

STATIC USE OF LOCKING CACHES VS. DYNAMIC USE OF LOCKING CACHES FOR REAL-TIME SYSTEMS

A. Martí Campoy, A. Perles, F. Rodríguez, J.V. Busquets-Mataix
Dep. de Informàtica de Sistemes y Computadores. Univ. Politècnica de Valencia. Spain
{amarti, aperles, prodrig, vbusque}@disca.upv.es

Abstract

Locking caches are a useful alternative to standard cache memories in order to reach both predictability and high performance for multitasking, preemptive, fixed-priority real-time systems. Two schemes of locking cache are possible: static and dynamic use. Both schemas present a high degree of predictability and like-cache performance. But these two schemes are not equivalent. Each one performs better for systems with particular characteristics. This work show that static use presents a greater degree of predictability than dynamic use, but dynamic use offers better performance for the major part of the cases. Systems are grouped as a function of the relationship between cache size and code size, allowing a fast and easy prediction about the gain or loss of performance given by each use of locking caches.

Keywords: *locking cache; schedulability analysis; performance analysis; cache memories.*

1. INTRODUCTION

Cache memories are widely used to reduce the speed gap between processor and main memory. However, their adaptive, dynamic and non-predictable behaviour introduces several problems for their use on real-time systems, where predictability is required. Several solutions have been presented in order to deal with cache memories in such systems. These solutions may be grouped in analysis techniques [1,2,3,4], which incorporate the cache effect into the schedulability analysis; and design solutions [5,6], which propose new cache architectures in order to achieve more predictable cache architectures.

Locking caches belong to the second group. The locking cache can be loaded with selected instructions and then locked, precluding new allocations. Therefore, execution time of each instruction is known a priori, depending if the instruction is loaded in cache or not.

Locking cache may be used in two ways: static [7] or dynamic [8]. Using the static schema, cache contents are preloaded during the system start-up and it remains unchanged for the entire system run. In dynamic use of locking cache, cache contents are changed during system execution, but only in specific points: in this proposal when a task is preempted by another one.

In static use, all tasks compete for the cache space. Cache is loaded with a merge of instructions from all tasks. Once the cache is loaded and locked, its contents remain unchanged. In dynamic use, every task may use the whole cache space. When a task begins or resumes execution after preemption, it loads and locks the cache with its own instructions. The cache remains unchanged until a new task is executed. For both static and dynamic use, the execution time of instructions is constant (from the point of view of memory access), because an instruction will be always or never in cache.

Both uses of locking cache offers several advantages in front of conventional, adaptive and non-predictable cache memories. Since execution time of instructions is now constant, Worst Case Execution Time can be estimated in a simple way, evaluating the longest path by well-known algorithms. Several enhancements to this estimation, like considering infeasible paths, may be directly used; schedulability analysis may be accomplished through Cache Response Time Analysis (CRTA), where cache-refill penalty is tight bounded and known. Other architectural improvements, like segmentation may be incorporated to the schedulability analysis without regard to cache [2], since the execution time of the instruction does not concern interactions between locking cache and these schemes.

Instructions to be preloaded and locked in cache are selected using a genetic algorithm [9]. This algorithm gives, as results, the set of instructions to lock in cache, the response time of each task in the system, and the result of the schedulability analysis evaluating the equation of CRTA.

However, static and dynamic use are not equivalents. The following sections shows a comparison of both schemas from two points of view: determinism and performance.

2. EXPERIMENTS

Results shown in next sections come from a set of experiments, each one of them is composed of a set of tasks and a definite cache size. The tasks used in the experiments are synthetic created in order to stress the locking cache and exercise the genetic algorithm used to select the instructions to load and lock in cache. More than 300 experiments have been defined. For each experiment four kinds of runs have been performed:

- Executing the genetic algorithm for static use of locking cache.
- Executing the genetic algorithm for dynamic use of locking cache.
- Simulating the static locking cache.
- Simulating the dynamic locking cache.

Simulations are accomplished using the SPIM tool [10], a MIPS R2000 simulator, and in all cases the simulation comprises the entire hyperperiod. Using the same set of experiments for both uses allows to make very accurate analysis, since differences in results are due to the schemes presented and not to difference on experiments.

These runs give the response time of each task and global system utilization. Global utilization is calculated considering the worst case for all executions of tasks. This way, global utilization is a safety upperbound of the actual utilization.

3. COMPARISON OF DETERMINISM

The degree of determinism provided by locking cache may be evaluated comparing the response time of each task estimated by the genetic algorithm (R_{es} for static use and R_{ed} for dynamic use) in front of the response time obtained from simulating the locking cache (R_{ss} for static use and R_{sd} for dynamic use). The error in estimating the response time of tasks is defined as the quotient between the estimated response time and the simulated response time: $S_s = (R_{es} / R_{ss}) - 1$ for static use and $S_d = (R_{ed} / R_{sd}) - 1$ for dynamic use. S show the overestimation in percentage, and never can be negative since response time estimated by the genetic algorithm is an upperbound of actual response time when locking cache is used.

Figure 1 and Figure 2 show the histogram of accumulated frequency for S_s and S_d respectively. Y-Axis value is the percentage of tasks with S values lower than x-axis value. Differences are clear. In static use, the error in estimating response time never surpasses 1%, and for more than the 50% of tasks the overestimation is below 0,01%. For dynamic use, near 50% of tasks present an overestimation greater than 1%, and the maximum error is near 40%. This relatively great overestimation in dynamic use is due to cache-refill penalty. Although its

value is tightly bounded, a very small error may led to a large overestimation if the task suffer a high number of preemptions. However, in static use of locking cache its contents never change, this way there is no cache-refill related overestimation.

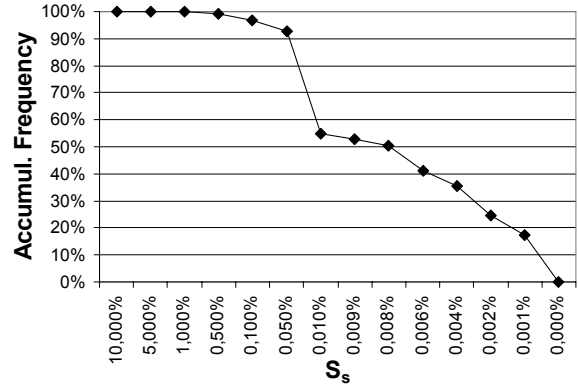


Figure 1. Overestimation in response time of tasks for static use.

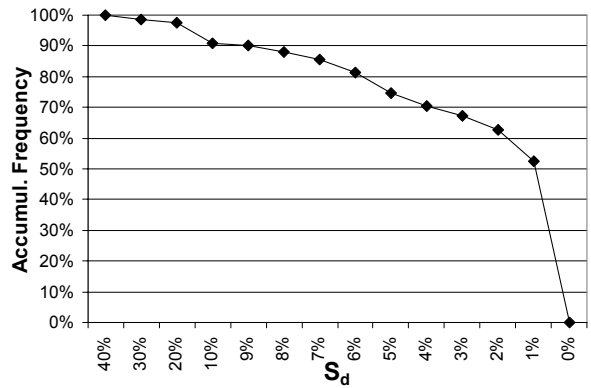


Figure 2. Overestimation in response time of tasks for dynamic use.

4. COMPARISON OF PERFORMANCE

Performance of static and dynamic use of locking cache are compared using system utilization of each experiment. All experiments have been designed to be schedulable, so utilization is a good metric to evaluate, in a global view, the gain or loss of performance of both schemes.

Performance (P) represents the gain or loss of performance obtained for each experiment by using dynamically the locking cache in front of using statically the locking cache. Performance is calculated as $P = U_s - U_d$, where U is the system utilization estimated by the genetic algorithm (U_s for static use and U_d for dynamic use) for each experiment. Values of P over 0 indicate that dynamic use offers better performance than static use,

since system utilization for the dynamic schema is lower than the system utilization for the static schema.

Also, System Size Ratio (SSR) is defined as the relationship between the cache size and the code size (as sum of sizes of all tasks), calculated as $SSR = \text{cache_size}/\text{system_size}$. Intuitively, this relationship is the major factor in the behavior of performance shown by locking cache.

Figure 3 shows the scatterplot of P in front of SSR (X-axis in log scale). Data in this figure may be grouped in three spaces with regard to values of SSR. First space, called E_1 comprises all experiments with SSR values below 0,1. Second space, called E_2 comprises experiments with SSR values equal or greater than 0,1 and lower than 1. Finally, third space, called E_3 , comprises experiments with SSR values equal or greater than 1. Grouping experiments by SSR value allows a very accurate analysis.

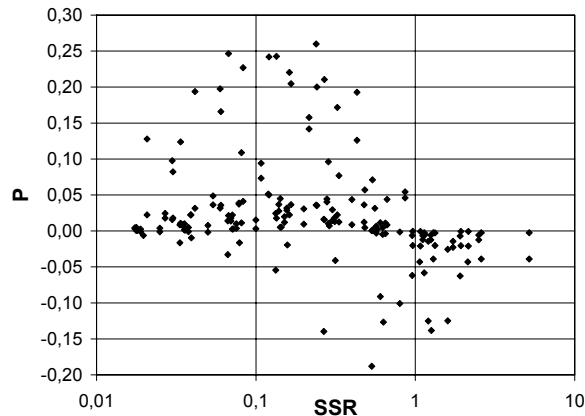


Figure 3. Scatterplot of $P = U_s - U_d$ versus SSR.

4.1. Analysis for space E_1

Experiments in space E_1 has a very low ratio between cache size and code size, that is, the cache is very small in front of code size. As shown in Figure 3, dynamic use presents better performance (lower utilization) for the major part of cases. Since cache is very small, the number of instructions reloaded after each preemption in dynamic use is low, so the value of cache-refill penalty is low. In the other hand, dynamic use presents lower WCET for all tasks than static use since every task can use all the cache space if necessary. Also, there is a soft-raising tendency in P as SSR grows. Increasing cache size helps to reducing the WCET of tasks, but the cache refill penalty remain low since cache size is still low, so dynamic use profits better from a slight grown in cache size. Table 1 shows a summary of statistics for space E_1 .

Table 1. Summary statistics for space E_1 .

Count	62
Average	0,0346298
Confidence interval (95%)	[0,0193494;0,0499103]
Median	0,0124745
Variance	0,00362049
Standard deviation	0,0601705
Minimum	-0,0328358
Maximum	0,246444
Lower quartile	0,00273985
Upper quartile	0,0358876
Interquartile range	0,0331477
# (%) of cases with $P \geq 0$	53 (85,5%)

4.2. Analysis for space E_2

Like the experiments in space E_1 , experiments in space E_2 shows better performance when locking cache is used dynamically. However, in this space, there are more cases with P values below 0 than in space E_1 , and negative values are very large. Also, a clear descending tendency in P is observed as SSR grows. In this space, cache size grows until SSR values are very close to 1. The value of cache-refill penalty for dynamic use is very high, since tasks must reload, before begin or resumes execution, a large number of instructions. This way, the reduction in the WCET due to a large cache is cancelled out by the cache-refill penalty.

Table 2 shows a summary of statistics for space E_2 . In spite of the existence of a descending tendency shown in Figure 3, experiments in space E_2 presents, in general, a greater gain of performance for dynamic use than the experiments in space E_1 . This is shown in the values of average, median, interquartile range and confidence interval, all of them with greater values for E_2 , and the percentage of cases where dynamic use presents a loss of performance, negligibly greater.

Table 2. Summary statistics for space E_2 .

Count	84
Average	0,0403668
Confidence interval (95%)	[0,0215822;0,0591515]
Median	0,0209883
Variance	0,00749262
Standard deviation	0,0865599
Minimum	-0,188171
Maximum	0,313682
Lower quartile	0,00467018
Upper quartile	0,0506257
Interquartile range	0,0459555
# (%) of cases with $P \geq 0$	68 (81,0%)

4.3. Analysis for space E₃

Experiments in space E₃ present a behavior very different than previous spaces. In all cases, P values are equal or lower than 0, that is, dynamic use of locking cache offers worse performance than static use in all cases. In this space, cache size is equal or greater than code size, so in static use there is no mandatory or conflict misses because locking cache is preloaded and locked before system begins execution. However, in dynamic use, cache contents are always reloaded before each task begins or resumes its execution. The existence of cache-refill penalty in dynamic use of locking cache provides a worse performance than in static use, where there is no cache-refill at all. Table 3 shows a summary of statistics for space E₃.

Table 3. Summary statistics for space E₃.

Count	36
Average	-0,0391734
Confidence interval (95%)	[-0,0588615;-0,0194853]
Median	-0,017381
Variance	0,00338587
Standard deviation	0,0581882
Minimum	-0,227715
Maximum	-0,0006855
Lower quartile	-0,0411056
Upper quartile	-0,0053433
Interquartile range	0,0357623
# (%) of cases with $P \geq 0$	0 (0,0%)

5. CONCLUSIONS

Developed experiments in this work clearly show that static use of locking cache is more predictable than dynamic use. But this lack of predictability does not represent any disadvantage for dynamic use because dynamic use offers better performance in the major part of the cases.

The schedulability analysis and the selection of the cache contents use the same tools and present the same complexity for both schemes, so the designer of real-time systems will choose between the two schemas only with regard to performance.

The comparison made in this paper shows that for systems with cache size lower than code size, dynamic use offers, in general, better performance –low utilization- in front of static use. When cache size is close to code size, still there is gain of performance using the dynamic schema, but the probability to loss performance rises as cache size is closer to code size. For systems with

cache size equal or greater than code size, static use presents better performance than the dynamic schema, this way, there is no advantage in using dynamically the locking cache.

Acknowledgments

This work was supported in part by Comisión Interministerial de Ciencia y Tecnología under project CICYT-TAP 990443-C05-02 and Generalitat Valenciana under project CTIDIA/2002/27.

References

- [1] J. V. Busquets, J. J. Serrano, R. Ors, P. Gil, A. Wellings. "Adding Instruction Cache Effect to Schedulability Analysis of Preemptive Real-Time Systems," *IEEE Real-Time Technology and Applications Symposium*, pp. 271-276, June 1996.
- [2] C. A. Healy, R. D. Arnold, F. Mueller, D. Whalley and M. G. Harmon. "Bounding Pipeline and Instruction Cache Performance," *IEEE Transaction on Computers*. Vol. 48, no 1, pp. 53-70, February 1999.
- [3] C. G. Lee, J. Hahn, Y. M. Seo, S. L. Min, R. Ha, S. Hong, C. Y. Park, M. Lee, C. S. Kim. "Enhanced Analysis of Cache-Related Preemption Delay in Fixed-Priority Preemptive Scheduling," *Proc. of the 18th IEEE Real-Time System Symposium*, pp. 187-198, December 1997.
- [4] S. S. Lim, Y. H. Bae, G. T. Jang, B. D. Rhee, S. L. Min, C. Y. Park, H. Shin, K. Park, and C. S. Kim. "An Accurate Worst Case Timing Analysis Technique for RISC Processors." *Proc. of the 15th IEEE Real-Time Systems Symposium*, December 1994.
- [5] D. B. Kirk. "SMART (Strategic Memory Allocation for Real-Time) Cache Design," *Proc. of the 10th IEEE Real-Time Systems Symposium*, pp. 229-237, December 1989.
- [6] Andrew Wolfe. "Software-Based Cache Partitioning for Real-Time Applications," *Proceedings of the Third International Workshop on Responsive Computer Systems*, September 1993.
- [7] A. Martí Campoy, A. Perles Ivars and J. V. Busquets Mataix. "Static Use Of Locking Caches In Multitask, Preemptive Real-Time Systems," *Proceedings of the IEEE Real-Time Embedded System Workshop*, December 2001
- [8] A. Martí Campoy, A. Perles Ivars and J. V. Busquets Mataix. "Dynamic Use Of Locking Caches In Multitask, Preemptive Real-Time Systems," *Proceedings of the 15th World Congress of the International Federation of Automatic Control*, July 2002
- [9] A. Martí Campoy, A. Perez Jimenez, A. Perles Ivars and J.V. Busquets Mataix. "Using Genetic Algorithms in Content Selection for Locking-Caches," *Proceedings of the IASTED International Symposia Applied Informatics*, pp. 271-276, February 2001
- [10] Patterson, D. and J. L. Hennessy. *Computer Organization and Design. The Hardware/ Software Interface*. San Mateo: Morgan Kaufmann, 1994.