed analysis is that the genetic algorithm used is to blame. This algorithm is not designed to optimize the cache size which is the way it has been used in this work. When the cache size is around the limit to make the system schedulable, the genetic algorithm behaves in an erratic way because the metrics used to evaluate individuals are not available (global utilization and response time of tasks). So it is clear that a specifically designed genetic algorithm, or any other kind of algorithm, may improve the results shown.

Also, there are dynamic ways to use a locking cache that have demonstrated better performance compared against conventional caches. Those techniques require additional hardware, but the cost of this extra hardware may be lower than the cost of saved cache memory.

4 Conclusions and future work

Locking caches have been shown to be a real and practical option to design real-time systems with the benefits provided by a cache memory but without the drawbacks the use of a conventional cache present when performing the system schedulability analysis.

But this paper has presented a novel application of the use of static locking caches. In systems where the hardware is designed ad hoc, the use of static locking cache may help to reduce costs and allow the designer to adjust the size of the memory cache to the minimum required to guarantee the real-time system is schedulable.

A notable reduction of the cache memory size is possible when the overall system utilization is low if a static locking mechanism is used, as can be seen in the experiment results of the majority of these cases. It is important to highlight that over-dimensioning the cache size (and therefore reducing the system utilization) is a common measure used in the design of real-time systems to tackle the problems a conventional cache imposes to the temporal reasoning of the system.

Future work offers three clear lines to be developed:

First, the design of a genetic algorithm to specifically find the minimum memory size of the cache quickly, efficiently and easily. The aim is to facilitate the work to the system designer, so this new algorithm is a real need if the locking cache is wanted to be of practical use. Moreover, this algorithm could include secondary objectives. That is, in addition to the minimum cache size the algorithm could also reduce the system utilization or improve the slack of the system tasks, within some given limits.

Second, a detailed analysis of the factors that results in the locking cache requiring fewer or more lines than a conventional cache. In this way, the system designer can determine, with a priori knowledge of some characteristics of the system, if the proposed technique is viable or not.

Finally, the evaluation of other, dynamic ways to use the locking cache, and the effect on the resulting performance of the cache main characteristics apart from the memory size: line size and hit and miss times.

5 References

[1] Albert M. K. Cheng. "Real-Time Systems: Scheduling, Analysis, and Verification". John Wiley & Sons, 2003.

[2] Z. Navabi. "Embedded core design with FPGAs." McGraw-Hill, 2007

[3] Hennessy, J. and Patterson, D. "Computer Architecture: A Quantitative Approach". Morgan Kaufmann Publishers, Inc., third edition, 2002.

[4] E. Tamura. "Towards a Predictable, High-Performance Instruction Memory Hierarchy in Fixed-Priority Preemptive Real-Time Systems." Ph. D. Thesis, 2008

[5] A. Martí Campoy, A. P. Ivars, and J. V. Busquets Mataix. "Static use of locking caches in multitask premptive real-time systems" In IEEE/IEE Real-Time Embedded Systems Workshop (Satellite of the IEEE Real-Time Systems Symposium), London, UK, December 2001

[6] Asaduzzaman, A.; Limbachiya, N.; Mahgoub, I.; Sibai, F.N., "Evaluation of I-Cache Locking Technique for Real-Time Embedded Systems," 4th International Conference on Innovations in Information Technology, pp.342-346, Nov. 2007

[7] A. Arnaud, I. Puaut. "Dynamic Instruction Cache Locking in Hard Real-Time Systems." 14th International Conference on Real-Time and Network Systems (RNTS), Poitiers, France, May 2006

[8] Tamura, E., Busquets-Mataix, J.V., Martí Campoy, A. "Towards Predictable, High-Performance Memory Hierarchies in Fixed-Priority Preemptive Multitasking Real-Time Systems" 15th International Conference on Real-Time and Network Systems , pp. 75-8. Nancy, France, March 2007

[9] Xavier Vera, Björn Lisper, Jingling Xue. "Data cache locking for higher program predictability". International Conference on Measurements and Modeling of Computer Systems (SIGMETRICS. Pp. 272-282. San Diego, CA, USA, June 2003.

[10] Staschulat, J. Schliecker, S. Ernst, R. "Scheduling analysis of real-time systems with precise modeling of cache related preemption delay." 17th Euromicro Conference on Real-Time Systems,. Palma de Mallorca, Spain, July, 2005.

[11] A. Martí Campoy, A. Perles, F. Rodríguez, J.V. Busquets Mataix. "Static Use of Locking Caches vs. Dynamic Use of Locking Caches for Real-Time Systems" IEEE Canadian Conference on Electrical and Computer Engineering. Montreal, Canada, May, 2003.

6 Acknowledgments

This work has been supported by MEC under project DPI2007-66796-C03-01 and CICYT under project Tra2006-15620-C02-01.