CISTER
Research Centre in
Real-Time & Embedded
Computing Systems

Journal Paper

Schedulability Analysis for 3-Phase Tasks with Partitioned Fixed-Priority Scheduling

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

*CISTER Research Centre

CISTER-TR-220801

2022

Schedulability Analysis for 3-Phase Tasks with Partitioned Fixed-Priority 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 memory bus that connects the cores to the off-chip main memory. This sharing of memory bus may cause tasks running on different cores to compete for access to the main memory whenever data/instructions are need to be read/written from/to the main memory. Such competition is problematic, as it may cause variations in the execution time of tasks in a non-deterministic way. To reduce the complexity of analysing this problem, the 3-phase task model was proposed that divides tasks' executions into distinct memory and execution phases. The distinctive memory phases are then scheduled to eliminate/minimize main memory contention between concurrently executing tasks. However, 3-phase tasks running on different cores may still compete to access the shared memory bus/main memory in order to execute memory phases. This paper presents a partitioned scheduling-based approach that allows one to derive memory bus contention-aware worst-case response time of tasks that follow the 3-phase task model. In particular, the bus-contention analysis is derived by considering two memory access models, i.e., (i) dedicated memory access model, where a core having allowed to access the main memory via memory bus is permitted to execute more than one memory phase, and (ii) fair memory access model, that restrict each core to execute only one memory phase in its allocated bus access. Both these models represent different system and application requirements, and the resulting bus contention of tasks may vary depending on the considered model. To evaluate the effectiveness of the proposed bus contention analysis, we compare its performance against an existing analysis in the state-of-the-art by performing (i) case-study experiments, using benchmarks from the Mälardalen Benchmark suite, and (ii) empirical evaluation using synthetic task sets. Results show that our proposed analysis can improve task set schedulability of 3-phase tasks by up to 88 percentage points.

Schedulability Analysis for 3-Phase Tasks with Partitioned Fixed-Priority Scheduling

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

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 memory bus that connects the cores to the off-chip main memory. This sharing of memory bus may cause tasks running on different cores to compete for access to the main memory whenever data/instructions are need to be read/written from/to the main memory. Such competition is problematic, as it may cause variations in the execution time of tasks in a non-deterministic way. To reduce the complexity of analysing this problem, the 3-phase task model was proposed that divides tasks' executions into distinct memory and execution phases. The distinctive memory phases are then scheduled to eliminate/minimize main memory contention between concurrently executing tasks. However, 3-phase tasks running on different cores may still compete to access the shared memory bus/main memory in order to execute memory phases. This paper presents a partitioned scheduling-based approach that allows one to derive memory bus contention-aware worst-case response time of tasks that follow the 3-phase task model. In particular, the bus-contention analysis is derived by considering two memory access models, i.e., (i) dedicated memory access model, where a core having allowed to access the main memory via memory bus is permitted to execute more than one memory phase, and (ii) fair memory access model, that restrict each core to execute only one memory phase in its allocated bus access. Both these models represent different system and application requirements, and the resulting bus contention of tasks may vary depending on the considered model. To evaluate the effectiveness of the proposed bus contention analysis, we compare its performance against an existing analysis in the state-of-the-art by performing (i) case-study experiments, using benchmarks from the Mälardalen Benchmark suite, and (ii) empirical evaluation using synthetic task sets. Results show that our proposed analysis can improve task set schedulability of 3-phase tasks by up to 88 percentage points.

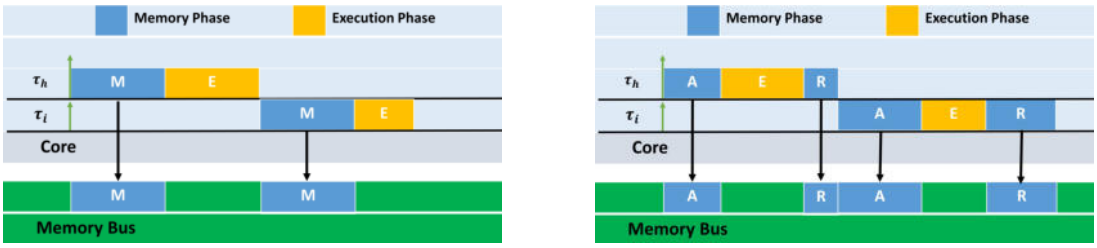
Keywords: Real-Time Systems, Multicore Processors, Partitioned Scheduling, Bus Contention, Schedulability Analysis

This work is an extended version of the paper entitled "Bus-Contention Aware Schedulability Analysis for the 3-Phase Task Model with Partitioned Scheduling [1]". The main extensions include the formulation of the Fair Memory Access Model (FMAM) (Section 2), maximum bus blocking analysis for the FMAM (Section 6), and the extensive empirical evaluation using a case study and synthetic tasksets (Section 8).

1. Introduction

Multicore processors offer several advantages such as higher computational power and lower energy consumption over traditional single-core computing platforms. However, the use of multicore processors

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



(a) The PRedictable Execution Model (PREM) (b) The 3-Phase Task Model

Figure 1: Phased Execution Models

in *hard* real-time systems, i.e., systems with stringent timing requirements, is still under the scrutiny of the real-time systems community due to their *unpredictable* behavior. This unpredictable behavior is a direct result of current designs which include shared resources, such as buses, caches, main memory, and I/O devices. When accessing any of these shared resources, a task executing on a given core may suffer *inter-core interference* from co-running tasks, i.e., tasks running on the other cores, potentially affecting the execution time of the tasks in a non-deterministic manner.

Analysing the inter-core interference suffered by a given task is extremely challenging as it depends on specific properties (i.e., number of memory requests, time required to serve each memory request, task priority, etc.) of tasks executing on other cores at the same time instant. To simplify this problem, the concept of phased execution models, e.g., PRedictable Execution Model (PREM) [2] was introduced. In PREM, the tasks' executions are divided into separate memory and execution phases. As shown in Figure 1a, whenever a PREM task is released, it first executes a memory phase followed by the execution phase. The memory phase is responsible for loading tasks' data and instructions into the core's local memory (e.g., cache or scratchpad) from the main memory. During the execution phase, the core executes the task's code by processing data/instructions already available in its local memory without the need of accessing the memory bus or the main memory. Effectively, in the PREM model, the shared bus/main memory is accessed by tasks only during their memory phases which reduces the complexity of analysing the maximum inter-core interference suffered by a task due to co-running tasks.

The 3-phase (or AER) task model [3, 4] is a generalization of the PREM model. As shown in Figure 1b, the 3-phase task model divides the task into three phases, namely *Acquisition*, *Execution*, and *Restitution*. 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 finally, in the restitution phase (i.e., R-phase), the processed data is written back to the main memory. Similarly to the PREM model, in the 3-phase task model, accesses to the main memory via a memory 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 are usually used with a co-scheduling

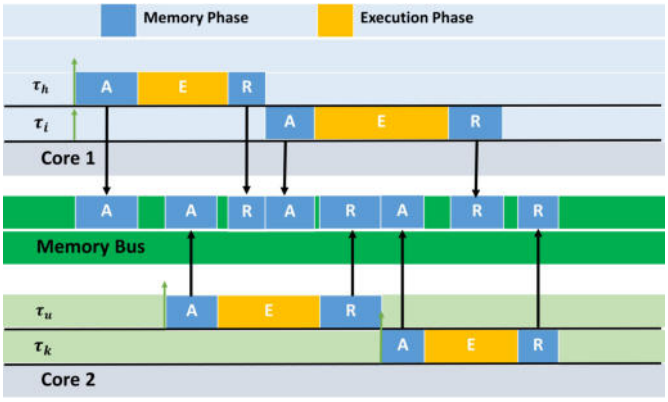


Figure 2: Example schedule using the 3-Phase task model to mitigate the problem of inter-core interference

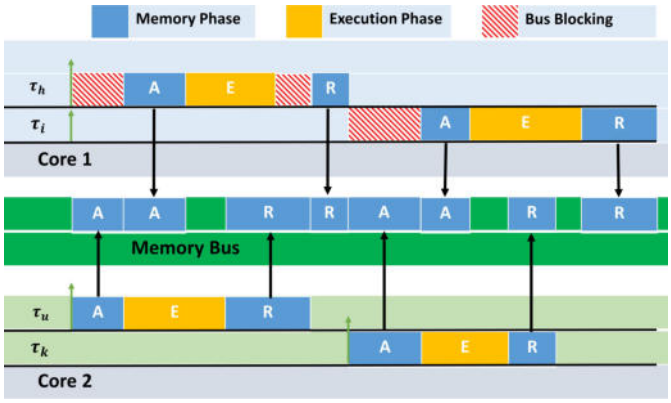


Figure 3: The problem of bus blocking in the 3-phase task model

algorithm to serialize the accesses to the bus/memory, thereby, eliminating/reducing contention. In this direction, works like [5, 6] have been proposed that generate an offline schedule of 3-phase tasks such that no two tasks can access the memory at the same time, thereby, eliminating inter-core bus/memory interference (e.g., see Figure 2). However, such solutions may not be applicable when an offline schedule cannot be forced due to the event-triggered/time-triggered nature of tasks. Consequently, when commonly used priority-based scheduling schemes are considered, 3-phase/PREM tasks may still compete to access the shared bus to execute their memory phases. For example, in Figure 3, task τ_i suffers bus contention from task τ_k when trying to execute its A-phase as the bus was busy serving a memory phase of task τ_k. The additional execution delay suffered by τ_i in Figure 3 is referred to as ***bus blocking***[1].

Bus blocking suffered by tasks executing on a multicore platform can have a significant impact on their schedulability. Hence, several works have been proposed in the state-of-the-art that focus on bounding bus blocking for the conventional task model [7, 8, 9, 10, 11, 12, 13, 14, 15, 16]. Similarly, works like [17] focused on bounding bus blocking for the 3-phase task model under global fixed priority scheduling. Therefore, in this work we focus on analysing bus blocking and deriving the worst-case response time (WCRT) for the 3-phase task model considering fixed-priority *partitioned* scheduling. Specifically, we show how bus blocking suffered by 3-phase tasks can be bounded under two different memory access models, namely, the *Dedicated Memory Access Model* (DMAM) and the *Fair Memory Access Model* (FMAM). In the DMAM, a core permitted to access the memory bus is allowed to execute more than one ready memory phase. This can improve the throughput of the system by executing multiple memory phases, e.g., R-phase followed by an A-phase, of tasks within a single memory access allocated to the processing core. On the other hand, the FMAM facilitates the fair distribution of the memory resources, i.e., memory bus/main memory, among all the cores that results in improving predictability. This is achieved by allowing a core to execute at most one memory phase, i.e., A- or R-phase, when it is granted an access to the bus. We will first show how the maximum bus blocking can be upper bounded under both these memory access models and then show how it can be integrated into the worst-case response time analysis of tasks.

Contributions: This paper has the following contributions:

1. We propose a fine-grained analysis to compute the maximum bus blocking for 3-phase tasks scheduled using partitioned fixed-priority scheduling. We show that the bus blocking suffered by the task can be different when considering different memory access models. As a consequence, we formulate the bus blocking analysis considering both DMAM and FMAM models.
2. We derive a schedulability test for the fixed-priority 3-phase task model by integrating the impact of maximum bus blocking into the WCRT analysis of each task.
3. We compare our presented analyses against the state-of-the-art by means of case study experiments, i.e., performed using Mälardalen Benchmarks suite [18], as well as through empirical evaluation, i.e.,

[1]State-of-the-art approaches used terms such as shared resource contention, bus contention, memory bus contention, bus interference, memory contention which is similar to the term bus blocking in our work.

using synthetic task sets. Results show that our presented analysis tightly bounds the bus blocking and improves task set schedulability by up to 88 percentage points.

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 problem formulation. The bus blocking analysis for the DMAM is presented in Section 5, followed by the bus blocking analysis for the FMAM in Section 6. The bus-contention aware schedulability analysis is presented in Section 7. The experimental results are presented in Section 8. Section 9 presents the related work and Section 10 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), large enough to store the data/instructions of the largest task in the taskset. Tasks are partitioned to cores at design-time and cannot migrate to any other core at run-time. Similarly to existing works [7, 10, 12, 13, 14, 17, 16], we assume a single-channel *shared memory bus* that connects all the cores to the main memory and the memory bus can only handle one memory phase[2] at a time, i.e., only one task can access the main memory at a time. A memory phase cannot be preempted once it accesses the memory bus to perform memory transactions. Furthermore, we assume that the memory bus arbitration policy is First-Come First-Served (FCFS) which is a work-conserving policy.

2.1. Task Model

We consider a task set Γ comprising n sporadic tasks from which the subset Γ' is assigned to each core according to the given task-to-core mapping strategy. Each task τ_i is characterized by its minimum inter-arrival time T_i and its 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* of a given core π_l is given by $\sum_{\tau_i \in \Gamma'_l} U_i$. 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 the taskset Γ is given by $\sum_{\tau_i \in \Gamma} \frac{C_i^A + C_i^R}{T_i}$. Each task releases potentially infinite number of jobs where each job instance is denoted by k. The response time of the k^{th} job of task τ_i is defined as the time difference between its release and its completion. 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 $R_{i,l}^{max}$ which is given by maximizing $R_{i,k,l}$ over all jobs of τ_i within a time interval of a given length.

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) executing 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.

For clarity, throughout the document, we refer to the core on which task τ_i (i.e., the task under analysis) executes as the *local core*, denoted by π_l. Similarly, any core other than the local core is referred to as a *remote core* and denoted by π_r.

2.2. Execution Model

In the 3-phase task model, the A-phase executes first to fetch tasks' data/instructions from the main memory and store them in the core's local memory. Then, the E-phase executes the task's code from the core's local memory. 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 memory bus is accessed

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

Symbol	Description
τ_i	i^{th} task
T_i	Minimum inter-arrival time between any two consecutive jobs of τ_i
D_i	Relative deadline of τ_i
C_i	WCET of τ_i in isolation
C_i^A	WCET of the A-phase of τ_i in isolation
C_i^E	WCET of the E-phase of τ_i in isolation
C_i^R	WCET of the R-phase of τ_i in isolation
U_i	Utilization of task τ_i
π_l	Local core (i.e., the core on which τ_i is running)
π_r	Remote core (i.e., any core other than the local core)
$hep_{i,l}$	Set of tasks with priority higher than or equal than that of τ_i running on core π_l
$hp_{i,l}$	Set of tasks with priority higher than that of τ_i running on core π_l
$lp_{i,l}$	Set of tasks with priority lower than that of τ_i running on core π_l
Γ'_l	Set of tasks assigned to the local core π_l
Γ'_r	Set of tasks assigned to a remote core π_r
$N_{\pi_l}(\Delta)$	The maximum number of times that tasks executing on core π_l can suffer bus blocking during any time interval of length Δ
$N_{\pi_r}(\Delta)$	The maximum number of times that tasks running on a core π_r can cause bus blocking during any time interval of length Δ
$Bus_{i,r}(\Delta)$	Maximum bus blocking suffered by τ_i due to tasks running on a remote core π_r during any time interval of length Δ
$Bus_{i,l}^{max}(\Delta)$	Total bus blocking suffered by τ_i due to tasks running on all remote cores during any time interval of length Δ
$W_{i,l}$	Level-i busy window for task τ_i executing on core π_l
$R_{i,k,l}$	Response time of k^{th} job of τ_i executing on core π_l
$R_{i,l}^{max}$	WCRT of τ_i

Table 1: Table of Symbols

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 assumed that at most one phase can execute on a given core at a time. The WCET of the E-phase is assumed to be greater than 0. This assumption is inline with the state-of-the-art [2].

Each core maintains its own *ready queue* with tasks that are ready to execute, sorted by task priority. Whenever a task in the queue becomes ready to execute, the core requests access to the memory bus and if the memory bus is available, the core executes the A-phase of that task. However, if the memory bus is busy serving a memory phase from another core, then the core busy-waits until the bus becomes available, at which point it executes the A-phase of the task with the highest priority in its ready queue. Once the A-phase of a task completes, the E-phase of the same task starts executing immediately on the core. Once 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 again if the bus is busy serving memory requests from other co-running tasks. Once the bus becomes available, the task can execute its R-phase and finalize its execution. Note that under the considered execution model, due to its non-preemptive nature, 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.

2.3. Memory Access Models

We consider two memory access models detailed as follows:

Dedicated Memory Access Model (DMAM): When a 3-phase task is scheduled using non-preemptive scheduling, after the completion of its A-phase, it will immediately start its E-phase followed by the R-phase. However, once the R-phase of a task completes, we may have an A-phase of a subsequent task ready to execute. At this point, the bus/memory scheduler has to decide whether it will execute the A-phase of the subsequent task on the same core or it will allocate the memory access to a different core. In the DMAM, the bus scheduler ensures that if a core has a ready A-phase after the completion of an R-phase, the A-phase must be served before allocating the bus to any other core. The main idea of the DMAM is to allow each core to execute all its pending memory phases within an access to the memory bus. However, due to the 3-phase task model, a core can execute at most one R- and one A-phase in an bus access, as the core has to release the bus during the E-phase execution. Once the memory phase(s) of the given core is served, the bus access can be granted to other cores. This type of memory access model can be useful in systems in which cores can made to execute a set of pending memory phases when access to the bus is granted.

Fair Memory Access Model (FMAM): In the FMAM, each core can execute at most one memory phase (i.e., either A- or R-phase) when access to the bus is granted and another core is waiting to access the memory bus to execute a memory phase. After the completion of the memory requests of a memory phase, the bus can be granted to other cores. Due to the work-conserving nature of the FCFS bus arbitration policy, a core can execute another memory phase after the completion of a memory phase if other cores are not waiting to access the memory bus.

3. Background

In this section, we introduce the WCRT analysis of Fixed-Priority Non-Preemptive (FPNP) scheduling for single-core systems. This analysis is later used to build the proposed memory bus-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 [19], which is defined as follows.

Definition 3.1. [Level-i busy window (from [20])] 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 following terms:

1. *Maximum blocking* that can be suffered by task τ_i from the tasks in lp_i taskset, and
2. *Maximum interference* that can be suffered by task τ_i from all the tasks in hep_i taskset during the level-i busy window.

Maximum Blocking Computation: In FPNP scheduling, only one job of a lower priority task in lp_i can block the execution of task τ_i [21, 19]. 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:

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

Maximum Interference Computation: Task τ_i can suffer interference from all higher or equal priority tasks in hep_i (including own jobs of τ_i) that execute during 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 [11].

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. This implies that the maximum number

of jobs released by task τ_h in any time interval of length Δ is given by $\eta_h^+(\Delta)$. Consequently, the maximum interference that can be caused by a task $\tau_h \in hep_i$ in any time interval of length Δ is upper bound by $\eta_h^+(\Delta) \times C_h$. Therefore, the maximum interference task τ_i can suffer during any time interval of length Δ due to the execution of tasks in hep_i is given by the following equation.

$$\sum_{\tau_h \in hep_i} (\eta_h^+(\Delta) \times C_h) \tag{2}$$

Using the upper bounds on the maximum blocking and maximum interference that a given 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:

$$W_i = C_{lp_i}^{max} + \sum_{\tau_h \in hep_i} (\eta_h^+(W_i) \times C_h) \tag{3}$$

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.

Having computed the length of the longest level-i busy window W_i using Equation 3, the maximum number of jobs of task τ_i that can execute within W_i is given by:

$$K_i = \eta_i^+(W_i) \tag{4}$$

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.

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:

$$s_{i,k} = C_{lp_i}^{max} + (k-1) \times C_i + \sum_{\tau_h \in hep_i \setminus \tau_i} \eta_h^+(s_{i,k}) \times C_h \tag{5}$$

where $C_{lp_i}^{max}$ is given by Equation 1, $\sum_{\tau_h \in hep_i \setminus \tau_i} \eta_h^+(s_{i,k}) \times C_h$ 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 both sides of Equation 5, it can be solved iteratively by initializing $s_{i,k} = C_{lp_i}^{max} + \sum_{\tau_h \in hep_i \setminus \tau_i} C_h$. The latest start time of k^{th} job of τ_i is then given by the smallest value of $s_{i,k}$ for which Equation 5 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.,

$$R_{i,k} = s_{i,k} + C_i \tag{6}$$

Finally, the WCRT of task τ_i is computed by maximizing Equation 6 over all the jobs of τ_i that can execute in the level-i busy window, i.e., from 1 to K_i,

$$R_i^{max} = \max_{k \in [1, K_i]} \{R_{i,k}\} \tag{7}$$

4. Problem Formulation

Unlike single core systems, in multicore platforms multiple cores can execute tasks at the same time. As all the cores share the memory bus to read/write the code/data from/to the main memory, tasks can suffer bus blocking if the requesting task has to wait for the bus while bus is busy serving memory requests of tasks executing on other cores.

Consequently, for any task τ_i executing on core π_l of a multicore platform, the length of its level-i busy window does not only depend on the tasks executing on the same core as τ_i but also on the bus blocking that τ_i can suffer during its execution from tasks executing on other cores. Formally, let $W_{i,l}$ be the length of level-i busy window of task τ_i when it executes on core π_l, where $W_{i,l}$ is given by the first positive fixed-point solution of the following equation:

$$W_{i,l} = C^{max}_{lp_{i,l}} + Bus^{max}_{i,l}(W_{i,l}) + \sum_{\tau_h \in hep_{i,l}} (\eta^+_h(W_{i,l}) \times C_h) \tag{8}$$

In Equation 8, $C^{max}_{lp_{i,l}}$ represents the maximum blocking caused by one job of a lower priority task in $lp_{i,l}$ on τ_i, which can be computed using Equation 1. Similarly, the term $\sum_{\tau_h \in hep_{i,l}} (\eta^+_h(W_{i,l}) \times C_h)$ captures the maximum interference that can be caused by all tasks in $hep_{i,l}$ (including own jobs of τ_i) during any time interval of length $W_{i,l}$ and can be computed using Equation 2.

The term $Bus^{max}_{i,l}(W_{i,l})$ in Equation 8 represents an upper bound on the total bus blocking that task τ_i can suffer from all co-running tasks executing on all the other cores during $W_{i,l}$. As we briefly discussed in Section 1, the bus blocking of tasks depends on the underlying memory access model. Therefore, in Section 5 we will first detail how to upper bound the maximum bus blocking of tasks when considering the DMAM. In Section 6, we will do the same for the FMAM.

5. Bus Blocking Analysis for the Dedicated Memory Access Model (DMAM)

As defined in Section 2, the Dedicated Memory Access Model (DMAM) allows each core to execute at most one R- and one A-phase back-to-back without granting the bus access to any other waiting core. Consequently, an A-phase cannot suffer bus blocking when it executes immediately after the completion of an R-phase running on the same core. An example scenario is shown in Figure 4c in which task τ_k running on the remote core π_r does not suffer any bus blocking before its A-phase as it executes immediately after the R-phase of τ_u on core π_r. We will explain the computation of maximum bus blocking for DMAM in this section.

Before explaining the proposed bus blocking analysis, we first present important properties on the DMAM that will be useful for deriving the maximum bus blocking in the next subsection.

5.1. Properties of the Dedicated Memory Access Model (DMAM)

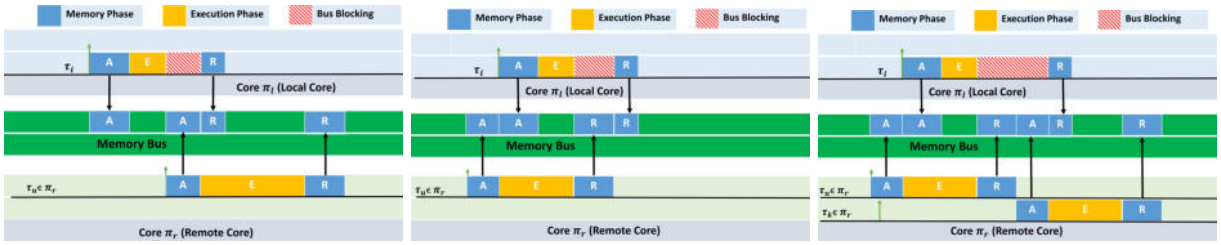
Property 5.1. For each bus blocking suffered by a job on the local core, a remote core can cause at most one bus blocking, either from one memory phase (A or R-phase) of a job or from one R and one A-phase of two different jobs running on that remote core.

Proof. When a job of task τ_i running on the local core requests access to the bus, the following scenarios are possible.

Scenario 1: A job of task τ_u running on the remote core is already executing its A-phase. Consequently, a job of task τ_i on the local core can only access the bus after the completion of the A-phase of the job of task τ_u currently executing on the remote core. Therefore, in this scenario, the bus blocking that can be caused by the remote core to one job running of task τ_i on the local core is equivalent to the WCET of the A-phase of task τ_u executing on the remote core. This scenario is depicted in Figure 4a.

Scenario 2: A job of task τ_u running on the remote core is executing its R-phase and the ready queue of the remote core is empty. In this scenario, the bus blocking caused by the remote core to a job executing on the local core is equivalent to the WCET of the R-phase of task τ_u executing on a remote core as depicted in Figure 4b.

Scenario 3: A job of task τ_u 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 in the remote core's ready-queue will execute immediately. Thus, the bus will only be released after the execution of one R and one A-phase of two different jobs of the remote core. In this case, the bus blocking



(a) Scenario 1 (b) Scenario 2 (c) Scenario 3

Figure 4: Bus blocking caused by a remote core for each bus blocking suffered at the local core

caused by the remote core is equivalent to the sum of the WCET of one R-phase and one A-phase of two different jobs running on that remote core. See Figure 4c 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 a memory phase (A or R-phase) of one job or a combination of one R and one A-phase of two different jobs running on that remote core. The property follows. □

Property 5.2. When a single job of a task on the remote core participates in one bus blocking, it can only participate by its A-phase or its R-phase.

Proof. Directly follows from Property 5.1. □

5.2. Bounding the Number of Bus Blockings for the Dedicated Memory Access Model (DMAM)

As discussed earlier, in multicore systems, all the jobs that execute on the local core π_l during 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.

Without loss of generality, we start by computing the maximum bus blocking that can be suffered by the local core π_l from a remote core π_r in any time interval of length $W_{i,l}$ (i.e., the longest level-i busy window on core π_l). We later generalize our analysis to account for the bus blocking that can be suffered by the local core π_l from all remote cores in Section 5.4.

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

- $N_{\pi_l}(W_{i,l})$: the maximum number of times that *tasks executing on the local core π_l can suffer bus blocking in a time window of length $W_{i,l}$*. This value is computed in Lemma 1.
- $N_{\pi_r}(W_{i,l})$: the maximum number of times that *tasks running on a remote core π_r can cause bus blocking during $W_{i,l}$*. This value is computed in Lemma 2.

Lemma 1. *The maximum number of times that tasks executing on the local core π_l can suffer bus blocking in any time window of length $W_{i,l}$ is upper bounded by:*

$$N_{\pi_l}(W_{i,l}) = \sum_{\tau_h \in hep_{i,l}} \eta_h^+(W_{i,l}) + 1 \tag{9}$$

Proof. By the definition of the level-i busy window, there is always a pending A-phase whenever an R-phase completes its execution; otherwise the level-i busy window would terminate with the execution of the R-phase. Also, knowing that under the DMAM, each core can simultaneously execute the R- and A-phases of two subsequent jobs, each job that executes during the level-i busy window (except for the first job) can suffer bus blocking only once, i.e., before its R-phase. The A-phases of all such jobs will not suffer any bus blocking because they will execute immediately after the R-phases of a previous job. Therefore, the maximum number of bus blockings that can be suffered by all jobs (except the first job) of all tasks that execute on core π_l during $W_{i,l}$ is upper bounded by $\sum_{\tau_h \in hep_{i,l}} \eta_h^+(W_{i,l})$.

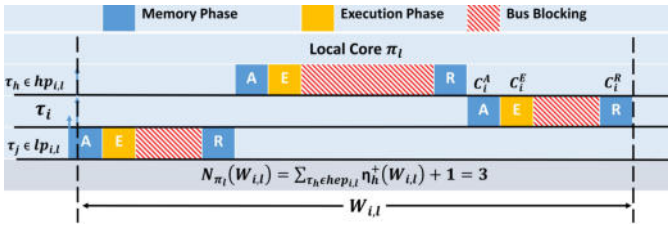


Figure 5: Maximum number of bus blockings when $\tau_j \in lp_{i,l}$ executes at the start of $W_{i,l}$

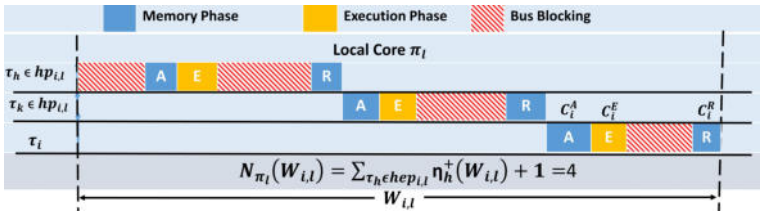


Figure 6: Maximum number of bus blockings when $\tau_h \in hp_{i,l}$ executes at the start of $W_{i,l}$

The additional 1 in Equation 9 represents two possible execution scenarios:

Scenario 1: If task τ_i is not the lowest priority task, one job of a lower priority task $\tau_j \in lp_{i,l}$ can cause blocking to tasks in $hep_{i,l}$, in the scenario when $\tau_j \in lp_{i,l}$ has started its execution before the start of the level-i busy window, i.e., starting by the execution of its A-phase. Consequently, the additional 1 in the Equation 9 accounts for the bus blocking that can be suffered by $\tau_j \in lp_{i,l}$ before executing its R-phase which can impact the length of the level-i busy window, e.g., see Figure 5.

Scenario 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 before its A-phase. In this scenario, the additional 1 accounts for the bus blocking suffered by the first job of task $\tau_h \in hep_{i,l}$ before starting its A-phase on core π_l, e.g., see Figure 6.

Hence, the maximum number of bus blockings that can be suffered by all tasks executing on the local core π_l during $W_{i,l}$ is upper bounded by $\sum_{\tau_h \in hep_{i,l}} \eta_h^+(W_{i,l}) + 1$. The Lemma follows. □

Lemma 2. *The maximum number of times that tasks running on a remote core π_r can cause bus blocking in any 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:*

$$N_{\pi_r}(W_{i,l}) = \sum_{\tau_u \in \Gamma'_r} \eta_u^+(W_{i,l}) \tag{10}$$

Proof. According to the system model, each job consists of two memory phases (i.e., one A- and one R-phase) that can cause bus blocking. As we cannot predict the schedule of a remote core, the bus blocking can be caused by all the jobs released on a remote core π_r in any time window of length $W_{i,l}$, where each bus blocking caused by a remote core π_r can be composed of one R and one A-phase. From the upper event arrival function, we know that the maximum number of jobs that can be released by a task τ_u on core π_r in any time window of length $W_{i,l}$ is upper-bounded by $\eta_u^+(W_{i,l})$. Consequently, the maximum number of bus blockings that can be caused by a task τ_u on core π_r during any time interval of length $W_{i,l}$ is also upper-bounded by $\eta_u^+(W_{i,l})$. Since any task released on core π_r during $W_{i,l}$ can participate in the bus blocking, the maximum number of bus blockings that can be caused by a remote core π_r can be bounded by considering all the jobs of all the tasks released on core π_r in any time window of length $W_{i,l}$. Thus, $N_{\pi_r}(W_{i,l})$ is upper bounded by $\sum_{\tau_u \in \Gamma'_r} \eta_u^+(W_{i,l})$. The Lemma follows. □

5.3. Maximum Bus Blocking Computation for the Dedicated Memory Access Model (DMAM)

Having bounded the values of $N_{\pi_l}(W_{i,l})$ and $N_{\pi_r}(W_{i,l})$, it is possible to compute the maximum bus blocking that can be suffered by tasks running on core π_l during $W_{i,l}$ from co-running tasks executing on a

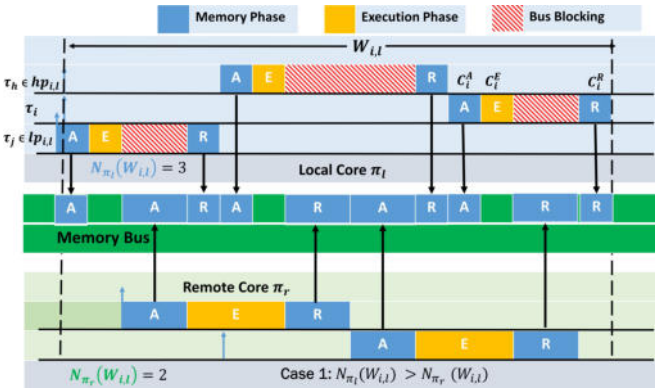


Figure 7: Maximum bus blocking for $N_{\pi_l}(W_{i,l}) > N_{\pi_r}(W_{i,l})$

remote core π_r. Before explaining how the maximum bus blocking can be computed, we first define some notations that will be used during computation.

Let M_r^A (resp. M_r^R) be an ordered set that contains the WCET 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 non-increasing order* as follows:

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

where $\hat{N}_{\pi_r}$ is equal to the value of $N_{\pi_r}(W_{i,l})$ computed using Equation 10. Note that $C_{r,x}^A$ and $C_{r,y}^R$ may belong to the same/different jobs released on core π_r during $W_{i,l}$.

We compute the maximum bus blocking for the DMAM using the following three cases.

(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 running on core π_r in any time window of length $W_{i,l}$.

(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 running on core π_r in any time window of length $W_{i,l}$.

(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 running on core π_r in any time window of length $W_{i,l}$.

5.3.1. ***Maximum Bus Blocking Computation for Case 1***

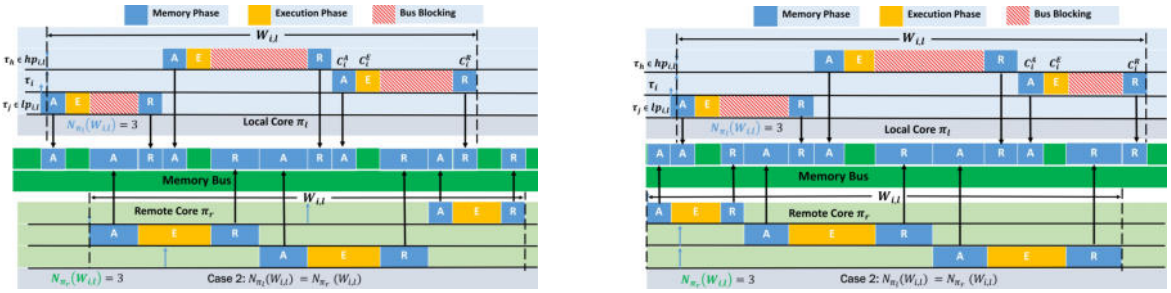
For $N_{\pi_l}(W_{i,l}) > N_{\pi_r}(W_{i,l})$, all memory phases of all jobs released on core π_r during $W_{i,l}$ can contribute to the bus blocking (e.g., see Figure 7). This leads to the following lemma.

Lemma 3. *If $N_{\pi_l}(W_{i,l}) > N_{\pi_r}(W_{i,l})$, then the maximum bus blocking suffered by tasks executing on the local core π_l due to tasks running on a remote core π_r in any time interval $W_{i,l}$ is upper bounded by $Bus_{i,r}(W_{i,l})$, given by:*

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

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

Proof. As proven in Property 5.1, under the DMAM, each bus blocking caused by a remote core 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 during $W_{i,l}$. Since the precise bus access times of tasks running on core π_r are unknown, if $N_{\pi_l}(W_{i,l}) > N_{\pi_r}(W_{i,l})$, then in the worst-case all the memory phases of all jobs that execute on π_r during $W_{i,l}$ can cause bus blocking to all tasks executing on π_l during $W_{i,l}$ (e.g., see Figure 7). Therefore, if $N_{\pi_l}(W_{i,l}) >$



(a) Possible scenario 1

(b) Possible scenario 2

Figure 8: Possible scenarios when $N_{\pi_l}(W_{i,l}) = N_{\pi_r}(W_{i,l})$

$N_{\pi_r}(W_{i,l})$, the maximum contribution of the memory phases of $\hat{N}_{\pi_r}$ jobs, i.e., $\sum_{x=1}^{\hat{N}_{\pi_r}} C^A_{r,x} + \sum_{y=1}^{\hat{N}_{\pi_r}} C^R_{r,y}$, upper bounds the maximum bus blocking. The Lemma follows. □

5.3.2. *Maximum Bus Blocking Computation for 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 explain, assume that the number of bus blockings that can be suffered (resp. caused) by tasks executing on core π_l (resp. core π_r) during $W_{i,l}$ 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 during $W_{i,l}$ (e.g., see Figure 8a) or the A-phase of the first job that executes on core π_r during $W_{i,l}$ (e.g., see Figure 8b) cannot participate in the bus blocking. This leads to the following Lemma.

Lemma 4. *If $N_{\pi_l}(W_{i,l}) = N_{\pi_r}(W_{i,l})$, then the maximum bus blocking suffered by tasks executing on the local core π_l due to tasks running on a remote core π_r in any time interval $W_{i,l}$ is upper bounded by $Bus_{i,r}(W_{i,l})$, given by:*

$$Bus_{i,r}(W_{i,l}) = \sum_{x=1}^{\hat{N}_{\pi_r}} C^A_{r,x} + \sum_{y=1}^{\hat{N}_{\pi_r}} C^R_{r,y} - \min(\min_{\forall x \in M^A_r}\{C^A_{r,x}\}, \min_{\forall y \in M^R_r}\{C^R_{r,y}\}) \tag{12}$$

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

Observation 1. If the A-phase of the first job on core π_r participates to the bus blocking of any job of core π_l released during $W_{i,l}$, then the first bus blocking is composed of only an A-phase (see Property 5.2) 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 Property 5.1). Consequently, the R-phase of the last job executing on core 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 during $W_{i,l}$, we assume that the job with the smallest R-phase is the last job that executes on core π_r during $W_{i,l}$, given by $\min_{\forall y \in M^R_r}\{C^R_{r,y}\}$, (e.g., see Figure 8a).

Observation 2. If the A-phase of the first job on core π_r does not block the memory phase of any job of core π_l released during $W_{i,l}$ (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 $W_{i,l}$ (see Property 5.1)), then all memory phases except the A-phase of the first job executed 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 and the length of that A-phase is given by $\min_{\forall x \in M^A_r}\{C^A_{r,x}\}$. See Figure 8b for an example scenario.

Building on the above observations, the maximum bus blocking $Bus_{i,r}(W_{i,l})$ is given by the sum of all the memory phases (expressed as $\sum_{x=1}^{\hat{N}_{\pi_r}} C^A_{r,x} + \sum_{y=1}^{\hat{N}_{\pi_r}} C^R_{r,y}$) *except* the smallest memory phase, i.e., either A- or R-phase (expressed as $\min(\min_{\forall x \in M^A_r}\{C^A_{r,x}\}, \min_{\forall y \in M^R_r}\{C^R_{r,y}\})$) of tasks released on core π_r during $W_{i,l}$. The Lemma follows. □

5.3.3. ***Maximum Bus Blocking Computation for 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 executing on core π_l during $W_{i,l}$. To extract the $N_{\pi_l}(W_{i,l})$ A and R-phases with the largest execution times among all jobs that execute on π_r during $W_{i,l}$, we first divide the set M_r^A (resp. M_r^R) into two subsets namely 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 largest execution times while 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:

$$
\begin{aligned}
M_r^{AH} =& \{C_{r,1}^A, C_{r,2}^A, \ldots, C_{r,\hat{N}_{\pi_l}}^A \mid C_{r,x}^A \geq C_{r,x+1}^A\} \\
M_r^{AL} =& \{C_{r,\hat{N}_{\pi_l}+1}^A, C_{r,\hat{N}_{\pi_l}+2}^A, \ldots, C_{r,\hat{N}_{\pi_r}}^A \mid C_{r,y}^A \geq C_{r,y+1}^A\} \\
M_r^{RH} =& \{C_{r,1}^R, C_{r,2}^R, \ldots, C_{r,\hat{N}_{\pi_l}}^R \mid C_{r,x}^R \geq C_{r,x+1}^R\} \\
M_r^{RL} =& \{C_{r,\hat{N}_{\pi_l}+1}^R, C_{r,\hat{N}_{\pi_l}+2}^R, \ldots, C_{r,\hat{N}_{\pi_r}}^R \mid C_{r,y}^R \geq C_{r,y+1}^R\}
\end{aligned}
$$

where $\hat{N}_{\pi_l} = N_{\pi_l}(W_{i,l})$ and can be computed using Equation 9.

We then identify two possible sub-cases:

Sub-case 3.1: All the elements of the M_r^{AH} and M_r^{RH} subsets can participate in the $\hat{N}_{\pi_l}$ number of bus blockings such that each bus blocking is composed of one R and one A-phase of tasks released on a remote core π_r during any time window of length $W_{i,l}$. 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} subsets. We discuss this sub-case in Lemma 5.

Sub-case 3.2: At least one element of the M_r^{AH} or M_r^{RH} subset cannot participate in the $\hat{N}_{\pi_l}$ number of bus blockings. This can only happen if all 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 6.

Lemma 5. *If all the elements of the M_r^{AH} and M_r^{RH} subsets can participate in the $\hat{N}_{\pi_l}$ number of bus blockings, then the maximum bus blocking suffered by tasks executing on the local core π_l due to tasks running on a remote core π_r in any time interval $W_{i,l}$ is upper bounded by $Bus_{i,r}(W_{i,l})$, given by*

$$
Bus_{i,r}(W_{i,l}) = \sum_{x=1}^{\hat{N}_{\pi_l}} C_{r,x}^A + \sum_{y=1}^{\hat{N}_{\pi_l}} C_{r,y}^R \tag{13}
$$

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

Proof. If all elements of M_r^{AH} and M_r^{RH} subsets can participate in $\hat{N}_{\pi_l}$ number of bus blockings caused by π_r such that each bus blocking is composed of one R- and one A-phase of two jobs, then all the memory phases of M_r^{AH} and M_r^{RH} can participate in the bus blocking. Since M_r^{AH} and M_r^{RH} are the subsets that contain memory phases with the largest execution times, the maximum bus blocking that can be caused by tasks running on core π_r to tasks running on core π_l in any time window of length $W_{i,l}$ is upper bounded by summing all the memory phases in the M_r^{AH} and M_r^{RH} subsets. The sum of the WCET of all the A-phases (resp. R-phases) in subset M_r^{AH} (resp. M_r^{RH}) is given by $\sum_{x=1}^{\hat{N}_{\pi_l}} C_{r,x}^A$ (resp. $\sum_{y=1}^{\hat{N}_{\pi_l}} C_{r,y}^R$). Consequently, Equation 13 bounds the maximum bus blocking for this sub-case. The Lemma follows. □

Lemma 6. *If at least one element of the M_r^{AH} or M_r^{RH} subset cannot participate in the $\hat{N}_{\pi_l}$ number of bus blockings, then the maximum bus blocking suffered by tasks executing on the local core π_l due to tasks running on a remote core π_r in any time interval $W_{i,l}$ is upper bounded by $Bus_{i,r}(W_{i,l})$, given by*

$$
Bus_{i,r}(W_{i,l}) = \sum_{x=1}^{\hat{N}_{\pi_l}} C_{r,x}^A + \sum_{y=1}^{\hat{N}_{\pi_l}} C_{r,y}^R - \min\Big((\min_{\forall x \in M_r^{AH}} \{C_{r,x}^A\} - \max_{\forall y \in M_r^{AL}} \{C_{r,y}^A\}), (\min_{\forall x \in M_r^{RH}} \{C_{r,x}^R\} - \max_{\forall y \in M_r^{RL}} \{C_{r,y}^R\}) \Big) \tag{14}
$$

where $\min_{\forall x \in M_r^{AH}} \{C_{r,x}^A\}$ (resp. $\min_{\forall x \in M_r^{RH}} \{C_{r,x}^R\}$) returns the smallest element of M_r^{AH} (resp. M_r^{RH}); and $\max_{\forall y \in M_r^{AL}} \{C_{r,y}^A\}$ (resp. $\max_{\forall y \in M_r^{RL}} \{C_{r,y}^R\}$) 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 can be 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} subsets 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 it is not possible to obtain $\hat{N}_{\pi_l}$ bus blockings such that each bus blocking is composed of one R- and one A-phase of two different jobs of core π_r. In such a scenario, at-least one memory phase from either M_r^{AH} or M_r^{RH} cannot participate to the bus blocking. This happens because either an A-phase (i.e., an element from M_r^{AH}) or an R-phase executing on π_r (i.e., an element from M_r^{RH}) cannot participate to the bus blockings. As $N_{\pi_l}(W_{i,l}) < N_{\pi_r}(W_{i,l})$, one memory phase from M_r^{AL} or M_r^{RL} subset can participate such that $\hat{N}_{\pi_l}$ 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. This is achieved by first considering the term $\sum_{x=1}^{\hat{N}_{\pi_l}} C_{r,x}^A + \sum_{y=1}^{\hat{N}_{\pi_l}} C_{r,y}^R$ which sums all the elements of M_r^{AH} and M_r^{RH} subset. Then, the next step is to remove an element from M_r^{AH} or M_r^{RH} and add an element from M_r^{AL} or M_r^{RL} such that the bus blocking is maximized. This is achieved by first computing the difference between the smallest element of M_r^{AH} (resp. M_r^{RH}) and largest element of M_r^{AL} (resp. M_r^{RL}), expressed as $(\min_{\forall x \in M_r^{AH}} \{C_{r,x}^A\} - \max_{\forall y \in M_r^{AL}} \{C_{r,y}^A\}), (\min_{\forall x \in M_r^{RH}} \{C_{r,x}^R\} - \max_{\forall y \in M_r^{RL}} \{C_{r,y}^R\})$. Finally, we take minimum of the difference between the smallest element of M_r^{AH} (resp. M_r^{RH}) and largest element of M_r^{AL} (resp. M_r^{RL}) and subtract it from the sum of the WCET of all the elements of M_r^{AH} and M_r^{RH}. The Lemma follows. □

5.4. Bus Blocking Analysis for all Remote Cores

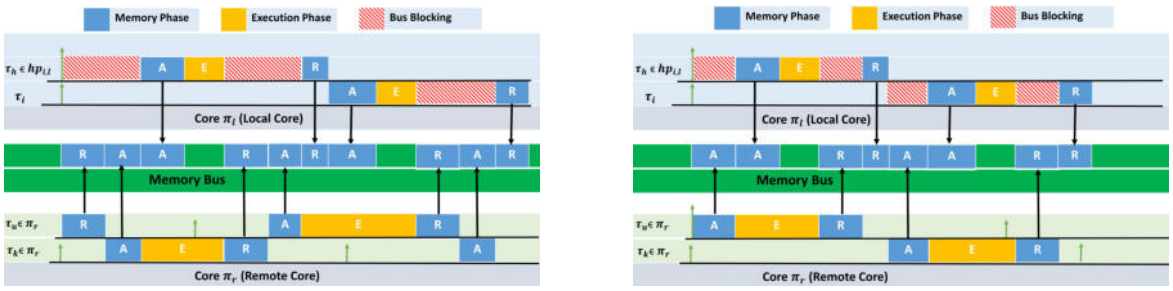
Under the FCFS bus arbitration policy, a task τ_i executing on the local core π_l will suffer the worst-case bus blocking when tasks released on all other remote cores, i.e., $\forall \pi_r \in m \setminus \pi_l$, execute their memory phases before τ_i. Considering that when we only have one remote core π_r bus blocking can be derived using Lemma 1 to Lemma 6. Similarly, to consider the worst-case under the FCFS bus arbitration, we need to repeat the same procedure for each remote core with respect to the core under analysis.

The total bus blocking that can be suffered by tasks that execute on the local core π_l during $W_{i,l}$ due to tasks running on *all remote cores* is denoted by $Bus_{i,l}^{max}(W_{i,l})$ and is computed using Algorithm 1.

Algorithm 1 Computing the total bus blocking that can be suffered by tasks that execute on the local core π_l due to tasks running on all remote cores during $W_{i,l}$

1: $Bus_{i,l}^{max}(W_{i,l}) := 0$
2: **for** $\pi_r \in [1, m]$ such that $\pi_r \neq \pi_l$ **do**
3: $\quad Bus_{i,r}(W_{i,l}) := 0$
4: $\quad$ Compute $N_{\pi_l}(W_{i,l})$ using Lemma 1.
5: $\quad$ Compute $N_{\pi_r}(W_{i,l})$ using Lemma 2.
6: $\quad$ Compute $Bus_{i,r}(W_{i,l})$ using Lemma 3 up to Lemma 6.
7: $\quad Bus_{i,l}^{max}(W_{i,l}) += Bus_{i,r}(W_{i,l})$
8: **end for**
9: Total bus blocking suffered by core π_l during $W_{i,l}$ due to all remote cores is given by $Bus_{i,l}^{max}(W_{i,l})$

Algorithm 1 iterates over all remote cores by first computing the value of $N_{\pi_l}(W_{i,l})$, and $N_{\pi_r}(W_{i,l})$, (line 4 and 5) for each remote core π_r. It then computes the maximum bus blocking $Bus_{i,r}(W_{i,l})$ that can be caused by tasks running on core π_r during $W_{i,l}$ using Lemma 3 to Lemma 6 (line 6). Finally, line 7 computes



(a) Bus blocking under DMAM

(b) Bus blocking under FMAM

Figure 9: Maximum bus blocking for DMAM and FMAM

the total bus blocking $Bus_{i,l}^{max}(W_{i,l})$ that can be suffered by tasks that execute on the local core π_l during any time interval of length $W_{i,l}$ due to tasks running on all remote cores by summing the bus blocking caused by each remote core $\pi_r \in [1, m]$ such that $\pi_r \neq \pi_l$.

6. Bus Blocking Analysis for the Fair Memory Access Model (FMAM)

In the DMAM, each core is allowed to execute up to two memory phases, i.e., the R-phase of a job and the A-phase of a subsequent next job, whenever it accesses the bus. However, to realize the DMAM in an actual system, a hardware/software mechanism will be required to manage the control of the bus, ensuring that each core will be able to execute an R-phase and an A-phase of any two jobs. Considering that the implementation of such hardware/software mechanism is non-trivial, a possible alternative is to use the Fair Memory Access Model (FMAM) that distributes the bus bandwidth among the cores in a fairer manner. The following example demonstrates an example scenario where the FMAM can tightly bound the bus blocking of tasks in comparison to the DMAM.

Example 1: *Let τ_i be the task under analysis which is executing on core π_l along with a higher priority task τ_h. Both τ_i and τ_h execute one job each during the level-i busy window $W_{i,l}$. During the same time interval of length $W_{i,l}$, tasks executing in parallel on core π_r releases several jobs, e.g., greater than 3. Figure 9a and 9b shows the task execution schedule under the DMAM and the FMAM, respectively.*

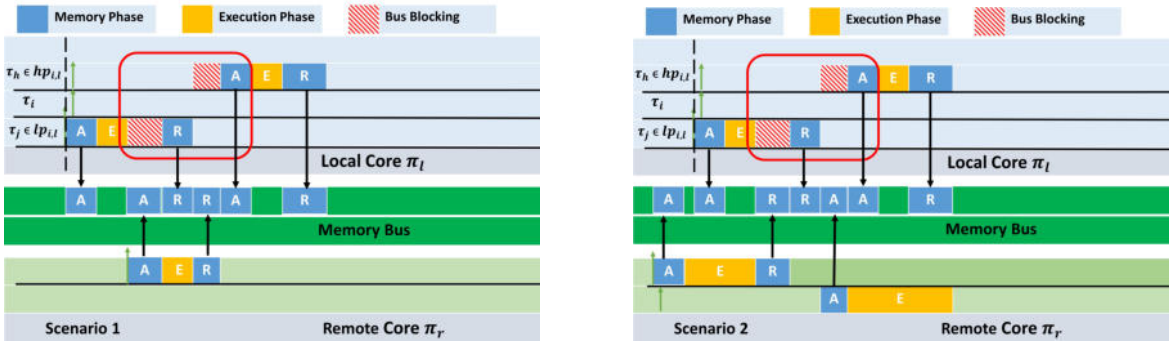
*Under the **Dedicated Memory Access Model (DMAM)**, for each bus blocking suffered by the tasks executing on the local core, there can be a combination of one R- and one A-phase of tasks released on the remote core π_r that can cause bus blocking. Knowing that two jobs are released on core π_l during the level-i busy window $W_{i,l}$ and τ_i is the lowest priority task on that core, the maximum number of bus blockings that can be suffered during $W_{i,l}$ are three (see Lemma 1). Consequently, considering that each bus blocking from the remote core π_r may be composed of two memory phases, i.e., an R-phase followed by an A-phase, as shown in Figure 9a. The worst-case bus blocking that will be suffered during $W_{i,l}$ will be equal to the sum of the WCET of six memory phases of tasks released on core π_r.*

*By definition of the **Fair Memory Access Model (FMAM)**, each core can execute only one memory phase during an access to the bus. So, in the worst-case, each memory phase that executes on the local core can suffer bus blocking from a memory phase executing on the remote core. Considering the scenario shown in Figure 9b, four memory phases are executed on core π_l during $W_{i,l}$. Therefore, the maximum bus blocking that can be suffered by all tasks executing on core π_l during $W_{i,l}$ is also upper bounded by the sum of the WCET of four memory phases that execute on core π_r during $W_{i,l}$.*

The simple example presented above shows that the FMAM can provide tighter estimates on the bus blocking suffered by the 3-phase tasks under an FCFS bus arbitration scheme. However, before we formally present the bus blocking analysis for the FMAM in Section 6.2 and 6.3, we will first introduce some properties pertaining to the model.

6.1. Useful Properties for the Fair Memory Access Model (FMAM)

Property 6.1. During the level-i busy window, the local core always executes an A-phase after the execution of an R-phase except for the A-phase of the first job and the R-phase of the last job that executes on the



(a) Scenario 1 of Property 6.2

(b) Scenario 2 of Property 6.2

Figure 10: Bus blocking suffered by a pair of one R and one A-phase on the local core

local core during the level-i busy window.

Proof. By the definition of the level-i busy window, the workload due to tasks in $hep_{i,l}$ taskset remains positive at all time instances within the level-i busy window except at the boundaries. So, within the level-i busy window whenever a job of tasks in $hep_{i,l}$ completes its R-phase, there is always a job that is ready to execute its A-phase; otherwise, the level-i busy window terminates with the execution of the R-phase. Therefore, within the level-i busy window, it is only the A-phase of the first job that does not execute after any R-phase on the local core because the level-i busy window begins with that A-phase. Similarly, the level-i busy window completes when the R-phase of the last job executes on the local core; thus, another A-phase does not execute. The property follows. □

Property 6.2. For each pair of R and A memory phases that are to be executed sequentially on the local core π_l during the level-i busy window, the bus blocking that can be caused by tasks executing on the remote core π_r will always be composed of one A-phase and one R-phase.

Proof. As proven in Property 6.1, during the level-i busy window, the local core always executes an A-phase after the execution of R-phase except the A-phase of the first job and the R-phase of the last job that executes during the level-i busy window. Now, if the bus blocking is suffered by both the memory phases in a pair (i.e., an R-phase followed by an A-phase), then the bus blocking that can be caused by tasks executing on the remote core π_r to that pair of R- and A-phases will also be composed of one A-phase and one R-phase. To explain further, consider the following scenarios.

Scenario 1: If an R-phase and a subsequent A-phase executing on the local core both suffer blocking from the remote core, then, if the blocking of the first R-phase is caused by an A-phase of the remote core, the bus blocking suffered by the next A-phase of the local core will intuitively be caused by R-phase of the remote core due to the 3-phase task model (e.g., see Figure 10a).

Scenario 2: If an R-phase and a subsequent A-phase executing on the local core both suffer blocking from the remote core, then, if the blocking of the first R-phase is caused by an R-phase of the remote core, the blocking suffers by the next A-phase of the local core will intuitively be caused by A-phase of the remote core due to the 3-phase task model (e.g., see Figure 10b).

Hence, for each pair of R and A memory phases that are to be executed sequentially on the local core π_l during the level-i busy window, the bus blocking that can be caused by tasks executing on the remote core π_r will always be composed of one A-phase and one R-phase. The property follows. □

6.2. Bounding the Number of Bus Blockings for the Fair Memory Access Model (FMAM)

Similarly to the DMAM, we first compute the values of $N_{\pi_l}(W_{i,l})$ and $N_{\pi_r}(W_{i,l})$ for the FMAM. The computation of $N_{\pi_l}(W_{i,l})$ and $N_{\pi_r}(W_{i,l})$ for the FMAM are given by the following lemmas.

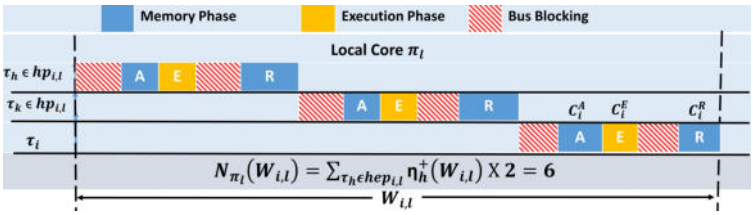


Figure 11: Maximum number of bus blockings suffered by the local core during $W_{i,l}$ when $lp_{i,l} = \emptyset$

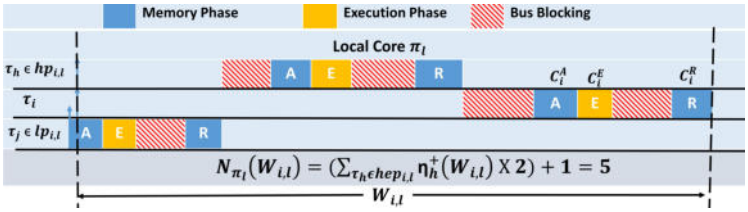


Figure 12: Maximum number of bus blockings suffered by the local core during $W_{i,l}$ when $lp_{i,l} \neq \emptyset$

Lemma 7. *The maximum number of times that tasks executing on the local core π_l can suffer bus blocking in any time interval of length $W_{i,l}$ is upper bounded by $N_{\pi_l}(W_{i,l})$, where $N_{\pi_l}(W_{i,l})$ is given by:*

$$N_{\pi_l}(W_{i,l}) = \begin{cases} (\sum_{\tau_h \in hep_{i,l}} \eta_h^+(W_{i,l}) \times 2) + 1, & \text{if } lp_{i,l} \neq \emptyset \\ \sum_{\tau_h \in hep_{i,l}} \eta_h^+(W_{i,l}) \times 2, & \text{otherwise} \end{cases} \tag{15}$$

Proof. We prove this lemma using two possible scenarios by considering the priority of $\tau_i \in \pi_l$:

Scenario 1. Task τ_i is the lowest priority task of the local core: In the FMAM, each core can execute at most one memory phase during an access to the bus. This also implies that each memory phase that executes on the local core π_l can suffer bus blocking. Knowing that each task in $hep_{i,l}$ that executes on the local core during $W_{i,l}$ can release at most $\eta_h^+(W_{i,l})$ jobs and each job has 2 memory phases (i.e., A-phase and R-phase), the maximum number of bus blockings that can be suffered by all the tasks in $hep_{i,l}$ during $W_{i,l}$ is upper bounded by $\sum_{\tau_h \in hep_{i,l}} \eta_h^+(W_{i,l}) \times 2$ (e.g., see Figure 11).

Scenario 2. Task τ_i is not the lowest priority task of the local core: If task τ_i is not the lowest-priority task, one job of a lower-priority task, e.g., $\tau_j \in lp_{i,l}$, can cause blocking to tasks in $hep_{i,l}$, when τ_j starts executing before the start of the level-i busy window. Nevertheless, there is the need to account for the bus blocking that can be suffered by τ_j while executing its R-phase as it can impact the length of the level-i busy window[3]. Therefore, the maximum number of bus blockings that can be suffered by tasks that execute on core π_l during any time interval of length $W_{i,l}$ is upper-bounded by $(\sum_{\tau_h \in hep_{i,l}} \eta_h^+(W_{i,l}) \times 2) + 1$ when $lp_{i,l} \neq \emptyset$ (e.g., see Figure 12). The Lemma follows. □

Lemma 8. *The maximum number of times that tasks running on a remote core π_r can cause bus blocking in any time interval 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:*

$$N_{\pi_r}(W_{i,l}) = \sum_{\tau_u \in \Gamma'_r} \eta_u^+(W_{i,l}) \times 2 \tag{16}$$

Proof. Due to the nature of the FMAM, each time the bus blocking is suffered by the local core, a remote core can cause bus blocking using one memory phase. Furthermore, the maximum number of jobs released by a task τ_u on a remote core π_r during $W_{i,l}$ is upper bounded by $\eta_u^+(W_{i,l})$. This implies that the maximum number of bus blockings that can be caused by a task τ_u released on a remote core π_r during $W_{i,l}$ is upper

[3]We do not need to account for the bus blocking that can be suffered by $\tau_j \in lp_{i,l}$ while executing its A-phase as the $\tau_j \in lp_{i,l}$ has started its A-phase execution before the start of the level-i busy window.

bounded by $\eta_u^+(W_{i,l}) \times 2$, i.e., using its A- and R-phases. Consequently, the maximum number of bus blockings that can be caused by all tasks released on a remote core π_r during $W_{i,l}$ is upper-bounded by $\sum_{\tau_u \in \Gamma'_r} \eta_u^+(W_{i,l}) \times 2$ where Γ'_r is the set of all tasks that are assigned to core π_r. The Lemma follows. □

6.3. Maximum Bus Blocking Computation for the Fair Memory Access Model (FMAM)

Having bounded the values of $N_{\pi_l}(W_{i,l})$ and $N_{\pi_r}(W_{i,l})$, we can now derive the maximum bus blocking $Bus_{i,r}(W_{i,l})$ that can be suffered by the local core π_l from a remote core π_r in any time interval of length $W_{i,l}$ using the following cases.

- **Case 1:** $N_{\pi_l}(W_{i,l}) \geq N_{\pi_r}(W_{i,l})$, i.e., the maximum number of bus blockings that can be suffered by tasks executing on core π_l is greater than or equal to the maximum number of bus blockings that can be caused by tasks running on core π_r in any time window of length $W_{i,l}$. The maximum bus blocking computation for this case is given in Lemma 9.
- **Case 2:** $N_{\pi_l}(W_{i,l}) < N_{\pi_r}(W_{i,l})$, i.e., 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 running on core π_r in any time window of length $W_{i,l}$. The maximum bus blocking computation for this case is given in Lemma 10.

Lemma 9. *If $N_{\pi_l}(W_{i,l}) \geq N_{\pi_r}(W_{i,l})$, then the maximum bus blocking suffered by tasks executing on the local core π_l due to tasks running on a remote core π_r in any time interval $W_{i,l}$ is upper bounded by $Bus_{i,r}(W_{i,l})$, given by:*

$$Bus_{i,r}(W_{i,l}) = \sum_{\tau_u \in \Gamma'_r} \eta_u^+(W_{i,l}) \times (C_u^A + C_u^R) \tag{17}$$

Proof. By Lemma 7, we know that the local core can suffer at most $N_{\pi_l}(W_{i,l})$ bus blockings that are caused by tasks running on a remote core π_r during $W_{i,l}$. As the exact schedule of the tasks executing on a remote core cannot be predicted, for $N_{\pi_l}(W_{i,l}) \geq N_{\pi_r}(W_{i,l})$, all the bus blockings caused by core π_r in any time interval of length $W_{i,l}$ can impact the tasks that execute on the local core π_l. Consequently, the maximum bus blocking that can be caused by a task τ_u released on a remote core π_r during $W_{i,l}$, is upper-bounded by the maximum number of jobs released during $W_{i,l}$ times the sum of the WCET of its memory phases, i.e., $\eta_u^+(W_{i,l}) \times (C_u^A + C_u^R)$. Extending this result for all tasks, the maximum bus blocking $Bus_{i,r}(W_{i,l})$ that can be suffered by the local core π_l due to the execution of all tasks released on a remote core π_r during $W_{i,l}$ is upper bounded by $\sum_{\tau_u \in \Gamma'_r} \eta_u^+(W_{i,l}) \times (C_u^A + C_u^R)$. The Lemma follows. □

For case 2, as the maximum number of bus blockings that can be suffered by the local core is less than the maximum number of bus blockings that can be caused by a remote core during $W_{i,l}$, we need to extract a set of memory phases released by all tasks on a remote core during $W_{i,l}$ that provide a safe and tighter bound on the bus blocking. To do this, we first introduce the following notations:

Let $\hat{P}$ denote the maximum number of jobs released by all the tasks in $hep_{i,l}$ during any time interval of length $W_{i,l}$, i.e., $\hat{P} = \sum_{\tau_h \in hep_{i,l}} \eta_h^+(W_{i,l})$. Let $\hat{Q}$ denote the maximum number of jobs released by all tasks on a remote core π_r during any time interval of length $W_{i,l}$ i.e., $\hat{Q} = \sum_{\tau_u \in \Gamma'_r} \eta_u^+(W_{i,l})$. These terms will be later used to extract a given number of memory phases among all the memory phases released on a remote core during $W_{i,l}$.

Now, we define M_r^{AH} and M_r^{RH} as ordered sets that contain the $\hat{P}$ largest A-phases and R-phases, respectively, released on core π_r in a time window of length $W_{i,l}$. We assume that M_r^{AH} and M_r^{RH} are sorted in a non-increasing order. Additionally, we define M_r^{AL} and M_r^{RL} as ordered sets that contain remaining A- and R-phases, respectively, released on core π_r in a time window of length $W_{i,l}$, sorted in a

non-increasing order:

$$
\begin{aligned}
M_r^{AH} =& \{C_{r,1}^A, C_{r,2}^A, \ldots, C_{r,\hat{P}}^A \mid C_{r,x}^A \geq C_{r,x+1}^A\} \\
M_r^{AL} =& \{C_{r,\hat{P}+1}^A, C_{r,\hat{P}+2}^A, \ldots, C_{r,\hat{Q}}^A \mid C_{r,y}^A \geq C_{r,y+1}^A\} \\
M_r^{RH} =& \{C_{r,1}^R, C_{r,2}^R, \ldots, C_{r,\hat{P}}^R \mid C_{r,x}^R \geq C_{r,x+1}^R\} \\
M_r^{RL} =& \{C_{r,\hat{P}+1}^R, C_{r,\hat{P}+2}^R, \ldots, C_{r,\hat{Q}}^R \mid C_{r,y}^R \geq C_{r,y+1}^R\}
\end{aligned}
$$

where $C_{r,x}^A$ (resp. $C_{r,x}^R$) is the WCET of an A-phase (resp. R-phase) of a task released on a remote core π_r in a time window of length $W_{i,l}$.

Furthermore, we introduce the terms $\vec{V}_r$, $\vec{X}_r$, $\vec{Y}_r$, and $\vec{Z}_r$ in order to simplify case 2 as follows:

$$
\begin{aligned}
\vec{V}_r =& \max(C_{r,\hat{P}+1}^A, C_{r,\hat{P}+1}^R) \\
\vec{X}_r =& C_{r,\hat{P}}^A + C_{r,\hat{P}}^R \\
\vec{Y}_r =& C_{r,\hat{P}}^A + C_{r,\hat{P}+1}^A \\
\vec{Z}_r =& C_{r,\hat{P}}^R + C_{r,\hat{P}+1}^R
\end{aligned}
$$

As M_r^{AH}, M_r^{RH}, M_r^{AL} and M_r^{RL} are ordered sets, the term $\vec{V}_r$ returns the largest memory phase among all the memory phases in M_r^{AL} and M_r^{RL} sets. The term $\vec{X}_r$ sums the smallest A-phase of M_r^{AH} and the smallest R-phase of the M_r^{RH} set. Similarly, the term $\vec{Y}_r$ sums the smallest A-phase in M_r^{AH} and the largest A-phase in M_r^{AL}. Finally, the term $\vec{Z}_r$ sums the smallest R-phase in M_r^{RH} and the largest R-phase in M_r^{RL}. Now we can compute the maximum bus blocking for case 2 using the following lemma.

Lemma 10. *If $N_{\pi_l}(W_{i,l}) < N_{\pi_r}(W_{i,l})$, then the maximum bus blocking suffered by tasks executing on the local core π_l due to tasks running on a remote core π_r, in any time interval of length $W_{i,l}$, is upper bounded by $Bus_{i,r}(W_{i,l})$, which is given by:*

$$
Bus_{i,r}(W_{i,l}) = \begin{cases} \sum_{x=1}^{\hat{P}} C_{r,x}^A + \sum_{y=1}^{\hat{P}} C_{r,y}^R + \vec{V}_r & , \text{ if } lp_{i,l} \neq \emptyset \\ \sum_{x=1}^{\hat{P}-1} C_{r,x}^A + \sum_{y=1}^{\hat{P}-1} C_{r,y}^R + \max\left(\vec{X}_r, \vec{Y}_r, \vec{Z}_r\right), & \text{otherwise.} \end{cases} \tag{18}
$$

where $C_{r,x}^A$ (resp. $C_{r,y}^R$) is the WCET of A-phase (resp. R-phase) that belongs to M_r^{AH} (resp. M_r^{RH}) set.

Proof. We prove this lemma using two possible scenarios on the basis of the priority of τ_i:

Scenario 1. Task τ_i is not the lowest priority task on the local core: It is proven in Lemma 7 that if task τ_i is not the lowest priority task of the local core π_l, all tasks that execute on core π_l during $W_{i,l}$ can suffer at most $(\sum_{\tau_h \in hep_{i,l}} \eta_h^+(W_{i,l}) \times 2) + 1$ bus blockings. As $N_{\pi_l}(W_{i,l}) < N_{\pi_r}(W_{i,l})$, we need to extract $(\sum_{\tau_h \in hep_{i,l}} \eta_h^+(W_{i,l}) \times 2) + 1$ bus blockings that can lead to the maximum bus blocking that can be caused by a remote core π_r during $W_{i,l}$.

As proven in Property 6.1, during the level-i busy window, the local always executes an A-phase after an R-phase, except the A-phase of the first job and the R-phase of the last job that executes during the level-i busy window. Consequently, by applying Property 6.2 to all the memory phases that execute on core π_l during $W_{i,l}$, except the first A-phase and last R-phase that executes on core π_l during $W_{i,l}$, the maximum bus blocking can be bounded by taking the sum of the execution time of $\hat{P}$ largest A- and R-phases released on a remote core π_r during $W_{i,l}$, i.e., using the sets M_r^{AH} and M_r^{RH}.

Furthermore, when $lp_{i,l} \neq \emptyset$, the level-i busy window starts when τ_i is released, but a lower priority task τ_j has started executing its A-phase. Consequently, we do not need to account for the bus blocking suffered by the A-phase of τ_j, i.e., the first A-phase that executes on the local core π_l during $W_{i,l}$. Finally, the maximum bus blocking that can be suffered by the last R-phase that executes on the core π_l during $W_{i,l}$ is computed using $\vec{V}_r$, where $\vec{V}_r$ returns the largest memory phase (i.e., A or R-phase) among M_r^{AL} and M_r^{RL} sets. Hence, Equation 18 upper-bounds the bus blocking when $lp_{i,l} \neq \emptyset$.

Scenario 2. Task τ_i is the lowest priority task on the local core: It is proven in Lemma 7 that if task τ_i is the lowest priority task executed on core π_l then it can suffer at most $(\sum_{\tau_h \in hep_{i,l}} \eta_h^+(W_{i,l}) \times 2)$ bus blockings. As $N_{\pi_l}(W_{i,l}) < N_{\pi_r}(W_{i,l})$, we need to extract the $(\sum_{\tau_h \in hep_{i,l}} \eta_h^+(W_{i,l}) \times 2)$ number of bus blockings that can lead to the maximum bus blocking that can be caused by a remote core π_r during $W_{i,l}$.

As τ_i is the lowest priority task, the bus blocking can also be suffered by the first job before its A-phase. Consequently, Property 6.2 can be applied to all the memory phases that execute on core π_l during $W_{i,l}$ *except* the first A-phase and last R-phase that execute on core π_l during $W_{i,l}$. Therefore, the maximum bus blocking that can be suffered by all memory phases except the first A-phase and last R-phase that execute on core π_l during $W_{i,l}$ can be bounded by taking the sum of the execution time of $\hat{P}-1$ largest A- and R-phases (from sets M_r^{AH} and M_r^{RH}) released on a remote core π_r during $W_{i,l}$, i.e., $\sum_{x=1}^{\hat{P}-1} C_{r,x}^A + \sum_{y=1}^{\hat{P}-1} C_{r,y}^R$.

The next step is to compute the maximum bus blocking that can be suffered by the first A-phase and last R-phase that execute on core π_l during $W_{i,l}$. To maximize the bus blocking that can be suffered by the first A-phase and last R-phase that execute on core π_l during $W_{i,l}$, we consider the two largest memory phases (i.e., two A-phases, two R-phases or a combination of one A and R-phases) that were not considered in the $\hat{P}-1$ largest A- and R-phases. This is achieved by taking the maximum among the values of $\vec{X}_r$, $\vec{Y}_r$, and $\vec{Z}_r$ where $\vec{X}_r$ returns the sum of the largest one A- and R-phase, $\vec{Y}_r$ (resp. $\vec{Z}_r$) returns the sum of the WCET of two largest A-phases (resp. R-phases) released on a remote core π_r during $W_{i,l}$ that were not previously considered in $\hat{P}-1$ A- and R-phases. The Lemma follows. □

Having bounded the maximum bus blocking $Bus_{i,r}(W_{i,l})$ that can be suffered by tasks executing on the local core π_l due to the tasks running on a remote core π_r during any time interval of length $W_{i,l}$, the next step is to compute the *total bus blocking* $Bus_{i,l}^{max}(W_{i,l})$ that can be suffered by the local core due to *all remote cores*. The total bus blocking $Bus_{i,l}^{max}(W_{i,l})$ that can be suffered by the local core due to all remote cores during $W_{i,l}$ can be computed using algorithm 1 by first computing $N_{\pi_l}(W_{i,l})$ (line 4) using Lemma 7, $N_{\pi_r}(W_{i,l})$ (line 5) using Lemma 8, and the maximum bus blocking $Bus_{i,r}(W_{i,l})$ caused by a remote core π_r (line 6) during $W_{i,l}$ using Lemma 9 to Lemma 10.

Finally, having bounded the total bus blocking $Bus_{i,l}^{max}(W_{i,l})$ that can be suffered by the local core π_l due to *all* remote cores in the level-i busy window, i.e., $W_{i,l}$, under both the DMAM and the FMAM, the length of the longest level-i busy window can be computed using Equation 8.

7. Schedulability Analysis

As proven in [19], to compute the WCRT of task τ_i, we need to determine the response time of each job of τ_i that executes during the level-i busy window $W_{i,l}$. Therefore, we first compute the maximum number of jobs of task τ_i that can execute within $W_{i,l}$ using the following Equation.

$$K_i = \eta_i^+(W_{i,l}) \tag{19}$$

To compute the response time of the k^{th} job of τ_i that execute on the local core π_l during $W_{i,l}$, i.e., denoted as $\tau_{i,k,l}$, we first compute the latest start 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 the start of the R-phase of $\tau_{i,k,l}$. The latest start time of the R-phase of $\tau_{i,k,l}$ on core π_l is computed using the following lemma.

Lemma 11. *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 following fixed-point iteration:*

$$s_{i,k,l}^R = C_{lp_{i,l}}^{max} + \sum_{\tau_h \in hep_{i,l} \setminus \tau_i} \eta_h^+(s_{i,k,l}^R - (C_i^A + C_i^E)) \times C_h + Bus_{i,l}^{max}(s_{i,k,l}^R) + (k-1) \times C_i + (C_i^A + C_i^E) \tag{20}$$

Proof. The proof is divided into 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 upper bound the impact of tasks running on all remote cores to 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 $C^{max}_{lp_{i,l}}$ i.e., computed using Equation 1. Knowing that $k-1$ jobs of task τ_i may have been 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, all jobs released by the higher or equal priority tasks in $hep_{i,l}$ except τ_i can cause interference on $\tau_{i,k,l}$ *until the start of its A-phase* due to the fixed-priority non-preemptive scheduling. Hence, the total interference that can be caused 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 $\eta^+_h(s^R_{i,k,l} - (C^A_i + C^E_i)) \times C_h$, where C_h is the WCET of task τ_h in isolation. Effectively, the total contribution from all tasks in $hep_{i,l}$ except τ_i to the start time of the A-phase of $\tau_{i,k,l}$ is upper bounded by $\sum_{\tau_h \in hep_{i,l} \setminus \tau_i} \eta^+_h(s^R_{i,k,l} - (C^A_i + C^E_i)) \times C_h$. Furthermore, to compute the start time of the R-phase of $\tau_{i,k,l}$, we add the WCET of the A- and E-phases of τ_i, given by $C^A_i + C^E_i$.

Step 2. It is possible that each job that executes on core π_l can suffer bus blocking due to tasks running on remote cores. Thus, the maximum bus blocking suffered by the local core until the start of the R-phase of $\tau_{i,k,l}$ due to tasks of all the remote cores is upper bounded by $Bus^{max}_{i,l}(s^R_{i,k,l})$, using Algorithm 1 by first computing the maximum bus blocking for the DMAM (computed using Lemma 1 to Lemma 6) and the FMAM (computed using Lemma 7 to Lemma 10). □

As $s^R_{i,k,l}$ appears on both sides of Equation 20, it can be solved iteratively by initializing $s^R_{i,k,l} = C^A_i + C^E_i + C^{max}_{lp_{i,l}} + \sum_{\tau_h \in hep_{i,l} \setminus \tau_i} C_h$. The start time $s^R_{i,k,l}$ is given by the smallest positive value of $s^R_{i,k,l}$ for which Equation 20 converges. The response time $R_{i,k,l}$ of $\tau_{i,k,l}$ can be computed by simply adding $s^R_{i,k,l}$ to the WCET of the R-phase of task τ_i, i.e., C^R_i, as following:

$$R_{i,k,l} = s^R_{i,k,l} + C^R_i \tag{21}$$

Finally, the WCRT of a given task τ_i can be computed by maximizing equation 21 over all jobs of τ_i that execute during the level-i busy window. Hence,

$$R^{max}_{i,l} = \max_{k \in [1, K_i]} \{R_{i,k,l}\} \tag{22}$$

where K_i is computed using Equation 19.

As the WCRT is computed using fixed-point iteration, the time complexity of the WCRT analysis is pseudo-polynomial in the sense that it enumerates all the jobs released by higher priority tasks and then computes the maximum bus blocking for the DMAM and FMAM.

Note that task τ_i is deemed schedulable if its WCRT (computed using Equation 22) is less than or equal to its relative deadline D_i, i.e., $R^{max}_{i,l} \leq D_i$. A task set is deemed schedulable only if all tasks in that task set are schedulable and the total bus utilization of the system is less than or equal to the capacity of the bus, i.e., 1, since the memory bus is saturated otherwise.

8. Experimental Evaluation

In this section, we evaluate the effectiveness of the proposed approaches. To the best of our knowledge, no work exists that focus on bounding the bus blocking for the 3-phase task model under the FCFS bus arbitration scheme. A similar work [22] exists that focus on memory centric scheduling of PREM tasks under partitioned fixed-task priority scheduling. The work in [22] considers a fixed processor priority bus arbitration policy and allows global memory preemption, i.e., the memory phases running on higher-priority processors can preempt the memory phases running on lower priority processors. This is different from the proposed work as we assume that memory phases execute non-preemptively. To compare the performance of our proposed Dedicated Memory Access Model (DMAM) and Fair Memory Access Model (FMAM) with the work in [22], we consider two variations of the analysis presented in [22], i.e., with/without allowing global memory preemption. The analysis that allows global memory preemption is the exact analysis presented

in [22]. The analysis without global memory preemption is a slightly modified version of [22] to allow the execution of non-preemptive memory phases.

To evaluate the performance of all the analyzed approaches, we perform two sets of experiments. A case study experiment performed using task parameters obtained from the Mälardalen benchmark suite [18] is presented in Section 8.1. Experiments performed using synthetic tasksets are detailed in Section 8.2.

8.1. Case Study

For the case study experiments, we use task parameters taken from Table 2 of [15]. Table 2 in [15] is generated from the Mälardalen benchmark suite using the gem5 instruction set simulator by modeling a quad-core multicore platform considering ARMv7 cores and a shared memory bus that connects the cores to the main memory.

Although Table 2 of [15] contains several task parameters for the analyzed benchmarks, we only consider the Processing Demand (PD) and Memory Demand (MD) of tasks in our experiments. Also, since we consider non-preemptive task scheduling which can suffer from the long task problem, i.e., task sets that contain some tasks with short deadlines and others with long WCETs are trivially unschedulable due to blocking from lower priority tasks. This problem has been identified in the state-of-the-art, e.g., see Section 6 of [23]. Therefore, to circumvent this problem, we only selected benchmarks from Table 2 of [15] such that the total WCET (i.e., $PD + MD$) of each task remains in the range of 2000 to 12000. Tasks' parameters considered for the case study experiments are given in Table 2. Instead of using the terms PD and MD (as in [15]), we use the WCET of the phases and total WCET of tasks in Table 2. As shown in Table 2, the value of C_i^E is considered equal to the task's processor demand, and the value of $C_i^A + C_i^R$ is considered equal to the task's memory demand, and the total WCET of task is given by $C_i = C_i^A + C_i^E + C_i^R$.

Name	C_i^E	$C_i^A + C_i^R$	C_i
cnt	7765	573	8338
compressdata	3166	494	3660
compress	8793	993	9786
cover	3661	696	4357
duff	3121	553	3674
expint	8058	716	8774
fdct	5923	1088	7011
fir	6938	1207	8145
insertsort	2218	415	2633
jfdctint	7771	1086	8857
ludcmp	8278	768	9046
nsichneu	8648	1582	10230
petrinet	2272	438	2710
qurt	8663	735	9398
recursion	5564	907	6471
select	7211	986	8197

Table 2: Benchmark parameters used in the experiments.

By default, we consider a multicore platform with 4 cores and a task set size of 32 tasks with 8 tasks per core. For task-to-core mapping, we randomly map tasks to core while ensuring that each core has the same number of tasks and the core utilization for each core is same. To assign the benchmark parameters to tasks mapped on the cores, we randomly select a benchmark from Table 2 and assign its C_i^A, C_i^E, C_i^R, and C_i, values to a task. We then randomly generate tasks' utilizations U_i using UUnifast discard [24] algorithm. Having assigned the values of C_i and U_i, we generate the task period by using the equation $T_i = C_i/U_i$. The task priorities are then assigned using rate monotonic [25] and tasks deadlines are equal to their periods.

In the case study, we performed two experiments by varying: 1) the core utilization (i.e., utilization of each core); 2) the number of cores in the system, and compared the performance of all the analyzed

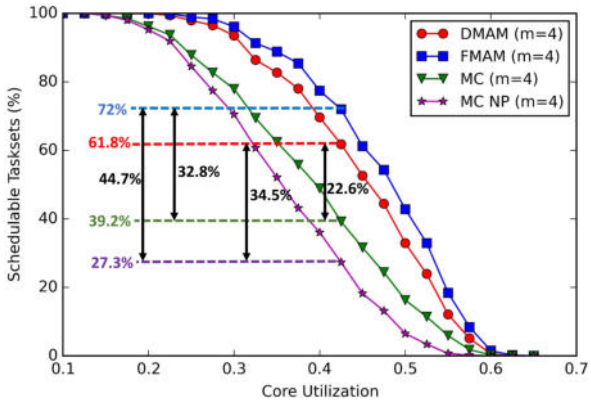


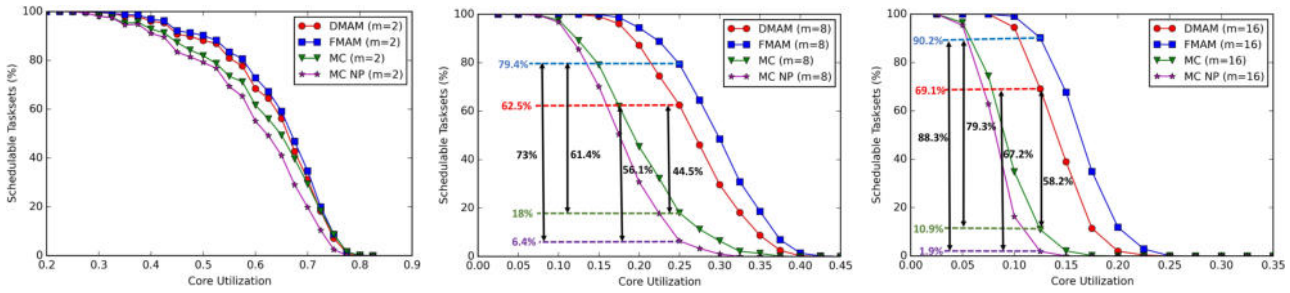
Figure 13: Varying Core Utilization

approaches in terms of task set schedulability. In the results for the case study (and also for the experiments in Section 8.2), the analysis for the dedicated memory access model is marked as "DMAM" whereas the analysis for the fair memory access model is marked as "FMAM". Similarly, the memory centric scheduling approach of [22] is marked as "MC" and the memory centric scheduling approach of [22] without global memory preemption is marked as "MC-NP". In each experiment, 1000 task sets were generated per point.

1. Core Utilization: In this experiment, we vary the core utilization of each core in the range of 0.025 to 1 in steps of 0.025. As shown in Figure 13, the schedulability using all the approaches decreases with the increase in core utilization. This is intuitive as increasing core utilization increases tasks utilizations, which directly impacts the task period/deadline. We observe that none of the approaches were able to schedule tasksets with core utilization higher than 0.60. This is mainly due to higher number of tasks in the task set under the default configuration, i.e., for 4 cores we have 32 tasks in the taskset. Effectively, this results in increasing bus blocking between tasks leading to reduced schedulability. However, we can see in Figure 13, that the FMAM and DMAM analyses outperform the MC and MC-NP analyses. For instance, at the core utilization value of 0.425, FMAM analysis was able to schedule 32.8% more tasksets as compared to MC analysis and 44.7% more taksets as compared to MC-NP analysis. This is mainly due to two reasons. The first reason is that unlike [22], the proposed analysis provides a fine-grained bus blocking analysis using different cases, that account for different scheduling scenarios that can be observed on the core under analysis as well as remote cores. This results in tighten the bound on bus blocking suffered by the tasks. The second reason is that the proposed work shares the bus among all cores in a more fair manner (e.g. FMAM), whereas the analysis of [22] assigns the bus to the higher priority cores. In such a case, there can be a scenario in which even the highest priority task running on the lowest priority core may suffer bus blocking from all the tasks released on the higher priority cores. On the contrary, the proposed analysis bounds the bus blocking on the basis of number of jobs/memory phases that can suffer/cause bus blocking during the response time of the task under analysis.

We also observe that the FMAM performs the best among all the analyses, whereas MC-NP performs the worst. This is because FMAM distributes the bus among all the cores in a fair manner and due to the fine-grained bus blocking analysis for FMAM. The MC-NP performs worse than MC as tasks can additionally suffer the bus blocking from lower priority cores due to the non-preemptive execution of memory phases under MC-NP.

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 varied from 2 to 16 along with core utilization that was varied from 0.025 to 1 in steps of 0.025. The percentage of task sets that were deemed schedulable by all approaches for different values of m is shown in Figure 14. We can see that by increasing the number of cores, the number of tasksets that were deemed schedulable by all the approaches decreases. This is mainly due to the fact that increasing the number of cores also increases the number of remote cores and the number of tasks in the taskset, which results in increasing the bus blocking that can be suffered by



(a) Varying Core Utilization for $m = 2$ (b) Varying Core Utilization for $m = 8$ (c) Varying Core Utilization for $m = 16$

Figure 14: Varying the Number of Cores and Core Utilization

the task under analysis from the remote cores.

We observe that the performance gain of FMAM and DMAM analysis against MC and MC-NP increased with an increase in the number of cores, i.e., $m = 8, 16$. For instance, at the core utilization value of 0.125, the FMAM analysis was able to schedule 79.3% more tasksets as compared to the MC analysis and 88.3% more taksets compared to the MC-NP analysis for $m = 16$ as shown in Figure 14c.

On the other hand, we observe that the performance gain of the FMAM and DMAM analysis against MC and MC-NP is reduced by decreasing the number of cores to $m = 2$. In fact, the MC analysis performed almost the same as FMAM and DMAM for some core utilization values. We explain these variations in the gain as follows:

An increase in the number of cores results in an increase in the number of remote cores. However, the bus blocking suffered by the tasks using the proposed analysis depends on several cases/sub-cases. This implies that even when the number of cores is increased, the bus blocking suffered by task under analysis may not increase significantly as there may be a few tasks from remote cores that participate in the bus blocking.

On the other hand, an increase in the number of cores results in increasing the number of higher priority cores, which can cause bus blocking to the lowest priority core. Consequently, the MC and MC-NP analyses can be significantly impacted as even the highest priority task (i.e., task that has the smallest period/deadline) running on the lowest priority core can suffer bus blocking from all the tasks released on all the higher priority cores.

Interestingly, we also observe that for higher values of m, the difference between FMAM and DMAM was significant. For instance, FMAM was able to schedule up to 67.7% tasksets whereas DMAM was able to schedule only 38.9% at 0.15 core utilization for $m = 16$ as shown in Figure 14c.

8.2. Experiments using Synthetic Tasks

In this section, we will explain the experiments that were performed using synthetic task sets to compare the performance of DMAM, FMAM, MC and MC-NP approaches. 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 algorithm [24]. Task periods were generated using log-uniform distribution in the range of [100,1000]. In each experiment, 1000 task sets were generated per point.

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 in the range of [10%, 50%]$\times C_i$, i.e., $MD = rand(10\%, 50\%) \times C_i$. The values of C_i^A, C_i^E and C_i^R are then chosen such that $C_i^A = C_i^R = MD/2$ [4] and $C_i^E = C_i - (C_i^A + C_i^R)$. Task deadlines were implicit with priorities assigned using Rate-Monotonic [25].

We performed several experiments by varying: 1) the core utilization; 2) the number of cores; 3) the task memory demands; and 4) the task periods.

1. Core Utilization: In this experiment, we varied each core utilization between 0.025 and 1 in steps of 0.025 and plotted the number of task sets that were deemed schedulable by all the analyzed approaches,

[4] Note that for the analysis in [22] we consider a single memory phase of length MD.

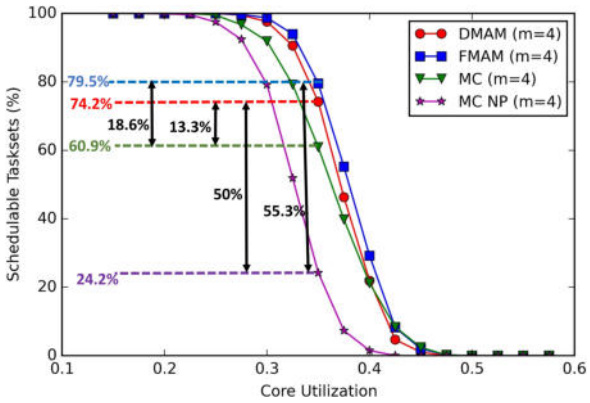
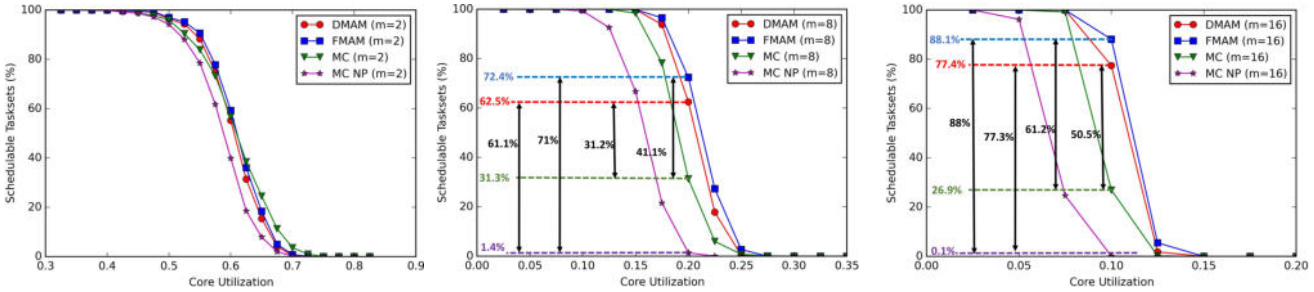


Figure 15: Varying Core Utilization



(a) Varying Core Utilization for $m = 2$ (b) Varying Core Utilization for $m = 8$ (c) Varying Core Utilization for $m = 16$

Figure 16: Varying the Number of Cores and Core Utilization

i.e., DMAM, FMAM, MC, and MC-NP. The percentage of task sets that were deemed schedulable using all the approaches for each core utilization value are shown in Figure 15. As shown in 15, the schedulability of all the approaches decreases with an increase in the core utilization. This is intuitive as increasing the core utilization can increase the values of C_i, C_i^A, C_i^E and C_i^R that can eventually increase the interference/blocking from the same core and bus blocking from other cores. We note that the overall task set schedulability for all the approaches is quite low as no tasksets were schedulable at 0.50 core utilization. This is intuitive as the MD value of tasks can be up to 50% of their WCET, which can directly contribute to the bus blocking that can be suffered/caused by tasks. We observe that the FMAM analysis performed slightly better than DMAM, as expected. As shown in 15, the MC analysis performs better than MC-NP. This is intuitive because tasks can additionally suffer the bus blocking from lower priority cores due to the non-preemptive execution of memory phases under MC-NP.

We can also see in Figure 15 that the proposed analysis for DMAM and FMAM outperforms the memory centric scheduling analysis of [22]. In particular, at the core utilization value of 0.35, FMAM can schedule up to 55.3% more tasksets as compared to MC-NP and up to 18.6% more tasksets as compared to MC. Similarly, at the core utilization value of 0.35, DMAM can schedule up to 50% more tasksets as compared to MC-NP and up to 13.3% more tasksets as compared to MC. As discussed earlier, the improved performance of DMAM and FMAM over MC and MC-NP is mainly due to a more fine-grained bus blocking analysis used by DMAM and FMAM.

Interestingly, we observe that no taskset is schedulable after the core utilization value of 0.475 using any of the approaches as shown in Figure 15. On the contrary, almost all the approaches were able to schedule tasksets up to 60% core utilization under the case study, i.e., see Figure 13. This is because the value of MD is quite small in benchmark parameters given in Table 2 whereas the value of MD can go up to $50\% \times C_i$ while randomly generating the tasks.

2. Number of Cores: In this experiment, we vary the number of cores along with the core utilization, keeping default values for all other parameters. The number of cores (m) was varied from 2 to 16, and for

each value of (m), the core utilizations varied from 0.025 to 1 in steps of 0.025. The percentage of task sets that were deemed schedulable for different values of m by all the approaches are shown in Figure 16. We can see in Figure 16 that by increasing the number of cores, the number of task sets that were deemed schedulable by all the approaches decreases. This is mainly due to the fact that by increasing the number of cores, the number of tasks in the taskset also increases, which results in increasing the bus blocking that can be suffered by the task under analysis from the remote cores/higher priority cores. For example, for two cores all task sets were deemed schedulable by all the approaches at the core utilization of 0.35 but no task set was schedulable at the same core utilization when the value of m is increased to 8 or 16.

Figure 16b and 16c show that the FMAM, and DMAM analysis can outperform MC and MC-NP analysis when the value of m is increased to 8 and 16. For instance, at the core utilization value of 0.20, the FMAM analysis can schedule up to 41.1% more tasksets as compared to MC and 71% more tasksets as compared to MC-NP for $m = 8$ (see Figure 16b). Similarly, at the core utilization value of 0.20, the DMAM analysis can schedule up to 31.2% more tasksets as compared to MC and 61.1% more tasksets as compared to MC-NP for $m = 8$ (see Figure 16b). This performance gains were further increased for $m = 16$ as shown in Figure 16c. However, all the approaches perform almost similarly for $m = 2$. In fact, MC analysis was able to perform better than FMAM and DMAM analysis for some of the core utilization values for $m = 2$ as shown in Figure 16a. We explain these performance gains as follows.

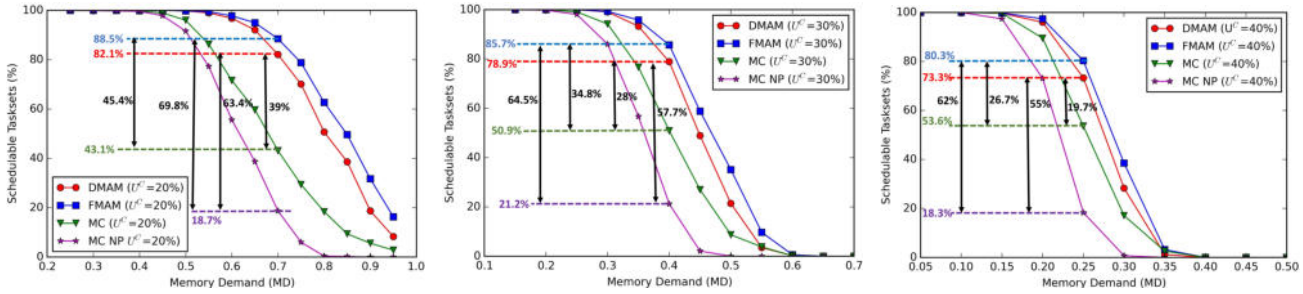
As discussed earlier, MC analysis gets significantly impacted by increasing/decreasing the number of cores as the analysis is based on processor priority whereas the FMAM/DMAM analysis is based on FCFS bus arbitration in which the bus blocking depends on several cases and subcases. In particular, for $m = 2$, under the MC analysis tasks running on only one core (i.e., all except the highest priority core) can suffer bus blocking whereas under the DMAM and FMAM analyses, the tasks running on both the cores can suffer bus blocking, i.e., 2x more than MC analysis, due to the FCFS bus arbitration. On the contrary, for the $m = 16$, under the MC analysis tasks running on 15 cores can suffer bus blocking whereas under the DMAM and FMAM analyses, the tasks running on 16 cores can suffer bus blocking, i.e., 1.066x more than MC analysis, due to the FCFS bus arbitration. Therefore, the FMAM and DMAM analyses performed significantly better than MC analysis at higher values of m, i.e., $m = 8$, 16, but performed slightly worse than MC at some core utilization values for the lower value of m, i.e., $m = 2$.

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 $C_i^A + C_i^R = MD$ 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 all the approaches are plotted in Figure 17. We choose different sets of core utilizations (denoted by U^C), i.e., 20%, 30%, and 40% to show the impact of MD on task set schedulability.

We can see in Figure 17 that for the values of core utilization of 20%, 30%, and 40%, the percentage of tasksets that were deemed scheduled using all the approaches decreases with the increase in MD. This is intuitive, as for higher values of MD, the values of C_i^A, and C_i^R also increase which may result in increasing bus blocking. Furthermore, we observe that for lower values of core utilization the number of task sets that were deemed schedulable by all approaches was much higher even for larger values of MD. For example, at a core utilization of 20% (i.e., U^C=20%), tasks with very high memory demand, i.e., up to 80% of their WCET, were still schedulable as shown in Figure 17a. However, the taskset schedulability decreases rapidly for higher values of core utilization as shown in Figure 17b, and 17c. Finally, we can also observe that the FMAM and DMAM analysis outperforms the MC and MC-NP analysis. For instance, the FMAM analysis was able to schedule up to 45.4% more tasksets as compared to MC and up to 69.8% more tasksets as compared to MC-NP for MD value of 70% at 20% core utilization, as shown in Figure 17a.

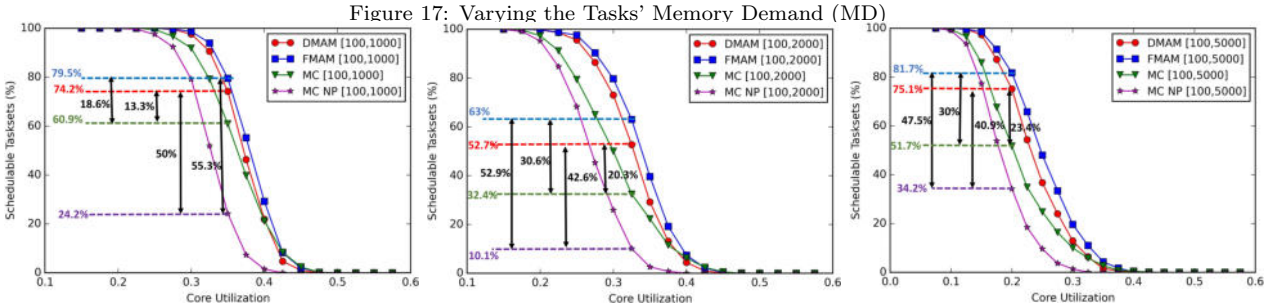
4. Task Periods: In this experiment, we varied the period range of tasks' and analyzed its impact on schedulability. As we generate the WCET C_i of tasks using the task periods T_i, i.e., $C_i = U_i \times T_i$, which is then used to generate C_i^A, C_i^E and C_i^R, therefore, the value of task periods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

In this experiment, the core utilization was varied for three different period ranges, i.e., [100,1000], [100,2000], [100,5000] and the percentage of task sets that were deemed schedulable using all the approaches is shown in Figure 18. We observe that an increase in the period range has a negative impact on task set



(a) Varying MD for 20% Core Utilization (b) Varying MD for 30% Core Utilization (c) Varying MD for 40% Core Utilization

Figure 17: Varying the Tasks' Memory Demand (MD)



(a) Tasks' Period in range of 100 to 1000 (b) Tasks' Period in range of 100 to 2000 (c) Tasks' Period in range of 100 to 5000

Figure 18: Varying the Tasks' Period Range and Core Utilization

schedulability. We explain these variations as follows:

Increasing the task period increases the WCET of tasks due to the relation between C_i and T_i, i.e., $C_i = U_i \times T_i$. This in turn increases the blocking from one job of a lower priority task, i.e., a larger period leads to a larger WCET which causes a larger blocking from lower priority tasks. This implies that increasing the task period range increases the blocking caused by a lower priority task. This increase in lower priority blocking also increases the length of the level-i busy window which may result in converging the level-i busy window at a later stage due to additional jobs released by higher priority tasks. This causes a degradation in task set schedulability when the period ranges are increased. However, we can still see that even for higher values of task periods the proposed FMAM and DMAM analyses dominate the MC and MC-NP analyses.

9. Related Work

The problem of timing unpredictability due to the shared memory bus in multicore systems is not new [26] and many existing works already attempted to solve this problem [27]. Some approaches [7, 8, 9] are based on Time Division Multiple Access (TDMA) in which time slots are divided among cores and a core can only access the memory bus in its defined time slot. Dasari et al. [12] 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 [14]. Rashid et al. [16] proposed the cache persistence-aware memory bus contention analysis for multicore systems. Even though these approaches are proposed to bound the bus contention for partitioned fixed-priority scheduling, they are proposed for generic task models and are not tailored for phased execution models.

On the other hand, approaches like [28, 29, 30, 31, 17, 32, 15, 33, 34, 35, 22] focus on phased execution models. Maia et al. [17] focus on the bus contention analysis for the fixed priority 3-phase task model under global scheduling. Arora et al. [32] proposed the bus contention analysis for the 3-phase task model considering partitioned scheduling and round-robin bus arbitration. Since the bus contention analysis of [32] was formulated for round-robin bus arbitration, it is based on the bus slot size and number of bus requests performed during those slots. Davis et al. [15] proposed an extensible framework for multicore timing

analysis. The authors compute the WCRT of tasks scheduled using partitioned fixed-priority preemptive 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 [30] 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, [29, 17] focus on global scheduling whereas [15] presents a response time analysis by considering the inter-core interference due to various shared resources of multicore systems. [33, 34] 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 Section8, 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, i.e., main memory, and it assumes that the main memory can handle only one request at a time. However, as the experimental evaluation has shown, our proposed approaches outperform the analysis in [22] by providing a much fine-grained bus blocking analysis.

10. Conclusion

In this work, we present a fine-grained analysis to compute the maximum bus blocking suffered by 3-phase tasks scheduled using partitioned fixed-priority non-preemptive scheduling. We show that the bus blocking suffered by the tasks executing on a multicore platform depends on the underlying memory access model. As a consequence, we present the bus blocking analysis for two memory access models referred to as the dedicated memory access model and the fair memory access model. For each model, the maximum bus blocking is derived using different cases and sub-cases to emulate different scheduling scenarios that can happen when concurrent tasks execute on a multicore platform and try to access the bus. This allows us to achieve tighter bounds on the maximum bus blocking that can be suffered by the tasks as well as improves taskset schedulability. We also show how the maximum bus blocking of tasks can be integrated into the WCRT analysis of tasks. Experimental evaluation shows that the proposed analysis can improve the number of task sets that are deemed schedulable by up to 88 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 their performance against the proposed approach. We would also like to investigate the impact of various task-to-core mapping strategies on the proposed analysis in a future work.

Acknowledgement

This work was partially supported by 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 National Funds through FCT/MCTES (Portuguese Foundation for Science and Technology), within the CISTER Research Unit (UIDP/UIDB/04234/2020); by FCT and the Portuguese National Innovation Agency (ANI), under the CMU Portugal partnership, through the European Regional Development Fund (ERDF) of the Operational Competitiveness Programme and Internationalization (COMPETE 2020), under the PT2020 Partnership Agreement, within project FLOYD (POCI-01-0247-FEDER-045912), also by FCT under PhD grant 2020.09532.BD.

References

[1] 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

[2] 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/

[3] 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

[4] 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/

[5] C. Pagetti, J. Forget, H. Falk, D. Oehlert, A. Luppold, Automated generation of time-predictable executables on multicore, in: Proceedings of the 26th International Conference on Real-Time Networks and Systems, RTNS '18, Association for Computing Machinery, New York, NY, USA, 2018, p. 104–113. doi:10.1145/3273905.3273907.
URL https://doi.org/10.1145/3273905.3273907

[6] M. Becker, D. Dasari, B. Nicolic, B. Akesson, V. Nélis, T. Nolte, Contention-free execution of automotive applications on a clustered many-core platform, in: ECRTS 2016, 2016, pp. 14–24. doi:10.1109/ECRTS.2016.14.

[7] 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.

[8] 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

[9] 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.

[10] 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.

[11] 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

[12] 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.

[13] 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.

[14] 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.

[15] 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).

[16] S. A. Rashid, G. Nelissen, E. Tovar, Cache persistence-aware memory bus contention analysis for multicore systems, in: DATE, 2020, pp. 442–447. doi:10.23919/DATE48585.2020.9116265.

[17] 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/

[18] J. Gustafsson, A. Betts, A. Ermedahl, B. Lisper, The Mälardalen WCET Benchmarks: Past, Present And Future, in: B. Lisper (Ed.), 10th International Workshop on Worst-Case Execution Time Analysis (WCET 2010), Vol. 15 of OpenAccess Series in Informatics (OASIcs), Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2010, pp. 136–146, the printed version of the WCET'10 proceedings are published by OCG (www.ocg.at) - ISBN 978-3-85403-268-7. doi:10.4230/OASIcs.WCET.2010.136.
URL http://drops.dagstuhl.de/opus/volltexte/2010/2833

[19] 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.

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

[21] K. W. Tindell, A. Burns, A. J. Wellings, An extendible approach for analyzing fixed priority hard real-time tasks, Real-Time Syst. 6 (2) (1994) 133–151. doi:10.1007/BF01088593.
URL https://doi.org/10.1007/BF01088593

[22] G. Schwäricke, T. Kloda, G. Gracioli, M. Bertogna, M. Caccamo, Fixed-Priority Memory-Centric Scheduler for COTS-Based Multiprocessors, in: M. Völp (Ed.), 32nd Euromicro Conference on Real-Time Systems (ECRTS 2020), Vol. 165 of

Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2020, pp. 1:1–1:24. doi:10.4230/LIPIcs.ECRTS.2020.1.
URL https://drops.dagstuhl.de/opus/volltexte/2020/12364

[23] R. I. Davis, S. Altmeyer, J. Reineke, Analysis of write-back caches under fixed-priority preemptive and non-preemptive scheduling, in: Proceedings of the 24th International Conference on Real-Time Networks and Systems, RTNS '16, Association for Computing Machinery, New York, NY, USA, 2016, p. 309–318. doi:10.1145/2997465.2997476.
URL https://doi.org/10.1145/2997465.2997476

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

[25] 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

[26] D. Dasari, B. Akesson, V. Nelis, M. A. Awan, S. M. Petters, Identifying the sources of unpredictability in COTS-based multicore systems, in: 2013 8th IEEE International Symposium on Industrial Embedded Systems (SIES), IEEE, Porto, 2013, pp. 39–48. doi:10.1109/SIES.2013.6601469.
URL http://ieeexplore.ieee.org/document/6601469/

[27] C. Maiza, H. Rihani, J. M. Rivas, J. Goossens, S. Altmeyer, R. I. Davis, A Survey of Timing Verification Techniques for Multi-Core Real-Time Systems, ACM Computing Surveys 52 (3) (2019) 1–38. doi:10.1145/3323212.
URL http://dl.acm.org/citation.cfm?doid=3341324.3323212

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

[29] A. Alhammad, R. Pellizzoni, Schedulability analysis of global memory-predictable scheduling, in: Proceedings of the 14th International Conference on Embedded Software - EMSOFT '14, ACM Press, New Delhi, India, 2014, pp. 1–10. doi:10.1145/2656045.2656070.
URL http://dl.acm.org/citation.cfm?doid=2656045.2656070

[30] G. Yao, R. Pellizzoni, S. Bak, E. Betti, M. Caccamo, Memory-centric scheduling for multicore hard real-time systems, Real-Time Systems 48 (6) (2012) 681–715. doi:10.1007/s11241-012-9158-9.
URL http://link.springer.com/10.1007/s11241-012-9158-9

[31] J. M. Rivas, J. Goossens, X. Poczekajlo, A. Paolillo, Implementation of Memory Centric Scheduling for COTS Multi-Core Real-Time Systems, in: S. Quinton (Ed.), 31st Euromicro Conference on Real-Time Systems (ECRTS 2019), Vol. 133 of Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2019, pp. 7:1–7:23. doi:10.4230/LIPIcs.ECRTS.2019.7.
URL http://drops.dagstuhl.de/opus/volltexte/2019/10744

[32] J. Arora, C. Maia, S. A. Rashid, G. Nelissen, E. Tovar, Bus-contention aware wcrt analysis for the 3-phase task model considering a work-conserving bus arbitration scheme, Journal of Systems Architecture 122 (2022) 102345. doi:https://doi.org/10.1016/j.sysarc.2021.102345.
URL https://www.sciencedirect.com/science/article/pii/S138376212100237X

[33] 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.

[34] 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.

[35] 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.

Jatin Arora is a real-time embedded systems researcher at CISTER (Research Centre in Real-Time and Embedded Computing Systems), a research centre co-hosted by the Faculty of Engineering of the University of Porto (FEUP) and the School of Engineering (ISEP) of the Polytechnic Institute of Porto, Portugal. He is also a PhD candidate in Electrical and Computer Engineering at the Faculty of Engineering of the University of Porto (FEUP). Jatin has been awarded the PhD Research Grant from Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) in September 2020. Before joining CISTER, he received his Master's degree (M.Tech) in Embedded Systems with distinction from JNTU, Hyderabad, India. His research interests include real-time embedded systems, scheduling theory, timing predictability in multicore architectures, and memory resource contention.

Cláudio Maia has a Bachelor's Degree (B.Sc.) and Master's degree (M.Sc.) in Computer Engineering from the School of Engineering of the Polytechnic Institute of Porto. In 2018, he received his Ph.D. degree in Electrical and Computer Engineering from the University of Porto under the supervision of professors Luís Miguel Nogueira and Luís Miguel Pinho, with a thesis titled "Scheduling Parallel Real-Time Tasks in Multiprocessor Platforms". He is a researcher whose research interests include the design and implementation of operating systems and hypervisors, multiprocessor architectures and real-time scheduling theory.

Syed Aftab Rashid is an embedded system research scientist at VORTEX-CoLab and also serves as an integrated Ph.D. researcher at CISTER Research Unit, an internationally renowned research centre focusing on real-time and embedded computing systems. His dissertation was on the timing analysis of multicore platforms for hard real-time systems with a focus on contention due to sharing of cache memories and interconnects. Before joining CISTER, he received his M.Sc. in Electrical Engineering from the National University of Computer and Emerging Sciences (NUCES)-FAST, Islamabad, Pakistan in 2014. He has worked on several international projects related to embedded system design and implementation. He has also co-authored several publications in reputed conferences (e.g., RTSS, RTAS, ECRTS, DATE) and journals. His research interests include real-time embedded systems, timing and scheduling analysis, resource contention, and multicore architectures' predictability.

Geoffrey Nelissen is an Assistant Professor in the IRIS group of the Mathematics and Computer Science department of the Eindhoven University of Technology (TU/e), the Netherlands. Prior to joining TU/e, Geoffrey was a researcher at the Polytechnic Institute of Porto where he was a member of the CISTER research unit. He received his Ph.D. from the Université Libre de Bruxelles (ULB), Belgium in 2013. His research interests span all theoretical and practical aspects of real-time embedded systems design, including the analysis and configuration of real-time parallel applications on multicore and distributed platforms.

Eduardo Tovar received the Licentiate, M.Sc., and Ph.D. degrees in electrical and computer engineering from the University of Porto, Porto, Portugal, in 1990, 1995, and 1999, respectively. He is currently a Professor with the Department of Computer Engineering, School of Engineering (ISEP), Polytechnic Institute of Porto (P.Porto), where he is also engaged in research on real-time distributed systems, wireless sensor networks, multiprocessor systems, cyber-physical systems, and industrial communication systems. He heads the CISTER Laboratory, an internationally renowned research center focusing on RTD in real-time and embedded computing systems. He is currently the Vice-Chair of ACM SIGBED and is a member of the Executive Committee of the IEEE Technical Committee on Real-Time Systems (TC-RTS). He is currently deeply involved in the core team setting-up a Collaborative (industry-academic) Laboratory on Cyber-Physical Systems and Cyber-Security Systems.