CISTER
Research Centre in
Real-Time & Embedded
Computing Systems

Conference Paper

Improved Bus Contention Analysis for 3-Phase Tasks

The paper is accepted as a full paper in RTCSA 2023.

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

*CISTER Research Centre

CISTER-TR-230505

2023/08/30

Improved Bus Contention Analysis for 3-Phase Tasks

Jatin Arora*, Syed Aftab Rashid*, Geoffrey Nelissen, Cláudio Maia*, 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, gnn@isep.ipp.pt, clrrm@isep.ipp.pt, emt@isep.ipp.pt

https://www.cister-labs.pt

Abstract

The 3-phase task execution model has shown to be a good candidate to tackle the memory bus contention problem. It divides the execution of tasks into computation and memory phases that enable a fine-grained memory bus contention analysis. However, existing works that focus on the bus contention analysis for 3-phase tasks, neglect the fact that memory bus contention strongly relates to the number of bus/memory requests generated by tasks, which, in turn, depends on the content of the cache memories during the execution of those tasks. These existing works assume that the worst-case number of bus/memory requests will be generated during all the memory phases of all tasks, irrespective of the already existing content in the cache memory. This overestimates the memory bus contention of tasks, leading to pessimistic worst-case response time (WCRT) bounds.

This work proposes a holistic approach towards bus contention analysis for 3-phase tasks by (1) deriving an upper bound on the actual cache misses of tasks that lead to bus/memory requests; (2) improving State-of-the-Art (SoA) bus contention analysis of two bus arbitration schemes that dominate all existing works on the bus contention analysis for 3-phase tasks; and (3) performing an extensive experimental evaluation under different settings to compare the proposed analysis against the SoA. Results show that incorporating a tighter bound on the number of cache misses of tasks into the bus contention analysis can lead to a significant improvement in task set schedulability.

© 2023 CISTER Research Center
www.cister-labs.pt

Improved Bus Contention Analysis for 3-Phase Tasks

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

***Abstract*—The 3-phase task execution model has shown to be a good candidate to tackle the memory bus contention problem. It divides the execution of tasks into computation and memory phases that enable a fine-grained memory bus contention analysis. However, existing works that focus on the bus contention analysis for 3-phase tasks, neglect the fact that memory bus contention strongly relates to the number of bus/memory requests generated by tasks, which, in turn, depends on the content of the cache memories during the execution of those tasks. These existing works assume that the worst-case number of bus/memory requests will be generated during all the memory phases of all tasks, irrespective of the already existing content in the cache memory. This overestimates the memory bus contention of tasks, leading to pessimistic worst-case response time (WCRT) bounds.**

This work proposes a holistic approach towards bus contention analysis for 3-phase tasks by (1) deriving an upper bound on the actual cache misses of tasks that lead to bus/memory requests; (2) improving State-of-the-Art (SOTA) bus contention analysis of two bus arbitration schemes that dominate all existing works on the bus contention analysis for 3-phase tasks; and (3) performing an extensive experimental evaluation under different settings to compare the proposed analysis against the SOTA. Results show that incorporating a tighter bound on the number of cache misses of tasks into the bus contention analysis can lead to a significant improvement in the task set schedulability.

I. Introduction

The adoption of multicore platforms in *hard real-time systems*, i.e., systems that run applications with stringent timing requirements, is still under the scrutiny of academia and industry. The main challenge that hinders the use of commercial off-the-shelf (COTS) multicore platforms in hard real-time systems is their unpredictability, which originates from the sharing of different hardware resources, e.g., Last-Level Cache (LLC), the interconnect (e.g., memory bus), and the main memory. A task executing on one core of a multicore platform has to compete with other co-running tasks (running on other cores) to access these *shared resources*. For instance, the shared memory bus connects all the cores to the main memory. Due to such sharing, tasks running on different cores may have to compete to access the memory bus in order to read/write data/code from/to the main memory, resulting in *bus contention*. This bus contention can significantly increase the Worst-Case Execution Time (WCET) and Worst-Case Response Time (WCRT) of tasks.

The *3-phase task model* [3], [19] has been extensively used by the state-of-the-art to tackle the memory/bus contention problem [1], [5], [8], [9], [16], [18]. The 3-phase task model divides the execution of each task into distinct *computation* and *memory phases* such that a task can only access the shared memory bus/main memory during its memory phase and the core execute the task during the computation phase without accessing the shared bus/main memory. Although the 3-phase task model makes bus/memory access patterns of tasks more predictable, 3-phase tasks can still suffer bus/memory contention, e.g., when multiple concurrent tasks running on different cores try to access the bus/memory. Considering the impact bus contention can have on the WCET/WCRT of tasks, several works in the state-of-the-art [1], [8], [9], [16], [25] have focused on analyzing the maximum bus contention that can be suffered by 3-phase tasks and its impact on taskset schedulability. In a few recent works, Arora et al. [8], [9] have presented bus contention analysis for 3-phase tasks that dominate all the existing bus contention analysis for 3-phase tasks. However, when computing bus contention, these works [8], [9] assume that the number of bus/memory requests that can be generated during the memory phases of each job of a task is always equal to its worst-case memory access demand in isolation. Although this assumption is safe, it can lead to pessimistic results, especially for platforms with cache memories. Caches are smaller faster memories that store the recently referenced data/instructions of tasks and allow for data re-use, i.e., cache content fetched during the execution of one job of a task may be re-used during the execution of a subsequent job of the same task [20]. This can significantly reduce the number of bus/memory requests generated during the execution of subsequent jobs of tasks. However, this assumption is not considered by [8], [9] and hence these works result in overestimating the bus contention of tasks.

In this work, we exploit the interdependence between the cache memory, bus requests and bus contention, i.e., the bus contention suffered by tasks depends on the number of bus requests which in turn depends on the number of cache misses, to improve the analysis presented in [8], [9]. First, we analyze the cache to tightly upper bound the number of cache misses that lead to bus/memory requests during the memory phases. We then improve the existing bus contention analyses [8], [9] by incorporating a tighter bound on the number of cache misses/bus requests into the analysis. Finally, we propose a WCRT-based schedulability analysis by integrating the bounds on bus contention into the WCRT of tasks. Formally, our main **contributions** are:

*Corresponding author

(1) Upper bounding cache misses of 3-phase tasks to compute bus/memory requests generated during the memory phases of tasks;
(2) Using the derived bound on cache misses, we improve the existing memory bus contention analyses for 3-phase tasks considering fixed-priority partitioned scheduling [8], [9];
(3) Integrating the bounds on bus contention into a WCRT formulation to perform schedulability analysis; and
(4) Extensive empirical evaluation under different settings to compare our proposed improved bus contention analyses to the existing bus contention analyses [8], [9]. Experimental results show that bus contention analyses that consider the interdependence between the cache misses and bus requests can improve taskset schedulability by up to 55%.

II. System Model

We assume a multicore system comprising m identical cores $(\pi_1, \pi_2, \ldots, \pi_m)$ that share the Last-Level Cache (LLC). The LLC is assumed to be partitioned among all the cores such that each core has an individual non-overlapping partition. Each cache partition assigned to the cores is assumed to be large enough to store all the data/code required by the task with the largest memory footprint that executes on that core. We assume that cache employs the write-back policy, is direct-mapped, and, is unified, i.e., it can store data as well as instructions. Furthermore, the write-allocate write-miss policy is assumed in the case of write-miss, which means that the memory block being written to is first loaded in the cache before performing the write operation. Note that the cache analysis presented in Section IV assumes a direct-mapped cache, however, it can be extended to set-associative caches by building on the analysis presented in [21].

We assume that a shared memory bus connects all cores to the main memory. Similarly to existing works [1], [6], [8], [9], [16], [22], [25], a single channel shared memory bus is assumed that can handle one memory request at a time and remains busy until the completion of the ongoing memory request. The maximum service time taken by the bus and the main memory to serve one memory request is given by t^{mem}. Although the proposed cache-aware analysis is applicable to all the bus arbitration policies, in this work, we focus on Round Robin (RR) and First-Come-First-Serve (FCFS) based bus arbitration schemes studied in [8] and [9], respectively.

A. Task Model

This work considers the 3-phase task model (also known as the AER model) in which the execution of a task is divided into three phases namely Acquisition (A), Execution (E), and Restitution (R). The A- and R-phases are considered memory phases, i.e., the time intervals in which the task can fetch and write-back data/code from/to the main memory via the memory bus, and the E-phase is the computation phase, i.e., the time interval in which the task only performs computations using the preloaded data and does not issue any main memory request. When a task is released, it executes its A-phase to fetch the required data/code from the main memory and store it in cache memory. It then executes its E-phase by accessing the data/code that is already available in the cache, without the need to access bus/memory. Finally, the task writes the modified data back to the main memory during the R-phase.

We consider a task set Γ comprising n sporadic tasks $(\tau_1, \tau_2, \ldots \tau_n)$ partitioned among cores at design time. T_i denotes the minimum inter-arrival time between two consecutive jobs of task τ_i, and D_i denotes its relative deadline. We assume that tasks have constrained deadline, i.e., $D_i \leq T_i$. The maximum number of memory requests that can be issued during the A-phase (resp. R-phase) of one job of task τ_i *in isolation* is denoted by MD_i^A (resp. MD_i^R). Similarly, the WCET of the E-phase is denoted by C_i^E. Note that the values of MD_i^A, MD_i^R, and C_i^E can be obtained by static analysis, measurement-based analysis, or by using the combination of both [27]. Finally, the WCET of task τ_i *in isolation* is denoted by C_i where $C_i = (MD_i^A + MD_i^R) \times t^{mem} + C_i^E$. We assume that tasks are scheduled using *fixed-priority non-preemptive scheduling* with priorities assigned using any fixed-priority algorithm such as Rate Monotonic/Deadline Monotonic [15].

Throughout the paper, 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. Similarly, any core other than the local core is referred to as a *remote core*, usually denoted by π_r.

For notational convenience, we define $hep_{i,l}$ to denote the set of tasks with a priority higher than or equal to that of τ_i running on a given core π_l. Similarly, $hp_{i,l}$ (and $lp_{i,l}$) denote the set of tasks with a priority higher (and lower) than that of τ_i, running on core π_l.

III. Background

This section presents the essential background on cache related concepts that we later use to build our analysis in Section IV.

When analyzing the worst-case memory access demand of tasks, many existing works [1], [2], [5]–[9], [11], [16], [24], [25] assume that each job of a task τ_i that execute during a time window of length Δ will always issue MD_i main memory accesses, i.e., the worst-case memory access demand of τ_i in isolation. This, in other words, means that each job of task τ_i that executes during Δ will always load all its ECBs from the main memory to the cache.

Evicting Cache Blocks (ECBs) [26]: *The set of all memory blocks that may be used by a task τ_i during its execution.*

Clearly, this assumption is pessimistic, as subsequent jobs of task τ_i that execute during Δ can *re-use* some ECBs already available in the cache due to a previous job of τ_i. These re-usable ECBs are called Persistent Cache Blocks (PCBs) [20].

Persistent Cache Blocks (PCBs) [20]: *All memory blocks used by a task, that once loaded in the cache, will never be evicted or invalidated by the task itself.*

For a task τ_i executing in isolation, if all its PCBs are already loaded in the cache, e.g., by a previous job of τ_i, the memory access demand for subsequent jobs of τ_i can be much lower than the worst-case memory access demand of

τ_i in isolation. This memory access demand of the task τ_i is called residual memory access demand.

Residual Memory Access Demand [20]*: The worst-case memory access demand of any job of task τ_i considering that all its PCBs are already loaded in the cache.*

Considering the PCBs and residual memory access demand of task τ_i, the total number of main memory accesses made by all the jobs of task τ_i when it executes *in isolation* during any time window of length Δ is given by (see Lemma 1 of [20]):

$$MD_i^{tot}(\Delta) = \min\left(\left\lceil\frac{\Delta}{T_i}\right\rceil \times MD_i, \left\lceil\frac{\Delta}{T_i}\right\rceil \times \bar{MD}_i + |PCB_i|\right) \quad (1)$$

where $\left\lceil\frac{\Delta}{T_i}\right\rceil$ bounds the maximum number of jobs released by task τ_i during any time window of length Δ; MD_i is the worst-case memory access demand of one job of τ_i measured in isolation; $\bar{MD}_i$ is the worst-case residual memory access demand of one job of task τ_i; and $|PCB_i|$ represents the cardinality of the set of PCBs of task τ_i, i.e., the total number of PCBs of task τ_i.

Equation 1 upper bounds the total memory access demand of a task in isolation. However, a task τ_i will likely have to share the core on which it executes with other tasks. So, the PCBs that were loaded by one job of τ_i can be evicted by other tasks executing on the same core. This results in generating additional main memory overhead called Cache Persistence Reload Overhead (CPRO) [20].

Cache Persistence Reload Overhead (CPRO) [20]*: The number of main memory accesses that τ_i must make due to the evictions of its PCBs caused by the execution of tasks in $hep_{i,l} \setminus \tau_i$.*

The maximum CPRO that can be suffered by one job of task τ_i is denoted by ρ_i, and is given by (from Theorem 1 of [20])

$$\rho_i = PCB_i \cap \left(\bigcup_{\forall \tau_k \in hep_{i,l} \setminus \tau_i} ECB_k\right) \quad (2)$$

where PCB_i is the set of PCBs of task τ_i; and $\bigcup_{\forall \tau_k \in hep_{i,l} \setminus \tau_i} ECB_k$ is the set union of the ECBs of all tasks in $hep_{i,l} \setminus \tau_i$ that can potentially evict PCBs of τ_i.

IV. Cache-aware Bus Contention Analysis for 3-Phase Tasks

For a task τ_i scheduled using fixed-priority non-preemptive scheduling, the WCRT is observed during the longest level-i busy window [12]. Formally, the **level-i busy window** is defined as follows.

[Level-i busy window (from [14])]: *A level-i busy window is a time interval (a, b) in which the pending workload of tasks with priorities higher or equal to that of task τ_i is positive for all $t \in (a, b)$ and 0 at the boundaries a and b.*

Due to the problem of bus contention, the length of the level-i busy window depends not only on the behavior of set of tasks running on the same core but also on the *bus contention* caused by tasks running on remote cores. Consequently, a plethora of works have been proposed in the literature to bound bus contention for 3-phase tasks [1], [8], [9], [16]. Among all these approaches, the works like [8], [9] have proven to be the best approaches to bound bus contention for 3-phase tasks considering partitioned fixed-priority scheduling. Specifically, the analysis presented in [8] bound the bus contention for 3-phase tasks considering the *RR bus arbitration policy*. Similarly, the analysis presented in [9] bound the bus contention for 3-phase tasks considering the *FCFS bus arbitration policy*.

Even though the solutions presented in [8], [9] are safe, the bounds on bus contention derived in [8], [9] can be pessimistic. This is mainly because [8], [9] assume that the number of bus/memory requests that can be generated during the memory phases of each job of a task is always equal to its worst-case memory access demand in isolation. As discussed in Section III, this assumption can be pessimistic, as the cache content fetched during the execution of one job of a task may be re-used during the execution of a subsequent job of the same task, resulting in a reduction in the number of bus/memory requests. In the following subsections, we show that a tighter bound on cache misses/bus requests of tasks can improve the bus contention analysis presented of [8], [9].

A. Cache-aware Bus Contention Analysis for RR Policy

In the RR bus arbitration policy, when multiple cores require access to the bus then each core can only access the bus during its *bus slot*. Considering this, the bus contention that can be suffered by tasks will also depend on the number of bus slots that the local core as well as the remote core requires during the level-i busy window. Building on this, the SOTA RR bus contention analysis presented in [8] upper bounds the bus contention by first computing the maximum number of bus slots required by the local core as well as the remote cores. The maximum number of *bus slots* required by tasks running on the ***local core*** π_l during the level-i busy window of length $W_{i,l}$ is upper-bounded by $\beta_{\pi_l}(W_{i,l})$, given by the following equation (From Lemma 1 of [8])

$$\beta_{\pi_l}(W_{i,l}) = \sum_{\tau_h \in hep_{i,l}} \left\lceil\frac{W_{i,l}}{T_h}\right\rceil \times \left(\left\lceil\frac{MD_h^A \times t^{mem}}{SS}\right\rceil + \left\lceil\frac{MD_h^R \times t^{mem}}{SS}\right\rceil\right) + \max_{\forall \tau_j \in lp_{i,l}} \left\{\left\lceil\frac{MD_j^A \times t^{mem}}{SS}\right\rceil + \left\lceil\frac{MD_j^R \times t^{mem}}{SS}\right\rceil\right\} \quad (3)$$

In Equation 3, the term $\left\lceil\frac{W_{i,l}}{T_h}\right\rceil$ upper bounds the maximum number of jobs that task τ_h can release during any time window of length $W_{i,l}$. MD_h^A (resp. MD_h^R) is the maximum number of main memory accesses that can be generated by the A-phase (resp. R-phase) of one job of task τ_h; t^{mem} is the maximum time required to serve one memory request; SS is the length of the bus slot which is always greater than or equal to t^{mem}. The term $\left\lceil\frac{MD_h^A \times t^{mem}}{SS}\right\rceil$ (resp. $\left\lceil\frac{MD_h^R \times t^{mem}}{SS}\right\rceil$) represents the maximum number of bus slots required by the A-phase (resp. R-phase) of one job of task τ_h. Similarly, due to fixed-priority non-preemptive scheduling, the term $\max_{\forall \tau_j \in lp_{i,l}} \left\{\left\lceil\frac{MD_j^A \times t^{mem}}{SS}\right\rceil + \left\lceil\frac{MD_j^R \times t^{mem}}{SS}\right\rceil\right\}$ integrates the maximum number of bus slots required by lower priority tasks.

Similarly, the maximum number of *bus slots* required by tasks running on a ***remote core*** π_r during the level-i busy

window of length $W_{i,l}$ is upper-bounded by $\beta_{\pi_r}(W_{i,l})$, given by the following equation (From Lemma 2 of [8])

$$\beta_{\pi_r}(W_{i,l}) = \sum_{\tau_u \in \Gamma'_r} \left\lceil \frac{W_{i,l}}{T_u} \right\rceil \times \left(\left\lceil \frac{MD_u^A \times t^{mem}}{SS} \right\rceil + \left\lceil \frac{MD_u^R \times t^{mem}}{SS} \right\rceil \right) \tag{4}$$

We can see in Equations 3 and 4 that when bounding $\beta_{\pi_l}(W_{i,l})$ and $\beta_{\pi_r}(W_{i,l})$, it is assumed that the number of memory requests issued by each A- and R-phase of every job of each task τ_i is given MD_i^A and MD_i^R, respectively. As discussed earlier, this can yield a pessimistic bound on bus contention as well on the length of the level-i busy window and WCRT. To address this, we will now present the cache-aware bus contention analysis to improve the bounds on bus contention considering the RR bus arbitration policy.

1) Upper Bounding Memory Access Requests of the Local Core: In this section, we will upper bound the maximum number of main memory accesses generated by all tasks that execute on the *local core* π_l during the level-i busy window $W_{i,l}$. We start by bounding the maximum number of main memory accesses during the A-phases of those tasks.

Applying the notion of PCBs and residual memory access demand to the 3-phase task model, the total number of 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 the level-i busy window $W_{i,l}$ is given by the following lemma.

Lemma 1. *The total number of main memory accesses that can be generated during the* A-phases *of all jobs of task τ_i when they execute* in isolation *within any time window of length $W_{i,l}$ is given by $MD_i^{A,tot}$, where*

$$MD_i^{A,tot}(W_{i,l}) = |PCB_i| + \bar{MD}_i^A + \left(\left\lceil \frac{W_{i,l}}{T_i} \right\rceil - 1 \right) \times \bar{MD}_i^A \tag{5}$$

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

Proof. τ_i releases at most $\left\lceil \frac{W_{i,l}}{T_i} \right\rceil$ jobs in the level-i busy window of length $W_{i,l}$. We know that 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 access demand $\bar{MD}_i^A$, the A-phases of subsequent jobs of τ_i can make at most $(\left\lceil \frac{W_{i,l}}{T_i} \right\rceil - 1) \times \bar{MD}_i^A$ memory accesses. Thus, Equation 5 bounds the maximum number of main memory accesses that can be 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 the memory accesses of its A-phases.

The maximum CPRO that can be suffered by an A-phase of *one job* of task τ_i is upper bounded by the following equation (from [20])

$$\rho_i = PCB_i \cap \left(\bigcup_{\forall \tau_h \in hep_{i,l} \setminus \tau_i} ECB_h \right) \tag{6}$$

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

The key insight for Equation 6 is that a task $\tau_h \in hep_{i,l}$ 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 within the level-i busy window as it can only execute at the start of the level-i busy window.

Using Equations 5 and 6, the maximum number of memory accesses that can be generated during all the A-phases of τ_i within a level-i busy window is given by the following lemma.

Lemma 2. *The maximum number of main memory accesses that can be generated during the A-phases of all the jobs of task τ_i during any time window of length $W_{i,l}$ is denoted by $\hat{MD}_i^A(W_{i,l})$, where*

$$\begin{aligned} \hat{MD}_i^A(W_{i,l}) = \min \Big(& \left\lceil \frac{W_{i,l}}{T_i} \right\rceil \times MD_i^A, |PCB_i| + \bar{MD}_i^A \\ & + (\left\lceil \frac{W_{i,l}}{T_i} \right\rceil - 1) \times (\bar{MD}_i^A + |\rho_i|) \Big) \end{aligned} \tag{7}$$

Proof. By the definition of MD_i^A, the maximum number of 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 $\left\lceil \frac{W_{i,l}}{T_i} \right\rceil \times MD_i^A$.

From Equation 5, 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 main memory accesses generated during the A-phases of τ_i during $W_{i,l}$ when it executes in isolation. Furthermore, from Equation 6, 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 $W_{i,l}$. Therefore, Equation 7 upper bounds the maximum number of main memory accesses that can be generated during all the A-phases of all jobs of task τ_i during $W_{i,l}$. The Lemma follows. □

Applying Lemma 2 to each task in $hep_{i,l}$, the maximum number of main memory accesses that can be generated during the *A-phases* of all tasks in $hep_{i,l}$ 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

$$\alpha_{i,l}^A(W_{i,l}) = \sum_{\forall \tau_h \in hep_{i,l}} \hat{MD}_h^A(W_{i,l}) \tag{8}$$

Having bounded the number of main memory accesses that can be generated during the A-phases, we can now bound the maximum number of main memory accesses generated during the *R-phases* of tasks.

The R-phase is mainly responsible to write-back all the dirty cache lines (after the execution of the E-phase) to the main memory. In order to tightly bound the number of main memory accesses generated during the *R-phase*, a task should only

write back a subset of dirty cache blocks that can potentially be used by other tasks to load their ECBs. However, achieving this from the implementation perspective can be extremely complex because: 1) determining the address of specific cache lines that will be dirty at the end of the E-phase of the task is complex, as it depends on the run-time state; 2) enforcing a task to write-back a subset of dirty cache blocks (based on the set of ECBs of other tasks) during the R-phase can be extremely challenging, as it may require additional run-time monitoring/control mechanisms. Therefore, we assume that 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. By definition, all such cache lines cannot hold PCBs. Consequently, the memory access demand of an R-phase of a task will account for write-backs due to non-persistent memory blocks and is given by MD_i^R. Building on this, the maximum number of main memory accesses that can be generated during the *R-phases* of all tasks in $hep_{i,l}$ executing on the local core during $W_{i,l}$ is upper bounded by $\alpha_{i,l}^R(W_{i,l})$, where

$$\alpha_{i,l}^R(W_{i,l}) = \sum_{\forall \tau_h \in hep_{i,l}} \left\lceil \frac{W_{i,l}}{T_h} \right\rceil \times MD_h^R \tag{9}$$

Now we will integrate the proposed cache analysis to compute the maximum number of bus slots required by the local core during the level-i busy window using the following lemma.

Lemma 3. *The maximum number of bus slots required by tasks executing on the local core π_l during any time window of length $W_{i,l}$ is upper-bounded by $\hat{\beta}_{\pi_l}(W_{i,l})$, where*

$$\begin{aligned}\hat{\beta}_{\pi_l}(W_{i,l}) = \sum_{\tau_h \in hep_{i,l}} \min\Big(\left\lceil \frac{W_{i,l}}{T_h} \right\rceil \times \left\lceil \frac{MD_h^A \times t^{mem}}{SS} \right\rceil, \\ \left\lceil \frac{MD_h^A \times t^{mem}}{SS} \right\rceil + (\left\lceil \frac{W_{i,l}}{T_h} \right\rceil - 1) \times \left\lceil \frac{(\bar{MD}_h^A + |\rho_h|) \times t^{mem}}{SS} \right\rceil \Big) \\ + \left\lceil \frac{W_{i,l}}{T_h} \right\rceil \times \left\lceil \frac{MD_h^R \times t^{mem}}{SS} \right\rceil \\ + \max_{\forall \tau_j \in lp_{i,l}} \{ \left\lceil \frac{MD_j^A \times t^{mem}}{SS} \right\rceil + \left\lceil \frac{MD_j^R \times t^{mem}}{SS} \right\rceil \}\end{aligned} \tag{10}$$

Proof. From Lemma 2, we know that the $\min\Big(\left\lceil \frac{W_{i,l}}{T_i} \right\rceil \times MD_i^A, |PCB_i| + \bar{MD}_i^A + (\left\lceil \frac{W_{i,l}}{T_i} \right\rceil - 1) \times (\bar{MD}_i^A + |\rho_i|) \Big)$ upper bounds the number of main memory accesses of the A-phases of all jobs of task τ_h during $W_{i,l}$. Furthermore, from Equation 3, we know that it is necessary to determine the maximum number of bus slots required by the memory phases of tasks that execute on the local core π_l during $W_{i,l}$. Consequently, using Lemma 2, the maximum number of bus slots require by A-phases of all jobs of a task τ_h that execute on the local core π_l during $W_{i,l}$ is upper bounded by $\min\Big(\left\lceil \frac{W_{i,l}}{T_h} \right\rceil \times \left\lceil \frac{MD_h^A \times t^{mem}}{SS} \right\rceil, \left\lceil \frac{MD_h^A \times t^{mem}}{SS} \right\rceil + (\left\lceil \frac{W_{i,l}}{T_h} \right\rceil - 1) \times \left\lceil \frac{(\bar{MD}_h^A + |\rho_h|) \times t^{mem}}{SS} \right\rceil \Big)$. This is further extended for all tasks in $hep_{i,l}$ set (including task τ_i). Since we do not apply cache persistence to the R-phases, the number of bus slots required by R-phases can be computed identically to that of Equation 3, i.e., $\left\lceil \frac{W_{i,l}}{T_h} \right\rceil \times \left\lceil \frac{MD_h^R \times t^{mem}}{SS} \right\rceil$. Finally, the maximum number of bus slots required by one job of a lower priority task is bounded by $\max_{\forall \tau_j \in lp_{i,l}} \{ \left\lceil \frac{MD_j^A \times t^{mem}}{SS} \right\rceil + \left\lceil \frac{MD_j^R \times t^{mem}}{SS} \right\rceil \}$. □

2) Upper Bounding Memory Access Requests of Remote Core: As discussed in Section IV-A1, the maximum number of main memory accesses of tasks depends on the PCBs, residual memory access demand, and CPRO. The notion of PCBs and residual memory access demand can be applied to the A-phases of tasks running on the remote core identically to tasks of the local core (using Equation 5). However, we cannot use Equation 6 to compute CPRO because the interfering task τ_u executing on π_r may not be executing within an uninterrupted level-u busy window during the whole interval $W_{i,l}$. Therefore, we cannot assume that only tasks of higher or equal priority execute between two jobs of τ_u. We must therefore consider that any task executing on core π_r may interfere with the PCBs of τ_u during $W_{i,l}$. Thus, 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

$$\bar{\rho_u} = PCB_u \cap \Big(\bigcup_{\forall \tau_k \in \Gamma^r \setminus \tau_u} ECB_k \Big) \tag{11}$$

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.

Lemma 4. *The maximum number of main memory accesses that can be generated during 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 $\hat{MD}_u^A(W_{i,l})$, where*

$$\begin{aligned}\hat{MD}_u^A(W_{i,l}) = \min\Big(\left\lceil \frac{W_{i,l}}{T_u} \right\rceil \times MD_u^A, |PCB_u| + \bar{MD}_u^A \\ + (\left\lceil \frac{W_{i,l}}{T_u} \right\rceil - 1) \times (\bar{MD}_u^A + |\bar{\rho_u}|) \Big)\end{aligned} \tag{12}$$

Proof. The proof directly follows from Lemma 2 except that the computation of $\bar{\rho_u}$ is given by Equation 11. □

Applying Lemma 4 to all tasks of the remote core, the maximum number of 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 $\sum_{\forall \tau_u \in \Gamma^r} \hat{MD}_u^A(W_{i,l})$.

As discussed earlier, we assume that a task can invalidate and write back all its non-persistent cache blocks during the R-phase. Considering this, the maximum number of memory requests that can be generated during the R-phase of a task τ_u is upper bounded by MD_u^R. Consequently, $\sum_{\tau_u \in \Gamma_r'} \left\lceil \frac{W_{i,l}}{T_u} \right\rceil \times MD_u^R$ bounds the maximum number of memory accesses that can be generated during the R-phases of *all tasks* released on a *remote core* π_r during $W_{i,l}$. We can now use Equation 12 to bound the number of memory accesses of tasks executing on a remote core π_r under the RR bus arbitration scheme.

Lemma 5. *The maximum number of bus slots required by tasks running on the remote core π_r during any time window of length $W_{i,l}$ is upper-bounded by $\hat{\beta}_{\pi_r}(W_{i,l})$, where*

$$\hat{\beta}_{\pi_r}(W_{i,l}) = \sum_{\tau_u \in \Gamma'_r} \min\Big(\left\lceil \frac{W_{i,l}}{T_u} \right\rceil \times \left\lceil \frac{MD_u^A \times t^{mem}}{SS} \right\rceil, \left\lceil \frac{MD_u^A \times t^{mem}}{SS} \right\rceil + (\left\lceil \frac{W_{i,l}}{T_u} \right\rceil - 1) \times \left\lceil \frac{(\bar{MD}_u^A + |\bar{\rho}_u|) \times t^{mem}}{SS} \right\rceil \Big) + \left\lceil \frac{W_{i,l}}{T_u} \right\rceil \times \left\lceil \frac{MD_u^R \times t^{mem}}{SS} \right\rceil \quad (13)$$

Proof. The proof directly follows from Lemma 3 except that the computation of $\bar{\rho_u}$ is given by Equation 11. □

In a similar manner to that of Equation 13, we can improve Equations 9 and 16 of [8].

Since a system is likely to have multiple remote cores, we need to bound the maximum number of bus slots that can be utilized by tasks running on each of the remote core $\pi_r \in m$ such that $\pi_r \neq \pi_l$ using Lemma 5. Based on the values of $\hat{\beta}_{\pi_l}(W_{i,l})$ and $\hat{\beta}_{\pi_r}(W_{i,l})$, we can compute the maximum bus contention $\hat{Bus}_{i,r}(W_{i,l})$ that the local core π_l can suffer from a remote core π_r during $W_{i,l}$ by improving equations 9 and 16 of [8]. Having bounded the maximum bus contention $\hat{Bus}_{i,r}(W_{i,l})$ w.r.t each remote core $\pi_r \in m$ such that $\pi_r \neq \pi_l$, we can compute the *total bus contention* $Bus_{i,l}^{max}(W_{i,l})$ that tasks of local core π_l can suffer due to tasks of *all remote cores* during $W_{i,l}$ using the following equation (from Equation 17 of [8]).

$$Bus_{i,l}^{max}(W_{i,l}) = \sum_{r=1, r\neq l}^{m} \hat{Bus}_{i,r}(W_{i,l}) \quad (14)$$

B. Cache-aware Bus Contention Analysis for FCFS Policy

The bus contention analysis presented in [9] considers the FCFS bus arbitration policy. The first step of their analysis is to compute the maximum number of *times* the bus contention that can be *suffered* by tasks running on the *local core* π_l during $W_{i,l}$ using $N_{\pi_l}(W_{i,l})$ where $N_{\pi_l}(W_{i,l})$ is upper bounded by the following equation (from Equation 9 of [9]).

$$N_{\pi_l}(W_{i,l}) = \sum_{\tau_h \in hep_{i,l}} \left\lceil \frac{W_{i,l}}{T_h} \right\rceil + 1 \quad (15)$$

The main insight behind Equation 15 is that the maximum number of times bus contention suffered by tasks of the local core depends on the number of jobs/memory phases that tasks release on the local core π_l during $W_{i,l}$.

Similarly, the next step is to compute the maximum number of *times* the bus contention that can be *caused* by tasks running on a *remote core* π_r during $W_{i,l}$ using $N_{\pi_r}(W_{i,l})$ where $N_{\pi_r}(W_{i,l})$ is upper bounded by the following equation (from Equation 10 of [9]).

$$N_{\pi_r}(W_{i,l}) = \sum_{\tau_u \in \Gamma'_r} \left\lceil \frac{W_{i,l}}{T_u} \right\rceil \quad (16)$$

Equation 16 is also derived on the basis of the number of jobs/memory phases that tasks running on a remote core π_r can issue during any time window of length $W_{i,l}$.

Finally, the maximum bus contention $Bus_{i,r}(W_{i,l})$ can be computed using different *cases* based on the values of $N_{\pi_l}(W_{i,l})$ and $N_{\pi_r}(W_{i,l})$. For example, Section 5.3 of [9] computes the bus contention using three cases:
1) $N_{\pi_l}(W_{i,l}) > N_{\pi_r}(W_{i,l})$; 2) $N_{\pi_l}(W_{i,l}) = N_{\pi_r}(W_{i,l})$; and 3) $N_{\pi_l}(W_{i,l}) < N_{\pi_r}(W_{i,l})$.

We will now briefly discuss how the analysis in [9] can be improved for case 1 and the same approach can be used for all the cases considered in [9].

If $N_{\pi_l}(W_{i,l}) > N_{\pi_r}(W_{i,l})$, the maximum bus contention that can be suffered by tasks executing on the local core due to tasks running on a remote core π_r during any time window of length $W_{i,l}$ is upper bounded by $Bus_{i,r}(W_{i,l})$, where (from Equation 11 of [9])

$$Bus_{i,r}(W_{i,l}) = \sum_{\tau_u \in \Gamma'_r} \left\lceil \frac{W_{i,l}}{T_u} \right\rceil \times (MD_u^A \times t^{mem} + MD_u^R \times t^{mem}) \quad (17)$$

We can see that the bound on bus contention given by Equation 17 is pessimistic since it considers the maximum number of memory requests issued during each memory phase in isolation. To address this pessimism, we can apply the exact cache analysis presented in Section IV-A to Equation 17. Using Lemma 4 proposed in Section IV-A2, we can improve and reformulate Equation 17 as follows.

$$\hat{Bus}_{i,r}(W_{i,l}) = \sum_{\tau_u \in \Gamma'_r} \min\Big(\left\lceil \frac{W_{i,l}}{T_u} \right\rceil \times MD_u^A \times t^{mem}, MD_u^A \times t^{mem} + (\left\lceil \frac{W_{i,l}}{T_u} \right\rceil - 1) \times (\bar{MD}_u^A + |\bar{\rho}_u|) \times t^{mem} \Big) + \left\lceil \frac{W_{i,l}}{T_u} \right\rceil \times MD_u^R \times t^{mem} \quad (18)$$

In a similar manner to that of Equation 18, we can improve all cases considered in [9]. For example, the bus contention for cases 2 and 3 in [9] is derived by forming sets M_r^A, M_r^R that contains the length of A- and R-phases in isolation. Consequently, applying the proposed persistence-aware cache analysis allows tightly bounding the length of each of the memory phases, thus, the maximum bus contention.

After bounding the maximum bus contention $\hat{Bus}_{i,r}(W_{i,l})$ that can be caused by each remote core $\pi_r \in m$ such that $\pi_r \neq \pi_l$, we can compute the *total bus contention* $Bus_{i,l}^{max}(W_{i,l})$ that tasks of local core π_l can suffer due to tasks of *all remote cores* during $W_{i,l}$ using Equation 14.

V. Worst Case Response Time Analysis

Having bounded the maximum number of memory requests using the analysis presented in Section IV, the bound on the maximum bus contention can be computed by improving the bus contention analysis presented in [8] and [9]. Now we can incorporate the resulting bound on the bus contention into the WCRT analysis of tasks. For this, we propose *improved WCRT* formulation that accounts for *cache reuse* when computing *bus contention* as well as the *maximum interference from higher priority tasks*. By applying cache persistence to higher priority tasks, we can tightly bound the number of memory accesses issued during their memory phases which in turn reduces the

length of their memory phases. Consequently, it can reduce the overall interference that can be caused by the memory phases of higher priority tasks that execute on the local core during the level-i busy window. Building on this, the length of the level-i busy window is given by the following lemma.

Lemma 6. *The length of the level-i busy window for a given task τ_i executing on core π_l is denoted by $W_{i,l}$, where $W_{i,l}$ is given by the first positive solution to the fixed-point iteration of the following equation*

$$W_{i,l} = (\alpha^A_{i,l}(W_{i,l}) + \alpha^R_{i,l}(W_{i,l})) \times t^{mem} + \max_{\tau_j \in lp_{i,l}} \{C_j\} + \sum_{\tau_h \in hep_{i,l}} \left\lceil \frac{W_{i,l}}{T_h} \right\rceil \times C^E_h + Bus^{max}_{i,l}(W_{i,l}) \quad (19)$$

Proof. From Equations 8 and 9, we know that $\alpha^A_{i,l}(W_{i,l})$ and $\alpha^R_{i,l}(W_{i,l})$ upper bounds the maximum number of memory requests that can be generated during the A- and R-phases of all tasks in $hep_{i,l}$ (including task τ_i) that execute on the local core π_l during any time window of length $W_{i,l}$. Assuming that each memory request will take t^{mem} time units, $(\alpha^A_{i,l}(W_{i,l})+\alpha^R_{i,l}(W_{i,l})) \times t^{mem}$ upper bounds the contribution of the memory phases of all tasks in $hep_{i,l}$ that execute on the local core π_l during $W_{i,l}$. Due to the fixed priority non-preemptive scheduling, at most one job of a task in $lp_{i,l}$ can cause blocking to τ_i. This blocking is maximized by considering a task with the largest WCET among all tasks in $lp_{i,l}$, given by $\max_{\tau_j \in lp_{i,l}} \{C_j\}$.[1] Similarly, the term $\sum_{\tau_h \in hep_{i,l}} \left\lceil \frac{W_{i,l}}{T_h} \right\rceil \times C^E_h$ upper bounds the contribution of the E-phases of all tasks in $hep_{i,l}$ (including task τ_i) that execute on the local core π_l during $W_{i,l}$. Finally, $Bus^{max}_{i,l}(W_{i,l})$ is the *total bus contention* that can be suffered by tasks that execute on the local core π_l from all remote cores during $W_{i,l}$ and can be computed using Equation 14 for the RR and FCFS bus arbitration policies. The Lemma follows. □

Note that $W_{i,l}$ appears on both sides of Equation 19 so it needs to be solved iteratively using $W_{i,l} = \sum_{\tau_h \in hep_{i,l}} C_h + \max_{\tau_j \in lp_{i,l}} \{C_j\}$ as the starting point.

Having bounded the length of the level-i busy window, we can compute the *response time* of the k^{th} job of τ_i on core π_l, i.e., $\tau_{i,k}$, using the following lemma.

Lemma 7. *The response time of $\tau_{i,k}$ is denoted by $R_{i,k}$, where $R_{i,k}$ is given by the first positive solution to the fixed-point iteration on the following equation:*

$$R_{i,k} = (\alpha^A_{i,l}(R_{i,k}) + \alpha^R_{i,l}(R_{i,k})) \times t^{mem} + \max_{\tau_j \in lp_{i,l}} \{C_j\} + \sum_{\tau_h \in hep_{i,l}} \left\lceil \frac{R_{i,k}}{T_h} \right\rceil \times C^E_h + Bus^{max}_{i,l}(R_{i,k}) \quad (20)$$

Proof. The proof directly follows from Lemma 6 except considering any time window of length $R_{i,k}$. □

[1]Note that a task in $lp_{i,l}$ with the largest A+R-phases was considered while deriving the number of memory requests to compute bus contention (see Lemma 3) so it is safe to consider $\max_{\tau_j \in lp_{i,l}} \{C_j\}$ for maximum blocking.

Finally, the WCRT of task τ_i is denoted by R^{max}_i and can be computed by maximizing Equation 20 over all jobs of τ_i that execute during the level-i busy window as follows.

$$R^{max}_i = \max_{k \in [1, K_i]} \{R_{i,k}\} \quad (21)$$

where $K_i = \left\lceil \frac{W_{i,l}}{T_i} \right\rceil$.

A taskset is only said to be schedulable if $R^{max}_i \leq D_i$ for each task $\tau_i \in \Gamma$ and the total bus utilization of the taskset is less than or equal to 1, i.e., $\sum_{\tau_i \in \Gamma} \frac{(MD^A_i + MD^R_i) \times t^{mem}}{T_i} \leq 1$.

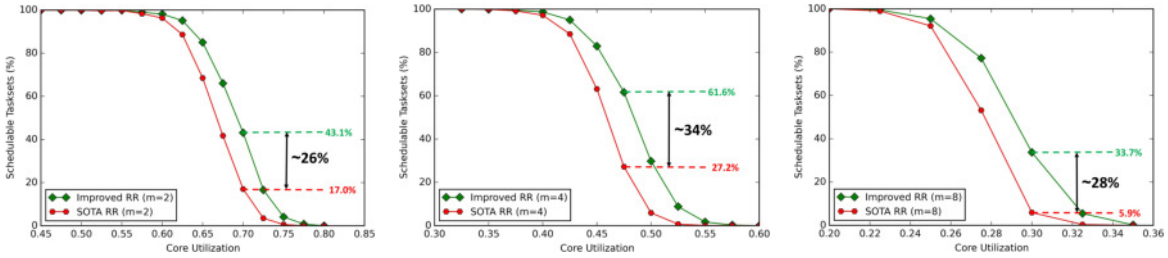
VI. Experimental Evaluation

In this section, we evaluate how much cache-aware bus contention analyses can improve the performance of the SOTA RR-based [8] and FCFS-based [9] bus contention analysis. For the RR analysis, we assume that the bus slot size is equal to t^{mem}. This is chosen due to the observation in [8] that the bus contention is least when the bus slot size is equal to t^{mem}. Similarly, for the FCFS bus policy, we only consider Fair Memory Access Model (FMAM) based bus contention analysis of [9] since it is the best performing FCFS-based analysis presented in [9] (see Section 8 of [9]).

For the default configuration, we model a quad-core platform with a direct-mapped unified LLC of 32KB (1024 cache sets, 32-byte block) evenly partitioned to the cores. By default, we assume that there were 32 tasks in each taskset with 8 tasks randomly assigned to each core. Tasks utilization U_i was generated using the UUnifast-discard algorithm [4]. Task periods T_i were randomly generated in the range of [1000-10000] using log-uniform distribution. The WCET in isolation C_i was then assigned by applying the relation $C_i = U_i \times T_i$. The total memory access demand (MD) of tasks was derived using C_i such that, $MD_i = rand(10\%, 40\%) \times C_i$. The length of the A-phase was chosen randomly in the range [60%-90%] of MD_i, i.e., $MD^A_i \times t^{mem} = rand(60\%, 90\%) \times MD_i$. The length of the R-phase was then given by $MD^R_i \times t^{mem} = MD_i - (MD^A_i \times t^{mem})$. Finally, the length of the E-phase was given by $C^E_i = C_i - (MD^A_i + MD^R_i) \times t^{mem}$. We assume that tasks are mapped to the cache partition sequentially and in priority order. The number of ECBs of tasks was generated using the length of A-phase, i.e., $|ECB_i| = \frac{MD^A_i}{t^{mem}}$. Similarly, the number of PCBs for each task was generated randomly in the range [20%-80%] of its ECBs. Task priorities were assigned using rate monotonic algorithm [15]. Task deadlines were equal to task periods, i.e., $D_i = T_i$.

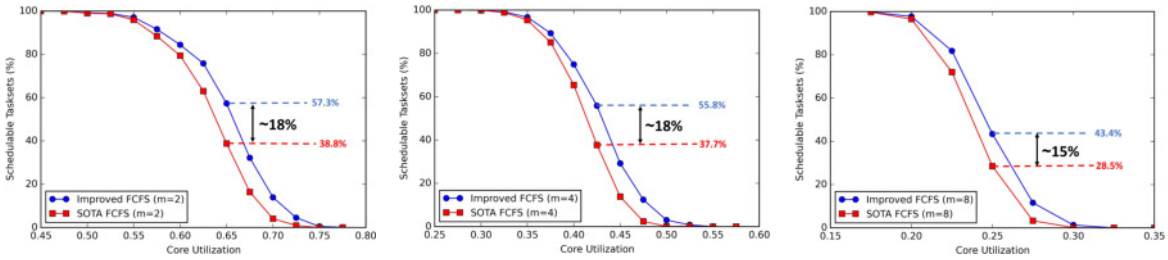
We compare the performance of the improved bus contention analysis with the existing RR [8] and FCFS policy-based bus contention analysis [9] by varying: 1) the core utilization; 2) the number of cores; 3) the memory access demand; and 4) the number of cache sets. We use taskset schedulability, i.e., the percentage of schedulable tasksets, as a metric to evaluate the performance of each approach.

In all figures, the improved bus contention analyses for the RR computed using Section IV-A and FCFS bus policy computed using IV-B are marked as "Improved RR" and "Improved FCFS", respectively. Similarly, the State-of-the-Art



(a) RR bus policy, m=2 (b) RR bus policy, m=4 (c) RR bus policy, m=8

Fig. 1: Varying Core Utilization and Number of Cores for the RR bus arbitration policy



(a) FCFS bus policy, m=2 (b) FCFS bus policy, m=4 (c) FCFS bus policy, m=8

Fig. 2: Varying Core Utilization and Number of Cores for the FCFS bus arbitration policy

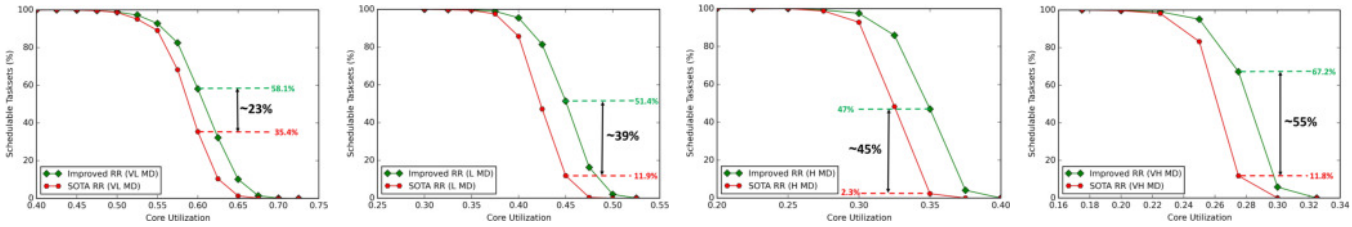
(SOTA) analysis for RR bus [8] is marked as "SOTA RR" and for FCFS bus [9] is marked as "SOTA FCFS". For all experiments, we randomly generated 1000 tasks per point. In all figures, the x-axis represents the core utilization and the y-axis represents the percentage of schedulable tasksets.

1) Varying Core Utilization: In this experiment, we varied the core utilization of each core under the default configuration, i.e., m=4, from 0.05 to 1 in steps of 0.025 and plotted the percentage of tasksets deemed schedulable by all the approaches. Figure 1b (resp. Figure 2b) shows the percentage of tasksets deemed schedulable using the improved analysis and SOTA analysis for the RR bus arbitration policy (resp. FCFS bus arbitration policy). For all approaches, we observe that increasing the core utilization decreases the taskset schedulability. This is because an increase in the core utilization also increases tasks utilization which in turn increases the WCET of tasks as $C_i = U_i \times T_i$. This increase in WCET also increases the number of memory requests, which in turn increases the bus contention suffered by tasks, resulting in a decrease in taskset schedulability. However, we note that the improved analyses for FCFS and RR bus policies outperform the SOTA analysis. For example, we can see in Figure 1b that at a core utilization of 0.475, the improved RR analysis was able to schedule up to 34% more tasksets than the SOTA RR analysis [8]. Similarly, we can see in Figure 2b that at a core utilization of 0.425, the improved FCFS analysis was able to schedule up to 18% more tasksets than the SOTA FCFS analysis [9]. These gains were observed because the improved bus contention analyses use a tighter bound on the number of LLC misses (computed using the analysis in Section IV) when computing bus requests and bus contention. On the contrary, the SOTA bus contention analyses [8], [9] are cache-oblivious policies as they always assume the worst-case number of LLC misses.

Interestingly, we observe that the gain of the improved analysis over the existing analysis was significant for the RR policy (see Figure 1b) but was relatively smaller for the FCFS bus arbitration policy (see Figure 2b). This is mainly because the FCFS bus contention analysis [9] mainly relies on the number of jobs/memory phases that can suffer/cause bus contention during a given time window. In such a case, applying the improved analysis to the FCFS bus may reduce the number of memory accesses per memory phase but it may not reduce the number of memory phases. On the other hand, the bus contention analysis for the RR policy [8] is fine-grained because it depends on the number of bus slots required by the local/remote core which further depends on the number of memory phases as well as the number of memory requests issued during each memory phase. Consequently, the improved analysis is more effective for the RR bus arbitration policy.

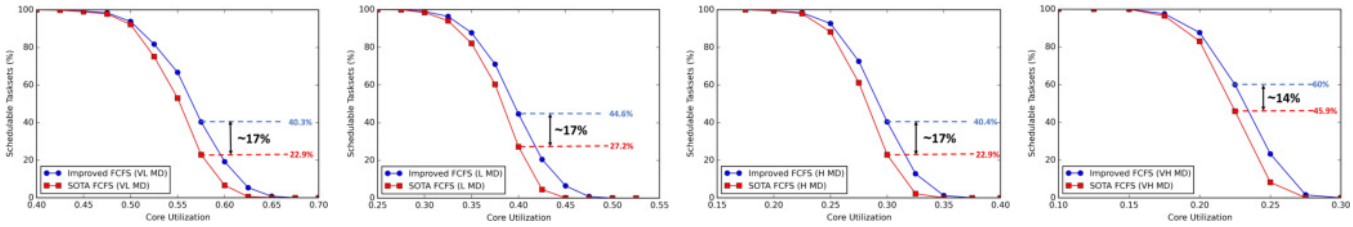
2) Varying Number of Cores: In this experiment, we varied the number of cores m from 2 to 8 and plotted the results for the RR and FCFS bus policies in Figures 1 and 2, respectively. As shown in Figures 1 and 2, for all the approaches, increasing the number of cores decreases the taskset schedulability. For instance, when $m = 8$, the number of remote cores as well as the total number of tasks in the taskset increases, i.e., 64 tasks. This results in increasing bus contention, which decreases taskset schedulability. Due to the same reason, decreasing the number of cores improves the performance of all the approaches. Nonetheless, the improved analyses outperform the SOTA bus contention analyses for the FCFS and RR bus policies for all the values of m.

3) Varying Memory Access Demand (MD): In this experiment, we varied the Memory Access Demand (MD) of all tasks in the taskset. For this, we consider four configurations: (a) Very Low (VL) MD, i.e., $MD_i = rand(5\%, 20\%) \times C_i$;



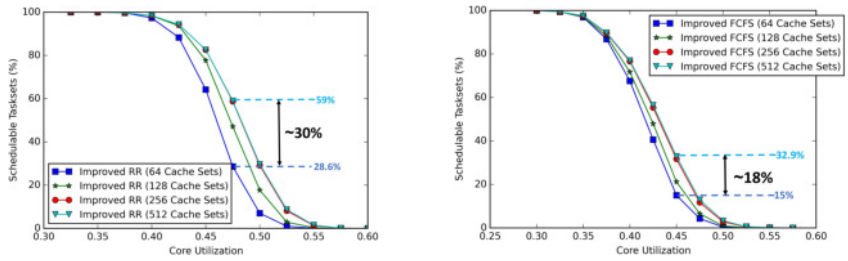
(a) RR bus policy, VL MD (b) RR bus policy, L MD (c) RR bus policy, H MD (d) RR bus policy, VH MD

Fig. 3: Varying Memory Demand (MD) for the RR bus arbitration policy



(a) FCFS bus policy, VL MD (b) FCFS bus policy, L MD (c) FCFS bus policy, H MD (d) FCFS bus policy, VH MD

Fig. 4: Varying Memory Demand (MD) for the FCFS bus arbitration policy



(a) Varying Cache Sets for RR bus policy (b) Varying Cache Sets for FCFS bus policy

Fig. 5: Varying Number of Cache Sets

(b) Low (L) MD, i.e., $MD_i = rand(20\%, 40\%) \times C_i$; (c) High (H) MD, i.e., $MD_i = rand(40\%, 60\%) \times C_i$; and (d) Very High (VH) MD, i.e., $MD_i = rand(60\%, 80\%) \times C_i$.

The resulting percentage of schedulable taskset for the RR and FCFS bus policies are plotted in Figures 3 and 4, respectively. From Figures 3 and 4, we can observe that for all approaches, the MD of tasks can significantly impact the taskset schedulability. Specifically, all approaches perform the best under the VL MD configuration and the worst under the VH MD configuration. This happens because increasing the MD value can increase the number of memory requests which in turn increases the bus contention and decreases the taskset schedulability. However, for all the MD configurations, the improved bus contention analyses dominate the existing bus contention analyses. In fact, the improved RR analysis was able to schedule up to 55% more tasksets than the SOTA RR analysis at the core utilization value of 0.275 under VH configuration as shown in Figure 3d.

4) Varying Cache Size: In the default configuration, we assume that the total number of sets in the cache are 1024 and 256 cache sets are allocated per-core. In this experiment, we consider different per-core cache set sizes, i.e., 64, 128, 256, 512, and plot the resulting taskset schedulability in Figure 5a for the RR bus policy and in Figure 5b for the FCFS bus policy. Note that increasing the number of cache sets allocated per-core will effectively increase the total size of the cache. For this experiment, we do not show the taskset schedulability for the existing bus contention analyses as their schedulability does not depend on the cache size.

We can see in Figures 5a and 5b that decreasing the per-core cache sets to 64 or 128 also decreases the schedulability for both the bus arbitration policies. Intuitively, this happens because for a smaller number of cache sets per-core, i.e., 64, 128, the overlap between the PCBs of a task with ECBs of other tasks increases. This leads to a higher CPRO which results in increasing the number of memory requests as well as bus contention. On the contrary, increasing the per-core cache sets to 256 or 512 also increases the schedulability for all approaches. This happens because increasing the number of cache sets allocated per-core reduces the overlap between the PCBs of tasks with ECBs of other tasks, thereby reducing CPRO. This results in improving taskset schedulability.

Interestingly, we observe that the difference between the taskset schedulability for cache set sizes of 256 and 512 is negligible. This happens because, in the default configuration, the value of MD is low, i.e., $MD_i = rand(10\%, 40\%) \times C_i$, which is further divided among A- and R-phases, with the length of A-phases used to generate ECBs of tasks. This implies that a per-core cache set size of 256 is sufficient enough to ensure that the PCBs of tasks do not overlap with the ECBs of other tasks, i.e., tasks do not suffer CPRO. Consequently, a further increase in the per-core cache set sizes

does not significantly impact taskset schedulability.

VII. Related Work

Several works have focussed on the problem of memory bus contention in multicore systems considering both the generic task model as well as the 3-phase task model (see survey [17]). For the generic task model, earlier works have analyzed bus contention by considering TDMA-based bus arbitration policies [10], [13], [24]. Dasari et al. [6] proposed an analysis that computes bus contention of tasks considering an unspecified work-conserving bus arbiter and integrates the resulting bounds on bus contention into the WCRT analysis. The work in [6] is extended in [2] to consider a wide range of bus arbitration policies. Davis et al. [11] have proposed the multicore response time framework that computes the WCRT of tasks for different cache configurations and memory bus arbitration policies. Similarly, for the 3-phase tasks, Maia et al. [16] have proposed the bus contention analysis for the 3-phase task model by considering fixed priority global scheduling. Arora et al. have proposed the bus contention analysis for 3-phase tasks considering partitioned fixed-priority scheduling using the RR [8] and FCFS bus arbitration policy [9].

Although all the above-mentioned works provide important solutions to the bus contention problem, most of these works focus only on the memory bus and ignore the interdependence between the cache memory and the memory bus. To the best of our knowledge, the first work that focused on the interdependence between the number of bus/memory requests of tasks and the cache memory was presented by Rashid et al. [22]. In that work, the authors used the notion of cache persistence (introduced in [20]) to produce tighter bounds on the number of cache misses of tasks and integrated them into the memory bus contention analysis. It was shown in [22] that a tighter bound on the bus contention can be obtained by analyzing and integrating the actual number of LLC misses of tasks into the bus contention analysis rather than assuming the worst-case number of LLC misses of tasks derived in isolation. However, the work in [22] considers a generic task model, and hence the bus contention analysis developed in [22] may not be used for the 3-phase task model, which is the focus of this work. Another recent work presented in [23] has focused on the cache-aware schedulability analysis of PREM tasks. However, the work in [23] only focuses on the cache analysis and does not focus on the bus contention suffered by tasks.

VIII. Conclusion

In this work, we presented improved bus contention analysis for 3-phase tasks. First, we present an analysis to upper bound the number of cache misses of tasks that lead to bus/memory requests. We then use that bound to improve bus contention that can be suffered by tasks under different bus policies and integrate it into a WCRT-based schedulability analysis. Experimental evaluation performed using synthetic tasksets under different settings show that improved bus contention analyses can improve taskset schedulability by up to 55%. In future works, we plan to extend our approach to heterogeneous platforms.

Acknowledgments. This work was supported by the CISTER Research Unit (UIDP/UIDB/04234/2020), financed by National Funds through FCT/MCTES (Portuguese Foundation for Science and Technology); by project ADACORSA (ECSEL/0010/2019 - JU grant nr. 876019) financed through National Funds from FCT and European funds through the EU ECSEL JU. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Austria, Sweden, Spain, Italy, France, Portugal, Ireland, Finland, Slovenia, Poland, Netherlands, 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. This work is also a result of the work developed under project Aero.Next Portugal (nº C645727867-00000066) and FLY-PT (grant nº 46079, POCI-01-0247-FEDER-046079), also funded by FCT under PhD grant 2020.09532.BD.

References

[1] J. Arora et al. Bus-contention aware schedulability analysis for the 3-phase task model with partitioned scheduling. In *29th RTNS*, 2021.
[2] D. Dasari et al. A framework for memory contention analysis in multi-core platforms. *Real-Time Systems*, 52, 06 2015.
[3] G. Durrieu et al. Predictable Flight Management System Implementation on a Multicore Processor. In *ERTS'14*, TOULOUSE, France, 2014.
[4] P. Emberson et al. Techniques for the synthesis of multiprocessor tasksets. *WATERS'10*, 01 2010.
[5] D. Casini et al. A holistic memory contention analysis for parallel real-time tasks under partitioned scheduling. In *RTAS*, pages 239–252, 2020.
[6] D. Dasari et al. Response time analysis of cots-based multicores considering the contention on the shared memory bus. In *ICESS*, 2011.
[7] G. Schwäricke et al. Fixed-Priority Memory-Centric Scheduler for COTS-Based Multiprocessors. In *ECRTS 2020*, LIPIcs, 2020.
[8] J. Arora et al. Bus-contention aware wcrt analysis for the 3-phase task model considering a work-conserving bus arbitration scheme. *Journal of Systems Architecture*, 122:102345, 2022.
[9] J. Arora et al. Schedulability analysis for 3-phase tasks with partitioned fixed-priority scheduling. *Journal of Systems Architecture*, 131, 2022.
[10] J. Rosen et al. Bus access optimization for predictable implementation of real-time applications on multiprocessor systems-on-chip. In *RTSS 2007*, pages 49–60, 2007.
[11] R. I. Davis et al. An extensible framework for multicore response time analysis. *Real-Time Systems*, July 2017.
[12] 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.
[13] T. Kelter et al. Bus-aware multicore wcet analysis through tdma offset bounds. In *2011 ECRTS*, pages 3–12, 2011.
[14] J. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. *[1990] 11th RTSS*, pages 201–209, 1990.
[15] C. L. Liu et al. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, January 1973.
[16] C. Maia et al. Schedulability analysis for global fixed-priority scheduling of the 3-phase task model. In *IEEE RTCSA*, Hsinchu, Taiwan, 2017.
[17] C. Maiza et al. A Survey of Timing Verification Techniques for Multi-Core Real-Time Systems. *ACM Computing Surveys*, 52(3):1–38, 2019.
[18] C. Pagetti et al. Automated generation of time-predictable executables on multi-core. In *RTNS 2018*, POITIERS, France, October 2018.
[19] R. Pellizzoni et al. A Predictable Execution Model for COTS-Based Embedded Systems. In *RTAS*, pages 269–279, USA, April 2011. IEEE.
[20] S. A. Rashid et al. Cache-persistence-aware response-time analysis for fixed-priority preemptive systems. In *28th ECRTS*, pages 262–272, 2016.
[21] S. A. Rashid et al. Bounding cache persistence reload overheads for set-associative caches. In *IEEE 26th RTCSA*, pages 1–10, 2020.
[22] S. A. Rashid et al. Cache persistence-aware memory bus contention analysis for multicore systems. In *DATE*, pages 442–447, 2020.
[23] S. A. Rashid et al. Cache-aware schedulability analysis of prem compliant tasks. In *DATE*, pages 1269–1274. IEEE, 2022.
[24] A. Schranzhofer et al. Timing analysis for tdma arbitration in resource sharing systems. In *2010 16th IEEE RTAS*, pages 215–224, 2010.
[25] T. Thilakasiri et al. An exact schedulability analysis for global fixed-priority scheduling of the aer task model. In *28th ASP-DAC*, pages 326–332, 2023.
[26] H. Tomiyama et al. Program path analysis to bound cache-related preemption delay in preemptive real-time systems. In *8th CODES2000 (IEEE Cat. No. 00TH8518)*, pages 67–71. IEEE, 2000.
[27] R. Wilhelm et al. The worst-case execution-time problem—overview of methods and survey of tools. *ACM TECS*, 7(3):1–53, April 2008.