27th IFAC/IFIP/IEEE Workshop on Real-Time
Porgramming. Lagow, Poland May 2003

PERFORMANCE COMPARISON OF LOCKING CACHES UNDER STATIC AND DYNAMIC SCHEDULERS

A. Martí Campoy, S. Sáez, A. Perles, J.V. Busquets-Mataix

Departamento de Informática de Sistemas y Computadores
E-46022, Universidad Politécnica de Valencia, SPAIN
{amarti,ssaez,aperles,vbusque}@disca.upv.es

Abstract: Static use of locking caches is a useful solution to take advantage of cache memories in real-time systems. Locking cache operates preloading and locking a set of instructions, thus cache contents are a-priori known and remain unchanged during system operation. This solution eliminates the unpredictable behavior of conventional caches, making easy to accomplish the schedulability test through simple and well-known tools. Once attained predictability, in this paper we analyze the performance of this schema compared to conventional cache, as function of system size and cache size. We also study the influence of the scheduler (either fixed or dynamic priority). *Copyright © 2003 IFAC*

Keywords: real-time systems, cache memories, scheduling algorithms, performance analysis, genetic algorithms.

1. INTRODUCTION

Microprocessors include cache memories in their memory hierarchy to increase system performance. General-purpose systems benefit directly from this architectural improvement, but hard real-time systems need additional hardware resources and/or system analysis to guarantee the time correctness of the system behavior when cache memories are present. In multitask, preemptive real-time systems, the use of cache memories presents two problems. The first problem is to calculate the Worst Case Execution Time (WCET) due to intra-task or intrinsic interference. Intra-task interference occurs when a task removes its own instructions from the cache due to conflict and capacity misses. This way, execution time of instructions is not constant. The second problem is to calculate the task response time due to inter-task or extrinsic interference. Inter-task interference occurs in preemptive multitasking systems when a task displaces the working set of any other task from the cache. When the preempted task resumes execution, a burst of cache misses increases its execution time. This effect, called cache-refill penalty or cache-related preemption delay must be considered in the schedulability analysis, since it situates task execution time over the precalculated WCET.

Several solutions have been proposed for the use of cache memories in real-time systems. In (Healy, *et al*., 1999; Lim, *et al*., 1994; Li, *et al*., 1996) cache behavior is analyzed to estimate task execution time considering the intra-task interference. In (Lee, *et al*., 1996; Busquets, *et al*., 1996) cache behavior is analyzed to estimate task response time considering the inter-task interference, using a precalculated cached WCET. Alternative architectures to conventional cache memories have been proposed, in order to eliminate or reduce cache unpredictability, making easy the sechedulability analysis. In (Kirk, 1989; Liedtke, *et al.*, 1997; Wolfe, 1993) hardware and software techniques are used to divide the cache memory into partitions, dedicating one or more partitions to each task, avoiding the inter-task interference.

The main drawback of these solutions is the complexity of the algorithms and necessary methods in order to accomplish the schedulability analysis or partitioning the cache. Also, each method considers only one face of the problem, intra-task interference or inter-task interference, but not together.

The use of locking caches has been proposed in (Martí, *et al.*, 2001b; Martí, *et al.*, 2003) as an alternative to conventional caches solving both intra-task and inter-task interference problem. The static use of locking caches fully eliminates the intra-task interference, allowing the use of simple algorithms in order to estimate the WCET of tasks. Regarding the inter-task interference, the static use of locking caches reduces the cache-refill penalty to a very low, constant time for all preemptions suffered by any task, so inter-task interference can be incorporated to schedulability analysis in a simple way.

This work presents a statistical analysis of worst-case performance offered by the static use of locking caches versus actual worst-case performance offered by conventional, dynamic and non-deterministic caches (hereinafter, we will abbreviate "worst-case performance" by "performance"). This analysis is presented for fixed-priority scheduler and Earliest Deadline First (EDF) scheduler. The behavior of the performance of locking cache under both schedulers is compared

2. OVERVIEW OF THE STATIC USE OF LOCKING CACHES.

The locking cache is a direct mapped cache with no replacement of contents when locked, joined with a temporal buffer of one-line size. The temporal buffer reduces access time to the memory blocks that are not loaded in the cache, since only references to the first instruction in the block produce cache miss. During system start-up, a small routine is executed to preload and lock the cache. Preloaded instructions can belong to any task of the system, and may be large consecutive instruction sequences or small, individual separate blocks. When the system begins its full-operational execution, the instruction cache is loaded with a well-known set of instructions, and its contents will never change, eliminating both intra-task and inter-task interference.

In locking caches, an instruction will always or never be in the cache. In this way and memory related, the execution time of the instruction is always constant and a-priori known. Thus, the WCET of a task running on a locking cache can be estimated using the worst-path analysis (Shaw, 1989) applied to machine code, taking into account the state, locked or not, of each instruction.

For fixed-priority schedulers when locking cache is used, the schedulability analysis is accomplished using CRTA (Busquets, *et al.*, 1996). Equation 1 shows the expression of CRTA, where cache-refill penalty is added for each preemption a task suffers. For EDF scheduler when locking cache is used, schedulability analysis is accomplised using the Initial Critical Instant (ICI) test (Ripoll, *et al.*, 1996). In EDF is not easy to know the number of preemptions a task suffers, but it is known the maximum number of preemptions a task produces. A task produces a preemption in its arrival or never will produce it. So, cache-refill penalty can be added to preempting task instead to preempted task. Equations 2 and 3 show how to include cache-refill penalty in the ICI test.

In equations 1 to 3, C_i is the WCET of τ_i without preemptions but considering locking-cache effects, and γ is the value of cache-refill penalty. In static use of locking cache, inter-task interference doesn't exist except for a small extrinsic interference introduced by the temporal buffer. After a preemption, a task must reload, in the worst case, only this temporal buffer. This way, the value of γ is T_{miss}, for all preemptions, all tasks, where T_{miss} is the time to transfer a block from main memory to temporal buffer.

Instructions to be loaded and locked in cache are selected by a genetic algorithm executed during system design. The developed algorithm provides the set of main memory blocks, and the result of the schedulability test. Also, this algorithm provides system utilization. Further details can be found in (Martí, *et al.*, 2001a).

$$w_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n}{T_j} \right\rceil x(C_j + g\ ) \qquad (1)$$

$$G(t) = \sum_{i=1}^{n} (C_i + g) \left\lceil \frac{t}{P_i} \right\rceil \qquad (2)$$

$$H(t) = \sum_{i=1}^{n} (C_i + g) \left\lfloor \frac{t + P_i - D_i}{P_i} \right\rfloor \qquad (3)$$

3. EXPERIMENTS

Experiments developed are composed of a set of tasks and a cache with definite size. The tasks used in the experiments are synthetic created in order to stress the locking cache and exercise the genetic algorithm. The main parameters of the experiments

Table 1 Main characteristics of experiments

Item	Minimum	Maximum
Number of tasks	3	8
Size of task	2 KB	32 Kb
System size (sum of tasks' size)	8 Kb	60 Kb
Instructions executed by task	50,000	8,000,000
Instructions executed by system	200,000	10,000,000
Cache size	1Kb	64Kb
Cache line size	16 bytes	16 bytes
Execution time from cache or temporal buffer (T_{hit})	1 cycle	1 cycle
Time to transfer from main memory (T_{miss})	10 cycles	10 cycles

are described in Table 1. More than 360 experiments have been defined. Four (two for each scheduler) kinds of runs have been accomplished for each experiment:

- Executing the genetic algorithm for each experiment.
- Simulating the conventional caches, using direct-mapped, two-way, four-way and full associative cache with LRU replacement algorithm. The best map-function is selected as performance value for each experiment. Simulations are accomplished using the SPIM tool (Patterson and Hennessy,1994), a MIPS R2000 simulator.

4. PERFORMANCE ANALYSIS

Regarding performance, it is not easy to compare the performance of a real-time system running over different hardware. If the same tasks-set is schedulable in both architectures, there are many characteristics and metrics useful to compare performance. The approach proposed in this work to compare the performance obtained from both locking cache and conventional cache is the processor utilization. Utilization of conventional cache come from the simulation of the hyperperiod (U_{cs} for fixed-

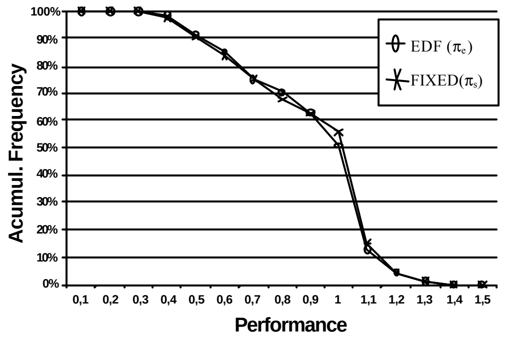

Fig. 1. Frequency histogram for π_s and π_e.

Table 2 Statistical summary for Performance

Statistic	Fixed sched.	EDF sched.
Average	0.90275396	0,900469293
Median	1.01112334	1,000110695
Minim	0.35296379	0,358470076
Maxim	1.35798524	1,357281619
Std. deviation	0.23985561	0,232068
Low quartile	0.715842316	0,71499322
High quartile	1.058288365	1,046493603
# of experiments	182	182
Exps with $\pi \geq 1$	101 (55.49%)	93 (51.1%)
Exps. with $\pi \geq 0.9$	140 (62.64%)	115 (63,19%)

priority scheduled experiments and U_{ce} for EDF scheduled experiments). This would be an upper bound of the performance obtained by any analysis tool. Utilization of locking cache is obtained by the execution of the genetic algorithm (U_{ls} for fixed-priority scheduled experiments and U_{le} for EDF scheduled experiments).

To compare the behaviors of the locking and conventional caches, *Performance* (π_s and π_e) is defined as the utilization of the conventional cache divided by the utilization of the locking cache (π_s= U_{cs}/U_{ls} and π_e= U_{ce}/U_{le}). π values greater than 1 indicate that locking caches provide a lower utilization than conventional caches, thus providing better performance.

Fig. 1 presents the accumulated frequency of π_s and π_e. Y-axis value is the percentage of experiments with a π value higher than x-axis value. Table 2 shows a statistical summary of π_s and π_e. Both figure and table show that in a great number of experiments, locking caches provide the same or better performance than conventional caches, reaching about 60% of experiments with no loss or very slight loss of performance. Besides, locking cache seems to behave the same way for both static and dynamic schedulers. Further analysis corroborates this hypothesis.

Intuitively, the most important factor in the behavior of π is the relationship between the cache size and the size of experiment code, since all tasks in the system compete for preloading their instructions in the cache. System Size Ratio (SSR) is defined as cache size divided by the sum of all system tasks size (equation 4).

$$SSR = \frac{Cache\_Size}{Code\_Size} \tag{4}$$

Fig. 2 shows the scatterplot of π_s versus SSR. Each point is the performance (π_s=U_{cs}/U_{ls}) of each experiment. The x-axis is shown in logarithmic scale. In the same way, Fig. 3 shows the scatterplot of π_e versus SSR. Again, the behavior of locking cache

Table 3 Upper limits and cache size for grouping

Group	Cache size	Upper limit
1	1 Kb	0,03125
2	2 Kb	0,0625
3	4 Kb	0,125
4	8 Kb	0,25
5	16 Kb	0,5
6	32Kb	1
7	64 Kb	∞

under both schedulers seems to be the same, and a very strong interaction may be noticed between performance of locking cache and the System Size Ratio.

In order to study this interaction, Fig. 4 and Fig. 5 presents the performance grouped by SSR values. Seven groups, representing the seven cache sizes used in experiments, have been defined, using the following rules. Table 3 shows the limits of each group:

- Upper limit of group n is defined as z/64, where $z=2^n$ and n = 1..6
- An experiment x_i, with ratio_size *r*, will belong to group *n* if: Upper limit of group *n-1* $\leq$ *r* < Upper limit of group *n*

Once again, behavior of locking cache is the same independently of the scheduler used. However, there are some slight differences between π_s and π_e. To study these differences with more detail, the data presented in Fig. 4 and Fig. 5 is divided into four spaces regarding the value of SSR. These four spaces, called from A to D are described below. Table 4 and Table 5 shows a statistical summary of spaces A to D for both schedulers.

Space A: This space comprises groups 1, 2, and part of 3 (SSR below 0.08). In this space, around 80% (84% for fixed-priority scheduler and 82% for EDF scheduler) of the experiments present a π value equal or greater than 1 with low variability. Also, π rises as the SSR increases. The behavior of this space is due to the large difference between cache size and code size. For small cache size, the intra-task interference is very high, and only a few number of instructions remain in the cache for two or more executions. However, the locking cache avoids replacement, thus the instructions locked in the cache always produce hit. For instructions not locked, their behavior is the same as in conventional small caches, replacing from temporal buffer after each execution. Space A shows a small increase of performance for the locking cache on the right side. While the conventional cache experiences the same intra-task interference when cache size increases in few bytes, locking caches profit from each byte added to cache size. In addition, the inter-task interference is high for the conventional cache, but not for the locking cache.

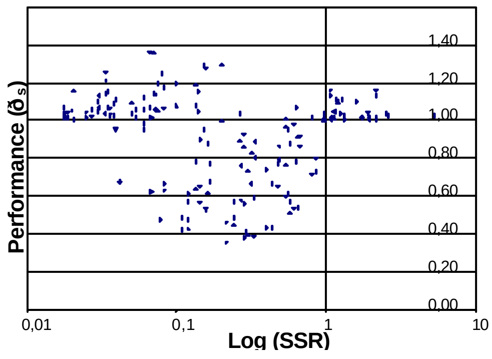

Fig. 2. Scatterplot of π_s versus SSR

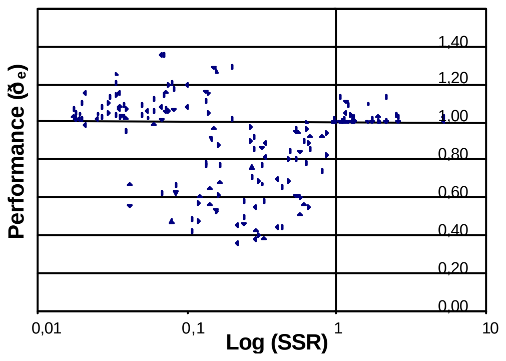

Fig. 3. Scatterplot of π_e versus SSR

Space B: This space comprises the part of group 3 not included in space A, group 4 and part of group 5 (SSR below 0.37). Only 21% of the experiments (for both schedulers) have a performance equal or greater than 1. The variability is very high, as the distance between upper and low quartile reflects. Performance of the locking cache falls down quickly as SSR rises. In this space, the dynamic behavior of the conventional cache profits from cache size, because many pieces of code with high degree of locality fit into cache. However, the locking cache must assign the cache lines to only a small set of instructions. The effect of inter-task interference is favorable to the locking cache, but its effect on performance of conventional cache is relatively low since cache size is still low.

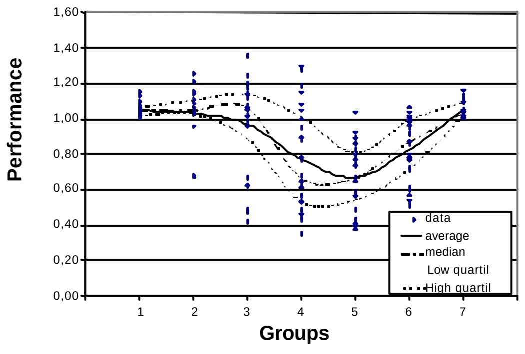

Fig. 4. π_s versus groups of SSR.

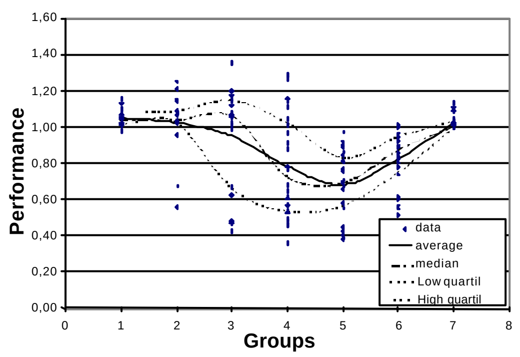

Fig. 5. π_e versus groups of SSR.

Space C: This space comprises part of group 5 and all experiments of group 6 (SSR lower than 1). In this space, only 17% of experiments presents a performance equal or greater than 1 for fixed-priority scheduler, and this percentage lowers to 6% for the EDF scheduler. The behavior of the locking cache in this space is the inverse than in space B. As SSR rises, the performance of the locking cache improves, because only instructions with very low degree of locality are forced to remain in the main memory. When the value of SSR is close to 1, the value of π is close to 1. Regarding inter-task interference, since cache size is now large, the impact of cache-refill penalty is very high and preemptions penalize performance in conventional caches.

Space D: This space comprises experiments fitting in group 7, all of them with SSR equal or greater than 1. In all cases, all experiments have performance equal or greater than 1, because all instructions are preloaded and locked in the cache. Neither intra-task nor inter-task interference exists, both in conventional and locking caches. But the locking cache may present a slight improvement in performance because when the system begins full operation, the cache is fully loaded, and no mandatory misses happen.

Conclusions from the four-spaces analysis are clear. The relationship between locking cache size and system size (as the sum of all tasks' sizes) allows the

Table 4 Statistical summary for spaces A to D of π_s

Space	A	B	C	D
Total experiments	56	56	34	36
Exps. with $\pi \geq 1$	83.9%	21.9%	17.6%	100%
Exps. with $\pi \geq 0.9$	91.1%	26.8%	35.3%	100%
Average	1.03	0.77	0.79	1.05
Median	1.05	0.67	0.79	1.03
Minimum	0.47	0.35	0.43	1.00
Maximum	1.36	1.29	1.07	1.16
Low quartile	1.01	0.53	0.65	1.02
High quartile	1.10	0.93	0.96	1.09
Std. deviation	0.17	0.27	0.19	0.05

Table 5 Statistical summary for spaces A to D of π_e

Space	A	B	C	D
Total experiments	56	56	34	36
Exps. with $\pi \geq 1$	82,5%	21,4%	6,1%	100%
Exps. with $\pi \geq 0.9$	89,5%	30,4%	33,3%	100%
Average	1,02	0,76	0,79	1,03
Median	1,05	0,70	0,83	1,00
Minimum	0,47	0,36	0,45	1,00
Maximum	1,36	1,29	1,02	1,13
Low quartile	1,02	0,55	0,66	1,00
High quartile	1,10	0,93	0,94	1,04
Std. deviation	0,19	0,26	0,17	0,04

designer of real-time systems to estimate the performance provided by static use of locking cache in front of conventional caches.

However, as opposed to the previously shown results, the analysis of the locking cache performance by means of four spaces show that there are differences between some characteristics of π_s and π_e. Apparently, locking caches provides worse performance under EDF scheduler than under fixed-priority scheduler, since the number of experiments with values of π above 1 is greater for fixed-priority scheduler, and the average value of π_s is greater than average value of π_e.

A paired-sample analysis (Jain, 1991) for each space provides information about the differences between average values of π_s and π_e. This paired-sample analysis is possible because each set of tasks has been evaluated under both schedulers. Table 6 shows the analysis summary. *Avg. of Diff.* is the average of differences of each pair of experiments, that is, π_s value of experiment *i* minus π_e value of the same experiment *i*. The result is the response given by the Null Hypothesis Test, using a t-test with alpha = 0.01 (99%) and hypothesis = 0.00. Reject means that the average significantly differs from 0. P-Value is a measure of significance. Value equal or greater than 0,01 indicates that the Null hypothesis (that is, average = 0,00) cannot be rejected at the 99% of confidence level.

The result of the test is the same for the four spaces: there is no significant difference between average values of π_s and π_e. That is, in a general sense, performance of locking cache is the same for both fixed-priority and EDF schedulers.

Table 6 Paired-sample analysis for π_s vs. π_e

Space	Avg. of Diff	P-Value	Result
A	0,00178593	0,518664	No Reject
B	-0,00471085	0,249892	No Reject
C	-0,00294133	0,737337	No Reject
D	0,00246365	0,0251184	No Reject

5. CONCLUSIONS

This work presents an analysis of performance of the static use of locking cache, under fixed-priority scheduler and Earliest Deadline First scheduler, with regard to two system main characteristics. Also, comparison of performance behavior under both schedulers has been presented. Performance has been defined as the relationship between the system utilization when conventional cache is used, versus system utilization when static use of locking cache is used.

The behavior of locking-cache performance versus System Size Ratio has been divided into four scenarios. A strong relationship between locking cache performance and this ratio has been noticed and identified, as well as the variability in the values of performance. Therefore, the real-time designer may estimate, without any experiment, the cost of using a locking cache, that is, the probability to lose performance and get a worse utilization than using conventional caches. For extreme values of the relationship between cache size and system size, the locking cache offers in most cases better performance than the conventional cache. For central values of this relationship, the locking cache offers worse performance than the conventional, unpredictable cache.

The paired-sample analysis shows that there is no significant difference in average values of performance when using static or dynamic scheduler. Results from this test, joined with the several figures and statistical summaries show that the behavior of locking cache, from the performance point of view, is the same independently of the algorithm used to schedule the tasks. This way, the kind of scheduler is not a major parameter in deciding the use or not of locking cache.

ACKNOWLEDGEMENTS

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

Busquets, J. V., J. J. Serrano, R. Ors, P. Gil, A. Wellings (1996). Adding Instruction Cache Effect to Schedulability Analysis of Preemptive Real-Time Systems. *IEEE Real-Time Technology and Applications Symposium*

Healy, C. A., R. D. Arnold, F. Mueller, D. Whalley and M. G. Harmon (1999). Bounding Pipeline and Instruction Cache Performance. *IEEE Transaction on Computers*. **Volume 48**, pages 53-70

Jain, Raj. (1991) *The Art of Computer Systems Performance Analysis*. Jhon Wiley & sons Ed. New York

Kirk, D. B. (1989) SMART (Strategic Memory Allocation for Real-Time) Cache Design. *Proc. of the 10th IEEE Real-Time Systems Symposium.*

Lee, C. G., J. Hahn, Y. M. Seo, S. L. Min, R. Ha, S. Hong, C. Y. Park, M. Lee, C. S. Kim (1997). Enhanced Analysis of Cache-Related Preemption Delay in Fixed-Priority Preemptive Scheduling. *Proc. of the 18th IEEE Real-Time System Symposium.*

Li, Y. S., S. Malik, and A. Wolfe (1996). Cache Modeling for Real-Time Software: Beyond Direct Mapped Instruction Caches. *Proc. of the 17th IEEE Real-Time Systems Symposium.*

Liedtke, J., H. Härtig, M. Hohmuth (1997). OS-Controlled Cache Predictability for Real-time Systems. *Proc of the IEEE Real-Time Technology and Applications Symposium.*

Lim, S. S., Y. H. Bae, G. T. Jang, B. D. Rhee, S. L. Min, C. Y. Park, H. Shin, K. Park, and C. S. Kim (1994). An Accurate Worst Case Timing Analysis Technique for RISC Processors. *Proc. of the 15th IEEE Real-Time Systems Symposium.*

Martí Campoy, A., A. Pérez, A. Perles, J.V. Busquets (2001a). Using Genetics Algorithms in Content Selection for Locking- Caches. *IAESTED International Conference on Applied Informatics.*

Martí Campoy, A., A. Perles, J.V. Busquets (2001b) Static Use of Locking Caches in Multitask Preemptive Real-Time Systems. *IEEE Real-Time Embedded Systems Workshop*. London, UK

Martí Campoy, A. ,S. Sáez, A. Perles and J.V.Busquets (2003). Schedulability Analysis in the EDF Scheduler with Cache Memories. *The 9th International Conference on Real-Time and Embedded Computing Systems and Applications.* Tainan, TW

Patterson, D. and J. L. Hennessy (1994). *Computer Organization and Design. The Hardware/ Software Interface*. Morgan Kaufmann. San Mateo.

Ripoll, I., A. Crespo and A. Mok (1996). Improvements in feasibility testing for real-time tasks. *The Journal of Real-Time Systems*, Vol. **11** pp. 19-39.

Shaw, A. (1989).Reasoning About Time in Higher-Level Language Software. IEEE Transaction on Software Engineering. **Vol. 15**, Num. 7.

Wolfe, A. (1993). Software-Based Cache Partitioning for real-time Applications. Proc of the 3th International Workshop on Responsive Computer Systems.