Correctness and Efficiency Criteria for the Multi-Phase Task Model

Rémi Meunier, Thomas Carle, Thierry Monteil

Correctness and Efficiency Criteria for the Multi-Phase Task Model

Rémi Meunier
IRIT, AUSY, INSA Toulouse, France

Thomas Carle
IRIT, Universite Toulouse 3 Paul Sabatier, CNRS, France

Thierry Monteil
IRIT, INSA Toulouse, CNRS, France

Abstract

This paper investigates how the multi-phase representation of real-time tasks impacts their implementation and the precision of the interference analysis in a multi-core context. In classical scheduling and interference analyses, tasks are represented as a single phase with a duration equal to their Worst-Case Execution Time (WCET) in isolation, annotated with their worst-case number of accesses. We propose a general formal definition of a task model in which tasks are represented as a sequence of such phases: the multi-phase model. We then provide a set of general correction criteria for the implementation of tasks represented in the multi-phase model, which is agnostic of the analysis method applied on the tasks. We also use the multi-phase model on an avionics case-study and study its impact on the interference analysis. Finally, we define a set of efficiency criteria using a statistical study of the most efficient multi-phase shapes.

2012 ACM Subject Classification Computer systems organization → Real-time systems; Computer systems organization → Multicore architectures; Computer systems organization → Embedded software

Keywords and phrases Task model, Interference, Multicore architectures

Digital Object Identifier 10.4230/LIPIcs.ECRTS.2022.9

Funding This work was supported by a grant overseen by the French National Research Agency (ANR) as part of the MeSCAliNe (ANR-21-CE25-0012) project.

1 Introduction

The growing adoption of multi-core processors in industrial real-time systems [21, 22] raises the challenge of providing safe and tight Worst-Case Execution Time (WCET) bounds for tasks running in parallel on separate cores. Indeed, in multi-core architectures, the cores execute their processes/threads independently from one another, but they share some hardware components such as caches, buses and memories. Interference may happen in these shared components: when a task requires to access a component which is already in use by another task running on another core, it has to wait until the component is free again. This phenomenon incurs execution delays which depend on the context of the task execution (which other tasks are running in parallel, and are they accessing the shared resources ?). In traditional single-core WCET analysis [1, 3], each task is analysed in isolation i.e. as if no other task was running in parallel. Then a schedulability or Worst-Case Response Time (WCRT) analysis is performed using a model in which each task is represented by its WCET, in order to guarantee that each individual task meets its deadline or that the system as a whole meets an end-to-end timing constraint. A direct consequence of the delays incurred by interference is that traditional WCETs no longer represent a safe upper-bound on the execution time of the tasks when they are run on multi-core processors. It becomes necessary to model tasks using at least their WCET in isolation and their worst-case number

of accesses to shared components, and to perform an additional interference analysis in order to obtain a safe over-estimation of their execution time. However, this classical model, which maps one task to one temporal phase was not designed with interference analysis in mind, and may not be the best-suited to analyse tasks running in parallel.

More recent models represent each task as a sequence of *phases*, each characterized by a WCET and a number of accesses, either as an attempt to increase the precision of the interference analysis [2, 4, 19], or in order to build schedules in which there is no interference [9, 20]. This *multi-phase* abstraction maps temporal phases to actual sections of code which are separated by synchronizations. The shape of the phases may be dictated by a programming model in which the programmers insert synchronizations at particular points (e.g. [9]), or may be designed during the analysis and enforced by synchronizations that are injected in the code afterwards (e.g. [5]).

In the remainder of the paper, we focus on memory accesses as the sole source for interference in the system. However, the abstractions that we describe naturally support any other kind of interference source: the only thing that changes is the analyses that must be performed on the code in order to obtain the abstract models of the tasks, which are not considered in the scope of the paper. In particular, we consider non-preemptive static (or fixed-priority time-triggered) scheduling, but limited preemptions could be supported by adapting classic cache-related preemption delay (CRPD) computation techniques.

In this paper we make the following contributions:

- we give a general, formal description of the multi-phase model that is agnostic of the method used to analyze the tasks. As such it can be seen as a generalization of the 2-phase [18] and 3-phase [9] models, and as an extension of the model of [5] to the notion of synchronizations.
- we investigate the relationship between the phases and the synchronizations that enforce them, that is to say between the multi-phase analysis model of tasks and their actual implementation. We show that synchronizations are not mandatory at the boundaries between two consecutive phases, and provide criteria to validate the implementation of a task w.r.t. the assumptions that were made during the analysis. Once again the criterion is agnostic about the way the analysis is actually performed.
- we apply the multi-phase model on an avionics case-study, and show that correlating the shape of the model to the actual behavior of the tasks may not lead to the best results.
- As a result we investigate how the shape of multi-phase representations impacts the result of static interference analysis, and deduce a promising heuristic objective to build multi-phase models of tasks. Since we discuss a general model and not a particular method, this study is based on a statistical experiment relying on synthetic artifacts.

The paper is organized as follows: we present the related work in Section 2 and we introduce formal definitions in Section 3. Then, in Section 4 we study the relationship between phases and synchronizations. In Section 5, we present our results on the ROSACE case-study. In Section 6, we detail our statistical study of the multi-phase model, and finally we conclude in Section 7.

2 Related Work

The problem of identifying, quantifying and possibly reducing the interference between real-time tasks running on multi-core processors is one of the most pressing issues that the real-time community is facing. Consequently, a lot of work has been done already, and various methods have been developed, although none of them seems to be entirely satisfying [13].

One of the most comprehensive approaches so far, presented in [6], uses the execution traces of the tasks composing the system, and models all the hardware components shared between cores (e.g. buses, RAMs) in order to quantify the exact worst-case amount of interference in the system, given a fixed priority scheduling policy. This analysis method is thus able to provide a safe and tight interference-aware WCRT for a task system. However the authors point out that working with all the possible execution traces of all the tasks composing a system is not feasible for realistic, industrial systems. They advocate the use of a more abstract representation of the tasks execution in order to overcome this intractability issue, but do not provide such an abstraction. It thus remains open to find a suitable candidate abstraction to achieve a trade-off between tractability and precision of the analysis.

Other approaches, inspired by compilation methods and low-level code analysis, use static scheduling approaches to handle, reduce or suppress the interference in the system. These methods are inspired by the PRedictable Execution Model (PREM) introduced in [18]. This model, originally designed for single-core processors with Input/Output registers, abstracts the execution of tasks as a sequence of so called memory and execution phases. In a memory phase, the core executes only memory (or I/O register) operations, which require to send or receive data across an interconnect. In an execution phase however, the code is executed locally in the core, and is guaranteed not to make accesses to the interconnect. As a result, only the memory phases are subject to interference. The Acquisition Execution Restitution (AER) model [9] can be seen as an extension of the PREM model to the context of avionics systems running on multi-core architectures. In AER, each task is divided into exactly three phases: an acquisition phase that loads to a core-local memory (cache or scratchpad) all the code and data which may be necessary to execute the task, an execution phase that executes the task code locally, and a restitution phase in which the results of the task are written back to the shared memory. A static schedule of the tasks is computed in which the acquisition and restitution phases of tasks are guaranteed not to happen simultaneously, and synchronizations are inserted to enforce this schedule. As a result, the system is completely free from interference. This line of work simplifies the analysis, by completely suppressing the interference in the system, but also introduces some limitations of its own. First, in these approaches, the tasks software has to be written with the phases in mind: this imposes constraints on the way the code is written, and makes it challenging to use legacy code. Automatic transformations of functions into the AER/PREM model have been developed as an attempt to shift the design burden from the programmer to the compiler [10, 11, 14–16, 23]. However, all these methods perform consequent modifications of the applications in order to make them comply with the AER/PREM model, and thus do not solve the limitations regarding legacy code. As a second limitation, the pre-loading of the tasks code and data increases the memory requirements of the system by forcing the load of code and data which may not be used during the execution phase (e.g. due to conditional execution). Finally, the static nature of these solutions is compatible to systems in which certification constraints are very high (e.g. avionics systems), but may be too rigid for less critical applications.

In [23], the authors evaluate the PREM and AER models in different scenarios in which interference is either prohibited or can be tolerated. They conclude that for a given task system, tolerating interference is more effective in terms of WCRT than building a schedule without interference at all. In the same spirit, the Time Interest Points (TIPs) approach [5] has been developed to reconcile the multi-phased task model with legacy code. In this model, a multi-phase representation of the tasks is obtained through static analysis of the binary code of the tasks. As a consequence, no restriction is put on the way the source code must be written. Each phase of a task may perform a certain amount of memory accesses,

which is determined by the analysis. Using this representation, an interference analysis can be performed as part of a WCRT analysis such as the one in [6], or as part of a static scheduling/compiling approach (e.g. [7,8]). Moreover, the analysis can be tuned in order to produce different representations of the same task (as in [5]). However, to the best of our knowledge, no study has yet been performed in order to define what a "good" multi-phased representation is. In this paper we provide a first answer to this question.

3 Formal Models

Notation	Definition
τ^i	task i
ϕ^i_k	phase k in the representation of τ^i
$\phi^i_k.d$	start date of ϕ^i_k without interference
$\phi^i_k.dur$	worst-case duration of ϕ^i_k without interference
$\phi^i_k.m$	maximum number of memory accesses performed within ϕ^i_k
t^i_j	execution trace j of task τ^i
$\eta^i_{j,k}$	node k in trace t^i_j
$\eta^i_{j,k}.it$	instruction represented by $\eta^i_{j,k}$
$\eta^i_{j,k}.d$	worst-case execution date of $\eta^i_{j,k}$ without interference
$\eta^i_{j,k}.m$	maximum number of memory accesses performed by $\eta^i_{j,k}$
$\eta^i_{j,k}.sync$	True if the node is synchronized, i.e. cannot be executed before its $\eta^i_{j,k}.d$
$s_{last}(\eta^i_{j,k})$	last synchronized node before $\eta^i_{j,k}$ in trace t^i_j
$t^i_j\vert_{\phi^i_k}$	restriction of trace t^i_j to ϕ^i_k, i.e. the set of nodes in t^i_j that may execute during ϕ^i_k

We model a system of real-time tasks τ^i ($i \geq 0$). Each task is represented in two separate ways, as depicted in Figure 1:

- a "time-centric" representation called *multi-phase*. In this abstraction, the task is modelled by a sequence of time slots, called *phases*, which covers its WCET. We call this sequence of phases a *profile*. Each phase is associated with an upper bound on the number of memory accesses that the task can perform during the corresponding time slot. This representation is used to statically compute the schedule and perform the interference analysis of the system (in a timing-compositional approach). The mapping between tasks and multi-phase profiles is not bijective: multiple profiles can be found which represent the same task.
- a "code-centric" representation. In this abstraction, a task is represented by all its possible *execution traces* (i.e. all the possible sequences of instructions executed from the start of the task to its end). Since this set may be too large to analyze in practice, we consider memory-centric traces: only instructions which may perform memory accesses[1] are represented in the traces, and the rest of the instructions is abstracted by computing local WCETs. This representation is an intermediate step to go from the binary code

[1] In modern processors, the actual accesses may not be performed as soon as the corresponding instruction is executed e.g. if a store buffer delays store operations. However it is possible to statically bound the time window during which the access can be performed. For clarity reasons, we consider in this paper only the date when the instruction initiates the access in the pipeline, but the model can be easily extended to work with an interval of potential access dates.

of a task to its multi-phase representation, and back: it allows the number of memory accesses in each phase to be bounded correctly, and to insert synchronization code at the correct locations in the binary to enforce the scheduling choices.

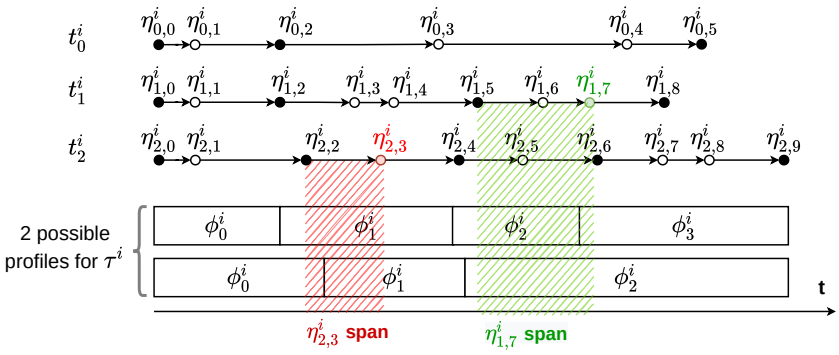


Figure 1 Three traces and two profiles for task τ^i. Red and green rectangles show the potential span of nodes $\eta^i_{2,3}$ and $\eta^i_{1,7}$ respectively.

The scope of this paper does not include the methods required to obtain trace-based and phase-based representations of tasks. We focus instead on the relationship between these two abstraction levels. In this section, we describe these models formally and present this relationship.

3.1 Task Models

We denote $\mathbb{P}^i = \{\phi^i_l | 0 \leq k < \Phi^i\}$ the ordered set of phases (i.e. the multi-phase *profile*) representing the execution of task τ^i, with Φ^i the number of phases. Each ϕ^i_l is defined by:

- $\phi^i_l.d$: its start date.
- $\phi^i_l.dur$: its worst-case duration in isolation (without interference).
- $\phi^i_l.m$: the worst-case number of memory accesses that may be performed within $[\phi^i_l.d, \phi^i_l.d + \phi^i_l.dur[$.

The date of ϕ^i_0, which is also the start date of task τ^i without interference, is set when the static schedule of the system is built. Then, for each ϕ^i_l $(l > 0)$ the start date is defined by:

$$\phi^i_l.d = \phi^i_0.d + \sum_{0 \leq q < l} \phi^i_q.dur \tag{1}$$

Alternatively, we can define recursively the start date of each phase (except the first one) by:

$$\forall l > 0, \phi^i_l.d = \phi^i_{l-1}.d + \phi^i_{l-1}.dur$$

In order to compute the worst-case number of memory accesses performed during a given phase (i.e. $\phi^i_l.m$), the code portions of τ^i that may be executed during ϕ^i_l must be identified and analyzed. To do so, we introduce $\mathbb{T}^i = \{t^i_j | 0 \leq j < T^i\}$ the set of *execution traces* of τ^i, where T^i is the number of traces. Each trace corresponds to a possible execution of τ^i (corresponding to a particular set of inputs) and is a sequence of nodes $\eta^i_{j,k}$ representing instructions with $0 \leq k < N^i_j$ the node's index in its sequence. $\eta^i_{j,0}$ is the *entry point* of task τ^i and each node is defined by:

- $\eta^i_{j,k}.it$: the instruction represented by $\eta^i_{j,k}$. Here, an instruction is not just understood as an element of the core ISA (e.g. the ADD instruction), but as a particular instruction in the binary code of the task. Thus, nodes from different traces may reference the same instruction in the code.
- $\eta^i_{j,k}.m \in \mathbb{N}$: the worst-case number of memory accesses performed by $\eta^i_{j,k}.it$.
- $\eta^i_{j,k}.d$: the worst-case execution date of $\eta^i_{j,k}.it$ in trace t^i_j.

3.2 Synchronized Nodes

As pointed out in Section 2, working on the complete set of execution traces of all tasks composing the system is not realistic. As a consequence, we formulate our correctness criteria using memory-centric *abstract traces*: the nodes composing the traces that we consider only represent the instructions that may perform memory accesses. The rest of the instructions is abstracted by computing local WCETs between consecutive memory accesses and accounting for these durations in the worst-case execution date of the nodes ($\eta^i_{j,k}.d$)[2]. As a result, in this model each node is guaranteed not to execute after its worst-case date, but is *a priori* able to execute anytime before this date. In order to account safely for the accesses in the phases, we thus would have to account for the accesses performed by a node in all phases that start before the worst-case date of this node. This would lead to huge over-approximations. In order to limit this approximation, some selected nodes must be *synchronized*: synchronization code is inserted in the program to ensure that the synchronized nodes cannot be executed before their worst-case date. The synchronization code can be added by the programmer directly in the source code of the tasks, by the compiler as part of a low-level compilation pass, or during an automatic code re-engineering process to adapt legacy code to the multi-phase model. We attract the reader's attention to two particular aspects of the model described in Section 3.1: (i) the execution date for nodes that reference the same instruction in different traces and (ii) the modelling of instructions inside loops which may appear multiple times in the same trace at different dates. Both these aspects have to do with the way synchronizations are implemented in the tasks. When complex synchronization mechanisms are used (e.g. that are aware of the current execution trace or of the iteration count in the current loop), the same memory instruction in the code may be modelled as two (or more) nodes with different dates, which perfectly fits the model. If, however, the synchronization mechanism is unaware of the context, the worst-case execution date of nodes that reference the same instruction on separate traces must be the same. Since the model uses worst-case dates, the date chosen for all these nodes must be the maximum date amongst them. Additionally, without a context-aware mechanism, synchronizations inside loops become impossible to implement, so the model naturally fits this case. In this paper, we voluntarily keep the model as general as possible and make no assumption on the implementation of the synchronization mechanisms in order to formulate correctness criteria that apply in all circumstances.

To keep track of the synchronized nodes, we add the boolean attribute $\eta^i_{j,k}.sync$ which is true if the node is synchronized and false otherwise.

Using these synchronizations, the accesses performed by any node must only be accounted for in the phases that: (1) finish after the last synchronization prior to the node **AND** (2) start before the worst-case date of the node.

[2] Our criteria are also valid for simple tasks for which obtaining and manipulating the exact timed execution traces is possible

This is illustrated in Figure 1, which depicts 3 execution traces (t_0^i, t_1^i and t_2^i) and 2 possible profiles for a task τ^i. Synchronized nodes are depicted in black in the traces. The red (resp. green) rectangle shows the time window in which the accesses of node $\eta_{2,3}^i$ (resp. $\eta_{1,7}^i$) must be accounted for. In the first profile, the accesses of $\eta_{1,7}^i$ must be considered in phases ϕ_2^i and ϕ_3^i, whereas in the second profile, they would only be considered in ϕ_3^i.

It is important to note that since $\eta_{j,k}^i.d$ is a worst-case date, if node $\eta_{j,k}^i$ is synchronized, then its execution date is exactly[3] $\eta_{j,k}^i.d$. We denote $s_{last}(\eta_{j,k}^i)$ the last synchronized node before $\eta_{j,k}^i$ in trace t_j^i. By convention, we set $s_{last}(\eta_{j,k}^i) = \eta_{j,k}^i$ when $\eta_{j,k}^i.sync$.

To account for the tasks schedule, for all tasks τ^i, the entry node (on any trace t_j^i) is synchronized and its worst-case execution date is set to the start of the first phase of the profile:

▶ **Property 1.** $\forall i, j : \eta_{j,0}^i.sync \wedge \eta_{j,0}^i.d = \phi_0^i.d$

The worst-case date of any other node $\eta_{j,k}^i$ with $k > 0$ is defined according to the date of the last synchronized node on its trace:

▶ **Property 2.** $\eta_{j,k}^i.d = \eta_{j,s}^i.d + \sum_{s \leq t < k} wcet(\eta_{j,t}^i.it, \eta_{j,t+1}^i.it)$

where $wcet(\eta_{j,t}^i.it, \eta_{j,t+1}^i.it)$ is the WCET between instructions $\eta_{j,t}^i.it$ and $\eta_{j,t+1}^i.it$, and $\eta_{j,s}^i$ is $s_{last}(\eta_{j,k}^i)$ if $\neg\eta_{j,k}^i.sync$ and $s_{last}(\eta_{j,k-1}^i)$ otherwise.

A node $\eta_{j,k}^i$ can only be executed in the interval $[s_{last}(\eta_{j,k}^i).d, \eta_{j,k}^i.d]$. As we saw in the example of Figure 1, this interval may overlap with several phases of the task profile.

We denote $t_j^i|_{\phi_l^i}$ the set of nodes in trace t_j^i that may be executed within $[\phi_l^i.d, \phi_l^i.d + \phi_l^i.dur[$, called the *restriction* of trace t_j^i to phase ϕ_l^i:

$$t_j^i|_{\phi_l^i} = \{\eta_{j,k}^i | (\eta_{j,k}^i.d \geq \phi_l^i.d) \wedge (s_{last}(\eta_{j,k}^i).d < \phi_l^i.d + \phi_l^i.dur)\}$$

The notion of restriction of a trace to a phase is illustrated in Figure 2 on 3 traces over phase ϕ_1^i.

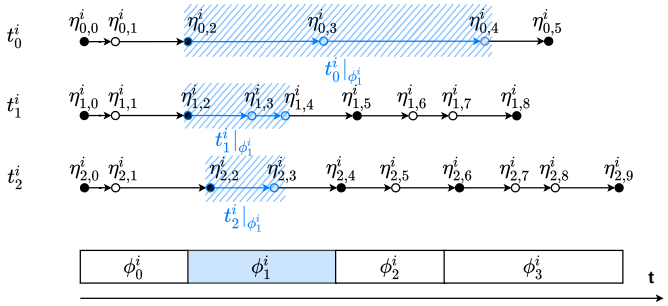


Figure 2 Restrictions of traces t_0^i, t_1^i and t_2^i to phase ϕ_1^i.

3.3 Maximum Number of Accesses in a Phase

The number of accesses that may be performed during a phase for an individual trace is equal to the sum of the accesses of the nodes from this trace that may be executed in the phase. During the execution of a task, only one trace executes (which one depends on the

[3] With a precision of a few cycles depending on the implementation of the synchronization mechanism.

execution context): as a consequence, the worst-case number of accesses performed during a phase is equal to the maximum number of accesses that may be performed by any execution trace during that phase.

▶ **Property 3.** *The worst-case number of accesses that may be performed during phase ϕ_l^i, denoted $\phi_l^i.m$, is equal to the maximum of accesses per trace during phase ϕ_l^i:*

$$\phi_l^i.m = \max_{0 \leq j < T^i} \Big(\sum_{\eta_{j,k}^i \in t_j^i |_{\phi_l^i}} \eta_{j,k}^i.m \Big)$$

▶ **Correctness criterion 1.** *The formula of Property 3 provides a conservative estimation of the number of memory accesses that can occur during the phases of a multi-phase profile.*

Since nodes may span over multiple phases, the number of accesses counted task-wise may be overestimated, even when some nodes are synchronized. However, nodes from a trace which span over multiple phases may be "covered" by other nodes from another trace performing more accesses on a given phase. For example, in Figure 2, if we consider that each node performs 1 access, trace t_2^i is the local worst trace on ϕ_3^i with 4 nodes performing accesses and trace t_1^i is the local worst trace on ϕ_2^i with 3 nodes performing accesses. On phase ϕ_1^i, traces t_0^i and t_1^i both have 3 nodes performing accesses. In such circumstances, although node $\eta_{0,4}^i$ spans over ϕ_3^i, ϕ_2^i and ϕ_1^i, it does not contribute to any over-approximation.

We quantify the task-wise over-approximation of memory accesses compared to the 1-phase model, by computing the difference between the sum of accesses accounted for in each phase, and the worst trace-wise number of accesses.

▶ **Property 4.** *The memory access over-approximation in a multi-phase profile of a task τ^i compared to its 1-phase representation is equal to:*

$$\Delta = \Big(\sum_{0 \leq l < \Phi^i} \phi_l^i.m \Big) - \max_{0 \leq j < T^i} \Big(\sum_{0 \leq k < N_j^i} \eta_{j,k}^i.m \Big)$$

4 Consequences of the Interference Analysis

Notation	Definition
$\phi_l^i.p$	timing penalty added to ϕ_l^i due to potential interference
$\phi_l^i.d^\#$	*post-analysis* start date of ϕ_l^i
$\eta_{j,k}^i.d^\#$	worst-case date of node $\eta_{j,k}^i$ in the presence of interference

In this section, we consider a task system for which an analysis has provided a multi-phase model as well as a selection of synchronized nodes for each task. We assume that this task system is scheduled statically (the $\phi_0^i.d$ for each τ^i are selected and the start dates of the other phases are computed using equation 1), and that an interference analysis (such as e.g. [7]) is applied to compute and account for the effect of potential interference between the tasks phases, assuming the timing-compositionality of the target processor [12]. In practice, each phase that potentially suffers from interference is extended using a time penalty, and the next phases are postponed accordingly. This extension may violate assumptions that were made on the correspondence between phases and traces: in particular the restrictions of traces to phases that were computed prior to the interference analysis may no longer be correct, resulting in the possibility that some contentions between cores may happen in phases in which they were not accounted for.

4.1 Example

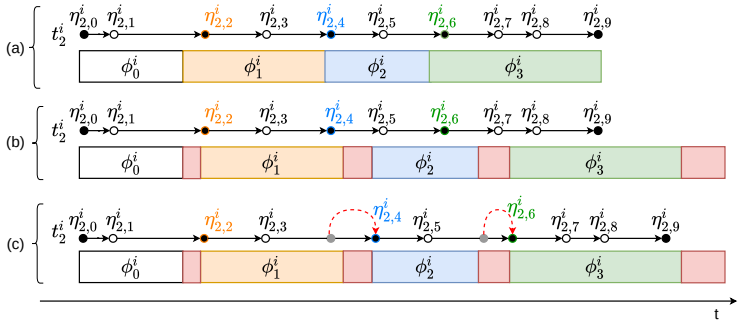


Figure 3 A trace and its corresponding phases representation : (a) in isolation, (b) after the interference analysis, red rectangles are the timing penalty added for each phase, (c) after a correction on nodes dates.

Figure 3 displays trace t_2^i and the profile from Figure 2, at three stages of the analysis:

- (a) depicts the trace and phases before the interference analysis. We have:
  $t_2^i|_{\phi_0^i} = \{\eta_{2,0}^i, \eta_{2,1}^i\}$; $t_2^i|_{\phi_1^i} = \{\eta_{2,2}^i, \eta_{2,3}^i\}$; $t_2^i|_{\phi_2^i} = \{\eta_{2,4}^i, \eta_{2,5}^i\}$; $t_2^i|_{\phi_3^i} = \{\eta_{2,6}^i, \eta_{2,7}^i, \eta_{2,8}^i, \eta_{2,9}^i\}$
  Additionally, we consider that for this task, $\phi_1^i.m = 2$ and $\phi_2^i.m = 2$.
- (b) shows the same trace and profile after the interference analysis (assuming other tasks in the system): the effect of interference is materialized by timing penalties on the phases (the red rectangles after each phase). $t_2^i|_{\phi_1^i}$, $t_2^i|_{\phi_2^i}$ and $t_2^i|_{\phi_3^i}$ are different than in (a):
  $t_2^i|_{\phi_0^i} = \{\eta_{2,0}^i, \eta_{2,1}^i\}$; $t_2^i|_{\phi_1^i} = \{\eta_{2,2}^i, \eta_{2,3}^i, \eta_{2,4}^i, \eta_{2,5}^i\}$; $t_2^i|_{\phi_2^i} = \{\eta_{2,5}^i, \eta_{2,6}^i, \eta_{2,7}^i, \eta_{2,8}^i\}$; $t_2^i|_{\phi_3^i} = \{\eta_{2,8}^i, \eta_{2,9}^i\}$
  As a consequence, the worst-case amount of accesses that can happen during phases ϕ_1^i and ϕ_2^i is higher than what was assumed and therefore their interference penalty and those of the tasks scheduled in parallel are no longer conservative.
- (c) represents a solution to respect the model's assumptions of (a): the synchronized date of $\eta_{2,4}^i$ (resp. $\eta_{2,6}^i$) is set to the new starting date of ϕ_2^i (resp. ϕ_3^i), which is the unique phase in which it was accounted for in (a). With this slight modification, the restrictions of t_2^i to each phase are identical to the ones in (a) and the $\phi_l^i.m$ that was computed in isolation for each phase remains correct.

4.2 Enforcing the Model's Assumptions and the Analysis Results

Since the duration and start dates of phases can be changed as a result of the interference analysis, new attributes are added to the formal model of the phases:

- $\phi_l^i.p \geq 0$ is the timing penalty added to ϕ_l^i due to potential interference. It is a conservative bound computed during the interference analysis.
- $\phi_l^i.d^\#$ is the *post-analysis* date of ϕ_l^i, i.e. its start date taking into account the potential interference in the system.

After the interference analysis, the start date of some tasks may be postponed due to interference that delays previous tasks. $\phi_0^i.d^\#$ is thus fixed by applying the interference analysis results to the initial schedule. The start dates of all other phases ϕ_l^i describing the execution of τ^i are computed as:

$$\phi_l^i.d^\# = \phi_0^i.d^\# + \sum_{0 \leq q < l} (\phi_q^i.dur + \phi_q^i.p) \tag{2}$$

▶ **Correctness criterion 2.** *The synchronization dates in the final implementation of tasks must at least be equal to the start date of the corresponding phase: for each synchronization node* $\eta^i_{j,k} \in t^i_j|_{\phi^i_n}$*, the synchronization date is set to at least* $\phi^i_n.d^\#$*. This way it is guaranteed that nodes after* $\eta^i_{j,k}$ *cannot execute and thus produce accesses before the start of* ϕ^i_n*.*

It seems straightforward that, by construction, a task set implemented using this rule is guaranteed to fulfill the assumptions made during the interference analysis: during the execution of the system, memory accesses will only occur at times that were accounted for during the analysis, and thus the amount of interference cannot be larger in practice than what was accounted for. However, although this implementation rule directly guarantees that accesses are not performed before the phases in which they are accounted for, it may be harder to convince oneself that they cannot occur later than the end of these phases. Consequently, and given the potentially critical nature of the tasks modelled in the multi-phase representation, we provide in the remainder of the section a formal proof of the correctness of this implementation scheme w.r.t. the result of the interference analysis. Once again, this is completely agnostic of the analysis method, as long as it correctly provides a conservative bound on the interference level.

We denote $\eta^i_{j,k}.d^\#$ the *post-analysis* worst-case date of node $\eta^i_{j,k}$. The post-analysis dates of nodes are upper bounds on the worst-case execution dates of nodes in the presence of interference. We start by characterizing those bounds in our formal model (Properties 5, 6 and 7), and then use them to prove the correctness of the implementation of a multi-phase model of tasks.

First, the post-analysis execution date of the entry point of each task τ^i is the post analysis start date of its first phase ϕ^i_0.

▶ **Property 5.** *For any task* τ^i*:* $\forall j < T^i, \eta^i_{j,0}.d^\# = \phi^i_0.d^\#$

Second, correctness criterion 2 has the following consequences for the post-analysis execution date of any synchronized node $\eta^i_{j,k}$ (except the entry point) of any task τ^i:

- if the phase ϕ^i_n in which the node was supposed to be executed is postponed due to interference penalties on previous phases, the node cannot be executed before the post-analysis start date of ϕ^i_n.
- if previous synchronized nodes see their execution dates postponed, the synchronization date of $\eta^i_{j,k}$ must be postponed accordingly, and thus computed from the post-analysis date of the previous synchronized node $\eta^i_{j,s}$. In this case, we must consider the interference that can take place between $\eta^i_{j,s}$ and $\eta^i_{j,k}$. If there exists one or more phases that span entirely between the two nodes, their penalties are added to the post-analysis date of $\eta^i_{j,k}$ (which is conservative). Moreover, by convention we count in the post-analysis date of $\eta^i_{j,k}$ the entire amount of penalty of the phase to which it belongs (which is also conservative since it accounts for the interference that can occur on each access in the phase prior to the synchronization node, and on each access that may occur until the next synchronization node).

▶ **Property 6.** *For any synchronized node* $\eta^i_{j,k}$ *of any trace* t^i_j *of task* τ^i*:*

$$(k > 0 \wedge \eta^i_{j,k}.sync \wedge \eta^i_{j,k} \in t^i_j|_{\phi^i_n} \wedge \eta^i_{j,s} = s_{last}(\eta^i_{j,k-1}) \wedge \eta^i_{j,s} \in t^i_j|_{\phi^i_m})$$

$$\Rightarrow \eta^i_{j,k}.d^\# = max(\phi^i_n.d^\#, \eta^i_{j,s}.d^\# + \sum_{s \leq l < k} wcet(\eta^i_{j,l}.it, \eta^i_{j,l+1}.it) + \sum_{m < b \leq n} \phi^i_b.p)$$

▶ **Correctness criterion 3.** *The synchronization dates in the final implementation of tasks must not be set to a value higher than the date computed in Property 6.*

Finally, for any non-synchronized node, its post-analysis date accounts for the possible postponing of the previous synchronized node $\eta^i_{j,s}$. Note that the potential interference occurring between them has been accounted for entirely in the post-analysis date of the previous synchronized node.

▶ **Property 7.** *For any non-synchronized node $\eta^i_{j,k}$ of any trace t^i_j of task τ^i:*

$$(\neg\eta^i_{j,k}.sync \wedge \eta^i_{j,s} = s_{last}(\eta^i_{j,k})) \Rightarrow \eta^i_{j,k}.d^\# = \eta^i_{j,s}.d^\# + \sum_{s \leq l < k} wcet(\eta^i_{j,l}.it, \eta^i_{j,l+1}.it)$$

4.3 Proof of Correctness

We now prove that any task system which respects the 3 correctness criteria is correct w.r.t. the results of the interference analysis i.e. cannot generate interference that was not accounted for.

First, the difference between the start date of a synchronized node $\eta^i_{j,k}$ before and after the interference analysis is bounded by the difference between the start date of the phase ϕ^i_l in which it is executed, before and after the interference analysis, added to the maximum amount of interference that can occur in ϕ^i_l.

▶ **Lemma 1.** $\forall\eta^i_{j,k}$: $(\eta^i_{j,k}.sync \wedge \eta^i_{j,k} \in t^i_j|_{\phi^i_l}) \Rightarrow \eta^i_{j,k}.d^\# - \eta^i_{j,k}.d \leq \phi^i_l.d^\# - \phi^i_l.d + \phi^i_l.p$

Proof. We will prove by induction that the property is true for all synchronized nodes. If $\eta^i_{j,k}$ is the entry node of τ^i, the proof is direct using Properties 1 and 5. Otherwise, using Property 6, $\eta^i_{j,k}.d^\#$ is either equal to $\phi^i_l.d^\#$ or must be computed from the previous synchronized node on trace t^i_j. Let $\eta^i_{j,s} = s_{last}(\eta^i_{j,k-1})$, and assume that the property is true for $\eta^i_{j,s}$. Then,

- If $\eta^i_{j,k}.d^\# = \phi^i_l.d^\#$:
  since $\eta^i_{j,k} \in t^i_j|_{\phi^i_l}$, by definition $\eta^i_{j,k}.d \geq \phi^i_l.d$, and thus $\eta^i_{j,k}.d^\# - \eta^i_{j,k}.d \leq \phi^i_l.d^\# - \phi^i_j.d$.
- Otherwise:
  $\eta^i_{j,k}.d^\# = \eta^i_{j,s}.d^\# + \sum_{s \leq a < k} wcet(\eta^i_{j,a}.it, \eta^i_{j,a+1}.it) + \sum_{m < b \leq l} \phi^i_b.p$. Using Property 2, we get: $\eta^i_{j,k}.d^\# - \eta^i_{j,k}.d = \eta^i_{j,s}.d^\# - \eta^i_{j,s}.d + \sum_{m < b \leq l} \phi^i_b.p$. The induction hypothesis gives us: $\eta^i_{j,s}.d^\# - \eta^i_{j,s}.d \leq \phi^i_m.d^\# - \phi^i_m.d + \phi^i_m.p$, where ϕ^i_m is the phase in which $\eta^i_{j,s}$ executes. If $m = l$ (i.e. both nodes execute in the same phase) the property is directly proven for node $\eta^i_{j,k}$. Otherwise, $m < l$ and then $\phi^i_l.d^\# - \phi^i_l.d = \phi^i_m.d^\# - \phi^i_m.d + \sum_{m \leq b < l} \phi^i_b.p$ (using Equations 1 and 2), and thus the property is also proven.

By induction, we just proved that the property holds for all synchronized nodes. ◀

We are now ready to prove the correctness property:

▶ **Theorem 1.** *For any task system that respects correctness criteria 1, 2 and 3, for any $\eta^i_{j,k}$ of any task τ^i, if $\eta^i_{j,k}$ spans over a phase ϕ^i_l after the interference analysis, then $\eta^i_{j,k}$ was necessarily accounted in the restriction of trace t^i_j to ϕ^i_l before the analysis:*

$$\forall 0 \leq j < T^i, \forall 0 \leq k < N^i_j, \forall 0 \leq l < \Phi^i :$$

$$[s_{last}(\eta^i_{j,k}).d^\#, \eta^i_{j,k}.d^\#] \cap [\phi^i_l.d^\#, \phi^i_l.d^\# + \phi^i_l.dur + \phi^i_l.p[\neq \emptyset \Rightarrow \eta^i_{j,k} \in t^i_j|_{\phi^i_l}$$

Proof. The case where $\eta^i_{j,k}$ is the entry node is direct. For all other nodes we consider separately the case of synchronized nodes and of non-synchronized nodes.

Case 1: $\eta^i_{j,k}.sync$ is true:
By convention, $s_{last}(\eta^i_{j,k}) = \eta^i_{j,k}$. Let us assume ϕ^i_l such that $\eta^i_{j,k}.d^\# \in [\phi^i_l.d^\#, \phi^i_l.d^\# + \phi^i_l.dur + \phi^i_l.p[$. Let us denote ϕ^i_z the phase such that $\eta^i_{j,k} \in t^i_j|_{\phi^i_z}$ (z is unique because $\eta^i_{j,k}$ is synchronized). We want to prove that $l = z$. Using Property 6, either $\eta^i_{j,k}.d^\# = \phi^i_z.d^\#$ or it is greater. If it is equal, then directly $\phi^i_l = \phi^i_z$ because phases of the same task do not overlap. Otherwise, if $z > l$ then $\eta^i_{j,k}.d^\# > \phi^i_z.d^\# \geq \phi^i_l.d^\# + \phi^i_l.dur + \phi^i_l.p$ which contradicts the assumption. So z would have to be less than l. Now, since $\eta^i_{j,k} \in t^i_j|_{\phi^i_z}$, $\eta^i_{j,k}.d - \phi^i_z.d < \phi^i_z.dur$. At the same time, $\eta^i_{j,k}.d^\# \geq \phi^i_l.d^\# \geq \phi^i_z.d^\# + \phi^i_z.dur + \phi^i_z.p$, so $\eta^i_{j,k}.d^\# - \phi^i_z.d^\# \geq \phi^i_z.dur + \phi^i_z.p$. This contradicts Lemma 1, from which we conclude that $l = z$. This concludes the proof for case 1.

Case 2: $\eta^i_{j,k}.sync$ is false:
Let ϕ^i_l such that $[s_{last}(\eta^i_{j,k}).d^\#, \eta^i_{j,k}.d^\#] \cap [\phi^i_l.d^\#, \phi^i_l.d^\# + \phi^i_l.dur + \phi^i_l.p[\neq \emptyset$. Let us denote ϕ^i_m the phase to which $s_{last}(\eta^i_{j,k}).d^\#$ belongs, and assume by absurd that $\eta^i_{j,k} \notin t^i_j|_{\phi^i_l}$.
Then by definition either $(s_{last}(\eta^i_{j,k}).d > \phi^i_l.d + \phi^i_l.dur)$ or $(\eta^i_{j,k}.d < \phi^i_l.d)$.
If $s_{last}(\eta^i_{j,k}).d > \phi^i_l.d + \phi^i_l.dur$: then $m > l$, and thus using Property 6: $s_{last}(\eta^i_{j,k}).d^\# \geq \phi^i_m.d^\# \geq \phi^i_l.d^\# + \phi^i_l.dur + \phi^i_l.p$, which contradicts the original assumption.
If $\eta^i_{j,k}.d < \phi^i_l.d$, then using Property 2: $s_{last}(\eta^i_{j,k}).d + \sum_{s \leq t < k} wcet(\eta^i_{j,t}.it, \eta^i_{j,t+1}.it) < \phi^i_l.d$.
Then, we can deduce:

$$
\begin{aligned}
& s_{last}(\eta^i_{j,k}).d^\# + \sum_{s \leq t < k} wcet(\eta^i_{j,t}.it, \eta^i_{j,t+1}.it) < \phi^i_l.d + s_{last}(\eta^i_{j,k}).d^\# - s_{last}(\eta^i_{j,k}).d \\
\overset{Prop.\ 7}{\Rightarrow} \quad & \eta^i_{j,k}.d^\# < \phi^i_l.d + s_{last}(\eta^i_{j,k}).d^\# - s_{last}(\eta^i_{j,k}).d \\
\Rightarrow \quad & \eta^i_{j,k}.d^\# < \phi^i_l.d + s_{last}(\eta^i_{j,k}).d^\# - s_{last}(\eta^i_{j,k}).d + \sum_{b=m+1}^{l-1} \phi^i_b.p \\
\overset{Lemma\ 1}{\Rightarrow} \quad & \eta^i_{j,k}.d^\# < \phi^i_l.d + \phi^i_m.d^\# - \phi^i_m.d + \phi^i_m.p + \sum_{b=m+1}^{l-1} \phi^i_b.p \\
\Rightarrow \quad & \eta^i_{j,k}.d^\# < \phi^i_m.d^\# + \phi^i_m.p + \sum_{b=m+1}^{l-1} \phi^i_b.p + \sum_{b=m}^{l-1} \phi^i_b.dur \\
\Rightarrow \quad & \eta^i_{j,k}.d^\# < \phi^i_l.d^\#
\end{aligned}
$$

which contradicts the initial hypothesis. We conclude that necessarily $\eta^i_{j,k} \in t^i_j|_{\phi^i_l}$. ◀

We just proved that the correctness criteria that we enumerated in the first part of the paper guarantee that the implementation of a task system described in the multi-phase model is correct w.r.t. a chosen interference-aware static schedule. These criteria are very simple, which makes them easy to verify and offers a lot of room for optimizations in the analysis of tasks, both in order to derive a profile for tasks and to select the synchronization nodes. In the remainder of the paper, we concentrate on the efficiency of the model w.r.t. the interference analysis. We start by experimenting the multi-phase model on a case study, and then perform a statistical analysis in order to derive general efficiency criteria which can in the future serve as an objective function for analysis heuristics.

5 Efficiency of the Multi-Phase Model on the ROSACE Case-Study

ROSACE [17] is a flight controller case-study composed of 15 communicating tasks running at different frequencies. We followed the Time Interest Points methodology described in [5] to obtain the worst-case execution traces and multi-phase profiles for the ROSACE tasks. Basically, we used the OTAWA static analysis tool to:

- Detect the instructions that are not statically guaranteed to result in a cache hit.
- Build an "abstract" CFG in which the nodes are the instructions that were detected in the previous step. Each edge of this graph is decorated with the WCET of the code portion between its source and sink nodes, computed using OTAWA.
- Build the execution traces by enumerating this graph. In our experiments on ROSACE, the average number of traces by task was around 88, with a peak at 1280 for the $aircraft\_dynamics$ task. The graph enumeration may lead to combinatorial explosion for arbitrarily complex applications. This issue can be mitigated by adding extra synchronizations in the traces (e.g. at the end of a if-then-else or loop construct) that factorize multiple traces for the rest of the enumeration.
- For each trace, generate a multi-phase profile in which each memory access has a dedicated phase spanning the duration of the access, using the worst-case dates in the trace.
- Build the intersection of the profiles of all traces. This intersection is a profile that keeps all access phases from all traces. The rest of the profile is composed of phases guaranteed to feature no access.
- From this profile, extract the phases with a size larger than a parameter δ in which no access occurs. Parameter δ, which we varied in our experiments, specifies a minimum size threshold for the phases of the generated profiles.
- For the remainder of the phases in the intersection profile, fuse them together if their duration is less than δ.

This method creates multi-phase profiles for tasks but says nothing about the selection of the synchronization nodes. We thus added a very simple method to select synchronization nodes, using the correctness criteria of Sections 3 and 4. For each phase ϕ_l^i, we selected as synchronized node for each trace t_j^i the first node $\eta_{j,k}^i$ with $\phi_l^i.d \leq \eta_{j,k}^i.d < \phi_l^i.d + \phi_l^i.dur$ (if such a node exists). None of the tasks needed context-aware synchronizations so the synchronization date was always chosen as the worst-case date of the synchronized instruction in the program. The combination of the heuristic and of our node selection pass does not perform any optimization. In our analysis, we considered a target hardware architecture composed of a multicore processor (2, 4 or 8 cores) in which each core features a L1 LRU data cache, and an instruction scratchpad which holds the totality of the code needed by the core to execute. Additionally, we considered a memory latency of 50 cycles for non-cached accesses. The tasks were compiled for ARM targets, and we considered that the cores were running at a frequency of 10MHz (otherwise the tasks WCETs were too small compared to their periods so there was no interference in the schedule).

Table 1 presents statistics on the multi-phase profiles of the ROSACE tasks for 3 values of δ. This parameter has a consequent impact on the number of phases, on the number of synchronizations and on the over-approximation (defined in Property 4) in the generated profiles. With a δ equal to 50 cycles (the memory latency of our targets), 569 phases are generated for all tasks. Our method determined 315 synchronizations nodes which corresponds to roughly 1 synchronization every 6 or 7 instructions on average. This number may be too high to be realistically used in practice, but is in part due to the small tasks of the case-study which are composed of only a few tens of instructions. As δ grows, the

Table 1 Rosace multi-phase tasks statistics for 3 multi-phase minimum duration (δ).

Task	instr (#)	traces (#)	$\delta = 50$ ov_app (%)	$\delta = 50$ sync (#)	$\delta = 50$ phases (#)	$\delta = 500$ ov_app (%)	$\delta = 500$ sync (#)	$\delta = 500$ phases (#)	$\delta = 1000$ ov_app (%)	$\delta = 1000$ sync (#)	$\delta = 1000$ phases (#)
engine	40	1	0	10	18	0	4	4	0	2	2
elevator	47	1	0	9	15	0	3	4	0	2	2
aircraft_dyn	1217	1280	21.18	92	167	12.94	37	38	11.76	22	23
h_filter	77	4	0	18	34	0	7	7	0	5	4
az_filter	77	4	0	17	32	0	7	7	0	5	4
q_filter	106	12	14.29	32	55	0	11	9	0	8	5
vz_filter	106	12	10.34	32	55	0	11	9	0	8	5
va_filter	77	4	0	17	32	0	7	7	0	5	4
h_command	18	1	0	7	13	0	2	2	0	1	1
altitude_hold	65	3	6.25	15	26	0	7	6	0	5	3
vz_control	70	1	0	23	44	0	7	8	0	4	4
va_control	73	1	0	24	45	0	7	8	0	4	4
va_command	18	1	0	7	13	0	2	2	0	1	1
delta_th	15	1	0	6	10	0	2	2	0	1	1
delta_e	15	1	0	6	10	0	2	2	0	1	1
TOTAL	2021	1327	7.88	315	569	3.33	116	115	3.03	74	64
Gain (2 cores)				7.83%			6.79%			6.01%	
Gain (4 cores)				13.21%			8.45%			7.48%	
Gain (8 cores)				7.60%			3.74%			2.14%	

number of generated phases and of synchronization nodes get lower: 115 phases and 116 synchronizations for $\delta = 500$, and 64 phases and 74 synchronizations for $\delta = 1000$, with 4 tasks having only 1 phase in their representation. Our method for selecting the synchronizations and counting the accesses in each phase is basic: it respects correctness criterion 1, but does not optimize the number of synchronizations. As a result, on each trace one node is selected as synchronization for each phase, even when it is not necessary. This explains in part the high count for synchronizations. These synchronizations can be implemented by a variety of methods. One naive method is to poll a register that counts the number of cycles (e.g. a time stamp counter) until the start date of the corresponding phase is reached. This can be implemented with a simple loop composed of only a few instructions (depending on the ISA it can be as small as 3 instructions – compare, conditional jump, jump back). The precision of each synchronization depends on the depth of the pipeline (which imposes the duration of the jumps), but these small overheads can be taken into account in the analysis, and they do not accumulate as the number of phases grows, because the synchronizations are based on dates, not durations. Context-aware synchronizations (e.g. inside loops) are more complex to implement, and may impose restrictions on the analysis. Since there was no need for those in the ROSACE case-study, we leave for future work the efficient and correct implementation of context-aware synchronizations.

Figure 4 displays the generated traces and two profiles for the *az_filter* task. When $\delta = 1000$, the profile is composed exclusively of phases in which the number of memory accesses is strictly positive. On the other hand, when $\delta = 50$ most of the profile is composed of phases guaranteed to perform no memory access.

Using the generated profiles, we produced static schedules of the application for target processors featuring 2, 4 and 8 cores. The schedules represent one hyper-period of the tasks. The tasks were mapped to cores following their utilization (using a simple greedy algorithm) and scheduled using the rate monotonic policy with the frequencies specified in the original paper [17]. We then performed an interference analysis: we detected the phases that overlap with each other 2 by 2 on different cores and extended them using a penalty computed as the maximum number of contentions multiplied by the cost of a

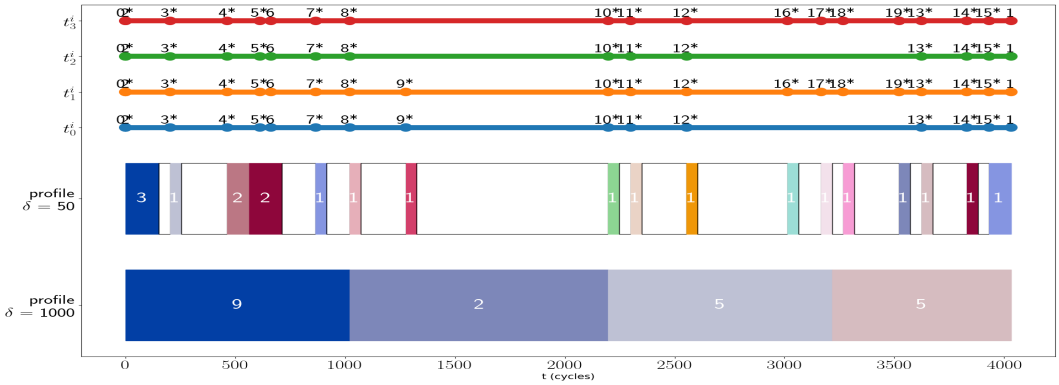


Figure 4 *az_filter* task: traces (top) and generated profile (bottom).

memory access. Moreover, the total penalty that a phase can suffer from any other core (taken separately) is bounded by the number of accesses of the phase. This is a classical interference computation assuming e.g. a FIFO bus. We measured the total reserved time for the tasks (including the interference penalties) on the hyper-period, using the classical 1-phase model, and using our generated multi-phase profiles. We computed the gain as: $gain = (time\_1\_phase - time\_multi\_phase)/time\_1\_phase$ and reported it in Table 1. The multi-phase model yields a gain of 7.83% (resp. 13.21% and 7.60%) with δ equal to 50 cycles on 2 cores (resp. 4 and 8 cores). The gain gets reduced as δ grows and the generated profiles resemble more and more the 1-phase model. However, even in the case of $\delta = 1000$, the multi-phase model outperforms the 1-phase model by 6.01% (resp. 7.48%, 2.14%) on 2 cores (resp. 4 cores, 8 cores). One noticeable point is that the profiles with a lower δ have a higher over-approximation and still get the best gains compared to the 1-phase model: over-approximation at the profile level is not an indicator of the good performance of the model at the task-system level.

Our conclusions from the case-study are the following:

- the multi-phase model can yield a substantial gain compared to the 1-phase model.
- the over-approximation in the profile of an individual task is not correlated to the gain obtained at the task system level. Consequently, the optimal profile for a task may not be derived from its execution behavior (when do the memory accesses happen?), but from extrinsic properties. As we show in Section 6.2, particular shapes of profiles behave significantly better than others during the interference analysis.
- a trade-off must be found during the construction of the tasks profiles between the number of synchronizations and the efficiency (the gain) of the model.
- this gain is computed after the schedule is built and the interference analysis is performed and is thus not accessible when the profiles are being constructed: other criteria must be found, which can be computed directly during the construction of the profiles.

In an attempt to find such criteria and to confirm these conclusions, we performed a statistical study which we describe in the next section.

6 Profile Shape-Based Efficiency Criteria

In this section, we investigate how the shape of multi-phase profiles impacts the result of the interference analysis. As we show in this section, we found efficiency criteria which concern the multi-phase model itself and are thus extrinsic to the analysed tasks. To do

so, we conducted a statistical study on synthetic profiles generated using multiple input parameters that are summed up in Table 2. Profiles are generated by choosing the values for the attributes of the phases, using random draws from normal distributions centered around the input parameters: for example inside a generated profile, each phase has its own duration and number of accesses, but in average the durations and number of accesses meet the input parameters. We do not consider system-level parameters such as task periodicity, data dependencies or elaborate mapping and scheduling strategies in this section because our focus is on showing how the model reacts in the presence of interference. We thus choose a setting in which a lot of tasks are executed in parallel with no slack time.

Table 2 Tests input parameters.

Parameter	Values section 6.2	Values section 6.3
Over-approximation of accesses (%)	0 to 30 (step 5)	0 to 30 (step 5)
Nb tasks per core	{1, 2, 3}	{2, 3, 4, 5}
Interference time penalty (cycles)	{5, 10, 20}	{5, 10}
Accesses per cycle	0.01 to 0.1 (step 0.01)	0.01 to 0.1 (step 0.01)
Nb long phases per UP L type task	{5, 10, 15, 20}	{5, 10, 15, 20}
Avg. number of short phases per long phase	{2, 3, 4, 5}	{2, 3, 4, 5}
Task duration (cycles)	17,500 to 175,000	17,500 to 175,000

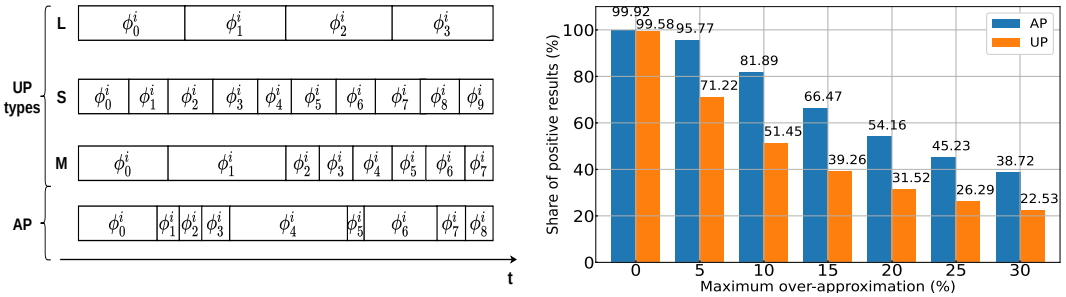
6.1 Tests Execution and Metrics

The execution of a test consists in three steps. First, we generate the set of multi-phase profiles corresponding to the input parameters. The generated tasks all share the same period, have no explicit data dependency and have a synchronous release. Then, we map the tasks into a number of cores specified for each test and schedule them as soon as possible, with no optimization in the mapping or scheduling choices. This context stresses the multi-phase model. Finally, we perform an interference analysis on the scheduled task system, as we did with the ROSACE case-study. The over-approximation level (cf. Property 4) is an input parameter to the tests. The ROSACE case-study showed that this over-approximation is not directly correlated with the gain yielded by the multi-phase model, so the study of synthetic profiles will provide us valuable insight on the performance of the model. Since we schedule the tasks as soon as possible with no slack, we redefine the notion of *gain* using the end dates of the schedules obtained with the 1-phase and multi-phase models: $gain = (end\_1\_phase - end\_multi\_phase)/end\_1\_phase$.

6.2 Looking for the Best Multi-Phase Profile Shapes

We started by generating shapes composed of sequences of phases of similar durations (with a standard deviation of 500 cycles), which we call Uniform Profiles (UP). The 3 UP kinds generated are depicted in Figure 5a: long (L) profiles composed only of long phases, short (S) profiles composed only of short phases or mix (M) profiles divided in 2 equal parts with respectively long and short phases. We performed 352,800 tests with different combinations of generation parameters. Each core is only assigned tasks of one UP kind (S, L or M) and has the same number of tasks, which all have the same duration and are released synchronously. With this setup, only the kind of UP assigned may differ for each core, so an easy comparison between the various combinations of UP kinds can be performed.

We tested all the combinations of the profiles on 2, 4 and 8 cores. Our results showed that the best profiles hosted the 3 categories S, M and L with a majority of S profiles. This is coherent because the multi-phase model increases its gain whenever, for a given phase, the



(a) Examples of possible UP types and AP.

(b) Tests with positive gain according to the maximum over-approximation value included.

Figure 5 AP vs UP profiles.

number of accesses it may perform is higher than the number of possible concurrent accesses from phases running in parallel (and it does not exceed the total accesses of the tasks in parallel). In this experiment each task has strictly the same size, so the probability to fall into this case is higher with S profiles which are composed of more phases.

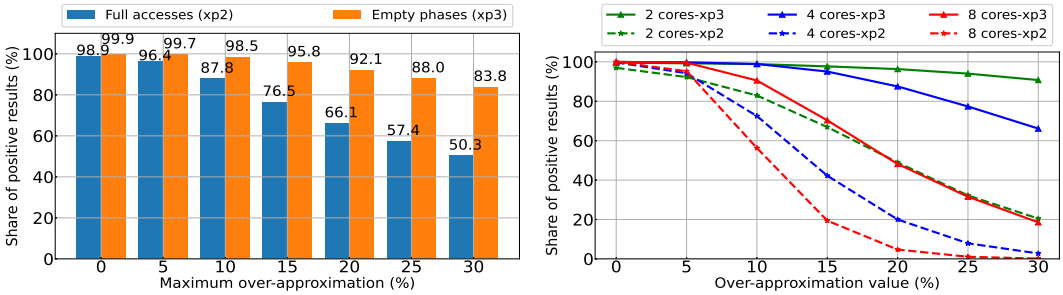
As the combination of short and long phases, with more short phases than long ones, performed better in our first experiment, we then tested a different profile shape that allows an alternation between short and long phases in the tasks as depicted in the bottom of Figure 5a. We name this profile shape Alternation Profile (AP). We compared the results of AP with the other profile shapes using the same test parameters. The results, given in Figure 5b, indicate that AP systematically outperforms UP regardless of the over-approximation value and of the other parameters. Therefore, we focus on AP as the best multi-phase profile for the rest of the experiments.

▶ **Efficiency criterion 1.** *Multi-phase profiles which alternate between long phases and grapes of small phases tend to perform better than other shapes.*

6.3 Comparison Between Multi-Phase AP and 1-Phase Model

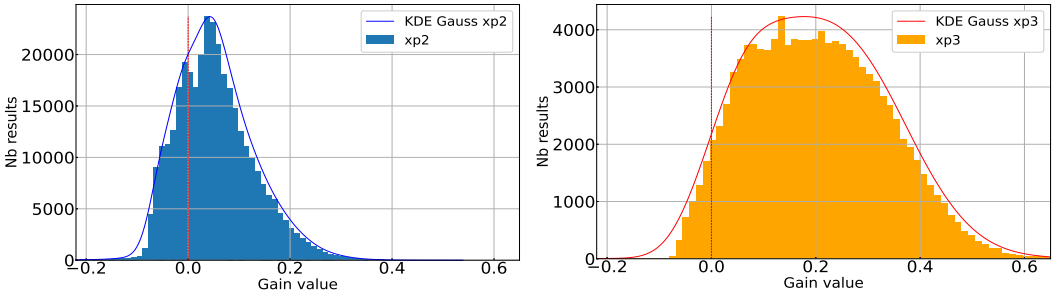
In this section we assess the performance of the AP shape using a pool of 806,400 tests in which tasks lengths vary in a range of ±25% around a value that is provided as input to the test generator. We first consider task profiles in which all phases perform memory accesses. Figure 6a gives the share of experiments in which the multi-phase AP outperforms its 1-phase counterpart for different values of maximum over-approximation (blue bars). When there is no over-approximation, the multi-phase model almost always performs better. The 1-phase model does not perform as well as the AP until all experiments with over-approximations ranging from 0% to 30% are included. According to Figure 6b (dashed lines), the share of positive results significantly decreases with the number of cores in our tests when the system-wise over-approximation value is superior to 10%, in particular when more than two cores are involved. This is coherent with the fact that the over-approximation of accesses is amplified as the number of cores grows when performing the interference analysis. In our experiments on the ROSACE case-study, the over-approximation always remained under 10% system-wise, regardless of the size of the phases. As a conclusion:

▶ **Efficiency criterion 2.** *The over-approximation plays a lesser role than the shape of the profile at the level of individual tasks, but should still be kept within acceptable levels system-wise, preferably under 10%.*



(a) Ratio of tests with positive gain. **(b)** Tests with positive gain by core number.

Figure 6 Experiments with accesses in each phase (*xp2*) and with phases without accesses (*xp3*).



(a) AP profiles without empty phases (*xp2*). **(b)** AP profiles with 1/3 of empty phases (*xp3*).

Figure 7 Gain distribution for AP profiles with 0 to 10% of over-approximation.

This can be achieved by increasing the number of synchronizations. A special effort must be put on the traces with the most accesses as the over-approximation at the phase-level is propagated to the task-level only if it concerns, for each phase, the trace having the most accesses. The over-estimation can also be reduced by fusing phases together when locally some traces perform most of their accesses in different phases.

Moreover, when the over-approximation cannot be lowered, scheduling optimizations can be applied in the same spirit as the ones used in AER in order to contain the negative effect of over-approximation.

In the profiles generated for the above experiments, all phases perform accesses. Nonetheless, the presence of phases without accesses in profiles is expected in practice e.g. due to cache effects, and is likely to improve the AP results. Indeed, phases running in parallel with other phases without accesses are guaranteed not to cause contention, while they would in the 1-phase model. Therefore, we performed a new series of tests to estimate the impact of phases without accesses in AP profiles. We modified the profiles already generated for the previous experiments, by randomly selecting one third of each task's phases and setting their accesses count to 0. The results are presented in Figures 6a (orange bars) and 6b (full lines). First, the share of positive results is significantly improved and still at more than 80% when including results with 30% of over-approximation, so the model is less sensitive to over-approximation. This is linked to situations where accesses due to over-approximation are in parallel with no other accesses so they do not lead to additional penalties. Second, the gain distribution with over-approximation ranging from 0 to 10%, given by Figures 7a and 7b, is also significantly improved with an average value of 20.1% while it is 5.3% for profiles without empty phases. Consequently:

▶ **Efficiency criterion 3.** *Phases that perform no access have a significant positive effect on the interference analysis results.*

As we saw with ROSACE the number of synchronizations, of phases that perform no access and the over-approximation can be adjusted as a trade-off in the analysis.

7 Conclusion and Future Work

We presented a formal framework for the multi-phase task model including a set of properties that guarantee the correctness of the implemented task system. These properties are agnostic about the methods that generate the profiles and select the location of the synchronizations that enforce them. We combined our criteria to a simple heuristic to obtain multi-phase representations of tasks on the ROSACE case-study, and concluded that the shape of the model impacts the efficiency of the interference analysis, regardless of the analyzed task. We thus conducted a statistical study in order to investigate which kinds of profiles perform the best. We concluded that profiles alternating long and short phases tend to perform better, and that profiles featuring phases that do not perform accesses are particularly efficient. As part of future work, we plan on defining an automatic method for the design of optimized multi-phase profiles from the tasks binary code, and on defining scheduling optimizations which benefit from the multi-phase model. Moreover, we will work on the efficient implementation of context-aware synchronizations, focusing on regular synchronization patterns inside loop iterations.

References

1 AbsInt. aiT. `https://www.absint.com/ait/index.htm`.

2 Jatin Arora, Cláudio Maia, Syed Aftab Rashid, Geoffrey Nelissen, and Eduardo Tovar. Bus-contention aware schedulability analysis for the 3-phase task model with partitioned scheduling. In Audrey Queudet, Iain Bate, and Giuseppe Lipari, editors, *RTNS'2021: 29th International Conference on Real-Time Networks and Systems, Nantes, France, April 7-9, 2021*, pages 123–133. ACM, 2021. `doi:10.1145/3453417.3453433`.

3 Clément Ballabriga, Hugues Cassé, Christine Rochange, and Pascal Sainrat. OTAWA: an open toolbox for adaptive WCET analysis. In Sang Lyul Min, Robert G. Pettit IV, Peter P. Puschner, and Theo Ungerer, editors, *Software Technologies for Embedded and Ubiquitous Systems - 8th IFIP WG 10.2 International Workshop, SEUS 2010, Waidhofen/Ybbs, Austria, October 13-15, 2010. Proceedings*, volume 6399 of *Lecture Notes in Computer Science*, pages 35–46. Springer, 2010. `doi:10.1007/978-3-642-16256-5_6`.

4 Thomas Carle and Hugues Cassé. Reducing timing interferences in real-time applications running on multicore architectures. In Florian Brandner, editor, *18th International Workshop on Worst-Case Execution Time Analysis, WCET 2018, July 3, 2018, Barcelona, Spain*, volume 63 of *OASIcs*, pages 3:1–3:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. `doi:10.4230/OASIcs.WCET.2018.3`.

5 Thomas Carle and Hugues Cassé. Static extraction of memory access profiles for multi-core interference analysis of real-time tasks. In Christian Hochberger, Lars Bauer, and Thilo Pionteck, editors, *Architecture of Computing Systems - 34th International Conference, ARCS 2021, Virtual Event, June 7-8, 2021, Proceedings*, volume 12800 of *Lecture Notes in Computer Science*, pages 19–34. Springer, 2021. `doi:10.1007/978-3-030-81682-7_2`.

6 Robert I. Davis, Sebastian Altmeyer, Leandro Soares Indrusiak, Claire Maiza, Vincent Nélis, and Jan Reineke. An extensible framework for multicore response time analysis. *Real Time Syst.*, 54(3):607–661, 2018. `doi:10.1007/s11241-017-9285-4`.

7 Maximilien Dupont de Dinechin, Matheus Schuh, Matthieu Moy, and Claire Maiza. Scaling up the memory interference analysis for hard real-time many-core systems. In *2020 Design, Automation & Test in Europe Conference & Exhibition, DATE 2020, Grenoble, France, March 9-13, 2020*, pages 330–333. IEEE, 2020. doi:10.23919/DATE48585.2020.9116460.

8 Keryan Didier, Dumitru Potop-Butucaru, Guillaume Iooss, Albert Cohen, Jean Souyris, Philippe Baufreton, and Amaury Graillat. Correct-by-construction parallelization of hard real-time avionics applications on off-the-shelf predictable hardware. *ACM Trans. Archit. Code Optim.*, 16(3):24:1–24:27, 2019. doi:10.1145/3328799.

9 G. Durrieu, M. Faugère, S. Girbal, D. Gracia Pérez, C. Pagetti, and W. Puffitsch. Predictable flight management system implementation on a multicore processor. In *ERTS'14*, 2014.

10 Björn Forsberg, Marco Solieri, Marko Bertogna, Luca Benini, and Andrea Marongiu. The predictable execution model in practice: Compiling real applications for COTS hardware. *ACM Trans. Embed. Comput. Syst.*, 20(5):47:1–47:25, 2021. doi:10.1145/3465370.

11 Frédéric Fort and Julien Forget. Code generation for multi-phase tasks on a multi-core distributed memory platform. In *25th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2019, Hangzhou, China, August 18-21, 2019*, pages 1–6. IEEE, 2019. doi:10.1109/RTCSA.2019.8864558.

12 Sebastian Hahn, Michael Jacobs, and Jan Reineke. Enabling compositionality for multicore timing analysis. In Alain Plantec, Frank Singhoff, Sébastien Faucou, and Luís Miguel Pinho, editors, *Proceedings of the 24th International Conference on Real-Time Networks and Systems, RTNS 2016, Brest, France, October 19-21, 2016*, pages 299–308. ACM, 2016. doi:10.1145/2997465.2997471.

13 Claire Maiza, Hamza Rihani, Juan Maria Rivas, Joël Goossens, Sebastian Altmeyer, and Robert I. Davis. A survey of timing verification techniques for multi-core real-time systems. *ACM Comput. Surv.*, 52(3):56:1–56:38, 2019. doi:10.1145/3323212.

14 Renato Mancuso, Roman Dudko, and Marco Caccamo. Light-prem: Automated software refactoring for predictable execution on COTS embedded systems. In *2014 IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications, Chongqing, China, August 20-22, 2014*, pages 1–10. IEEE Computer Society, 2014. doi:10.1109/RTCSA.2014.6910515.

15 Joel Matejka, Björn Forsberg, Michal Sojka, Zdenek Hanzálek, Luca Benini, and Andrea Marongiu. Combining PREM compilation and ILP scheduling for high-performance and predictable mpsoc execution. In Quan Chen, Zhiyi Huang, and Pavan Balaji, editors, *Proceedings of the 9th International Workshop on Programming Models and Applications for Multicores and Manycores, PMAM@PPoPP 2018, February 25, 2018, Vienna, Austria*, pages 11–20. ACM, 2018. doi:10.1145/3178442.3178444.

16 Claire Pagetti, Julien Forget, Heiko Falk, Dominic Oehlert, and Arno Luppold. Automated generation of time-predictable executables on multicore. In Yassine Ouhammou, Frédéric Ridouard, Emmanuel Grolleau, Mathieu Jan, and Moris Behnam, editors, *Proceedings of the 26th International Conference on Real-Time Networks and Systems, RTNS 2018, Chasseneuil-du-Poitou, France, October 10-12, 2018*, pages 104–113. ACM, 2018. doi:10.1145/3273905.3273907.

17 Claire Pagetti, David Saussié, Romain Gratia, Eric Noulard, and Pierre Siron. The ROSACE case study: From simulink specification to multi/many-core execution. In *20th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2014, Berlin, Germany, April 15-17, 2014*, pages 309–318. IEEE Computer Society, 2014. doi:10.1109/RTAS.2014.6926012.

18 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 *17th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2011, Chicago, Illinois, USA, 11-14 April 2011*, pages 269–279. IEEE Computer Society, 2011. doi:10.1109/RTAS.2011.33.

19 Rodolfo Pellizzoni, Andreas Schranzhofer, Jian-Jia Chen, Marco Caccamo, and Lothar Thiele. Worst case delay analysis for memory interference in multicore systems. In Giovanni De Micheli, Bashir M. Al-Hashimi, Wolfgang Müller, and Enrico Macii, editors, *Design, Automation and Test in Europe, DATE 2010, Dresden, Germany, March 8-12, 2010*, pages 741–746. IEEE Computer Society, 2010. doi:10.1109/DATE.2010.5456952.

20 Benjamin Rouxel, Stefanos Skalistis, Steven Derrien, and Isabelle Puaut. Hiding communication delays in contention-free execution for spm-based multi-core architectures. In Sophie Quinton, editor, *31st Euromicro Conference on Real-Time Systems, ECRTS 2019, July 9-12, 2019, Stuttgart, Germany*, volume 133 of *LIPIcs*, pages 25:1–25:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ECRTS.2019.25.

21 O. Sander, F. Bapp, L. Dieudonne, T. Sandmann, and J. Becker. The promised future of multi-core processors in avionics systems. *CEAS Aeronautical Journal*, 2017. doi:10.1007/s13272-016-0228-x.

22 J. Schneider, M. Bohn, and R. Rößger. Migration of automotive real-time software to multicore systems: First steps towards an automated solution. In *22nd EUROMICRO Conference on Real-Time Systems*, 2010.

23 Matheus Schuh, Claire Maiza, Joël Goossens, Pascal Raymond, and Benoît Dupont de Dinechin. A study of predictable execution models implementation for industrial data-flow applications on a multi-core platform with shared banked memory. In *41st IEEE Real-Time Systems Symposium, RTSS 2020, Houston, TX, USA, December 1-4, 2020*, pages 283–295. IEEE, 2020. doi:10.1109/RTSS49844.2020.00034.