Journal of Systems Architecture 157 (2024) 103300

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

Using MAST for modeling and response-time analysis of real-time applications with GPUs

Iosu Gomez [a,b,*], Unai Díaz de Cerio [a], Jorge Parra [a], Juan M. Rivas [b], J. Javier Gutiérrez [b], Michael González Harbour [b]

[a] Distributed and Connected Intelligence Department, IKERLAN Research Centre, Basque Research and Technology Alliance (BRTA), Arrasate-Mondragón, Spain
[b] Software Engineering and Real-Time Group, Universidad de Cantabria, Spain

ARTICLE INFO

Keywords:
Real time
Modeling
Schedulability analysis
Time partitioning
GPUs

ABSTRACT

The ever increasing computing demands in embedded systems is driving the adoption of hardware accelerators such as GPUs, which offer powerful platforms that can compute parallel workloads efficiently. Relevant critical applications that benefit from such platforms, for instance autonomous driving, usually impose additional real-time requirements that must be met to guarantee the correctness of the systems. In this paper, we propose exploiting readily available and extensively validated techniques to model and analyze real-time systems with GPUs. Specifically, we propose a methodology to employ the MAST model to characterize such systems, and different variants of the Offset-Based Response-Time Analysis techniques to validate the real-time requirements. We verify our approach with a real industrial application sourced from the railway industry. Through a comprehensive evaluation involving synthetic and real task-sets, we characterize the applicability of the approach, and we also show how estimated worst-case response times are aligned with real measurements up to 87.2%.

1. Introduction

Modern cyber–physical systems must handle increasingly computationally intensive workloads. For this reason, nowadays there is a trend to employ heterogeneous architectures with several processors and specific hardware accelerators to meet the new demands. Smart mobility applications use computationally intensive algorithms, such as Artificial Intelligence algorithms, to achieve advanced perception, planning or decision-making tasks. Additionally, these systems must meet both functional safety and real-time requirements. Thus, it is not only essential for the computation result to be functionally correct, it also has to comply with imposed time constrains. This field of research is of great industrial interest, as it can be seen in the various industrial challenges proposed at international conferences [1–3].

One of the more common hardware accelerators are the integrated GPUs (Graphics Processing Unit) architectures, which are platforms that contain a general purpose GPU integrated together with a CPU cluster in a single MPSoC (Multiprocessor System on Chip) connected by the system's global memory. GPUs enhance the performance of cyber–physical systems, enabling efficient parallel processing for data-intensive workloads. Although the computational power and efficiency of GPUs would clearly benefit emerging real-time applications such as smart mobility, their adoption for these types of applications is hindered by the unknown nature of their internal scheduling mechanisms. Managing these internal mechanisms is crucial for obtaining predictable response times and, subsequently, for the correct implementation of real-time applications. For that reason, there is presently an intense research effort to propose solutions that enable the safe usage of GPUs in real-time applications [4,5].

Furthermore, the concept of partitioning is a strategy extensively adopted in highly critical applications such as avionics, to ensure spatial and temporal isolation. This method creates time partitions, where applications allocated in different partitions can be effectively executed in isolation. In this paper, we will follow a similar terminology to the one defined in the ARINC 653 [6] avionics standard. Time partitioning is also a crucial aspect to consider in safety-critical systems running on heterogeneous platforms because of the isolation on critical functionalities that can be achieved [7]. For instance, one benefit of this kind of isolation is the possibility to control the CPU–GPU memory interference [8,9] caused by an intensive memory usage. Partitioning techniques have also been used before in multi-core systems [10] to mitigate the effects of interference. Another essential aspect that time

* Corresponding author at: Distributed and Connected Intelligence Department, IKERLAN Research Centre, Basque Research and Technology Alliance (BRTA), Arrasate-Mondragón, Spain.
E-mail address: iosu.gomez@ikerlan.es (I. Gomez).

https://doi.org/10.1016/j.sysarc.2024.103300
Received 31 January 2024; Received in revised form 26 September 2024; Accepted 30 October 2024
Available online 8 November 2024

partitioning can leverage [11], is managing concurrent accesses to the GPU when multiple tasks request off-loading workload to the accelerator. Coexisting workloads on the GPU can result in unpredictable timing behavior due to its unknown internal scheduling mechanisms [12].

This paper aims to model GPU-based real-time applications by using MAST (Modeling and Analysis Suite for real-Time applications) [13], which provides us with a model aligned with the OMG MARTE standard [14], as well as a set of response-time analysis tools for distributed systems. The latest version of the meta-model, MAST-2 [15], adds modeling elements to support time partitioning. In addition, GPUs are not directly supported by MAST. To overcome these limitations, the method proposed in this paper enables modeling GPUs with existing modeling elements and also allows the application of the latest response-time analysis techniques for hierarchically scheduled time-partitioned distributed real-time systems [16,17] using the currently available tools.

In order to demonstrate the applicability of the proposed method, we will model an industrial use case related to railway systems, in which the application, consisting of a genetic algorithm accelerated on a GPU, runs on a partitioned environment with temporal isolation. This kind of optimization algorithms are highly parallelizable, so they can benefit from the use of GPUs. Two different implementations of the parallel version of the genetic algorithm will be explored.

Furthermore, the analysis techniques that can be applied to these systems have a certain pessimism when calculating the upper bounds on the response times. Accordingly, this work will also include a characterization of the precision that is achieved by the application of this methodology. Different analysis techniques are proposed and evaluated in multiple configurations, first on a simple example that will help to understand the behavior of these systems, and later on the complete industrial use case.

The paper is organized as follows. Section 2 presents an overview of GPUs and related work on modeling real-time applications with GPUs. Section 3 describes the industrial use case addressed in this work. In Section 4, an overview of the MAST model and the available analysis techniques is given. This section also proposes how to model this kind of applications with MAST. Section 5 shows a characterization of the proposed methodology for modeling applications with GPUs through a simple example. In Section 6, this methodology is applied to the industrial use case. Section 7 presents an extensive evaluation of the proposed methodology. Finally, Section 8 shows the main conclusions, as well as possible future work.

2. Overview and related work

2.1. Overview on GPUs

In integrated GPU architectures, the GPU is embedded in the system together with the CPUs, sharing the global memory as a communication medium. In order to offload workload into the GPU, the CPU has to provide the necessary data to the GPU, launch the Kernel (the job to be executed in the GPU), and get the results back. There are three main memory management configurations to perform the data transfer between the CPU and the GPU: Standard Copy (SC), Zero-Copy (ZC), and Unified Memory (UM). The Standard Copy method involves copying the data between dedicated memory spaces of the CPU and GPU. The Zero-Copy method creates a common memory space accessible by both CPU and GPU; in this way, data can be exchanged efficiently via memory pointers. Lastly, the Unified Memory abstracts internal data copies by unifying CPU an GPU memory spaces, which simplifies data handling for developers. However, its underlying mechanisms are unknown and thus unpredictable [18].

Once a Kernel is launched, it is divided into thread blocks. The thread blocks go through different stages and schedulers until they are processed by the GPU's processing units, called Stream Multiprocessors (SM). There, the thread blocks are divided into 32 thread collections (Warps) and executed concurrently in a Single Instruction Multiple Data (SIMD) manner, where each thread of the Kernel is assigned to one GPU physical core. The behavior of these internal scheduling mechanisms is not well documented by the GPU manufacturers, and despite multiple reverse engineering efforts [19–21], they remain generally unknown to the public.

2.2. Overview on time partitioning

In our proposal, we consider ARINC-like partitioning as the first scheduler layer. This standard specifies that partitions are executed in fixed time windows, following a predefined order inside a cyclic interval called Major Frame (MAF). Applications within each partition are scheduled inside the partition's designated time windows. We will refer to the time consumed by a partition inside the MAF as the Available Utilization (AU), which represents the percentage of CPU utilization consumed by a certain partition. A single MAF can contain more than one time window for the same partition. The number of windows for the partition is fixed. According to [22], increasing the number of time windows within a fixed-length MAF reduces the worst-case response times of applications executed in a partition, due to decreasing the size of the unavailability gaps, i.e., the intervals during which the CPU is unavailable to the application, between the different partition windows. The impact of context switches between partitions is not discussed in this paper. As observed in [22], we considered it a non-critical aspect for the scope of our work.

2.3. Related work

The growing interest in GPU platforms for critical applications can be observed in the number of recent papers published in this field. The work [4] presents an exhaustive review of functional, safety and real-time aspects of GPUs. Additionally, in [5] a review of the use of GPUs for real-time applications is presented, which can be considered as an extension of the previously mentioned work, focusing on techniques for optimizing temporal parameters and meeting deadlines.

Concerning the modeling and response-time analysis of GPU-based heterogeneous platforms, several studies [23–25] have arisen in response to the industrial challenge proposed in [1]. The challenge requested proposals for computing response times and mapping tasks in an autonomous driving application accelerated by GPUs.

The works [23,24] presented similar approaches to the challenge. In [23], the authors proposed a solution based on response-time analysis considering fully-preemptive tasks for both CPU and GPU tasks. CPU tasks are scheduled under rate monotonic scheduling while GPU tasks are scheduled under weighted round robin (WRR). The solution takes into account CPU–GPU memory contention models, memory access and different data offloading mechanism. The authors in [24] presented an approach to analyze end-to-end latencies for GPU-based applications assuming that all tasks, including CPU and GPU tasks, have the same read-execute-write structure and are fully-preemptive. CPU tasks follow a fixed-priority scheduling policy where they are statically assigned to cores, while the GPU uses a WRR policy with fixed window lengths. Both works also present optimization techniques based on genetic algorithms. These execution models enable the computation of GPU's response times. Neither temporal isolation nor Directed Acyclic Graphs (DAGs) are considered in these works.

The work [25] proposed a response-time analysis and an optimization approach for applications running on heterogeneous platforms. This paper also considers a fixed-priority scheduling policy with static allocation for CPU tasks, while GPU tasks are modeled as self-suspending tasks. The work uses jitter-based analysis to calculate end-to-end response times. This work also includes a response-time analysis for GPU operations based on two scheduling policies: WRR and non-preemptive fixed-priority. The proposed optimization is based on a Mixed-Integer Linear Programming (MILP) technique that provides

a task-to-core allocation and priority assignment to reduce end-to-end latencies and ensure schedulability. However, this work does not incorporate temporal isolation and the possibility to employ DAGs.

Besides the responses to the previously mentioned industrial challenge, a different approach considering a DAG model executing on heterogeneous platforms is presented in [26]. This work includes a DAG transformation algorithm that enhances the response-time analysis by reducing CPU blocking times when offloading workload on the GPU. In [27], the authors present another approach for modeling DAGs in heterogeneous platforms. They consider that tasks are statically assigned to available processing resources. The presented model supports using different scheduling policies for each processing resource. In addition to the model, a response-time analysis is presented, as well as multiple algorithms for task allocation. As the rest of the aforementioned works, time partitioning is not considered.

Our work contributes to the analysis of complex distributed systems executed on heterogeneous platforms, enabling the incorporation of time partitioning. The presented proposal is supported by pre-existing analysis techniques included in MAST. Thus, it potentially supports DAGs (as the analysis techniques do), although this issue is out of the scope of the paper and its specific study will not be explicitly addressed.

3. Motivating industrial use case

In the context of smart mobility, the demand in railway systems for increasing the Grade of Automation (GoA) of a train requires higher computational loads, as well as incorporating new functionalities into the system. The European Rail Traffic Management System (ERTMS) [28] is the European standard that aims to ensure interoperability among national railway networks while improving the safety of train traffic. ERTMS defines the Automatic Train Operation (ATO) subsystem [29], which provides non-safety functions to the train, while the safety functions are ensured by the European Train Control System (ETCS). The set of functionalities incorporated in the ATO includes actions that traditionally were achieved manually by the train driver, such as achieving a specific speed profile, traction and brake control, and door opening and closing, among others.

The industrial use case addressed in this work is based on a prototype developed in CAF Signalling [30], called Profile Optimizer (PO). The PO is an application tasked with the computation of the operational speed profile, an ATO functionality that determines the optimal speed profile for a minimum energy consumption while simultaneously maximizing comfort and punctuality. This represents an optimization problem that is solved by a genetic algorithm. The algorithm takes an initial population of potential speed profile solutions, and subjects them in an iterative way to selection, crossover, and mutation operations until the most suitable profile is obtained.

A high-level diagram of the Profile Optimizer algorithm is shown in Fig. 1. First, the PO loads the genetic algorithm configuration parameters (population size, crossover factor, etc.), train information, and track data into the memory. It then generates random solutions until an initial population is completed. After processing this initial population, the CPU starts executing the genetic algorithm's iterative cycle. This cycle is executed in a loop until any of the following conditions is fulfilled: (1) the algorithm achieved a suitable solution, or (2) it has reached a maximum number of iterations. At the beginning of each iteration, the algorithm prepares the necessary data for calculating the new generation. For every individual in the population, the algorithm calculates the new individual (solution), obtains the solution's complete speed profile, evaluates the new solution, and calculates the energy consumption. It computes this sequentially for each individual in the population. When the new population is obtained, the algorithm mixes it with the previous population and selects the best individuals for the next iteration. It also checks if an improvement has been made and if any of the loop conditions have been fulfilled, in which case the PO ends its execution. Otherwise, it continues with the next iteration.

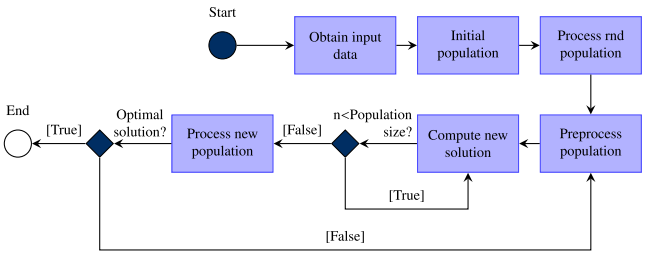


Fig. 1. Profile Optimizer algorithm's diagram.

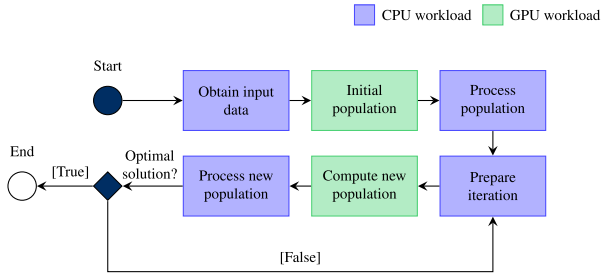


Fig. 2. Profile Optimizer algorithm executed in a GPU using Zero-Copy memory management.

The pre-existing implementation of the PO algorithm relies on a sequential architecture designed for CPUs. Upon a deeper inspection of the algorithm, it was determined that the individuals in a population are completely independent from each other, and thus can be processed independently and concurrently, for example on multiple cores or accelerated on GPUs.

We propose adapting the PO application so that the parallelizable parts of the application could run on GPU devices, enabling the concurrent computation of large populations. For this purpose, we modified the PO algorithm to enable parallel processing on GPUs using the CUDA programming model [31]. We identified two specific sections within the original code that can be parallelized: the initial population computation and the generation of the new population inside the iterative cycle of the genetic algorithm. In the modified PO version, the operations for each individual in both sections are computed concurrently on the GPU, using one GPU thread for each individual.

Two different sub-variants of the GPU-enabled PO algorithm have been implemented, that differ on how the data transfer between the CPU and GPU is performed: data transfer with (1) Zero-Copy, or with (2) Standard Copy.

The PO algorithm implemented with Zero-Copy is shown in Fig. 2. The sections executed on the CPU are shown in blue, while the ones executed on the GPU are shown in green. In this implementation, both the GPU and CPU directly operate on data that is stored in shared memory, and thus there is no need for any dedicated copy operation.

Fig. 3 illustrates the PO algorithm with Standard Copy. Here, the most substantial change is that in the absence of a common memory space, it is necessary to make explicit data copies between the CPU and GPU private memory spaces. These data transfers add extra operations to the execution of the algorithm: the CPU requests the GPU driver to perform the data copy, and then the data copy itself is performed by the GPU's Copy Engine. Every time a Kernel is launched, at least one data transfer must be made to provide the necessary input data to the Kernel, and another data transfer to get the results back from the GPU. Specifically, in the PO algorithm, the Kernel in charge of generating the initial random population requires as input the algorithm configuration parameters, and generates the initial random population as output. Then, the Kernel responsible for calculating the population

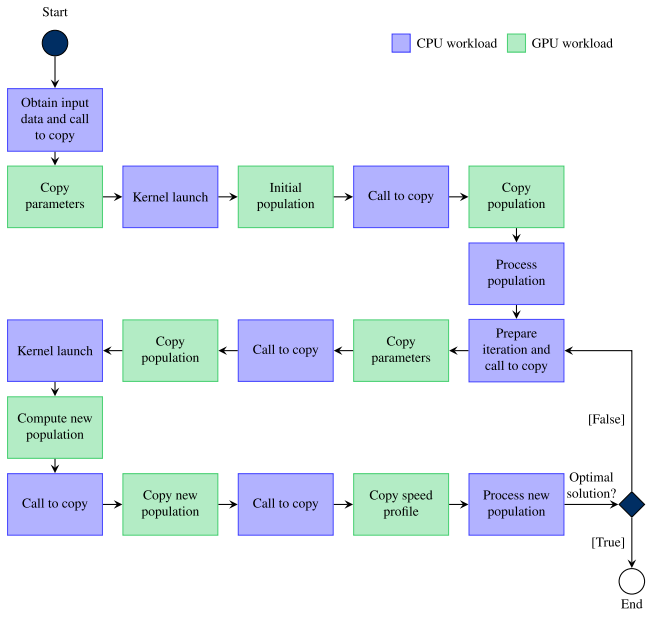


Fig. 3. Profile Optimizer algorithm executed in a GPU using Standard Copy memory management.

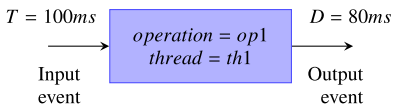


Fig. 4. A simple e2e flow triggered by a periodic event.

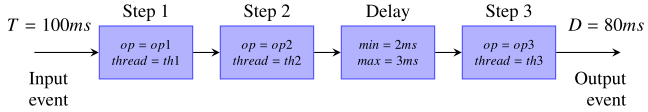


Fig. 5. Linear e2e flow with a delay element.

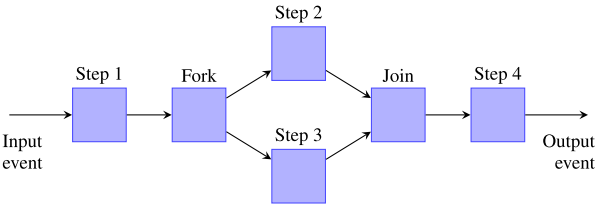


Fig. 6. Graph representing a multipath e2e flow with fork and join elements.

and solving the speed profile requires as input the previous population in addition to the algorithm configuration parameters, and produces the new population and the speed profile of each individual.

This section has provided the adaptations of the PO algorithm to use GPUs, with Standard Copy and Zero-Copy memory management schemes. Section 6 will describe how these specific adaptations can be modeled with MAST in the context of a time-partitioned real-time system.

4. Modeling and analysis

4.1. MAST model overview

As mentioned before, MAST (Modeling and Analysis Suite for real-Time applications) [13] is a modeling technique together with a set of analysis tools. The MAST model allows describing the timing properties of a real-time system, focusing on the relevant aspects for the analysis of the timing behavior. The model describes the software architecture of the application, as well as its deployment on the hardware platform. It supports complex distributed systems.

MAST also includes schedulability analysis tools that can work on the model to calculate worst-case response times that can be compared to the deadlines to check the system schedulability. The tools can make an automatic calculation of blocking times due to the use of mutually exclusive resources. They can also perform an automatic assignment of scheduling parameters such as thread priorities and ceilings for mutual exclusion resources. In addition, the tools are capable of performing sensitivity analysis to find out how close or far a system is from being schedulable, thus providing valuable information to the system developer, that can help in identifying bottlenecks. There are tools available for different analysis techniques (see Section 4.2).

When modeling a real-time system with MAST, the application is described as a number of responses to events, called end-to-end flows (e2e flows). The events that trigger e2e flows can be generated inside the system, for instance from the system clock in case of periodic events, or may come from external hardware through an interrupt. The simplest e2e flow just contains one step representing the execution of an operation (a piece of sequential software) by a thread scheduled by the system scheduler. An operation is characterized by its worst-case execution time (WCET). It also allows specifying a best-case execution time (BCET), which enables analysis techniques to obtain tighter bounds of worst-case response times [32]. The scheduler obeys to some scheduling policy. Fig. 4 represents a simple e2e flow with just one step. This step is executed each time a periodic workload event is generated (period T=100 ms in this case). As a result of its execution, the step generates an output internal event, on which a deadline can be imposed (D=80 ms in this case).

The supported scheduling policies are mainly fixed priorities and EDF. In this paper, we focus on fixed priorities. This implies that the threads have a priority assigned offline as a scheduling parameter. It should be noted that, although the MAST-2 model supports partitioning, the used analysis and priority assignment tools do not work with this version of the model. However, techniques addressing these limitations have been proposed [33].

The MAST model also supports complex e2e flows in which each step can trigger the execution of another step. It is also possible to specify delays between a pair of such steps, as is shown in Fig. 5, where the execution of step 3 is initiated after a time interval between 2 and 3 ms elapses since the finalization of step 2.

The *delay* element specifies a time interval relative to the finalization of the previous step. However, it is also possible to specify a similar *offset* element, which represents a waiting time relative to the arrival of the external periodic event. Maximum and minimum values can be specified for both model elements.

The MAST model, and some of its analysis tools, also supports multipath e2e flows, which are built using *fork*, *branch*, *join* and *merge* elements. Fig. 6 shows an e2e flow with a fork and a join element. At the fork element, when the input event is received it is replicated through each of its outputs. The join element waits for the arrival of each of its input events, before it generates its output event.

4.2. Response-time analysis techniques

MAST includes different schedulability analysis tools that implement various analysis techniques for fixed priority scheduling that can handle distributed systems with arbitrary deadlines and linear e2e flows. All these techniques are approximate, and provide upper bounds on the response times, to various degrees of precision. The first such technique, called the holistic analysis [34], is available as a reference. However, it is quite pessimistic because it considers that all the steps in

an e2e flow are independent, leading to scenarios that are not possible in practice, given that the steps are executed one after the other.

The offset-based techniques improve the holistic analysis by introducing offsets for the different steps in linear e2e flows. These offsets are time intervals measured from the arrival of the period and represent a minimum activation time for the steps. This activation time allows the e2e flow to be analyzed as a whole. A difficulty appears when the offset-based analysis is applied, because it is not known how to create the combination of steps of the different e2e flows initiating the worst possible case for the analysis. Different solutions apply for solving this difficulty:

- Offset-Based Brute Force [35]: This technique applies the offset-based analysis exploring all the possible combinations of steps initiating the analysis case. Since the number of combinations is combinatorially explosive, this technique can only be applied to small systems.
- Offset-Based Approximate [36]: This technique makes upper-bound approximations to avoid covering all the possible combinations of steps initiating the analysis case. The results are therefore not as precise as with the brute force technique, but the technique can be applied to large systems.
- Offset-Based Slanted [37]: This technique makes a similar approximation as the previous one, but using a slightly improved analysis. It can also be applied to large systems.

The techniques mentioned above do not take into account the precedence relations among the steps in an e2e flow. A more precise analysis can be obtained with the Offset-Based with Precedences technique [38], which further optimizes the Offset-Based Approximate analysis to take into account these precedence relations.

All the aforementioned techniques apply to linear e2e flows. The work in [17] has extended all but the Offset-Based with Precedences technique to support multipath flows with fork, join and merge elements.

Although the MAST-1 model does not support partitioned scheduling, the MAST-2 version [15] supports the specification of time partitioning. In any case, the work in [16] shows how to analyze time-partitioned systems with the available analysis techniques implemented in MAST-1. When analyzing a particular partition, the time windows are taken into account by introducing a highest-priority flow that contains tasks that model the complement of the time windows, i.e., the time intervals during which the processing resource is not available. This is called the *unavailability flow*. In addition, the work in [16] describes improvements to the Offset-Based Approximate technique that optimize the treatment of this unavailable flow. It also describes how to analyze a mixture of tasks that are synchronized, or not, with the major frame (MAF). Finally, the work in [17] extends this work to multipath flows.

4.3. Modeling systems with GPUs

MAST does not directly support GPUs. It is possible to model coprocessors as additional processors and allocate threads to them, allowing us to calculate the contention caused by the different threads competing for the coprocessor. However, the GPU is not a conventional coprocessor because it contains multiple processing resources that are capable of executing their allocated work in parallel. A detailed analysis of this contention is extremely difficult, given that we usually do not have accurate models of the internal structure and scheduling policies on the GPUs. The detailed analysis would have to include the effects of the interactions between the GPU and the regular CPUs caused by the use of shared memory. In this work, we use time partitioning to ease the calculation of these interactions, which can only occur at specific time intervals. Given this property, these interactions can be included in the measured response times.

Given the framework described above, we analyze the execution on the GPUs as a black box by working with estimations on the overall response times of the GPUs when working on a particular task. We will model these response times through delay elements with a minimum and maximum interval. If the GPU's work is synchronized with the e2e flow period it would also be possible to model the GPU's response times using offset elements. Fig. 5 shows an example of a linear e2e flow with a delay element which could represent the execution of a particular workload in a GPU.

4.4. Controlling GPU access through time partitioning

While GPUs support the execution of multiple Kernels at the same time, their poorly documented internal scheduling policies can lead to difficult to predict behavior under real-time scenarios. Thus, external control mechanisms to avoid concurrent Kernels are necessary to achieve a predictable execution suitable for such scenarios. In this paper, we propose employing time-partitioning as a way to control the workload sent to the GPU. The main idea of this proposal is that partition plans can be designed in order to guarantee, by construction, that at most only one Kernel will be executing in the GPU at any given time.

To illustrate our proposal, Fig. 7 represents the workload of a simple system, composed of two e2e flows, each accessing the same GPU once per period. Each e2e flow is mapped to a different partition (P1 and P2). Fig. 8 depicts two possible time partition plans that guarantee a predictable access to the GPU by preventing Kernel co-existence, with MAF equal to 20 and 40 ms respectively. The blue areas represent the partition windows for P1 and P2, and the green areas depict the GPU time slots in which a Kernel from the respective partition can be executed. The gray areas represent CPU time slots available for other partitions. Notice that the proposed time partition plans do not follow a typical design used in high-integrity systems, where functions are allocated in partition windows by taking into account the least common multiple (LCM) and the greatest common divisor (GCD) of the activation periods. Our modeling and analysis proposal does not depend on any particular plan configuration. Its goal is to address general mixed-criticality systems, where some partitions may have multiple tasks scheduled by fixed priorities that may access the GPU. Thus, plan configuration is considered an optimization issue, and in this simple example, the proposed plans are only focused on meeting the GPU execution requirements.

The main characteristic of these plans is that the minimum time span between slots of different partitions is long enough to fit the execution of the GPU Kernel. For instance, in Plan 1 (Fig. 8(a)), the minimum time span after a window of P1 and another of P2 is 6 ms, which is long enough to fit the GPU Kernel mapped to P1 in the worst case (6 ms). Similarly, the minimum distance between P2 and a subsequent window of P1 is 10 ms, which fits the Kernel mapped to P2 (10 ms). Consequently, Plan 1 guarantees, by construction, that the Kernels of different e2e flows are not going to execute concurrently in the GPU. A similar rationale is used to build Plan 2 for a longer MAF of 40 ms (Fig. 8(b)). In this case, a second window for P1 is added. Nevertheless, the minimum time span between windows of different partitions still allows for the execution of the GPU Kernels with no risk of overlapping.

It should be noted that through time partitioning, an implicit reservation of GPU usage is established for each partition. Thus, Plan 1 has 100% of the GPU reserved, while Plan 2 still has 30% (12/40) available to other partitions that may require the GPU. In this paper, we focus on providing and validating the mathematical framework to analyze systems with GPU employing time partitioning as a technique to achieve predictability. It is also important to highlight that both Plan 1 and Plan 2 remain valid only if the estimations of the worst-case execution times of the GPU Kernels are accurate. Should these estimations be exceeded at runtime, the partitions will fail to ensure

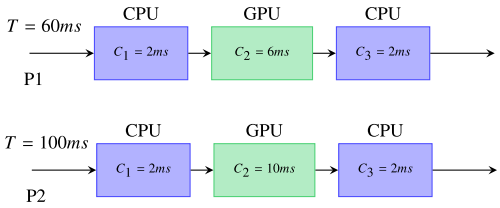
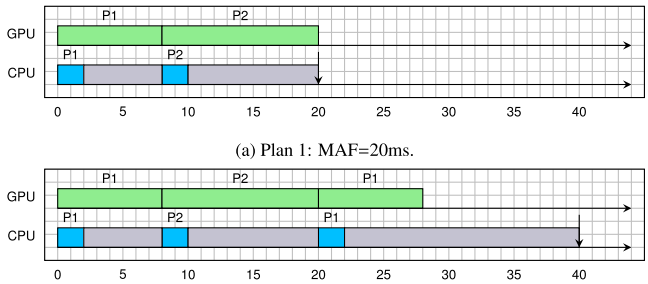


Fig. 7. Simple example with two e2e flows using the GPU.



(a) Plan 1: MAF=20ms.

(b) Plan 2: MAF=40ms.

Fig. 8. Examples of time partition plans that ensure non-concurrent access to the GPU by construction, with (a) MAF=20 ms, and (b) MAF=40 ms.

that the Kernels do not execute concurrently. Therefore, an accurate estimation of the WCETs in the GPU is crucial for a safe deployment of the time partitions. Both the actual configuration and optimization of the time partition plans and WCETs estimations are out of the scope of this work.

4.5. Limitations and workarounds

Taking into account the above considerations, using delays to model the GPU workload works well for e2e flows with deadlines within the periods. In this scenario, it may even be possible to use the same thread for all CPU steps in the e2e flow. However, if the deadline exceeds the period, only e2e flows where each step is executed by a different thread can be considered, because the original MAST-1 model on which the used analysis techniques are based handles steps as if they were independent. The analysis or the interference between steps of different e2e flow instances executed by the same thread is not supported in MAST-1.

Exceeding the activation period may result in overlapping among steps from different activations of the same e2e flow. Under these situations, time partitioning can be unable to avoid concurrent accesses to the GPU, and additional GPU serialization mechanisms should be considered, such as the FIFO queue proposed in [11]. This additional serialization mechanism can be analyzed by modeling the GPU as an independent processing resource (instead of a delay), where steps are dispatched using a FIFO scheduling policy. For this purpose, it is necessary to model the GPU in MAST as a processing resource based on fixed priorities, where each thread assigned to the GPU schedules steps at the same priority. This approach emulates the FIFO policy. In the case of assigning more than one thread (e.g., for different e2e flows located in different partitions), isolation between these activities should be ensured through partitioning. In this work, we have focused on analyzing the effects of time partitioning, and thus we have limited the evaluation to e2e flows where the response times do not exceed the activation periods.

The decision to model the GPU as a delay element instead of an independent processing resource resides mainly on modeling constraints. Modeling the GPU as a processing resource would require incorporating several artificial modeling elements (such as processing

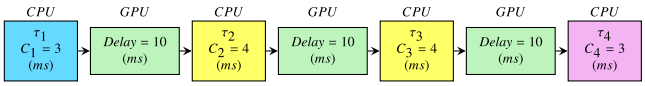


Fig. 9. Simple example of a GPU-based application with an e2e flow.

Table 1
Description of different plans for time partitioning (times in ms).

	Plan 1	Plan 2	Plan 3	Plan 4	Plan 5
Partition window	12	6	2	1	0.5
MAF	30	15	5	2.5	1.25

resources, schedulers, threads or steps) for the GPU. Moreover, the analysis execution process also becomes more efficient as it does not have to analyze additional steps. Therefore, using a delay element offers a more simple and easier to analyze alternative.

Finally, this work is focused on studying a use case based on a linear e2e flow, but the proposed methodology could also be potentially valid for analyzing DAGs. To achieve this, a proper design of the partition windows would be necessary in order to isolate the GPU workload from the different branches of a same DAG, in addition to using the appropriate analysis techniques designed for multipath. This study is left for future work.

5. Characterization of response-time analysis

This section presents a characterization of the applicability of response-time analysis to applications using GPUs which have been modeled with the method proposed in Section 4. For this purpose, we design the example of a simple application shown in Fig. 9, which is simple enough to be manually analyzed, while keeping an architecture that is relevant and has similar elements as the industrial example shown in Section 3.

The example consists of an e2e flow composed of alternating computations in CPU and GPU. It contains four steps τ_i executed on a CPU and characterized by their worst-case execution times C_i, as shown in Fig. 9. Similarly, there is a GPU operation between each step, which will be modeled as a delay as described in Section 4.3. In this simple example and for the sake of simplicity, we assume that execution time and delay values are fixed, i.e., their minimum values are equal to their maximum values.

The proposed e2e flow can represent, for example, a simplified version of the PO algorithm of the industrial use case, with three iterations in the GPU, or one iteration with two memory copies. Step τ_1 in the e2e flow, colored in blue, could represent the initial processing of the algorithm and the launch of the first GPU operation. GPU operations are represented in green indicating their maximum interval delay. In yellow, steps τ_2 and τ_3 could represent the processing of the results of a GPU operation and launching of the next one. Finally, in magenta, step τ_4 processes the last GPU operation and terminates the e2e flow.

The objective of this section is to compare the worst-case response times (WCRT) of the e2e flow obtained by analytical means, with the exact values obtained by inspection. We consider 5 different time partitioning plans, defined in Table 1. We want to check how the accuracy of the proposed methodology depends on the unavailability gap between two partition windows. Accordingly, partitions with a single time window and the same available utilization are proposed while the MAF is varied.

Figs. 10 to 14 depict the worst-case scenarios found for the e2e flow, for each plan defined in Table 1. Each diagram contains two time axis: the first one represents the execution on the GPU, and the other one is the execution in the partition windows on the CPU. The steps within the e2e flow are represented according to the color scheme shown in Fig. 9. The gray space represents the idle time inside the partition window while the blank parts represent the unavailable gaps.

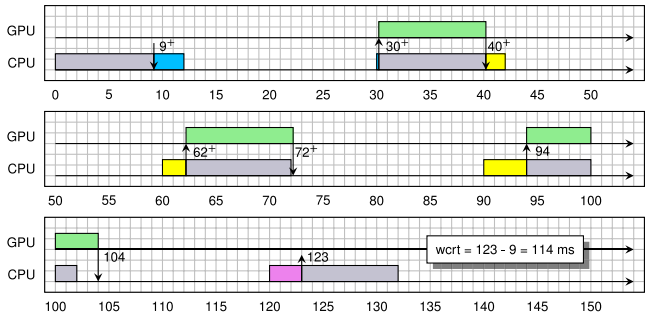


Fig. 10. Worst-case scenario for the simple example over Plan 1.

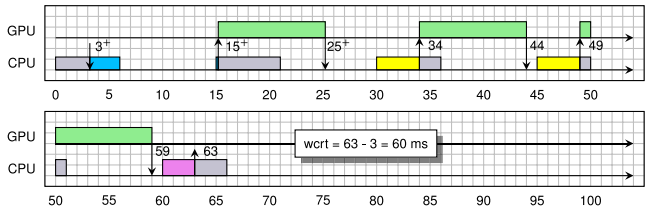


Fig. 11. Worst-case scenario for the simple example over Plan 2.

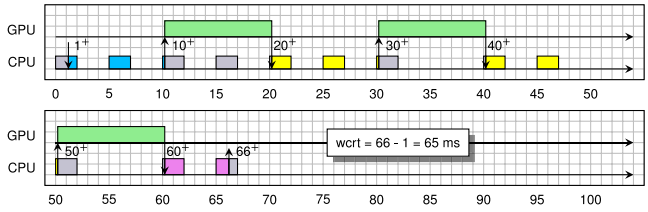


Fig. 12. Worst-case scenario for the simple example over Plan 3.

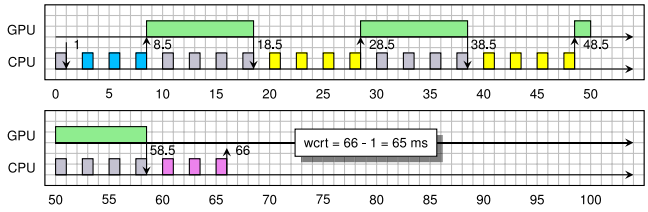


Fig. 13. Worst-case scenario for the simple example over Plan 4.

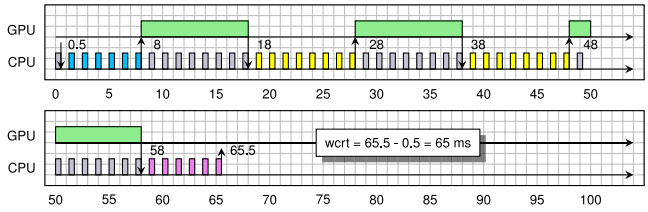


Fig. 14. Worst-case scenario for the simple example over Plan 5.

For each plan, we searched for the release instant of the e2e flow that produces its worst-case response time. For this purpose, the sign (+) is used to indicate a slightly delayed activation after a specific instant. For example, in Fig. 10, the e2e flow is activated at $t = 9^+$, indicating that τ_1 is released a very small time after $t = 9$, causing the start of the execution of step τ_2 to be delayed until the following temporal window, at $t = 40^+$.

Table 2 collects the observed WCRTs of the e2e flow for each of the plans. Additionally, the table also includes the WCRTs analytically computed by the Holistic and Offset-Based with Precedences response-time analysis techniques. From these results we can clearly note the effect of the unavailability gap (i.e, the MAF length for this case) on the WCRTs, for both observed and analytically obtained results: increasing the MAF length tends to enlarge the response times. This is expected, as higher MAFs also introduce longer unavailability time windows in which the partition is not permitted to execute, and thus more opportunities to further delaying the execution of the steps in the worst-case scenarios.

Table 2
Comparison between analysis in MAST and observed behavior for different plans (times in ms).

	Plan 1	Plan 2	Plan 3	Plan 4	Plan 5
Holistic	155	137	122	117.5	117.5
Offset W Pr	116	80	68	65	65
Observed	114	60	65	65	65

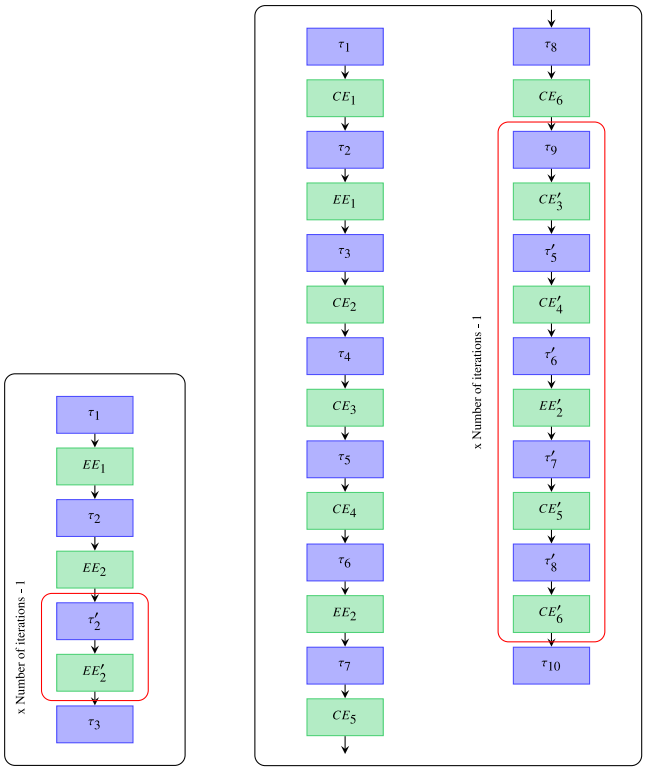
The results in Table 2 also show that reducing the MAF length tends to produce less pessimistic analytical WCRTs, that is, analytical WCRTs that are closer to the observed exact values. For instance, for Plan 4 and Plan 5, which have the lowest MAF values, the Offset-Based with Precedences analysis technique was able to match the observed WCRTs. We can also identify that, as expected, the Holistic analysis is quite more pessimist than the Offset-Based with Precedences analysis. Finally, it can also be observed that Plan 2 would be the best one, as it has the lowest observed WCRT, but it is Plan 3 onward where the best trade-off between the exact and the analytical values is achieved.

6. Modeling the profile optimizer

The previous section illustrated how to apply our methodology to a simple example that could be manually analyzed. In this section, we will apply this methodology to a realistic industrial use case, the Profile Optimizer (PO), which was presented in Section 3. Two variants of the PO algorithm for GPUs were proposed, taking into account how memory is managed: Zero-Copy (ZC) and Standard Copy (SC). The high-level description of the PO algorithm with Zero-Copy (PO-ZC) is depicted in Fig. 2. The main characteristic of this variant is that the CPU and GPU share a common memory space, thus no explicit memory copies are needed to transfer data. Additionally, the high-level description of the PO algorithm with Standard Copy (PO-SC) is depicted in Fig. 3. Here, CPU and GPU operate on private and independent memory spaces, and need memory copy operations before and after every GPU computation is performed.

The first challenge to tackle when modeling the PO algorithm is its iterative nature, with an unknown number of iterations. Thus, the first step to model the PO algorithm with MAST is its transformation into a linear e2e flow, considering a fixed maximum number of iterations. From an empirical evaluation of the PO algorithm, we have determined that it generally requires between 15 and 30 iterations to reach a suitable solution. We exploit this knowledge to linearize the PO algorithm by assuming that this fixed number can be set, and then unroll the loop accordingly. We define this maximum number of iterations as a system parameter to be studied. Functionally, the more iterations, the more accurate the result is. However, there is a number of iterations from which the results converge and no significant further improvement is reported.

In the linearization process of the PO algorithm, contiguous CPU sections have been grouped together to form a single step. In this way, steps are always interleaved with GPU operations. When measuring the execution times for each step (see Section 7.2), we have observed that the execution times in the first iteration are significantly higher than in the rest of iterations. This is due to a warm-up on the GPU that needs some initialization processes, e.g., loading the Kernel in the GPU or configuring memory spaces. To reduce the pessimism of



(a) PO-ZC model. (b) PO-SC model.

Fig. 15. Model for PO's linear e2e flow, with Zero-Copy and Standard Copy memory managements.

over-provisioning the computing resources, the WCETs (Worst-Case Execution Times) assigned to the first iteration include these warm-up overheads, while subsequent iterations are assigned the WCETs observed after the warm-up process has completed.

Applying these considerations, the linear e2e flows corresponding to the MAST models for PO-ZC and PO-SC can be built as shown in Figs. 15(a) and 15(b) respectively. Elements shaded in blue and labeled as τ_i represent steps, i.e. sections of code executed on the CPU. On the other hand, elements in green represent sections executed on the GPU, which are modeled as delays and can be of two sub-types: EE_i (Execution Engine) represents the computations performed by the GPU, and CE_i (Copy Engine) models the memory copy operations needed in the Standard Copy memory management scheme. Additionally, the elements encircled in a red box represent one iteration loop (except the first one), which should be repeated to complete the maximum number set. These elements are annotated with a prime symbol. In summary, the result of modeling the PO algorithm is a linear e2e flow (one for PO-ZC, another one for PO-SC), where the number of steps and delays is determined by the number of iterations.

As mentioned in Section 4.2, the MAST model (version 1) does not support partitioned scheduling, so the steps of the e2e flows built for the PO algorithm are executed on a CPU modeled as a regular processor, which is scheduled by a fixed-priority preemptive scheduling policy. The real industrial application is executed by a single thread with a single priority, but steps in the e2e flow are assigned a priority in a descending order, with the first step in the e2e flow having the highest priority. This assignment, for modeling purpose only, allows the response-time analysis to obtain more accurate results [33], and can support the execution of other tasks in the same partition by assigning priority bands. For example, if an e2e flow with these characteristics has up to 100 steps, we can model real priorities by multiplying them by 100, so a band of 100 different priorities will be available for the e2e flow. The unavailability flow is also modeled as described in Section 4.2.

7. Evaluation

In this section, we evaluate the proposed methodology for modeling GPU-based real-time applications. For this purpose, different configurations of the Profile Optimizer (PO) industrial application, described in Section 3, are evaluated both analytically and via measurements on a testing platform.

This section is organized as follows. First, Section 7.1 describes the hardware and software platform in which the evaluation is performed. Then, Section 7.2 presents a characterization of the WCETs of the industrial use case via extensive measurements in a controlled and isolated environment. Next, in Section 7.3, the proposed modeling and analysis methodology is put to test, by analytically obtaining WCRTs of the PO algorithm modeled as described in Section 6. In this study, different configurations of the PO algorithm and response-time analysis techniques are tested. In Section 7.4, the full PO algorithm with emulated time partitioning is deployed on the testing platform, and the response times are measured for different configurations. Finally, Section 7.5 provides a comparison between the analytical WCRTs and the response times measured on the testing platform.

7.1. Test platform

We have used an NVIDIA Jetson AGX Xavier [39] as our test platform, which includes an 8-core Carmel ARM CPU, and an integrated 512-core NVIDIA Volta GPU. The Volta GPU architecture contains two cache levels: a first level cache per SM (Stream Multiprocessor) of 128 KB, and a shared second level cache of 512 KB. The CPU operates at 2.26 GHz and the GPU at 1.37 GHz. This board represents a low cost and low power platform that is commonly deployed in industrial and safety-critical settings that require the computing benefits of GPUs [40].

As its Operating System, a Yocto Linux [41] image with minimal services necessary for its operation has been built. The configuration is aimed to minimize Linux system's processes that could interfere with the measurements. Additionally, the Linux Kernel has been patched with the PREEMPT_RT patch [42] to enable real-time capabilities.

The frequency scaling functionalities of the test platform were disabled to obtain consistent results. By default, the system is configured to scale the CPU and GPU frequencies when very high energy consumption or device overheating are detected. This is a protection mechanism, as an intense use of the GPU or CPU can rapidly increase the platform's temperature and produce physical damage to the platform. Keeping all the frequencies constant ensures that the system runs under consistent conditions throughout all the measurements. Both the CPU and GPU were set to their standard operating frequencies for the whole evaluation process, with no indication of overheating. The External Memory Controller (EMC) has been also configured to its maximum frequency of 2.13 MHz.

Firstly, a measurement of the overheads that the operating system may incur is made in order to estimate their impact on response time measurements. We performed this evaluation using the *cyclictest* [42] evaluation tool, which measures system's latencies by comparing the difference between a thread's planned activation time and its actual activation time. The results of this evaluation are shown in Fig. 16. A total of 100 million samples were collected for each CPU, providing a comprehensive view of platform's performance. The figure shows a distribution of the obtained results and includes the minimum, maximum and average latency values in microseconds. These results demonstrate very low latencies (average of 6.75 microseconds) that are orders of magnitude lower than the expected execution times, and should not interfere with the validity of the results of the complete evaluation.

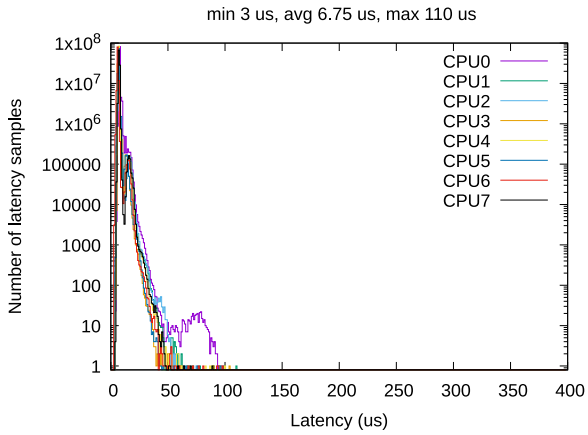


Fig. 16. Test platform's latency benchmark.

Table 3
WCETs and BCETs measured for PO-ZC, in milliseconds.

	C_i	C_i^b
τ_1	34.6	29.9
EE_1	83.5	67.8
τ_2	0.9	0.5
EE_2	135.6	105.6
τ_2'	0.5	0.3
EE_2'	83.7	31.8
τ_3	1	0.1

Table 4
WCETs and BCETs measured for PO-SC, in milliseconds.

	C_i	C_i^b
τ_1	33.6	29.9
CE_1	0.2	0.1
τ_2	0.2	0.2
EE_1	82	69.7
τ_3	0.2	0.1
CE_2	0.2	0.1
τ_4	0.6	0.3
CE_3	0.1	0.1
τ_5	0.1	0.1
CE_4	0.1	0.1
τ_6	0.3	0.3
EE_2	240.6	199.2
τ_7	0.1	0.1
CE_5	0.2	0.2
τ_8	1.6	1.4
CE_6	4.5	3.9
τ_9	1.6	0.4
CE_3'	0.4	0.1
τ_5'	0.1	0.1
CE_4'	0.1	0.1
τ_6'	0.1	0.1
EE_2'	82.6	36
τ_7'	0.1	0.1
CE_5'	0.2	0.1
τ_8'	0.6	0.2
CE_6'	0.7	0.4
τ_{10}	0.9	0.1

7.2. Execution time measurements

In order to perform the response-time analysis, measurements of worst-case execution times (WCETs) and best-case execution times (BCETs) for every task in the system have been done. This section describes the process followed to provide safe estimations of the WCETs and BCETs of the tasks that conform the PO algorithm.

The main challenge of estimating WCETs and BCETs of the PO algorithm lies on the heterogeneity of the computing platform, as some tasks execute on CPU, while others are offloaded into GPU. In this paper, we follow an empirical approach, by executing the tasks on the real testing platform in isolation, thus minimizing possible scheduling interferences.

The internal CPU clock has been used to measure the execution times of the CPU tasks, while to accurately measure the response times on the GPU, a different technique has been employed. In order to avoid synchronization latencies, CUDA provides a functionality called *CUDA events*, which allows capturing timestamps on the GPU by launching an event before and after the execution of a GPU operation [43,44].

Tables 3 and 4 show the worst and best cases for the measured execution times, for PO-ZC and PO-SC respectively. For each measurement the PO algorithm has been run 1000 times. From this results, it can be observed that the GPU operations consume the largest proportion of time. This is the expected result, as the main computational workload is executed on the GPU, while the CPU mainly performs results collection and basic processing. This makes the CPU utilization of the algorithm to be very low. We can also see how the measurements of the first execution of a GPU operation are, in most cases, significantly higher than in the subsequent iterations. This phenomenon also affects the CPU tasks when requesting a computation from the GPU for the first time, impacting the execution time on the CPU. An example of this phenomena can be seen, for example, in Table 3 with the execution of the second Kernel, where in the first iteration (EE_2) the observed WCET is 135.6 ms and in the subsequent iterations (EE_2') it is 83.5 ms.

7.3. Response-time analysis results

In this section, we explore the analytical worst-case response times obtained by different analysis techniques available in MAST for different configurations of the PO algorithm modeled, as described in Section 6.

The analyses executions have been carried out on a Windows computer, with an Intel Core i7 vPro 9th Generation processor, which has a 12-core CPU that operates at a frequency of 3 GHz. Each analysis is single threaded, therefore several analyses where run in parallel to leverage the host's multi-core capabilities. In total, the analytical evaluation carried out required less than one hour of computation time for the PO-ZC and about 21 days for the PO-SC.

To facilitate an extensive evaluation with different configurations of the PO algorithm, a tool to automatically generate models was created. Different number of iterations of the PO algorithm are considered (15, 20, 25 and 30 iterations). Additionally, a MAF with a single partition window is swept from 2.5 to 100 ms with a step of 2.5 ms. Different values of the Available Utilization (AU) are generated (10%, 20%, 40%, and 80%). The execution times of the steps and the delays of the GPU operations in these generated examples are taken from the measurements performed on the testing platform, presented in Section 7.2.

As described in Section 4, MAST implements several techniques that support the response-time analysis of the PO algorithm that we study in this paper, namely: Holistic, Offset-Based Approximate (Offset-Approx), Offset-Based Slanted (Offset-Slanted), Offset-Based with Precedences (Offset-W-Pr), and Offset-Based Brute Force (Offset Brute Force).

We performed an initial evaluation to determine which response-time analysis technique generally obtains the lowest WCRTs, that is, the results with less over-provisioning or less pessimistic. This initial evaluation consists of obtaining the WCRTs of the PO algorithm with Zero Copy (PO-ZC), configured to perform 30 optimization iterations, and 10% of Available Utilization (90% of the CPU time is reserved to other partitions). The MAF is swept from 2.5 to 100 ms with a step of 2.5 ms.

For this particular initial evaluation we have allowed response times to exceed the activation periods (5 s in this evaluation). This allows

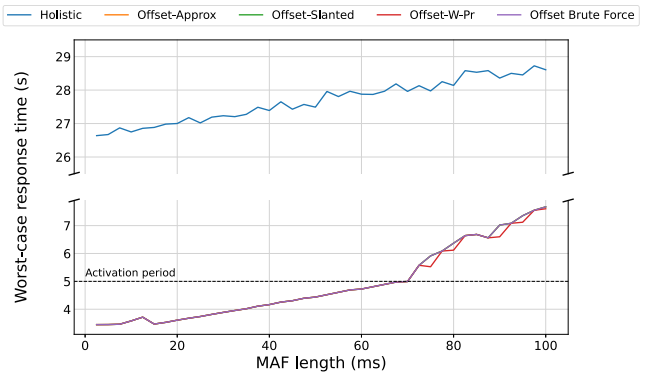


Fig. 17. Comparative of different analysis techniques in MAST for the PO with ZC memory management, 30 iterations and 10% AU, allowing response times higher than the activation period (5 s).

to study the behavior of the analysis techniques under two different situations: (1) when no overlapping occurs (WCRTs of less than 5 s), and (2) overlapping of the same step occurs (WCRTs higher than 5 s). In the latter, the analysis is performed on a model built as indicated in Section 4.5, i.e., considering the GPU as a different processing resource scheduled under a FIFO policy.

The results of this initial evaluation are shown in Fig. 17. It is observed that the Holistic technique presents a very pessimistic approach compared to the other techniques. Although it cannot be precisely appreciated in the figure, the results of Offset-Based Approx, Offset-Based Slanted and Offset-Based Brute Force techniques, overlap throughout the whole MAF range. Overall, the least pessimistic technique according to this study is the Offset-Based with Precedence relations technique, as can be observed particularly when the WCRTs exceed the periods.

In conclusion, if greater precision is required, the most recommended technique is the Offset-Based with Precedence relations. However, since this technique currently only supports linear e2e flows, any of the other three offset-based techniques will be suitable for obtaining a precise response-time estimation for multipath e2e flows (including DAGs).

The results obtained with any of these techniques do not incur in a significant penalty in pessimism compared to Offset-Based with Precedence, since the differences are minimal as shown in the figure, or non-existent if only e2e flows with response times lower than the periods are considered. In any case, the Holistic technique is considered unsuitable due to its high pessimism. Based on this results, and considering that the PO algorithm consists of a linear e2e flow, the Offset-Based with Precedence technique has been chosen for all subsequent experiments.

The analytical WCRTs obtained by the Offset-Based with Precedences technique for PO-ZC and PO-SC are shown in Figs. 18 and 19 respectively. Four diagrams are shown for each memory management scheme, corresponding to 4 different number of optimization iterations in the PO algorithm (15, 20, 25 and 30 iterations). Different values of the AU are represented (10%, 20%, 40%, and 80%), and the MAF is swept from 2.5 to 100 ms with a step of 2.5 ms. For evaluation purposes and in order to meet the requirement that response times cannot exceed the e2e flow's activation period, as described in Section 4.5, a period of 50 s has been set for the PO algorithm.

In both figures, despite some small peaks, the obtained estimated worst-case response times increase linearly with higher MAF lengths. However, a steeper slope can be observed in the results of PO-SC. This is due to the fact that PO-SC has a significantly higher number of steps compared to PO-ZC, due to the copy operations. Therefore, the accumulated pessimism is multiplied, specially for higher MAF lengths (see Section 7.5). Furthermore, in can be observed that the WCRTs increase when the AU decreases. This is an expected result, as a lower

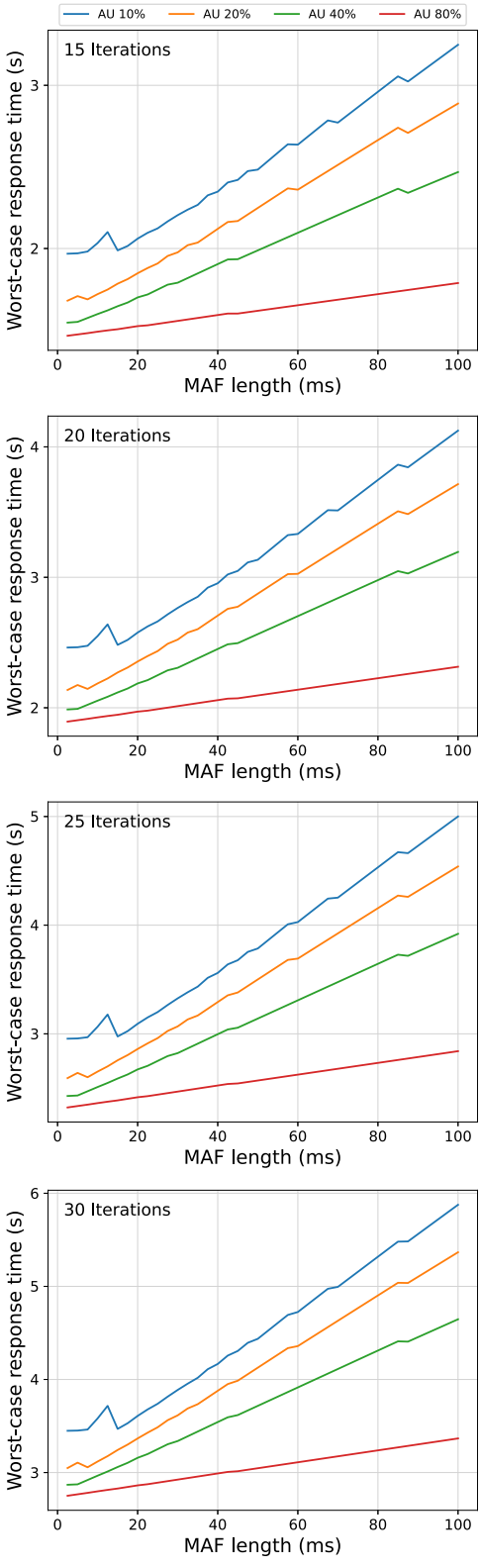


Fig. 18. Analysis results for the PO with ZC memory management.

AU results in a higher unavailable gap suffered by steps in the e2e flow. Additionally, the analysis confirms that there are no cases where the WCRT of any e2e flow exceed its activation period.

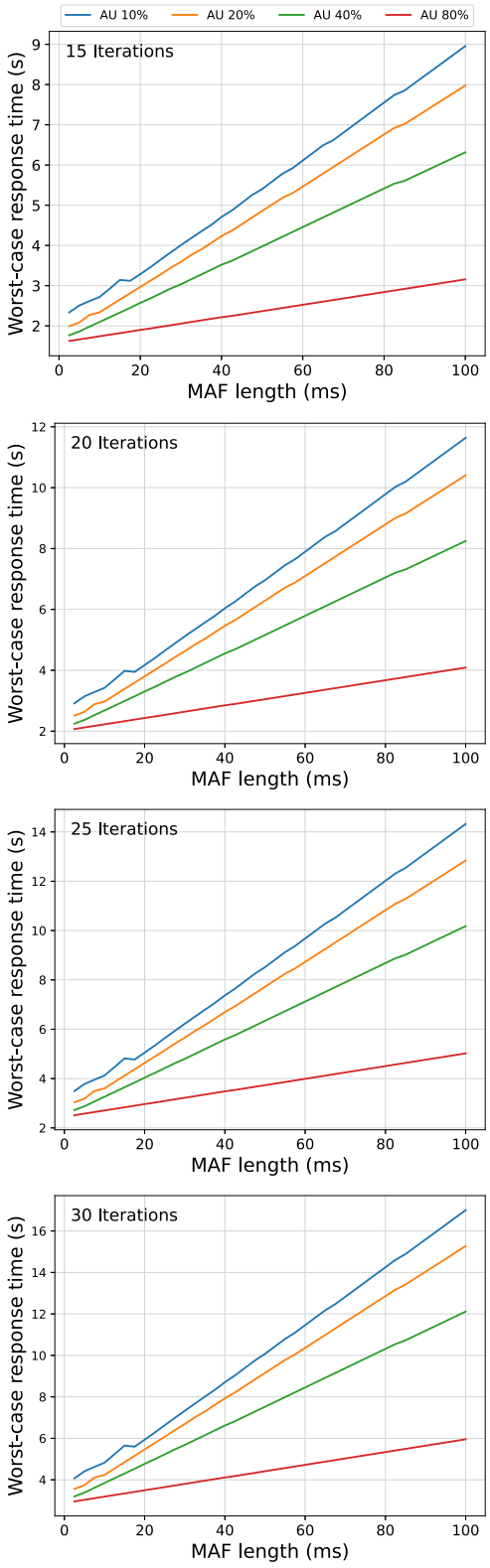


Fig. 19. Analysis results for the PO with SC memory management.

7.4. Measuring response times on the real platform

In this section we present the measured response times of the PO algorithm on the real testing platform, whose characteristics were described in Section 7.1. We test the same configurations as those analyzed in the previous section (except for the MAF lengths): Zero-Copy (PO-ZC) and Standard-Copy (PO-SC) memory managements; Available Utilizations of 10%, 20%, 40% and 80%; 15, 20, 25 and 30 optimization iterations; and MAF lengths of 20, 40, 60, 80 and 100 ms. This makes a total of 160 different test cases. For each specific testing configuration, 1000 measurements of the response times of the PO algorithm were registered. Each set of 1000 measurements takes about one hour to complete for PO-ZC and one and a half hours for PO-SC, due to its longer activation period.

To configure the time partitioning required by our proposal, we had to face the problem of not finding any readily available hypervisor that supported our testing platform. As a solution to this problem, we relied on emulating the time partitioning mechanism with an *ad-hoc* approach, consisting on deploying two periodic threads running in the same CPU: a high priority thread which consumes a certain amount of time, that emulates the unavailability flow, and another lower priority thread that executes the PO algorithm with the defined activation period.

The measured response times are shown in Fig. 20 and Fig. 21 for PO-ZC and PO-SC respectively. These results are represented by box and whisker diagrams. Each box encompasses 90% of the measurements, while the upper and lower whiskers represent the maximum and minimum measured response times, respectively. The horizontal line inside each box indicates the average of the measurements. For each MAF length value (x axis), 4 boxes are shown, one for each evaluated AU.

For both PO-ZC and PO-SC, the general trend of the measured response times is to increase as the MAF increases, as was predicted in Section 5. However, at a MAF length of 80 ms, there is a drop in the measurements. An explanation for this is that at this MAF length, the response time of the Kernel on the GPU (which represents the majority of the workload of the PO), becomes synchronized with the activation of the partition window. The execution on the GPU in almost all cases, at least in the ones that we have measured, terminates before the next activation of the partition window executing one iteration of the algorithm per MAF activation. It is never activated in the same partition window (except for AU of 80%). Additionally, this synchronization effect also produces a remarkable decrease in the variability of the results.

When comparing the different AU tested, the results show that a higher AU gives lower response times. In some cases, especially for MAF length of 80 ms and beyond, the difference of the response-times between 10, 20 and 40% of AU is minimal. This is because the CPU utilization inside the partition is very low compared to the workload on the GPU and to the window size. As the delay caused by the GPU workload is almost always bigger than the window size at these AU percentages, the next step's activation occurs in the next partition window. For an AU of 80%, the partition window takes most of the MAF length, resulting on measured response times that are barely impacted by the MAF length, with just a slight increase reported for MAF=100 ms. At this high AU value of 80%, there is a lower probability that the activation of a step arrives in an unavailability gap. Furthermore, this gap has a much smaller length, which prevents any step from being blocked for a long period.

Comparing the partitioning plans with respect to different number of iterations, similar results are obtained across all graphs. The primary distinction is that, with the addition of more iterations, the measured response times increase proportionally.

With the exception of a few minor variations, there is no substantial difference between the results obtained between PO-ZC and PO-SC memory management. This can be explained with the occurrence of the following phenomena. The works [45,46] have shown that using SC achieves better performance than ZC, because with ZC the GPU must access the global memory every time it needs data. On the other hand, using SC, the GPU's data can be located in its local cache levels, allowing faster data accesses. In the case of the PO algorithm, different

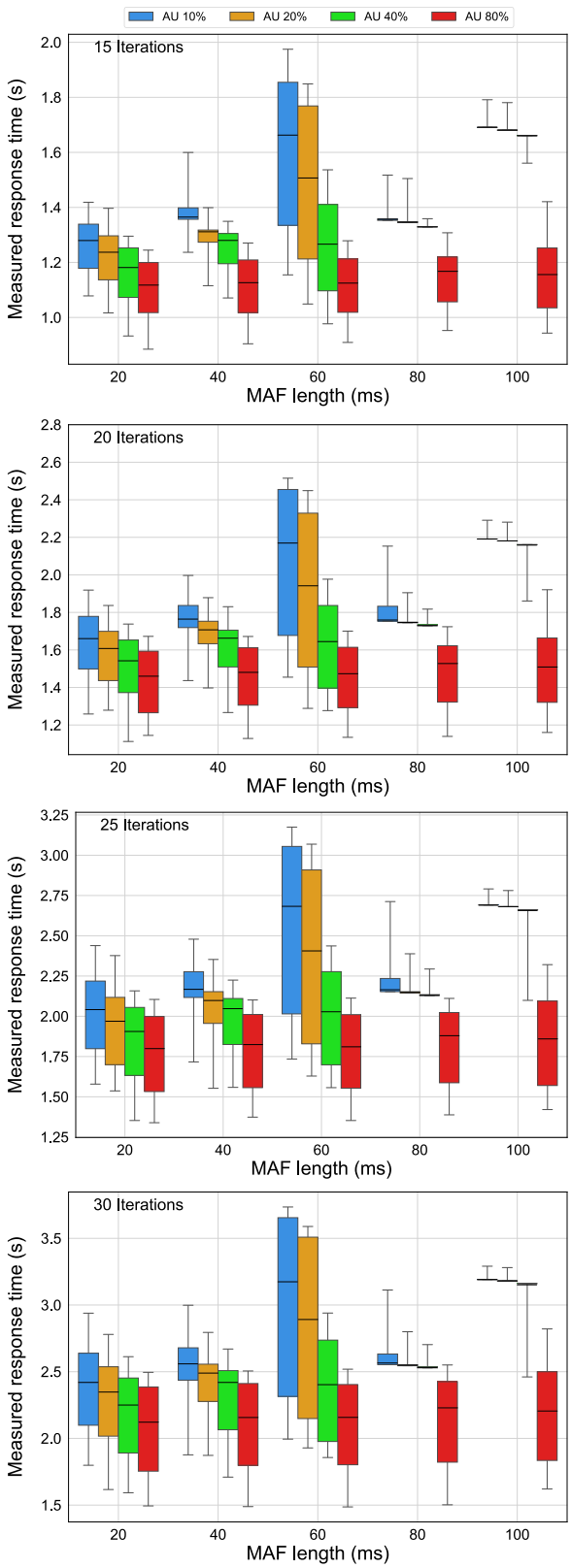


Fig. 20. Measured response times for the PO with ZC memory management.

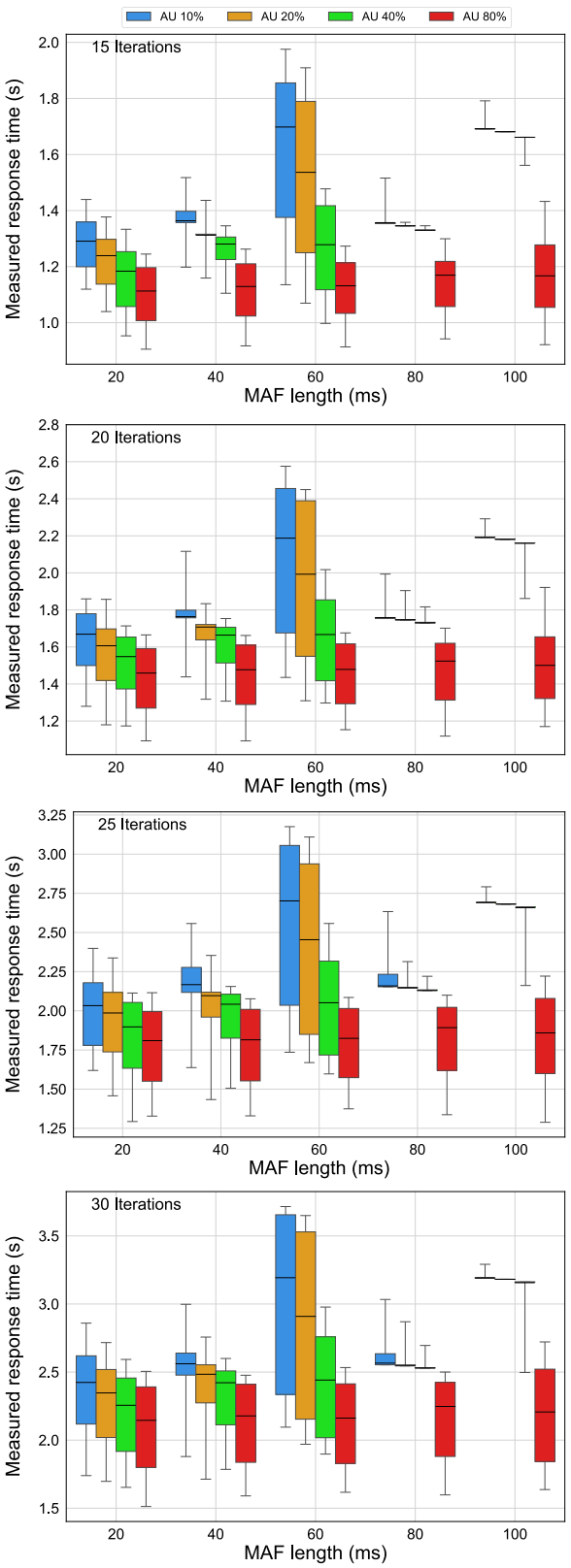


Fig. 21. Measured response times for the PO with SC memory management.

amounts or data are moved between the CPU and the GPU, up to 3.2 MB. This amount of data exceeds the capacity of the GPU cache levels. As a result, the GPU is forced to access global memory to find the necessary data as in the ZC case.

The measurements presented provide us with valuable data to help understanding the behavior of our modeling framework on GPU-based applications. The next step is to compare these results with the analytical worst-case response-time estimations.

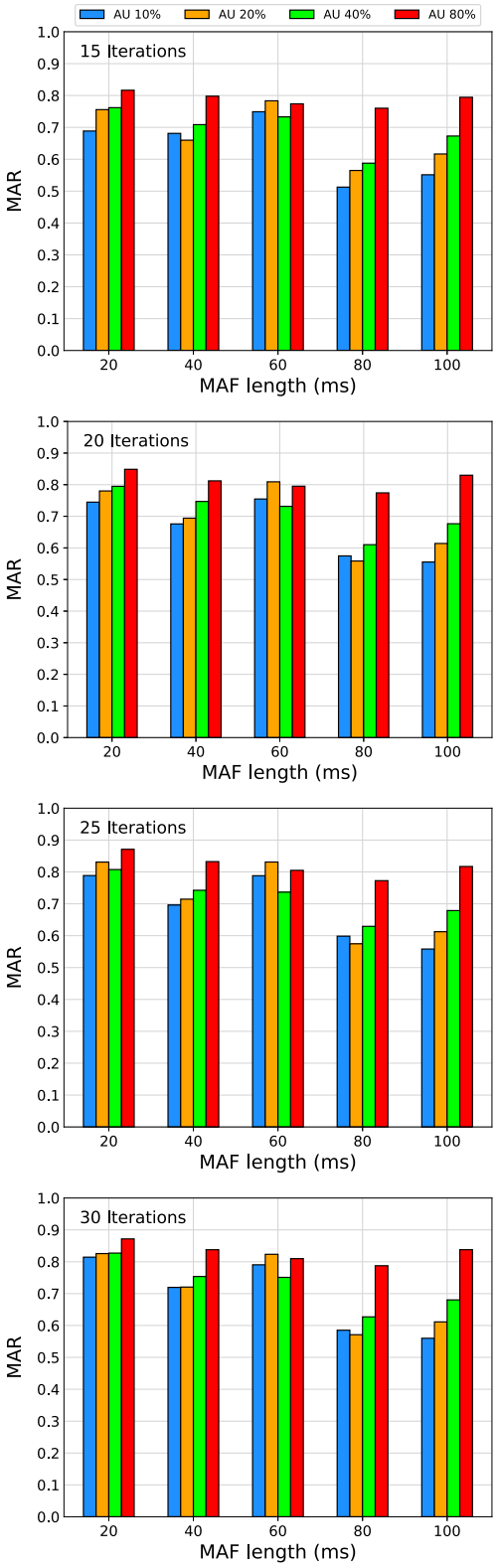


Fig. 22. Measured to Analytical response times Ratio for PO-ZC.

7.5. Comparing analytical and measured response times

In this section, we compare the analytical WCRTs obtained in Section 7.3 with the measured response times obtained in Section 7.4. Usually, the response-time analysis techniques for distributed real-time systems rely on approximations to obtain safe upper bounds for the response times. The objective of this section is to determine how precise the proposed modeling and analysis methodology is, compared to real executions.

For this purpose, we employ the Measured to Analytical response-times Ratio (MAR) as a metric, which is the ratio between the largest response-time measured divided by the analytical WCRT. Therefore, a ratio of 1 indicates that the largest measured response time and the analytical WCRT are the same. A ratio lower than 1 indicates that the analytical value is larger than the measurement. Finally, a ratio higher than 1 signals that the measurement is larger than the analytical WCRT, which would imply that the methodology is incorrect, as the result would not represent a safe upper bound of the response time.

Figs. 22 and 23 show the MARs for PO-ZC and PO-SC, respectively. We can observe that no ratio above 1 is reported, indicating that the analysis technique never underestimated the response time. The difference between the observed and analytical response times is due to two factors: the analysis pessimism and the fact that the measured results do not guarantee reaching the worst case.

From these results, we can determine two main general trends:

1. The analysis provides significantly more pessimistic results for larger e2e flows due to an accumulative effect among the increasing number of modeling elements. This degradation in the accuracy increases with PO-SC, as it requires more steps to model the copy operations.
2. Increasing the unavailability gap between partition windows (i.e. increasing the MAF in our experiments or reducing the Available Utilization (AU)), increases the pessimism.

These observations could be exploited in future optimization techniques, in which scheduling plans could be built with the objective of producing less pessimistic analytical results.

In more detail, the results for PO-ZC (Fig. 22), show a high overall accuracy of the analysis, especially for smaller MAF lengths. For a constant MAF length, higher AU percentages yield a better accuracy. Specifically, for PO-ZC, the accuracy ranged from 51.2% to 87.2%. The synchronization effect observed in the measurements for a MAF length of 80 ms and 100 ms (see Section 7.4), had a negative impact in the obtained results as can be observed in the presented diagrams. As the measured WCRTs have decreased due to this effect (in contrast to the analysis), the MAR shows a decrease in the accuracy.

Focusing on PO-SC (Fig. 23), the obtained results are significantly more pessimistic compared to PO-ZC. Despite both PO-SC and PO-ZC having similar measured response times, the analysis obtained higher estimations for PO-SC. From this evaluation we can report an accuracy between 18.9% and 71.2%. In this case, both the MAF and the AU values had a higher effect on the accuracy of the analysis. This increased pessimism compared to PO-ZC can be explained by the additional steps needed to model the copy operations.

8. Conclusions and future work

In this paper, we have presented a methodology for modeling and analyzing real-time systems that accelerate parts of their workload on GPUs. The proposal is based on leveraging the existing and validated MAST suite of tools, which includes an advanced system model and a selection of state-of-the-art response-time analysis techniques. Time partitioning is used as the first scheduling layer, as it enables controlling concurrent accesses to the GPU. With this approach, the measurement of the execution times on the GPU is simplified, and can be modeled as a delay by using the MAST set of tools. This allows

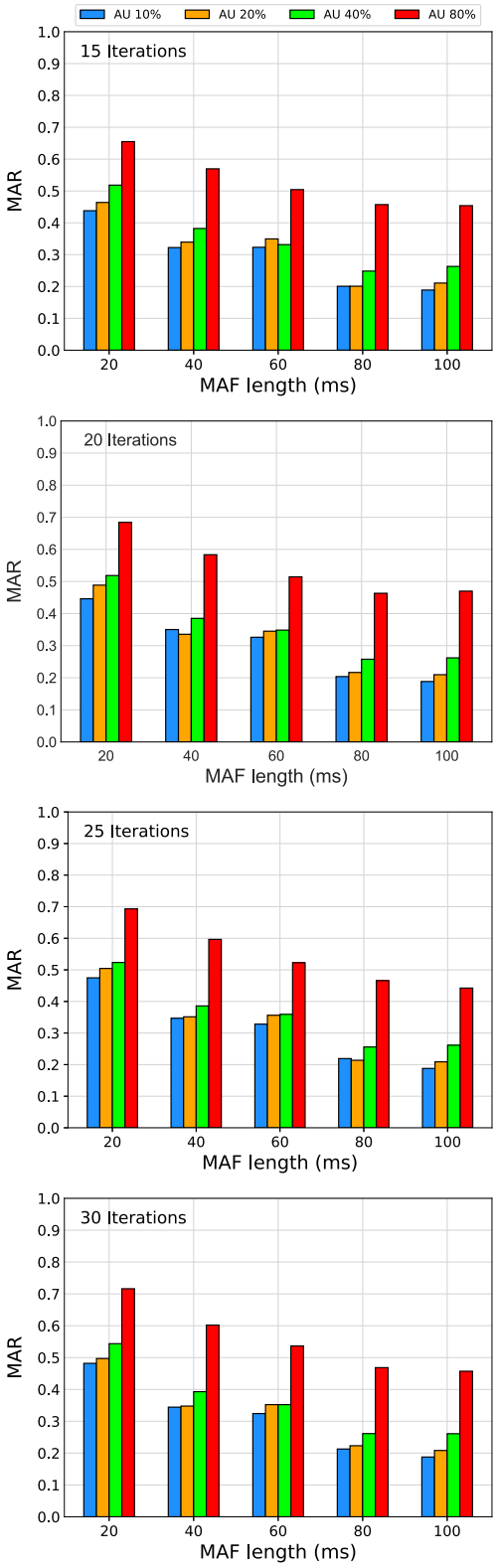


Fig. 23. Measured to Analytical response times Ratio for PO-SC.

modeling and analyzing applications using GPUs in the context of a complex distributed real-time system.

We have presented an industrial use case from the railway sector on which the proposed techniques were evaluated. Multiple time partitioning configurations were tested, involving a range of different MAF lengths and AU proportions. Additionally, the configurations included two different memory management setups for CPU–GPU communication (Zero-Copy and Standard Copy). The evaluation obtained both analytical results, by applying an Offset-Based with Precedences response-time analysis, and experimental results, by performing measurements on a real platform. From this evaluation, it was reported that the analytical results can match the measurements with an accuracy of up to 87.2% for the Zero-Copy memory management. On the other hand, we have found that the Standard Copy memory management requires more elements for modeling that produce higher pessimism in the analytical results.

Different aspects have been studied that can either benefit or adversely affect the accuracy of the analysis. Regarding the partitioning configuration, the gap between partition windows directly influences the analysis accuracy. A MAF where a partition has more windows with lower unavailability gaps, not only achieves better overall response times but also yields more accurate estimations of response times. Although it was expected, it has also been observed that a higher Available Utilization for a partition resulted in lower response times. Another important aspect that influences the accuracy is the length of the modeled e2e flow. As demonstrated in this work, very large e2e flows can lead to significant pessimism. Additionally, besides impacting accuracy, large e2e flows also increase the model's complexity and, consequently, the overall computation time for performing the analysis.

An important aspect, that is left for future work, is the effect of memory interference due to concurrent access to global memory by CPU–GPU or CPU–CPU. Memory interference can increase the execution times, so it should be estimated depending on the configuration of the partitions for the different applications that can coexist in the system. In our work, as the industrial case has been analyzed in isolation, the memory interference does not have effects, but it could be a determining factor in other applications. Based on the methodology presented in this work, we will explore the development of optimization techniques that include all these effects.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: All the authors reports financial support was provided by Spanish Government and FEDER funds. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors want to thank the anonymous reviewers for their many detailed comments which have allowed enhancing the paper significantly.

This work was partially supported by MCIN/ AEI / 10.13039/ 501100011033/ FEDER "Una manera de hacer Europa" under grants PID2021-124502OB-C42 and PID2021-124502OB-C44 (PRESECREL).

References

[1] A. Hamann, D. Dasari, F. Wurst, I. Sañudo, N. Capodieci, P. Burgio, WATERS industrial challenge 2019 final, 2019.

[2] F. Boniol, S. Mohan, IEEE RTSS 2022 industry challenge, 2022, URL: http://2022.rtss.org/industry-session.

[3] M. Andreozzi, G. Gabrielli, B. Venu, G. Travaglini, Industrial challenge 2022: A high-performance real-time case study on arm, in: Leibniz International Proceedings in Informatics, LIPIcs, vol. 231, Schloss Dagstuhl- Leibniz-Zentrum fur Informatik GmbH, Dagstuhl Publishing, 2022, http://dx.doi.org/10.4230/LIPIcs.ECRTS.2022.1.

[4] J. Perez-Cerrolaza, J. Abella, L. Kosmidis, A.J. Calderon, F. Cazorla, J.L. Flores, GPU devices for safety-critical systems: A survey, ACM Comput. Surv. 55 (2022) http://dx.doi.org/10.1145/3549526.

[5] I. Gomez, U.D. de Cerio, J. Parra, J.M. Rivas, J.J. Gutiérrez, Using GPUs in real-time applications - A review of techniques for analyzing and optimizing the timing parameters, Rev. Iberoamericana Autom. Inform. Ind. 21 (1) (2023) 1–16, http://dx.doi.org/10.4995/riai.2023.20321.

[6] Airlines Electronic Engineering Committee, Avionics application software standard interface (ARINC-653), 2010.

[7] E. Cittadini, M. Marinoni, A. Biondi, G. Cicero, G. Buttazzo, Supporting AI-powered real-time cyber-physical systems on heterogeneous platforms via hypervisor technology, Real-Time Syst. 59 (2023) 609–635, http://dx.doi.org/10.1007/s11241-023-09402-4.

[8] R. Cavicchioli, N. Capodieci, M. Bertogna, Memory interference characterization between CPU cores and integrated GPUs in mixed-criticality platforms, in: 2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation, ETFA, 2017, pp. 1–10, http://dx.doi.org/10.1109/ETFA.2017.8247615.

[9] R. Cavicchioli, N. Capodieci, M. Bertogna, Contending memory in heterogeneous SoCs:Evolution in NVIDIA tegra embedded platforms, in: 2020 IEEE 26th International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA, 2020, pp. 1–10, http://dx.doi.org/10.1109/RTCSA50079.2020.9203722.

[10] T. Lugo, S. Lozano, J. Fernández, J. Carretero, A survey of techniques for reducing interference in real-time applications on multicore platforms, IEEE Access 10 (2022) 21853–21882, http://dx.doi.org/10.1109/ACCESS.2022.3151891.

[11] T. Amert, Z. Tong, S. Voronov, J. Bakita, F.D. Smith, J.H. Anderson, TimeWall: Enabling time partitioning for real-time multicore+accelerator platforms, in: 2021 IEEE Real-Time Systems Symposium, RTSS, 2021, pp. 455–468, http://dx.doi.org/10.1109/RTSS52674.2021.00048.

[12] G. Gilman, R.J. Walls, Characterizing concurrency mechanisms for NVIDIA GPUs under deep learning workloads, Perform. Eval. 151 (2021) 102234, http://dx.doi.org/10.1016/j.peva.2021.102234.

[13] M.G. Harbour, J.J. Gutiérrez, J.C. Palencia, J.M. Drake, MAST: Modeling and analysis suite for real time applications, in: Proceedings of 13th Euromicro Conference on Real-Time Systems, IEEE Computer Society Press, Delft, Netherlands, 2001, pp. 125–134.

[14] OMG, An OMG UML profile for MARTE TM publication UML profile for MARTE TM : Modeling and analysis of real-time embedded systems. Version 1.2., 2019.

[15] M.G. Harbour, J.J. Gutiérrez, J.M. Drake, P.L. Martínez, J.C. Palencia, Modeling distributed real-time systems with MAST 2, J. Syst. Archit. 59 (2013) 331–340, http://dx.doi.org/10.1016/j.sysarc.2012.02.001.

[16] J.C. Palencia, M.G. Harbour, J.J. Gutiérrez, J.M. Rivas, Response-time analysis in hierarchically-scheduled time-partitioned distributed systems, IEEE Trans. Parallel Distrib. Syst. 28 (2017) 2017–2030, http://dx.doi.org/10.1109/TPDS.2016.2642960.

[17] A. Amurrio, E. Azketa, J.J. Gutierrez, M. Aldea, M.G. Harbour, Response-time analysis of multipath flows in hierarchically-scheduled time-partitioned distributed real-time systems, IEEE Access 8 (2020) 196700–196711, http://dx.doi.org/10.1109/ACCESS.2020.3033461.

[18] NVIDIA Corporation, CUDA for Tegra, 2024, (Last updated Apr 2024), URL: https://docs.nvidia.com/cuda/cuda-for-tegra-appnote/.

[19] A.J. Calderón, L. Kosmidis, C.F. Nicolas, F.J. Cazorla, P. Onaindia, Understanding and exploiting the internals of GPU resource allocation for critical systems, in: IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers, ICCAD, vol. 2019-November, Institute of Electrical and Electronics Engineers Inc., 2019, http://dx.doi.org/10.1109/ICCAD45719.2019.8942170.

[20] I.S. Olmedo, N. Capodieci, J.L. Martinez, A. Marongiu, M. Bertogna, Dissecting the CUDA scheduling hierarchy: A performance and predictability perspective, in: IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS, vol. 2020-April, Institute of Electrical and Electronics Engineers Inc., 2020, pp. 213–225, http://dx.doi.org/10.1109/RTAS48715.2020.000-5.

[21] J. Singh, I.S. Olmedo, N. Capodieci, A. Marongiu, M. Caccamo, Reconciling QoS and concurrency in NVIDIA GPUs via warp-level scheduling, in: 2022 Design, Automation and Test in Europe Conference and Exhibition, Institute of Electrical and Electronics Engineers Inc., 2022, pp. 1275–1280, http://dx.doi.org/10.23919/DATE54114.2022.9774761.

[22] A. Amurrio, J.J. Gutiérrez, M. Aldea, E. Azketa, Partition window assignment in hierarchically scheduled time-partitioned distributed real-time systems with multipath flows, J. Syst. Archit. 130 (2022) http://dx.doi.org/10.1016/j.sysarc.2022.102671.

[23] R. Höttger, J. Ki, T.B. Bui, B. Igel, O. Spinczyk, CPU-GPU response time and mapping analysis for high-performance automotive systems, in: Proceedings Euromicro Conference on Real-Time System, ECRTS, 2019, pp. 1–7.

[24] L. Krawczyk, C. Wolff, M. Bazzal, R.P. Govindarajan, An analytical approach for calculating end-to-end response times in autonomous driving applications, in: Proceedings Euromicro Conference on Real-Time System, ECRTS, 2019, pp. 1–7.

[25] D. Casini, P. Pazzaglia, A. Biondi, M.D. Natale, Optimized partitioning and priority assignment of real-time applications on heterogeneous platforms with hardware acceleration, J. Syst. Archit. 124 (2022) http://dx.doi.org/10.1016/j.sysarc.2022.102416.

[26] M.A. Serrano, E. Quiñones, Response-time analysis of DAG tasks supporting heterogeneous computing, in: Design Automation & Test in Europe Conference & Exhibition (DATE), vol. Part F137710, Institute of Electrical and Electronics Engineers Inc., 2018, http://dx.doi.org/10.1145/3195970.3196104.

[27] Z. Houssam-Eddine, N. Capodieci, R. Cavicchioli, G. Lipari, M. Bertogna, The HPC-DAG task model for heterogeneous real-time systems, IEEE Trans. Comput. 70 (2021) 1747–1761, http://dx.doi.org/10.1109/TC.2020.3023169.

[28] ERTMS/ECTS: system requirements specification - subset 026 1 - version 4, 2023.

[29] ERTMS/ATO: system requirements specification - subset 125 - version 4, 2023.

[30] CAF signalling - railway signaling solutions, 2024, (Last update Jan 2024), URL: https://www.cafsignalling.com.

[31] NVIDIA Corporation, CUDA toolkit documentation, 2024, (Last updated Jan 2024), URL: https://docs.nvidia.com/cuda/.

[32] J.C. Palencia, J.J. Gutiérrez, M.G. Harbour, Best-case analysis for improving the worst-case schedulability test for distributed hard real-time systems, in: Proceedings 10th EUROMICRO Workshop on Real-Time Systems, 1998, pp. 35–44, http://dx.doi.org/10.1109/EMWRTS.1998.684945.

[33] A. Amurrio, J.J. Gutiérrez, M. Aldea, E. Azketa, Priority assignment in hierarchically scheduled time-partitioned distributed real-time systems with multipath flows, J. Syst. Archit. 122 (2022) 102339, http://dx.doi.org/10.1016/j.sysarc.2021.102339.

[34] K. Tindell, J. Clark, Holistic schedulability analysis for distributed hard real-time systems, Microprocess. Microprogramm. 40 (2) (1994) 117–134, http://dx.doi.org/10.1016/0165-6074(94)90080-9, Parallel Processing in Embedded Real-time Systems.

[35] K. Tindell, Adding Time-Offsets to Schedulability Analysis, Tech. Rep. YCS-221, Department of Computer Science, University of York, 1994.

[36] J.C. Palencia, M.G. Harbour, Schedulability analysis for tasks with static and dynamic offsets, in: Proceedings of 19th IEEE Real-Time Systems Symposium, IEEE, Madrid, Spain, 1998, pp. 26–37, http://dx.doi.org/10.1109/REAL.1998.739728.

[37] J. Mäki-Turja, M. Nolin, Efficient implementation of tight response-times for tasks with offsets, Real-Time Syst. 40 (2008) 77–116, http://dx.doi.org/10.1007/s11241-008-9050-9.

[38] J.C. Palencia, M.G. Harbour, Exploiting precedence relations in the schedulability analysis of distributed real-time systems, in: Proceedings of 20th IEEE Real-Time Systems Symposium, Phoenix, AZ, USA, 1999, pp. 328–339, http://dx.doi.org/10.1109/REAL.1999.818860.

[39] NVIDIA Corporation, Jetson AGX Xavier series, 2023, (Last updated Dec 2023), URL: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-xavier-series/.

[40] B. Mullins, NVIDIA Unveils Jetson AGX Xavier industrial module, 2021, URL: https://blogs.nvidia.com/blog/jetson-agx-xavier-industrial-use-ai/.

[41] The Linux Foundation, Yocto project documentation, 2023, (Last updated Dec 2023), URL: https://docs.yoctoproject.org/.

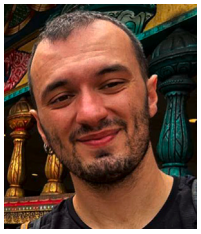
[42] The Linux Foundation, Real-time Linux, 2023, (Last updated Oct 2023), URL: https://wiki.linuxfoundation.org/realtime/start.

[43] A. Asaduzzaman, A. Martinez, A. Sepehri, A time-efficient image processing algorithm for multicore/manycore parallel computing, in: Proceedings of the IEEE SoutheastCon 2015, 2015, pp. 1–5, http://dx.doi.org/10.1109/SECON.2015.7132924.

[44] M. Ji, S. Yi, S. Ahn, D. Seo, N. Dutt, J.-C. Kim, Demand layering for real-time DNN inference with minimized memory usage, in: Proceedings of 2022 IEEE Real-Time Systems Symposium, 2022, pp. 291–304, http://dx.doi.org/10.1109/RTSS55097.2022.00033.

[45] N. Otterness, M. Yang, S. Rust, E. Park, J.H. Anderson, F.D. Smith, A. Berg, S. Wang, An evaluation of the NVIDIA TX1 for supporting real-time computer-vision workloads, in: Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS, Institute of Electrical and Electronics Engineers Inc., 2017, pp. 353–363, http://dx.doi.org/10.1109/RTAS.2017.3.

[46] F. Lumpp, H.D. Patel, N. Bombieri, A framework for optimizing CPU-iGPU communication on embedded platforms, in: Proceedings of the 58th ACM/IEEE Design Automation Conference, DAC, vol. 2021-December, Institute of Electrical and Electronics Engineers Inc., 2021, pp. 685–690, http://dx.doi.org/10.1109/DAC18074.2021.9586304.



Iosu Gomez received the B.S. degree in industrial electronics and automation engineering in 2020, and the M.S. degree in embedded systems engineering in 2022, both from the University of the Basque Country, Donostia-San Sebastian. He is currently pursuing the Ph.D. degree in science and technology in a collaboration framework between the Ikerlan Research Center and the University of Cantabria. During the period from 2020 to 2022, he was involved in research activities as a student at Ikerlan within the Reliable Software team.

Unai Díaz de Cerio received his Bachelor in Computer Engineering from the University of Mondragon, Spain, and his M.Sc in Computation from the University of Cantabria, Spain, in 2010 and 2012 respectively. He pursued his Ph.D. degree in the Department of Computer Science and Electronics at the University of Cantabria. He is working in Ikerlan Research Center as a researcher in the Distributed and Connected Intelligence area. His current research interests are focused on schedulability analysis in distributed hard real-time systems.

Jorge Parra joined Ikerlan Research Centre in 1999, and currently works as full time researcher within the Distributed and Connected Intelligence department. His main activity includes the design and development of smart environments and embedded safety-critical systems, focusing on dependable software aspects. In recent years, he has been actively involved in the design and development of railway systems such as an onboard ERTMS/ETCS system (SIL4) and power loading for tramways (SIL2). Holding an M.Sc. in industrial engineering from the University of Basque Country, he received his Ph.D. degree in telecommunication engineering from the same university. He is a certified TOV Functional Safety Engineer for the design of hardware and software based on the IEC-61508. standard (Fs/Eng.5280/12) since 2009.

Juan M. Rivas is an Assistant Professor in the Software Engineering and Real-Time Group at the University of Cantabria (Spain). He received his B.Sc. degree in Telecommunications Engineering and M.Sc. in Computer Science from the University of Cantabria in 2008 and 2009 respectively. He obtained his Ph.D. degree in Computer Science from the same institution in 2015. He has been involved in several national and European research projects, including industrial collaborations, focusing on topics such as the optimization of distributed hard-real-time systems, modeling, and scheduling in novel platforms such as GPUs.

J. Javier Gutiérrez received his B.S. and Ph.D. Degrees from the University of Cantabria (Spain) in 1989 and 1995 respectively. He is a Professor in the Software Engineering and Real-Time Group at the University of Cantabria, which he joined in the early 90s. His research activity deals with the scheduling, analysis and optimization of embedded real-time distributed systems (including communication networks and distribution middleware). He has been involved in several research projects building real-time controllers for robots, evaluating Ada for real-time applications, developing middleware for real-time distributed systems, and proposing models along with the analysis and optimization techniques for distributed real-time applications.

Michael González Harbour is a Professor in the Department of Computer Science and Electronics at the University of Cantabria. He works in software engineering for real-time systems, and particularly in modeling and schedulability analysis of distributed real-time systems, real-time operating systems, and real-time languages. He is a co-author of "A Practitioner's Handbook on Real-Time Analysis". He has been involved in several industrial projects using Ada to build real-time controllers for robots. Michael has participated in the real-time working group of the POSIX standard for portable operating system interfaces. He is one of the principal authors of the MAST suite for modeling and analyzing real-time systems.