Proceedings of the 3rd WSEAS Int. Conference on Automation and Information. Tenerife, Spain December 2002.

Performance analysis of the static use of locking caches.

A. MARTÍ CAMPOY, A. PERLES, S. SÁEZ, J.V. BUSQUETS-MATAIX
Departamento de Informática de Sistemas y Computadores
Universidad Politécnica de Valencia
E-46022 Valencia
SPAIN
amarti@disca.upv.es http://www.upv.es

Abstract: The unpredictable behavior of conventional caches presents several problems when used in real-time multitask systems. It is difficult to know its effects in the Worst Case Execution Time and it introduces additional delays when different tasks compete for cache contents in multitask systems. This complexity in the analysis may be reduced using alternative architectures to cache memories, that improves predictability but obtaining similar performance. This is the case of locking caches, that may preload and lock cache contents, precluding the replacement during system operation, thus making cache and system behavior more predictable by means of simple, well-known and easy-to-use algorithms.
This work presents an analysis of worst-case performance obtained with the static use of locking caches versus worst-case performance obtained with conventional (non-locking) caches. Analysis results show that predictability can be reached with no loss of performance, and the scenarios where the locking cache provides similar performance to conventional caches may be estimated from system parameters like task size, cache size, level of locality and level of interference, without running any experiment.

Key-words: real-time systems, cache memories, locking cache, schedulability analysis, performance.

1 Introduction

Modern 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's 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. When removed instructions are executed again, a cache miss increases the execution time of the task. This way, the delay caused by the cache memory interference must be included in the WCET calculation. The second problem is to calculate the task response time due to inter-task or extrinsic interference. Inter-task interference occurs in preemptive multitask 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 [1,2,3] cache behavior is analyzed to estimate task execution time considering the intra-task interference. In [4,5] cache behavior is analyzed to estimate task response time considering the inter-task interference, using a precalculated cached WCET. The main drawback of these solutions is the complexity of the algorithm and methods needed to accomplish the analysis. Also, each method considers only one face of the problem, the intra-task interference or the inter-task interference, but not both. Alternative architectures to conventional cache memories have been proposed, in order to eliminate or reduce cache unpredictability, making the sechedulability analysis easy. In [6,7,8] hardware and software techniques are used to divide the cache memory, dedicating one or more partitions to each task, avoiding the inter-task interference. The main drawback of these proposals is the no action over the intra-task interference, leaving one side of the problem unresolved. Also, in several cases the inter-task interference is only reduced but not fully eliminated, so the inter-task problem is also unresolved.

The use of locking caches has been proposed in [9,10] as an alternative to conventional caches solving both intra-task and inter-task interference analysis. 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-related-preemption delay to a very low and constant time for all preemptions suffered by any task, allowing the use of the well-known RTA analysis with minor changes.

This work presents a statistical analysis of worst-case performance offered by the static use of locking caches versus worst-case performance offered by conventional, dynamic and non-deterministic caches (hereinafter, we will abbreviate "worst-case performance" by "performance").

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. A fully locked cache allows obtaining the maximum possible performance, while making the cache deterministic. 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, 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 [11] applied to machine code, taking into account the state, locked or not, of each instruction. The worst path analysis is accomplished using an extension of task's Control Flow Graph, called Cached-Control Flow Graph (c-cfg) modeling not only the task's paths but also how the cache is used.

Schedulability analysis is accomplished using CRTA [5]. Equation (1) shows the expression of CRTA, where C_i is the WCET of τ_i without preemptions but considering locking-cache effects, and γ_j is the increase in the response time that task τ_i experiences due to task τ_j. But in locking caches, inter-task interference doesn't exist since the cache contents remain unchanged during task switch, except for a small extrinsic interference introduced by the temporal buffer. This way, the value of γ_j 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 [12] executed during system design. Developed algorithm provides the set of main memory blocks, and both estimation of the WCET of each task and its Response Time when system is executed in a locked cache with the selected set of blocks loaded and locked. Further details can be found in [9].

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

3 Experiments

Experiments must provide two main results. First, evaluate the accuracy in the estimation of response time and second, show if predictability is reached with or without loss of performance. More than 180 experiments, with a total of about 700 tasks have been defined.

Each experiment is composed of a set of tasks and a cache size. The tasks used in the experiments are artificially created in order to stress the locking cache and the genetic algorithm. The main parameters of the experiments are described in Table 1. For each system, two qualitative parameters are also defined:

- Interference: Relationship between periods, defined at two levels: high level of interference and null level of interference.
- Locality: Level of spatial and temporal locality. The Level of locality may be divided into four ranges: null, poor, good, and excellent. The tasks used in the experiments vary between poor and good.

	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 form main memory (T_{miss})	10 cycles	10 cycles

Table 1 Main characteristics of the experiments

Three kinds of runs have been accomplished for each experiment:

- Execution of genetic algorithms for each experiment.
- Simulation of 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.
- Simulation of direct-mapped locking cache, loading and locking selected blocks by the genetic algorithm.

Simulations for both locking and conventional caches are accomplished using the SPIM tool [13], a MIPS R2000 simulator.

Fig. 1 shows the accumulated frequency of overestimation in the response time of each task estimated by the genetic algorithm, with respect to the actual (simulated) response time when a locking cache is used. Y-axis value is the percentage of tasks with an overestimation lower than x-axis value. This figure confirm the accuracy of estimating the response time, since more than 90% of tasks have an overestimation below 0.05%, and the simulated response time using a locking cache never exceeds the estimated one.

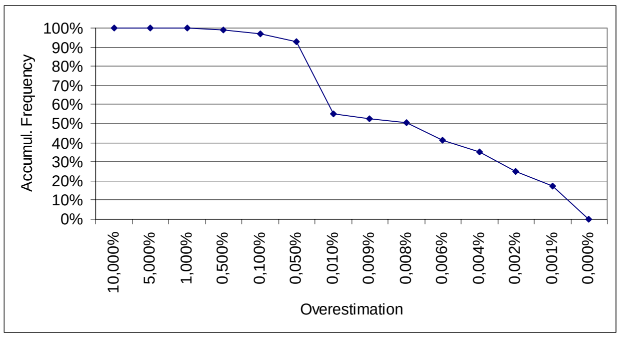

Fig. 1 Frequency histogram for overestimation when using a locking cache.

4 Performance analysis

Regarding performance, it is not easy to compare the performance of a real-time system running over two different hardware. If the same set of tasks is schedulable in both architectures, there are a lot of 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 schedulable utilization.

The schedulable utilization is the processor's utilization the system makes, assuming that every execution of every task in the system takes the maximum and worst possible response time. This is an upper bound of the actual utilization. The response time of tasks vary for each execution, but in the worst scenario the response time of each task is constant and therefore the maximum one, so the system's utilization is always lower or equal to schedulable utilization. Since the real-time designer must only use estimated and safe values, the utilization must be calculated with the same criterion.

The utilization is traditionally calculated as the sum of C_i/P_i, where C_i is the WCET of task τ_i considering cache, and P_i is the period of task τ_i. But the schedulable utilization used in this work is calculated from response time because the inter-task interference must be accounted. In a cached system, tasks spend processor time while executing its instructions, but also while reloading cache contents after preemption. WCET considers task executing alone in the system, so cache-related preemption delay is not incorporated in this time. Response time, calculated using CRTA and taking into account the cache includes the cache-related preemption delay for each preemption, so both inter-task and intra-task interference is considered.

But response time of a task τ_i includes not only execution time and cache reload time, but also execution time and cache reload times of all higher priority tasks. Therefore, the execution time including cache-related preemption delay is extracted from response time using RTA expression (equation 2), where R_i, T_j and C_j (j has a higher priority than i) are known. Note that R_i includes cache-related preemption delay, since R_i comes from the CRTA equation used in genetic algorithm.

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil x C_j \tag{2}$$

changing the position of terms:

$$C'_1 = R_1$$
$$C'_i = R_i - \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil x C_j \tag{3}$$

that is, the execution time of task τ_i is its response time minus the time other higher-priority tasks have been executing during task τ_i response time. Since R_i includes all cache effects (both intra and inter task interference), C'_i include them as well.

Estimated schedulable utilization of the locking cache (U_l) is the sum of the utilization of all tasks in the system, as shown in equation (4)

$$U_l = \sum \frac{C'_i}{T_i} \tag{4}$$

where C'_i is the execution time of task τ_i calculated in equation 4 and T_i is the period of task τ_i.

Actual (simulated) schedulable utilization of conventional caches (U_c) is calculated in the same way from equations (2) to (4), but using the worst response time from the hyperperiod simulation instead the estimated one.

To compare the behaviors of the locking and conventional caches, Performance (π) is defined as the schedulable utilization of the conventional cache divided by the schedulable utilization of the locking cache ($\pi = U_c/U_l$). π values greater than 1 indicate that locking caches provide a lower utilization than conventional caches, providing better performance.

Fig. 2 presents the accumulated frequency of π. Y-axis value is the percentage of tasks with a π higher than x-axis value. Table 2 shows a statistical summary of π. These figures 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. However, the variability is very high, since π value range from 0.3 to 1.4.

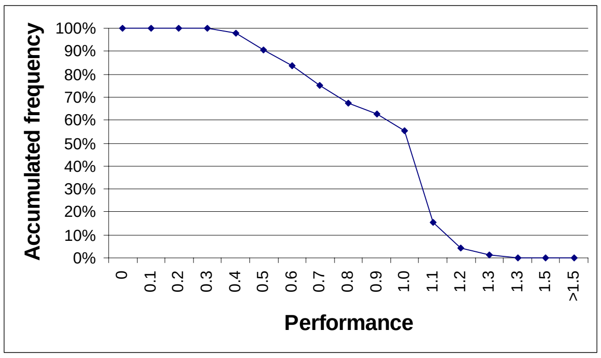

Fig. 2 Frequency histogram for Performance (π).

Average	0.90275396
Median	1.01112334
Minim	0.35296379
Maxim	1.35798524
Standard deviation	0.23985561
Low quartile	0.715842316
High quartile	1.058288365
Std. Skewness	-3.69349
Std Kurtosis	-1.6793
# of experiments	182
Experiments with π>= 1	101 (55.49%)
Experiments with π>= 0.9	140 (62.64%)

Table 2 Statistical summary for Performance

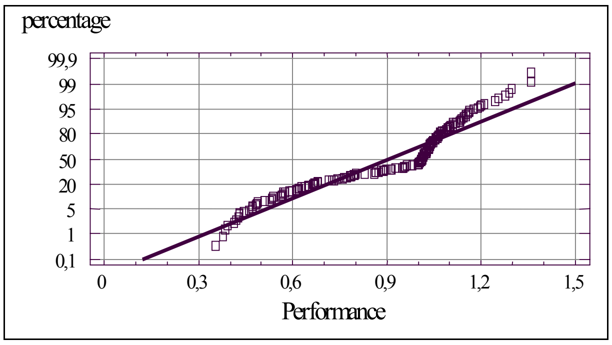

Fig. 3 Normal probability plot for Performance

Standardized skewness and standardized kurtosis can be used to determine whether the data comes from a normal distribution. Values of these statistical data outside the range of -2 to +2 indicate significant departures from normality, which would tend to invalidate any statistical test regarding the standard deviation. In this case, the standardized skewness value is not within the range expected for data from a normal distribution. Fig. 3 shows the normal probability plot for π. However the points are very close to the reference line, indicating a quasi-normal distribution, in this graph can be noticed the existence of two populations (subsequent analysis must take account of this question)

Intuitively, the most important factor in the behavior of π is the relationship between cache size and experiment size, since all tasks in the system compete for preloading their instructions in the cache. Ratio_size is defined as cache size divided by the sum of all system's tasks sizes:

(Ratio_size = Cache_Size/System_Size).

Fig. 4 shows the scatterplot of π versus ratio_size. Each point is the performance ($\pi = U_c/U_l$) of each experiment. The x-axis is shown in logarithmic scale. This figure can be divided into three spaces. The first space, on the left side of the graph and comprising values of ratio_size lower than 0.1, where most of the experiments present a π value higher than 1.0. The second space, at the center of the graph and comprising ratio_size values between 0.1 and 1, where the major part of experiments have a π value below 1, with great variability. Finally, a third space, comprising ratio_size values above 1, where all experiments have a π value higher than 1. This shows that there is a strong relationship between ratio_size and performance of the locking cache. In order to study this interaction, Fig. 5 presents the performance grouped by ratio_size 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 $z=2^{(n-1)}$ and n = 1..7
- An experiment x_i, with ratio_size r, will belong to group n if: Upper limit of group n-1< r < Upper limit of group n

Group	Cache size	Upper limit	Group	Cache Size	Upper limit
1	1 Kb	0,0313	5	16 Kb	0,4900
2	2 Kb	0,0525	6	32Kb	1,0000
3	4 Kb	0,1150	7	64 Kb	∞
4	8 Kb	0,2400			

Table 3 Upper limits and cache size for grouping

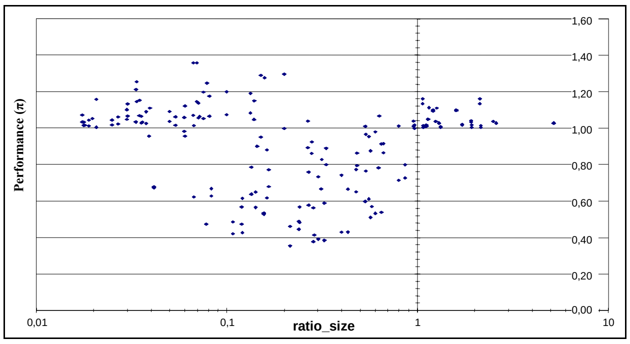

Fig. 4 Scatterplot of π (U_c/U_l) versus ratio_size

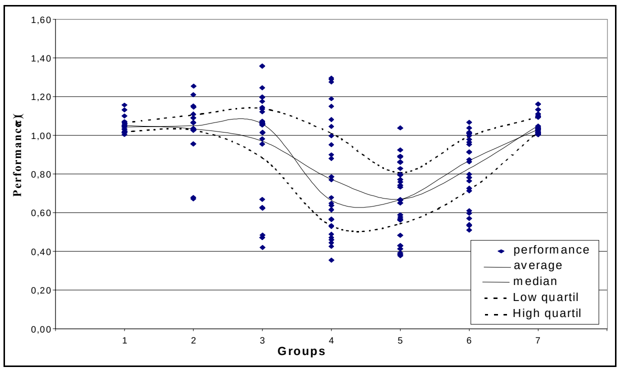

Fig. 5 π (U_c/U_l) versus groups of ratio_size.

In Fig. 5 performance behavior can be divided into four spaces regarding the value of ratio_size. These four spaces, called from A to D are described below. Table 4 shows a statistical summary of spaces A to D.

• Space A: This space comprises groups 1, 2, and part of 3 (ratio_size below 0.08). In this space, nearly 84% of the experiments present a π value equal or greater than 1 with low variability. Also π rises as the ratio_size 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, so 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 for each byte added to cache size. In addition, the inter-task interference is very high for the conventional cache, but not for the locking cache.

• Space B: This space comprises the part of group 3 not included in space A, group 4 and part of group 5 (ratio_size below 0.37). Only 20% of the experiments 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 ratio_size 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 from the temporal buffer. The effect of inter-task interference is favorable to the locking cache, but its effect on performance is low since cache size is still low.

• Space C: This space comprises part of group 5 and all experiments of group 6 (ratio_size lower than 1). In this space, only 17% of the experiments present a performance equal or greater than 1. The behavior of the locking cache in this space is the inverse than in space B. As ratio_size rises, the performance of the locking cache improves, because only instructions with very low degree of locality are forced to rest in the main memory. When the value of ratio_size is close to 1, the value of π is close to 1. Regarding inter-task interference, since cache size is now great, the impact of cache-related preemption is very high and preemptions penalize performance in conventional caches.

• Space D: This space comprises experiments fitting group 7, all of them with ratio_size equal or greater than 1. In all cases, all experiments have performance greater than 1, because all instructions are preloaded and locked in the cache. No intra- nor inter-task interference exist, both in conventional and locking caches. But the locking cache presents a slight improvement in performance because when the system begins full operation, the cache is fully loaded, and no mandatory misses happen.

Conclusions from Table 4 and Fig. 5 are clear. The relationship between locking cache size and system size (as the sum of all tasks' sizes) allows the designer of real-time systems to estimate the performance provided by static use of locking cache in front of conventional caches.

Space	A	B	C	D
Total experiments	56	56	34	36
Exps. with $\pi \geq 1$	83.93%	21.43%	17.65%	100%
Exps. with $\pi \geq 0.9$	91.07%	26.79%	35.29%	100%
Average	1.03	0.75	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
Lower quartile	1.01	0.53	0.65	1.02
Upper quartile	1.10	0.93	0.96	1.09
Std. deviation	0.17	0.27	0.19	0.05

Table 4 Statistical summary for spaces A to D

The following analysis provides more information about performance behavior of locking caches, concerning software parameters like Locality and Interference degree. The analyses are based on the Analysis of Variance (ANOVA), and the normality of data used in each analysis has been checked, in order to guarantee the veracity of results [14].

The first analysis is a paired-sample analysis in order to identify the effect of the Interference factor. Each set of tasks generates two experiments, one with low degree of interference and other with high degree of interference. Also, experiments with high and low Locality are analyzed separately, allowing the identification of interaction between both factors.

The analysis is performed for each space described previously. Table 5 shows the analysis summary. Mean is the mean of difference of each pair of experiments, that is, π value of experiments with low degree of Interference minus π value of experiments with high degree of interference. The result is the response given by the Null Hypothesis Test, using a t-test with alpha = 0.05 (95%) and hypothesis = 0.00. Reject means that the mean significatilly differs from 0.

Space	Locality	Mean of Diff	P-Value	Result
A	Low	-0.110092	0.001	Reject
	High	0.0269674	0.541124	No reject
B	Low	-0.211077	0.000006	Reject
	High	-0.0024647	0.946495	No Reject
C	Low	-0.292277	0.0018468	Reject
	High	-0.0814585	0.289005	No Reject
D	Low	-0.0072157	0.0898641	No Reject
	High	-0.0052957	0.0505166	No Reject

Table 5 Paired-sample analysis for π, concerning Interference and Locality factors

For spaces A, B and C, the Interference factor has a statistical effect when the degree of locality is low. The mean value of the differences is negative, so experiments with high Interference values have better performance than experiments with low Interference values. This effect is because the locking cache does not perform any cache reload (but the temporal buffer) after preemptions, while conventional caches experience a significant cache-related preemption delay, so performance of conventional cache is worse when a large number of preemptions occur, that is, when the interference between tasks is high. For systems with a high degree of locality, the conventional cache is used more efficiently than the locking cache, improving performance for the first type of cache. This profit from locality is cancelled by cache-related preemption delay, so there is no performance difference when interference is high or low.

Regarding space D, all instructions fit in the cache, and there is no replacement of cache contents after preemptions, so the effect of the interference factor is null. So there is no statistically significant difference between conventional and locking caches.

The locality factor can not be analyzed using the paired-samples, since it is not possible to generate two experiments with the same characteristics but with different degree of locality. Locality is intrinsic to tasks' structure and changing the degree of locality also changes other task parameters. Therefore, the locality factor is analyzed using the null hypothesis test to the difference of means instead of using the mean value of difference. Using difference of means provides less accurate results than using the mean value of differences.

Table 6 shows the summary for the hypothesis test applied to mean_L-$\text{mean}_H = 0$, using a t-test with alpha = 0.05 (95%), where mean_L is the mean of π obtained in experiments with low locality, and mean_H is the mean of π obtained in experiments with high locality. The result is the response given by the test, indicating the difference is not 0 (reject hypothesis) or the difference is 0 (do not reject hypothesis).

Space	Mean_L	Mean_H	P-value	Result
A	1.09603	1.00928	0.0276242	Reject
B	0.877057	0.591776	0.00003252	Reject
C	0.796151	0.783419	0.853871	No Reject
D	1.06987	1.01978	0.00076907	Reject

Table 6 Summary for effect of Locality factor analysis

For spaces A and B, the hypothesis is rejected, that is, the means for low and high degree of locality are statistically different. Since the value of mean_L-mean_H is positive, mean_L is greater than mean_H, so experiments with high degree of locality give worse performance from the locking cache point of view. This effect result from the profit obtained from conventional caches when the degree of locality is high, since the level of intra-task interference is minor in this scenario. Also, for lower ratio values, the locking cache preloads and locks a small part of all system code, leaving in competition for the temporal buffer pieces of the code with high locality, while in the conventional cache these pieces of the code are loaded when needed. The dynamic and adaptive behavior of the conventional cache gives better performance than the static behavior of the locking cache.

For space C, the difference between means is zero. Locality of tasks does not modify performance. In this space, the locking cache is large enough to preload and lock all pieces of the code with high locality, giving a behavior similar to the conventional cache.

Regarding space D, cache size is large enough to load all system instructions, so locality does not affect performance, since there is not intra or inter task interference. However, the test indicates that there is a statistical effect of locality, giving better performance (from the locking cache point of view) when locality is low. This result is due to mandatory misses. The locking cache gives better performance than the conventional cache because the locking cache is preloaded during system start-up, while the conventional cache is loaded during system operation, generating mandatory misses, one for each main memory block. If the degree of code locality is high, the structure of tasks is plenty of loops, or loops have a large number of iterations, or both cases. This way, in experiments with high locality, mandatory misses are weakened by the intensive execution of instructions. The locking cache, however, does not present mandatory misses, so locality does not modify its behavior.

5 Conclusions

The analysis accomplished in this work shows that the locking cache provides predictability in estimating the response time of tasks, providing very accurate results, and there is no loss of performance in more than 55% of the cases.

The analysis of performance has been accomplished using schedulable utilization, calculating it from the response time of tasks, thus intra-task and inter-task interference are considered. The performance of the locking cache is obtained from estimated values, giving a conservative and safety value. The behavior of locking-cache performance versus conventional-cache performance has been divided into four scenarios regarding the ratio between size of the cache where system runs and size of the system. A strong relationship between locking-cache performance and this ratio has been noticed and identified, as well as the variability in the results of the performance values. Therefore, the real-time designer may estimate, without any experiment, the cost of using a locking, that is, the probability to lose performance and get a worse schedulable utilization than using conventional caches. For extreme values of relationship between cache and system sizes, the locking cache offers in most cases better performance than the conventional cache. For central values of relationship, the locking cache offers worse performance than the conventional cache. Also, the effect of software parameters (locality and interference) on the performance of the locking cache has been analyzed and identified, taking into account the scenario where the system runs.

References:

[1]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

[2]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*.

[3]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.*

[4]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*.

[5]Busquets, J. V., J. J. Serrano, R. Ors, P. Gil, A. Wellings (1996). Adding Instruction Cache Effect to an Exact Schedulability Analysis of Preemptive Real-Time Systems. *Proc. of the IEEE Euromicro Workshop on Real-Time Systems*

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

[7]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*.

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

[9]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.*

[10]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

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

[12]Goldberg, D. E. (1989). Genetic Algorithms in Search, Optimization and machine Learning Addison-Wesely Co.

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

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

Acknowledgements: *We would like to thank the Foreign Language Co-ordination Office at Universidad Politécnica de Valencia for their help in revising this paper.*