Conference Paper

Cache Persistence Aware Response Time Analysis for Fixed Priority Preemptive Systems

Syed Aftab Rashid
Geoffrey Nelissen
Eduardo Tovar

CISTER-TR-160207

Cache Persistence Aware Response Time Analysis for Fixed Priority Preemptive Systems

Syed Aftab Rashid, Geoffrey Nelissen, Eduardo Tovar

CISTER Research Center

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail:

http://www.cister.isep.ipp.pt

Abstract

A task can be preempted by several jobs of a higher priority task during its response time. Assuming the worst-case memory demand for each of these jobs leads to pessimistic worstcase response time (WCRT) estimations. Indeed, there is a high chance that a big portion of the instructions and data associated with the preempting task τj, are still available in the cache when τj releases its next jobs. We call this content “persistent cache blocks” (PCBs). Accounting for PCBs in the memory demand of the preempting task allows to significantly reduce the pessimism on the total memory demand considered by the WCRT analysis. In this work, we propose a refined WCRT analysis for fixed priority preemptive systems considering (i) the effect of PCBs on the memory demand of the preempting task, and (ii) accounting for the number of PCBs that can be evicted by the preempted tasks between two successive job releases of the preempting tasks.

Cache Persistence Aware Response Time Analysis for Fixed Priority Preemptive Systems

Syed Aftab Rashid, Geoffrey Nelissen, Eduardo Tovar
CISTER/INESC TEC, ISEP
Polytechnic Institute of Porto, Portugal

***Abstract*—A task can be preempted by several jobs of a higher priority task during its response time. Assuming the worst-case memory demand for each of these jobs leads to pessimistic worst-case response time (WCRT) estimations. Indeed, there is a high chance that a big portion of the instructions and data associated with the preempting task τ_j, are still available in the cache when τ_j releases its next jobs. We call this content "persistent cache blocks" (PCBs). Accounting for PCBs in the memory demand of the preempting task allows to significantly reduce the pessimism on the total memory demand considered by the WCRT analysis. In this work, we propose a refined WCRT analysis for fixed priority preemptive systems considering (i) the effect of PCBs on the memory demand of the preempting task, and (ii) accounting for the number of PCBs that can be evicted by the preempted tasks between two successive job releases of the preempting tasks.**

I. Introduction

The existing gap between the processor and main memory operating speeds necessitates the use of intermediate cache memories to accelerate the average case access time to instructions and data that must be executed or treated on the processor. The introduction of cache memories in modern computing platforms is the cause of big variations in the execution time of each instruction depending on whether the instruction and the data it treats are already loaded in the cache (cache hit) or not (cache miss).

These last years, a lot of attention has been placed on the analysis of the impact of preemptions on the worst-case execution time (WCET) and worst-case response time (WCRT) of tasks in systems where preemptions are allowed. Indeed, the preempted tasks may suffer additional cache misses if memory blocks are evicted from the cache during the execution of the preempting tasks. These evictions cause extra accesses to the main memory, which result in additional delays in the task execution. This extra-cost is usually referred to as cache-related preemption delays (CRPDs).

Over the years, different approaches have been proposed to counter the effect of preemptions. Some (e.g., [1], [2]) use non- or limited preemption scheduling schemes to eliminate or reduce the number of preemptions. Others [3]–[9] use information about the task memory access pattern to bound and incorporate the preemption costs into the WCET and the WCRT analyses. In this work, we focus on the latter and propose a method, complementary to [3]–[9], to bound the memory overhead during the task response time.

Several approaches have been proposed in the literature to compute accurate bounds on CRPDs. They are based on the study of memory access patterns of the preempted task [4], the preempting tasks [3], [5], or both [5]–[9]. However, all these approaches still result in pessimistic WCRT bounds due to the fact that they only consider the effect of preemptions on the memory demand of the preempted task but do not consider the variation in the memory demand of the preempting tasks. They all assume that every job of a high priority task τ_j preempting a low priority task τ_i will ask for its maximum memory demand, i.e., its worst-case memory demand in isolation. Although true for the first job released by the preempting task τ_j, subsequent jobs of τ_j may re-use most of the data and instructions that were already loaded in the cache during the execution of its previous jobs. Noticeably, the memory blocks loaded by a job $J_{j,k}$ of τ_j remain in the cache until the execution of the next job $J_{j,k+1}$ of τ_j unless evicted by any other task executing between the completion of $J_{j,k}$ and the beginning of the execution of $J_{j,k+1}$.

Therefore, in order to propose tighter bounds on the memory overhead, and hence on the WCRT of each task τ_i executing in a preemptive system, we (i) model the impact of persistent cache contents associated with each preempting tasks on the WCRT of the preempted task, and (ii) analyze the effect of the preempted task cache accesses on the memory demand of the preempting tasks.

II. System Model

This work targets single-core platforms with a single level (L1) data/instruction cache. The cache is assumed to be direct-mapped, that is, each memory block in the main memory can be mapped to only one block in the cache.

We consider sporadic tasks with constrained deadlines where each task has a fixed priority. Any priority assignment scheme (e.g., Rate Monotonic [10]) is acceptable. We also assume that the tasks are independent and do not suspend themselves during their execution. A task τ_i is defined by a triplet (C_i, T_i, D_i); where C_i is the WCET of τ_i, T_i is its minimum inter-arrival time and D_i is the relative deadline of each instance (or job) of τ_i. We assume that the tasks have constrained deadlines, i.e., $D_i \leq T_i$. In this work, we further decompose each task WCET into two terms, namely, the worst-case processing demand P_i and the worst-case memory demand MD_i. P_i denotes the worse case execution time of τ_i considering that every memory access is a cache hit. Consequently, it only accounts for execution requirements of the task and does not include the time needed to fetch data and instructions from the main memory. MD_i is the worst-case memory demand of any job of task τ_i, that is, it is the maximum amount of time during which any job of τ_i is performing memory operations. Because the worst-case processing demand and the worst-case memory demand may not necessarily be experienced on the same execution path of τ_i, it results that $C_i \leq P_i + MD_i$. The WCRT of task τ_i is denoted by R_i and is defined as the longest amount of time between the arrival and the completion of any of its jobs. A task τ_i is said to be schedulable if $R_i \leq D_i$. Similarly, a task set is schedulable if all of its tasks are schedulable.

In this work, we consider that preemption costs only refer to additional cache reloads due to those preemptions. Other overheads due to context switches, scheduler invocations and pipeline flushes are assumed to be included in the WCET.

For notational convenience, we define the following task sets:

- $hp(i)$: the set of tasks with a priority higher than that of τ_i.
- $hep(i)$: the set of tasks with priorities higher than or equal to that of τ_i.
- $aff(i,j)$: the set of tasks with priorities higher than or equal to the priority τ_i but strictly lower than that of τ_j. This set contains the intermediate priority tasks, which may affect the response time of τ_i but may also be preempted by τ_j.

III. State of the Art

As already explained in the introduction, when a task τ_i is preempted by a higher priority task τ_j, it is likely that τ_j will evict memory blocks of τ_i from the cache. On resumption, τ_i might consequently require to reload cache blocks from the main memory along with its normal memory requirements. This CRPD caused by τ_j on τ_i is denoted by $\gamma_{i,j}$. Several methods have been proposed in the literature to compute $\gamma_{i,j}$. In one of the earlier works, Lee et al. [4] introduced the concept of *useful cache block* (UCB). As defined in [9], *"a memory block m is called a useful cache block (UCB) at program point P, if it is cached at P and will be reused at program point Q that may be reached from P without eviction of m"*. Lee et al. [4] used the maximum number of UCBs among all the tasks in $aff(i,j)$ to upper bound the preemption cost $\gamma_{i,j}$. Busquets et al. [3] and Tomiyama et al. [5] rather used the notion of *evicting cache block* (ECB), i.e, any cache block accessed during the execution of the task and which can then evict the memory block cached by another task, to upper bound the preemption cost that can be caused by each preempting task. Other approaches by Tan and Mooney [7], Staschulat et al. [6] and Altmeyer et al. [8] used both the UCBs of the preempted tasks and ECBs of the preempting tasks in order to come up with more precise bounds on the preemption cost. Notably, the ECB and UCB-union and the multi-set approaches presented in [8] and [9] dominate all the existing approaches for CRPD calculation. We first detail the ECB-union approach and then the UCB-union multi-set. Readers are referred to [9] for the description of UCB-union and ECB-union multi-set approaches.

The ECB-union approach [8] uses the ECBs of all tasks in $hep(j)$ maximized over the UCBs of tasks in $aff(i,j)$ to calculate the preemption cost $\gamma_{i,j}$. The resulting value for the preemption cost, denoted as $\gamma_{i,j}^{ecb}$, is given by

$$\gamma_{i,j}^{ecb} = d_{mem} \times \max_{\forall k \in aff(i,j)} \left(\left| UCB_k \cap \Big(\bigcup_{\forall h \in hep(j)} ECB_h \Big) \right| \right) \tag{1}$$

where d_{mem} is the time required to reload one memory block from the main memory to the cache, and UCB_k and ECB_j are the sets of UCBs and ECBs of task τ_k and τ_j, respectively. The preemption cost can then be accounted for in the WCRT analysis using the following formulation:

$$R_i^{k+1} = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i^k}{T_j} \right\rceil \times (C_j + \gamma_{i,j}^{ecb}) \tag{2}$$

When combined, the ECB and UCB-union approaches provide a reasonably precise upper bound on the preemption cost. However, it can also lead to overestimations in different situations as shown in [9]. To further reduce the pessimism associated to the ECB and UCB-union approaches, Altmeyer et al. [9] proposed two new solutions, namely, the UCB-union multi-set and the ECB-union multi-set approaches. These multi-set versions of the UCB-union and ECB-union approaches additionally take into account the maximum number of jobs $E_j(R_i) \overset{\text{def}}{=} \left\lceil \frac{R_i}{T_j} \right\rceil$ that each higher priority task τ_j can release during the response time of τ_i. Under that framework, the WCRT equation becomes:

$$R_i^{k+1} = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i^k}{T_j} \right\rceil \times C_j + \sum_{\forall j \in hp(i)} \gamma_{i,j}^{mul} \tag{3}$$

where $\gamma_{i,j}^{mul}$ accounts for the total preemption cost that can be caused by all the jobs of τ_j released during the response time of τ_i. Using the UCB-union multi-set approach $\gamma_{i,j}^{mul}$ is upper bounded by $\gamma_{i,j}^{ucb-m}$ defined as follows:

$$\gamma_{i,j}^{ucb-m} = d_{mem} \times \left| M_{i,j}^{ucb} \cap M_{i,j}^{ecb} \right| \tag{4}$$

where $M_{i,j}^{ucb}$ and $M_{i,j}^{ecb}$ are multi-sets defined as

$$M_{i,j}^{ucb} = \bigcup_{\forall k \in aff(i,j)} \left(\bigcup_{E_j(R_k) E_k(R_i)} UCB_k \right) \tag{5}$$

and

$$M_{i,j}^{ecb} = \bigcup_{E_j(R_i)} ECB_j \tag{6}$$

Note that the ECB-union multi-set approach dominates the ECB-union approach [8] whereas the UCB-union multi-set approach dominates the UCB-union approach [7]. Yet, it is shown in [9] that the ECB-union and UCB-union multi-set approaches are incomparable.

For a more detailed description on the formulation of Equations (2) to (6), the reader is referred to [9].

IV. Motivational Example

As presented in the previous section, the impact of a high priority task τ_j on the WCRT of lower priority task τ_i can be estimated in a fairly accurate manner by analyzing the mapping of UCBs and ECBs in the cache. The impact of τ_i on the memory demand of τ_j is however ignored during the WCRT analysis of τ_i. Yet, high priority tasks may often execute more than one job during the response time of a lower priority task. Therefore, to accurately estimate the WCRT of a low priority task τ_i, one must consider the impact of the preempted tasks on the memory demand of each job released by the preempting tasks. In the literature, this is dealt with by assuming that the memory demand for each job of a high priority task τ_j executing within the response time of a low priority task τ_i is always maximum, i.e, equal to the maximum memory demand MD_j. As a result, the total memory overhead MO_i that must be accounted by τ_i during its WCRT is upper bounded by the following equation derived in [11].

$$MO_i = MD_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil \times (MD_j + \gamma_{i,j}) \qquad (7)$$

There is a significant level of pessimism involved in Equation (7) as we will demonstrate using the example given below.

Example 1. *Consider the two tasks τ_1 and τ_2 (where τ_1 has a higher priority than τ_2) presented in Fig. 1. We assume that the time d_{mem} needed to access the main memory and load a memory block to the cache is equal to 1 time unit, and that the memory demand of τ_1 and τ_2 are $MD_1 = 6$ and $MD_2 = 8$[1], respectively. We also assume that the memory block $\{9\}$ accessed by τ_1 contains some data that must be reloaded at the beginning of each of its job's execution. Fig. 1 depicts a possible schedule together with the evolution of the cache content over time. The memory blocks that must be loaded/reloaded from the main memory after each preemption or resumption are shown on a grey background.*

Initially, the cache is empty and τ_2 loads all its ECBs from the main memory as soon as it starts to execute. When τ_1 preempts τ_2 for the first time, it loads $MD_1 = 6$ memory blocks into the cache. Since there is an overlap between the ECBs of τ_1 and the UCBs of τ_2, τ_1 evicts some of the useful cache blocks of τ_2. When τ_2 resumes its execution, it has to reload $\gamma_{2,1} = 2$ cache blocks from the main memory. As the second job of τ_1 preempts τ_2, one can notice that its memory demand is no longer equal to MD_1. In fact, most of the memory blocks needed by τ_1 are still in the cache. As a consequence, τ_1 must only reload the memory blocks $\{5, 6\}$ which have been evicted by τ_2, as well as the memory block $\{9\}$ which must be reloaded at each new job execution. The same scenario happens for all the jobs released by τ_1 at the exception of the first one. Therefore, the actual memory demand for the second and third job of τ_1 is much less (i.e., 3) than $MD_1 = 6$.

[1]Note that because the same cache block may be used by several memory blocks of the same task τ_i, the worst-case memory demand MD_i of τ_i may be larger than the number of ECBs of τ_i multiplied by d_{mem}.

In the presented example, the memory blocks $\{5, 6, 7, 8, 10\}$ are called *persistent cache blocks* (PCBs) as they are never evicted from the cache when τ_1 executes in isolation. A PCB is therefore a memory block that remains cached during the entire execution of a task unless evicted by another task executing on the same processor. The cache block $\{9\}$ however is called *non-persistent cache block* (nPCB) as it must be reloaded at the beginning of each job execution. nPCBs may be cache blocks that are shared by several memory blocks of the same task, or simply some data that must be reloaded before each job execution of a task. One must note that PCBs and nPCBs are different from the notions of UCBs and ECBs in the sense that it does not matter if they are referenced more than once during a single execution of a task. However, a PCB must never be evicted from the cache by the task itself once it is fetched from the main memory.

The state-of-the-art does not consider PCBs while calculating the memory overhead suffered by a task τ_i in case of preemptions. This results in pessimistic memory overhead evaluations and hence pessimistic WCRT computations. This can easily be shown using the example of Fig. 1. If τ_2's memory overhead is computed using Eq. (7), one would get:

$$MO_2 = MD_2 + 3 \times MD_1 + 3 \times \gamma_{2,1} = 8 + 3 \times 6 + 3 \times 2 = 32$$

Equation (7) considers the worst-case memory demand, i.e., MD_1 for each job of τ_1 that executes during the response time of τ_2. As we have shown in Example 1, the actual memory demand of the second and third job of τ_1 is in fact much less. Considering the PCBs of τ_1 while calculating the memory overhead MO_2, the resulting value is given as:

$$\begin{aligned} MO_2 &= MD_2 + MD_1 + 2 \times (MD_1 - |PCB_1| \times d_{mem}) \\ &\quad + 3 \times \gamma_{2,1} \\ &= 8 + 6 + 2 \times (6 - 5 \times 1) + 3 \times 2 = 22 \end{aligned}$$

This simple example demonstrates why it is important to account for PCBs when calculating the memory demand and hence the WCRT of a task.

V. WCRT Analysis using memory overhead cost of high Priority Tasks

Two interesting properties can be observed in the example of Section IV:

1) The tasks with a high number of PCBs will have a lower memory demand after the execution of their first job than their worst-case memory demand in isolation. Therefore, we define MD_i^r as the worst-case memory demand over all the jobs of τ_i except the first one.
2) The PCBs of a task τ_j can be evicted due to the execution of lower and high priority tasks (i.e., tasks in $aff(i, j) \cup hp(j)$) between the arrivals of two successive jobs of τ_j. This requires to consider the effect of the tasks in $aff(i, j) \cup hp(j)$ on the memory demand of τ_j during the WCRT of τ_i. This extra memory demand caused by the eviction of the PCBs of τ_j by the tasks in $aff(i, j) \cup hp(j)$ is denoted by $\rho_{j,i}$.

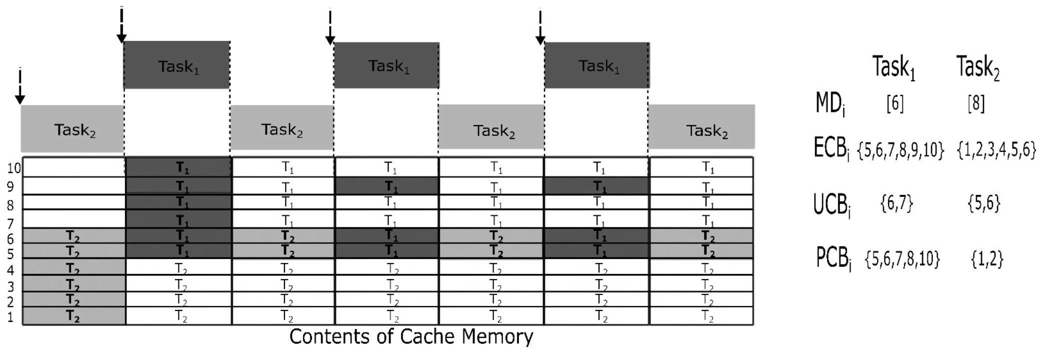

Fig. 1. Task schedule and cache content for Example 1.

$\rho_{j,i}$ can be computed using a similar formulation to the ECB-union approach described in Section III (see Eq. (2)). First, we note that every task $\tau_k \in \mathit{aff}(i,j) \cup hp(j)$ can execute between the releases of two successive jobs of τ_j. Second, we consider the fact that each task τ_k may need to load new content in all its ECBs at any time of its execution. Since we are interested in upper bounding the number of PCBs of τ_j that can be evicted by the tasks in $\mathit{aff}(i,j) \cup hp(j)$, we therefore check how many PCBs of τ_j intersect with the ECBs of the tasks in $\mathit{aff}(i,j) \cup hp(j)$. Consequently, the memory overhead $\rho_{j,i}$ is given by:

$$\rho_{j,i} = d_{mem} \times \left| PCB_j \cap \Big(\bigcup_{\forall k \in \mathit{aff}(i,j) \cup hp(j)} ECB_k \Big) \right| \quad (8)$$

Considering the two properties identified at the beginning of this section, we present a more elaborate formulation of the WCRT equation presented in Section III (see Eq. (3)):

$$\begin{aligned} R_i = & P_i + MD_i + \sum_{\forall j \in hp(i)} (P_j + MD_j) + \sum_{\forall j \in hp(i)} \gamma_{i,j}^{mul} \\ & + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i}{T_j} - 1 \right\rceil \times \left(P_j + MD_j^r + \rho_{j,i} \right) \end{aligned} \quad (9)$$

In this equation, we separately account for the processing and memory demand of each task, i.e., P_i and MD_i. Similarly, so as incorporate the effect of both MD_i and MD_i^r, we separate the execution of the first job of each preempting task from the execution of their next jobs. While the first job of each task τ_j in $hp(i)$ has a worst-case memory demand MD_j, all the other jobs have a worst-case memory demand $MD_j^r + \rho_{j,i}$, where $\rho_{j,i}$ is calculated using Equation (8). The CRPD $\gamma_{i,j}^{mul}$ is calculated using the multi-set approach given by Eq. (4). In cases where PCBs are also UCBs, double accounting of same preemption overheads can be avoided by choosing the $\min$ between C_j and $(P_j + MD_j^r + \rho_{j,i})$ for each job of τ_j after the execution of the first job.

VI. Conclusion

This work proposes a method to calculate the memory overhead of high priority tasks executing during the response time of a low priority task. In order to bound this overhead, we identified the existence of persistent and non-persistent cache blocks (i.e., PCBs and nPCBs) associated with each task. We showed with an example that, due to existence of PCBs, the memory demand of a task can significantly vary over time. We also presented an approach, complementary to [9], to upper bound the number of PCBs of a preempting task that can be evicted by the execution of the preempted tasks. Finally, we reformulated the WCRT analysis so as to consider the effect of the PCBs and the memory demand overhead. In future, we plan to extend our approach to set associative caches. We also aim to present a less pessimistic multi-set approach for memory demand overhead calculation. We further plan to generate results for the proposed WCRT analysis using available benchmarks.

Acknowledgments. This work was partially supported by National Funds through FCT/MEC (Portuguese Foundation for Science and Technology) and co-financed by ERDF (European Regional Development Fund) under the PT2020 Partnership, within project UID/CEC/04234/2013 (CISTER); also by FCT/MEC and the EU ARTEMIS JU within project(s) ARTEMIS/0003/2012 - JU grant nr. 333053 (CONCERTO) and ARTEMIS/0001/2013 - JU grant nr. 621429 (EMC2).

References

[1] G. C. Buttazzo, M. Bertogna, and G. Yao, "Limited preemptive scheduling for real-time systems. a survey," *Industrial Informatics, IEEE Transactions on*, vol. 9, no. 1, pp. 3–15, 2013.

[2] K. Jeffay, D. F. Stanat, and C. U. Martel, "On non-preemptive scheduling of periodic and sporadic tasks," in *RTSS'91*. IEEE, 1991, pp. 129–139.

[3] J. V. Busquets-Mataix, J. J. Serrano, R. Ors, P. Gil, and A. Wellings, "Adding instruction cache effect to schedulability analysis of preemptive real-time systems," in *RTAS'96*. IEEE, 1996, pp. 204–212.

[4] C.-G. Lee, J. Hahn, Y.-M. Seo, S. L. Min, R. Ha, S. Hong, C. Y. Park, M. Lee, and C. S. Kim, "Analysis of cache-related preemption delay in fixed-priority preemptive scheduling," *Computers, IEEE Transactions on*, vol. 47, no. 6, pp. 700–713, 1998.

[5] H. Tomiyama and N. D. Dutt, "Program path analysis to bound cache-related preemption delay in preemptive real-time systems," in *Proceedings of the eighth international workshop on Hardware/software codesign*. ACM, 2000, pp. 67–71.

[6] J. Staschulat, S. Schliecker, and R. Ernst, "Scheduling analysis of real-time systems with precise modeling of cache related preemption delay," in *ECRTS'05*. IEEE, 2005, pp. 41–48.

[7] Y. Tan and V. Mooney, "Timing analysis for preemptive multitasking real-time systems with caches," *ACM (TECS)*, vol. 6, no. 1, p. 7, 2007.

[8] S. Altmeyer, R. Davis, C. Maiza *et al.*, "Cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems," in *RTSS'11*. IEEE, 2011, pp. 261–271.

[9] S. Altmeyer, R. I. Davis, and C. Maiza, "Improved cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems," *Real-Time Systems*, vol. 48, no. 5, pp. 499–526, 2012.

[10] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *JACM*, vol. 20, no. 1, pp. 46–61, 1973.

[11] S. Altmeyer, R. I. Davis, L. Indrusiak, C. Maiza, V. Nelis, and J. Reineke, "A generic and compositional framework for multicore response time analysis," in *RTNS'15*. ACM, 2015, pp. 129–138.