CISTER
Research Centre in
Real-Time & Embedded
Computing Systems

Conference Paper

Work-in-Progress: A Holistic Approach to WCRT Analysis for Multicore Systems

Jatin Arora*
Syed Aftab Rashid*
Cláudio Maia*
Geoffrey Nelissen
Eduardo Tovar*

*CISTER Research Centre

CISTER-TR-220907

2022/12/05

Work-in-Progress: A Holistic Approach to WCRT Analysis for Multicore Systems

Jatin Arora*, Syed Aftab Rashid*, Cláudio Maia*, Geoffrey Nelissen, Eduardo Tovar*

*CISTER Research Centre

Polytechnic Institute of Porto (ISEP P.Porto)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail: jatin@isep.ipp.pt, syara@isep.ipp.pt, clrrm@isep.ipp.pt, gnn@isep.ipp.pt, emt@isep.ipp.pt

https://www.cister-labs.pt

Abstract

Multicore platforms share the hardware resources such as caches, interconnects, and main memory among all the cores. Due to such sharing, tasks running on different cores compete to access these shared resources which increases the execution times of those tasks in a non-deterministic manner. This is problematic for systems that run applications with stringent timing requirements. To address this issue, we propose a holistic analysis to bound the maximum inter-core contention that can be suffered by tasks from tasks running on other cores.

Work-in-Progress: A Holistic Approach to WCRT Analysis for Multicore Systems

Jatin Arora*, Syed Aftab Rashid*†, Cláudio Maia*, Geoffrey Nelissen‡, Eduardo Tovar*
*CISTER, ISEP, Porto, Portugal †VORTEX CoLab, Portugal ‡Eindhoven University of Technology, Eindhoven, the Netherlands

2022 IEEE Real-Time Systems Symposium (RTSS) | 978-1-6654-5346-2/22/$31.00 ©2022 IEEE | DOI: 10.1109/RTSS55097.2022.00054

I. Introduction

Commercial-off-the-shelf (COTS) multicore processors have become a preferable choice for modern systems to meet the increasing functionalities and computational demand of modern applications. However, the adoption of multicore platforms in *hard real-time systems*, i.e., systems that run applications with stringent timing requirements, is still under scrutiny. The main challenge that hinders the use of COTS multicore platforms in hard real-time systems is their unpredictability, which originates from the sharing of different hardware resources. A task executing on one core of a multicore platform has to compete with other co-running tasks (running on other cores) to access hardware resources such as the last-level cache (LLC), the interconnect (e.g., memory bus), and the main memory. This competition leads to inter-core contention which can significantly impact the Worst-Case Execution Time (WCET) and Worst-Case Response Time (WCRT) of tasks.

Considering the impact of shared resource contention on the WCET/WCRT of tasks, several works have been proposed in the literature that focus on analyzing cache contention (see [8]), memory bus contention (see Sections 2.2 and 4.1 of [12]) and main memory contention (see Section 4.2 of [12]). However, most of these works mainly focus on only one shared resource and make assumptions about the behavior of other resources that can lead to sometimes optimistic, sometimes pessimistic WCET/WCRT bounds. For example, most works that focus on analyzing bus/memory contention assume that each job of each task that executes during a given time interval will always have the worst-case memory access demand, i.e., the maximum number of main memory accesses issued during the execution of one job of a task in isolation. However, this assumption can be quite pessimistic since the LLC should allow for data re-use and thus reduce the overall number of accesses to main memory [13]. Similarly, works that focus on cache contention [14] assume a constant service time for each access to the memory bus and the main memory. This assumption can result in pessimistic or optimistic results if all factors that can affect the service time, e.g., bus arbitration policy, number and type of competing requests, memory controller behavior, etc., are not considered.

In multicore platforms, when a task executing on one core needs code/data, it first checks the local cache or LLC. If the requested code/data is not available in the cache, i.e., cache miss, a memory bus request is initiated. Depending on the arbitration policy at the memory bus, the code/data request will then be forwarded to the memory controller, which then serves the request from the main memory depending on its type, i.e., read or write, and the memory arbitration policy. As a consequence of this hierarchy of shared resources, the inter-core contention caused by each shared resource is interdependent. For example, the inter-core bus contention depends on the number of bus requests issued by tasks, which in turn depends on the number of cache misses. Similarly, the main memory contention suffered by tasks can be directly influenced by the number of cache misses, bus arbitration policy and the type of memory request. This highlights the importance to analyze the inter-core contention at each shared resource considering their interdependence on other shared resources. Few existing works [9], [14] have already considered the interdependence between shared resources when analyzing multicore systems. [9] proposed a multicore response time analysis framework that analyzes contention due to caches, memory bus and the main memory. However, the main focus of [9] is the memory bus, and the models considered for cache and main memory are coarse-grained. Similarly, [14] considers a relatively fine-grained cache and memory bus model, but abstracts the main memory contention by assuming a constant service time for each memory request.

This work follows the footsteps of [9], [14] by providing a holistic approach to multicore WCRT analysis (see Figure 1 for a high-level overview) but uses a fine-grained analysis to bound the inter-core contention at the LLC, memory bus and the main memory. We start by considering a more predictable task execution model, i.e., the 3-phase task model (see Section II for details), in comparison to the generic task model considered by [9], [14]. We then model the cache behavior of tasks to bound the worst-case number of LLC misses of tasks. The bus contention analysis uses the LLC misses as input and computes the maximum bus contention of tasks considering the arbitration policy at the bus and the type of LLC misses,

This work was financed by FCT and EU ECSEL JU within project ADACORSA (ECSEL/0010/2019 - JU grant nr. 876019) - The JU receives support from the EU's Horizon 2020 R&I Programme and Germany, Netherlands, Austria, France, Sweden, Cyprus, Greece, Lithuania, Portugal, Italy, Finland, Turkey (Disclaimer: This document reflects only the author's view and the Commission is not responsible for any use that may be made of the information it contains); it is also a result of the work developed under the CISTER Unit (UIDP/UIDB/04234/2020), financed by FCT/MCTES (Portuguese Foundation for Science and Technology); and under project POCI-01-0247-FEDER-045912 (FLOYD), financed in the scope of the CMU Portugal, by the European Regional Development Fund (ERDF) under COMPETE 2020, also by FCT under PhD grant 2020.09532.BD.

2576-3172/22/$31.00 ©2022 IEEE
DOI 10.1109/RTSS55097.2022.00054

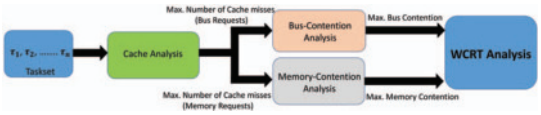

Fig. 1: Holistic WCRT Analysis

i.e., read/write. The memory contention analysis also uses LLC misses as input, however, it also considers the underlying arbitration policy used by the memory controller to bound memory contention. Finally, the WCRT of tasks is computed by considering contention at all the shared resources. This WiP mainly focuses on deriving tighter bounds on the number of bus/memory requests generated during the execution of 3-phase tasks. A quick discussion on how those bounds can be integrated into a holistic analysis is provided in Sections IV and V. The actual integration is left as future work.

II. System Model

We assume a multicore system with m identical cores, i.e., $\pi_1, \pi_2, \ldots, \pi_m$. All cores share the Last-Level Cache (LLC), the memory bus, and the main memory. The LLC is assumed to be *partitioned* among cores such that each core has a non-overlapping cache partition. We assume that the cache partition of each core is sufficiently large to store the code and data of the task with the largest memory footprint among all tasks assigned to that core. The cache employs a write-back policy and the memory bus arbitration policy is round-robin. Similarly to [2], [4], [7], [10], we assume that the main memory is a DRAM which is composed of several ranks. Each rank consists of multiple banks and each bank contains rows and columns to store the data/code. The per-bank scheduling policy is First-Ready First-Come-First-Serve (FR-FCFS) and the inter-bank scheduling policy is round-robin.

Task Model: In the considered 3-phase task model [3], the execution of each task is divided into three phases: Acquisition (A), Execution (E), and Restitution (R). During the A-phase, the task prefetches the required code/data from the main memory via the memory bus and stores it in a cache partition. In the E-phase, the core executes the task by using the data already available in its cache partition without accessing the bus/main memory. Finally, during the R-phase, the task write-back the modified data/code to the main memory. Note that the task accesses the bus and main memory only during the A- and R-phases (also called the memory phases). There is no access to the bus or the main memory during the E-phase.

We consider a task set Γ comprising n sporadic tasks partitioned among cores at design time. Each task τ_i is characterized by T_i, i.e., the minimum inter-arrival time between two jobs of τ_i, C_i, i.e., the Worst-Case Execution Time (WCET) of τ_i in isolation, and D_i, i.e., the relative deadline of τ_i. We assume fixed-priority non-preemptive scheduling with task priorities assigned using any fixed task priority algorithm such as Rate/Deadline monotonic. The maximum number of main memory accesses that can be generated during the A-phase (resp. R-phase) of a task τ_i is termed as its worst-case memory access demand and is denoted by MD_i^A (resp. MD_i^R). Similarly, the WCET of the E-phase is denoted by E_i. The values of MD_i^A, MD_i^R, and E_i can be computed using static and/or measurement based techniques. For notational convenience, we define the sets of tasks: $hep(i)$, $hp(i)$, and $lp(i)$ as those containing the tasks with priorities higher or equal, higher, and lower than that of τ_i, respectively. For clarity, throughout the document, we refer to the core on which task τ_i (i.e., the task under analysis) executes as the ***local core***, denoted by π_l. Any core other than the local core is referred to as a ***remote core***, usually denoted by π_r.

III. Cache Analysis

For a task τ_i scheduled using fixed-priority non-preemptive scheduling, the WCRT is observed during the longest level-i busy window [1]. Formally, the **level-i busy window** is defined as [11] "*a time interval (a, b) in which the pending workload of tasks with priorities higher than or equal to that of task τ_i is positive for all $t \in (a, b)$ and 0 at the boundaries a and b*". In this section, we present the cache analysis to bound the maximum number of bus/main memory accesses (i.e., LLC misses) that can be generated during the A-phases and R-phases of the tasks executed in a level-i busy window (remember there is no main memory access during the E-phases).

Most of the existing works [2], [6] that analyze the bus/main memory contention for the 3-phase task model assume that the bus/main memory accesses generated during the A-phase and the R-phase of every job of each task τ_i is equal to MD_i^A and MD_i^R, respectively. This implies that, whenever it executes a new job, a task must load all its ***Evicting Cache Blocks*** (ECBs), i.e., "*the set of all memory blocks that a task τ_i can use during its execution*" [8]. This assumption is pessimistic as there can be some ECBs that will not be evicted from the caches between the execution of two jobs. Consequently, the number of ECBs to be loaded from the main memory by the subsequent jobs of the same task can be less than its worst-case memory access demand. This issue can be addressed using the concept of ***Persistent Cache Blocks*** (PCBs) [13], i.e., "*memory blocks that once loaded in the cache by the task, will never be evicted or invalidated by the task itself*". This implies that if one job of a task has loaded all its PCBs, the maximum number of bus/main memory accesses that can be generated during the subsequent jobs of the task when it executes in isolation can be less than its worst-case memory demand. This memory demand is known as the ***Residual Memory Demand*** and is defined as [13] "*the worst-case memory access demand of any job of a task when assuming all its PCBs are already loaded in the cache*". The residual memory demand is defined for a task executing in isolation. However, a task τ_i will likely have to share the core on which it executes with other tasks. Consequently, the PCBs that were loaded by one job of τ_i can be evicted by those other tasks. This phenomenon is known as ***Cache Persistence Reload Overhead*** (CPRO) [13], i.e., "*the overhead suffered by a task τ_h during the response time of another task τ_i executing on the same core due to evictions of its PCBs by tasks in $hep(i) \setminus \tau_h$*".

Even though the concept of cache persistence was initially proposed for the generic task model, the 3-phase tasks can

also use it to tightly bound the number of bus/main memory accesses that can be generated during the memory phases.

A. Upper Bounding Memory Accesses by the Local Core

To upper bound the maximum number of bus/main memory accesses generated by all the tasks that execute on the *local core* π_l during the level-i busy window $W_{i,l}$, we first need to bound the maximum number of bus/main memory accesses during the A- and R-phases of those tasks.

The total number of bus/main memory accesses that can be generated during the *A-phases* of all the jobs of task τ_i when it executes *in isolation* during any time interval of length $W_{i,l}$ is given by the following equation.

$$|PCB_i|+\bar{MD}_i^A + (\left\lceil \frac{W_{i,l}}{T_i} \right\rceil - 1) \times \bar{MD}_i^A \tag{1}$$

where PCB_i is the set of PCBs of task τ_i and $\bar{MD}_i^A$ is the residual memory demand of the A-phase of task τ_i.

Proof sketch: τ_i releases at most $\left\lceil \frac{W_{i,l}}{T_i} \right\rceil$ jobs in the level-i busy window of length $W_{i,l}$, and the A-phase of the first job of τ_i must load all its ECBs, i.e., $|PCB_i|+\bar{MD}_i^A$. Furthermore, by definition of the residual memory demand $\bar{MD}_i^A$, the subsequent jobs of τ_i can make at most $(\left\lceil \frac{W_{i,l}}{T_i} \right\rceil - 1) \times \bar{MD}_i^A$ bus/memory accesses. Thus, Equation 1 bounds the maximum number of bus/memory accesses generated during all the A-phases of task τ_i when it executes in isolation during $W_{i,l}$.

As discussed earlier, other tasks that can execute on the same core as τ_i can evict the PCBs of τ_i. Thus, we also need to account for the maximum CPRO that can be suffered by task τ_i, when computing its memory accesses.

The maximum CPRO that can be suffered by the A-phase of *one job* of task τ_i is upper bounded by the following equation (See Theorem 1 of [13] for a formal proof)

$$\rho_i = PCB_i \cap \Big(\bigcup_{\forall \tau_h \in hep(i) \setminus \tau_i} ECB_h \Big) \tag{2}$$

where PCB_i is the set of PCBs of task τ_i, and $\bigcup_{\forall \tau_h \in hep(i) \setminus \tau_i} ECB_h$ is the set union of the ECBs of all tasks in $hep(i) \setminus \tau_i$ that can potentially evict the PCBs of τ_i.

The key insight for Equation 2 is that a task $\tau_h \in hep(i)$ can only evict the PCBs of task τ_i if the ECBs of τ_h shares the same cache lines as the PCBs of τ_i.

Note that a lower priority task cannot evict the PCBs of τ_i as it can only execute at the start of the level-i busy window.

Lemma 1. *The maximum number of bus/main memory accesses that can be generated during all the A-phases of all the jobs released by task τ_i during any time window of length $W_{i,l}$ is denoted by $\hat{MD}_i^A(W_{i,l})$, where*

$$\begin{aligned} \hat{MD}_i^A(W_{i,l}) = \min \Big(& \left\lceil \frac{W_{i,l}}{T_i} \right\rceil \times MD_i^A, \\ & |PCB_i|+\bar{MD}_i^A + (\left\lceil \frac{W_{i,l}}{T_i} \right\rceil - 1) \times (\bar{MD}_i^A + |\rho_i|) \Big) \end{aligned} \tag{3}$$

Proof. From Equation 1, we know that $|PCB_i|+\bar{MD}_i^A + (\left\lceil \frac{W_{i,l}}{T_i} \right\rceil - 1) \times \bar{MD}_i^A$ upper bounds the maximum number of bus/main memory accesses generated during the A-phases of τ_i during $W_{i,l}$ when it executes in isolation. From Equation 2, we know that ρ_i bounds the maximum CPRO that can be suffered by the A-phase of one job of τ_i. In the worst-case, the CPRO can be suffered by the A-phases of all the jobs except the A-phase of the first job of τ_i (as it loads all its ECBs) that execute during $W_{i,l}$. Consequently, $(\left\lceil \frac{W_{i,l}}{T_i} \right\rceil - 1) \times |\rho_i|$ bounds the maximum CPRO that can be suffered by task τ_i during $W_{i,l}$. Finally, by definition of MD_i^A, the maximum number of bus/main memory accesses that can be generated during the A-phases of $\left\lceil \frac{W_{i,l}}{T_i} \right\rceil$ jobs of τ_i cannot be greater than $\left\lceil \frac{W_{i,l}}{T_i} \right\rceil \times MD_i^A$. Thus, Equation 3 upper bounds the maximum number of bus/main memory accesses that can be generated during all the A-phases of all jobs of task τ_i during $W_{i,l}$. □

Using Lemma 1, the maximum number of bus/main memory accesses that can be generated during all the *A-phases* of *all the tasks* that can execute on the *local core* π_l during any time window of length $W_{i,l}$ is given by $\alpha_{i,l}^A(W_{i,l})$, where

$$\alpha_{i,l}^A(W_{i,l}) = \max_{\forall \tau_j \in lp(i)} \{MD_j^A\} + \sum_{\forall \tau_h \in hep(i)} \hat{MD}_h^A(W_{i,l}) \tag{4}$$

Proof sketch: In Equation 4, $\max_{\forall \tau_j \in lp(i)} \{MD_j^A\}$ upper bounds the bus/memory requests generated during the A-phase of *one job* of a lower priority task that can execute at the start of a level-i busy window. Furthermore, by applying Lemma 1 to each task $\tau_h \in hep(i)$, the maximum number of bus/main memory requests generated during the A-phases of all the tasks in $hep(i)$ during $W_{i,l}$ is given by $\sum_{\forall \tau_h \in hep(i)} \hat{MD}_h^A(W_{i,l})$. Therefore, Equation 4 upper bounds the maximum number of bus/main memory accesses that can be generated during all the A-phases of all tasks that can execute during $W_{i,l}$.

Having bounded the number of bus/main memory accesses that can be generated during the A-phases, we can use a similar approach to bound the number of bus/main memory accesses during the R-phases of tasks. However, we know that in the 3-phase task model, an E-phase always precedes an R-phase. So, all cache lines that are dirty at the end of the E-phase of a task, will be written-back (and invalidated) during the R-phase. Therefore, all such cache lines can not hold PCBs. Consequently, the memory access demand of an R-phase of a task will only account for write-backs due to non-persistent memory blocks and is given by MD_i^R. Building upon this, the maximum number of bus/main memory accesses that can be generated during the *R-phases* of *all tasks* executing on the local core during $W_{i,l}$ is upper bounded by

$$\alpha_{i,l}^R(W_{i,l}) = \max_{\forall \tau_j \in lp(i)} \{MD_j^R\} + \sum_{\forall \tau_h \in hep(i)} \left\lceil \frac{W_{i,l}}{T_h} \right\rceil \times MD_h^R \tag{5}$$

where $\sum_{\forall \tau_h \in hep(i)} \left\lceil \frac{W_{i,l}}{T_h} \right\rceil \times MD_h^R$ bounds the maximum number of bus/main memory requests issued by the R-phases of all the tasks in $hep(i)$ during $W_{i,l}$; and $\max_{\forall \tau_j \in lp(i)} \{MD_j^R\}$ bounds the maximum number of bus/main memory requests generated by the R-phase of one job of a lower priority task.

B. Upper Bounding Memory Accesses by the Remote Core

As discussed in Section III-A, the maximum number of bus/main memory accesses of tasks depends on the PCBs, residual memory demand and CPRO. The notion of PCBs and residual memory demand can be applied to the tasks running on the remote core identically to the tasks of the local core (using Equation 1). However, the upper bound on the CPRO given by Equation 2 does not hold for a task that executes on the remote core because unlike the local core, we cannot estimate the set of tasks that execute on a remote core during any time interval of length $W_{i,l}$. Considering this, a task that executes on the remote core can suffer the CPRO from any task that can execute on that remote core. Therefore, the maximum CPRO that can be suffered by the A-phase of one job of task τ_u that executes on a remote core π_r is given by $\bar{\rho_u}$, where

$$\bar{\rho_u} = PCB_u \cap \Big(\bigcup_{\forall \tau_k \in \Gamma^r \setminus \tau_u} ECB_k \Big) \tag{6}$$

where Γ^r is the set of all tasks running on a remote core π_r, PCB_u is the set of PCBs of task τ_u, and $\bigcup_{\forall \tau_k \in \Gamma^r \setminus \tau_u} ECB_k$ is the set union of all ECBs of all tasks in Γ^r except τ_u. The maximum number of bus/main memory accesses that can be generated during all the *A-phases* of a task τ_u running on a remote core π_r during $W_{i,l}$ is given by the following lemma.

Lemma 2. *The maximum number of bus/main memory accesses that can be generated during all the A-phases of all the jobs of a task τ_u running on the remote core π_r during any time window of length $W_{i,l}$ is given by $\hat{MD}^A_u(W_{i,l})$ where*

$$\begin{aligned} \hat{MD}^A_u(W_{i,l}) = \min \Big(& \left\lceil \frac{W_{i,l}}{T_u} \right\rceil \times MD^A_u, \\ & |PCB_u| + \bar{MD}^A_u + (\left\lceil \frac{W_{i,l}}{T_u} \right\rceil - 1) \times (\bar{MD}^A_u + |\bar{\rho_u}|) \Big) \end{aligned} \tag{7}$$

Proof. The proof directly follows from Lemma 1 except that the computation of $\bar{\rho_u}$ is given by Equation 6. □

Applying Lemma 2 to all tasks of the remote core, the maximum number of bus/main memory accesses that can be generated during all the *A-phases* of *all tasks* released on the remote core π_r during any time window of length $W_{i,l}$ is upper bounded by $\beta^A_{i,r}(W_{i,l}) = \sum_{\forall \tau_u \in \Gamma^r} \hat{MD}^A_u(W_{i,l})$.

Finally, we can compute the maximum number of bus/main memory accesses generated during the *R-phases* of *all tasks* running on the remote core π_r assuming that all non-persistent cache blocks will be written-back and invalidated during the R-phase is given by $\beta^R_{i,r}(W_{i,l}) = \sum_{\forall \tau_u \in \Gamma^r} \left\lceil \frac{W_{i,l}}{T_u} \right\rceil \times MD^R_u$.

IV. Bus Contention Analysis

The existing works [5], [6] that analyze the bus contention for the 3-phase task model does not consider cache persistence and assume the maximum number of bus requests issued by all the A/R-phases of tasks. Furthermore, they assume the maximum service time for each request and do not take into account the type of request, i.e., read/write. To mitigate this pessimism, we will use the persistence-aware cache analysis presented in Section III to tightly upper bound the number of bus requests and will model the memory bus at a fine-grained level by considering the bus transaction time to serve the read and write requests. The main insight is that a read request may incur two bus transactions, i.e., 1) incoming read request from the core to the main memory; and 2) outgoing read response from the main memory to the core, whereas the write request only incurs one bus transaction, i.e., write request from the core to the memory. The bus contention can then be computed by considering the number of bus transactions, service time for each bus transaction, bus arbitration policy, etc.

V. Main Memory Contention Analysis

The existing work [2] that analyzes the main memory contention for the 3-phase task model does not consider the cache persistence and assumes the worst-case memory access demand for each memory phase. Furthermore, it is tailored for a specific architecture that: 1) facilitates point-to-point connection between cores and memory; and 2) employs write-batching, i.e., memory controller prioritize read requests over write requests. To address these open issues, we will bound the main memory contention by incorporating the persistence-aware cache analysis and by considering different possible configurations of the memory controller, e.g., with/without write-batching, and its impact on the main memory contention.

VI. Conclusion

This work presents the idea for a holistic WCRT analysis for the 3-phase task model by considering the inter-core contention at shared resources such as caches, memory bus, and main memory. We show how the notion of cache persistence can be used to tightly bound the number of bus/main memory accesses of 3-phase tasks. We will be using these results to compute bus/main memory contention in our future work.

References

[1] R. J. Bril et al. Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption revisited. In *ECRTS'07*, pages 269–279, 2007.

[2] D. Casini et al. A holistic memory contention analysis for parallel real-time tasks under partitioned scheduling. In *RTAS 2020*, 2020.

[3] G. Durrieu et al. Predictable Flight Management System Implementation on a Multicore Processor. In *ERTS'14*, 2014.

[4] H. Kim et al. Bounding memory interference delay in cots-based multi-core systems. In *RTAS 2014*, pages 145–154, 2014.

[5] J. Arora et al. Bus-contention aware wcrt analysis for the 3-phase task model considering a work-conserving bus arbitration scheme. *JSA*, 2022.

[6] J. Arora et al. Schedulability analysis for 3-phase tasks with partitioned fixed-priority scheduling. *Elsevier JSA*, 131:102706, 2022.

[7] M. Hassan et al. Analysis of Memory-Contention in Heterogeneous COTS MPSoCs. In *ECRTS*, 2020.

[8] M. Lv et al. A survey on static cache analysis for real-time systems. *Leibniz Trans. Embed. Syst.*, 3(1):05:1–05:48, 2016.

[9] R. I. Davis et al. An extensible framework for multicore response time analysis. *Real-Time Systems*, July 2017.

[10] M. Hassan et al. Bounding dram interference in cots heterogeneous mpsocs for mixed criticality systems. *IEEE TCAD*, 37(11), 2018.

[11] J. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. *RTSS 1990*, pages 201–209, 1990.

[12] C. Maiza et al. A Survey of Timing Verification Techniques for Multi-Core Real-Time Systems. *ACM Computing Surveys*, 52(3), 2019.

[13] S. A. Rashid et al. Cache-persistence-aware response-time analysis for fixed-priority preemptive systems. In *ECRTS*, 2016.

[14] S. A. Rashid et al. Cache persistence-aware memory bus contention analysis for multicore systems. In *DATE*, 2020.