Accurate ILP-based contention modeling on statically scheduled multicore systems

Xavier Palomo*,†, Enrico Mezzetti*, Jaume Abella*, Reinder J. Bril†, Francisco J. Cazorla*
{*name.surname*}*@bsc.es* *r.j.bril@tue.nl*
*Barcelona Supercomputing Center (BSC) †Eindhoven University of Technology (TU/e)

***Abstract*—Commercially available Off The Shelf (COTS) multicores have been assessed as the baseline computing platform even in the most conservative real-time domains. Multicore contention arising on shared hardware resources, with its circular dependence with scheduling, is among the most challenging issues that require urgent attention before multicores can be fully embraced for real-time computing. In the context of static scheduling, still the most used scheduling approach in real-time industries, we propose an ILP formulation for computing the worst-case contention delay suffered by a task due to interference on a shared bus. Our model provides accurate contention delay bounds that avoid unnecessary over-accounting of conflicts between bus requests, by considering contention effects at system-level (i.e., across tasks) rather than at task-level only. This allows precisely capturing the interdependence between timing interference of conflicting requests, issued in parallel by other cores (tasks), and the identification of the particular set of tasks co-running on those cores. We assess our technique both analytically and empirically on a real COTS multicore platform. We show, via extensive evaluation, that jointly accounting for worst-case task overlapping and request distribution scenarios always provides tighter contention bounds when compared to state-of-the-art solutions.**

***Index Terms*—Multicore contention, ILP, COTS, Static scheduling**

I. Introduction

Critical embedded real-time systems are witnessing an unprecedented growth in performance requirements and computational complexity in response to the advent of next-generation software functionalities such as, for example, those related to the increased level of automation involved in autonomous driving and unmanned vehicles [8], [25]. Multicore systems are being widespreadly assessed as the reference solution to meet those emerging requirements, even in the most conservative critical embedded real-time domains, such as avionics, automotive and space. In particular, Commercially-available-Off-The-Shelf (COTS) hardware solutions are typically preferred over custom hardware design in consideration of their reduced non-recurring costs, greater flexibility, and shorter time to market [2], [1], [38].

Critical systems, however, need to undergo a rigorous assessment on both their functional and non-functional properties, generally dictated by domain-specific standards and certification bodies [45], [26]. The timing behavior is a fundamental non-functional property in critical embedded real-time systems, and timing analysis approaches [43] are advocated by standards in order to derive trustworthy guarantees on the timely execution of software functions. The amount of effort devoted to Verification and Validation (V&V) and the precision required to abide by a standard is typically proportional to the consequence of a system failure, which corresponds to different criticality levels (e.g., DAL in avionics [45] or ASIL in automotive [26]) assigned to each function.

Providing strong performance guarantees on top of COTS hardware is complicated by the fact that tasks' execution time can be heavily affected by the contention on accessing shared hardware resources, which can cause a severe increase of the response time of a program [41], [28]. While it is recommended to counter and mitigate all *interference channels* [16], to the best of our knowledge, no COTS hardware exists that allows excluding all sources of interference [49]. Unfortunately, the effects of contention are difficult to characterize as they depend on when, and how often, the program and its co-runners will access the shared resources; and conservatively assuming the worst-case theoretical contention turns out to be an overly-pessimistic assumption. Pessimism translates into a reduction of the system's guaranteed performance, which is generally unaffordable. The unused (wasted) computational power could be used, for example, to accommodate more functions and hence contributing to reduce hardware procurement costs to allocate a fixed set of software functionalities; to perform best-effort activities; or to reduce the energy profile.

Several approaches have been proposed for the characterization of inter-core interference on hardware resources. Some of them aim at exploiting hardware or software level mechanisms to control [29], [50], [38] or even to entirely avoid [41], [40], [6], [11], [13] the effects of contention. While provably effective, these approaches typically build on some assumptions on hardware, RTOS support, and/or application characteristics (e.g., phased execution [41]) that cannot always be guaranteed to hold in practice, especially in COTS platforms. When none of the above solutions can be applied, the only practical way to provide trustworthy performance guarantees under multicore execution that remains is accounting for the worst-case contention delay (WCD) [47], [19]. An important aspect in bounding contention effects is that the computed bounds should be as tight as possible, which in turn depends on the amount of information available to the analysis. Some conservative assumptions are somehow unavoidable: for instance, it is virtually impossible to model precisely *how* requests from different cores align in the access to shared hardware resources, and analysis approaches are forced to assume worst-case alignments. Some other aspects,

instead, as how tasks may overlap on different cores, can be modeled to a certain degree; and tight contention bounds can be obtained by combining information on the activity on shared resources of both the program or task under analysis and a concrete set of potential co-runners.

Context. We focus on the characterization of the worst-case contention effects in statically-scheduled multicore systems. Static scheduling is a prevalent solution in real-time software specifications (e.g., AUTOSAR RTE [9], ARINC [7], ARINC in Space [22]) in critical embedded real-time system domains, as a means to guarantee better determinism, predictability and isolation, even for multicores. In particular, we consider a multicore instance of the ARINC 653 framework, where a static scheduler repeatedly executes a major frame (MAF), further subdivided into a sequence of minor frames (MIF). A statically-mapped set of periodic tasks are non-preemptively executed within each MIF on a multicore platform where cores share a common (partitioned) L2 cache and a main memory, both accessed via a shared round-robin bus. The system (static) schedule is typically automatically generated from a model (e.g., Simulink or SCADE) defining the set of functionalities with their functional and timing requirements. Task-to-core mapping and task ordering is normally determined by considerations on functional cohesiveness and data sharing patterns (e.g., consumer-producer).

Wasting computational resources is especially unwelcome in statically scheduled systems since resources cannot be dynamically reclaimed by the scheduling algorithm. As a result, a potentially large share of computational power may be squandered when excessively loose (un-tight) timing budgets are assigned to tasks. The ability to derive tight contention bounds is, therefore, of utmost importance in such systems. As an example, contention effects have been shown to cause up to 20x increase in execution time in the NXP P4080 [37], and 21x in the NGMP [24]. Alike, the amount of pessimism that can stem in the budget apportionment phase when considering contention effects is potentially huge.

Contribution. In this work, we propose an Integer Linear Programming (ILP) formulation for the accurate computation of the WCD suffered collectively by a set of tasks within a MIF owing to the task running on the other cores in the same interval. The proposed method aims at reducing the amount of pessimism in the WCD computation in view to determine an increase in the guaranteed performance within a MIF. The resulting spare timing budget can be used to improve the quality of the current functionalities (e.g., producing more accurate results) or to accommodate other functionalities, thus avoiding costly system over-dimensioning. Additionally, on a more generic scenario, the accurate WCD computation enabled by our approach can also be leveraged as a heuristic to guide the task-to-core mapping and scheduling optimization. We differ from similar approaches in two aspects: first, we avoid unnecessary pessimism in considering conflicts among requests, by considering contention effects at system-level, rather than at task-level only; and second, we fully exploit the circular dependence among contention and task timing: contention effects on response time change how tasks overlap in time, which in turn determines the contention that can arise in the access to shared hardware resources.

The proposed ILP formulation models all possible overlapping between tasks within a MIF, and computes the worst case contention effects by considering all possible alignments of memory accesses through the bus. Notably, bus accesses may exhibit different latencies depending on the triggering operation: while the number of contention events is bounded by the number of accesses performed by the task under analysis, the contention effects (incurred delays) are determined by the access type of the co-runners. The WCD is, therefore, computed by also taking the access types of the co-runners into account. Our model, therefore, exploits in full the available task-level and system-level information: more accurate WCD bounds can only be obtained under specific scenarios, by leveraging on additional details or constraints on access distribution. Although we make no assumption on access distribution, we show that our ILP formulation is flexible enough to model even those restrictive scenarios.

We extensively evaluate our contention model on randomly generated synthetic task sets, showing that the derived contention bounds support significantly higher overall utilization within MIFs when compared to state-of-the-art approaches [46], [19], thus allowing to deploy more functionalities while keeping strong performance guarantees.

The remainder of this paper is organized as follows. Section II motivates our work and introduces the fundamentals of our approach. Section III discusses related works. Section IV presents the assumptions on which our model builds. Section V presents our ILP formulation for the WCD problem. Section VI provides an analytical and empirical assessment of our technique against representative approaches in the state of the art. Section VII draws some conclusions.

II. Modeling contention based on access pairing

Accesses to the off-chip memory are one of the most critical sources of interference [46], [19], [21]. Every access to the memory hierarchy (either direct or in response to a cache miss) must pass through the interconnect, which easily becomes a performance bottleneck and, equally important, a huge source of timing variability. The WCD suffered by a task τ_i on bus accesses depends on the specific arbitration policy implemented on the bus, and is a function of both (i) the number of accesses performed by τ_i, and (ii) the number and type of accesses performed by its co-runner tasks [18], [28], [21]. Observation (i) is simply dictated by the fact that, based on conservative assumptions on access alignment and considering the specific arbitration policy in the bus, each of τ_i's access can, in the worst-case, suffer a contention delay. However, as per observation (ii), the WCD is also determined by the set of potential co-runners of a task: first, the number of accesses causing contention delay on τ_i cannot be larger than the number of accesses of other tasks running in parallel; and second, the WCD depends on the type of accesses performed by the contender as not all accesses exhibit the same latency.

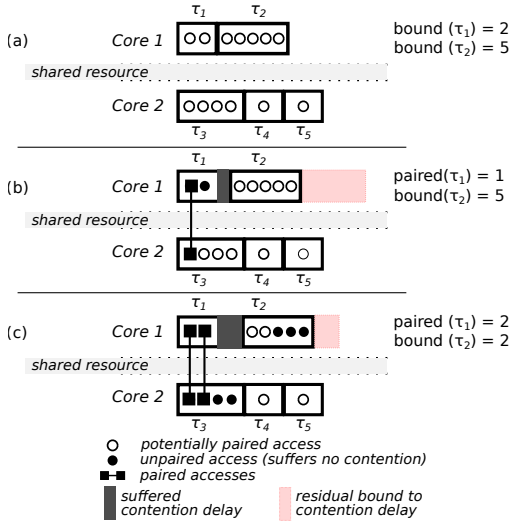


Fig. 1: Dependence btw access pairing and task overlappings.

In order to conservatively but accurately capture the effect of different types of accesses, we use the concept of *access pairing*. Pairing models the fact that requests from different cores can collide in the access to a shared hardware resource. Contention causes an increase in the execution time of the interfered task: the impact on the WCET of the latter is bounded by pairing victim and culprit accesses. Accesses of interfering tasks will cause a contention delay (up to the interfering access latency) on the interfered task for each access they can be paired with. Pairing is ultimately determined by the way accesses to the shared hardware resources are arbitrated. Fair and predictable arbitration policies, like Round-Robin, allow only one-to-one pairings, in the sense that one access from one core cannot be interfered by more than one access per each of the other cores. Priority-based arbitration schemes instead enable different pairings, based on the arbiter semantics. In this paper, we focus on contention arising when accessing a bus implementing FIFO or Round-Robin arbitration, but the approach can be adapted to model any predictable arbitration policy. Note that pairing can only happen when the interfered task and the interfering task, in different cores, overlap in time (i.e., run in parallel). As an example, task τ_1 in Core 1, in Figure 1(a), performs two accesses to the shared resource, symbolized with $\circ$, that can be paired with two accesses in τ_3, executing in parallel in Core 2. These two accesses in τ_3, will not be available for pairing with other accesses from Core 1. Since contenders' accesses may exhibit different latencies depending on the access type, it is important to conservatively pair accesses starting from the most interfering ones (the one exhibiting higher latency).

Access pairing, as a metric to compute contention, and tasks overlapping are in a *mutual relationship* as pairing is determined on the accesses from the set of overlapping tasks, and the latter is affected and potentially altered by the increase in execution time caused by the WCD. At a system level, when considering all tasks in each core, such relationship can be counter-intuitive as local worst-case pairing for a task may lead to a shorter makespan (thus resulting in a timing anomaly [34]). Figure 1 provides an illustrative example of such an effect, avoiding any consideration on access types, for the sake of simplicity. Tasks τ_1, τ_2 execute on Core 1 and perform respectively 2 and 5 accesses, while tasks τ_3, τ_4, τ_5 execute on Core 2, with 4, 1 and 1 accesses respectively (see Figure 1(a)). We focus on the WCD of tasks running on Core 1 and the induced makespan. In principle, 6 out of 7 accesses in τ_1 and τ_2 can potentially be paired, thus conflict, with accesses from tasks running on Core 2. However, this ultimately depends on how tasks overlap in time, which is influenced by how much contention tasks suffer. Let us assume a locally-good scenario, Figure 1(b), in which only 1 access in τ_1 is paired with accesses in τ_3 (i.e. τ_1 will be delayed by exactly one access in τ_3). The resulting overlapping still allows τ_2 having all its 5 accesses paired with accesses from τ_{3-5}, totaling 6 delays. The local worst-case scenario, where τ_1 has all its accesses paired, instead, leads to a task overlapping that is compatible with τ_2 having at most 2 accesses paired with accesses from τ_{4-5}, totaling 4 delays. Thus a local worst-case assumption leads to a shorter, optimistic worst-case makespan.

As a motivation for our work, this simple example shows that trying to compute the WCD on a purely per-task basis can lead to unsafe system-level bounds. We observe that tasks overlapping conditions and access pairing lend themselves to be captured with ILP. We use an ILP formulation to model and explore all possible (feasible) combinations of tasks overlapping and access pairing, thus providing a safe system-level bound to the worst-case contention delay. System-level bounds are particularly relevant in those real-time systems whose execution is clearly organized as the cyclic succession of time frames, such as cyclic-executive and Integrated Modular Avionics (IMA) systems.

III. Related works

The transition to multicores in critical embedded real-time domains has disruptive effects on the consolidated practice for V&V and timing analysis in particular. Several efforts have been invested in the last decades to capture the effect of hardware resource sharing in multicore systems [23], unequivocally identified as the main source of timing interference[1]. While several sources of interference exist at system and processor levels, most works focus on the interference stemming from the shared interconnect.

A first class of approaches aims at limiting the amount of contention in a system or avoiding contention altogether. Some approaches [41], [40], [6], [11], [13] rely on an execution model where task execution is split into memory and computation phases, where accesses to shared resources (i.e., memory system) exclusively happen during memory phases. Tasks are, therefore, scheduled in a way to avoid overlapping of memory phases. The underlying assumptions in these works are that memory phases can be clearly isolated, which is generally made possible by loading the task working set into

[1]We do not consider the effect of software resource sharing and inter-core synchronization, for which analyzable protocols have been devised (e.g., [15]).

local scratchpad memories, and that the platform supports some degree of resource partitioning. While phased execution is a reasonable assumption in parallel applications, it lacks of generality, and clashes with the use of legacy code. Controlling how hardware resources are made available to cores, instead, seems to offer a more generic solution to enforce conflict avoidance. Authors in [29] focus on DRAM and memory controller operations to bound the delay potentially suffered on memory accesses and propose bank partitioning as a way to reduce the amount of inter-core interference. The work in [30] combines hardware management and mixed-criticality based resource provisioning as a means to trade off isolation and sharing of resources and make multicore execution more predictable (i.e., performance guarantees for high-criticality tasks). When hardware isolation is not an option, contention can be controlled at the software level by exploiting specific RTOS support to enforce analysis-time utilization bounds. A memory bandwidth management system to enforce memory usage quotas has been proposed in [50]. In [38], instead, run-time monitoring is used to ensure that co-runner requests stay under a given utilization threshold determined at analysis time. Our proposal does not build on any assumption on task semantics, hardware-level segregation or RTOS-level support, but our ILP formulation is flexible enough to model and eventually take advantage of those same solutions.

In the absence of specialized hardware and/or RTOS-level mechanisms, tight upper bounds for contention effects must be derived, which are often used to guide task to core mappings and co-scheduling decisions. Different heuristics are explored in [14] to classify tasks according to the (negative) effects of running in parallel, and a core-mapping algorithm that avoids tasks with high cumulative miss rates to run together is presented. Early works focused on bounding the number of requests to a shared hardware resource for each single task: assuming uniform distribution of accesses, with a minimum distance between consecutive accesses [46], or assuming dedicated task phases [47]. An upper bound to the interference suffered from a task is determined by summing up the number of accesses from all potential co-runners. The interference bound can be tightened by limiting the analysis to co-runners that may actually overlap with the task under analysis. This observation is exploited in [18], [19], where an upper bound to the effect of contention suffered by a task is derived from an analysis of the maximum number of bus requests issued from the other cores in a given interval. The computation of contention effect is embedded in the standard response time analysis. As a common trait in these approaches, contention analysis is primarily focused on the activity of co-runners. Our approach is similar to [18], [19] in that we do not rely on task phases or specific RTOS support. However, we exploit the fact that tighter bounds to the number of resource requests that may potentially incur contention can be obtained by considering both, task's own requests and co-runners' requests [29]. As a paramount difference with respect to other approaches *we use access pairing to avoid over-accounting of conflicting accesses among tasks*. Additionally, we model the fact that different request types typically exhibit different latencies to further tighten the worst-case contention effects. The work in [48] addresses different sources of contention in NoCs, in a view to optimize task mapping and scheduling optimization, but does not distinguish between request types.

Other works explore the use of ILP for the characterization of contention effect, especially tailored to domain-specific platforms in the automotive [11], [21] and parallel computing [44], [36] application domains. The work in [11] uses an ILP formulation to schedule tasks with phased execution (as in [6]). The ILP formulation presented in [21] models contention delays between pairs of tasks, without any consideration on whether the addressed tasks were overlapping, or on the system-level view of conflictive accesses. Authors in [44] propose contention-aware time-triggered scheduling strategies using some knowledge of the application's structure to find a schedule that minimizes contention under the read-execute-write semantics assumption. We differentiate from these approaches in that we do not seek for optimal schedules and, instead, we assume both static schedule and task-to-core mapping. The work in [36] addresses parallel applications (still assuming read-execute-write semantics) where task-core mapping and schedule are statically defined, and aims at introducing additional slack time in the schedule to reduce resource contention. While sharing some similarities with our work, the method proposed in [36] builds on pessimistic task-level bounds. Our ILP formulation, instead, accurately models worst-case tasks overlapping at system-level and supports pairwise mapping of different request types.

IV. System-level assumptions

Focus on statically scheduled periodic systems: In this work, we focus on statically scheduled, periodic systems, which remain the preferred solution to guarantee timing predictability in the most critical embedded real-time systems. In doing so, we refer to IMA and ARINC-653 [7] concepts, but the same reasoning can be extended to statically scheduled systems in general. In our reference systems, tasks or functions are executed within pre-determined scheduling slots of a given duration, known as MInor Frames (MIF) in ARINC terminology. In the scope of this work we are interested in MIFs as the smallest time interval at which timing guarantees are required to hold. For the time being, we will consider MIFs with duration in the order of milliseconds. The statically determined succession of MIFs finds its larger, collective hyperperiod in the so-called MAjor frame (MAF).

In our multicore scenario, computational resources in each core are assigned according to the same static scheme (as determined by MIFs and MAF), which means that cores execution is synchronized at MIF boundaries. Core synchronization at MIF boundaries is featured, for example, in the IMA multicore implementation supported by SYSGO PikeOS [3] RTOS, and in the Multicore Multiple Independent Levels of Security/Safety (MILS) paradigm supported by VxWorks RTOS [4], [39]. Despite sharing common scheduling slots,

functions on different cores do not engage in any synchronization or communication protocol, unless they preserve time-composability [10]. Inter-core communication (including sharing data) happens through ports or buffers, usually accessed at MIF boundaries to avoid interference from unavailable, incoherent data.

Hardware and software assumptions: We consider a set $\mathcal{T}$ of m independent, periodic tasks with constrained deadlines, denoted $\tau_1, \tau_2, \ldots, \tau_m$. Tasks are statically mapped to cores (i.e., no migration allowed) and are non-preemptively executed according to a time-triggered [31], cyclic static schedule, also known as cyclic executive. We do not consider the effect of release jitter in time-triggered architectures, which we could accommodate in our model building on existing solutions [35]. A task is, therefore, defined as a tuple (c_i, r_i, P_i, D_i), where c_i, r_i represent respectively the worst-case execution and release time, and P_i, D_i represent the task's period and relative deadline. Tasks priorities implicitly model the precedence constraints between tasks. The assumed schedule is not work-conserving as a job will not be executed before the end of the timing budget associated with the job of its predecessor task, even if the latter has already terminated. Without loss of generality, we assume that tasks mapped to the same MIF share the MIF period as their common period, and their relative deadline is set as the MIF size (i.e., the MIF represents the timing budget collectively allocated for the functions thereby mapped). The static schedule is defined by a static allocation of tasks to MIFs that execute in sequence within the MAF. Accordingly, we are not interested in finding an optimal task schedule and/or assignment of tasks to cores [44]. The baseline notation w.r.t. the task set used in the paper is shown in Table I.

We consider a homogeneous system comprising a set Π of n-cores, denoted as $\pi_1, \pi_2, \ldots \pi_n$. Each core exploits its own on-chip private instruction and data caches. Accesses to the shared L2 cache – and eventually to memory – are performed over a shared interconnect (e.g., memory bus), which is therefore the source of contention in the system. The L2 is partitioned, to prevent it from producing inordinate timing interference among tasks. This same setting is found, for example, in the Cobham Gaisler NGMP COTS [5] widely adopted in the space domain. Different types of requests t are served through the interconnect, each one exhibiting a different latency, for which an upper bound l^t is derived, typically through exhaustive measurements [27]. Since we assume homogeneous cores, access targets and types are the same across the system. The information on the target of an off-chip access can be relevant in the presence of multiple interconnects or interconnects that support some degree of parallelism. For each task τ_i we consider the number of issued accesses a_i^t for each access type t. We always refer to worst-case access counts that hold valid for any path in the program, as argued in [18]. The total number of accesses of a task is the summation over all access types of that task. We further assume that requests from different cores queued to a shared resource are served according to a predictable policy (e.g., round-robin). Under round-robin, for example, an access can be paired with at most $|\Pi| - 1$ accesses, each from a different core, regardless of the type of such accesses (which in turn may determine the suffered delay). The notation used to address accesses and their types is summarized in Table I.

Section V describes how these assumptions are factored in our ILP model for the computation of the WCD at MIF level.

V. Contention modeling via ILP

A task may suffer a contention delay whenever it tries to send a request to a shared resource while another task, running on another core, is accessing the same resource. More precisely, contention is suffered only when the request from the task under analysis arrives after a request from the contender (interfering) task. In the impossibility of determining the exact arrival time of other requests to the shared resources, the computation of the WCD cannot escape from conservatively assuming the worst-case request alignment. Under this assumption (and with predictable arbitration policies), we do not need to model the state of the resource access queue to compute the maximum interference incurred by each access as they will be assumed to be served last, after all interfering accesses (e.g., up to one per contending core in round-robin). For this reason, each *paired* access from an interfering core will systematically contribute to the interference suffered by the (interfered) task under analysis. The task timing budget is inflated by adding the (worst-case) latency for each paired access, as determined by the access type of the contender tasks.

Tight WCD calculation for a given task, therefore, depends on two factors: (i) the set of overlapping tasks; and (ii) the number and types of conflicting accesses to a shared resource. To complicate the computation of the WCD, those variables are not independent but can affect each other in a counter-intuitive way, as shown in the illustrative example in Section II. On the other hand, making conservative assumptions on task overlapping, conflicts (e.g., allowing one access to cause more than one conflict on a given core), or incurred latencies (e.g., always assuming the worst-case latency $l^{\max}$ regardless of the request type) easily leads to overly-pessimistic results, that might not be usable in practice. For this same reason, an iterative approach for computing the WCD is unfit since it would imply making potentially unsafe task-level assumptions over conflicting accesses.

Modeling both, task overlapping and access pairing, as an ILP problem allows to compute a tight WCD by implicitly accounting for all possible (feasible) task overlapping and access pairing. In formulating such an ILP problem, we are mainly interested in modeling the worst-case effects of contention at system level rather than at task level. In fact, under the cyclic executive paradigm, performance guarantees are typically enforced at each scheduling interval: in this sense, the MIF duration represents the time budget cumulatively allocated for its mapped functions.

In the following we formally describe our ILP formulation, using the notation reported in Table I.

TABLE I: Notation used in the paper.

Tasks, cores, and mapping	
$\Pi \stackrel{\text{def}}{=} \{\pi_1, \cdots, \pi_n\}$	Considered set of homogeneous cores
$\mathcal{T} \stackrel{\text{def}}{=} \{\tau_1, \tau_2, \ldots, \tau_m\}$	Considered task set
$P_i \geq D_i$	Period (P_i) and deadline (D_i) of task τ_i
c_i	Worst-case execution time in isolation of τ_i
r_i	Release instant of τ_i
$\mathcal{T}_s \stackrel{\text{def}}{=} \{\tau_i : \tau_i \mapsto \pi_s\}$	Subset of tasks statically mapped to core π_s
$\Gamma \stackrel{\text{def}}{=} \{t^1, \cdots, t^k\}$	Access types admitted in the system
l^t	Upper bound to the latency incurred by a single access of type $t \in \Gamma$
$l^{\max}$	Worst-case latency incurred by a single access of any type in $t \in \Gamma$
a_i^t	Number of accesses of type t in τ_i
$a_i \stackrel{\text{def}}{=} \sum_{t \in \Gamma} a_i^t$	Total number of accesses issued by task τ_i
Worst-case delay computation	
$\bowtie_i(\mathcal{T}_s)$	Set of tasks $\mathcal{T}_s$ mapped to core $\pi_s \nleftrightarrow \tau_i$, potentially overlapping with the task under analysis τ_i
$a_{j \triangleright i}^t$	Number of accesses of type t in $\tau_j \in \bowtie_i(\mathcal{T}_s)$ that are *paired* with accesses in τ_i
$a_{j \triangleright i}$	Total number of accesses (of any type) in $\tau_j \in \bowtie_i(\mathcal{T}_s)$ that are *paired* with accesses from τ_i
$\Delta_{j \triangleright i}$	Interference suffered by τ_i because of contention triggered by *paired* accesses of **any** type in τ_j
Δ_i	Overall interference suffered by τ_i
$c_i + \Delta_i = e_i$	Execution-time budget reserved for multicore execution of τ_i after accounting for WCD effects (Δ_i)

A. Objective function

Our system-level WCD computation implies considering the WCD over a sequence of tasks (under the non-preemptive assumption, common in cyclic executive systems). This allows discarding the possibility of considering an interfering access to cause conflicts on more than one task in the same core. As an example, in contrast with other works, we model the fact that even in case a task τ_i in Core 1 is running in parallel with two tasks τ_j and τ_k in Core 2, each access of τ_i can generate contention on at most one request in Core 2, either from τ_j or τ_k. Still, an access of τ_i can generate conflicts on another task in Core 3. When not otherwise specified, our analysis applies at the granularity of scheduling intervals or MIFs: the analysis has to be separately applied to all MIFs in the MAF. Since the model and its formulations apply to a given MIF, we avoid overloading our notation and do not add an indicator of the considered MIF.

For a given task τ_i, we define Δ_i as the WCD suffered by that task when executed within the considered MIF. The inflated execution-time budget reserved for τ_i in multicore execution is denoted as $e_i = c_i + \Delta_i$. Consequently, our objective function maximizes the makespan of the set of tasks mapped to a core and executing within a given scheduling interval (MIF). This is equivalent, in terms of maximization, to the overall cumulative effect of contention suffered by those tasks. For example, considering core π_s (and a MIF):

$$\max makespan_s \equiv \max \sum_{\tau_i \mapsto \pi_s} e_i \equiv \max \sum_{\tau_i \mapsto \pi_s} (c_i + \Delta_i) \quad (1)$$

It is worth observing that while the ILP models the contention effects on all cores, it only maximizes the delay cumulatively incurred by tasks running in the core under analysis, which normally does not coincide with the maximum delay incurred by the tasks in the other cores.

An alternative objective function would be maximizing the cumulative makespan across all cores within a MIF, such as $\max \sum_{\pi_s \in \Pi} \sum_{\tau_i \mapsto \pi_s} e_i$. However, this formulation would only provide realistic makespan figures, which cannot be considered as valid upper bounds for the single cores.

The term Δ_i, used in the objective function to indicate the WCD suffered by τ_i, is the cumulative result of the contention caused by (paired) accesses from overlapping tasks mapped to contender (interfering) cores in the MIF under consideration. For each task τ_i, $\bowtie_i(\mathcal{T}_s)$ identifies the set of tasks mapped to (interfering) core $\pi_s \nleftrightarrow \tau_i$ that can overlap in time with τ_i. At a task-to-task level, the variable of interest is $\Delta_{j \triangleright i}$ to represent the interference suffered by τ_i because of accesses triggered by $\tau_j \in \bowtie_i(\mathcal{T}_s)$.

The interference depends on the number *and* types of accesses from τ_j that are assumed to collide with accesses from τ_i. In order to model the different latencies entailed by multiple access types, we define $\Delta_{j \triangleright i}$ as follows:

$$\Delta_{j \triangleright i} \stackrel{\text{def}}{=} \sum_{t \in \Gamma} a_{j \triangleright i}^t \times l^t \quad (2)$$

where $a_{j \triangleright i}^t$ represents the number of accesses of type t in τ_j that are *paired* with accesses in τ_i, and l^t represents the maximum latency for that type of access.

Accumulating the interference generated by all overlapping tasks in all contender (interfering) cores yields Δ_i:

$$\Delta_i = \sum_{\mathcal{T}_s : \pi_s \nleftrightarrow \tau_i} \sum_{\tau_j \in \bowtie_i(\mathcal{T}_s)} \Delta_{j \triangleright i}$$

B. Modeling pairing of contenders accesses

The WCD caused by τ_j on τ_i is computed based on the number and type of paired accesses since the latter are conservatively assumed to always generate interference on τ_i (i.e., τ_j requests always precedes τ_i ones). To compute the WCD for τ_i we need to consider all possible access parings for τ_i accesses, which depend on, and at the same time potentially affect, the set of (overlapping) contender tasks τ_j. The WCD analysis is performed per contender core (i.e., $\mathcal{T}_s : \pi_s \nleftrightarrow \tau_i$) and per MIF. The interference suffered due to each core is summed up to derive the global WCD for τ_i.

The tightness of the ILP formulation depends on its capability to exclude infeasible pairings. We identified the following constraints on access pairings:

1) Bounds on task-to-task access pairing: The total number of accesses in τ_j *paired* (hence conflicting) with accesses in τ_i is bounded by the access counts in both tasks:

$$a_{j \triangleright i} \leq \min(a_i, a_j) \tag{3}$$

where $a_{j \triangleright i}$ is a generalization of $a^t_{j \triangleright i}$ and indicates the total accesses (of any type) in $\tau_j \in \bowtie_i(\mathcal{T}_s)$ that are *paired* with accesses from τ_i.

More precisely, task pair-wise interference (i.e., the sum of the paired accesses on one direction or the other) cannot be greater than the access counts in both tasks:

$$a_{i \triangleright j} + a_{j \triangleright i} \leq \min(a_i, a_j) \tag{4}$$

In fact, either a task is causing interference or is suffering from it. This constraint, however, will be automatically invalidated by maximizing the contention on one of the two tasks (i.e., setting one term on the left side of Eq. 4 to 0).

When considering different access types, the above constraints can be redefined on a per-type basis (to allow a consistent pairing of latencies) as follows:

$$\sum_{t \in \Gamma} a^t_{j \triangleright i} \leq \min(a_i, a_j) \tag{5}$$

A further constraint can be formulated based on the observation that the number of paired accesses of a given type in the interfering task (τ_j) is limited by the number of accesses of that type in the interfering task and the number of accesses in the interfered task:

$$a^t_{j \triangleright i} \leq \min(a_i, a^t_j) \tag{6}$$

2) Bounds on core-to-core access pairing: While the above task-level constraints are fundamental blocks in the ILP model, they still fail to accurately model the way accesses are paired from a system perspective. The system level perspective considers pairings over the set (sequences) of tasks mapped to a core. Core-level constraints allow modeling the fact that one specific access of a given task cannot be delayed (i.e., be paired) with more than one conflicting access per each contender core. For example, a given task access cannot suffer contention from two accesses performed by two distinct tasks, mapped to the same contender core. These constraints are improving the tightness of the computed WCD, and are one of the aspects that differentiate our approach from related works [44], [36], [21]. In particular, Eq. 8 and 10 are modeling the system-level aspect of our approach and, together with the overlapping condition, are causing a swift increase in the computational complexity of the problem.

We formulate the constraint first from the standpoint of the interfering task, observing that any of its accesses cannot be paired multiple times with tasks on the same interfered core (it can be paired with at most one access per core). Accordingly, we constrain the sum of (interfering) accesses of an interfering task τ_j, that are *paired* with accesses of tasks on a given core $\pi_s \nleftrightarrow \tau_j$, not to exceed the number of accesses in τ_j itself:

$$\sum_{i:\tau_j \in \bowtie_i(\mathcal{T}_s)} a_{j \triangleright i} \leq a_j \tag{7}$$

From the opposite perspective, we also enforce that the number of accesses paired with accesses triggered by the set of overlapping tasks from the same interfering processor can never exceed the number of accesses in the interfered task τ_i:

$$\sum_{\tau_j \in \bowtie_i(\mathcal{T}_s)} a_{j \triangleright i} \leq a_i \tag{8}$$

In a scenario admitting different types of requests (and latencies) we can redefine the constraints by modeling also access types as a further dimension to the problem, as we did for the task-to-task constraints. The number of *paired* accesses of a given type, triggered by a given contender (τ_j) on the overlapping tasks on a given core cannot exceed the number of accesses of that type in the contender itself.

$$\sum_{i:\tau_j \in \bowtie_i(\mathcal{T}_s)} a^t_{j \triangleright i} \leq a^t_j \tag{9}$$

In its dual formulation, the number of *paired* accesses of a given type and triggered by the set of overlapping contender tasks on a given processor, is bounded by the overall number of accesses of any type in the interfered task (τ_i):

$$\sum_{\tau_j \in \bowtie_i(\mathcal{T}_s)} a^t_{j \triangleright i} \leq a_i \tag{10}$$

It is worth noting that a necessary conservative assumption in the analysis of the WCD *at task level* is to assume that the task under analysis always suffers the worst-case possible contention. As a consequence, pairing in the presence of typed accesses should conservatively pair those accesses that incur higher-latency first (see Eq. 6, 10). However, we do not need to model this constraint as the worst-case pairing (of a request of type t) to each task request is already induced by maximizing the overall *makespan*, as part of the objective function.

C. Modeling task overlapping

The ILP also needs a convenient definition of the overlapping condition between tasks. For each scheduling interval, the static schedule defines the set of tasks running on each core, which in turn identifies the set of possible task overlappings. At any instant, each task will only execute in parallel with at most one task per contending core. Further, since tasks follow statically-defined precedence constraints, only a subset of overlappings are possible in practice.

The ILP formulation seeks the worst-case overlapping among the feasible ones. In order to do so, we need to define an overlapping condition. We formalize the negative sufficient condition for task τ_j overlapping with τ_i as follows:

$$\tau_j \notin \bowtie_i(\mathcal{T}_s) \iff (r_i + e_i \leq r_j) \vee (r_j + e_j \leq r_i) \tag{11}$$

That is, τ_i ends in the worst case no later than τ_j release instant or vice versa, as illustrated in Figure 2. We also observe

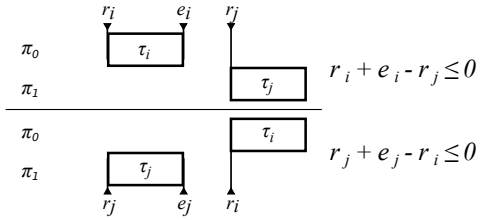


Fig. 2: Non-overlapping tasks.

that Eq.11 is a reflexive relation, so that $\tau_j \in \bowtie_i(\mathcal{T}_s) \iff \tau_i \in \bowtie_j(\mathcal{T}_{s'})$. Moreover, a task cannot overlap with tasks mapped to the same core: $\tau_i \in \mathcal{T}_s \Rightarrow \bowtie_i(\mathcal{T}_s) = \emptyset$.

The condition we need to model builds on integer variables e_i, e_j and r_i, r_j that represent tasks' inflated execution time budget and release instant respectively. When overlapping occurs, these absorb the contention effect on the tasks themselves and their predecessors in the scheduling slot. In fact, the time in isolation and the WCD will determine the budget allocated to each task in the slot. Given a task τ_i and the triggering time of its scheduling slot t_{MIF}, we define a recursive relation to compute the release time of a task, where $pred(i)$ is the predecessor of τ_i, executing right before τ_i on the same core.

$$r_i = \begin{cases} t_{MIF} & \text{if } \tau_i \text{ is the first task in its MIF} \\ r_{pred(i)} + e_{pred(i)} & \text{otherwise} \end{cases}$$

Note that e_i represents the execution time budget assigned to τ_i to account for the WCD interference (i.e., $e_i = c_i + \Delta_i$). The interval $r_i + e_i$ is thus irrevocably reserved for τ_i's execution (as per not work-conserving assumption).

However, it must be noted that both r_x and e_x are indeed variables in the model: for a task under consideration, r_x indirectly models the contention delay incurred by the previous tasks on the same core (as $r_{n+1} = r_n + e_n$); e_x instead models the delay incurred by the task itself ($c_i + \Delta_i$). The ILP evaluates every feasible overlapping scenario (as determined by access pairing) under every feasible value of r_x and e_x, (for both tasks τ_i and τ_j in Eq. 11), and chooses the values that lead to the (cumulative) WCD for the core under analysis, which does not necessarily correspond to the local task-to-task worst-case delay.

The overlapping condition has to be *linearized* in order to be included in the ILP model. In simpler cases, conditionals are typically modeled using binary variables in combination with boolean algebra. In our case, for each pair of potentially overlapping tasks τ_i, τ_j we model the overlapping condition by encoding a constraint on $\Delta_{j \triangleright i}$ (equivalent to $\Delta_{i \triangleright j}$), which is set to be nil when $\tau_j \notin \bowtie_i(\mathcal{T}_s)$ (and $\tau_i \mapsto \pi_s$). To this extent we used a pair of auxiliary boolean variables $B_{i,j}, B'_{i,j}$ as well as a large-enough constant M, typically in the order of hundred thousands.

We encode the overlapping condition as a pair of constraints (one for each operand in the disjunction in Eq.11) as follows:

$$\Delta_{j \triangleright i} \leq 0 + M * (1 - B_{i,j}) \tag{12}$$
$$\Delta_{j \triangleright i} \leq (r_i + c_i - r_j) * \max(a_i, a_j) * l^{\max} + M * B_{i,j} \tag{13}$$

and

$$\Delta_{j \triangleright i} \leq 0 + M * (1 - B'_{i,j}) \tag{14}$$
$$\Delta_{j \triangleright i} \leq (r_j + c_j - r_i) * \max(a_i, a_j) * l^{\max} + M * B'_{i,j} \tag{15}$$

We show the correctness of the above formulation by cases, focusing on the constraints modeling the first operand (Eq. 12-13) as the same reasoning applies to the dual operand. We recall that $r_i + c_i - r_j \leq 0$ meets the non-overlapping condition.

1) $r_i + c_i - r_j < 0$*:* If τ_i and τ_j are not overlapping, $r_i + c_i - r_j$ will take a negative value. Under all scenarios, the ILP solver will strive to maximize the bounds on $\Delta_{j \triangleright i}$ (i.e., having looser bounds). Therefore, for example, if $r_i + c_i - r_j < 0$, the solver will set $B_{i,j} = 1$ (thus activating the big M), to avoid $\Delta_{j \triangleright i}$ being upperbounded by a negative number in Eq. 13. As a side effect, Eq. 12 will set $\Delta_{j \triangleright i} \leq 0$, which is exactly what we wanted to model.

2) $r_i + c_i - r_j = 0$*:* In this case, the tasks are not overlapping either. In fact, $\Delta_{j \triangleright i} \leq 0$ for any choice of $B_{i,j}$.

3) $r_i + c_i - r_j > 0$*:* In this case, tasks might be overlapping (depending on the dual conditions in Eq. 14-15). The solver will set $B_{i,j} = 0$, to avoid $\Delta_{j \triangleright i}$ being upperbounded by 0 in Eq. 12. By setting $B_{i,j} = 0$, Eq. 13 will simply bound the variable $\Delta_{j \triangleright i}$ to a quantity that is *in all cases* larger than the theoretical maximum contention delay. In fact, a task cannot suffer more collisions than the number of its requests (a_i) or the requests from the contender (a_j), and for each colliding request it cannot incur an additional latency larger than the maximum latency for any request type ($l^{\max}$). That is, $\Delta_{j \triangleright i} \leq x * \max(a_i, a_j) * l^{\max}$ is a tautology for any value of x, and will be superseded (discarded) by the constraints on access pairing.

VI. Experimental evaluation

Our experiments focus on evaluating the capabilities of our approach to compute tight system-level WCD bounds. To this extent, we assess our approach both analytically, against other WCD computation approaches in the state of the art, and empirically, on a real multicore COTS, reference platform in the space domain. As part of the analytical assessment, we compare the cumulative effect on the MIF makespan of the WCDs computed using our ILP formulation, against those obtained with comparable state-of-the-art approaches. In fact, since our approach does not rely on any assumption on the application semantics (e.g., separated memory and computation phases) or specific support from the underlying hardware or RTOS, we first assess it against approaches with similar and comparable assumptions [46], [19], [40]. However, for the sake of completeness, we also consider a more restrictive computational model where task's execution is divided into phases (e.g., [41], [40]), to assess the flexibility of our technique to adapt to and support different scenarios.

We complement the evaluation with an empirical assessment of our technique on a real target. Again, we focus on the MIF-level timing behavior and compare the WCD-aware timing budgets, computed with our method and comparable approaches, against the maximum observed makespan.

A. Characterization of the evaluation platform

We assessed our technique on the Cobham-Gaisler LEON4 Next Generation Multicore Platform (NGMP) [5], a reference multicore COTS platform in the space domain, frequently adopted in European Space Agency (ESA) initiatives. The NGMP was considered for both the instantiation of the ILP model to a concrete platform and the empirical assessment of the analytical WCD. The NGMP comprises 4 homogeneous cores operating at 250MHz, with private, first-level instruction and data caches. An Advanced Microcontroller Bus Architecture (AMBA) connects the four cores to a set-associative, unified L2 cache, and to the main memory. The AMBA is therefore the main source of contention in the considered architecture. As a relevant attribute for the WCD modeling, the AMBA implements a Round-Robin arbitration policy, and requests block the execution until they are completely served. In the considered configuration, the L2 cache is partitioned, to avoid further, inordinate timing interference among tasks in different cores. The data cache implements a write-through no write allocate policy, while the L2 cache is write-back.

Different types of requests can go through the AMBA bus to reach either the L2 or the main memory: L2 store hits (*s2h*) and load (*l2h*) hits, L2 clean and dirty misses triggered by load (*l2mc*, *l2md*) and store (*s2mc*, *s2md*) operations. We assume that number and types of accesses can be derived either by static analysis [43] or by collecting information from the performance monitoring counters, as done in [18]. Each access type is characterized by a worst-case latency. The considered per-type latencies were retrieved from [20], [5] and empirically verified by means of the Performance Monitoring Counters (PMC) available on the target, and a set of specific micro-benchmarks. Access types and worst-case latencies, inclusive of low-level memory access delays, such as DRAM refreshes and bank conflicts, are summarized in Table II.

TABLE II: Worst latency per access

Operation	*Latency*
s2h	1
l2h	8
l2mc, s2mc	28
l2md, s2md	31

TABLE III: Categories created from per-access profiles

Profile	Description
CPU	$\leq$ 75 APKI, $\leq$ 1 MPKI
BUS	$>$ 75 APKI, $\leq$ 1 MPKI
MEM	$\leq$ 75 APKI, $>$ 1 MPKI
B+M	$>$ 75 APKI, $>$ 1 MPKI

B. Analytical evaluation of the WCD

The analytical evaluation aims at assessing the precision of the ILP-based WCD computation against previous approaches. The experiments consist in computing the WCD for a task set and under a given contention scenario, and then consider how the overall makespan may vary depending on the adopted approach. The analysis is applied within the scope of one core (the analysis has to be applied to each core separately) and a single scheduling slot (MIF) as it is at its boundaries that timing budgets must be enforced. The analysis can be straightforwardly extended to fit the hyperperiod (MAF) boundaries by applying it to each MIF individually.

Experiments were performed on a large number of synthetic tasks sets, with various overall utilizations, assuming a fixed scheduling slot of 100 milliseconds, which is a representative value for statically scheduled systems [33]. Any other slot size can be considered. Task sets were randomly generated using the UUniFast algorithm [12], filtering out task sets with no overlapping. In particular, we generated 4,000 task sets for each utilization threshold in the range [0.1, 1] with 0.05 steps, for a total of 76,000 task sets. On average, each task set consisted of 32 tasks. An experiment entails a task set to be allocated to each core, to be later run under contention in this particular scenario. Experiments were executed on a cluster with 2x Intel Xeon E5-2630L v4 running at 2.2GHz, with 128GB of RAM memory. *ILP-WCD* took only 2 minutes in the average case to compute the cumulative WCD for a given core, using the IBM CPLEX solver [17]. As part of the evaluation, we developed a set of python scripts supporting a fully automated experimental process covering the generation of the task sets, formatting them into the ILP formulation inputs, and calling the external ILP solver to compute the WCD. The tool is highly adaptable to allow the definition of flexible ILP models, with respect to scheduling assumptions, request types and latencies, and to interface with most popular ILP solvers. The toolset can run experiments in batches.

From the standpoint of inter-core interference, the amount of bus activity performed by an application is a critical aspect of the problem. To warrant a fair evaluation, we focused on few representative access profiles, based on the memory access and cache statistics of benchmarks in the EEMBC [42] and mediabench [32] suites. We characterize four representative profiles based on the performed number of memory *accesses* and L2 *misses per kilo (thousand) instructions*, which we abbreviate into APKI and MPKI respectively, as summarized in Table IV. Profiles range from the CPU-bound profile (*CPU*), relatively robust in terms of contention, to the BUS- and MEM-bound profile (B+M), which instead is prone to consistent contention effects. Task accesses/misses are determined proportionally to the number of instructions in the task itself.

The ILP model is then populated with the data from the task specification and access profiles and experiments are repeated for all access profiles. For a given utilization threshold, each experiment consists in selecting one task set per core: one as the focus of the analysis, whereas the others are assumed to be mapped to the contender cores. We compared the original task set makespan against the one obtained by factoring in the WCD for each task in the set. In particular, we determine the ratio of task sets whose makespan does not overrun the slot boundaries after accounting for the WCD, thus not resulting in an unfeasible schedule. The tighter the WCD bounds the smaller the increase in the makespan and the lower the probability of overruns. We compare our solution against similar approaches in the state-of-the-art, as will be detailed in the description of the individual experiments. Each experiment is specifically designed to underline an aspect that differentiates our proposed approach from the state-of-the-art.

1) Focusing on system-level bounds: Our ILP-based WCD formulation proceeds by pairing accesses at system-level, across job boundaries, by considering the sequence of tasks

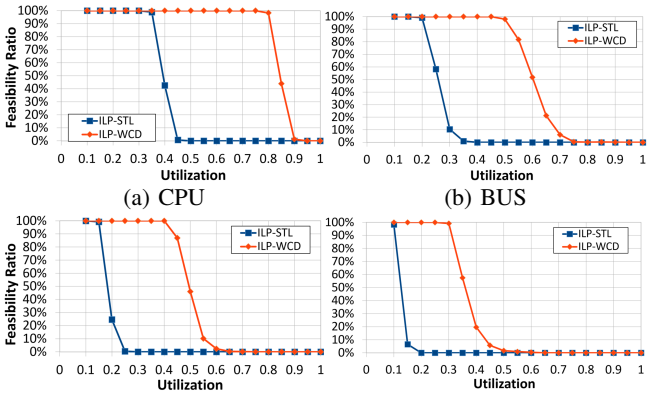


Fig. 3: *ILP-WCD* vs task-level interference (*ILP-STL*).

executing in a core. This allows capturing the core-to-core pairing constraints we introduced in Section V-B2. Mainly, the model benefits from the fact that we are preventing the accesses of a task to interfere with more than one access on the same interfered core (it can be paired with at most one access per core). To evaluate this aspect, we compare our ILP approach, which we call *ILP-WCD*, against the baseline approach presented in [46], which instead operates exclusively at task level, without considering the actual contender tasks. The approach in [46] considers task-level constraints, linking the maximum suffered contention to the number of issued requests. However, all tasks running on contender cores are assumed to be always potential contenders, regardless of the concrete task overlapping. To perform our evaluation, we adapted our ILP model to mimic [46] in our particular setting by disregarding task overlapping and core-to-core constraints. We refer to the implemented technique as *ILP single-task-level* (*ILP-STL*), as it exploits detailed information on the task under analysis but uses only core-level cumulative information on the interfering tasks. To isolate the effect of not considering overlapping, we slightly diverged from [46] by making *ILP-STL* capable of discriminating between request types.

We assess both approaches based on the ratio of task sets whose timing requirement stays within the schedule slot when the WCD is added to their execution time bounds in isolation. Figure 3 shows the ratio of schedules that stay feasible for each utilization threshold, and all workload types (CPU, BUS, MEM, and B+M). Each sample consisted in applying the WCD analysis to 1,000 randomly generated task sets. It is worth noting that utilization threshold refers to the ratio between slot size and tasks WCET in isolation, i.e., not factoring in multicore contention. As a result, as the pressure on the shared resources increases – which mainly happens for MEM and B+M workloads – the resulting multicore CPU utilization goes beyond 100% (at core level) simply making the task set not schedulable any more.

We observe that *ILP-WCD* consistently surpasses *ILP-STL*, achieving good success rates across all utilizations. In fact, the drop in success ratio ('knee' in the figure) for *ILP-STL* occurs, across all workloads, in the utilization range $[0.15-0.25]$, compared to $[0.35-0.85]$ for *ILP-WCD*. In terms

TABLE IV: Assessment of WCD computation methods.

		CPU	BUS	MEM	BUS+MEM
ILP-STL	MAX	2,535233001	4,924514214	4,956007407	6,999572144
	MIN	1,898782342	1,666684591	1,734024544	1,451548745
	AVG	2,149146167	2,331153634	2,679636015	2,982557487
	StdDev	0.082178947	0.209694737	0.310715789	0.569968421
ILP-IRT	MAX	4,531218917	8,151250861	3,352102488	5,791334261
	MIN	1,181862802	1,216282028	1,203235147	1,300552258
	AVG	2,844155223	4,548735984	2,052623178	2,347722482
	StdDev	0.437821053	0.907668421	0.307968421	0.455689474

of workloads, as the pressure on the shared resources increases (from CPU to B+M), all approaches suffer a proportional reduction in the feasibility ratio, with *ILP-WCD* still improving over *ILP-STL*.

Table IV provides further insight into the assessment of *ILP-STL* across utilization threshold, reporting maximum, minimum, average difference and standard deviation, normalized to *ILP-WCD* results. While maxima essentially confirm the sensitivity of *ILP-STL* to all types of requests, minimum values exhibit bounds that are closer to those computed with our method. This arguably happens when the task under analysis is CPU-bound. The very low standard deviation indicates that values are, in general, very close to the average case, thus at around 2.15x slowdown for *ILP-STL* w.r.t. *ILP-WCD*.

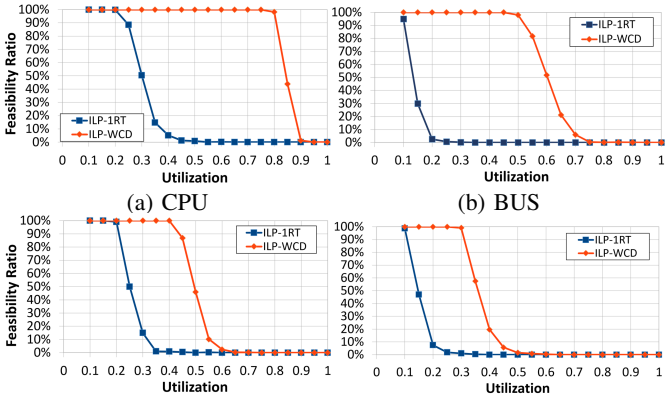
The pessimism of *ILP-STL* when compared to *ILP-WCD* stems from the fact that the latter works at MIF (makespan) level, preventing that a given request from the same contender core is paired more than once with accesses from the task (and core) under analysis. Instead, *ILP-STL* suffers from two main drawbacks: first, the approach considers *all* tasks in the contender core as potentially overlapping; and, second, it works at task level not keeping track of which requests from a given task have been already paired. As a result, in the worst case, one request of a task in the core under analysis can be paired up to K times, where K is the number of tasks in all the contender cores.

In contrast with *ILP-WCD*, the *ILP-STL* approach is not placing any constraint on overlapping nor on requests mapping. Indeed, assuming all tasks in the contender cores to be running in parallel with the task under analysis overturns Eq. 8:

$$\sum_{\tau_j \in \bowtie_i(\mathcal{T}_s)} a_{j \triangleright i} > a_i$$

This causes *ILP-STL* to assume a_i conflicts with way more accesses than practically possible, which in turns entails a notable increase in pessimism in the WCD computation.

2) Supporting single or multiple access types: Another interesting characteristic of *ILP-WCD* is that it captures the fact that the latency incurred by a conflicting access typically varies, even within the same resource, as it depends on the type of the interfering request. Our solution considers several request types and their associated latencies, as can be observed in all typed constraints in Section V. To evaluate this aspect, we compare our approach against the technique presented in [19], which improves over [46] by considering information on both, interfering and interfered tasks, but (i) fails to account for accurate access pairing; and (ii) does not consider that interconnects typically exhibit variable latencies, depending on access types. The approach in [19], still focused on task level,



(a) CPU (b) BUS

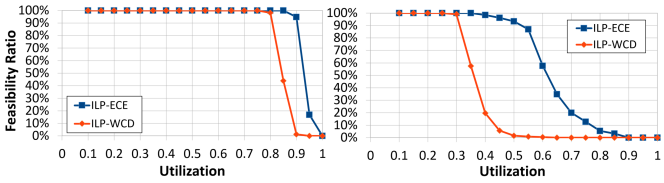
(c) MEM (d) B+M

Fig. 4: *ILP-WCD* vs. single access type (*ILP-1RT*).

allows one access to conflict *at the same time* with accesses from different tasks on the same core. Further, it assumes a single request type, which is equivalent to assuming that all requests incur the highest (worst) latency. To conduct the evaluation, we modeled the ILP to mimic [19]. We refer to this technique as *ILP with one-request-type* (*ILP-1RT*).

Again, we evaluate the ratio of task sets whose timing requirements do not exceed the schedule slot when the WCD is accounted for. Figure 4 shows the feasibility ratio obtained for *ILP-WCD* and *ILP-1RT* for workload types and utilization thresholds. As expected, our *ILP-WCD* model outperforms *ILP-1RT* across all workload types and utilizations. The gap between the two curves identifies the increase in pessimism incurred by *ILP-1RT* with respect to *ILP-WCD*. Interestingly, the pessimism gap between *ILP-1RT* and *ILP-WCD* is more sensitive to the number of bus requests rather than memory ones. This is explained by the fact that each request is assumed to take the longest latency: for bus requests, either loads or stores, that reach L2 but do not miss in the cache (hence not contributing to the MPKI), the incurred pessimism is larger than that associated to actual memory requests. The worst-case latency $l^{\max}$ in our platform is 31, which corresponds to the latency incurred in case of dirty L2 cache misses (either loads or stores). We can compare the overestimation incurred by each bus request as follows. For stores hits ($s2h$), the pessimism added is $l^{\max} - l^{s2h} = 31 - 1 = 30$ cycles, whereas for loads hits it results in $l^{\max} - l^{l2h} = 31 - 8 = 23$ cycles. Instead, *ILP-1RT* incurs notably less pessimism for L2 clean misses, where $l^{\max} - l^{s2mc} = 31 - 28 = 3$ (the same holds for l^{l2mc}). Finally, no additional pessimism is introduced on dirty misses, as $l^{s2md} = l^{l2md} = l^{\max}$. This behavior can be observed comparing Figure 4(b) for BUS and Figure 4(c) for MEM. In the former, tasks trigger a larger amount of bus requests than in the latter, resulting in reduced feasibility ratio for *ILP-1RT*.

As reported in Table IV, the WCD computed with *ILP-1RT* is in the best case quite close to those computed with *ILP-WCD* (just 1.3x in the worst-case contention scenario BUS+MEM). Maxima confirm the sensitivity of *ILP-1RT* to bus requests rather than memory ones, producing 4 times worse contention bounds then *ILP-WCD*. Although the difference between the maximum and minimum is large, the relatively low standard deviation indicates that values are



(a) CPU (d) B+M

Fig. 5: Comparison of *ILP-WCD* and *ILP-ECE*.

generally close to the average case, i.e., 2.84x slowdown of *ILP-1RT* w.r.t. *ILP-WCD*.

The *ILP-1RT*, while improving over *ILP-STL* by considering task overlapping, still fails in placing any constraint on requests mapping, and does not distinguish between access types. Indeed, the final effect, despite a reduced set $\bowtie_i(\mathcal{T}_s)$, is that Eq. 8 gets (again) often overturned, with similar negative effects than in *ILP-STL*. In addition, *ILP-1RT* does not distinguish between access types. The amount of pessimism stemming from this limitation is platform dependent. In our case, this is equivalent to assume that all requests take 31 cycles, as shown in Table II.

C. Supporting multiple execution phases

We further extended our evaluation by considering the flexibility of *ILP-WCD* by extending the underlying model to represent execution phases within tasks. Several studies [41], [40], [6], [11], [13] assume an application semantics that clearly separates phases dedicated to data exchange (E) (usually from/to a local on-chip memory) from phases devoted to pure computation (C). Phases are usually exploited to devise scheduling solutions [40], [6], [11], [13] aiming at conflict avoidance, but they also naturally constrain the possible task overlappings and, hence, access pairing. While we are not interested in imposing restrictions, we are interested in evaluating the potential WCD reduction that can be achieved under a favorable – and more constrictive – computation model and scenario, where additional details are available.

We compare the baseline *ILP-WCD* against its adaptation that supports a scenario where tasks are split into three phases: a relatively large C phase is preceded and followed by E phases, with accesses to shared resources exclusively occurring during the latter. We assume a uniform and fixed duration of each phase in the proportion of 20% and 60% of the task execution budget, for the E phases and the C phase respectively. Tasks carry out half of their accesses in each E phase. We restricted our evaluation to the workloads with the highest and lowest pressure on shared resources, i.e., B+M and CPU respectively. Figure 5 compares the success ratio of *ILP-WCD* and *ILP-ECE*. As expected, *ILP-ECE* always outperforms *ILP-WCD*. The WCD reduction is entirely ascribable to the restrictive assumptions on access distribution. The separation into phases rules out several pairing scenarios that are instead necessarily considered in *ILP-WCD*, leading to better results. However, the ECE execution model requires changes to the application, and it is hard to apply on cache-based systems, where accesses to the bus cannot be scheduled at will. In any case, our ILP formulation also supports it.

For the particular workloads and setup in this experiment, we see that the improvement of *ILP-ECE* over *ILP-WCD* is considerably larger under the B+M workload. This is in line with the characterization of the workloads: B+M is meant to incur the highest contention effects, and the advantage it takes from the *ECE* setting is proportional to the number of accesses.

D. Empirical evaluation on COTS hardware

We also performed a set of experiments to observe the effect of contention on real tasks executing on top of an NGMP target. These experiments aimed at providing a comparative assessment of our ILP model against Maximum Observed Contention Delays (MOCD). It is worth noting that the MOCD is not representative of the worst-case scenario, as it is not possible to control the experiments to incur the worst-case overlapping of bus requests. The *ILP-WCD*, instead, captures the possibility of the worst-case overlapping to occur.

An automatic code generator was used to generate a set of C dummy functions, mimicking the access profile of a number of randomly-generated tasks. These functions have been subsequently serialized in a single sequence (to simulate the behavior of a SCADE static schedule) totaling a cumulative utilization of approx. 50ms. In order to stress our model, we created four task sets with high contention profiles, corresponding to the BUS and B+M workloads. We executed them concurrently on the four NGMP cores and collected 1,000 runs, to capture small variations in the alignment of requests. The PMC on the NGMP were used to track the execution time in isolation (corresponding to the makespan) of those aggregated functions, as well as the number of AMBA requests issued per type. The collected profile information was used to compute the analytical WCD according to the different methods. The same function aggregation is finally executed against other function aggregations in the other cores, to collect the Maximum Observed Execution Time (MOET) under contention, inclusive of the MOCD.

Figure 6 compares the average (coefficient of variation less than 0.3%) MOET under contention against the execution time in isolation, inflated by the analytical WCD computed according to *ILP-WCD, ILP-1RT* and *ILP-STL*. Results are normalized to the MOET under contention. As can be observed, *ILP-WCD* provides the most realistic (tighter) WCD bounds. Despite the conservative assumptions, the obtained timing bound (i.e., accounting for the WCD) was approximately only 2 to 2.5 times the MOET under contention, which we know is typically very optimistic. More pessimistic bounds were obtained with the *ILP-1RT* and *ILP-STL* approaches. The observed behavior is in line with the evidence from analytical assessment, with *ILP-STL* exhibiting larger pessimism due to high MPKI in the task sets. In particular, the experiments confirm that *ILP-STL* behaves better when applied to the task set running in core π_3 as it is characterized by smaller MPKI, while *ILP-STL* is better performing with the task set in core π_4 showing a smaller APKI. Finally, as expected, *ILP-ECE* always outperforms *ILP-WCD* (but still upper-bounding the MOET), owing to the more favorable computation model.

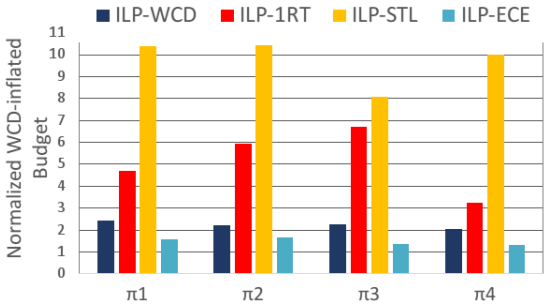


Fig. 6: Empirical assessment on the NGMP target.

E. Computational complexity and scalability considerations

Our method computes the WCD incurred by a set of tasks on a given core and within a given interval (e.g., MIF). As a matter of fact, it is not possible to compute the WCD for all cores at the same time. Assuming two cores π_1 and π_2, the particular request alignment that makes π_2 to cause the WCD on π_1 is not necessarily – and is normally not – the one leading to the WCD for π_2 itself. The overall complexity of the approach is thus linear on the number of cores, intervals in the system and tasks in those intervals, that is $\mathcal{O}(|\Pi| \times |\Pi| \times \#MIF \times |\mathcal{T}|)$.

However, the cost of solving a single instance of the ILP problem – $\mathcal{O}(|\Pi| \times |\mathcal{T}|)$ – is quite modest in practice. In our evaluation, with interval size and number of tasks in line with typical IMA applications [33] (100ms MIFs and averaging 32 tasks per MIF), the CPLEX solver [17] took just 2 minutes on average to find an optimal solution for single *ILP-WCD* instances. Moreover, each *ILP-WCD* instance is completely independent from the other instances, which enables massive parallelization of the computation. Finally, while our evaluation is limited to a 4-cores scenario, we observe that larger number of cores are typically deployed with some form of clustering/segregation, which will limit the scope of the ILP and prevent serious scalability issues.

VII. Conclusions

The way multiple tasks running in a multicore overlap in time has a key effect on the contention they suffer in the access to shared hardware resources. Reciprocally, the latter changes tasks makespan and hence, affects tasks overlapping in time. In this paper, we study the circular dependence between these factors and propose an ILP formulation (*ILP-WCD*) for the computation of tight worst-case contention delay for all tasks. In the context of statically scheduled multicore systems, and in particular of a multicore instance of an ARINC-like system, we have shown that the ability of *ILP-WCD* to operate at core level, on the entire scheduling makespan, allows to outperform other techniques that restrict WCD computation to the task level. Lower WCD bounds can only be obtained by leveraging more restrictive computation models, which are hardly available in general. As a by-product of our evaluation, we also demonstrated the flexibility of *ILP-WCD*, which has been successfully adapted to model different WCD assumptions and scenarios.

ACKNOWLEDGMENTS

This work has been partially supported by the Spanish Ministry of Economy and Competitiveness (MINECO) under grant TIN2015-65316-P, the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant agreement No. 772773), the European Union's Regional Development Fund (ERDF) within the framework of the ERDF (FEDER) program in Catalonia 2014-2020 under the grant SDESI (2016 PROD00115), and the HiPEAC Network of Excellence. Jaume Abella and Enrico Mezzetti have been partially supported by MINECO under Ramon y Cajal and Juan de la Cierva-Incorporación postdoctoral fellowships number RYC-2013-14717 and IJCI-2016-27396 respectively.

REFERENCES

[1] Intel GO Automated Driving Solution Product Brief. https://www.intel.es/content/dam/www/public/us/en/documents/platform-briefs/go-automated-accelerated-product-brief.pdf.

[2] QUALCOMM Snapdragon 820 Automotive Processor. https://www.qualcomm.com/products/snapdragon/processors/820-automotive.

[3] SYSGO PikeOS RTOS. http://www.sysgo.com/, 2018.

[4] WIND RIVER VxWorks MILS Platform 3.0. https://www.windriver.com/, 2018.

[5] Aeroflex Gaisler. *Quad Core LEON4 SPARC V8 Processor - LEON4-NGMP-DRAFT - Data Sheet and Users Manual*, 2011.

[6] A. Alhammad, S. Wasly, and R. Pellizzoni. Memory efficient global scheduling of real-time tasks. In *21st IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 285–296, April 2015.

[7] ARINC. *Specification 651: Design Guide for Integrated Modular Avionics*. Aeronautical Radio, Inc, 1997.

[8] ARM. ARM Expects Vehicle Compute Performance to Increase 100x in Next Decade. https://www.arm.com/about/newsroom/arm-expects-vehicle-compute-performance-to-increase-100x-in-next-decade.php, 2015.

[9] AUTOSAR. *Specification of RTE Software - AUTOSAR CP Release 4.3.1*, 2017.

[10] A. Baldovin, E. Mezzetti, and T. Vardanega. A time-composable operating system. In *12th International Workshop on Worst-Case Execution Time Analysis, (WCET)*, pages 69–80, July 2012.

[11] M. Becker, D. Dasari, B. Nicolic, B. Akesson, V. Nelis, and T. Nolte. Contention-free execution of automotive applications on a clustered many-core platform. In *28th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 14–24, July 2016.

[12] E. Bini and G. C. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1):129–154, May 2005.

[13] A. Biondi and M. Di Natale. Achieving predictable multicore execution of automotive applications using the LET paradigm. In *24th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2018.

[14] S. Blagodurov, S. Zhuravlev, and A. Fedorova. Contention-aware scheduling on multicore systems. *ACM Trans. Comput. Syst.*, 28(4):8:1–8:45, December 2010.

[15] A. Burns and A. J. Wellings. A Schedulability Compatible Multiprocessor Resource Sharing Protocol – MrsP. In *25th Euromicro Conference on Real-Time Systems*, pages 282–291, July 2013.

[16] Certification Authorities Software Team. Multi-core Processors - Position Paper. Technical report, CAST-32A, November 2016.

[17] IBM ILOG Cplex. 12.2 users manual. *Book 12.2 User's Manual, Series 12.2 User's Manual*, 2010.

[18] D. Dasari, B. Andersson, V. Nelis, S. M. Petters, A. Easwaran, and J. Lee. Response Time Analysis of COTS-Based Multicores Considering the Contention on the Shared Memory Bus. In *IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*, pages 1068–1075, Nov 2011.

[19] D. Dasari and V. Nelis. An Analysis of the Impact of Bus Contention on the WCET in Multicores. In *IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems*, HPCC '12, pages 1450–1457, 2012.

[20] E. Díaz, M. Fernández, L. Kosmidis, E. Mezzetti, C. Hernandez, J. Abella, and F.J. Cazorla. MC2: Multicore and Cache Analysis via Deterministic and Probabilistic Jitter Bounding. In *Reliable Software Technologies – Ada-Europe 2017: 22nd Ada-Europe International Conference on Reliable Software Technologies*, pages 102–118. Springer International Publishing, 2017.

[21] E. Diaz, E. Mezzetti, L. Kosmidis, J. Abella, and F. J. Cazorla. Modelling Multicore Contention on the AURIX^{TM} TC27x. In *Design& Automation Conference (DAC)*, June 2018.

[22] N. Diniz and J. Rufino. ARINC 653 in Space. In *DASIA - Data Systems in Aerospace*, ESA Special Publication, 2005.

[23] G. Fernandez, J. Abella, E. Quiñones, C. Rochange, T. Vardanega, and F. J. Cazorla. Contention in Multicore Hardware Shared Resources: Understanding of the State of the Art. In *14th International Workshop on Worst-Case Execution Time Analysis*, volume 39 of *OpenAccess Series in Informatics (OASIcs)*, pages 31–42. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2014.

[24] M. Fernández et al. Assessing the suitability of the NGMP multi-core processor in the space domain. In *EMSOFT*, 2012.

[25] Intel. Next-Generation Transportation. http://www.intel.com/content/www/us/en/automotive/automotive-overview.html., Intel Press Release, 2017.

[26] International Organization for Standardization. *ISO/DIS 26262. Road Vehicles – Functional Safety*, 2009.

[27] J. Jalle, M. Fernandez, J. Abella, J. Andersson, M. Patte, L. Fossati, M. Zulianello, and F. J. Cazorla. Bounding Resource Contention Interference in the Next-Generation Microprocessor (NGMP). In *8th European Congress on Embedded Real Time Software and Systems (ERTS)*, January 2016.

[28] J. Jalle, M. Fernandez, J. Abella, J. Andersson, M. Patte, L. Fossati, M. Zulianello, and F. J. Cazorla. Contention-aware performance monitoring counter support for real-time mpsocs. In *11th IEEE Symposium on Industrial Embedded Systems (SIES)*, pages 1–10, May 2016.

[29] H. Kim, D. de Niz, B. Andersson, M. Klein, O. Mutlu, and R. Rajkumar. Bounding memory interference delay in COTS-based multi-core systems. In *IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 145–154, April 2014.

[30] N. Kim, B. C. Ward, M. Chisholm, C. Y. Fu, J. H. Anderson, and F. D. Smith. Attacking the one-out-of-m multicore problem by combining hardware management with mixed-criticality provisioning. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 1–12, April 2016.

[31] H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Springer Publishing Company, Incorporated, 2nd edition, 2011.

[32] C. Lee, M. Potkonjak, and W. H. Mangione-Smith. Mediabench: a tool for evaluating and synthesizing multimedia and communications systems. In *30th Annual International Symposium on Microarchitecture*, pages 330–335, 1997.

[33] D. Locke, David R. Vogel, L. Lucas, and J. Goodenough. Generic avionics software specification. Technical report, Software Engineering Institute, Carnegie Mellon University, 1990.

[34] T. Lundqvist and P. Stenstrom. Timing anomalies in dynamically scheduled microprocessors. In *Real-time System Symposium (RTSS)*, pages 12–21, December 1999.

[35] P. Marti, J. M. Fuertes, G. Fohler, and K. Ramamritham. Jitter compensation for real-time control systems. In *22nd IEEE Real-Time Systems Symposium (RTSS)*, pages 39–48, 2001.

[36] S. Martinez, D. Hardy, and I. Puaut. Quantifying WCET reduction of parallel applications by introducing slack time to limit resource contention. In *International Conference on Real-Time Networks and Systems (RTNS)*, October 2017.

[37] J. Nowotsch and M. Paulitsch. Leveraging multi-core computing architectures in avionics. In *9th European Dependable Computing Conference (EDCC)*, pages 132–143, May 2012.

[38] J. Nowotsch, M. Paulitsch, D. Bhler, H. Theiling, S. Wegener, and M. Schmidt. Multi-core interference-sensitive WCET analysis leveraging runtime resource capacity enforcement. In *26th Euromicro Conference on Real-Time Systems*, pages 109–118, July 2014.

[39] P. J. Parkinson. Multicore mils. In *9th IET International Conference on System Safety and Cyber Security*, pages 1–8, 2014.

[40] R. Pellizzoni, E. Betti, S. Bak, G. Yao, J. Criswell, M. Caccamo, and R. Kegley. A predictable execution model for COTS-based embedded systems. In *17th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 269–279, April 2011.

[41] R. Pellizzoni, B. D. Bui, M. Caccamo, and L. Sha. Coscheduling of CPU and I/O Transactions in COTS-Based Embedded Systems. In *Real-Time Systems Symposium (RTSS)*, pages 221–231, Nov 2008.

[42] J. Poovey. *Characterization of the EEMBC Benchmark Suite*. North Carolina State University, 2007.

[43] R. Wilhelm et al. The worst-case execution-time problem overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems*, 7:1–53, May 2008.

[44] B. Rouxel, S. Derrien, and I. Puaut. Tightening Contention Delays While Scheduling Parallel Applications on Multi-core Architectures. *ACM Transactions on Embedded Computing Systems (TECS)*, 16(5s):1 – 20, October 2017.

[45] RTCA and EUROCAE. *DO-178C / ED-12C, Software Considerations in Airborne Systems and Equipment Certification*, 2011.

[46] S. Schliecker, M. Negrean, and R. Ernst. Bounding the shared resource load for the performance analysis of multiprocessor systems. In *Conference on Design, Automation and Test in Europe*, DATE, pages 759–764, 2010.

[47] A. Schranzhofer, R. Pellizzoni, J. J. Chen, L. Thiele, and M. Caccamo. Worst-case response time analysis of resource access models in multi-core systems. In *Design Automation Conference*, pages 332–337, June 2010.

[48] S. Skalistis and A. Simalatsar. Worst-case execution time analysis for many-core architectures with noc. In Martin Fränzle and Nicolas Markey, editors, *Formal Modeling and Analysis of Timed Systems*, pages 211–227, Cham, 2016. Springer International Publishing.

[49] P. K. Valsan, H. Yun, and F. Farshchi. Taming non-blocking caches to improve isolation in multicore real-time systems. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), April, 2016*, pages 1–12.

[50] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha. Memguard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms. In *19th IEEE Real-Time and Embedded Technology and Applications Symposium, (RTAS)*, pages 55–64, April 2013.