Work-In-Progress: WCRT Analysis for the 3-Phase Task Model in Partitioned Scheduling

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

*CISTER Research Centre

CISTER-TR-201005

2020/12/01

Work-In-Progress: WCRT Analysis for the 3-Phase Task Model in Partitioned Scheduling

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

*CISTER Research Centre

Polytechnic Institute of Porto (ISEP P.Porto)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

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

https://www.cister-labs.pt

Abstract

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

Work-In-Progress: WCRT Analysis for the 3-Phase Task Model in Partitioned Scheduling

Jatin Arora*, Cláudio Maia*, Syed Aftab Rashid*, Geoffrey Nelissen†, Eduardo Tovar*
*CISTER, ISEP, Polytechnic Institute of Porto, Portugal †Eindhoven University of Technology (TU/e), Eindhoven, the Netherlands

***Abstract*—Multicore platforms are being increasingly adopted in Cyber-Physical Systems (CPS) due to their advantages over single-core processors, such as raw computing power and energy efficiency. Typically, multicore platforms use a shared system bus that connects the cores to the memory hierarchy (including caches and main memory). However, such hierarchy causes tasks running on different cores to compete for access to the shared system bus whenever data reads or writes need to be made. Such competition is problematic as it may cause large variations in the execution time of tasks in a non-deterministic way. This paper presents an analysis that allows one to derive bus contention-aware worst-case response-time of tasks that follow the 3-phase task model executing under partitioned scheduling.**

I. Introduction

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

Solutions that use *phased* execution models [2]–[4] are promising candidates to circumvent the problem of inter-core interference due to shared resources. In these models, tasks' executions are divided into separate memory and execution phases. The memory phase is responsible for loading tasks' data and instructions into a core's local memory (e.g., cache or scratchpad) and to push back the processed data into the main memory. During the execution phase, the core executes the task's code by processing data/instructions already available in the core's local memory without any need to access the system bus or the main memory. Still, even under these models, tasks may contend to access the shared bus. This situation happens when a task tries to access the bus to load its data/instructions from the main memory and the bus is already busy serving the memory phase of another task executing on another core. Such situation forces the requesting task to hold its execution until the bus is free. This phenomenon is referred to as *bus blocking* in this work. Since bus blocking can significantly impact task schedulability, even under phased execution models, works like [3] have been proposed to bound the bus blocking under *global* scheduling. Contrary to [3], in this work we focus on analyzing the bus contention and deriving the worst-case response time (WCRT) for the 3-phase task model assuming fixed-priority *partitioned* scheduling. We derive the maximum bus blocking that may be experienced by each task and then use this value to determine their worst-case response-time.

This work was partially supported by National Funds through FCT/MCTES (Portuguese Foundation for Science and Technology), within the CISTER Research Unit (UIDB/04234/2020); also by the Operational Competitiveness Programme and Internationalization (COMPETE 2020) under the PT2020 Partnership Agreement, through the European Regional Development Fund (ERDF), and by national funds through the FCT, within project POCI-01-0145-FEDER-029119 (PREFECT)

II. System Model

We consider a multicore platform comprising m identical cores $(\pi_1, \pi_2, ..., \pi_m)$ where each core has a local memory (i.e., cache or scratchpad) that can store the task's data/instructions during runtime. Each core uses a shared system bus to access the main memory and cores can access the bus concurrently which, as explained previously, may lead to contention. Furthermore, we assume that the system bus can handle only one access at a time and the system bus arbitration policy assumed is First-Come First-Served (FCFS).

Task Model: We consider a task set Γ comprising n sporadic tasks. Each task τ_i is executed following a 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) phase. The worst-case execution time (WCET) of each phase of τ_i is denoted by C_i^A, C_i^E, and C_i^R, respectively. Thus, the WCET of 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 response time of the k^{th} job of task τ_i executing on a given core π_l is denoted by $R_{i,k,l}$. Consequently, the worst-case response time (WCRT) of task τ_i, denoted by $R_{i,l}^{max}$, is given by maximizing $R_{i,k,l}$ over all jobs of τ_i. We assume partitioned scheduling where task to core mapping is given at design time and any fixed task-priority algorithm is used to assign task priorities. Additionally, we define the following set of tasks: $hep_{i,l}$ denotes the set of tasks with higher or equal priority than τ_i on core π_l; $hp_{i,l}$ (resp. $lp_{i,l}$) denotes the set of tasks with priority higher (resp. lower) than τ_i on core π_l.

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

Each core maintains its own *ready queue* sorted by priority with tasks that are ready to execute. Whenever a task in the queue becomes ready to execute, the core requests access to the system bus and if the system bus is free, the core executes the A-phase of that task. However, if the system bus is busy serving a memory phase from any other core, then the core will busy-wait until the bus becomes available, at which point it will execute the A-phase of the task with highest priority in the ready queue. Once the A-phase of a task completes, the E-phase of the same task starts executing immediately on the core. After the E-phase completes, the task requests access to the bus to execute its R-phase. At this point, the core may have to busy-wait for the bus if the bus is busy serving requests of co-running tasks. Once the bus becomes available, the task can execute its R-phase and finalize its execution. In addition, if there are other tasks waiting in the core's ready-queue, we assume that the A-phase of another ready task can execute immediately after an R-phase that just completed its execution on the same core in order to avoid any bus blocking during this transition of phases/tasks.

When more than one core requests access to the system bus simultaneously, it is assumed that, in the worst-case, the core under analysis accesses the bus after the completion of the bus requests of all the other cores (i.e., the request of the core under analysis is the last to arrive on a FCFS basis).

III. Busy Window Computation

According to our system model, a task that started executing its A-phase cannot be preempted until completion of its R-phase. It thus behaves similarly to a non-preemptive system. For single-core platforms that use fixed-priority non-preemptive (FPNP) scheduling, the worst-case response time of a task τ_i is observed in the longest level-i busy window [1].

Definition III.1. Level-i busy window: A level-i busy window is a time interval (a, b) in which the pending workload of tasks with priorities higher or equal to that of task τ_i is positive for all $t \in (a, b)$ and 0 at the boundaries a and b.

To compute the longest level-i busy window w.r.t a task τ_i, we must compute the maximum *interference* and maximum *blocking* τ_i can suffer during its execution. It was proven in [1], [6] that on single-core platforms, a task τ_i can suffer blocking from at most one lower-priority job, and suffers interference from all higher priority jobs that execute before τ_i. However, when considering a multicore platform, task τ_i may additionally suffer bus blocking due to co-running tasks that are executing on other cores than τ_i. These co-running tasks can cause additional delays in the execution of τ_i by blocking its accesses to the system bus leading to an increase in the length of level-i busy window.

Let $W_{i,l}$ be the length of the level-i busy window w.r.t a task τ_i executing on core π_l, where $W_{i,l}$ can be computed using the following iterative equation:

$$W_{i,l} = C_{lp,i,l}^{max} + Bus_{i,l}^{max}(W_{i,l}) + \sum_{\tau_h \in hep_{i,l}} (\eta_h^+(W_{i,l}) \times C_h) \tag{1}$$

$C_{lp,i,l}^{max}$ is the maximum blocking that can be caused by one job of a lower-priority task on core π_l, i.e., $C_{lp,i,l}^{max} = \max_{\tau_j \in lp_{i,l}} \{C_j\}$, and $Bus_{i,l}^{max}(W_{i,l})$ is the maximum bus blocking suffered by τ_i in a time window of length $W_{i,l}$ (this value is bounded in next subsections).

To bound the maximum number of jobs of any task τ_h that may interfere with the execution of task τ_i, we use the concept of upper event arrival function [5]. The upper event arrival function $\eta_h^+(W_{i,l})$ returns the maximum number of jobs released by task τ_h in any time interval of length $W_{i,l}$. The upper event arrival function can efficiently capture the variability in the load and can represent complex task activation patterns such as periodic with jitter, burst, etc. Considering that C_h is the WCET of task τ_h in isolation, i.e., $C_h = C_h^A + C_h^E + C_h^R$, $(\eta_h^+(W_{i,l}) \times C_h)$ upper-bounds the maximum interference the higher or equal priority task τ_h may generate on τ_i in the longest level-i busy window. Note that $\eta_h^+(W_{i,l})$ can be computed and used as proposed in [5], e.g., see Equation 3 in [5].

A. Bounding the Bus Blocking

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

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

Lemma 1. *The maximum number of times the tasks running on a local core π_l can suffer bus blocking in a window of length $W_{i,l}$ is given by:*

$$N_{\pi_l}(W_{i,l}) = \sum_{\tau_h \in hep_{i,l}} \eta_h^+(W_{i,l}) + 1 \tag{2}$$

Proof. We know that in the longest level-i busy window all jobs (except the first job) of the local core can only execute after the completion of the R-phase of the job executed just before on the same core. Since the A-phase of a task starts immediately after the R-phase execution of the previous job, each job (except the first) does not suffer blocking before its A-phase and thus can only suffer bus blocking once i.e., before its R-phase. Thus, the bus blocking in $W_{i,l}$ is bounded by the maximum number of jobs released by tasks in $hep_{i,l}$ in $W_{i,l}$, which is bounded by $\sum_{\tau_h \in hep_{i,l}} \eta_h^+(W_{i,l})$. Two cases are considered for the additional 1 in the equation: (case 1) if the first job in the busy window is a job from a lower priority task, it will suffer bus blocking only at its R-phase as lower priority task can only cause blocking to task τ_i after it starts executing its A-phase on the bus. Therefore, the additional 1 in the equation accounts for the bus blocking of one job of this lower priority task at its R-phase; (case 2) if τ_i does not suffer any blocking from lower priority task (e.g. if τ_i is the lowest priority task) then the additional one accounts for bus blocking suffered by the first job executed in the longest level-i busy window at its A-phase. Hence, the maximum number of times the tasks running on core π_l can suffer bus blocking in $W_{i,l}$ are bounded by $\sum_{\tau_h \in hep_{i,l}} \eta_h^+(W_{i,l}) + 1$. □

Lemma 2. *The maximum number of times tasks running on a remote core π_r can cause bus blocking in a time window of length $W_{i,l}$ is upper bounded by $N_{\pi_r}(W_{i,l})$, where*

$$N_{\pi_r}(W_{i,l}) = \sum_{\tau_u \in \Gamma_r} \eta_u^+(W_{i,l}) \quad (3)$$

The proof is similar to that of Lemma 1 except that it accounts for all tasks executing on core because any task running on core π_r can participate in the bus blocking.

B. Maximum Bus Blocking

Having bounded the maximum number of bus blockings suffered by the core under analysis π_l in $W_{i,l}$ (i.e. $N_{\pi_l}(W_{i,l})$) and caused by any remote core π_r in $W_{i,l}$ (i.e. $N_{\pi_r}(W_{i,l})$), we can now derive the maximum bus blocking that a task τ_i executing on core π_l can suffer due to tasks that may execute in parallel with τ_i on core π_r in the level-i busy window of length $W_{i,l}$. For this, three cases must be considered: (i) $N_{\pi_l}(W_{i,l}) > N_{\pi_r}(W_{i,l})$, (ii) $N_{\pi_l}(W_{i,l}) = N_{\pi_r}(W_{i,l})$ and (iii) $N_{\pi_l}(W_{i,l}) < N_{\pi_r}(W_{i,l})$. We will compute the maximum bus blocking for case 1 using Lemma 3 and for case 2 using Lemma 4. The maximum bus blocking for case 3 is then derived by dividing case 3 into two sub-cases.

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

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

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

where $\hat{N}_{\pi_r}$ is equal to the value of $N_{\pi_r}(W_{i,l})$ computed using Equation 2. Note that $C_{r,x}^A$ and $C_{r,y}^R$ may or may not belong to different jobs released on core π_r in $W_{i,l}$.

Lemma 3. *If $N_{\pi_l}(W_{i,l}) > N_{\pi_r}(W_{i,l})$, then the maximum bus blocking i.e., $Bus_{i,r}(W_{i,l})$ caused by tasks running on core π_r to the tasks running on core π_l in time window $W_{i,l}$ is:*

$$Bus_{i,r}(W_{i,l}) = \sum_{x=1}^{\hat{N}_{\pi_r}} C_{r,x}^A + \sum_{y=1}^{\hat{N}_{\pi_r}} C_{r,y}^R \quad (4)$$

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

Proof. Each bus blocking caused by π_r can be composed of either an A-, or an R-phase of a job, or one R- and one A-phase of two different jobs released on core π_r in $W_{i,l}$ (remember that the A-phase of a job can execute immediately after the R-phase of another job on π_r). Since, the precise bus access time of tasks running on core π_r is unknown, there can be a scenario in which all the memory phases of all the jobs of core π_r released in $W_{i,l}$ participate to the bus blocking if $N_{\pi_l}(W_{i,l}) > N_{\pi_r}(W_{i,l})$. Therefore, the maximum contribution of $\hat{N}_{\pi_r}$ jobs of core π_r to bus blocking, i.e., $\sum_{x=1}^{\hat{N}_{\pi_r}} C_{r,x}^A + \sum_{y=1}^{\hat{N}_{\pi_r}} C_{r,y}^R$ upper bounds the maximum bus blocking core π_r can cause on the tasks of core π_l in $W_{i,l}$. □

Lemma 4. *If $N_{\pi_l}(W_{i,l}) = N_{\pi_r}(W_{i,l})$, then the maximum bus blocking $Bus_{i,r}(W_{i,l})$ caused by tasks running on core π_r to tasks running on core π_l in a time window $W_{i,l}$ is given by:*

$$\sum_{x=1}^{\hat{N}_{\pi_r}} C_{r,x}^A + \sum_{y=1}^{\hat{N}_{\pi_r}} C_{r,y}^R - \min\left(\min_{\forall x \in M_r^A}\{C_{r,x}^A\}, \min_{\forall y \in M_r^R}\{C_{r,y}^R\}\right) \quad (5)$$

Proof. To prove this, we consider two cases:

Case 1. If the A-phase of the first job on π_r participate to the bus blocking of any job of π_l released in $W_{i,l}$, then the first bus blocking is composed of only one A-phase 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 $W_{i,l}$. Consequently, the R-phase of the last job executing on π_r within $W_{i,l}$ cannot participate to $Bus_{i,r}(W_{i,l})$.

Case 2. If the A-phase of the first job on π_r participating to the bus blocking of τ_i does not block the memory-phase of any job of π_l released in $W_{i,l}$, then all the memory phases except the A-phase of the first job executing on π_r within $W_{i,l}$ can contribute to $Bus_{i,r}(W_{i,l})$, as the first bus blocking is composed of an R-phase of first job and A-phase of any other job executed on π_r within $W_{i,l}$.

Considering both cases above, either one A-phase or one R-phase does not participate to $Bus_{i,r}(W_{i,l})$. Thus, $Bus_{i,r}(W_{i,l})$ is maximised when the non-participating memory phase is the smallest among those in M_r^A and M_r^R, hence proving the lemma. □

If $N_{\pi_l}(W_{i,l}) < N_{\pi_r}(W_{i,l})$, then only $N_{\pi_l}(W_{i,l})$ bus blockings can be caused by tasks running on core π_r to the tasks running on core π_l in $W_{i,l}$. To extract $N_{\pi_l}(W_{i,l})$ number of A and R-phases with higher memory demand, we can simply divide the set M_r^A (resp. M_r^R) into two sub-sets named M_r^{AH} and M_r^{AL} (resp. M_r^{RH} and M_r^{RL}). The subset M_r^{AH} (resp. M_r^{RH}) contains $N_{\pi_l}(W_{i,l})$ number of A-phases (resp. R-phases) with maximum memory demand and rest of the A-phases (resp. R-phases) are in the M_r^{AL} (resp. M_r^{RL}) as follows:

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

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

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

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

The maximum bus blocking can be computed by considering the memory phases of M_r^{AH} and M_r^{RH}. If each element of M_r^{AH} and M_r^{RH} belongs to the exact same set of jobs then there will be $\hat{N}_{\pi_l}$ (where $\hat{N}_{\pi_l} = N_{\pi_l}(W_{i,l})$) number of jobs that are involved in the bus blocking, otherwise it is greater than $\hat{N}_{\pi_l}$. Therefore, we will consider two sub-cases; (1) when the number of jobs involved in the bus blocking is greater than $\hat{N}_{\pi_l}$ and (2) when the number of jobs involved in the bus blocking is equal to $\hat{N}_{\pi_l}$.

Sub-case 1: If the number of jobs involved in the bus blocking is greater than $\hat{N}_{\pi_l}$ then the maximum bus blocking $Bus_{i,r}(W_{i,l})$ can be derived by considering all the memory phases of M_r^{AH} and M_r^{RH} as given below.

$$Bus_{i,r}(W_{i,l}) = \sum_{x=1}^{\hat{N}_{\pi_l}} C_{r,x}^A + \sum_{y=1}^{\hat{N}_{\pi_l}} C_{r,y}^R \tag{6}$$

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

Sub-case 2: If the number of jobs involved in the bus blocking is equal to $\hat{N}_{\pi_l}$ then either one A-phase or R-phase needs to be removed from the bus blocking (similarly to Lemma 4). Unlike Lemma 4, we have $N_{\pi_l}(W_{i,l}) < N_{\pi_r}(W_{i,l})$ that means possibly a memory phase from M_r^{AL} or M_r^{RL} can participate in the bus blocking. To compute the maximum bus blocking in this sub-case, we will remove the smallest A-phase or R-phase from M_r^{AH} or M_r^{RH} and will add the largest A-phase or R-phase from M_r^{AL} or M_r^{RL}, as follows:

$$\begin{aligned} Bus_{i,r}(W_{i,l}) &= \sum_{x=1}^{\hat{N}_{\pi_l}} C_{r,x}^A + \sum_{y=1}^{\hat{N}_{\pi_l}} C_{r,y}^R \\ - \min\Big(&(\min_{\forall x \in M_r^{AH}} \{C_{r,x}^A\} - \max_{\forall y \in M_r^{AL}} \{C_{r,y}^A\}), \\ &(\min_{\forall x \in M_r^{RH}} \{C_{r,x}^R\} - \max_{\forall y \in M_r^{RL}} \{C_{r,y}^R\})\Big) \end{aligned} \tag{7}$$

As the bus arbitration policy is FCFS, the maximum bus blocking suffered by core π_l from all the remote cores in a time window of length $W_{i,l}$ is given by $Bus_{i,l}^{max.}(W_{i,l})$, where

$$Bus_{i,l}^{max.}(W_{i,l}) = \sum_{r=1, r\neq l}^{m} Bus_{i,r}(W_{i,l}) \tag{8}$$

IV. WCRT Analysis

As proven in [1], 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 $W_{i,l}$. Having computed $W_{i,l}$ by Equation 1, the maximum number of jobs of task τ_i that can execute within $W_{i,l}$ is given by:

$$K_i = \eta_i^+(W_{i,l}) \tag{9}$$

To compute the response time of the k^{th} job of τ_i on core π_l, denoted by $\tau_{i,k,l}$, we first need to compute the starting time of the R-phase of $\tau_{i,k,l}$. This is due to the fact that each job executing on core π_l, including $\tau_{i,k,l}$, can suffer bus blocking at its R-phase. Assuming $s_{i,k,l}^R$ denote the starting time of the R-phase of $\tau_{i,k,l}$ on core π_l, then $s_{i,k,l}^R$ can be computed using the following iterative equation:

$$\begin{aligned} s_{i,k,l}^R = C_{lp,i,l}^{max} + \sum_{h \in hep_{i,l} \setminus \tau_i} \eta_h^+(s_{i,k,l}^R - (C_i^A + C_i^E)) \times C_h + \\ Bus_{i,l}^{max.}(s_{i,k,l}^R) + (k-1) \times C_i + (C_i^A + C_i^E) \end{aligned} \tag{10}$$

For Equation 10, the computation of $C_{lp,i,l}^{max}$ is similar to Equation 1. Similarly, $Bus_{i,l}^{max.}(s_{i,k,l}^R)$ is computed using Equation 8; $(k-1) \times C_i$ represents the time $k-1$ jobs of τ_i take to execute before starting $\tau_{i,k,l}$. Since any task of core π_l cannot preempt the execution of $\tau_{i,k,l}$ once it starts executing its A-phase on the bus, the maximum interference from higher or equal priority tasks (except τ_i) running on core π_l is captured by $\eta_h^+(s_{i,k,l}^R - (C_i^A + C_i^E)) \times C_h$; where C_h is the WCET of task τ_h in isolation. As $s_{i,k,l}^R$ appears on both sides of Equation 10, it can be solved iteratively by initializing $s_{i,k,l}^R = C_i^A + C_i^E + C_{lp,i,l}^{max} + C_h$. The starting time $s_{i,k,l}^R$ will then be given by the smallest positive value of $s_{i,k,l}^R$ for which Equation 10 converges.

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

$$R_{i,k,l} = s_{i,k,l}^R + C_i^R \tag{11}$$

Finally, the WCRT of task τ_i can be computed by maximizing equation 11 over all jobs of τ_i that execute during the level-i busy window, i.e., $R_{i,l}^{max} = \max_{k \in [1,K_i]} \{R_{i,k,l}\}$

If the WCRT of each task in the task set is less than or equal to its relative deadline, then the task set is deemed schedulable, otherwise it is not.

V. Conclusion

In this work, we propose an approach to analyze bus blocking suffered by tasks that execute using the 3-phase execution model. For future work, we will evaluate the accuracy of our analysis.

References

[1] R. J. Bril, J. J. Lukkien, and 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)*, pages 269–279, 2007.

[2] Guy Durrieu, Madeleine Faugère, Sylvain Girbal, Daniel Gracia Pérez, Claire Pagetti, and W. Puffitsch. Predictable Flight Management System Implementation on a Multicore Processor. In *Embedded Real Time Software (ERTS'14)*, TOULOUSE, France, February 2014.

[3] Claudio Maia, Geoffrey Nelissen, Luis Nogueira, Luis Miguel Pinho, and Daniel Gracia 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)*, pages 1–10, Hsinchu, Taiwan, August 2017. IEEE.

[4] Rodolfo Pellizzoni, Emiliano Betti, Stanley Bak, Gang Yao, John Criswell, Marco Caccamo, and Russell Kegley. A Predictable Execution Model for COTS-Based Embedded Systems. In *2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 269–279, Chicago, IL, USA, April 2011. IEEE.

[5] Simon Schliecker and Rolf Ernst. Real-time performance analysis of multiprocessor systems with shared memory. *ACM Transactions on Embedded Computing Systems*, 10(2):1–27, December 2010.

[6] K W Tindell. An Extendible Approach for Analysing Fixed Priority Hard Read-Time Tasks. page 16.