Bus-Contention Aware WCRT Analysis for the 3-Phase Task Model Considering a Work-Conserving Bus Arbitration Scheme

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

*CISTER Research Centre

CISTER-TR-211004

2021/12/13

Bus-Contention Aware WCRT Analysis for the 3-Phase Task Model Considering a Work-Conserving Bus Arbitration Scheme

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

https://www.cister-labs.pt

Abstract

Today multicore processors are used in most modern systems that require computational logic. However, their applicability in systems with stringent timing requirements is still an ongoing research. This is due to the difficulty of ensuring the timing correctness of tasks executing on a multicore platform that comprises a number of shared hardware resources, e.g., caches, memory bus and the main memory. Concurrent accesses to any of these shared resources can generate uncontrolled interference, which complicates the estimations of tasks' worst-case execution time (WCET) and the worst-case response time (WCRT).The use of the 3-phase task execution model helps in upper bounding the contention due to the sharing of bus/main memory in multicore systems. It divides the execution of tasks into distinct memory and execution phases, where tasks can only access the bus/main memory during their memory phases. This makes bus/memory access patterns of tasks more predictable, enabling a preciser computation of bus/memory contention.In this work, we show how the bus contention can be computed for the 3-phase task model considering a work-conserving, i.e., round-robin (RR) based, arbitration policy at the memory bus. This is different from existing works that analyze the time-division multiple access (TDMA) and first-come-first-serve (FCFS) based bus arbitration policies. First, we present a solution to model the bus contention that can be suffered/caused by tasks executing on the same/remote cores of a multicore system under an RR-based bus arbitration scheme. We then evaluate the impact of resulting bus contention on taskset schedulability. Experimental results show that our proposed RR-based bus contention analysis can improve taskset schedulability by up to 100 percentage points than the TDMA-based analysis and up to 40 percentage points than the FCFS-based bus contention analysis.

Bus-Contention Aware WCRT Analysis for the 3-Phase Task Model Considering a Work-Conserving Bus Arbitration Scheme

Jatin Arora[a,*], Cláudio Maia[a], Syed Aftab Rashid[a,c], Geoffrey Nelissen[b], Eduardo Tovar[a]

[a]*CISTER Research Centre, ISEP, IPP, Porto, Portugal*
[b]*Eindhoven University of Technology, Eindhoven, the Netherlands*
[c]*VORTEX CoLab, Porto, Portugal*

Abstract

Nowadays multicore processors are used in most modern systems. However, their applicability in systems with stringent timing requirements is still ongoing research. The main reason behind this is the difficulty of ensuring the timing correctness of tasks executing on such systems as they comprise a number of shared hardware resources, as for instance, caches, memory bus, and the main memory. Concurrent accesses to any of these shared resources can generate uncontrolled interference, which complicates the estimations of the worst-case execution time and the worst-case response time of tasks.

The use of the 3-phase task execution model helps in upper bounding the contention due to the sharing of bus/main memory in multicore systems. This model divides the execution time of tasks into distinct memory and execution phases, where tasks can only access the bus/main memory during their memory phases. This makes bus and memory access patterns of tasks predictable by enabling a precise computation of bus/memory contention.

In this work, we show how the bus contention can be computed for the 3-phase task model considering a round-robin-based arbitration policy at the memory bus. We differ from existing works that analyze the time-division multiple access and first-come-first-serve-based bus arbitration policies. First, we present a solution that models the bus contention that can be suffered/caused by tasks executing on the same/remote cores of a multicore system under an RR-based bus arbitration scheme. Then, the impact of the resulting bus contention on taskset schedulability is evaluated.

Experimental results show that our proposed RR-based bus contention analysis can improve taskset schedulability by up to 100 percentage points when compared to a state-of-the-art TDMA-based analysis, and up to 40 percentage points when compared to the state-of-the-art FCFS-based bus contention analysis.

Keywords: Real-Time Systems, Multicore Processors, Partitioned Scheduling, Phased Execution Model, Bus Contention, Schedulability Analysis

1. Introduction

Multicore processors were introduced to overcome the limitations of single-core processors, e.g., computing power, energy efficiency, and heat dissipation. Most modern systems (e.g., mobile phones, computers, digital cameras, etc.) use Commercial-Off-The-Shelf (COTS) multicore processors to take benefit of the above-mentioned features. However, the adoption of COTS multicore processors in systems that run applications with *stringent* timing requirements is still a work-in-progress. The main reason behind this is the *non-deterministic* behavior of COTS multicore processors. This non-determinism exists due to the sharing of different hardware resources among the different tasks. A typical example of a hardware component that

*Corresponding author
Email addresses: `jatin@isep.com` (Jatin Arora), `crrm@isep.com` (Cláudio Maia), `syara@isep.com` (Syed Aftab Rashid), `g.r.r.j.p.nelissen@tue.nl` (Geoffrey Nelissen), `emt@isep.com` (Eduardo Tovar)

is shared in a COTS multicore processor is the system bus which is used to fetch task's data/instructions from the main memory.

Due to this sharing, when a task running on a given core requests access to the bus to fetch its data/instructions from the main memory, it may suffer *bus contention* if the bus is already busy serving the requests of another task executing on some other core of the same multicore platform. It has been shown by several works [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12] that the bus contention suffered by tasks executing on a multicore platform can have a significant impact on the Worst-Case Execution Time (WCET) and the Worst-Case Response Time (WCRT) of tasks.

The use of *phased* execution models [13, 14, 15] is one approach that addresses this problem. Under a phased execution model, the execution of a task is divided into distinct memory and execution phases. This ensures that tasks can issue memory requests only during their memory phase(s), which facilitates the analysis of the bus contention that can be suffered by tasks. Under the umbrella of phased-execution models, the *3-phase* execution model has received much attention from industry and academia [14, 15, 8, 16, 10, 17, 18, 19, 12, 20, 21, 22, 23]. In the 3-phase task execution model, the execution of a task is divided into three phases *Acquisition* (*A-phase*), *Execution* (*E-phase*), and *Restitution* (*R-phase*). When a task is released, it first executes the A-phase to fetch all its data/instructions from the main memory to the core's local memory via the system bus. It then executes the E-phase using the data/instructions already available in the core's local memory without generating any bus/memory requests. Finally, in the R-phase the task write-back the modified data to the main memory using the system bus. Thus, the A- and R-phase are memory phases, i.e., time intervals in which accesses to bus/main memory are allowed, and the E-phase is a computation phase, i.e., no bus/memory access occurs in this phase. As the memory accesses can only happen during the memory phases, the complexity of analyzing the WCET/WCRT of applications running on COTS multicore processors can be reduced by using the 3-phase task model. However, tasks can still suffer bus contention. For instance, a memory phase of a given task τ_i executing on one core of a multicore platform can be delayed due to the bus contention caused by the memory phases of tasks executing on other cores of the same multicore platform.

Few existing works [10, 8, 12] have focused on analyzing the bus contention that can be suffered by tasks executed using the 3-phase execution model. Maia et al. [10] have presented a bus contention aware WCRT analysis for the 3-phase tasks scheduled using global fixed-priority non-preemptive scheduling. Schranzhofer et al. [8] have proposed a bus contention aware WCRT analysis for partitioned fixed-priority non-preemptive scheduling assuming a Time-Division Multiple Access (TDMA) based bus arbiter. Similarly, Arora et al. [12] have presented a schedulability analysis for the 3-phase tasks that are scheduled using partitioned fixed-priority non-preemptive scheduling and the bus arbitration policy is assumed to be First-Come-First-Serve (FCFS). The arbitration policy considered at the bus can have a strong impact on the bus contention that can be suffered by the tasks. The TDMA [8] based bus contention analysis can be pessimistic due to the non-work-conserving nature of TDMA bus arbitration and hence it may not fully utilize the benefits of the 3-phase execution model. Similarly, FCFS-based bus contention analysis [12] can also be pessimistic as it does not uniformly allocate the bus among all the cores. On the contrary, a fairer work-conserving bus arbitration policy like Round-Robin (RR) can uniformly distribute the bus among cores, resulting in reducing the bus contention that can be suffered by the tasks.

Furthermore, most existing works in the state-of-the-art [12] that focus on bounding bus contention considering partitioned fixed-priority non-preemptive scheduling of 3-phase tasks may underestimate the blocking that can be caused by the lower-priority tasks running on the same core. This is mainly because most of these works assume that under partitioned fixed-priority non-preemptive scheduling, any task τ_i will always suffer blocking from a lower priority task with the largest WCET. This assumption is valid only for single-core systems since on a multicore system a task with the largest execution time may not always suffer the maximum bus contention. In this work, we address these issues by making the following **contributions**:

1. We present an approach to bound the maximum bus contention that can be suffered by 3-phase tasks executing on multicore architectures that use partitioned fixed-priority scheduling and Round-Robin (RR) bus arbitration policy.
2. We show that when computing the WCRT of a task τ_i under the 3-phase execution model scheduled

using partitioned fixed-priority non-preemptive scheduling on a multicore system, each task τ_j having a lower priority than τ_i can block the execution of τ_i and each τ_j may contribute differently to the total bus contention. Therefore, we propose an algorithm to accurately quantify the contribution of every lower priority task τ_j in order to maximize the delay that can be suffered by a task τ_i during its response time interval. We then integrate the resulting bounds on the bus contention and blocking from lower-priority tasks into the schedulability analysis of partitioned fixed-priority non-preemptive systems.

3. We perform an extensive experimental evaluation under different settings to show the effectiveness of the proposed analysis in comparison to the state-of-the-art analysis presented in [8, 12]. Experimental results show that our proposed RR-based bus contention analysis improves taskset schedulability by up to 100 percentage points than the TDMA-based analysis [8] and up to 40 percentage points than the FCFS-based bus contention analysis [12].

2. System Model

We assume a multicore platform that is composed of m identical cores $(\pi_1, \pi_2, \ldots, \pi_m)$. Each core has a local memory (e.g., scratchpad or local cache) that is sufficiently large to store the data/code required by the task with the largest memory footprint running on that core. This is to ensure that tasks suffer no self-evictions, i.e., all memory blocks used by a task during its execution fit in the core's local memory, without overlap. It is assumed that the *system bus* is shared among all the cores and connects all the cores to the main memory (e.g. DRAM). Similar to existing works [8, 5, 6, 7, 10, 9, 12], we assume a single-channel system bus, i.e., the system bus can only handle one memory request at a time. As the proposed work mainly focus on the main memory accesses, we assume that a shared cache is not present in the system or disabled if present.

2.1. Task Model and Useful Notations

We consider a task set Γ comprising n sporadic tasks from which a subset Γ' is assigned to each core according to the given task-to-core mapping strategy. Each task τ_i is characterized by a triplet (C_i, T_i, D_i) where C_i is the total Worst-Case Execution Time (WCET) of τ_i *in isolation*, T_i is its minimum inter-arrival time, and D_i is its relative deadline. We assume that tasks are partitioned among cores at design time and are scheduled using a partitioned fixed task priority algorithm such as rate monotonic or deadline monotonic [24]. Additionally, we do not make any assumption on unique task priorities[1].

Each task τ_i is executed as per the 3-phase task model, i.e., the execution of each task is divided into Acquisition (A), Execution (E), and Restitution (R) phases. The WCET of A, E and R-phases of task τ_i when it executes in isolation is given by C_i^A, C_i^E, C_i^R, respectively, and the total WCET of task τ_i in *isolation* is given by $C_i = C_i^A + C_i^E + C_i^R$. We assume that tasks have constrained deadlines, i.e., $D_i \leq T_i$, and are independent, i.e., tasks do not share data among themselves. Each task τ_i has a utilization $U_i = \frac{C_i}{T_i}$ and the *utilization* of taskset $\Gamma_l^{'}$ assigned to core π_l is given by $\sum_{\tau_i \in \Gamma_l^{'}} U_i$. The *total utilization* of taskset Γ is given by $\sum_{\tau_i \in \Gamma} U_i$. Similarly, the bus utilization of task τ_i is given by $\frac{C_i^A + C_i^R}{T_i}$ and the *total bus utilization* of taskset Γ is given by $\sum_{\tau_i \in \Gamma} \frac{C_i^A + C_i^R}{T_i}$. The response time of the k^{th} job of task τ_i executing on a given core π_l is denoted by $R_{i,k,l}$ and the Worst-Case Response Time (WCRT) of task τ_i, denoted by $R_{i,l}^{max}$, is given by maximizing $R_{i,k,l}$ over all jobs of τ_i. The taskset Γ is deemed unschedulable if the utilization of a task set $\Gamma_l^{'}$ assigned to a core π_l is greater than 1. Similarly, the taskset is also deemed unschedulable if the total bus utilization is greater than 1 since the system bus saturates and the communication delay with the memory cannot be bounded anymore.

For notational convenience, we define the following set of tasks: $hep_{i,l}$ denote the set of tasks with higher or equal priority than τ_i (including τ_i) assigned to core π_l and $hp_{i,l}$ (resp. $lp_{i,l}$) denote the set of tasks with higher (resp. lower) priority than τ_i assigned to core π_l.

[1]It implies that multiple tasks running on a given core can have same priority.

2.2. System Bus Model

We assume a single-channel system bus that can handle only one request at a time. A memory phase is composed of one or multiple memory requests. A memory request is said to be initiated when the requesting memory phase gains access to the bus and starts executing. A memory request is said to be completed when the system bus completes the required read/write operation from the main memory. The system bus handles each memory request non-preemptively and remains busy while handling a request. Similarly to the existing works [8, 5, 25, 26, 27], we assume that each memory request will be served in MT time units, i.e., the maximum time required to perform a single read/write operation from/to the main memory. The maximum number of memory requests that can be issued during the A-phase (resp. R-phase) of a task τ_i when it executes in isolation is given by MR_i^A (resp. MR_i^R). We assume that the values of MR_i^A, MR_i^R and MT can be derived using any static WCET analysis tools or by using any measurement-based techniques [28].

We assume that the bus arbitration policy is Round-Robin (RR). In the RR bus arbitration policy, a core can access the bus only during the *bus access slot* assigned to that core. In this paper, we use the term *bus slot* or *slot* to refer to the bus access slot. The bus slot for a given core is said to be *active* only when that core starts performing memory requests during its assigned bus slot. The maximum number of memory requests that a core can perform during its assigned bus slot depends on the slot size SS. We assume that SS is an integer multiple of MT such that at least one memory request can be performed during a single bus access slot. It is assumed that the value of SS is the same for each core. Due to the work-conserving nature of the RR bus arbitration policy, if a core does not have any pending memory requests, it will not use its assigned slot and the system can then grant the slot to the next core waiting for the bus.

2.3. Execution Model

In the 3-phase task model, when a task is released, it first executes the A-phase by fetching all its data/instructions from the main memory to the core's local memory via the system bus. It then executes the E-phase using the data/instructions already available in the core's local memory without generating any bus/memory request. Finally, in the R-phase, the task write-back the modified data to the main memory using the system bus. This categorizes the A- and R-phase into memory phases, i.e., time intervals in which accesses to bus/main memory are allowed, and the E-phase into a computation phase, i.e., no bus/memory access can be generated in this phase. Each task executes non-preemptively, i.e., once a task starts executing its A-phase, it cannot be preempted by any other task running on the same core until the completion of its R-phase. A memory phase can only request access to the bus when it is ready to execute. For instance, the A-phase of a task τ_i is ready to execute when task τ_i is released whereas its R-phase becomes ready to execute after the completion of its E-phase. The task is said to be completed when the execution of its R-phase terminates.

Whenever a task is ready to execute on a given core, the core requests access to the system bus in order to execute the A-phase of the ready task. If the bus is free, the slot assigned to the core become active immediately and the A-phase starts executing. However, if the bus is busy serving the memory requests of co-running tasks, the requesting core has to wait for its turn i.e., for the completion of bus slots assigned to the other cores. Once the slot for the core becomes active, the core starts executing the A-phase. If all the memory requests of an A-phase cannot be served in one bus slot, the core waits for its next active slot to execute the pending memory requests of the same A-phase. Once the *A-phase is completed, the core releases the bus, even if there is time available in the bus slot,* in order to execute the E-phase of the same task. Once the task completes the execution of its E-phase, the core waits for its slot to execute the R-phase of the same task. Once the *R-phase is completed, the core releases the bus even if the slot is not fully utilized by the core.*

3. Background and Problem Formulation

In this section, we introduce a number of key concepts and summarize an existing schedulability analysis for the 3-phase task model when using partitioned fixed-priority non-preemptive scheduling. Later, we use this analysis to build our proposed analysis.

It is proved in [29] that for a task τ_i executing on a single-core platform under Fixed-Priority Non-Preemptive (FPNP) scheduling, the WCRT is observed in the longest level-i busy window. The level-i busy window w.r.t a task τ_i is defined as follows.

Definition 3.1. [Level-i busy window (from [30])] 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.

Let W_i be the length of the level-i busy window that can be computed by first bounding the

- *Maximum workload* that can be generated by all jobs released by all tasks in hep_i (including task τ_i) during the level-i busy window, and
- *Maximum blocking* that can be caused by one job of a task in lp_i, that may start its execution before the arrival of τ_i.

Consequently, the length of the longest level-i busy window W_i is given by the first positive fixed-point solution to the following iterative equation.

$$W_i = C_{lp_i}^{max} + \sum_{\tau_h \in hep_i} (\eta_h^+(W_i) \times C_h) \tag{1}$$

where $C_{lp_i}^{max}$ is the maximum blocking that can be caused by one job of a lower-priority task in lp_i, i.e., $C_{lp_i}^{max} = \max_{\forall \tau_j \in lp_i} \{C_j\}$. The term $\eta_h^+(W_i)$ is the upper event arrival function [1] that captures the maximum number of jobs released by a task τ_h in any time interval of length W_i. Consequently, the maximum workload that can be generated by all tasks in hep_i in any time interval of length W_i is given by $\sum_{\tau_h \in hep_i} (\eta_h^+(W_i) \times C_h)$; where C_h is the WCET of a task $\tau_h \in hep_i$ computed in isolation.

Unlike single-core processors, where only one task can execute at a time, multicore processors allow parallel task executions on different cores. These concurrently executing tasks may suffer additional execution delays due to the bus contention as they use the same shared system bus to fetch data/code from the main memory. Therefore, for a task τ_i executing on a multicore platform, the length of the longest level-i busy window does not only depend on the workload generated by the tasks that execute on the same core as τ_i, but, also on the bus contention that can be suffered by all the jobs running on the core under analysis during the busy window. To address this issue, the work in [12] proposed a bus-contention aware schedulability analysis for the 3-phase task model considering partitioned fixed-priority non-preemptive scheduling.

Under the analysis presented in [12], the length of the longest level-i busy window w.r.t a task τ_i executing on a core π_l of multicore platform is given by $W_{i,l}$, where

$$W_{i,l} = C_{lp_{i,l}}^{max} + Bus_{i,l}^{max}(W_{i,l}) + \sum_{\tau_h \in hep_{i,l}} (\eta_h^+(W_{i,l}) \times C_h) \tag{2}$$

where $\sum_{\tau_h \in hep_{i,l}} (\eta_h^+(W_{i,l}) \times C_h)$ is the maximum workload that can be generated by all tasks in $hep_{i,l}$ (including τ_i) in any time interval of length $W_{i,l}$. Similarly, the term $C_{lp_{i,l}}^{max}$ captures the maximum blocking that can be caused by one job of a task in $lp_{i,l}$ at the start of level-i busy window $W_{i,l}$ on core π_l. The analysis in [12] computes both $\sum_{\tau_h \in hep_{i,l}} (\eta_h^+(W_{i,l}) \times C_h)$ and $C_{lp_{i,l}}^{max}$ in a similar manner as in Equation 1.

In Equation 2, $Bus_{i,l}^{max}(W_{i,l})$ denotes the total bus contention that can be suffered by the tasks executing on the local core π_l (i.e., core on which task under analysis is running) during $W_{i,l}$, from all other tasks that may execute on all other remote cores (i.e., all cores except the local core) during $W_{i,l}$. The bus contention analysis in [12] computes the value of $Bus_{i,l}^{max}(W_{i,l})$ by considering the maximum number of memory phases that can be executed on the local/remote cores during $W_{i,l}$. It then uses the number of memory phases of local/remote cores to identify different possible execution scenarios, i.e., denoted as "cases" in [12], and the resulting bus contention for each case. The total bus contention is then computed by assuming a first-come first-serve (FCFS) bus arbitration policy (see Section 4.2 and 4.3 of [12] for details).

Having determined the length of the longest level-i busy window $W_{i,l}$ using Equation 2, the maximum number of jobs of task τ_i that can execute during $W_{i,l}$ are computed as follows [12]

$$K_i = \eta_i^+(W_{i,l}) \tag{3}$$

The WCRT of a task τ_i executing on core π_l is then computed by first computing the response time of each job of task τ_i that executes on core π_l during $W_{i,l}$.

For the 3-phase tasks scheduled using non-preemptive scheduling, the response time of the k^{th} job of a task τ_i on core π_l, i.e., $\tau_{i,k,l}$, is computed by first evaluating the latest start time of the R-phase of $\tau_{i,k,l}$.

The latest start time of the R-phase of $\tau_{i,k,l}$ is denoted by $s_{i,k,l}^R$ and is given by the first positive solution to the fixed-point iteration on the following equation:

$$s_{i,k,l}^R = C_{lp_{i,l}}^{max} + \sum_{\tau_h \in hep_{i,l} \setminus \tau_i} \eta_h^+(s_{i,k,l}^R - (C_i^A + C_i^E)) \times C_h + Bus_{i,l}^{max}(s_{i,k,l}^R) + ((k-1) \times C_i) + (C_i^A + C_i^E) \tag{4}$$

where $C_{lp_{i,l}}^{max}$ is the same as in Equation 2.

Due to the non-preemptive execution of tasks, a task $\tau_h \in hep_{i,l}$ can only impact $\tau_{i,k,l}$ until the start of the A-phase of $\tau_{i,k,l}$. Thus, the term $\eta_h^+(s_{i,k,l}^R - (C_i^A + C_i^E)) \times C_h$ captures the maximum workload generated by a task $\tau_h \in hep_{i,l}$ until the start of the A-phase of $\tau_{i,k,l}$. Effectively, the maximum workload generated by all the tasks in $hep_{i,l}$ (excluding τ_i) until the start of the A-phase of $\tau_{i,k,l}$ is captured using $\sum_{\tau_h \in hep_{i,l} \setminus \tau_i} \eta_h^+(s_{i,k,l}^R - (C_i^A + C_i^E)) \times C_h$. The term $(k-1) \times C_i$ represents the contribution of previous jobs of task τ_i that can delay the start time of the R-phase of $\tau_{i,k,l}$. The term $(C_i^A + C_i^E)$ accounts for the WCET of the A-phase and the E-phase of τ_i while computing the starting time of R-phase of $\tau_{i,k,l}$. The total bus contention that can be suffered by core π_l from all remote cores during any time interval of length $s_{i,k,l}^R$ is computed by $Bus_{i,l}^{max}(s_{i,k,l}^R)$.

Using $s_{i,k,l}^R$, the response time $R_{i,k,l}$ of $\tau_{i,k,l}$ can finally be computed as:

$$R_{i,k,l} = s_{i,k,l}^R + C_i^R \tag{5}$$

The WCRT of task τ_i is then given by the largest response time of any job of τ_i that executes during the level-i busy window $W_{i,l}$. Hence,

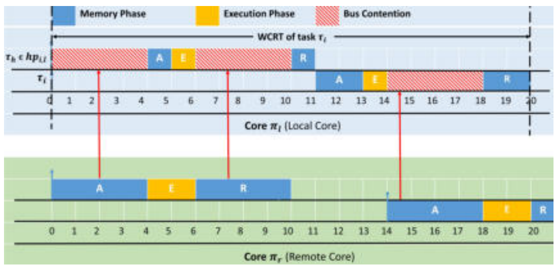
$$R_{i,l}^{max} = \max_{k \in [1,K_i]} \{R_{i,k,l}\} \tag{6}$$

where K_i is computed using Equation 3. Readers are referred to [12] for details on the formulation and proofs of Equation 2-6.

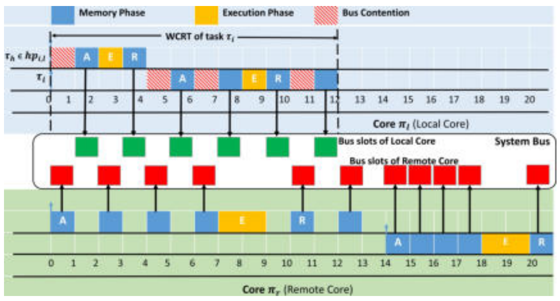
3.1. Problem Formulation

The bus contention analysis presented in [12] assumes that the bus arbitration policy is FCFS. Hence, a memory phase that executes on the local core can be served only after the completion of the memory phases of the tasks executing on all the remote cores. This assumption is clearly very pessimistic as we show using the example scenario depicted in Figure 1. Figure 1a shows the execution of two tasks on the local core π_l, i.e., τ_h and task τ_i, along with different job releases on a remote core π_r during the same time interval. We can clearly see that memory phases of the tasks running on the local core are smaller, i.e., they issue a smaller number of memory requests. On the contrary, the memory phases of the tasks running on the remote core are larger, i.e., they issues a larger number of memory requests. Under FCFS, in the worst-case, a memory phase running on the local core has to wait for the completion of all the memory requests made by the memory phases running on the remote core. Consequently, we see in Figure 1a that task executing on the local core π_l suffer larger bus contention under FCFS bus arbitration scheme.

On the other hand, RR bus arbitration makes use of bus slots in which a core can use the bus only during its assigned slot. Thus, the bus contention that can be suffered/caused by the memory phases running on the local/remote core depends on the number of memory requests that can be performed during those bus slots. As shown in Figure 1b, the slot size is set such that each core can execute at most one memory request

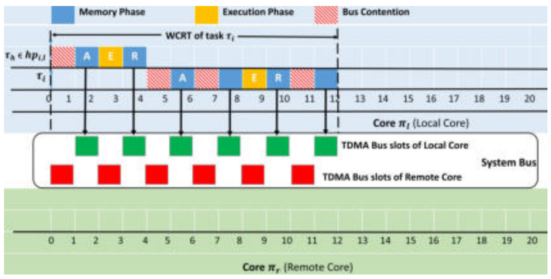


(a) Example Scenario for FCFS Analysis of [12]

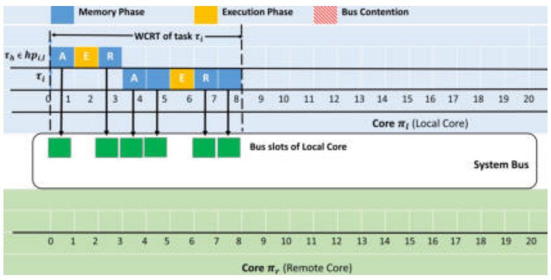


(b) Example Scenario for RR bus arbitration

Figure 1: WCRT of tasks under FCFS (a) and RR Bus Arbitration Policy (b)


Memory Phase
Execution Phase
Bus Contention
WCRT of task τ_i
$\tau_h \in hp_{i,l}$
τ_i
Core π_l (Local Core)
TDMA Bus slots of Local Core
System Bus
TDMA Bus slots of Remote Core
Core π_r (Remote Core)

(a) Example Scenario for TDMA Analysis of [8]



(b) Example Scenario for RR bus arbitration

Figure 2: WCRT of tasks under TDMA (a) and RR Bus Arbitration Policy (b)

during its bus slot. Consequently, we can see in the Figure 1b that the same tasks running on the local core, i.e., τ_h and τ_i, suffer lesser bus contention under the RR-based bus arbitration policy.

Similarly, few other works [8] that compute the bus contention for the 3-phase task model assuming a TDMA-based bus arbitration scheme also overestimate the bus contention and therefore the WCRT of tasks due to the non-work-conserving nature of the TDMA bus. We explain this overestimation using the example scenario depicted in Figure 2. Figure 2a shows execution of two tasks, i.e., τ_h and task τ_i, on the local core π_l whereas no task is released on the remote core π_r. Since TDMA is a non-work-conserving bus arbitration policy, each core reserves their bus slots disregard of the memory requests issued by the tasks running on that core. Consequently, the memory phases running on the local core suffer bus contention due to the TDMA bus slots assigned to the remote core. If instead, an RR-based bus arbitration was used (as shown in Figure 2b) task will not suffer the bus contention.

Simple example scenarios discussed above shows that the bus contention that can be suffered by tasks executing on a multicore platform can be reduced by using a RR-bases bus arbitration scheme. In the following sections, we will explain how to upper bound the bus contention of 3-phase tasks under the RR bus arbitration scheme.

4. Bus Contention Analysis for RR-based Bus Arbitration Policy

To reduce the pessimism of the existing bus contention analyses for the 3-phase task model [8, 12], in this section, we present a bus contention analysis for the 3-phase task model considering a Round-Robin-based bus arbitration policy.

When computing the maximum bus contention that can be suffered by tasks under a RR-based bus arbitration policy, we assume that task τ_i is the task under analysis, executing on some core π_l (referred to as the local core in this work) of a multicore platform. Our goal is to bound the maximum bus contention that can be suffered by all jobs released by all tasks in $hep_{i,l}$ (including τ_i) on core π_l, in a level-i busy window $W_{i,l}$. For now, we only consider the bus contention that can be caused due to tasks executing on one remote core, (denoted as π_r). We later generalize our analysis to account for the bus contention that can be caused by multiple cores.

Under an RR-based bus arbitration policy, tasks executing on the local core π_l can suffer bus contention when they have to wait for the completion of bus access slots assigned to all other remote cores. In the worst-case, the bus access slot(s) required by the tasks executing on the local core π_l may become active after the completion of the bus access slot(s) utilized by a remote core π_r. This means that the bus contention that can be suffered by tasks executing on the local core π_l not only depends on the number of bus slots required by the tasks executing on the local core but also on the number of bus access slots required by the tasks executing on the remote core π_r. Building on this insight, we propose a two-step solution to bound the maximum bus contention that can be suffered by tasks executing on the local core π_l due to other co-running tasks on a remote core π_r, within a time interval of length $W_{i,l}$. The two-steps are explained as follows:

- **Step 1:** Bound the maximum number of bus slots required by the tasks executing on the local core π_l and remote core π_r within $W_{i,l}$. This step is discussed in detail in Section 4.1.
- **Step 2:** Compute the maximum bus contention that can be suffered by the tasks executing on the local core π_l from a remote core π_r during $W_{i,l}$. This step is discussed in detail in Section 4.2.

4.1. Step 1: Bounding the Maximum Number of Bus Slots required by the Local/Remote Core

As we discussed previously, under the RR-based bus arbitration policy, the bus contention that can be suffered by tasks executing on the local core π_l due to the execution of tasks on a remote core π_r depends on the number of bus slots required by the local and remote core. Therefore, we first bound the maximum number of bus slots required by the local core π_l and remote core π_r in any time window of length $W_{i,l}$ using the following two lemmas.

Lemma 1. *The maximum number of bus slots required by the tasks running on the local core π_l in any time interval of length $W_{i,l}$ is upper-bounded by $\beta_{\pi_l}(W_{i,l})$, where*

$$\beta_{\pi_l}(W_{i,l}) = \left(\sum_{\tau_h \in hep_{i,l}} \eta_h^+(W_{i,l}) \times \left(\left\lceil \frac{MR_h^A \times MT}{SS} \right\rceil + \left\lceil \frac{MR_h^R \times MT}{SS} \right\rceil \right) \right) + \left(\left\lceil \frac{MR_j^A \times MT}{SS} \right\rceil + \left\lceil \frac{MR_j^R \times MT}{SS} \right\rceil \right) \tag{7}$$

In Equation 7, MR_h^A (resp. MR_h^R) is the maximum number of memory requests issued during the A-phase (resp. R-phase) of a task $\tau_h \in hep_{i,l}$. Similarly, MR_j^A (resp. MR_j^R) is the maximum number of memory requests issued during the A-phase (resp. R-phase) of a task $\tau_j \in lp_{i,l}$.

Proof. By definition, the maximum number of memory requests that can be generated by a task $\tau_h \in hep_{i,l}$ during its A- and R-phase is upper bounded by MR_h^A and MR_h^R, respectively. Also, we know that the maximum time needed to perform one memory request is given by MT. Consequently, the term $MR_h^A \times MT$ (resp. $MR_h^R \times MT$) gives the maximum time needed to complete the A-phase (resp. the R-phase) of task τ_h on core π_l. Knowing that under the RR-based bus arbitration policy, a core can access the bus only during its assigned bus slot for at most SS time units. Therefore, one job of task $\tau_h \in hep_{i,l}$ that execute on core π_l during $W_{i,l}$ will use $\left\lceil \frac{MR_h^A \times MT}{SS} \right\rceil + \left\lceil \frac{MR_h^R \times MT}{SS} \right\rceil$ bus slots and all the jobs of τ_h that execute during $W_{i,l}$, i.e., upper bounded by $\eta_h^+(W_{i,l})$, will require $\eta_h^+(W_{i,l}) \times \left(\left\lceil \frac{MR_h^A \times MT}{SS} \right\rceil + \left\lceil \frac{MR_h^R \times MT}{SS} \right\rceil \right)$ bus slots. Hence, $\sum_{\tau_h \in hep_{i,l}} \eta_h^+(W_{i,l}) \times \left(\left\lceil \frac{MR_h^A \times MT}{SS} \right\rceil + \left\lceil \frac{MR_h^R \times MT}{SS} \right\rceil \right)$ bounds the maximum number of bus slots required by all the tasks in $hep_{i,l}$ during $W_{i,l}$.

Finally, we know that due to non-preemptive scheduling, if task τ_i is not the lowest priority task executing on core π_l, it can suffer blocking from one job of a lower priority task, e.g., $\tau_j \in lp_{i,l}$. Since, that one blocking job of τ_j will also require bus slots to complete its A- and R-phase, the maximum number of bus access slots required by that job of task τ_j is given by $\left\lceil \frac{MR_j^A \times MT}{SS} \right\rceil + \left\lceil \frac{MR_j^R \times MT}{SS} \right\rceil$. For now, it can be assumed that the task τ_j can be any task from $lp_{i,l}$. Later in Section 5, we present an algorithm to correctly select a specific task from $lp_{i,l}$. The Lemma follows. □

Lemma 2. *The maximum number of bus slots required by the tasks running on a remote core π_r in any time interval of length $W_{i,l}$ is upper-bounded by $\beta_{\pi_r}(W_{i,l})$, where*

$$\beta_{\pi_r}(W_{i,l}) = \sum_{\tau_u \in \Gamma'_r} \eta_u^+(W_{i,l}) \times \Big(\Big\lceil \frac{MR_u^A \times MT}{SS} \Big\rceil + \Big\lceil \frac{MR_u^R \times MT}{SS} \Big\rceil \Big) \tag{8}$$

In Equation 8, the term MR_u^A (resp. MR_u^R) is the maximum number of memory requests issued during the A-phase (resp. R-phase) of a task $\tau_u \in \Gamma'_r$

Proof. The proof directly follows from Lemma 1. However, we need to account for the bus slots required by all memory phases of all tasks released on a remote core π_r in any time interval of length $W_{i,l}$. □

4.2. Step 2: Bounding Maximum Bus Contention

Having bounded the maximum number of bus slots required by the tasks executing on the local core π_l and a remote core π_r during $W_{i,l}$, we can now compute the maximum bus contention $Bus_{i,r}(W_{i,l})$ that can be suffered by the tasks executing on the local core π_l from co-running tasks on a remote core π_r. Depending on the values of $\beta_{\pi_l}(W_{i,l})$ and $\beta_{\pi_r}(W_{i,l})$, two cases must be considered.

- **Case 1:** $\beta_{\pi_l}(W_{i,l}) \geq \beta_{\pi_r}(W_{i,l})$, i.e., the maximum number of bus slots required by tasks executing on the local core π_l during $W_{i,l}$ is greater than or equal to the maximum number of bus slots required by tasks executing on a remote core π_r during $W_{i,l}$. This case is discussed in detail in Section 4.2.1.
- **Case 2:** $\beta_{\pi_l}(W_{i,l}) < \beta_{\pi_r}(W_{i,l})$, i.e., the maximum number of bus slots required by tasks executing on the local core π_l during $W_{i,l}$ is less than the maximum number of bus slots required by tasks executing on a remote core π_r during $W_{i,l}$. This case is discussed in detail in Section 4.2.2.

4.2.1. Computing Maximum Bus Contention for Case 1

When the maximum number of bus slots required by the tasks executing on the local core π_l is greater than or equal to the maximum number of bus slots required by a remote core π_r in any time interval of length $W_{i,l}$, then, the maximum bus contention is computed using the following lemma.

Lemma 3. *If $\beta_{\pi_l}(W_{i,l}) \geq \beta_{\pi_r}(W_{i,l})$, then the maximum bus contention that can be suffered by tasks executing on the local core π_l from tasks running on a remote core π_r in any time interval of length $W_{i,l}$ is upper-bounded by $Bus_{i,r}(W_{i,l})$, where*

$$Bus_{i,r}(W_{i,l}) = \sum_{\tau_u \in \Gamma'_r} \eta_u^+(W_{i,l}) \times \Big((MR_u^A \times MT) + (MR_u^R \times MT) \Big) \tag{9}$$

In Equation 9, the term $MR_u^A \times MT$ (resp. $MR_u^R \times MT$) gives the maximum time required to serve all the memory requests generated during an A-phase (resp. R-phase) of task τ_u executing on core π_r.

Proof. In the worst-case, each bus slot required by the tasks executing on the local core π_l can only be active after the completion of one bus slot used by tasks executing on the remote core π_r. We know that the number of bus slots required by the remote core, i.e., $\beta_{\pi_r}(W_{i,l})$, are less than or equal to $\beta_{\pi_l}(W_{i,l})$. This means that all the memory requests generated by all the tasks executing on core π_r during $W_{i,l}$ can contribute to the bus contention suffered by tasks running on core π_l.

Knowing that each task τ_u assigned to core π_r can execute at most $\eta_u^+(W_{i,l})$ jobs during $W_{i,l}$, and the time required to complete the A-phase (resp. R-phase) of one job of task τ_u is given by $MR_u^A \times MT$ (resp. $MR_u^R \times MT$), the total time required to complete the A- and R-phases of all jobs of all tasks that execute on core π_r during $W_{i,l}$ is given by $\sum_{\tau_u \in \Gamma'_r} \eta_u^+(W_{i,l}) \times \Big((MR_u^A \times MT) + (MR_u^R \times MT) \Big)$. Equation 9 also upper bounds the maximum bus contention that can be suffered by the tasks executing on the local π_l due to the tasks released on a remote core π_r during $W_{i,l}$, if $\beta_{\pi_l}(W_{i,l}) \geq \beta_{\pi_r}(W_{i,l})$. The Lemma follows. □

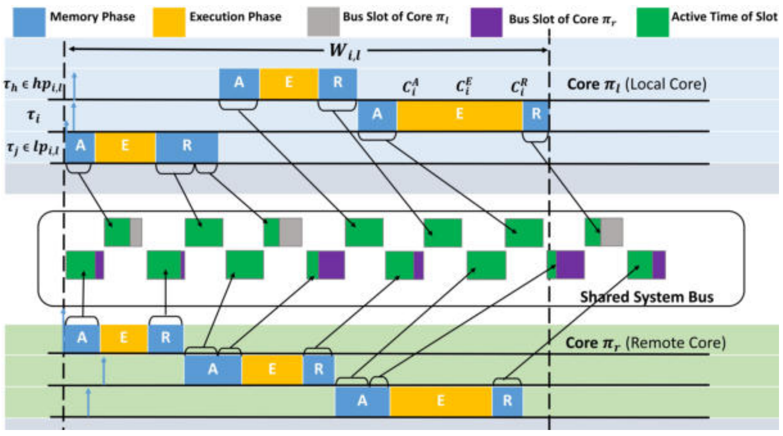


Figure 3: Example Scenario to Derive Maximum Bus Contention for Case 2

Note that it is also possible to directly compute the maximum bus contention for case 1 using the maximum number of slots required by the tasks executing on the remote core π_r during $W_{i,l}$ ($\beta_{\pi_r}(W_{i,l})$) and the size of one bus slot (SS). Effectively, $\beta_{\pi_r}(W_{i,l}) \times SS$ upper bounds the bus contention for case 1. Although, this bound is safe, it can be pessimistic as the remote core may not always fully utilize all its $\beta_{\pi_r}(W_{i,l})$ bus slots for SS time units as the utilization of a bus slot depends on the number of memory requests performed during that bus slot. Therefore, Equation 9 which is built considering the size of memory phases of tasks, provides a tighter bound on the bus contention that can be caused by a remote core π_r during a time window of length $W_{i,l}$.

4.2.2. Computing Maximum Bus Contention for Case 2

For any given time interval of length $W_{i,l}$, if the maximum number of bus slots required by the tasks executing on the local core π_l is less than the maximum number of bus slots required by tasks executing on a remote core π_r, then, all bus slots used by the remote core can not contribute to the bus contention. This is mainly because each bus slot required by the local core π_l can only suffer bus contention from at most one slot used by the remote core π_r. Hence, $\beta_{\pi_l}(W_{i,l})$ slots requested by the local core can be served after the execution of at most $\beta_{\pi_l}(W_{i,l})$ bus slots used by the remote core. Consequently, when computing the bus contention that can be suffered by tasks executing on the local core π_l due to co-running tasks executing on core π_r, we need to consider only $\beta_{\pi_l}(W_{i,l})$ bus slots.

Knowing that at most $\beta_{\pi_l}(W_{i,l})$ bus slots used by a remote core π_r can cause bus contention to tasks executing on core π_l, we can upper bound the bus contention that can be caused by core π_r to core π_l during $W_{i,l}$ by simply multiplying $\beta_{\pi_l}(W_{i,l})$ with the size of one bus slot, i.e.,

$$Bus_{i,r}(W_{i,l}) = \beta_{\pi_l}(W_{i,l}) \times SS \tag{10}$$

Equation 10 assumes that each of $\beta_{\pi_l}(W_{i,l})$ bus slots from the remote core π_r that can contribute to the blocking of core π_l will be fully utilized, i.e., up to SS time units, by tasks that execute on π_r during $W_{i,l}$. This assumption is safe but can be pessimistic as we explain using the below example.

Example 1: *Figure 3 shows a schedule of tasks executing on the local core π_l and the remote core π_r, along with the utilization of bus slots by the cores. We can see in the figure that the maximum number of bus slots required by the tasks running on the local core π_l during $W_{i,l}$ is seven, i.e., $\beta_{\pi_l}(W_{i,l}) = 7$. Moreover, the maximum number of bus slots required by the tasks running on the remote core π_r during $W_{i,l}$ is eight, i.e., $\beta_{\pi_r}(W_{i,l}) = 8$. Since $\beta_{\pi_l}(W_{i,l}) < \beta_{\pi_r}(W_{i,l})$, the local core can suffer bus contention from at most $\beta_{\pi_l}(W_{i,l})$ bus slots from the remote core in this example. While an upper bound on the total bus contention can be computed using Equation 10, by looking closely at the utilization of the bus slots in Figure 3, we can see that this bound can be very pessimistic. This is mainly because for the scenario shown in Figure 3, the active time, i.e., the time in which memory requests are performed during a bus slot, of some of the bus slots used* *by the remote core π_r is much less than the size of the bus slot, i.e., SS. Hence, assuming that each bus slot of the local core suffers a delay of SS from the remote core can be very pessimistic.*

The above example shows that for $\beta_{\pi_l}(W_{i,l}) < \beta_{\pi_r}(W_{i,l})$ a tighter bound on the bus contention can only be obtained by considering the active time of each bus slot accessed by the tasks executing on π_r during $W_{i,l}$. Consequently, the following Lemmas are used to incorporate the active time of each bus slot used by the A- and R-phases of all the tasks running on the remote core π_r in a time window of length $W_{i,l}$.

Lemma 4. *If n represents the number of bus slots required to complete the execution of an A-phase of a task τ_u executing on a remote core π_r during $W_{i,l}$, then the active time of each bus slot in the range 1 to $n-1$, used by the A-phase of task τ_u is given by $\sigma^A_{u,r,y}$, where*

$$\sigma^A_{u,r,y} = SS \qquad \forall y \in [1, n-1] \tag{11}$$

In Equation 11, u represents the task index, r the core index, and y is used to represent the bus slot index, i.e., y^{th} bus slot such that $y \in [1, n-1]$.

Proof. We prove that if a task τ_u executing on core π_r requires n bus slots to complete an A-phase, then, τ_u will fully utilize at least $n-1$ bus slots for SS time units.

The A-phase of a task τ_u executing on core π_r will only require n bus slots if it cannot complete its execution within $n-1$ bus slots. This means that for each bus slot, until the $n-1^{th}$ bus slot, the time required to serve the pending memory requests of the A-phase of task τ_u is always greater than SS. Hence, it can only be the case that the n^{th} bus slot used by the A-phase may or may not be fully utilized. The Lemma follows. □

Lemma 5. *If n represents the number of bus slots required to complete the execution of an A-phase of a task τ_u executing on a remote core π_r during $W_{i,l}$, then the active time of the n^{th} bus slot used by the A-phase of task τ_u is given by $\sigma^A_{u,r,n}$, where*

$$\sigma^A_{u,r,n} = (MR^A_u \times MT) - ((n-1) \times SS) \tag{12}$$

Proof. The maximum number of memory requests that can be issued during an A-phase of a task τ_u executing on a core π_r is upper bounded by MR^A_u. Moreover, in the worst-case, each memory request can be served in MT time units. Consequently, the total time required to complete an A-phase of task τ_u on core π_r is given by $MR^A_u \times MT$.

From Lemma 4, if an A-phase of task τ_u needs n bus slots to complete its execution, then, it will fully utilize at least $n-1$ bus slots, and the maximum workload that can be done during $n-1$ bus slots is given by $(n-1) \times SS$. Consequently, the maximum time that can be used by the A-phase of task τ_u during the n^{th} bus slot, i.e., the active time of the n^{th} bus slot, is given by $(MR^A_u \times MT) - ((n-1) \times SS)$. The Lemma follows. □

Having computed the active time of bus slots used by the A-phases of tasks, we can use the same approach to compute the active time of bus slots used during the R-phases of tasks.

Lemma 6. *If n represents the number of bus slots required to complete the execution of an R-phase of a task τ_u executing on a remote core π_r during $W_{i,l}$, then the active time of each bus slot in the range 1 to $n-1$, used by the R-phase of task τ_u is given by $\sigma^R_{u,r,y}$, where*

$$\sigma^R_{u,r,y} = SS \qquad \forall y \in [1, n-1] \tag{13}$$

Proof. The proof directly follows from Lemma 4 considering the bus slots required by the R-phase of task τ_u. □

Lemma 7. *If n represents the number of bus slots required to complete the execution of an R-phase of a task τ_u executing on a remote core π_r during $W_{i,l}$, then the active time of the n^{th} bus slot used by the R-phase of task τ_u is given by $\sigma^R_{u,r,n}$, where*

$$\sigma^R_{u,r,n} = (MR^R_u \times MT) - ((n-1) \times SS) \tag{14}$$

Proof. The proof directly follows from Lemma 5 considering the bus slots required by the R-phase of task τ_u. □

Having bounded the active time of each bus access slot used by all the memory phases of all the tasks released on a remote core π_r in any time interval of length $W_{i,l}$, we will now explain how to compute the maximum bus contention for case 2, i.e., $\beta_{\pi_l}(W_{i,l}) < \beta_{\pi_r}(W_{i,l})$.

Let V be an ordered set that contains the active time of each bus slot utilized by all the memory phases (i.e., both A- and R-phases) released on a remote core π_r in any time interval of length $W_{i,l}$, *sorted in non-increasing order*, i.e.,

$$V = \{\sigma_{r,1}, \sigma_{r,2}, \dots \sigma_{r,Q} \mid \sigma_{r,x} \geq \sigma_{r,x+1}\} \tag{15}$$

where $\sigma_{r,x}$ denotes the active time of bus slot x used by any memory phase (i.e., A- or R-phase) of any task that may execute on core π_r during $W_{i,l}$. In Equation 15, the index Q is equal to the maximum number of bus slots used by all the tasks that execute on π_r during $W_{i,l}$, i.e., $Q = \beta_{\pi_r}(W_{i,l})$.

Using the above defined notations, the maximum bus contention for case 2 can be derived using the following lemma.

Lemma 8. *If $\beta_{\pi_l}(W_{i,l}) < \beta_{\pi_r}(W_{i,l})$, then the maximum bus contention that can be suffered by tasks executing on the local core π_l from the tasks running on a remote core π_r in any time interval of length $W_{i,l}$ is upper-bounded by $Bus_{i,r}(W_{i,l})$, where*

$$Bus_{i,r}(W_{i,l}) = \sum_{x=1}^{x=\beta_{\pi_l}(W_{i,l})} \sigma_{r,x} \tag{16}$$

Proof. Since the tasks executing on the local core π_l require $\beta_{\pi_l}(W_{i,l})$ bus slots during $W_{i,l}$, at most $\beta_{\pi_l}(W_{i,l})$ bus slots utilized by the tasks executing on the remote core π_r can cause bus contention. However, as we cannot predict the schedule of tasks on the remote core, we do not know which memory phases of which tasks may use these bus slots. Therefore, to maximize the bus contention, we choose $\beta_{\pi_l}(W_{i,l})$ bus slots with the largest active times among all the bus slots used by the tasks released on the remote core π_r in any time interval of length $W_{i,l}$. This is achieved by extracting the first $\beta_{\pi_l}(W_{i,l})$ values from set V, that contains the active times of all bus slots utilized on core π_r during $W_{i,l}$ (see Equation 15). □

Having bounded the maximum bus contention $Bus_{i,r}(W_{i,l})$ that can be suffered by the local core π_l from a remote core π_r in any time interval of length $W_{i,l}$, we can now compute the *total bus contention* that can be suffered by tasks executing on core π_l due to tasks running on all other cores. Knowing that under a RR-based bus arbitration policy, the worst-case scenario may happen when a bus slot required by a given task executing on the local core is served after the completion of one bus slot from each of the remotes cores.

Therefore, under the RR-based bus arbitration policy, the *total bus contention* that can be suffered by the local core π_l from *all remote cores* in any time interval of length $W_{i,l}$ is given by $Bus^{max}_{i,l}(W_{i,l})$, where

$$Bus^{max}_{i,l}(W_{i,l}) = \sum_{r=1, r\neq l}^{m} Bus_{i,r}(W_{i,l}) \tag{17}$$

5. Accurately Estimating the Impact of Lower Priority Blocking

Under FPNP scheduling, the execution of a task τ_i can be blocked due to the execution of a lower priority task, e.g., task τ_j in $lp_{i,l}$, that contributes to the length of level-i busy window. The blocking that τ_j can cause during the level-i busy window is usually upper bounded by choosing τ_j such that it has the maximum execution time among all the tasks in $lp_{i,l}$, i.e., $C^{max}_{lp_{i,l}} = \max_{\forall \tau_j \in lp_{i,l}} \{C_j\}$. While this assumption is sound when considering a single-core platform, it may lead to unsafe results for multicore architectures. This is mainly because when τ_j, the task that blocks the execution of τ_i, is executing on a single-core processor the system bus and all other resources are dedicated to τ_j only. So, in the worst-case τ_j executes for its entire WCET at the beginning of the level-i busy window. However, on a multicore processor, task τ_j can suffer

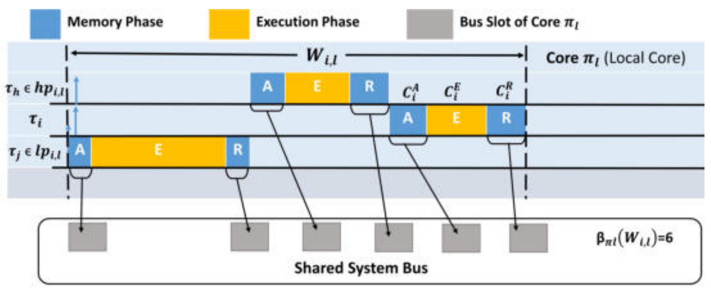


Figure 4: Scenario 1 when task $\tau_j \in lp_{i,l}$ cause blocking to task τ_i during $W_{i,l}$

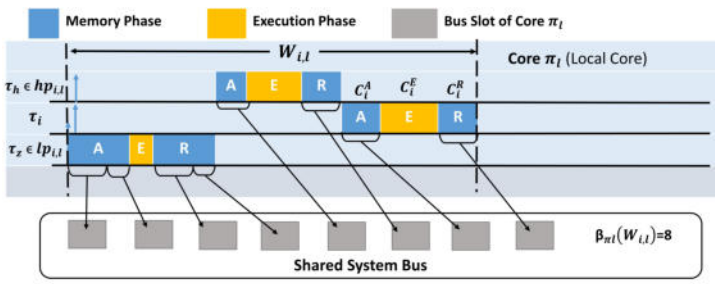


Figure 5: Scenario 2 when task $\tau_z \in lp_{i,l}$ cause blocking to task τ_i during $W_{i,l}$

execution delays in addition to its WCET due to the bus contention it may suffer during its execution. This bus contention does not entirely depends on the WCET time of τ_j but also on its memory access demand, i.e., the total number of memory requests that can be generated by τ_j's memory phases. For instance, we can have a scenario where another task $\tau_z \in lp_{i,l}$ with $\tau_z \neq \tau_j$, having a smaller WCET than τ_j but a higher memory access demand, may suffer higher bus contention than τ_j. This will eventually result in τ_z contributing more to the blocking of τ_i than τ_j. To illustrate, consider the two scenarios shown in Figure 4 and Figure 5.

Scenario 1, depicted in Figure 4, shows that a task $\tau_j \in lp_{i,l}$ is blocking the execution of tasks in $hep_{i,l}$ on core π_l at the start of the level-i busy window. τ_j has the largest WCET among all tasks in $lp_{i,l}$ but it has smaller A- and R-phases. We can see in Figure 4, that task τ_j require two bus slots to complete its A- and R-phase. Similarly, all other tasks in $hep_{i,l}$ executing on core π_l require four bus slots to complete their memory phases. Eventually, in scenario 1, the maximum number of bus slots required to complete memory phases of all tasks that execute on core π_l during $W_{i,l}$ is equal to 6, i.e., $\beta_{\pi_l}(W_{i,l}) = 6$.

Figure 5, depicts another scenario, where a task $\tau_z \in lp_{i,l}$, with $\tau_z \neq \tau_j$, is causing blocking at the start of level-i busy window on core π_l. τ_z has a smaller overall WCET than τ_j but it has larger A- and R-phases. Consequently, τ_z needs four bus slots to complete its A- and R-phases. All other tasks in $hep_{i,l}$ executing on core π_l are the same as in Figure 4 and require four bus slots to complete their memory phases. Eventually, for the execution scenario shown in Figure 5, the maximum number of bus slots required to complete the memory phases of all tasks that execute on core π_l during $W_{i,l}$ is equal to 8, i.e., $\beta_{\pi_l}(W_{i,l}) = 8$. As shown previously, the bus contention that can be suffered by tasks executing on π_l during $W_{i,l}$ depends on the value of $\beta_{\pi_l}(W_{i,l})$ and a larger value of $\beta_{\pi_l}(W_{i,l})$ may lead to more bus contention. Therefore, when analyzing multicore systems, the scenario depicted in Figure 5 may lead to a higher bus contention and eventually a larger level-i busy window than the scenario depicted in Figure 4.

To the best of our knowledge, the only existing work in the state-of-the-art that accurately accounts for the lower priority blocking when computing bus contention is presented in [31] (See Equation 11 of [31]). However, the solution provided in [31] is developed considering the generic task model and therefore, can not be used when considering the 3-phase task model.

To accurately quantify the impact of blocking from a given task in $lp_{i,l}$, on tasks that execute on core π_l

during $W_{i,l}$, we have to evaluate the impact of bus contention on each task in $lp_{i,l}$ and the resulting length of the level-i busy window. We will then select the task from $lp_{i,l}$ that maximizes the $W_{i,l}$ as the blocking task. We use Algorithm 1 to evaluate how each task in $lp_{i,l}$ can impact bus contention and also the length of level-i busy window.

Algorithm 1 Computing the maximum delay suffered by the local core π_l during $W_{i,l}$ due to total bus contention and blocking from tasks in $lp_{i,l}$

```
1:  $\alpha_{i,l}^{max}(W_{i,l}) := 0$
2:  for $\forall \tau_j \in lp_{i,l}$ do
3:      Compute $\beta_{\pi_l}(W_{i,l})$ using Lemma 1 assuming $\tau_j$ will cause blocking to task $\tau_i$ on core $\pi_l$.
4:      Compute $\beta_{\pi_r}(W_{i,l})$ using Lemma 2.
5:      Compute $Bus_{i,r}(W_{i,l})$ using Lemma 3 up to Lemma 8.
6:      Compute the total bus contention $Bus_{i,l}^{max}(W_{i,l})$ using Equation 17.
7:      $\alpha_{i,l}(W_{i,l}) := Bus_{i,l}^{max}(W_{i,l}) + C_j$
8:      if $\alpha_{i,l}(W_{i,l}) > \alpha_{i,l}^{max}(W_{i,l})$ then
9:          $\alpha_{i,l}^{max}(W_{i,l}) := \alpha_{i,l}(W_{i,l})$
10:     end if
11: end for
```

Algorithm 1 computes the maximum delay that can be suffered by the local core π_l during $W_{i,l}$ due to the total bus contention caused by all the remote cores and blocking caused by one job from any task $\tau_j \in lp_{i,l}$. Since we need to consider all tasks in $lp_{i,l}$, Algorithm 1 iterates over all tasks in $lp_{i,l}$ (lines 2 to 10). For every task $\tau_j \in lp_{i,l}$, it first computes the maximum number of slots required by all the tasks that execute on the local core π_l during $W_{i,l}$, i.e., $\beta_{\pi_l}(W_{i,l})$, using Lemma 1 (line 3). The maximum number of bus slots required by all the tasks released on a remote core π_r in any time interval of length $W_{i,l}$, i.e., $\beta_{\pi_r}(W_{i,l})$, are computed using Lemma 2 (line 4).

Values of $\beta_{\pi_l}(W_{i,l})$ and $\beta_{\pi_r}(W_{i,l})$ derived in lines 3 and 4 are then used as input to Lemma 3 up to Lemma 8, to compute the maximum bus contention that can be caused by a remote core π_r to the tasks running on the local core π_l during $W_{i,l}$, i.e., $Bus_{i,r}(W_{i,l})$. The total bus contention that can be suffered by the local core π_l from all remote cores during $W_{i,l}$, i.e., $Bus_{i,l}^{max}(W_{i,l})$, is then computed using Equation 17 (line 6). Line 7 computes $\alpha_{i,l}(W_{i,l})$, i.e., the total delay that can be suffered by the local core π_l during $W_{i,l}$ due to the total bus contention caused by all the remote cores, i.e., $Bus_{i,l}^{max}(W_{i,l})$ plus the blocking caused by one job of task τ_j. Lines 8 to 10 compare the derived values of $\alpha_{i,l}(W_{i,l})$ for each $\tau_j \in lp_{i,l}$ to find $\alpha_{i,l}^{max}(W_{i,l})$ which is the *maximum delay* that can be suffered by the local core π_l during $W_{i,l}$.

6. Schedulability Analysis

In this section, we derive the schedulability analysis for the 3-phase task model while using partitioned fixed-priority scheduling, by integrating the bus contention computed in Section 4.2 and 5. Similarly to the work in [12], we first compute the length of the longest level-i busy window $W_{i,l}$ on the local core, i.e., π_l.

The longest level-i busy window $W_{i,l}$ for the local core π_l w.r.t task τ_i is given by the first positive fixed-point solution of the following equation:

$$W_{i,l} = \alpha_{i,l}^{max}(W_{i,l}) + \sum_{\tau_h \in hep_{i,l}} (\eta_h^+(W_{i,l}) \times C_h) \tag{18}$$

where $\eta_h^+(W_{i,l})$ gives the maximum number of jobs released by $\tau_h \in hep_{i,l}$ in any time interval of length $W_{i,l}$. Consequently, the term $\sum_{\tau_h \in hep_{i,l}} (\eta_h^+(W_{i,l}) \times C_h)$ captures the contribution of all the jobs from $hep_{i,l}$ task set in any time interval of length ${W_{i,l}}$[2]. The term $\alpha_{i,l}^{max}(W_{i,l})$ captures the maximum delay suffered

[2]Having accounted for the total bus contention that can be suffered by all tasks of local core during $W_{i,l}$, we can use the value C_h to account for time required to execute task $\tau_h \in hep_{i,l}$.

by the local core π_l in any time interval $W_{i,l}$ due to the total bus contention caused by all the remote cores and blocking caused by one job from $lp_{i,l}$ task set. $\alpha_{i,l}^{max}(W_{i,l})$ is computed using Algorithm 1.

Having bounded the value of $W_{i,l}$, we can compute the maximum number of jobs of task τ_i that can execute on core π_l during $W_{i,l}$ using Equation 3, i.e., $K_i = \eta_i^+(W_{i,l})$.

To compute the response time of the k^{th} job of τ_i on core π_l, i.e., $\tau_{i,k,l}$, we compute the latest starting time of the *R-phase* of $\tau_{i,k,l}$ using the following Lemma.

Lemma 9. *The latest start time of the R-phase of* $\tau_{i,k,l}$ *is denoted by* $s_{i,k,l}^R$*, where* $s_{i,k,l}^R$ *is given by the first positive solution to the fixed-point iteration on the following equation:*

$$s_{i,k,l}^R = \alpha_{i,l}^{max}(s_{i,k,l}^R) + ((k-1) \times C_i) + (C_i^A + C_i^E) + \sum_{\tau_h \in hep_{i,l} \setminus \tau_i} \eta_h^+(s_{i,k,l}^R - (C_i^A + C_i^E)) \times C_h \tag{19}$$

Proof. As the latest starting time of the R-phase of $\tau_{i,k,l}$ is being computed, the sum of the execution time of the A-phase and the E-phase of task τ_i needs to be considered, i.e., $C_i^A + C_i^E$.

The execution of $\tau_{i,k,l}$ can be impacted by all the jobs of task τ_i that execute before $\tau_{i,k,l}$. Hence, $(k-1) \times C_i$ considers the maximum delay that can be caused by the $k-1$ jobs of τ_i.

Due to the fixed-priority non-preemptive scheduling, all the jobs released by all the tasks in $hep_{i,l}$ (excluding τ_i) can delay the execution of $\tau_{i,k,l}$ until the start of the A-phase of $\tau_{i,k,l}$. This implies that the maximum workload that can be generated by a task $\tau_h \in hep_{i,l}$ until the start of the A-phase of $\tau_{i,k,l}$ is upper bounded by $\eta_h^+(s_{i,k,l}^R - (C_i^A + C_i^E)) \times C_h$. Hence, the total workload that can be generated by all tasks in $hep_{i,l}$ taskset until the start of the A-phase of $\tau_{i,k,l}$ is upper bounded by $\sum_{\tau_h \in hep_{i,l} \setminus \tau_i} \eta_h^+(s_{i,k,l}^R - (C_i^A + C_i^E)) \times C_h$.

Due to the non-preemptive execution, one job of a task in $lp_{i,l}$ taskset can delay the execution of $\tau_{i,k,l}$. Furthermore, each job that executes on the local core π_l until the completion of $\tau_{i,k,l}$ can suffer bus contention. The term $\alpha_{i,l}^{max}(s_{i,k,l}^R)$ captures the maximum delay that can be suffered by core π_l in any time interval of length $s_{i,k,l}^R$ due to the total bus contention, i.e., $Bus_{i,l}^{max}(s_{i,k,l}^R)$, and the blocking caused by a task in $lp_{i,l}$ taskset[3]. Hence, proving the lemma. □

As $s_{i,k,l}^R$ appears on both sides of Equation 19, it can be solved iteratively by initializing $s_{i,k,l}^R = C_i^A + C_i^E + C_{lp_{i,l}}^{max} + \sum_{\tau_h \in hep_{i,l} \setminus \tau_i} C_h$. The starting time $s_{i,k,l}^R$ will then be given by the smallest positive value of $s_{i,k,l}^R$ for which Equation 19 converges.

Having computed the value of $s_{i,k,l}^R$, the response time $R_{i,k,l}$ of $\tau_{i,k,l}$ can be computed by adding the execution time of the R-phase of τ_i, i.e.,

$$R_{i,k,l} = s_{i,k,l}^R + C_i^R \tag{20}$$

Finally, the WCRT of task τ_i is then given by the largest response time of any job of τ_i that executes during the level-i busy window. Hence,

$$R_{i,l}^{max} = \max_{k \in [1, K_i]} \{R_{i,k,l}\} \tag{21}$$

where the computation of K_i is obtained using Equation 3.

A task τ_i is deemed schedulable if its WCRT $R_{i,l}^{max}$ is less than or equal to its relative deadline D_i. Similarly, a task set Γ is deemed schedulable if all tasks $\tau_i \in \Gamma$ are schedulable. Also, note that for a task set to be schedulable, the total core utilization of each individual core should not be greater than 1 and the total bus utilization of the taskset should be less than or equal to 1.

7. Experimental Evaluation

In this section, we evaluate the performance of our proposed analysis. To the best of our knowledge, no work exists in the state-of-the-art that focuses on the bus contention analysis for the 3-phase task model,

[3]The value of $\alpha_{i,l}^{max}(s_{i,k,l}^R)$ can be computed using Algorithm 1 by simply replacing $W_{i,l}$ with $s_{i,k,l}^R$ in each line of Algorithm 1.

considering partitioned fixed priority non-preemptive scheduling and Round-Robin (RR) bus arbitration policy. The most relevant works that present similar bus contention analyses as ours are [8] and [12]. The work in [8] focus on the bus-contention analysis for various task execution models including the 3-phase task model (referred to as the dedicated phase model in [8]). Their bus contention analysis is based on a TDMA-based bus arbitration policy in which cores reserve their TDMA bus slots disregarding the number of memory requests issued by the tasks running on the cores. The work in [12] provides a fine-grained bus contention analysis for the 3-phase task model assuming a FCFS (First-Come First-Serve) bus arbitration scheme.

To compare our proposed approach against the analyses in [8] and [12], we performed several experiments using synthetic task sets under different settings. As a default configuration, we assume a multicore architecture with 4 cores. The total number of tasks per task set are 32, where each core is assigned 8 tasks at design time. Tasks utilizations are generated using Uunifast-discard [32]. Tasks' periods are generated using log-uniform distribution in the range of [100,500]. We assume that the slot size SS is same for each core and its value is set such that $SS = 2 \times MT$.

The WCET C_i of each task τ_i is set using its utilization and period, i.e., $C_i = U_i \times T_i$. The memory access demand MD_i of task τ_i, i.e., the maximum time required by τ_i to complete its A and R-phase when executing in isolation, is chosen randomly in the range $[10\%, 50\%] \times C_i$. The value of C_i^A, C_i^R and C_i^E are then set such that $C_i^A = C_i^R = (MD_i/2)$ and $C_i^E = C_i - (C_i^A + C_i^R)$. Furthermore, we assume that the maximum time required to serve all the memory requests issued during the A-phase (resp. R-phase) of a task τ_i executing on core π_l is given by $MR_i^A \times MT = C_i^A$ (resp. $MR_i^R \times MT = C_i^R$). Task deadlines are implicit (i.e., $D_i = T_i$) with priorities assigned using Rate-Monotonic [24].

We compared the performance of our proposed analysis against the analyses in [8, 12] by varying the core utilization, memory access demand, number of cores, tasks' periods, and slot size SS. We use task set schedulability, i.e., the number of task sets deemed schedulable as the performance metric and evaluate 1000 randomly generated task sets per point for all the analyzed approaches.

1. Core Utilization: In this experiment, we vary the core utilization of each core from 0.025 to 1.0 in steps of 0.025 and evaluate its impact on task set schedulability. The percentage of task sets that were deemed schedulable using all the approaches are shown in Figure 6. The label marked as "OUR" in Figure 6 represents our proposed bus contention analysis for RR-based bus arbitration policy. The state-of-the-art analysis presented in [8] is marked as "TDMA" whereas the analysis of [12] is labeled as "FCFS". The plot in Figure 6 also shows a line marked as "No Bus Contention"[4], which represents an ideal scenario where a task set is deemed schedulable if its bus utilization is less than 1. Consequently, the No Bus Contention analysis provides an upper bound on the number of task sets that may be schedulable at any given core utilization.

In Figure 6, the x-axis represents the core-utilization and the y-axis represents percentage of schedulable tasksets for all the analyzed approaches. We can see in Figure 6 that for all the approaches the percentage of schedulable tasksets decreases with an increase in the core utilization. This is intuitive as increase in core utilization can increase the utilization of tasks that may result in increase in the WCET as $C_i = U_i \times T_i$. Higher WCET of tasks eventually result in increasing the interference/blocking that tasks can suffer from the same core as well as bus contention from remote cores. However, we can also see in Figure 6 that our proposed RR-based bus contention analysis outperforms the FCFS-based [12] and the TDMA-based [8] analysis. In fact, our proposed analysis can schedule up to 86.1% more tasksets as compared to [8] and up to 29.7% more tasksets as compared to [12]. This improvement in performance in comparison to the existing analysis can be explained as follows. When computing bus contention the analysis presented in [8] assumes a TDMA-based arbitration at the bus. Due to the non-work-conserving nature of TDMA, bus contention that can be suffered by the task executing on the local core is always computed without considering the actual workload of the remote cores. Hence, tasks are assumed to suffer more bus contention which leads to a lower task set schedulability.

[4]In No Bus Contention, we assume that tasks do not suffer any bus contention and may deemed schedulable if the total bus utilization is less than or equal to 1

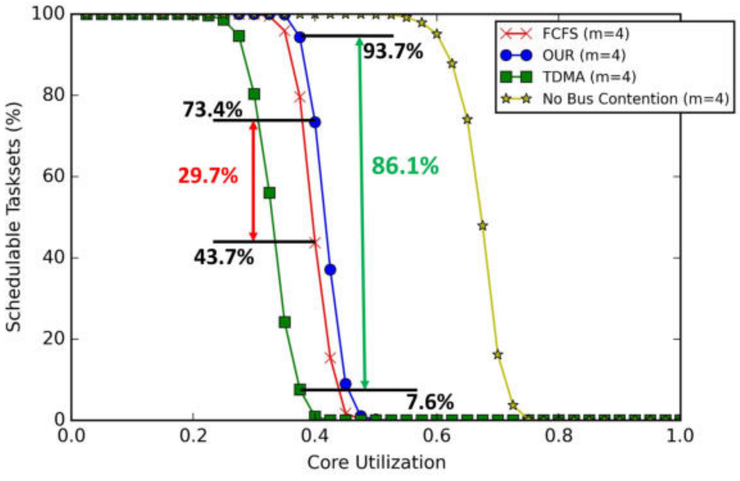
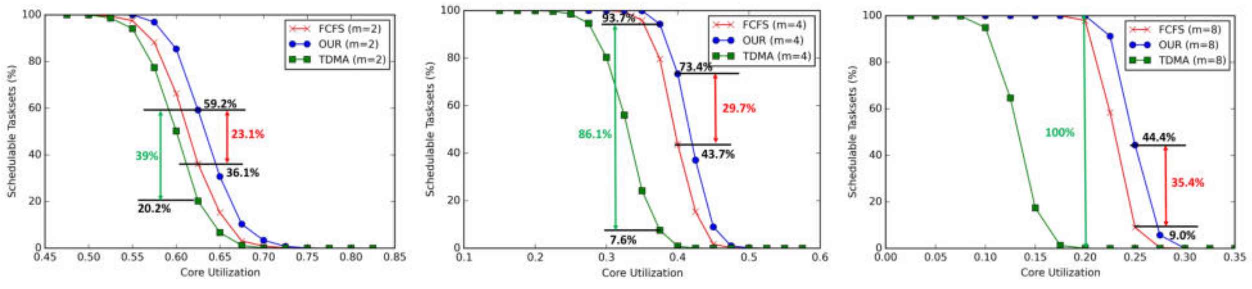


Figure 6: Varying Core Utilization



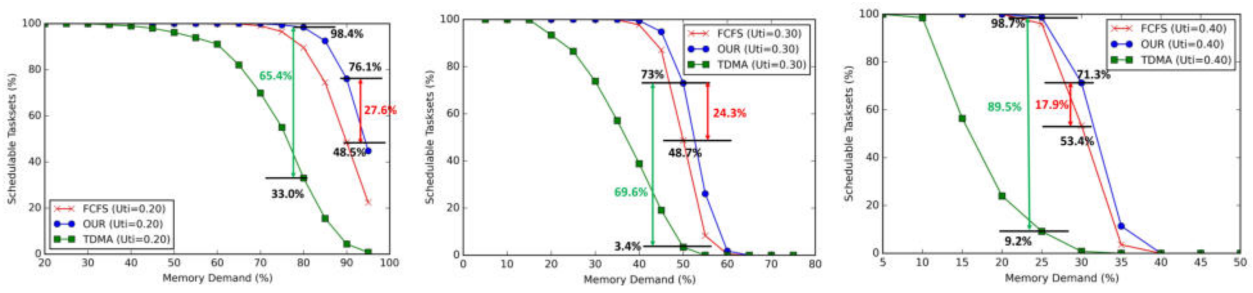
(a) Varying the Core Utilization for m=2 (b) Varying the Core Utilization for m=4 (c) Varying the Core Utilization for m=8

Figure 7: Varying the Number of Cores and Core Utilization

The analysis presented in [12] focus on FCFS-based bus arbitration, which is a work-conserving bus arbitration policy. However, the analysis of [12] only focus on the number of memory phases released on the local/remote cores when bounding the bus contention and does not account for the number of memory requests issued during the memory phase or the size of bus slots assigned to each core as shown in Figure 1a. Hence, the FCFS-based analysis also overestimates the bus contention that can be caused by the remote cores. In contrast to the TDMA and FCFS-based analysis, our proposed RR-based efficiently regulates the bus utilization among cores which leads to a significant reduction in the bus contention that can be suffered by the tasks.

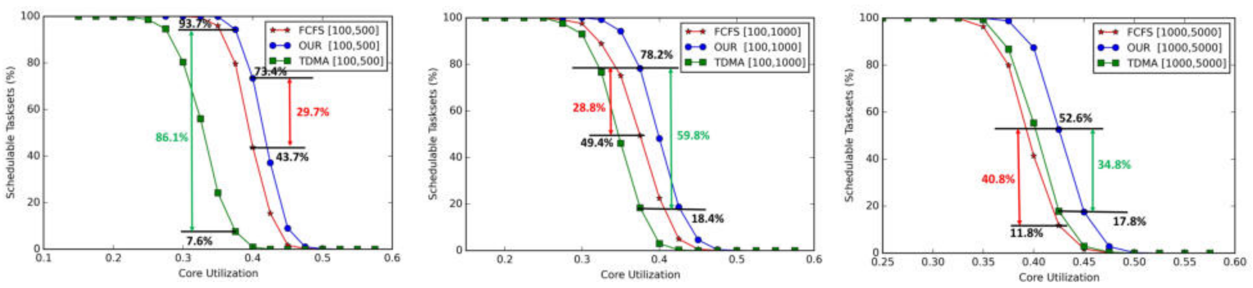
2. Number of Cores: To evaluate the impact of the number of cores (and the number of tasks) on the performance of all analyzed approaches, we re-do experiment 1 by varying the value of m (i.e., number of cores) between 2 and 8 along with core utilizations. We observe in Figure 7 that by increasing the number of cores the task set schedulability for all approaches decreases. This is mainly because, by increasing the number of cores, the total number of tasks executing on the remote cores also increases. This increase in the number of tasks leads to a higher contention at the bus. Hence, we see in Figure 7c, a very low percentage of task sets are schedulable by all the approaches for m=8, i.e., a total of 64 tasks in the system. However, we can also note that our proposed analysis always dominates other analyses by improving task set schedulability by up to 100 percentage points in comparison to TDMA-based analysis and up to 35 percentage points in comparison to FCFS-based analysis.

3. Varying Memory Access Demand: In this experiment, we vary the size of memory phases of each task τ_i by varying the memory access demand MD_i between $[5\%, 95\%] \times C_i$. We performed this experiment for different core utilizations (marked as Uti), i.e., 20%, 30%, and 40%. The results are shown in Figure 8



(a) Varying MD for Core Utilization of 0.20 (b) Varying MD for Core Utilization of 0.30 (c) Varying MD for Core Utilization of 0.40

Figure 8: Varying Memory Access Demand (MD)



(a) Task Periods in range of 100 to 500 (b) Task Periods in range of 100 to 1000 (c) Task Periods in range of 1000 to 5000

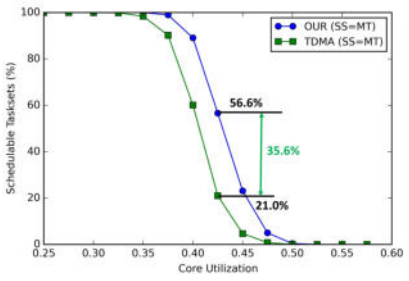
Figure 9: Varying the Tasks' Period Range and Core Utilization

where the x-axis represents memory access demand (MD) of tasks and the y-axis represents percentage of schedulable tasksets.

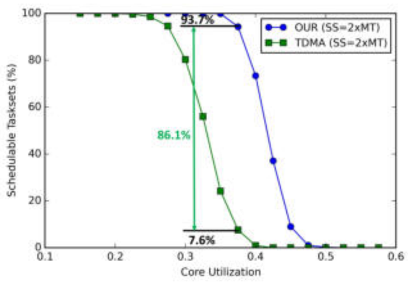
We can see in Figure 8 that by increasing the memory access demand of tasks the schedulability of all the approaches decreases. This is intuitive, since higher values of MD also increase the length of memory phases, which in turn increases the number of memory requests that can be generated by tasks. Consequently, tasks will suffer more bus contention for higher values of MD. Looking at the results, one can observe that our proposed analysis still outperforms the existing analysis for all values of MD at different core utilizations.

4. Varying Task Period Range: In this experiment, we varied tasks' period ranges from [100,500], [100,1000], to [1000,5000], along with core utilization, and plot the percentage of schedulable tasksets. The results are shown in Figure 9. One can see in Figure 9 that the FCFS analysis [12] is most sensitive to task periods and its performance decreases as the period range increases. This is mainly because the FCFS analysis [12] only considers the number of memory phases of tasks to bound bus contention. So, larger task periods can generate larger values of C_i, which translates in to higher values for C_i^A, C_i^R. This results in increasing the bus contention tasks suffer under the FCFS analysis [12]. Contrary to the FCFS analysis, the performance of the TDMA analysis improves by increasing the task period ranges, i.e., [100,1000] and [1000,5000]. Intuitively, this is because larger periods may generate larger WCET and therefore, larger number of memory requests generated during the memory phases. In such a case, it is possible that the local core may perform large number of memory requests during its assigned bus slots that may result in better utilization of the bus slots assigned to the local core. Our proposed work is less impacted by varying the task periods and outperforms the analyses of TDMA and FCFS for the all the task period ranges, as shown in Figure 9. Intuitively, this is due to the work-conserving nature of the RR-based arbitration policy and fine-grained bus contention analysis proposed in this paper.

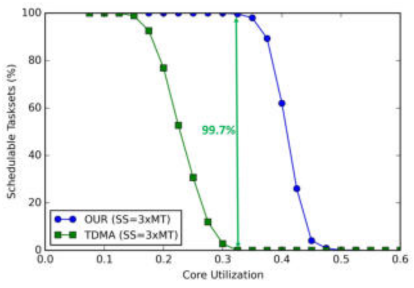
5. Varying Slot Size: In this experiment, we vary the value of SS (i.e., slot size) along with core utilization and evaluate their impact on task set schedulability. The results are shown in Figure 10. We can see in the figure that for a smaller value of slot size SS the task set schedulability for both RR and TDMA-based anlysis is higher. This is mainly because for lower values of SS, the bus contention that can



(a) Slot Size of $SS = MT$



(b) Slot Size of $SS = 2 \times MT$



(c) Slot Size of $SS = 3 \times MT$

Figure 10: Varying the Slot Size and Core Utilization

be generated by remote cores is also lower and the slots can be better utilized. For higher values of SS, the task set schedulability decreases due to an increase in bus contention that can be generated due to remote cores. For instance, if the $SS = 3MT$ and the local core has to execute only one memory request, then, in the worst-case, the local core may have to wait $3MT \times m - 1$ time units to access the bus. However, we can also see in Figure 10 that our proposed work is less impacted by the increase in slot size since it is based on the work-conserving RR bus arbitration policy in contrast to the TDMA-based analysis of [8].

Discussion: In all the experiments, we observe that the proposed analysis was able to perform significantly better than existing TDMA analysis of [8]. This result is intuitive as TDMA is non-work-conserving bus arbitration policy and may over estimate the bus contention that can be suffered by the tasks. Similarly, the proposed analysis was able to perform better than FCFS-based analysis under all the settings. However, we can see a difference between the RR and the existing FCFS based analysis only for a narrow range of utilizations due to the overall lower schedulability of the generated task sets under the default configuration, i.e., 4 cores, 32 tasks, tasks' periods between 100 to 500. For example, we can see in Figure 6, that even for a perfect bus (i.e., marked as No Bus Contention), the utilization range for which the number of schedulable tasksets are less than 100% and more than 0% is quite small. This is because of a higher interference between tasks contending for the bus. Understandably, the range of utilization increase as we decrease the number of tasks/cores (Figure 7a) or increase the period range (Figure 9c).

The only downside of the proposed RR analysis in comparison to [8] and [12] is a slight increase in computational complexity. The FCFS analysis is based on the analysis of memory phases, whereas the proposed RR analysis also considers the number of memory requests performed by tasks during the bus slots. Similarly, the proposed RR-based analysis also accurately estimate the blocking due to lower priority tasks which leads to a longer analysis time in comparison to existing FCFS and TDMA based analysis.

8. Related Work

Several works in the state-of-the-art have focused on the problem of bus contention[5] in multicore platforms considering partitioned fixed-priority scheduling [2, 3, 4, 1, 5, 6, 7, 9]. Dasari et al. proposed several works [5, 6, 7] that bound the memory bus contention in multicore platforms assuming FPNP scheduling. The work in [5] computes the WCRT of tasks considering a non-specified work-conserving bus arbiter. The work in [6] computes the bus-contention aware WCET of tasks considering a multicore platform that uses a RR-based bus arbitration policy. Dasari et al. later extended the work in [5, 6] to [7] that provides a general framework for memory bus-contention analysis. All of the above-mentioned works focus on the problem of bus contention considering a generic task model, i.e., tasks can generate bus/memory accesses anytime during their execution. Hence, their solutions are not directly applicable to the 3-phase execution model studied in this work.

[5]Some existing works use the term shared resource contention, memory bus contention, Front-Side-Bus (FSB) contention, etc., which is same as the term bus contention used in this paper.

The concept of phased execution model was introduced in the form of PRedictable Execution Model (PREM) [13] in which the execution of each task is divided into a memory phase and an execution phase. The concept of PREM was extended to the 3-phase task execution model [14, 15] in which the execution of each task is divided into three phases. Since the tasks can only issue the memory requests during their memory phases, it is possible to obtain a tighter bound on the bus/memory contention that can be suffered by tasks running on a multicore platform. Several existing works [8, 33, 34, 10, 16, 17, 20, 19, 12, 35] focus on the problem of the bus/memory contention suffered by the phased execution model. The memory-centric scheduling of PREM tasks using fixed-priority partitioned scheduling was proposed in [33]. The main idea of memory-centric scheduling [33] is to use TDMA bus slots for all the cores and allow memory promotion at the core level, i.e., memory phases can preempt the computation phases running on the same core to better utilize the TDMA slots. The memory-centric scheduling is extended in [35] that focuses on the fixed-processor priority-based bus/memory arbitration policy. Unlike [33], the approach in [35] introduces the concept of global memory preemption, i.e., the memory phases running on the higher-priority processors can preempt the memory phases running on the lower-priority processors.

Casini et al. [19] proposed the memory-contention aware WCRT analysis for the 3-phase task model using fixed-priority partitioned scheduling. Even though the solution provided in [19] is important, it assumes an architecture with crossbar switch that is responsible for point-to-point communication between the cores and the main memory. Consequently, the work proposed in [19] may not be applicable to multicore architectures that use a shared system bus. Maia et al. [10] presented a bus-contention analysis for the 3-phase task model using fixed-priority global scheduling. Since the focus of the work of [10] is global scheduling, it does not address the problem for fixed-priority partitioned scheduling perspective.

Schranzhofer et al. [8] proposed the bus contention analysis considering various task models including the 3-phase task model (referred to as the dedicated phase model in [8]). The work in [8] assumes TDMA-based non-work-conserving bus arbitration policy. The results in [8] show that the 3-phase task model outperforms all the other task models e.g., the general access model studied in [8]. Arora et. al [12] presented a fine-grained bus-contention analysis for the 3-phase task model using fixed-priority partitioned scheduling. The work in [12] assumes an FCFS-based bus arbitration scheme.

To the best of our knowledge, no existing work focuses on the bus-contention analysis for the 3-phase task model considering the fixed-priority partitioned scheduling and RR-based bus arbitration policy. The similar works that can be compared against the proposed analysis in this paper are the analysis of [8] for TDMA and [12] for FCFS because they provide a bus-contention analysis for the 3-phase task model using fixed-priority partitioned non-preemptive scheduling.

9. Conclusion

In this work, we propose the bus-contention analysis for the 3-phase task model considering partitioned fixed-priority non-preemptive scheduling and RR bus arbitration scheme. We show that a fairer work-conserving bus arbitration scheme such as RR can significantly reduce the bus contention of 3-phase tasks executing on a multicore platform in comparison to FCFS and TDMA based bus arbitration scheme. We present a fine-grained analysis to bound the maximum bus contention of 3-phase tasks under a RR-based bus arbitration policy. We also propose an algorithm to to soundly estimate the blocking that can be suffered by tasks due to lower priority tasks running on the same core. The bus-contention aware schedulability analysis is then formulated for the 3-phase task model using fixed-priority partitioned scheduling by additionally integrating the impact of the bus-contention and blocking from lower-priority tasks running on the same core. The extensive experimental evaluation reveals that the proposed work can schedule up to 100% more tasksets as compared to TDMA-based analysis [8] and up to 40% more tasksets as compared to FCFS-based analysis [12]. As future work, we would like to extend our work by considering priority based bus arbitration schemes, e.g., task priority based bus arbitration scheme and processor priority bases bus arbitration scheme.

Acknowledgement

This work was partially supported by National Funds through FCT/MCTES (Portuguese Foundation for Science and Technology), within the CISTER Research Unit (UIDB-UIDP/04234/2020); also by the Operational Competitiveness Programme and Internationalization (COMPETE 2020) under the PT2020 Partnership Agreement, through the European Regional Development Fund (ERDF), and by national funds through the FCT, within project POCI-01-0145-FEDER-029119 (PREFECT); also by the European Union's Horizon 2020 - The EU Framework Programme for Research and Innovation 2014-2020, under grant agreement No. 732505. Project "TEC4Growth - Pervasive Intelligence, Enhancers and Proofs of Concept with Industrial Impact/NORTE-01-0145-FEDER000020" financed by the North Portugal Regional Operational Programme (NORTE 2020), under the PORTUGAL 2020 Partnership Agreement; also by FCT, under PhD grant 2020.09532.BD.

References

[1] S. Schliecker, R. Ernst, Real-time performance analysis of multiprocessor systems with shared memory, ACM Transactions on Embedded Computing Systems 10 (2) (2010) 1–27. doi:10.1145/1880050.1880058.
URL http://portal.acm.org/citation.cfm?doid=1880050.1880058

[2] J. Rosen, A. Andrei, P. Eles, Z. Peng, Bus access optimization for predictable implementation of real-time applications on multiprocessor systems-on-chip, in: 28th IEEE International Real-Time Systems Symposium (RTSS 2007), 2007, pp. 49–60.

[3] S. Chattopadhyay, A. Roychoudhury, T. Mitra, Modeling shared cache and bus in multi-cores for timing analysis, in: Proceedings of the 13th International Workshop on Software & Compilers for Embedded Systems, SCOPES '10, Association for Computing Machinery, New York, NY, USA, 2010. doi:10.1145/1811212.1811220.
URL https://doi.org/10.1145/1811212.1811220

[4] T. Kelter, H. Falk, P. Marwedel, S. Chattopadhyay, A. Roychoudhury, Bus-aware multicore wcet analysis through tdma offset bounds, in: 2011 23rd Euromicro Conference on Real-Time Systems, 2011, pp. 3–12.

[5] D. Dasari, B. Andersson, V. Nelis, S. M. Petters, A. Easwaran, J. Lee, Response time analysis of cots-based multicores considering the contention on the shared memory bus, in: 2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications, 2011, pp. 1068–1075.

[6] D. Dasari, V. Nelis, An analysis of the impact of bus contention on the wcet in multicores, in: 2012 IEEE 14th International Conference on High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems, 2012, pp. 1450–1457. doi:10.1109/HPCC.2012.212.

[7] D. Dasari, V. Nelis, B. Akesson, A framework for memory contention analysis in multi-core platforms, Real-Time Systems 52 (06 2015). doi:10.1007/s11241-015-9229-9.

[8] A. Schranzhofer, J.-J. Chen, L. Thiele, Timing analysis for tdma arbitration in resource sharing systems, in: 2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium, 2010, pp. 215–224. doi:10.1109/RTAS.2010.24.

[9] S. A. Rashid, G. Nelissen, E. Tovar, Cache persistence-aware memory bus contention analysis for multicore systems, in: 2020 Design, Automation Test in Europe Conference Exhibition (DATE), 2020, pp. 442–447. doi:10.23919/DATE48585.2020.9116265.

[10] C. Maia, G. Nelissen, L. Nogueira, L. M. Pinho, D. G. Perez, Schedulability analysis for global fixed-priority scheduling of the 3-phase task model, in: 2017 IEEE 23rd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), IEEE, Hsinchu, Taiwan, 2017, pp. 1–10. doi:10.1109/RTCSA.2017.8046313.
URL http://ieeexplore.ieee.org/document/8046313/

[11] R. I. Davis, S. Altmeyer, L. S. Indrusiak, C. M. and·Vincent Nelis, J. Reineke, An extensible framework for multicore response time analysis, Real-Time Systems (July 2017).

[12] J. Arora, C. Maia, S. Aftab Rashid, G. Nelissen, E. Tovar, Bus-contention aware schedulability analysis for the 3-phase task model with partitioned scheduling, in: 29th International Conference on Real-Time Networks and Systems, RTNS'2021, Association for Computing Machinery, New York, NY, USA, 2021, p. 123–133. doi:10.1145/3453417.3453433.
URL https://doi.org/10.1145/3453417.3453433

[13] R. Pellizzoni, E. Betti, S. Bak, G. Yao, J. Criswell, M. Caccamo, R. Kegley, A Predictable Execution Model for COTS-Based Embedded Systems, in: 2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium, IEEE, Chicago, IL, USA, 2011, pp. 269–279. doi:10.1109/RTAS.2011.33.
URL http://ieeexplore.ieee.org/document/5767117/

[14] G. Durrieu, M. Faugère, S. Girbal, D. Gracia Pérez, C. Pagetti, W. Puffitsch, Predictable Flight Management System Implementation on a Multicore Processor, in: Embedded Real Time Software (ERTS'14), TOULOUSE, France, 2014.
URL https://hal.archives-ouvertes.fr/hal-01121700

[15] C. Maia, L. Nogueira, L. M. Pinho, D. G. Perez, A closer look into the AER Model, in: 2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA), IEEE, Berlin, Germany, 2016, pp. 1–8. doi:10.1109/ETFA.2016.7733567.
URL http://ieeexplore.ieee.org/document/7733567/

[16] R. Tabish, R. Mancuso, S. Wasly, A. Alhammad, S. S. Phatak, R. Pellizzoni, M. Caccamo, A real-time scratchpad-centric os for multi-core embedded systems, in: 2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), 2016, pp. 1–11. doi:10.1109/RTAS.2016.7461321.

[17] R. Tabish, R. Mancuso, S. Wasly, R. Pellizzoni, M. Caccamo, A real-time scratchpad-centric os with predictable inter/intra-core communication for multi-core embedded systems, Real-Time Systems 55 (10 2019). doi:10.1007/s11241-019-09340-0.

[18] M. Soliman, G. Gracioli, R. Tabish, R. Pellizzoni, M. Caccamo, Segment streaming for the three-phase execution model: Design and implementation, 2019. doi:10.1109/RTSS46320.2019.00032.

[19] D. Casini, A. Biondi, G. Nelissen, G. Buttazzo, A holistic memory contention analysis for parallel real-time tasks under partitioned scheduling, in: 2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), 2020, pp. 239–252. doi:10.1109/RTAS48715.2020.000-3.

[20] C. Pagetti, J. Forget, H. Falk, D. Oehlert, A. Luppold, Automated generation of time-predictable executables on multi-core, in: RTNS 2018, Proceedings of the 26th International Conference on Real-Time Networks and Systems, POITIERS, France, 2018.
URL https://hal.archives-ouvertes.fr/hal-01888728

[21] F. Fort, J. Forget, Code generation for multi-phase tasks on multicore with distributed memory, in: JRWRTC 2018, 2018.

[22] G. Gracioli, R. Tabish, R. Mancuso, r. mirosanlou, R. Pellizzoni, M. Caccamo, Designing mixed criticality applications on modern heterogeneous mpsoc platforms, 2019.

[23] S. Park, M.-Y. Kwon, H.-K. Kim, H. Kim, Execution model to reduce the interference of shared memory in arinc 653 compliant multicore rtos, Applied Sciences 10 (7) (2020). doi:10.3390/app10072464.
URL https://www.mdpi.com/2076-3417/10/7/2464

[24] C. L. Liu, J. W. Layland, Scheduling algorithms for multiprogramming in a hard-real-time environment, J. ACM 20 (1) (1973) 46–61. doi:10.1145/321738.321743.
URL https://doi.org/10.1145/321738.321743

[25] R. Pellizzoni, M. Caccamo, Toward the predictable integration of real-time cots based systems, in: Proceedings of the 28th IEEE International Real-Time Systems Symposium, RTSS '07, IEEE Computer Society, USA, 2007, p. 73–82. doi:10.1109/RTSS.2007.51.
URL https://doi.org/10.1109/RTSS.2007.51

[26] M. Paolieri, E. Quiñones, F. Cazorla, G. Bernat, M. Valero, Hardware support for wcet analysis of hard real-time multicore systems, Vol. 37, 2009, pp. 57–68. doi:10.1145/1555754.1555764.

[27] G. Yao, H. Yun, Z. P. Wu, R. Pellizzoni, M. Caccamo, L. Sha, Schedulability analysis for memory bandwidth regulated multicore real-time systems, IEEE Transactions on Computers 65 (2) (2016) 601–614. doi:10.1109/TC.2015.2425874.

[28] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, P. Stenström, The worst-case execution-time problem—overview of methods and survey of tools, ACM Trans. Embed. Comput. Syst. 7 (3) (May 2008). doi:10.1145/1347375.1347389.
URL https://doi.org/10.1145/1347375.1347389

[29] R. J. Bril, J. J. Lukkien, W. F. J. Verhaegh, Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption revisited, in: 19th Euromicro Conference on Real-Time Systems (ECRTS'07), 2007, pp. 269–279.

[30] J. Lehoczky, Fixed priority scheduling of periodic task sets with arbitrary deadlines, [1990] Proceedings 11th Real-Time Systems Symposium (1990) 201–209.

[31] M. Negrean, R. Ernst, Response-time analysis for non-preemptive scheduling in multi-core systems with shared resources, in: 7th IEEE International Symposium on Industrial Embedded Systems (SIES'12), 2012, pp. 191–200. doi:10.1109/SIES.2012.6356585.

[32] P. Emberson, R. Stafford, R. Davis, Techniques for the synthesis of multiprocessor tasksets, WATERS'10 (01 2010).

[33] G. Yao, R. Pellizzoni, S. Bak, E. Betti, M. Caccamo, Memory-centric scheduling for multicore hard real-time systems, Real-Time Systems 48 (11 2012). doi:10.1007/s11241-012-9158-9.

[34] A. Alhammad, R. Pellizzoni, Schedulability analysis of global memory-predictable scheduling, in: 2014 International Conference on Embedded Software (EMSOFT), 2014, pp. 1–10. doi:10.1145/2656045.2656070.

[35] G. Schwäricke, T. Kloda, G. Gracioli, M. Bertogna, M. Caccamo, Fixed-priority memory-centric scheduler for cots-based multiprocessors, in: M. Volp (Ed.), 32nd Euromicro Conference on Real-Time Systems, ECRTS 2020, Leibniz International Proceedings in Informatics, LIPIcs, Schloss Dagstuhl- Leibniz-Zentrum fur Informatik GmbH, Dagstuhl Publishing, 2020, 32nd Euromicro Conference on Real-Time Systems, ECRTS 2020 ; Conference date: 07-07-2020 Through 10-07-2020. doi:10.4230/LIPIcs.ECRTS.2020.1.