CISTER
Research Centre in
Real-Time & Embedded
Computing Systems

Conference Paper

Bus-Contention Aware Schedulability Analysis for the 3-Phase Task Model with Partitioned Scheduling

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

*CISTER Research Centre

CISTER-TR-210206

2021/04/07

Bus-Contention Aware Schedulability Analysis for the 3-Phase Task Model with Partitioned Scheduling

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

Multicore platforms are being increasingly adopted in Cyber-Physical Systems (CPS) due to their advantages over single-core processors, such as raw computing power and energy efficiency. Typically, multicore platforms use a shared system bus that connects the cores to the memory hierarchy (including caches and main memory). However, such hierarchy causes tasks running on different cores to compete for access to the shared system bus whenever data reads or writes need to be made. Such competition is problematic as it may cause large variations in the execution time of tasks in a non-deterministic way. This paper presents an approach that allows one to derive the worst-case response-time of tasks that follow the 3-phase task model executing under partitioned scheduling. Experiments on synthetic task sets were performed to evaluate the effectiveness of the proposed analysis in comparison to state-of-the-art. The experimental results reveal an increase of up to 34 percentage points of schedulable task sets in comparison to the state-of-the-art.

Bus-Contention Aware Schedulability Analysis for the 3-Phase Task Model with Partitioned Scheduling

Jatin Arora
CISTER Research Centre, ISEP, IPP
Porto, Portugal
jatin@isep.ipp.pt

Cláudio Maia
CISTER Research Centre, ISEP, IPP
Porto, Portugal
crrm@isep.ipp.pt

Syed Aftab Rashid
CISTER Research Centre, ISEP, IPP
Porto, Portugal
syara@isep.ipp.pt

Geoffrey Nelissen
Eindhoven University of Technology
Eindhoven, the Netherlands
g.r.r.j.p.nelissen@tue.nl

Eduardo Tovar
CISTER Research Centre, ISEP, IPP
Porto, Portugal
emt@isep.ipp.pt

ABSTRACT

Multicore platforms are being increasingly adopted in Cyber - Physical Systems (CPS) due to their advantages over single-core processors, such as raw computing power and energy efficiency. Typically, multicore platforms use a shared system bus that connects the cores to the memory hierarchy (including caches and main memory). However, such hierarchy causes tasks running on different cores to compete for access to the shared system bus whenever data reads or writes need to be made. Such competition is problematic as it may cause variations in the execution time of tasks in a non-deterministic way. This paper presents a partitioned scheduling based approach that allows one to derive bus contention-aware worst-case response-time of tasks that follow the 3-phase task model. Experiments on synthetic task sets were performed to evaluate the effectiveness of the proposed analysis in comparison to a state-of-the-art approach. The experimental results reveal an increase of up to 34 percentage points of schedulable task sets in comparison to the compared approach.

ACM Reference Format:
Jatin Arora, Cláudio Maia, Syed Aftab Rashid, Geoffrey Nelissen, and Eduardo Tovar. 2021. 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), April 7–9, 2021, NANTES, France.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3453417.3453433

1 INTRODUCTION

Multicore processors offer several advantages over the traditional single-core computing platforms such as higher computational power and lower energy consumption, among others. However, the use of multicore processors in *hard* real-time systems, i.e., systems with stringent timing requirements, is still under scrutiny of the real-time systems community due to their *unpredictable* behavior. This behavior is a direct result of current designs which include shared resources such as a system bus, caches, main memory and I/O devices. When accessing any of these shared resources, a task running on a given core may suffer *inter-core interference* from co-running tasks, i.e., tasks running on the other cores, hence causing non-deterministic variations in the tasks' execution times.

Solutions that use *phased* execution models [1, 2, 10, 14, 15, 17] are promising candidates to circumvent the problem of inter-core interference due to bus contention. In these models, tasks' executions are divided into separate memory and execution phases. The memory phases are responsible for loading tasks' data and instructions into a core's local memory (e.g., cache or scratchpad) and to push back the processed data into the main memory. During the execution phase, a core executes the task's code by processing data/instructions already available in the core's local memory without any need to access the system bus or the main memory.

In this work, we focus on the 3-phase (or AER) task model [10, 15] which is a generalization of the PREM model. The PRedictable Execution Model (PREM) [17] uses one memory and execution phase. In the 3-phase model, each task's execution is divided into three phases, namely, *Acquisition* (A), *Execution* (E), and *Restitution* (R). During the acquisition phase (also called A-phase), task's data/instructions are loaded from the main memory into the core's local memory. During the execution phase (also called E-phase), pre-loaded data/instructions are executed by the core, and in the restitution phase (i.e., R-phase), the processed data are written back to the main memory. Similarly to the PREM model, in the 3-phase model, accesses to the main memory via a system bus are only performed during the memory phases, i.e., A and R phases.

Phased execution models such as PREM and the 3-phase task model take advantage of the fact that a task running on one core can execute a memory phase while other cores are executing E-phases to reduce inter-core interference. Unfortunately, even using these models, tasks may compete to access the shared bus to execute their memory phases. This situation happens when a task tries to access the bus to load its data/instructions from/to the main memory and the bus is already busy serving the memory phase of another task executing on a different core. Such situation forces the requesting task to hold its execution until the bus is free. In this work, this phenomenon is referred to as *bus blocking*. Since bus blocking can significantly impact task schedulability, even under phased execution models, works like [14] have been proposed to bound

RTNS'2021, April 7–9, 2021, NANTES, France

ACM ISBN 978-1-4503-9001-9/21/04…$15.00
https://doi.org/10.1145/3453417.3453433

the maximum bus blocking under a *global* scheduling approach. Contrary to [14], in this work we focus on analyzing bus contention and deriving the worst-case response time (WCRT) for the 3-phase task model assuming fixed-priority *partitioned* scheduling. Instead of bounding the bus blocking suffered by each task, we focus on bounding the total bus blocking that may be experienced by a sequence of jobs executing on the same core. To do so, we derive the maximum number of jobs released on the other cores that can cause bus blocking to the task under analysis in any time interval of a given length. This allows us to identify the scenarios that may lead to the maximum bus blocking and compute the WCRT of the task under analysis.

Contributions: This paper has the following contributions: (1) We propose a fine-grained analysis to compute the maximum bus blocking for fixed-priority 3-phase tasks executing under a partitioned scheduling policy. Instead of analyzing the bus blocking that can be suffered/caused by individual tasks, we focus on bounding the bus blocking suffered/caused at core level. This allows us to have a tighter bound on the bus blocking because different execution scenarios are considered in the analysis. (2) We derive a schedulability test for fixed-priority 3-phase tasks scheduled under a non-preemptive partitioned scheduling approach by integrating the impact of maximum bus blocking into the WCRT analysis of each task; (3) We perform extensive empirical evaluation under different settings to show the effectiveness of the proposed approach.

Paper Organization: The rest of the paper is organized as follows: Section 2 describes the system and execution models. Section 3 presents the background concepts. Section 4 discusses the analysis for the maximum bus blocking and the computation of the level-i busy window by integrating the additional delay due to the bus blocking. The schedulability analysis is presented in Section 5. The experimental results are presented in Section 6. Section 7 presents the related work. Finally, Section 8 concludes the paper.

2 SYSTEM MODEL

We consider a multicore platform with m identical cores ($\pi_1, \pi_2, \ldots, \pi_m$) where each core has a local memory (i.e., scratchpad or local cache) that can store the tasks' data/instructions during runtime. In addition, each core uses a *shared system bus* to access the main memory and cores can access the bus in a concurrent manner which, as explained previously, may lead to contention. We assume that the bus can only handle one memory phase[1] at a time and while doing so, it can not be preempted. Once the on-going memory phase is completed, the bus is then assigned to the next requesting task. Consequently, only one task can access the main memory at a time. Furthermore, we assume that the system bus arbitration policy is First-Come First-Served (FCFS). As for task scheduling, we assume partitioned scheduling is used where the task to core mapping is given at design time and a fixed task-priority algorithm such as Rate Monotonic [13] is used to assign task priorities.

2.1 Task Model

We consider a task set Γ comprising n sporadic tasks from a which a subset of n' tasks is assigned to each core according to the given task to core mapping strategy. Each task τ_i is characterized by minimum inter-arrival time T_i and constrained deadline D_i, where $D_i \leq T_i$. Each task τ_i is executed according to the 3-phase task model. In this model, the execution of a task τ_i is divided into three phases, namely: *Acquisition* (A), *Execution* (E) and *Restitution* (R). The worst-case execution time (WCET) of A, E and R-phases of τ_i is denoted by C_i^A, C_i^E, and C_i^R, respectively. Thus, the WCET of task τ_i in *isolation* is given by the sum of the WCET of each of the phases, i.e., $C_i = C_i^A + C_i^E + C_i^R$. The task utilization of task τ_i is given by $U_i = \frac{C_i}{T_i}$ and the *core utilization* is given by $\sum_{i=1}^{n'} U_i$. The bus utilization of task τ_i is given by $BU_i = \frac{C_i^A + C_i^R}{T_i}$ and the *total bus utilization* is given by $\sum_{i=1}^{n} \frac{C_i^A + C_i^R}{T_i}$. 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 system is not schedulable if the core utilization of any core is greater than 1. Moreover, the system is not schedulable if the total bus utilization is greater than 1 as it is assumed that system bus will be saturated if the total bus utilization is greater than 1.

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

2.2 Execution Model

In the 3-phase task model, the A-phase executes first to fetch data from the main memory and store it in the core's local memory. Then, the E-phase executes the task's code using the data previously fetched by the A-phase. Finally, the R-phase writes the modified data, resulting from the E-phase execution, to the main memory. Thus, the A-phase and R-phase are memory phases in which the system bus is accessed to read/write data from/to main memory. Each task executes non-preemptively, i.e., once a task starts executing its A-phase, it cannot be preempted by any other task until the completion of its R-phase. It is also assumed that a core remains idle during the execution of a memory phase. Similarly, the bus handles memory phases in a non-preemptive fashion, i.e., if a bus is handling the memory requests of a memory phase, the memory phase cannot be preempted[2] until completion.

Each core maintains its own *ready queue* with tasks that are ready to execute, sorted by task priority. Whenever a task in the queue become ready to execute, the core requests access to the system bus and if the system bus is free, the core executes the A-phase of that task. However, if the system bus is busy serving a memory phase from another core, then the core will busy-wait until the bus becomes available, at which point it will execute the A-phase of the task with highest priority in the ready queue. Once the A-phase of a task completes, the E-phase of the same task starts executing immediately on the core. After the E-phase completes, the task requests access to the bus to execute its R-phase. At this point, the core may have to busy-wait for the bus if the bus is busy serving requests of co-running tasks. Once the bus becomes available, the task can execute its R-phase and finalize its execution.

[1] A memory phase, e.g., A or R, may comprise multiple memory requests.

[2] This is different from for example the work in [22] that allows global memory preemptions during a memory phase.

Note that under the considered execution model, a lower priority task τ_j running on the same core can only cause blocking to a higher priority task τ_i, if τ_j starts executing before τ_i.

We assume that if there are other tasks waiting in the core's ready-queue, the A-phase of another ready task of the same core executes immediately once the R-phase of the currently executing task completes and without requiring explicit access to the bus. This is done to avoid blocking during this transition of phases/tasks.

When more than one core requests access to the system bus simultaneously, it is assumed that in the worst-case, the core under analysis accesses the bus after the completion of the bus requests of all the $m-1$ cores (the request of the core under analysis is the last to arrive according to the FCFS policy). We understand that this assumption can be pessimistic and may negatively impact schedulability. Consequently, we plan to address other bus arbitration policies e.g., Round Robin, TDMA, processor priority, etc. in a future work.

3 BACKGROUND

In this section, we briefly discuss the Worst-Case Response Time (WCRT) analysis of Fixed-Priority Non-Preemptive (FPNP) scheduling for single-core systems. This is essential as we later use this analysis to build our proposed bus contention-aware WCRT analysis for multicore systems.

For **single-core platforms** that use **FPNP scheduling**, the WCRT of a task τ_i is observed in the longest level-i busy window [3]. The level-i busy window is defined as follows.

Definition 3.1. Level-i busy window: 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.

For any task τ_i executing under FPNP scheduling on a single core processor, the longest level-i busy window is computed by bounding the maximum *blocking* and the maximum *interference* τ_i may suffer due to tasks executing on the same core.

In FPNP scheduling, only one job of a lower priority task in lp_i can block the execution of task τ_i [3, 25]. Consequently, τ_i suffers maximum blocking if that job has the maximum execution time over all tasks in lp_i. We denote this term by $C^{max}_{lp_i}$ and its computation is given by:

$$C^{max}_{lp_i} = \max_{\tau_j \in lp_i} \{C_j\} \tag{1}$$

All jobs of task τ_i can suffer interference from all higher or equal priority tasks in hep_i (including own jobs of τ_i) that are executing on the same core in the level-i busy window. Consequently, the maximum interference τ_i may suffer due to tasks in hep_i depends on the maximum number of jobs released by all tasks in hep_i in the level-i busy window. Several different methods have been proposed in the literature to bound the maximum number of jobs of any task τ_h that may interfere with the execution of task τ_i. The use of event arrival curves is one such technique proposed in [21].

When using event arrival curves, the upper event arrival function $\eta^+(\Delta)$ is used to denote the maximum number of events that can occur in an event stream in any time interval of length Δ. Using the same concept, each job released by a task $\tau_h \in hep_i$ can be considered as an event. Consequently, the maximum number of jobs released by task τ_h in any time interval of length Δ is given by $\eta^+_h(\Delta)$ and the maximum interference task τ_i can suffer due to those jobs is upper bounded by $\eta^+_h(\Delta) \times C_h$.

Using the upper bounds on the maximum blocking and maximum interference task τ_i can suffer, the length of the longest level-i busy window W_i is given by the first fixed-point solution of the following equation:

$$W_i = C^{max}_{lp_i} + \sum_{\tau_h \in hep_i} (\eta^+_h(W_i) \times C_h) \tag{2}$$

where $\eta^+_h(W_i)$ gives the maximum number of jobs released by any task $\tau_h \in hep_i$ in any time window of length W_i, and C_h is WCET of task τ_h in isolation[3].

Having computed the length of the longest level-i busy window W_i using Equation 2, the maximum number of jobs of task τ_i that can execute within W_i is given by:

$$K_i = \eta^+_i(W_i) \tag{3}$$

Under FPNP scheduling, the WCRT of task τ_i is computed by computing the response time of each job of τ_i that executes within W_i. To compute the response time of any job of task τ_i that executes within W_i, we must first compute the latest start time of that job because once that job starts executing, it cannot be preempted by any other job executing on the same core.

Let $\tau_{i,k}$ be the k^{th} job of task τ_i executing during W_i, then the latest start time $s_{i,k}$ of $\tau_{i,k}$ is given by:

$$s_{i,k} = C^{max}_{lp_i} + (k-1) \times C_i + \sum_{h \in hep_i \setminus \tau_i} \eta^+_h(s_{i,k}) \times C_h \tag{4}$$

where $C^{max}_{lp_i}$ is given in Equation 1, $\sum_{h \in hep_i \setminus \tau_i} \eta^+_h(s_{i,k}) \times C_h$ captures the maximum interference suffered by $\tau_{i,k}$ from hep_i task set (excluding τ_i) in a time window of length $s_{i,k}$ and $(k-1) \times C_i$ accounts for the execution time of previous jobs of task τ_i.

As $s_{i,k}$ appears on the both sides in Equation 4, it can be solved iteratively by initializing $s_{i,k} = C^{max}_{lp_i} + \sum_{h \in hep_i \setminus \tau_i} C_h$.

The latest start time of k^{th} job of τ_i will then be given by the smallest value of $s_{i,k}$ for which Equation 4 converges.

Once the latest start time of $\tau_{i,k}$ is computed, the response time $R_{i,k}$ can then be simply computed by adding to it the WCET C_i of task τ_i, i.e.,

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

Finally, the WCRT of task τ_i is computed by maximizing Equation 5 over all the jobs of τ_i that can execute in the level-i busy window, i.e., from 1 to K_i,

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

4 BUSY WINDOW COMPUTATION

In this section, we explain how to compute the longest level-i busy window executing on a given core π_l for the task model presented in Section 2. To do so, we must first compute the maximum *interference* and maximum *blocking* suffered by τ_i from tasks running on the same core.

[3] $\eta^+_h(W_i) \times C_h$ in Equation 2 is similar to in Equation 3 in [21], simply replacing the set $hp(i)$ by the set hep_i.

However, as explained earlier in the introduction, in a multicore system multiple tasks may execute in parallel with τ_i. Hence, the longest level-i busy window of a task τ_i does not only depend on the tasks running on the same core but also on the tasks running on the other cores due to the bus blocking. In this work, the core whose tasks are under analysis is referred to as the local core (denoted as π_l) whereas a core whose tasks can cause bus blocking is referred to as a remote core (usually denoted as π_r).

The jobs running on the local core π_l in the longest level-i busy window can suffer bus blocking from tasks executing on remote cores when accessing the shared system bus to read/write the data from/to the main memory. Bus blocking can increase the length of the level-i busy window which may lead to an increase in the WCRT of task τ_i. Therefore, when computing the length of level-i busy window of a task τ_i executing on a multi-core processor, it is essential to consider the effect of bus blocking. To address this effect, we extend Equation 2 by integrating the additional delay due to bus blocking in order to compute the longest level-i busy window.

Let $W_{i,l}$ be the length of the level-i busy window w.r.t a task τ_i executing on core π_l, where $W_{i,l}$ is given by the first fixed-point solution of the following equation:

$$W_{i,l} = C^{max}_{lp_{i,l}} + Bus^{max}_{i,l}(W_{i,l}) + \sum_{\tau_h \in hep_{i,l}} (\eta^+_h(W_{i,l}) \times C_h) \quad (7)$$

where $C^{max}_{lp_{i,l}}$ is the maximum blocking caused by a job from an lower priority task in $lp_{i,l}$ on τ_i, and $\sum_{\tau_h \in hep_{i,l}}(\eta^+_h(W_{i,l}) \times C_h)$ is the maximum interference caused to τ_i by the tasks in $hep_{i,l}$ w.r.t core π_l. These are computed in a similar manner as in Equation 2. In Equation 7, $Bus^{max}_{i,l}(W_{i,l})$ denotes the maximum bus blocking suffered by tasks running on core π_l from all co-running tasks executing on remote cores in any time window of length $W_{i,l}$.

Before explaining how the maximum bus blocking can be computed (in Section 4.2), we will first derive important properties on the 3-phase task execution model that we later build upon to upper bound the bus blocking.

4.1 Useful Properties

The properties derived in this section are based on the 3-phase task execution model as explained earlier in Section 2.2.

Property 1 (P1): On each core, a job can suffer bus blocking only once, i.e., at its R-phase, when it executes immediately after any other job on the same core.

Proof. Recall from section 2.2, when a job running on a core completes its R-phase execution, the scheduler checks the status of the ready-queue of the same core. The status of the ready-queue can be either empty (if no other job is released on the same core) or non-empty (if at least one job on the same core is ready to execute). If the ready-queue of the same core is empty then the core will release the bus.

On the other hand, if the ready-queue of the same core is non-empty, there is at-least one job on the same core ready to execute. The scheduler gives the bus access to the A-phase of the job with highest priority among the jobs of the ready-queue of the same core. Thus, a core can execute R- and A-phases back to back which means that the A-phase does not suffer any bus blocking when it executes immediately after the R-phase of any other job of the same core.

Therefore, on each core, a job that executes immediately after another job of the same core can only suffer bus blocking once i.e., at its R-phase. □

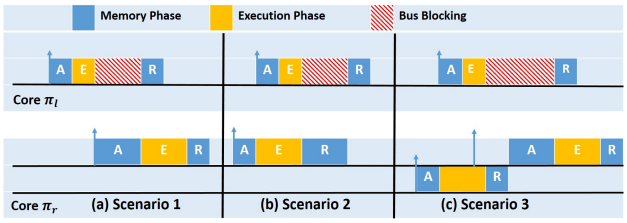

Figure 1: Bus blocking caused by a remote core for each bus blocking suffered at the local core

Property 2 (P2): For each bus blocking suffered by a job on a local core, a remote core can cause at most one bus blocking by either from a memory phase of one job (A or R-phase) or one R and one A phases of two different jobs running on that remote core.

Proof. When a job running on a local core requests access to the bus, the following scenarios are possible.
Scenario 1: A job running on the remote core is already executing its A-phase. Consequently, the job on the local core can only access the bus after the completion of the A-phase of the job currently executing on the remote core. Therefore, in this case, the bus blocking that can be caused by the remote core to a job running on the local core is equivalent to the execution time of the A-phase of the remote core's job. This scenario is depicted in Figure 1 (a).
Scenario 2: A job on the remote core is executing its R-phase and the remote core's ready queue is empty. In this case, bus blocking caused by the remote core to a job executing on the local core is equivalent to the execution time of the R-phase of remote core's job, e.g., see Figure 1(b).
Scenario 3: A job on the remote core is executing its R-phase and the remote core's ready queue is non-empty. Once the R-phase of the currently executing job is completed, the A-phase of the next job from the remote core's ready-queue will execute immediately. Thus, the bus will only be released after the execution of R+A phases of two different jobs of the remote core. In this case, the bus blocking caused by the remote core to one job of the local core is equivalent to the sum of the execution times of the R-phase and A-phase of two different jobs running on that remote core. See Figure 1 (c) for an example scenario.

Therefore, for each bus blocking suffered by a job on the local core, a remote core can cause at most one bus blocking by either memory phase of a job (A or R-phase) or by one R and one A-phase of two different jobs running on that remote core. □

Property 3 (P3): When a single job of remote core participates in one bus blocking, it can only participate by its A-phase or its R-phase.

Proof. Directly follows from Property 2. □

4.2 Bounding the Number of Bus Blockings

In case of a multicore system, all the jobs of the local core π_l that execute in the level-i busy window $W_{i,l}$ can suffer bus blocking from co-running tasks executing on remote cores. This can directly impact the length of the level-i busy window and the WCRT of the task under analysis τ_i.

As identified in [22], tasks may exhibit release jitter i.e., the time difference between the release of two consecutive jobs (i.e., A-phases) can be less than its minimum inter-arrival time. The release jitter of a task not only impacts the timing behaviour of the tasks running on the same core but it can also impact the timing behaviour of the tasks running on other cores. This is due to the fact that when tasks exhibit jitter, they can impose more memory demand that can eventually cause more bus blocking to the tasks running on other cores. While deriving the upper-bound on the bus blocking, it is necessary to integrate the impact of the release jitter of the tasks. In this work, we assume that the impact of release jitter is addressed by the upper event arrival function [21] used to bound the number of jobs that can be released by tasks in any time interval of a given length (e.g., see Equation 2).

To bound the maximum bus blocking suffered by tasks running on the local core π_l due to co-running tasks executing on a remote core π_r, we start by computing the following values:

- $N_{\pi_l}(W_{i,l})$, i.e., the maximum number of times *tasks running on the local core π_l can suffer bus blocking in a time window of length $W_{i,l}$*, and
- $N_{\pi_r}(W_{i,l})$, i.e., the maximum number of times *tasks running on the remote core π_r can cause bus blocking in $W_{i,l}$*.

Note that both values are needed in order to accurately upper-bound the bus blocking as its value can be derived by comparing $N_{\pi_l}(W_{i,l})$ against $N_{\pi_r}(W_{i,l})$. For instance, assume that core π_l can suffer bus blocking ten times in $W_{i,l}$ but core π_r can only cause bus blocking two times in the same time interval. In this case, the maximum number of bus blockings that tasks running on core π_l can suffer due to π_r in $W_{i,l}$ is upper bounded by the value of $N_{\pi_r}(W_{i,l})$, that is two.

The maximum number of times tasks running on the local core π_l (resp. remote core π_r) can suffer (resp. cause) bus blocking in a time window of length $W_{i,l}$ is dependent on the maximum number of memory phases released on the local (resp. remote) core in that time interval. Since a memory phase is a part of a job, values of $N_{\pi_l}(W_{i,l})$ and $N_{\pi_r}(W_{i,l})$ can be derived by considering the maximum number of jobs released on the local as well as remote cores in any time window of length $W_{i,l}$. This leads to the following lemmas.

Lemma 4.1. *The maximum number of times tasks running on a local core π_l can suffer bus blocking in a window of length $W_{i,l}$ is upper bounded by:*

$$N_{\pi_l}(W_{i,l}) = \sum_{\tau_h \in hep_{i,l}} \eta_h^+(W_{i,l}) + 1 \tag{8}$$

Proof. In the longest level-i busy window, all jobs that execute on the local core (except the first job) can only execute after the completion of the R-phase of a previous job on the same core. Since we assume that the A-phase of a task can start immediately after the completion of R-phase of a previous job, each job (except the first) will not suffer any bus blocking at its A-phase and thus can only suffer bus blocking once, i.e., before starting its R-phase (see Property P1). Consequently, the maximum number of bus blockings suffered by tasks running on a local core π_l in a time window of length $W_{i,l}$ is upper bounded by the maximum number of jobs released by tasks in $hep_{i,l}$ in $W_{i,l}$, i.e., $\sum_{\tau_h \in hep_{i,l}} \eta_h^+(W_{i,l})$.

To account for the additional 1 in Equation 8, two possible cases are considered.

Case 1: If the first job in the busy window is a job from a lower priority task, then it can only suffer bus blocking at its R-phase as a lower priority task can only cause blocking to task τ_i after it starts executing its A-phase on the bus (see Section 2.2). Therefore, the additional 1 in the Equation 8 accounts for the bus blocking suffered by this lower priority task at its R-phase. An example scenario is depicted in Figure 2.

Case 2: If τ_i does not suffer any blocking from a lower priority task (e.g. if τ_i is the lowest priority task) then the first job executed in the longest level-i busy window can also suffer bus blocking at its A-phase. In this case, the additional 1 accounts for the bus blocking suffered by the first job of task $\tau_h \in hep_{i,l}$ or τ_i before starting its A-phase on core π_l. An example scenario is depicted in Figure 3.

Hence, the maximum number of bus blockings tasks running on core π_l can suffer in $W_{i,l}$ is bounded by $\sum_{\tau_h \in hep_{i,l}} \eta_h^+(W_{i,l}) + 1$. □

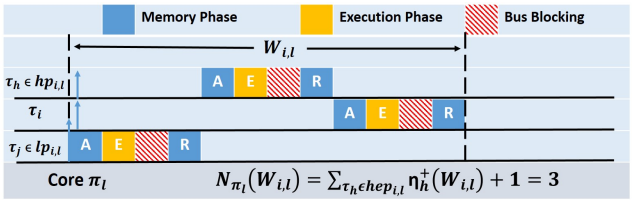

Figure 2: First job that executes in $W_{i,l}$ on core π_l is from $lp_{i,l}$

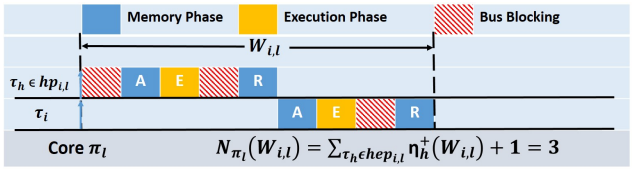

Figure 3: First job that executes in $W_{i,l}$ on core π_l is from $hp_{i,l}$

Lemma 4.2. *The maximum number of times tasks running on a remote core π_r can cause bus blocking in a time window of length $W_{i,l}$ is upper bounded by $N_{\pi_r}(W_{i,l})$, where $N_{\pi_r}(W_{i,l})$ is given by:*

$$N_{\pi_r}(W_{i,l}) = \sum_{\tau_u \in \Gamma_r} \eta_u^+(W_{i,l}) \tag{9}$$

where Γ_r is the set of all tasks that are running on core π_r.

Proof. The maximum number of bus blockings that can be caused by the tasks running on core π_r is dependent on the number of jobs released on core π_r in a time window of length $W_{i,l}$. Since any task running on core π_r in $W_{i,l}$ can participate in the bus blocking, the value of $N_{\pi_r}(W_{i,l})$ is upper bounded by $\sum_{\tau_u \in \Gamma_r} \eta_u^+(W_{i,l})$. □

4.3 Bounding Maximum Bus Blocking

Having bounded the values of $N_{\pi_l}(W_{i,l})$ and $N_{\pi_r}(W_{i,l})$, we can now compute the maximum bus blocking that can be suffered by tasks on core π_l during $W_{i,l}$ from co-running tasks executing on a remote core π_r. For this, three cases must be considered:

(i) **Case 1:** $N_{\pi_l}(W_{i,l}) > N_{\pi_r}(W_{i,l})$, the maximum number of bus blockings that can be suffered by tasks executing on core π_l is greater than the maximum number of bus blockings that can be caused by tasks executing on core π_r in the time window $W_{i,l}$.

(ii) **Case 2:** $N_{\pi_l}(W_{i,l}) = N_{\pi_r}(W_{i,l})$, the maximum number of bus blockings that can be suffered by tasks executing on core π_l is equal to the maximum number of bus blockings that can be caused by tasks executing on core π_r in the time window $W_{i,l}$.

(iii) **Case 3:** $N_{\pi_l}(W_{i,l}) < N_{\pi_r}(W_{i,l})$, the maximum number of bus blockings that can be suffered by tasks executing on core π_l is less than the maximum number of bus blockings that can be caused by tasks executing on core π_r in the time window $W_{i,l}$.

Before explaining how the maximum bus blocking can be computed in each case, we first define useful notations.

Let M_r^A (resp. M_r^R) be an ordered set that contains the execution time of the A-phases (resp. R-phases) of all jobs released on core π_r in a time window of length $W_{i,l}$, *sorted in a non-increasing order*, i.e.,

$$M_r^A = \{C_{r,1}^A, C_{r,2}^A, \ldots, C_{r,\hat{N}_{\pi_r}}^A \mid C_{r,x}^A \geq C_{r,x+1}^A\}$$
$$M_r^R = \{C_{r,1}^R, C_{r,2}^R, \ldots, C_{r,\hat{N}_{\pi_r}}^R \mid C_{r,y}^R \geq C_{r,y+1}^R\}$$

where $\hat{N}_{\pi_r}$ is equal to the value of $N_{\pi_r}(W_{i,l})$ computed using Equation 9. Note that $C_{r,x}^A$ and $C_{r,y}^R$ may or may not belong to different jobs released on core π_r in $W_{i,l}$.

Case 1: For $N_{\pi_l}(W_{i,l}) > N_{\pi_r}(W_{i,l})$, all memory phases of all jobs released on core π_r in $W_{i,l}$ can contribute to bus blocking (e.g., see Figure 4). This leads to the following lemma.

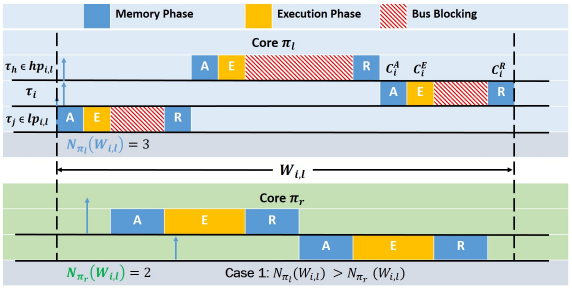

Figure 4: Maximum bus blocking for $N_{\pi_l}(W_{i,l}) > N_{\pi_r}(W_{i,l})$

Lemma 4.3. *If $N_{\pi_l}(W_{i,l}) > N_{\pi_r}(W_{i,l})$, then the maximum bus blocking that can be caused by tasks running on core π_r to tasks running on core π_l in a time window of length $W_{i,l}$ is upper bounded by $Bus_{i,r}(W_{i,l})$, given by:*

$$Bus_{i,r}(W_{i,l}) = \sum_{x=1}^{\hat{N}_{\pi_r}} C_{r,x}^A + \sum_{y=1}^{\hat{N}_{\pi_r}} C_{r,y}^R \tag{10}$$

where $C_{r,x}^A$ (resp. $C_{r,y}^R$) is the execution time of an A-phase (resp. R-phase) in the set M_r^A (resp. $C_{r,y}^R \in M_r^R$).

Proof. Each bus blocking caused by π_r can be composed of either an A-, or an R-phase of a job, or one R- and one A-phase of two different jobs released on core π_r in $W_{i,l}$, as stated in P2. Since, the precise bus access times of tasks running on core π_r are unknown, and if $N_{\pi_l}(W_{i,l}) > N_{\pi_r}(W_{i,l})$, all the memory phases of all the jobs of π_r released in $W_{i,l}$ can cause bus blocking to the tasks running on π_l in $W_{i,l}$ in the worst-case (e.g., see Figure 4).

Therefore, for $N_{\pi_l}(W_{i,l}) > N_{\pi_r}(W_{i,l})$, the maximum contribution of $\hat{N}_{\pi_r}$ jobs, i.e., $\sum_{x=1}^{\hat{N}_{\pi_r}} C_{r,x}^A + \sum_{y=1}^{\hat{N}_{\pi_r}} C_{r,y}^R$, upper bounds the maximum bus blocking. □

Case 2: If $N_{\pi_l}(W_{i,l}) = N_{\pi_r}(W_{i,l})$ then all the memory phases *except one* from all the jobs released on core π_r in the time window $W_{i,l}$ can contribute to the bus blocking. To understand this, assume that the number of bus blockings that can be suffered (resp. caused) by tasks executing on core π_l (resp. core π_r) in $W_{i,l}$ is three. In this case, there can be two possible scenarios, either the R-phase of the last job that executes on core π_r in $W_{i,l}$ (e.g., see Figure 5) or the A-phase of the first job that executes on core π_r in $W_{i,l}$ (e.g., see Figure 6) cannot participate in the bus blocking. This leads to the following Lemma.

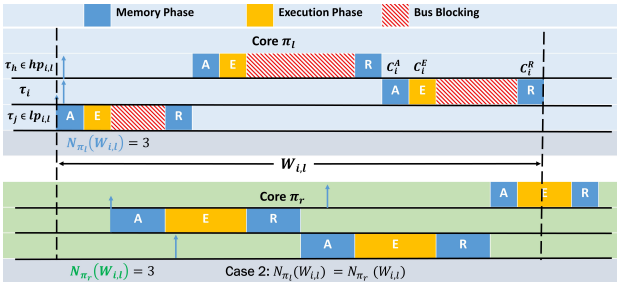

Figure 5: Possible scenario 1 when $N_{\pi_l}(W_{i,l}) = N_{\pi_r}(W_{i,l})$

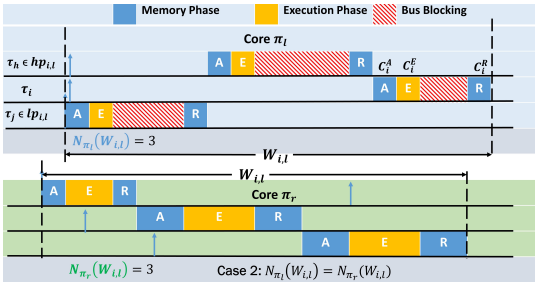

Figure 6: Possible scenario 2 when $N_{\pi_l}(W_{i,l}) = N_{\pi_r}(W_{i,l})$

Lemma 4.4. *If $N_{\pi_l}(W_{i,l}) = N_{\pi_r}(W_{i,l})$, then the maximum bus blocking $Bus_{i,r}(W_{i,l})$ that can be caused by tasks running on core π_r to tasks running on core π_l in a time window of length $W_{i,l}$ is upper bounded by:*

$$Bus_{i,r}(W_{i,l}) = \sum_{x=1}^{\hat{N}_{\pi_r}} C_{r,x}^A + \sum_{y=1}^{\hat{N}_{\pi_r}} C_{r,y}^R - \min(\min_{\forall x \in M_r^A}\{C_{r,x}^A\}, \min_{\forall y \in M_r^R}\{C_{r,y}^R\}) \tag{11}$$

Proof. We prove the lemma using the following two observations:

1. If the A-phase of the first job on π_r participates to the bus blocking of any job of π_l released in $W_{i,l}$, then the first bus blocking is composed of only an A-phase (see P3) while the rest of the bus blockings can be composed of one R- and one A-phase of two different jobs running on π_r within $W_{i,l}$ (see P2). Consequently, the R-phase of the last job executing on π_r within $W_{i,l}$ cannot participate to $Bus_{i,r}(W_{i,l})$. Since we do not know which job on core π_r will be the last to execute in $W_{i,l}$, we assume that the job with the smallest R-phase is the last job that executes on core π_r in $W_{i,l}$ (e.g., see Figure 5).

2. If the A-phase of the first job on π_r does not block the memory-phase of any job of π_l released in $W_{i,l}$, i.e., the first bus blocking is composed of an R-phase of the first job and an A-phase of any other job executed on π_r within $W_{i,l}$ (refer to P2), then all the memory phases except the A-phase of the first job executing on π_r within $W_{i,l}$ can contribute to $Bus_{i,r}(W_{i,l})$. Since we do not know which job on core π_r will execute first within $W_{i,l}$, we assume that the job with the smallest A-phase is the first job that executes on core π_r. See Figure 6 for an example scenario.

Building on the above observations, either one A-phase or one R-phase cannot participate to $Bus_{i,r}(W_{i,l})$. Therefore, $Bus_{i,r}(W_{i,l})$ is maximised when the non-participating memory phase is the smallest among those in M_r^A and M_r^R, hence proving the lemma. □

Case 3: If $N_{\pi_l}(W_{i,l}) < N_{\pi_r}(W_{i,l})$, then at most $N_{\pi_l}(W_{i,l})$ bus blockings can be caused by tasks running on core π_r to tasks running on core π_l in $W_{i,l}$.

To extract the $N_{\pi_l}(W_{i,l})$ A and R-phases with the highest memory demands among all jobs that execute on π_r in $W_{i,l}$, we first divide the set M_r^A (resp. M_r^R) into two sub-sets named M_r^{AH} and M_r^{AL} (resp. M_r^{RH} and M_r^{RL}). The subset M_r^{AH} (resp. M_r^{RH}) contains $N_{\pi_l}(W_{i,l})$ A-phases (resp. R-phases) with the highest memory demands and the rest of the A-phases (resp. R-phases) are in the subset M_r^{AL} (resp. M_r^{RL}). Formally, these subsets are defined as follows:

$$M_r^{AH} = \{C^A_{r,1}, C^A_{r,2}, \ldots, C^A_{r,\hat{N}_{\pi_l}} \mid C^A_{r,x} \geq C^A_{r,x+1}\}$$
$$M_r^{AL} = \{C^A_{r,\hat{N}_{\pi_l}+1}, C^A_{r,\hat{N}_{\pi_l}+2}, \ldots, C^A_{r,\hat{N}_{\pi_r}} \mid C^A_{r,y} \geq C^A_{r,y+1}\}$$
$$M_r^{RH} = \{C^R_{r,1}, C^R_{r,2}, \ldots, C^R_{r,\hat{N}_{\pi_l}} \mid C^R_{r,x} \geq C^R_{r,x+1}\}$$
$$M_r^{RL} = \{C^R_{r,\hat{N}_{\pi_l}+1}, C^R_{r,\hat{N}_{\pi_l}+2}, \ldots, C^R_{r,\hat{N}_{\pi_r}} \mid C^R_{r,y} \geq C^R_{r,y+1}\}$$

where $\hat{N}_{\pi_l} = N_{\pi_l}(W_{i,l})$ and can be computed using Equation 8.

We then identify two possible sub-cases:

Sub-case 1: The core π_r can cause $\hat{N}_{\pi_l}$ number of bus blockings using all the elements of the M_r^{AH} and M_r^{RH} sub-sets in which each bus blocking is composed of one A- and one R-phase of two different jobs. The maximum bus blocking in this sub-case can be simply derived by considering the sum of all the A- and R-phases in M_r^{AH} and M_r^{RH} sub-sets. We discussed this sub-case in Lemma 4.5.

Sub-case 2: The core π_r cannot cause $\hat{N}_{\pi_l}$ number of bus blockings using all the elements of M_r^{AH} and M_r^{RH}. This can only happen if all the elements of M_r^{AH} and M_r^{RH} are associated to the same set of jobs. In other words, the A- and R-phases pertain to the exact same job. In this sub-case, one memory phase in M_r^{AH} or M_r^{RH} does not participate to the bus blockings. This sub-case is discussed in Lemma 4.6.

Lemma 4.5. *If core π_r can cause $\hat{N}_{\pi_l}$ number of bus blockings using all the elements of M_r^{AH} and M_r^{RH} sub-sets, then the maximum bus blocking $Bus_{i,r}(W_{i,l})$ that can be caused by tasks running on core π_r to tasks running on core π_l in a time window of length $W_{i,l}$ is upper bounded by:*

$$Bus_{i,r}(W_{i,l}) = \sum_{x=1}^{\hat{N}_{\pi_l}} C^A_{r,x} + \sum_{y=1}^{\hat{N}_{\pi_l}} C^R_{r,y} \tag{12}$$

where $C^A_{r,x}$ (resp. $C^R_{r,y}$) is the execution time of the A-phase (resp. R-phase), $C^A_{r,x} \in M_r^{AH}$ (resp. $C^R_{r,y} \in M_r^{RH}$).

Proof. If core π_r can cause $\hat{N}_{\pi_l}$ bus blockings, in which each bus blocking is composed of one R- and one A-phase of two different jobs by using all the elements of M_r^{AH} and M_r^{RH} sub-sets, then all the memory phases of M_r^{AH} and M_r^{RH} can participate to the bus blocking. Since, M_r^{AH} and M_r^{RH} are the sub-sets that contain memory phases with higher memory demand, the maximum bus blocking that can be caused by tasks running on core π_r to tasks running on core π_l in a time window of length $W_{i,l}$ can be upper-bounded by taking the sum of all the memory phases in M_r^{AH} and M_r^{RH} sub-sets. Hence, Equation 12 bounds the maximum bus blocking in this sub-case. □

Lemma 4.6. *If core π_r cannot cause $\hat{N}_{\pi_l}$ number of bus blockings using all the elements of M_r^{AH} and M_r^{RH} sub-sets, then the maximum bus blocking $Bus_{i,r}(W_{i,l})$ that can be caused by tasks running on core π_r to tasks running on core π_l in a time window of length $W_{i,l}$ is upper bounded by:*

$$\begin{aligned} Bus_{i,r}(W_{i,l}) = &\sum_{x=1}^{\hat{N}_{\pi_l}} C^A_{r,x} + \sum_{y=1}^{\hat{N}_{\pi_l}} C^R_{r,y} \\ &- \min\Big(\big(\min_{\forall x \in M_r^{AH}}\{C^A_{r,x}\} - \max_{\forall y \in M_r^{AL}}\{C^A_{r,y}\}\big), \\ &\big(\min_{\forall x \in M_r^{RH}}\{C^R_{r,x}\} - \max_{\forall y \in M_r^{RL}}\{C^R_{r,y}\}\big)\Big) \end{aligned} \tag{13}$$

where $\min_{\forall x \in M_r^{AH}}\{C^A_{r,x}\}$ (resp. $\min_{\forall x \in M_r^{RH}}\{C^R_{r,x}\}$) returns the smallest element of M_r^{AH} (resp. M_r^{RH}); and $\max_{\forall y \in M_r^{AL}}\{C^A_{r,y}\}$ (resp. $\max_{\forall y \in M_r^{RL}}\{C^R_{r,y}\}$) returns the largest element of M_r^{AL} (resp. M_r^{RL}).

Proof. We know that core π_r can cause at most $\hat{N}_{\pi_l}$ bus blockings in which each bus blocking is from one R- and one A-phase of two different jobs. To derive the maximum bus blocking, it is necessary to consider all the elements of M_r^{AH} and M_r^{RH} sub-sets as they contain the memory phases with the largest execution times. However, if all the elements of M_r^{AH} and M_r^{RH} are associated to the exact same set of jobs of core π_r, then at-least one memory phase from either M_r^{AH} or M_r^{RH} cannot participate to the bus blocking. This happens because either the A-phase of the first job (i.e., an element from M_r^{AH}) or the R-phase of the last job that execute on π_r (i.e., an element from M_r^{RH}) cannot participate to the bus blockings. Since, we have $N_{\pi_l}(W_{i,l}) < N_{\pi_r}(W_{i,l})$, one memory phase

from M_r^{AL} or M_r^{RL} sub-set can participate. Consequently, $\hat{N}_{\pi_l}$ bus blockings can be obtained in which each bus blocking is composed of one R- and one A-phase of two different jobs of core π_r.

Considering the above, the bus blocking is maximized when the non-participating memory phase in M_r^{AH} or M_r^{RH} is smallest and the participating memory phase in M_r^{AL} or M_r^{RL} is largest. Hence, Equation 13 bounds the maximum bus blocking in this sub-case. □

Assuming that the bus arbitration policy is FCFS, the maximum bus blocking $Bus_{i,l}^{max}(W_{i,l})$ suffered by the local core due to *all* remote cores is thus given by:

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

5 SCHEDULABILITY ANALYSIS

Having derived the maximum bus blocking that can be suffered by tasks executing on the local core using Equation 14, we can now compute the length of the longest level-i busy window $W_{i,l}$ using Equation 7. Similarly, the maximum number of jobs of task τ_i that can execute within $W_{i,l}$ can be computed using Equation 3.

As proven in [3], to compute the WCRT of task τ_i, we need to first determine the response time of each job of τ_i that executes during the level-i busy window $W_{i,l}$. Moreover, we also know that on a multicore platform, all jobs of all tasks executing on the local core π_l within $W_{i,l}$ can suffer bus blocking due to co-running tasks that are executing on the remote cores. This must be factored in WCRT analysis.

To compute the response time of the k^{th} job of τ_i on core π_l, i.e., denoted by $\tau_{i,k,l}$, we first need to compute the latest starting time of the *R-phase* of $\tau_{i,k,l}$. This is due to the fact that each job that executes on core π_l during the response-time of $\tau_{i,k,l}$, including $\tau_{i,k,l}$, can suffer bus blocking until starting the R-phase of $\tau_{i,k,l}$. The computation of the latest starting time of the R-phase of $\tau_{i,k,l}$ on core π_l is given by the following Lemma.

Lemma 5.1. *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 value of the fixed-point iteration of the following equation:*

$$s_{i,k,l}^R = C_{lp,i,l}^{max} + \sum_{h\in hep_{i,l}\setminus\tau_i} \eta_h^+(s_{i,k,l}^R - (C_i^A + C_i^E)) \times C_h + Bus_{i,l}^{max}(s_{i,k,l}^R) + (k-1)\times C_i + (C_i^A + C_i^E) \tag{15}$$

Proof. The proof is divided in two steps. In the first step, we upper bound the contributions of tasks executing on the same core to the start time of R-phase of $\tau_{i,k,l}$. In step two, we will upper bound the impact of tasks running on the remote cores π_l on the start time of R-phase of $\tau_{i,k,l}$.

Step 1. Task τ_i can suffer blocking from at most one job from lower priority tasks in $lp_{i,l}$. This blocking is upper bounded by $C_{lp,i,l}^{max}$ i.e., computed using Equation 1.

All jobs released by the higher priority tasks in $hep_{i,l}$ can cause interference on $\tau_{i,k,l}$ until the start of its A-phase due to non-preemptive scheduling. Hence, the total interference that can be caused by a task $\tau_h \in hep_{i,l}$ until the start of A-phase of $\tau_{i,k,l}$ is upper bounded by $\eta_h^+(s_{i,k,l}^R - (C_i^A + C_i^E)) \times C_h$, where C_h is the WCET of task τ_h in isolation. Effectively, the total contribution from all tasks in $hep_{i,l}$ to the start time of R-phase of $\tau_{i,k,l}$ is upper bounded by $\sum_{h\in hep_{i,l}\setminus\tau_i} \eta_h^+(s_{i,k,l}^R - (C_i^A + C_i^E)) \times C_h$.

Knowing that $k-1$ jobs of task τ_i may have executed before $\tau_{i,k,l}$, their contribution to the latest start time of the R-phase of $\tau_{i,k,l}$ is given by $(k-1)\times C_i$.

Finally, to compute the start time of the R-phase of $\tau_{i,k,l}$, we add the WCET of A- and E-phase of τ_i, given by $(C_i^A + C_i^E)$.

Step 2. It is possible that each job on core π_l executing before $\tau_{i,k,l}$ (and $\tau_{i,k,l}$ itself) can suffer bus blocking due to tasks running on the remote cores. Thus, the maximum bus blocking caused by co-running tasks executing on all the remote cores until the start of R-phase of $\tau_{i,k,l}$ is upper bounded by $Bus_{i,l}^{max}(s_{i,k,l}^R)$ which is given by Equation 14. □

As $s_{i,k,l}^R$ appears on both sides of Equation 15, it can be solved iteratively by initializing $s_{i,k,l}^R = C_i^A + C_i^E + C_{lp,i,l}^{max} + \sum_{h\in hep_{i,l}\setminus\tau_i} C_h$. 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 15 converges.

Using $s_{i,k,l}^R$, the response time $R_{i,k,l}$ of $\tau_{i,k,l}$ can be computed by simply adding to it the execution time of R-phase C_i^R of task τ_i.

$$R_{i,k,l} = s_{i,k,l}^R + C_i^R \tag{16}$$

Finally, the WCRT of task τ_i can be computed by maximizing equation 16 over all jobs of τ_i that execute during the level-i busy window. Hence,

$$R_{i,l}^{max} = \max_{k\in[1,K_i]} \{R_{i,k,l}\} \tag{17}$$

where the computation of K_i is given in Equation 3.

5.1 Schedulability Test

If the WCRT of each task in the system is less than or equal to its relative deadline, then the system is deemed schedulable, otherwise it is not. Furthermore, the task set can only be schedulable if the total bus utilization of the system is less than or equal to 1 since the system bus is saturated otherwise.

6 EXPERIMENTAL EVALUATION

In this section, we evaluate the effectiveness of the proposed approach and compare its performance with a similar analysis from the state-of-the-art.

6.1 Proposed Work vs. State-of-the-Art [22]

The work proposed in [22] focuses on bounding the memory interference for PREM-based tasks scheduled on a multicore platform using partitioned fixed-priority non-preemptive scheduling while we focus on the 3-phase task model. In addition, the work in [22] assumes a processor-priority based bus arbitration policy which allows Higher Priority (HP) processors to preempt memory phases that might be executing on Lower Priority (LP) processors while we assume a FCFS approach. In [22], the maximum memory interference suffered by the LP processor from the tasks running on the HP processors in any time window of length Δ is derived using all the memory phases released on HP processors in any time window of length Δ (see Equation 3 of [22]) or by upper bounding

the maximum interference each memory phase on the LP processor can suffer from tasks running on HP processors (see Equation 6 of [22]).

To have a meaningful comparison between our work and the work in [22], we assume that all the cores have the same priority. Moreover, we also do not allow global memory preemptions. Note that by doing so the analysis presented in [22] also behaves like a FCFS-based analysis. Equation 6 of [22] integrates the impact of global memory preemptions. Therefore, we adapted the formulation of Equation 6 of [22] to account for non-preemptive execution of memory phases[4].

6.2 Experiments

To compare the performance of our proposed analysis against the analysis in [22], we performed different sets of experiments using synthetic task sets. The default configuration was a multicore platform with 4 cores and a task set size of 32 tasks with 8 tasks per core. Task utilizations were generated using Uunifast-discard [11]. Task periods were generated using log-uniform distribution in the range of [100,1000].

The WCET C_i of each task τ_i was obtained by the product $U_i \times T_i$. The memory demand (MD) for each task was assigned randomly between [10%, 30%]. The values of C_i^A, C_i^E and C_i^R are then chosen such that $C_i^A + C_i^R = MD \times C_i$ [5] and $C_i^E = C_i - (C_i^A + C_i^R)$. Task deadlines are implicit with priorities assigned using Rate-Monotonic [13].

We evaluated the impact of both analyses on task set schedulability by varying different parameters, i.e., core utilization, number of cores, task memory demands, and task periods.

1. Core Utilization: In this experiment, we varied each core utilization between 0.05 and 1 in steps of 0.05 and plotted the number of task sets that were deemed schedulable by all the analyzed approaches. In this experiment, 1000 task sets were generated per point. The percentage of task sets that were deemed schedulable using our approach, i.e., marked as OUR, and the approach in [22], i.e., marked as SCHW, for each core utilization value are shown in Figure 7. We can see in Figure 7 that our proposed analysis outperforms the analysis in [22] because of a tighter bound on the bus contention that can be caused by the remote cores. The analysis in [22] provides a safe upper-bound on the memory interference but it does not consider the variations in the number of jobs released on the local/remote core(s) which results in overestimating the memory/bus contention. On the contrary, our approach accurately estimates the bus contention with the help of different cases identified in Section 4. For instance, case 3 of our approach extract a specific set of memory phases (A- and R-phases) running on the remote core that leads to the maximum bus blocking that can be caused by the tasks running on the remote core to the local core in practice. We can see in Figure 7 that at a core utilization of 0.45 our approach was able to schedule up to 29% more task sets compared to SCHW [22]. However, we also note that the overall task set schedulability for both approaches is quite low which is intuitive to understand, due to the assumption of FCFS-based bus arbitration.

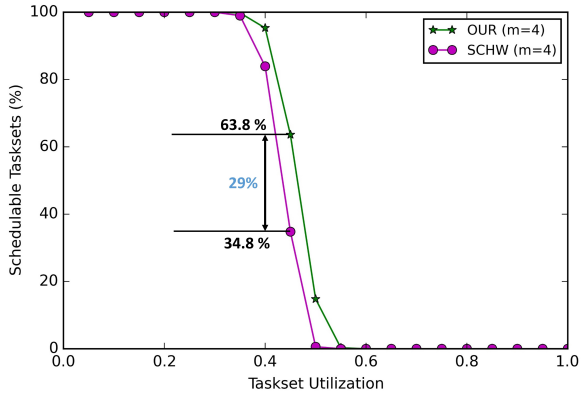

Figure 7: Varying Core Utilization

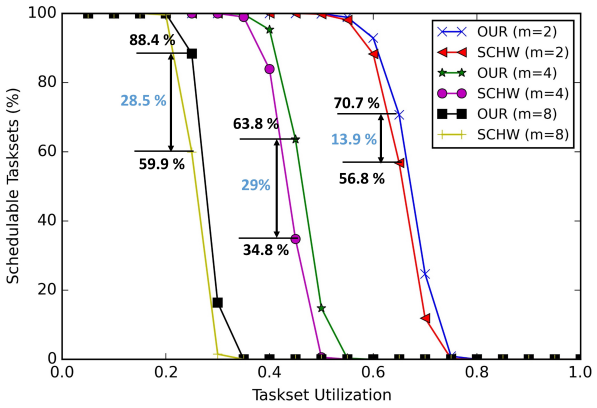

Figure 8: Varying Number of Cores

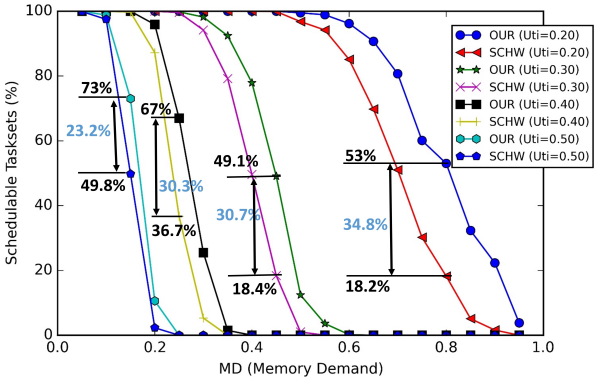

Figure 9: Varying Memory Demand

2. Number of Cores: In this experiment, we re-do the previous experiment by varying the number of cores along with the core utilization. The number of cores (m) was increased from 2 to 8 along with core utilizations that were varied from 0.05 to 1 in steps of

[4] We observed experimentally that by using the exact same formulation of Equation 6 from [22] that allows global preemptions of memory phases, the performance of [22] tends to decline under FCFS bus arbitration policy.

[5] Note that for the analysis in [22] we consider a single memory phase of length $MD \times C_i$.

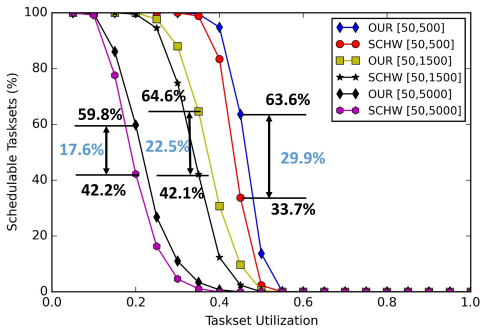

Figure 10: Varying Task Periods

0.05. The percentage of task sets that were deemed schedulable for different values of m by the two approaches are shown in Figure 8. We can see in Figure 8 that by increasing the number of cores, the number of task sets that were deemed schedulable by both analyses decreases. This is mainly due to the fact that by increasing the number of cores, the number of tasks also increases, which results in increasing the bus blocking that can be suffered by the task under analysis from the remote cores. For example, for two cores all task sets were deemed schedulable by both approaches at a core utilization of 0.4 but no task set was schedulable at same core utilization when the value of m is increased to 8. However, we can still see that even for higher values of m, the proposed approach outperforms the analysis of [22].

3. Task Memory Demands: In this experiment, we varied the memory demand (MD) of tasks w.r.t their WCET and analyzed its impact on the task set schedulability. Effectively, we used the value of MD to determine C_i^A, C_i^E, and C_i^R such that $C_i^A + C_i^R = MD \times C_i$ and $C_i^E = C_i - (C_i^A + C_i^R)$. The value of MD was varied from 0.05 to 0.95 (i.e., 5% to 95%) in steps of 0.05 and the number of task sets that were deemed schedulable by both the analyzed approaches are plotted in Figure 9. We choose different set of core utilizations (Uti), i.e., 0.20, 0.30, 0.40 and 0.50, to show the impact of MD on task set schedulability.

We can see in Figure 9, that for lower values of core utilization the number task sets that were deemed schedulable by both approaches was much higher even for larger values of MD. For example, at a core utilization of 20%, tasks with very high memory demands, i.e., up to 80% of their WCET, were still schedulable. However, the schedulability ratio decreases rapidly for higher values of core utilization. We can also see that the schedulability was reduced with an increase in the memory demand for both the approaches. This is intuitive, as for higher values of MD, the values of C_i^A, and C_i^R also increases which results in increasing bus blocking. Finally, we can also observe that the gains of the proposed analysis over the state-of-the-art analysis [22] remains relatively constant with improvements from 23.2% up to 34.8% percentage points in terms of schedulability.

4. Task Periods: In this experiment, we varied the range of task periods and analyzed its impact on schedulability. As we generate the WCET C_i of tasks using the task periods T_i, i.e., $C_i = U_i \times T_i$, which is then used to generate C_i^A, C_i^E and C_i^R, therefore, the value of task periods can significantly impact schedulability.

We used three different period ranges, i.e., [50,500], [50,1500], [50, 5000] in our experiment and observed that an increase in the period range has a negative impact on task set schedulability. When task periods are increased, their WCET also increases which results in generating higher values for C_i^A, C_i^E and C_i^R. Hence, the bus blockings that can be suffered/caused by the tasks also increases resulting in decreasing schedulability. However, we can still see that even for higher values of task periods the proposed analysis dominate the state-of-the-art analysis.

In all the experiments, the time taken by the proposed analysis was roughly 5x more than the state-of-the-art analysis of SCHW [22]. Intuitively, this is due to our more fine-grained analysis of bus blocking with of help of cases and sub-cases.

7 RELATED WORK

The problem of timing unpredictability due to the shared bus contention in multicore systems is not new [6] and many existing works already attempted to solve this problem [16]. Some approaches [5, 12, 20] are based on Time Division Multiple Access (TDMA) in which time slots are divided among cores and a core can only access the system bus in its defined time slot. Dasari et al. [7] proposed a response time analysis considering the maximum bus interference for an unspecified work-conserving arbiter under partitioned scheduling. A general framework for memory bus contention analysis that covers a wide range of bus arbitration policies is proposed in [8]. Rashid et al. [18] proposed the cache persistence-aware memory bus contention analysis for multicore systems. Even though these approaches are proposed to bound the bus contention in partitioned scheduling, they are proposed for generic task models and are not tailored for phased execution models.

On the other hand, approaches like [1, 2, 4, 9, 14, 19, 22–24, 26] focus on phased execution models. Maia et al. [14] focus on the bus contention analysis for the fixed priority 3-phase task model under global scheduling. Davis et al. [9] proposed an extensible framework for multicore analysis. The authors compute the WCRT of fixed-priority preemptive tasks in partitioned scheduling by incorporating the inter-core interference caused by co-running tasks due to shared bus, shared main memory, and shared caches access. Works like [26] are based on memory centric scheduling in which the access to the main memory is divided among all the cores in the system using TDMA. The memory phases can then access the main memory during their allocated time slot. Recent work on memory-centric scheduling [22] focused on a fixed-priority memory centric scheduler for COTS multiprocessors as the authors suggest that TDMA may result in underutilization of the resource.

None of the above-mentioned works can be directly compared against the proposed approach in this paper due to different set of assumptions followed. For instance, [1, 14] focus on global scheduling whereas [9] presents a response time analysis by considering the inter-core interference due to various shared resources of multicore systems. [23, 24] focus on a specific hardware architecture that has a dedicated I/O bus, dual-port memories with DMA support, and scratchpad memories.

As discussed in Section 6.1, the closest work that can be compared against our approach is [22] as it proposed for fixed-priority

non-preemptive PREM task model under partitioned scheduling and focus on only one source of inter-core interference under partitioned scheduling i.e., main memory, and it assumes that the main memory can handle only one request at a time.

8 CONCLUSION

In this work, we presented a fine-grained analysis to compute the maximum bus blocking suffered by 3-phase tasks under fixed-priority partitioned scheduling when FCFS is the bus arbitration policy. The maximum bus blocking is derived using different cases and sub-cases that may happen in practice which allows us to acheive tighter bounds on schedulability. We formulated a bus-contention aware schedulability analysis that accounts for the delay due to bus blocking when computing the WCRT of tasks. The experimental evaluation shows that the proposed approach can improve the number of task sets that are deemed schedulable by up to 34 percentage points in comparison to a state-of-the-art approach. As future work, we would like to extend our analysis to support different bus arbitration policies and evaluate its performance using benchmarks and industrial case studies.

ACKNOWLEDGMENTS

This work was partially supported by National Funds through FCT/MCTES (Portuguese Foundation for Science and Technology), within the CISTER Research Unit (UIDB/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).

REFERENCES

[1] Ahmed Alhammad and Rodolfo Pellizzoni. 2014. Schedulability analysis of global memory-predictable scheduling. In *Proceedings of the 14th International Conference on Embedded Software - EMSOFT '14*. ACM Press, New Delhi, India, 1–10. https://doi.org/10.1145/2656045.2656070

[2] A. Alhammad and R. Pellizzoni. 2014. Time-predictable execution of multithreaded applications on multicore systems. In *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*. 1–6. https://doi.org/10.7873/DATE.2014.042

[3] R. J. Bril, J. J. Lukkien, and W. F. J. Verhaegh. 2007. 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)*. 269–279.

[4] D. Casini, A. Biondi, G. Nelissen, and G. Buttazzo. 2020. 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)*. 239–252. https://doi.org/10.1109/RTAS48715.2020.000-3

[5] Sudipta Chattopadhyay, Abhik Roychoudhury, and Tulika Mitra. 2010. Modeling Shared Cache and Bus in Multi-Cores for Timing Analysis. In *Proceedings of the 13th International Workshop on Software & Compilers for Embedded Systems* (St. Goar, Germany) *(SCOPES '10)*. Association for Computing Machinery, New York, NY, USA, Article 6, 10 pages. https://doi.org/10.1145/1811212.1811220

[6] Dakshina Dasari, Benny Akesson, Vincent Nelis, Muhammad Ali Awan, and Stefan M. Petters. 2013. Identifying the sources of unpredictability in COTS-based multicore systems. In *2013 8th IEEE International Symposium on Industrial Embedded Systems (SIES)*. IEEE, Porto, 39–48. https://doi.org/10.1109/SIES.2013.6601469

[7] D. Dasari, B. Andersson, V. Nelis, S. M. Petters, A. Easwaran, and J. Lee. 2011. 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*. 1068–1075.

[8] Dakshina Dasari, Vincent Nelis, and Benny Akesson. 2015. A framework for memory contention analysis in multi-core platforms. *Real-Time Systems* 52 (06 2015). https://doi.org/10.1007/s11241-015-9229-9

[9] Robert I. Davis, Sebastian Altmeyer, Leandro S. Indrusiak, Claire Maiza and·Vincent Nelis, and Jan Reineke. 2017. An extensible framework for multicore response time analysis. *Real-Time Systems* (July 2017).

[10] Guy Durrieu, Madeleine Faugère, Sylvain Girbal, Daniel Gracia Pérez, Claire Pagetti, and W. Puffitsch. 2014. Predictable Flight Management System Implementation on a Multicore Processor. In *Embedded Real Time Software (ERTS'14)*. TOULOUSE, France. https://hal.archives-ouvertes.fr/hal-01121700

[11] P. Emberson, R. Stafford, and R.I. Davis. 2010. Techniques For The Synthesis Of Multiprocessor Tasksets. *WATERS'10* (01 2010).

[12] T. Kelter, H. Falk, P. Marwedel, S. Chattopadhyay, and A. Roychoudhury. 2011. Bus-Aware Multicore WCET Analysis through TDMA Offset Bounds. In *2011 23rd Euromicro Conference on Real-Time Systems*. 3–12.

[13] C L Liu. [n.d.]. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. ([n. d.]), 16.

[14] Claudio Maia, Geoffrey Nelissen, Luis Nogueira, Luis Miguel Pinho, and Daniel Gracia Perez. 2017. 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, 1–10. https://doi.org/10.1109/RTCSA.2017.8046313

[15] Claudio Maia, Luis Nogueira, Luis Miguel Pinho, and Daniel Gracia Perez. 2016. A closer look into the AER Model. In *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, Berlin, Germany, 1–8. https://doi.org/10.1109/ETFA.2016.7733567

[16] Claire Maiza, Hamza Rihani, Juan M. Rivas, Joël Goossens, Sebastian Altmeyer, and Robert I. Davis. 2019. A Survey of Timing Verification Techniques for Multi-Core Real-Time Systems. *Comput. Surveys* 52, 3 (June 2019), 1–38. https://doi.org/10.1145/3323212

[17] Rodolfo Pellizzoni, Emiliano Betti, Stanley Bak, Gang Yao, John Criswell, Marco Caccamo, and Russell Kegley. 2011. 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, 269–279. https://doi.org/10.1109/RTAS.2011.33

[18] Syed Aftab Rashid, Geoffrey Nelissen, and Eduardo Tovar. 2020. Cache Persistence-Aware Memory Bus Contention Analysis for Multicore Systems. https://doi.org/10.23919/DATE48585.2020.9116265

[19] J. M. Rivas, J. Goossens, Xavier Poczekajlo, and Antonio Paolillo. 2019. Implementation of Memory Centric Scheduling for COTS Multi-Core Real-Time Systems. In *ECRTS*.

[20] J. Rosen, A. Andrei, P. Eles, and Z. Peng. 2007. 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)*. 49–60.

[21] Simon Schliecker and Rolf Ernst. 2010. Real-time performance analysis of multiprocessor systems with shared memory. *ACM Transactions on Embedded Computing Systems* 10, 2 (Dec. 2010), 1–27. https://doi.org/10.1145/1880050.1880058

[22] Gero Schwäricke, Tomasz Kloda, Giovani Gracioli, Marko Bertogna, and Marco Caccamo. 2020. Fixed-priority memory-centric scheduler for COTS-based multiprocessors. In *32nd Euromicro Conference on Real-Time Systems, ECRTS 2020 (Leibniz International Proceedings in Informatics, LIPIcs)*, Marcus Volp (Ed.). Schloss Dagstuhl- Leibniz-Zentrum fur Informatik GmbH, Dagstuhl Publishing. https://doi.org/10.4230/LIPIcs.ECRTS.2020.1 32nd Euromicro Conference on Real-Time Systems, ECRTS 2020 ; Conference date: 07-07-2020 Through 10-07-2020.

[23] R. Tabish, R. Mancuso, S. Wasly, A. Alhammad, S. S. Phatak, R. Pellizzoni, and M. Caccamo. 2016. A Real-Time Scratchpad-Centric OS for Multi-Core Embedded Systems. In *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 1–11. https://doi.org/10.1109/RTAS.2016.7461321

[24] Rohan Tabish, Renato Mancuso, Saud Wasly, Rodolfo Pellizzoni, and Marco Caccamo. 2019. A real-time scratchpad-centric OS with predictable inter/intra-core communication for multi-core embedded systems. *Real-Time Systems* 55 (10 2019). https://doi.org/10.1007/s11241-019-09340-0

[25] K. W. Tindell, A. Burns, and A. J. Wellings. 1994. An Extendible Approach for Analyzing Fixed Priority Hard Real-Time Tasks. *Real-Time Syst.* 6, 2 (March 1994), 133–151. https://doi.org/10.1007/BF01088593

[26] Gang Yao, Rodolfo Pellizzoni, Stanley Bak, Emiliano Betti, and Marco Caccamo. 2012. Memory-centric scheduling for multicore hard real-time systems. *Real-Time Systems* 48, 6 (Nov. 2012), 681–715. https://doi.org/10.1007/s11241-012-9158-9