



CISTER

Research Centre in
Real-Time & Embedded
Computing Systems

Conference Paper

Uneven memory regulation for scheduling IMA applications on multi-core platforms

Muhammad Ali Awan

Pedro Souto

Benny Åkesson

Konstantinos Bletsas

Eduardo Tovar

CISTER-TR-190505

2019/07/09

Uneven memory regulation for scheduling IMA applications on multi-core platforms

Muhammad Ali Awan, Pedro Souto, Benny Åkesson, Konstantinos Bletsas, Eduardo Tovar

CISTER Research Centre

Polytechnic Institute of Porto (ISEP P.Porto)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail: muaan@isep.ipp.pt, pfs@fe.up.pt, kbake@isep.ipp.pt, ksbs@isep.ipp.pt, emt@isep.ipp.pt

<https://www.cister-labs.pt>

Abstract

The adoption of multi-cores for mixed-criticality systems has fueled research on techniques for providing scheduling isolation guarantees to applications of different criticalities. These are especially hard to provide in the presence of contention in shared resources of the system, such as buses and DRAMs. The state-of-the-art Single-Core Equivalence (SCE) framework improves timing isolation by enforcing periodic memory access budgets per core, which allows computing safe stall delays for the cores as input to the schedulability analysis. In this work, we extend the theoretical toolkit for this state-of-the-art framework by considering EDF and server-based scheduling, instead of partitioned fixed-priority scheduling which SCE has assumed so far. A second extension to the theory of SCE consists in additionally allowing memory access budgets to be uneven and defined on a per-server basis, rather than just on a per-core basis, which is what was supported until now. This added flexibility allows better memory bandwidth efficiency, especially when servers with dissimilar memory access requirements co-exist on a given core, and this in turn improves schedulability. Finally, we also formulate an Integer-Linear Programming Model (ILP) guaranteed to find a feasible mapping of a given set of servers to processors, including their execution time and memory access budgets, if such a mapping exists. Our experiments with synthetic task sets confirm that considerable improvement in schedulability can result from the use of per-server memory access budgets under the SCE framework.



Uneven memory regulation for scheduling IMA applications on multi-core platforms

Muhammad Ali Awan¹ · Pedro F. Souto² · Benny Akesson³ ·
Konstantinos Bletsas¹ · Eduardo Tovar¹

Published online: 16 November 2018

© Springer Science+Business Media, LLC, part of Springer Nature 2018

Abstract

The adoption of multi-cores for mixed-criticality systems has fueled research on techniques for providing scheduling isolation guarantees to applications of different criticalities. These are especially hard to provide in the presence of contention in shared resources of the system, such as buses and DRAMs. The state-of-the-art Single-Core Equivalence (SCE) framework improves timing isolation by enforcing periodic memory access budgets per core, which allows computing safe stall delays for the cores as input to the schedulability analysis. In this work, we extend the theoretical toolkit for this state-of-the-art framework by considering EDF and server-based scheduling, instead of partitioned fixed-priority scheduling which SCE has assumed so far. A second extension to the theory of SCE consists in additionally allowing memory access budgets to be uneven and defined on a per-server basis, rather than just on a per-core basis, which is what was supported until now. This added flexibility allows better memory bandwidth efficiency, especially when servers with dissimilar memory access requirements co-exist on a given core, and this in turn improves schedulability. Finally, we also formulate an Integer-Linear Programming Model (ILP) guaranteed to find a feasible mapping of a given set of servers to processors, including their execution time and memory access budgets, if such a mapping exists. Our experiments with synthetic task sets confirm that considerable improvement in schedulability can result from the use of per-server memory access budgets under the SCE framework.

Keywords Safety critical systems · Real-time scheduling · IMA applications · Uneven memory bandwidth · Single core equivalence · Server-based scheduling

✉ Muhammad Ali Awan
muaan@isep.ipp.pt

Extended author information available on the last page of the article

1 Introduction

Cyber-Physical Systems are getting increasingly complex, with the integration of more functionalities and computationally intensive applications. To satisfy this increasing demand for computational power, designers are turning to commercial-of-the-shelf (COTS) multi-core processor architectures, which offer significant advantages in terms of raw computing power, energy consumption, and weight over single-cores. This has led to the wide adoption of multi-core platforms in many application domains, including those that typically employ embedded computing systems.

The transition from single-core to multi-core systems implies that applications must share resources, such as the cores themselves, caches, main memory and I/O devices. While sharing resources reduces cost, it also makes the temporal behaviour of the applications executing on the multi-core processors more complex and highly inter-dependent. This poses problems in some industrial domains, such as avionics, where many applications have stringent timeliness requirements and are *safety-critical*, which means they must always execute correctly, both from the functional and temporal perspectives (Nowotsch et al. 2014). As a result, their design, implementation and testing must be done according to rigorous domain-specific certification guidelines and standards (RTCA 2012a, b), which is both costly and time-consuming. To manage this effort, the strategy of Integrated Modular Avionics (IMA) development (RTCA 2005) is to *isolate applications* from each other to minimise their interactions. A recent position paper from the certification authorities (Certification authorities software team (cast) 2014) provides guidelines for certification of safety-critical avionics systems on multi-core platforms and discusses the issues of resource sharing.

The most prominent approach to address this problem is the Single-Core Equivalence (SCE) framework (Sha et al. 2014), which combines software-based interference mitigation mechanisms for cores, caches, off-chip memory, and I/O. The SCE approach assumes fixed-priority scheduling, which is a well-understood and predictable scheduling policy, but which still falls short of providing the desired degree of isolation, unless used in conjunction with servers. Mancuso et al. (2015) provide a schedulability analysis which assumes that the memory bandwidth of the platform is evenly divided among the cores. Recently, Pellizzoni and Yun (2016) relaxed this assumption by considering uneven per-core budgets. However, even so, all tasks on a given core share the same budget, even if their memory access requirements are very dissimilar, and this may create inter-dependency in their timing behaviour, e.g., when one task “monopolises” the memory access budget to the detriment of other tasks on the same core. The use of servers with separate budgets might mitigate such effects, but this has not been supported until now.

In this paper, we therefore extend the SCE framework to address the issues of isolation and very dissimilar memory bandwidth requirements. The three main contributions of this paper are:

Contribution 1 A stall time computation for the general case of uneven (arbitrary) bandwidth allocation among cores and tasks scheduled using the EDF (Earliest-Deadline-First) scheduling policy for better memory bandwidth usage efficiency and improved schedulability, for the memory system model presented in Mancuso et al.

(2015) that considers the two limits L_{max} and L_{min} (for upper and lower bound, respectively) on the access time of a single memory transaction.

Contribution 2 A schedulability analysis for *server-based* scheduling that enables isolation between IMA applications, each scheduled within a corresponding server. This analysis relies on the previous contribution and adapts the stall-term calculation to periodic, partitioned servers. This analysis assumes that each server has its own per-server memory access budget, which is implemented by adjusting the memory access budget associated with the core where it is assigned, at run time and that the memory arbitration policy is round robin.

Contribution 3 An integer-linear programming (ILP) formulation that computes execution and bandwidth budgets for every IMA application, such that the requirements of all tasks are satisfied. This ILP formulation can be applied to other server-based scheduling approaches used in the IMA-domain, such as fixed-priority scheduling, with fairly straight-forward changes. We evaluate our approach through simulations with synthetic workload and show that uneven and server-based memory access bandwidth allocation indeed increases schedulability. The trade-off between schedulability and computation time for the ILP formulation is also quantified.

The rest of this paper is organized as follows. Section 2 discusses related work before our system model is introduced in Sect. 3. An overview of our approach is then provided in Sect. 4, followed by the derivation of the stall time in Sect. 5. Section 6 then shows how the stall time is integrated with the schedulability analysis for server-based scheduling and Sect. 7 presents the ILP formulation. We experimentally evaluate our approach in Sect. 8, before conclusions are drawn in Sect. 9.

2 Related work

Recent years have witnessed the emergence of several software-based approaches for mitigating memory interference in multi-core platforms (Yun et al. 2012, 2013; Nowotsch et al. 2014; Flodin et al. 2014; Behnam et al. 2013). All of them consider a periodic server implemented in software that manages the memory budgets of the cores. This is combined with run-time monitoring through performance counters that keep track of the number of memory accesses and with an enforcement mechanism that suspends tasks whenever they exhaust their budget. Our work is similar in this respect. Practical issues related to the implementation of such mechanisms on COTS multi-core platforms are discussed in Inam et al. (2014).

An implication of adding memory regulation to mitigate interference is that it invalidates existing work on schedulability analysis, unless it is adapted to account for the new stalls that can occur. This is done for partitioned fixed-priority preemptive scheduling in Yun et al. (2012), Mancuso et al. (2015) and for hierarchical scheduling in Behnam et al. (2013). The hierarchical scheduling in Behnam et al. (2013) is server-based and provides isolation between independent applications. It uses fixed-priority preemptive scheduling as the local scheduler in the server, and provides a per-core schedulability analysis. By comparison, our approach is also server-based,

but uses EDF as the internal server scheduling policy and its schedulability analysis incorporates holistic consideration of memory requirements of all cores.

Once the effects of the regulation mechanism on the schedulability of a task set have been established, the next problem is to allocate execution time budgets such that all tasks meet their deadlines. An algorithm addressing this problem under fixed-priority preemptive scheduling is proposed in Yun et al. (2012), although it only considers the timing requirements of tasks on a single critical core.

Mancuso et al. (2015), on the other hand, target multi-cores, with the problem of preserving schedulability guarantees when porting applications from single-core to multi-core platforms. Under their Single-Core Equivalence (SCE) framework Sha et al. (2014), the task set is partitioned onto the available cores, with fixed-priority scheduling being used on each core. Additionally, the periodic software-based memory regulation mechanism MemGuard Yun et al. (2013), (an intrinsic part of the SCE framework) ensures that, over the regulation interval (or period), all cores get an equal share of the overall memory bandwidth and never more. This assumes that all cores access the main memory through the same memory controller. MemGuard forces a stall of whichever core exceeds its share, until the start of the next regulation period. Such stalls, resulting from the memory regulation, need to be accounted for in the schedulability analysis, in addition to the conventional stalls that occur due to contention between different cores at the memory controller.

The stall-aware schedulability analysis by Mancuso et al. (2015), in addition to the relative deadline and inter-arrival time, characterises each task by its *Worst-Case Execution Time (WCET) in isolation* and its worst-case number of *residual memory accesses*. These correspond to the WCET of the task when no other task is present in the system and an upper bound on the number of memory accesses by the task that are not cache hits, i.e., that go all the way to main memory. It is additionally assumed that each task has its most frequently accessed pages locked in place in the shared last-level cache; specifically, Colored Lockdown (Mancuso et al. 2013) is the mechanism used to guarantee this. Such a design arrangement promotes determinism by eliminating inter-task interference in the cache and Mancuso et al. use this information to characterise the WCET in isolation and the number of residual memory accesses more tightly. Both quantities decrease as the number of locked pages increases; for details see Mancuso et al. (2015). Using these derived task attributes for each task, Mancuso et al. are able to calculate stall terms for the tasks that add to their response time assuming a round robin memory arbitration policy. Their analysis also assumes *DRAM Bank Partitioning* via the OS-level memory allocator PALLOC (Yun et al. 2014).

Although the fair sharing of memory bandwidth among cores under SCE has the benefit of simplicity and of facilitating porting applications from a single-core to a multi-core platform by making the analysis akin to that of a uniprocessor problem case, it is inefficient if the memory requirements of tasks on different cores are too diverse. Therefore, Yao et al. (2016), and Pellizzoni and Yun (2016) generalise the arrangement (and its analysis) to uneven memory budgets per core. Whereas Yao et al. (2016) considers round-robin memory arbitration, fixed-priority scheduling algorithm assuming constant memory access time without any servers, Pellizzoni and Yun (2016) proposes a new analysis for First-Ready First Come First Served (FR-FCFS) memory scheduling. By comparison, our work improves on the theory for the SCE model

Mancuso et al. (2015) by considering periodic server-based scheduling with EDF and *uneven per-server memory budgets* by assuming upper and lower bounds on the access time of a single memory transaction. This means that our stall time analysis works with per-core memory budgets that are variable at run time and can provide inter-server isolation on the same core. We also propose an ILP formulation to find a budget assignment that satisfies the timing requirements of the tasks.

In summary, existing works on memory regulation rely on evenly shared per-core memory budgets. Our work hence goes beyond the state-of-the-art by proposing an ILP formulation supporting uneven bandwidth allocation for an arbitrary number of servers that determines the memory budget and execution budget such that all tasks are schedulable under server-based scheduling. This brings important schedulability improvements over existing approaches.

3 System model

This section presents the system model used in this paper. First, we describe the platform, followed by the applications.

3.1 Platform

This work assumes a multi-core platform composed of m identical physical cores that access main memory via a shared memory controller. The cores are allowed to have more than one outstanding memory request each, but their prefetchers and speculative execution units are disabled. Our assumptions are inspired by those of the Single-Core Equivalence (SCE) framework Sha et al. (2014), but are less restrictive as we explain next.

1. The last-level cache is shared among the cores. A mechanism like Colored Lock-down Mancuso et al. (2013) that mitigates interference between tasks by locking their most frequently accessed pages is compatible with our model, but is not strictly required¹. It would make the number of residual memory accesses for tasks more deterministic (and hence facilitate upper-bounding the number of residual memory accesses per task). However, our work only requires (i) some mechanism for ensuring that the worst-case number of residual memory accesses per task does not depend on the other tasks and the scheduler, and (ii) upper bounds for these numbers to be provided as input. We do not prescribe the way these bounds are computed, but leave it up to the designer to determine the required level of accuracy and what is considered reasonable computation time to achieve it.
2. Partitioning of DRAM banks among the cores using a bank-aware OS-level memory allocator like PALLOC Yun et al. (2014) to reduce interference in the shared DRAM may be desirable, but is not required. What is assumed is that both a lower bound, L_{min} , and an upper bound, L_{max} , for the access time of a single memory

¹ For example, one could alternatively consider partitioning the cache among the tasks without locking their pages in place, but instead allowing the cache partitions to be populated dynamically.

transaction are known. Such bounds can be determined as described in Mancuso et al. (2015).

3. The DRAM memory controller uses a round-robin scheduling policy. Just as it is assumed in Mancuso et al. (2015), “the DRAM controller, as well as the bus arbiter, implement a round-robin scheduling policy”. Such features are indeed supported by modern COTS platforms such as the Freescale MPC56xx and MPC57xx (Mancuso et al. 2015).

Just like in the SCE framework, we also assume an OS-level memory bandwidth regulation mechanism, similar to MemGuard (Yun et al. 2013), to mitigate contention in the memory controller and interconnect. MemGuard uses the worst-case memory access time L_{max} to determine the maximum guaranteeable number of memory accesses, K , during a regulation period P , as $K \stackrel{\text{def}}{=} \frac{P}{L_{max}}$. Under the SCE framework (Mancuso et al. 2015), this memory budget is subdivided into individual memory budgets per core, where each core gets an even share. If a core exhausts its memory budget, the regulation mechanism stalls that core until the end of the current regulation period, when the budget is replenished. We also follow this arrangement and generalise it to allow per-core memory budgets to be (i) uneven and (ii) variable at run time, with their adjustment possible at the start of each new regulation period (by resetting the performance monitoring counter).

3.2 Applications and scheduling

Following the terminology used in the IMA context, we assume a set of N applications, each of which comprises a set of sporadic independent tasks. A task $\tau_i \stackrel{\text{def}}{=} (T_i, D_i, C_i, \mu_i)$ is characterised by its minimum inter-arrival time T_i , relative deadline D_i , a worst-case execution time (WCET) C_i and a worst-case number of residual memory accesses μ_i . Task deadlines are arbitrary, i.e., it does not necessarily hold that $D_i \leq T_i$. C_i is the WCET in isolation, i.e., when the task is the only task executing in the system, but is still limited to using only the memory resources assigned to it: for example, the cache lines and DRAM banks assigned to it via the Colored Lockdown or PALLOC, respectively, if those mechanisms are used. Similarly for μ_i , which only depends on the allocated cache lines. This means that, in essence, C_i and μ_i are functions of the mentioned memory resources, even if in this paper, we assume that the memory resources are allocated a priori, allowing C_i and μ_i to be treated as constants.

All IMA applications are scheduled hierarchically, using EDF, inside periodic, fixed-budget servers. This approach aims for fairness and isolation among different IMA applications. EDF offers better schedulability than Fixed-Priorities² while also being priority-based, as expected by the safety standards (Avionics Application 2010). Note that preemptions do not necessarily occur at the regulation period boundary.

For simplicity, we create one EDF server for each IMA application, even if in practice an application could be split to multiple servers. These servers are then partitioned

² With Fixed-Priority Scheduling, designers sometimes have to artificially shorten periods to make them harmonic, to match EDF’s utilisation bound, however this still entails performance loss, from the artificial task utilisation increase.

offline to the available cores and the servers assigned to each core are scheduled using a form of cyclic executive. The period of the different server-level cyclic schedules on all cores is common and denoted by S . This repeating interval of length S on each core is divided into contiguous time windows, one for each server mapped to that core. Tasks can only execute within the confines of the respective server they were assigned to. A server must be appropriately sized to ensure that its tasks meet all their deadlines at run time.

A server $\tilde{P}_i \stackrel{\text{def}}{=} (K_i, X_i)$ is characterised by its memory budget K_i , i.e., its number of memory accesses within a memory regulation period P , and its execution (time) budget X_i within the server period S . Additionally, the execution budget X_i and the server period S are integer multiples of the regulation period P and synchronized with it.

4 Overview of the approach

We consider the problem of mapping IMA applications to a multicore platform with periodic-server-based EDF scheduling, in the presence of contention over shared hardware resources such that all deadlines are satisfied. In terms of the system model described earlier, this means mapping the servers corresponding to IMA applications to the platform cores and assigning appropriate execution and memory access budgets to these servers, in a holistic process, in order to ensure schedulability. In more detail:

In this work, each server corresponds to one IMA application, whose underlying tasks are given a priori and thus form part of the problem instance. However, the designer has control over the execution budget X_i and the memory access budget K_i assigned to each server. These two server attributes should be considered together for each server because a tradeoff exists. For example, consider a server with a pair (K_i, X_i) that guarantees the schedulability of its tasks. If the designer chooses to shrink the memory budget K_i , the execution budget X_i might need to be increased to preserve the schedulability of the servers' tasks, as compensation for the increased number of stalls that those tasks may face. On a higher level, the (K_i, X_i) budget pairs of all servers need to be considered together, because the overall processing capacity and memory bandwidth of the platform are finite.

To solve this complex problem we use Integer Linear Programming as described in Sect. 7. Our ILP formulation takes as inputs, in addition to the platform parameters, a set of many candidate budget pairs (K_i, X_i) for every server and selects one such pair (K_i, X_i) per server, if a solution is found. Furthermore, it produces the periodic server schedule for each core, as exemplified in Fig. 1. Figure 2 illustrates our approach on a high-level. For every given server, the computation of each candidate (K_i, X_i) pair that is provided as input to the ILP problem is a process with two steps, in which K_i is an input. In the first step, described in Sect. 5, we compute for each K_i the maximum stall time that each task of a server may suffer as a result of sharing the memory bus among cores and of the memory regulation mechanism used. In the second step, described in Sect. 6, we use the stall time of each task computed in the first step to determine the minimum value of X_i that ensures that all server tasks meet their deadline, for each memory budget K_i , by carrying out a schedulability analysis.

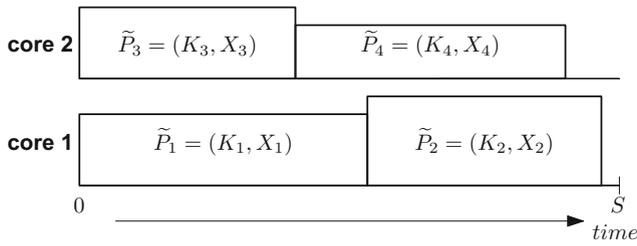


Fig. 1 Example of schedule output by the ILP solver. Server i is represented by a rectangle with width X_i , its execution budget, and height K_i , its memory budget

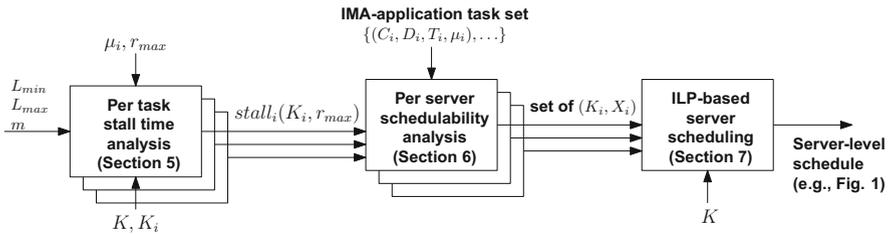


Fig. 2 High-level illustration of the proposed approach. By design the per-task stall time analysis is independent from other tasks and their mapping to servers

5 Stall analysis for uneven assignment

Mancuso et al. (2015) provide a worst-case regulation-induced stall-time for a task i performing μ_i residual memory accesses with even memory bandwidth assignment among different cores when the policy for scheduling DRAM transactions at the bus arbiter and at the DRAM controller is round-robin. (In the remainder of this paper, for the sake of conciseness, when we mention the policy for scheduling DRAM transactions, we omit the reference to the bus arbiter and the DRAM controller.) We generalise their analysis for an uneven memory bandwidth assignment, i.e., not necessarily assuming that $K_i = K/m$, but simply that the memory bandwidth is not overcommitted, i.e. $\sum_i K_i \leq K$.

In this section, we analyze the maximum stall that a task may suffer when it executes μ_i (residual) memory accesses in at most r_{max} regulation periods in a core with memory budget K_i . This analysis is rather abstract, as r_{max} is just an input parameter. It derives an upper bound for the memory stall for any value of r_{max} , even for values of r_{max} that can occur only if the task under analysis is preempted, independently of where preemptions occur³. In Sect. 6, we integrate this analysis in the schedulability analysis under our model assumptions.

Our analysis distinguishes between two kinds of stalls: (1) the *regulation stall* which is the stall that arises upon enforcement of a core’s memory budget by the memory bandwidth regulation mechanism, e.g., MemGuard, and (2) the *contention stall* which

³ In Mancuso et al. (2015) and Yao et al. (2016) the stall analysis implicitly assumes no preemption. However, by using the concept of a synthetic equivalent task, comprising the task under analysis as well as higher priority tasks, the schedulability analysis is valid regardless of whether or not there are preemptions.

Table 1 List of stall analysis symbols

Symbol	Description
m	No. of cores
L_{max}	Maximum latency of a memory transaction
L_{min}	Minimum latency of a memory transaction
μ_i	No. of residual memory transactions per job of task i
r_{max}	Maximum number of regulations periods with residual memory transactions
P	Memguard regulation period
K	No. of memory transactions guaranteed by the memory system in P
K_i	Memory budget per P of core i (in Sect. 5.1) or of task i 's core (in Sect. 5.2).
a_0	Minimum number of memory transactions per regulation period by one core for the upper bound of the stall time per regulation period of that core given in Lemma 1 to be tighter than the upper bound given by Observation 1. (See Sect. 5.1.)

is the stall that arises from the sharing of memory system resources among the different cores, or, more precisely, among tasks running on different cores.

For ease of reference, Table 1 lists the main symbols used in this analysis.

5.1 Worst-case regulation-induced stall

We begin by presenting one observation and a simple lemma⁴ that are used in the proofs of subsequent lemmas.

Observation 1 *In a platform with m cores and a round-robin policy for scheduling DRAM transactions, $(m - 1)L_{max}$ is an upper-bound on the contention stall time per memory access by any core. This is because, in the worst case, a core will have to wait for one memory access by each of the other cores.*

Lemma 1 *In a regulation period where core i does not incur a regulation stall and performs a total of a memory accesses, it holds that 1) $(K - K_i)L_{max}$ is an upper bound on the stall time suffered by core i , and 2) this bound is at least as tight as the bound (i.e., $a \cdot (m - 1) \cdot L_{max}$) derived from Observation 1, if the number of accesses a in that period is at least $a_0 = \left\lceil \frac{K - K_i}{m - 1} \right\rceil$.*

Proof In a regulation period, cores other than i may perform at most $K - K_i$ memory transactions. In the worst case, each of these transactions causes a contention stall on core i . Therefore, the maximum stall time that core i may suffer in a period where it does not suffer a regulation stall is $(K - K_i)L_{max}$.

By Observation 1, if DRAM transactions are scheduled using the round-robin policy, then the maximum stall per memory access is $(m - 1)L_{max}$. Thus, an upper bound of the stall determined by the round robin policy is $a \cdot (m - 1) \cdot L_{max}$, where a is the number of accesses by core i during the regulation period in consideration.

⁴ We start the numbering of lemmas at 0, so that lemmas corresponding to those in Mancuso et al. (2015) have matching numbers.

So the bound $((K - K_i)L_{max})$ proven earlier in this lemma is tighter than the latter bound, if and only if:

$$a \cdot (m - 1)L_{max} > (K - K_i)L_{max}$$

or equivalently

$$a > \frac{(K - K_i)}{m - 1}$$

□

The next two lemmas are the lemmas with matching number of Mancuso et al. (2015). (The phrasing of these lemmas is the one used in Mancuso et al. (2015), except for the changes required by the different policies in the assignment of the memory bandwidth to the different cores: even vs. uneven.)

Lemma 2 *Consider a system with memory bandwidth regulation, where K_i is the number of DRAM accesses of core i during a regulation period of length P . If the policy for scheduling DRAM transactions at the bus arbiter and at the DRAM controller is round-robin, then each core i is guaranteed to always perform up to K_i memory transactions within a regulation period P .*

This lemma follows trivially from the properties of the bandwidth regulation system, e.g. MemGuard Yun et al. (2013), which, by definition, stalls any core that exhausts its memory budget in a regulation period P .

Lemma 3 *For any single regulation period P , $P - K_i L_{min}$ is an upper bound on the amount of stall suffered by core i .*

Proof The total stall time in a regulation period can be expressed as $st = st_c + st_r$, where st_c is the (contention) stall time before the completion of memory transaction K_i of core i , and st_r is the (regulation) stall time after the completion of that memory transaction. Thus, st_c arises from concurrent memory transactions on behalf of other cores, whereas st_r arises from the action of the regulation system. We consider 2 cases depending on whether or not st_r is zero.

Case 1 $st_r = 0$: In this case, by Lemma 1, an upper bound of the stall is $(K - K_i)L_{max}$, thus:

$$\begin{aligned} (K - K_i)L_{max} &= \left(\frac{P}{L_{max}} - K_i \right) L_{max} && \text{by definition of } K \\ &= P - K_i L_{max} \\ &\leq P - K_i L_{min} && \text{by } L_{min} \leq L_{max} \end{aligned}$$

Case 2 $st_r \neq 0$: In this case, core i performs K_i memory transactions. Therefore, $st \leq P - Mem(K_i)$, where $Mem(K_i)$ is the time it takes for core i to perform K_i memory transactions. That is, we bound the stall of core i by deducting from the

regulation period the time core i takes to perform the K_i memory transactions. Thus, the faster core i performs its memory transactions, the higher the upper bound for its stall time may be. So, for any regulation period in which the regulation mechanism stalls core i , the stall time of core i cannot be larger than $P - K_i L_{min}$. \square

Lemma 4 in Mancuso et al. (2015) states that the worst-case regulation-induced stall on a given task occurs when all the accesses of that task are clustered. However, under uneven memory bandwidth assignment, this is not the case as shown by the following lemma.

Lemma 4 *Under uneven memory bandwidth assignment, the clustering of memory accesses does not always lead to the worst-case regulation-induced stall-time.*

Proof Consider an uneven memory bandwidth assignment. Let K_{min} be the minimum bandwidth assigned to any core. Let $K_i \geq K/(m - 1)$ be the memory budget assigned to core i , and $K_{min} < a_0$.

Consider K_i memory accesses by a given task on core i . These accesses may be clustered in a single regulation period or spread over several regulation periods.

In the latter case, if the accesses are evenly distributed over a number of regulation periods equal to $\lceil K_i/K_{min} \rceil$, then each one of the K_i transactions can suffer the maximum contention stall under round robin, $(m - 1)L_{max}$. Therefore the total stall time will be:

$$\begin{aligned}
 K_i(m - 1)L_{max} &\geq \frac{K}{m - 1}(m - 1)L_{max} && \text{by } K_i \geq \frac{K}{m - 1} \\
 &= KL_{max} = P && \text{by definition of } K \\
 &> P - K_i L_{min}
 \end{aligned}$$

Thus, by Lemma 3, under uneven memory bandwidth assignment, the total stall time incurred by the K_i transactions when spread evenly over $\lceil K_i/K_{min} \rceil$ rounds is larger than the stall time of the same transactions when clustered in a single period. \square

5.2 Stall term

A corollary of Lemma 4 is that the stall term analysis in Mancuso et al. (2015), which assumes that the total stall time is maximum when the regulation stall time is maximum, is not applicable to uneven assignment of memory bandwidth. Thus, in this section, we derive an upper bound for the memory regulation induced stall term of a task under uneven assignment of the memory bandwidth across the cores.

Our approach is to determine the distribution of the μ_i residual memory accesses of task i by up to r_{max} that leads to the maximum stall, under the restriction that such a job cannot perform more than K_i DRAM accesses per regulation period. (Without loss of generality, we assume that task i executes on core i , thus avoiding cluttering some symbols with subscripts or superscripts.)

In the previous section, we determined three upper-bounds on stall-time per regulation period of a core whose memory budget is K_i DRAM accesses per regulation

period: one upper bound for regulation periods with regulation stalls, $P - K_i L_{min}$, see Lemma 3, and two upper bounds for regulation periods without regulation stalls, $a(m - 1)L_{max}$ and $(K - K_i)L_{max}$, depending on whether or not the total number of DRAM accesses, a , in a regulation period is smaller than a_0 , see Lemma 1. Therefore, our analysis is case based. First, we consider the case when the upper bound for regulation periods with regulations stalls is larger than or equal to the upper bounds for regulation periods without regulation stalls, and then the opposite. Obviously, the two cases combined are exhaustive. Before that, we make an observation that applies to all cases:

Observation 2 *A job of a task i may suffer a regulation stall on its first memory access. Indeed, consider a job of task i that is scheduled to run when its core has already performed $K_i - 1$ memory accesses in the current regulation period. Upon the first memory access by that job, the core will exhaust its memory budget for the regulation period, and therefore it will suffer a regulation stall.*

Although a job needs to perform one memory access in order to stall, we assume that a job suffers a regulation stall when it is first scheduled, even though it does not perform any memory access. This is safe, and makes the mathematical expressions slightly shorter.

We now present the upper bounds for the two cases mentioned above as lemmas, which are used in deriving an upper bound of the regulation-induced stall-term of a task under uneven assignment of the memory bandwidth across the cores. The proofs of these lemmas are also case based, and both of them are lengthy and require attention to details. To allow the reader to focus on the main ideas, we present these proofs in Appendix A.

For ease of reference, Table 2 lists the main symbols, in addition to those defined in Table 1, used in these lemmas. We use a for the number of memory accesses, r , for the number of regulation periods and Δ for the additional per memory access stall time on specific memory accesses. As for the subscripts, we use 0 for variables related to a_0 , defined in Lemma 1, r for parameters related to regulation periods with regulation stalls, \bar{r} for parameters related to periods without regulation stalls. All these symbols are per task, i.e. each parameter may have a different value depending on the task, and may also depend on the memory budget of the core on which the task runs. However, they do not depend on the parameters of other tasks or other cores, therefore, to prevent clutter we do not use subscripts or superscripts identifying the task or the core.

Figure 3 illustrates the total stall time per regulation period as a function of the number of memory accesses, when the stall time upper bound in a regulation period with a regulation stall is larger than or equal to the upper bounds in a regulation period without a regulation stall, assuming $K_i > a_0$. (In the proof in Appendix A, we consider also the case when $K_i \leq a_0$.) The $P - K_i L_{min}$ bound applies only if the number of accesses in a regulation period is K_i . For a smaller number of accesses we apply the lowest of the $a(m - 1)L_{max}$ and the $P - K_i L_{max}$ bounds, since they are both upper bounds when there is no regulation stall and we want to derive the tightest upper bound for the stall term. Thus, by applying a case analysis to the distribution of the μ_i (residual) memory accesses by up to r_{max} regulation periods during which a job of task i may execute, we determined the following upper bound for the stall term.

Table 2 List of symbols used in the stall term analysis in Sect. 5.2

Symbol	Description
$a_{\bar{r}}$	Number of memory accesses in periods without regulation stall, when the number of regulation stalls is maximum
$a_{\bar{r}0}$	Number of memory accesses in periods both without regulation stall that have less than a_0 memory accesses.
r_0	Number of regulation periods without regulation stalls that have at least a_0 memory accesses
r_r	Number of regulation periods with regulation stalls that have at least a_0 memory accesses
r_{max}	Maximum number of regulation periods in an execution of a job of task i
$r_{\bar{r}max}$	Maximum number of regulation periods without regulation stall
Δ_0	Additional stall upon the a_0 th access on a regulation period P
Δ_r	Average additional stall per memory access after the $(a_0 - 1)$ th access in a regulation period with a regulation stall

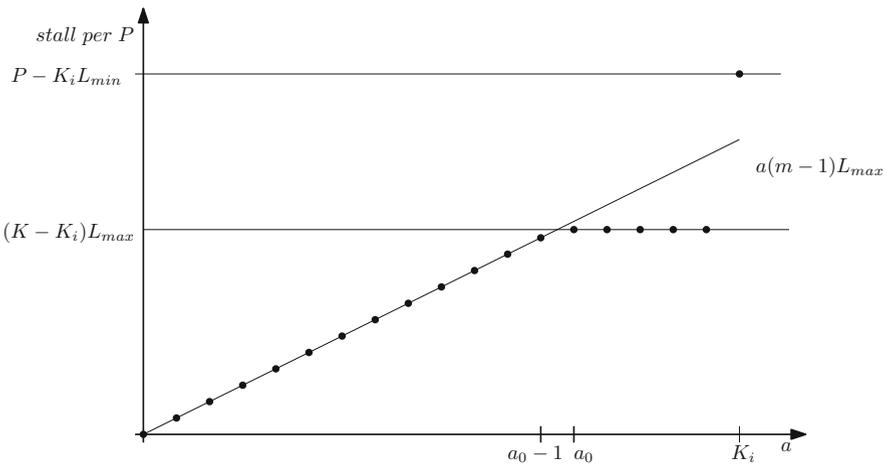


Fig. 3 Stall time in a regulation period as a function of the number of memory accesses, when $P - K_i L_{min} \geq K_i(m - 1)L_{max}$

Lemma 5 Let $\mu_i \leq K_i \cdot r_{max}$. If $P - K_i L_{min} \geq K_i(m - 1)L_{max}$, then the worst-case stall time of a job of task i that executes for up to r_{max} regulation periods, ignoring any regulation stall upon its first memory access, is upper bounded by:

$$\begin{aligned}
 stall_i^r(K_i, r_{max}) = & \left\lfloor \frac{\mu_i}{K_i} \right\rfloor (P - K_i L_{min}) + r_0(K - K_i)L_{max} \\
 & + \min(a_{\bar{r}}, (r_{\bar{r}max} - r_0)(a_0 - 1))(m - 1)L_{max} \quad (1)
 \end{aligned}$$

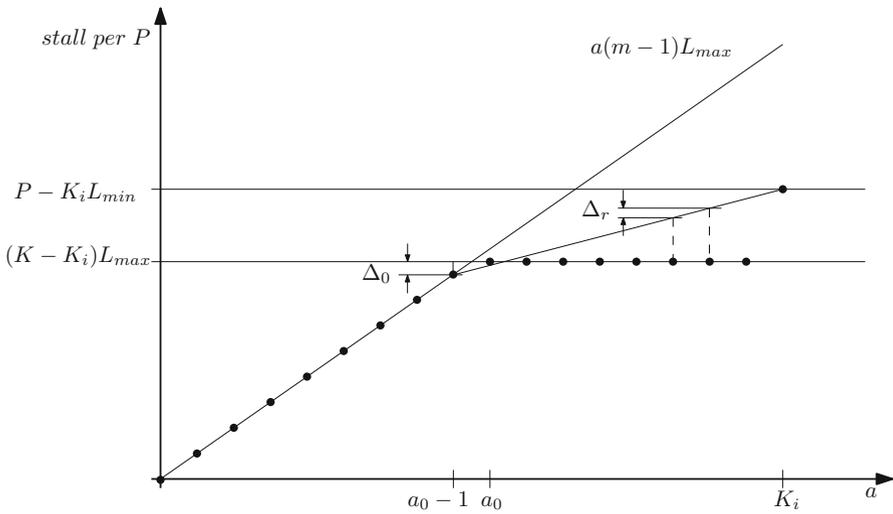


Fig. 4 Stall time in a regulation period as a function of the number of memory accesses, when $P - K_i L_{min} < K_i(m - 1)L_{max}$ and $\Delta_0 > \Delta_r$

where:

$$\begin{aligned}
 a_0 &= \left\lceil \frac{K - K_i}{m - 1} \right\rceil \\
 a_{\bar{r}} &= \mu_i - \left\lfloor \frac{\mu_i}{K_i} \right\rfloor K_i \\
 r_0 &= \begin{cases} 0 & \text{if } K_i \leq a_0 \\ \max(\min(a_{\bar{r}} - (a_0 - 1)r_{\bar{r}max}, r_{\bar{r}max}), 0) & \text{otherwise} \end{cases} \\
 r_{\bar{r}max} &= r_{max} - \left\lfloor \frac{\mu_i}{K_i} \right\rfloor
 \end{aligned}$$

Proof See Appendix A. □

Note that each term on the right-hand side of (1) factors in one of the three bounds presented earlier. For each term, the other factor is the number of times the respective bound should apply to maximize the total stall time of a job of task i that executes for up to r_{max} regulation periods.

By Lemma 4, with uneven bandwidth assignment, the worst-case stall time does not necessarily occur when the regulation stall time is maximum. Figure 4 illustrates the total stall time per regulation period as a function of the number of memory accesses, when the stall time upper bound in a regulation period with a regulation stall may be smaller than the stall time upper bounds for a regulation period without a regulation stall, assuming $K_i > a_0$ and $\Delta_0 > \Delta_r$. Although Fig. 4 appears very similar to Fig. 3, there are several more cases to consider. On the one hand, because $P - K_i L_{min} \geq K_i(m - 1)L_{max}$ does not hold any more, it may happen that the

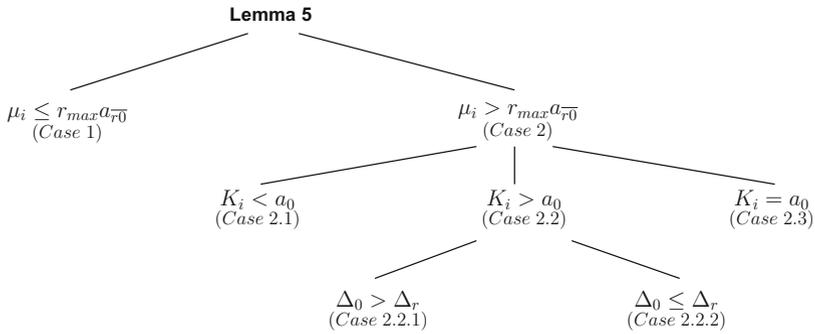


Fig. 5 Overview of the cases used to prove Lemma 6

worst-case stall time occurs when all μ_i residual memory accesses are distributed over up to r_{max} regulation periods in such a way that no regulation stall ever occurs, as shown in the proof of Lemma 4. On the other hand, if $\mu_i > (a_0 - 1)r_{max}$, the worst-case stall time may occur when the additional $\mu_i - (a_0 - 1)r_{max}$ are distributed in such a way that the number of regulation periods with regulation stalls is maximized. Thus, although the goal of the case analysis, like for Lemma 5, is to determine the distribution of the μ_i memory accesses over up to r_{max} regulation periods that leads to the worst-case stall time, the number of cases almost doubles. Figure 5 summarizes the different cases considered in the proof, see Appendix A, of the worst-case stall time upper-bound presented in the following lemma:

Lemma 6 *Let $\mu_i \leq K_i \cdot r_{max}$. If $P - K_i L_{min} < K_i(m - 1)L_{max}$, the worst-case stall time of a job of task i that executes for up to r_{max} regulation periods, ignoring any regulation stall upon its first memory access, is upper bounded by:*

$$stall_i^c(K_i, r_{max}) = \begin{cases} \mu_i(m - 1)L_{max} & \text{if } \mu_i \leq r_{max}a_{r0_bar} \\ (r_{max} - r_0 - r_r)a_{r0_bar}(m - 1)L_{max} \\ \quad + r_0(K - K_i)L_{max} \\ \quad + r_r(P - K_i L_{min}) & \text{otherwise} \end{cases} \tag{2}$$

where:

$$a_0 = \left\lceil \frac{K - K_i}{m - 1} \right\rceil$$

$$a_{r0_bar} = \min(K_i, a_0) - 1$$

$$\Delta_0 = (K - K_i)L_{max} - (a_0 - 1)(m - 1)L_{max}$$

$$\Delta_r = \frac{(P - K_i L_{min}) - (a_0 - 1)(m - 1)L_{max}}{K_i - (a_0 - 1)}$$

$$r_r = \begin{cases} \max(0, \mu_i - r_{max}(K_i - 1)) & \text{if } K_i < a_0 \text{ or } (K_i > a_0 \text{ and } \Delta_0 > \Delta_r) \\ \max\left(0, \left\lfloor \frac{\mu_i - r_{max}(a_0 - 1)}{K_i - (a_0 - 1)} \right\rfloor\right) & \text{if } K_i > a_0 \text{ and } \Delta_0 \leq \Delta_r \\ 0 & \text{if } K_i = a_0 \end{cases}$$

$$r_0 = \begin{cases} 0 & \text{if } K_i < a_0 \\ \max(0, \min(\mu_i - r_{max}(a_0 - 1), r_{max} - r_r)) & \text{if } K_i > a_0 \text{ and } \Delta_0 > \Delta_r \\ \max(0, \min(\mu_i - (r_{max} - r_r)(a_0 - 1) - r_r K_i, r_{max} - r_r)) & \text{if } (K_i > a_0 \text{ and } \Delta_0 \leq \Delta_r) \text{ or } K_i = a_0 \end{cases}$$

Proof See Appendix A. □

Note that the first case in (2) corresponds to the case where the bound given by Observation 1 provides the tightest worst-case upper bound. The second case in (2) is similar to (1), in that it has three terms each of which has one of the three upper bounds derived in Sect. 5 as a factor.

By using Lemmas 5 and 6, we derive the worst-case memory stall time for a job of task i that runs for at most r_{max} regulation periods on a core whose memory budget is K_i .

Theorem 7 *In a system that uses round-robin to schedule memory transactions, the worst-case regulation-induced stall-time for a job of task i that performs at most μ_i memory accesses, and runs for at most r_{max} regulation periods on a core that was assigned a budget of K_i memory accesses per regulation period, P , with $\mu_i \leq K_i \cdot r_{max}$, is upper bounded by:*

$$\begin{aligned} \text{stall}_i(K_i, r_{max}) = & (P - K_i L_{min}) \\ & + \begin{cases} \text{stall}_i^r(K_i, r_{max}) & \text{if } P - K_i L_{min} \geq K_i(m - 1)L_{max} \\ \text{stall}_i^c(K_i, r_{max}) & \text{otherwise} \end{cases} \end{aligned} \tag{3}$$

where:

$\text{stall}_i^r(K_i, r_{max})$ is given by (1)

$\text{stall}_i^c(K_i, r_{max})$ is given by (2)

Proof By Observation 2, a job of task i may suffer a regulation stall upon its first memory access, which may occur immediately after being scheduled. By Lemma 3, this stall time is upper bounded by the first term in (3).

If $P - K_i L_{min} \geq K_i(m - 1)L_{max}$, by Lemma 5, $\text{stall}_i^r(K_i, r_{max})$, see (1), is an upper bound on the worst-case stall time suffered by a job of task i that runs for at most r_{max} regulation periods, ignoring the regulation stall upon the first memory access.

If $P - K_i L_{min} < K_i(m - 1)L_{max}$, by Lemma 6, $\text{stall}_i^c(K_i, r_{max} S)$, see (2), is an upper bound of the worst-case stall time suffered by job of task i that runs for at most r_{max} regulation periods, ignoring the regulation stall upon the first memory access. □

6 Schedulability analysis

In this section, we integrate the stall-time term, derived in the previous section, into the schedulability analysis. In Sect. 6.1, we consider the schedulability of a task under EDF. This is then used in Sect. 6.2 to size the execution time budgets of the periodic EDF servers. We choose EDF, because it allows us to apply readily the approach by Sousa et al. (2014). However, the stall analysis in the previous section is independent of EDF and can be integrated with other scheduling policies such as fixed priority.

6.1 Demand bound function and preemptions

Exact schedulability analysis for sporadic tasks scheduled on a uniprocessor using EDF can be done using the demand bound function (Baruah et al. 1990):

$$dbf(t) = \sum_{i=1}^n dbf(\tau_i, t) = \sum_{i=1}^n \max \left(0, 1 + \left\lfloor \frac{t - D_i}{T_i} \right\rfloor \right) \cdot C_i$$

On a multi-core platform, we also need to take into account the maximum memory stall time that a job can incur and that arises from sharing the memory system among the cores. Since this stall time may be incurred on every job, it can be viewed as part of the computation demand of the respective task; indeed, while a job on core i is stalled, no other job on the same core may execute. Thus the effective computation demand of task i is:

$$C_i + stall_i \left(K_i, \left\lceil \frac{D_i}{P_i} \right\rceil + 1 \right) \quad (4)$$

where $stall_i \left(K_i, \left\lceil \frac{D_i}{P_i} \right\rceil + 1 \right)$ is given by (3) derived in the previous section, and is the worst-case stall time of a job of task i that completes by its deadline, since the execution window of such a job, i.e. the time interval between its release and its deadline, spans over at most $\left\lceil \frac{D_i}{P_i} \right\rceil + 1$ regulation periods.

Thus, given a set of n tasks τ_i mapped to a core with a memory bandwidth K_i , if:

$$\sum_{i=1}^n \max \left(0, 1 + \left\lfloor \frac{t - D_i}{T_i} \right\rfloor \right) \cdot \left(C_i + stall_i \left(K_i, \left\lceil \frac{D_i}{P_i} \right\rceil + 1 \right) \right) \leq t, \forall t$$

then the task set is schedulable under EDF. Indeed, if this inequality is satisfied, all jobs of all tasks in the set will meet their deadlines even when each of these jobs suffers $stall_i \left(K_i, \left\lceil \frac{D_i}{P_i} \right\rceil + 1 \right)$, given by (3), which upper bounds the regulation-induced stall of any job of task i , assuming that the job completes by its deadline. If the above inequality is not satisfied, then some job may miss its deadline, and $stall_i \left(K_i, \left\lceil \frac{D_i}{P_i} \right\rceil + 1 \right)$ may underestimate the regulation-induced stall of that job, because its execution might span over more than $\left\lceil \frac{D_i}{P_i} \right\rceil + 1$ regulation periods. In fact,

this case is not an issue, as dbf-based uniprocessor EDF analysis is sustainable, the task set is not schedulable anyway.

Actually, the derivation of (3) does not consider all the stall time that a task i may suffer upon being preempted by other tasks running on the same core. Under (partitioned) EDF, a job can be preempted by jobs on the same core whose deadlines are earlier. With memory regulation, every preemption may lead to one additional regulation stall, which occurs if the core's memory budget is exhausted immediately before resumption of a preempted job. A common approach, see e.g. Liu (2000), Brandenburg (2011), Souto et al. (2015), to take into account overheads that occur on every preemption of a preempted task is to add them to every job of the preempting task. Thus, by Lemma 3, we add $P - K_i L_{min}$ to (4), obtaining the effective computation demand of task i :

$$C'_i(K_i) = C_i + stall_i \left(K_i, \left\lceil \frac{D_i}{P_i} \right\rceil + 1 \right) + (P - K_i L_{min}) \quad (5)$$

where the last term in (5)'s right-hand side upper bounds the stall that job i can cause when it preempts.

6.2 Server sizing

As discussed, the execution budget X_k of a periodic EDF server \tilde{P}_k should be large enough to ensure the schedulability of the tasks it serves, but ideally, no larger than that, or else this would be wasteful of processing capacity. In this subsection we discuss our approach to the optimal sizing of these execution budgets. It extends the approach by Sousa et al. (2014), which considered the exact same problem of sizing a periodic EDF server, albeit in the absence of memory access regulation and without consideration of the effects of memory contention—and in the different context of semi-partitioned scheduling.

Sousa et al.'s approach was a form of sensitivity analysis and relied on the classic dbf-based analysis (Baruah et al. 1990), with the following twist. To equivalently model the execution of tasks inside a fixed-budget server as execution on a uniprocessor [i.e., without the use of servers, which is the model in Baruah et al. (1990)], the time intervals corresponding to the idleness of the server (which are periodic and of fixed length) are modelled as jobs by an interfering periodic high-priority task. The attributes of this “fake task” τ_f [(as called in Sousa et al. (2014)] were $C_f = D_f = S - X_k$ (i.e, the portion of the server period that the server is idle) and $T_f = S$ (i.e, equal to the server period). These attributes jointly ensure that τ_f always has priority over any real task under pure EDF scheduling rules, which allow reuse of the schedulability test from Baruah et al. (1990). Then, consistently with the fact that uniprocessor EDF is a sustainable algorithm (Baruah and Burns 2006), sizing the server becomes equivalent to maximizing the execution time of the fake task, in the modified task set, without compromising schedulability, under some form of sensitivity analysis. In further adapting the schedulability testing technique by Sousa et al., for all the actual tasks served by \tilde{P}_k we consider the effective computation demand of each task, C'_i (rather than the WCET in isolation C_i). Additionally, since we impose/assume that

both the slot size and the servers' sizes are multiples of the regulation period P , the reservation of the fake task C'_f , must also be a multiple of the regulation period. For the same reason, and given that the start and end of every server is synchronised with the regulation period, switching from one server to another one does not cause any stall.

In Sect. 6.1, the upper bound on the stall time assumes that a job may execute in every regulation period from its release until its deadline. Since a job may execute only inside the confines of its server, it follows that the value of r_{max} used as second argument of the *stall()* term in (5) should be adjusted to the value given by the following lemma.

Lemma 8 *The maximum number of regulation periods during which a schedulable job of task i of server k executes is:*

$$r_{max} = \left\lfloor \frac{D_i}{S} \right\rfloor \frac{X_k}{P} + \min \left(\left\lfloor \frac{D_i \bmod S}{P} \right\rfloor + 1, \frac{X_k}{P} \right) \quad (6)$$

Proof The maximum number of regulation periods contained within the time window of execution of server k during any interval of duration D_i occurs when the interval begins in the first regulation period of that server's execution time window.

The number of whole server periods contained in this interval is $\left\lfloor \frac{D_i}{S} \right\rfloor$. Because the length of a server's time window of execution is a multiple of the regulation period, the number of regulation periods of server \tilde{P}_k in each of these time windows is $\frac{X_k}{P}$. The number of regulation periods in the last time window for the server's execution is: $\left\lfloor \frac{D_i \bmod S}{P} \right\rfloor + 1$. Out of these, server k can execute for at most $\frac{X_k}{P}$ regulation periods. Therefore, the maximum number of regulation periods during which a schedulable job of task i of server k is r_{max} as given by (6). \square

Because (6) depends on X_k , and the latter depends on the stall term of each task of server k , which depend on r_{max} —see (1) in Lemma 5 and (2) in Lemma 6,—to size server k for a given number of memory accesses K_k per regulation period P , we use a fixed point computation. In the first iteration, X_k is computed using, for each task of server k , its maximum regulation stall computed assuming that task i of server k may execute for $r_{max} = \left\lfloor \frac{D_i}{P} \right\rfloor + 1$ regulation periods, which is an upper bound on the number of regulation periods a job of task i that meets its deadline may execute. In each of the following iterations, we compute X_k using the maximum stall of each task computed with r_{max} as given by (6) for the value of X_k computed in the previous iteration. We stop the computation in the first iteration when the X_k value computed is equal to that computed in the previous iteration.

6.3 Per server memory budgets

For better resource utilisation, it is important to be able to assign different memory budgets to different servers independently of the core where they execute. For example, this is necessary to implement schedules as the one illustrated in Fig. 1. This means

that one may be able to, on the same system, run more applications, or, for a given set of applications, buy a cheaper platform.

The schedulability analysis and the server sizing presented earlier in this section is oblivious to the memory budgets assigned to the other servers assigned to the same core. Thus, supporting per server memory budgets is only an implementation issue. In the remainder of this section, we outline a possible implementation.

Without memory regulation, a server k may be implemented using (1) the data structures, e.g. the runnable tasks queue, typically used in the implementation of the scheduling policy on a uniprocessor; and (2) a timer, to switch from one server to the next, that is set to X_k when server k becomes active. Upon expiration of the timer, the scheduler preempts the currently running job, if any, switches from the runnable queue of the current server to the runnable queue of the next server, resets the timer to that server's size, and picks the runnable job of the new server according to the scheduling policy of the server. An implementation similar to this one for EDF servers in the context of semi-partitioned scheduling is described in Souto et al. (2015).

This implementation can be easily extended to support memory regulation, when the size of the servers is an integer multiple of the regulation period, P , and synchronized with it, by using an approach similar to that of MemGuard (Yun et al. 2013). MemGuard uses a per-core regulator that relies on hardware performance monitoring counters. At the beginning of each regulation period, MemGuard configures the hardware counter of each core to generate an exception when the core completes some threshold DRAM memory accesses. Upon such an exception, MemGuard idles the respective core until the end of the regulation period. Thus by using different thresholds for different cores, it is possible to assign the memory bandwidth unevenly across the cores. Likewise, by configuring at the beginning of each regulation period the hardware performance monitoring counter of a given core with a different threshold, depending on the server to which the regulation period belongs, it is possible to assign different memory budgets to different servers on the same core.

7 ILP formulation

In this section, we develop an ILP formulation of our server scheduling problem. We assume that each IMA application is mapped to a corresponding server to ensure isolation.

The k -th server, \tilde{P}_k , has a budget (K_k, X_k) , where K_k is the server's memory budget per regulation period, P , and X_k is the server's execution budget per server period S . K_k is the independent variable and X_k is determined by K_k , i.e. $X_k = f(K_k)$ ⁵, so that all tasks in server k meet their deadlines when the memory bandwidth assigned to the server is K_k .

Our problem is, for each server, to choose a value of K_k (and therefore X_k) and to map it to one of the system's m cores, such that both the execution demand and the memory bandwidth demand of all servers do not exceed the system's capacity.

⁵ We do not know whether there is a closed-form expression for function $f(K_k)$.

Table 3 List of all decision variables

Symbols	Decision variable description
q_{ijk}	$q_{ijk} = 1$ (binary), if server k is mapped to a core j on quantum i , and 0 otherwise.
c_{jk}	$c_{jk} = 1$ (binary), if server k is mapped to core j , and 0 otherwise.
f_{ik}	$f_{ik} = 1$ (binary), if i is the first execution quantum of server k , and 0 otherwise.
l_{ik}	$l_{ik} = 1$ (binary), if i is the last execution quantum of server k , and 0 otherwise.
K_k	K_k (ranges between 1 and K) is the memory budget allocated to server k (i.e. is the number of memory accesses allowed in MemGuard’s replenishment period).
X_k	X_k (ranges between 1 and Q) is the execution budget allocated to server k .
x_{kv}	$x_{kv} = 1$ (binary), if the v^{th} point of the $X_k = f(K_k)$ function is selected for server k , and 0 otherwise.

Table 4 Range of all indices used in decision variables

Index	Represents	Range
i	Time quantum	$i \in \{0, 1, \dots, Q - 1\}$
j	Core	$j \in \{0, 1, \dots, m - 1\}$
k	Server	$k \in \{0, 1, \dots, N - 1\}$
v	Sampled point	$v \in \{0, 1, \dots, \beta - 1\}$

Objective function We formulate this problem as a feasibility problem because it is not clear which objective should be used in this context. Indeed, roughly, the smaller a server’s memory budget, the larger its execution budget, i.e. the smaller K_k the larger X_k . Thus, minimising the memory bandwidth may lead to “maximising” the execution budget, which may not be appropriate in all scenarios. The converse is also true. Nevertheless, the designer may pick any objective function that matches the needs of the particular system.

Constraints In the rest of the section, we explain the different constrains used to map the servers on the given platform. All decision variables used in this formulation are presented in Table 3. The notation and range of the indices used in the decision variables are given in Table 4 for quick reference.

Two key issues in the formulation are:

1. Time representation, i.e. whether discrete or continuous;
2. Representation of the $X_k = f(K_k)$ function.

Regarding (1), we have chosen to quantize the length S of the server period in Q fixed-duration time quanta. The time quantum is represented with an index i in this section. Furthermore, the size of each server, X_k , is a multiple of a time quantum. The reason for this choice is to allow a fairly straightforward implementation of the model, more specifically of the constraints related to the system’s memory bandwidth,

in commonly available ILP-tools. To prevent interference among servers, the duration of each time quantum, $\frac{S}{Q}$, is a multiple of the memory regulation period, P , and is synchronised with that period.

Regarding (2), since we have not derived a closed form expression of $X_k = f(K_k)$, we sample a set of K_k values⁶ in a given interval, and, for each value of this set, we compute the corresponding X_k , using the server sizing technique described in Sect. 6.2. This set of sampled values is provided as an input to the ILP solver and it outputs a single budget pair (K_k, X_k) for each server k . In order to find a single budget pair, we use a binary variable x_{kv} defined as follows.

$$x_{kv} = \begin{cases} 1, & \text{if the } v\text{th point of the } f(K_k)\text{function is} \\ & \text{selected for server } k \\ 0, & \text{otherwise} \end{cases}$$

By using this binary variable, we derive the following set of constraints;

$$K_k = \sum_{\forall v} x_{kv} K_{kv}, \forall k \tag{7}$$

$$X_k = \sum_{\forall v} x_{kv} X_{kv}, \forall k \tag{8}$$

$$\sum_{\forall v} x_{kv} = 1, \forall k \tag{9}$$

where, K_{kv} is a constant and $X_{kv} = f(K_{kv})$ is also a constant and is computed from $K_{kv} \in [0, K]$, using the server sizing technique described in Sect. 6.2. Basically, the constraints (7) (8) (9) ensure that, for each server k , the ILP solver will output one point of the set of points (K_{kv}, X_{kv}) provided as inputs, if a solution is found.

We need to ensure that the servers are allocated to different cores in such a manner that the total memory budget at any time and the execution budgets on any core do not exceed the memory system’s bandwidth and the timeslot length, respectively. Because of time quantization, we define another set of key decision variables as follows.

$$q_{ijk} = \begin{cases} 1, & \text{if server } k \text{ is mapped to core } j \text{ on quantum } i \\ 0, & \text{otherwise} \end{cases}$$

The values of these variables determine which servers are mapped to which cores at which quanta of a time slot. For the sake of clarity, in the following we use i , j and k as indices over the time quanta, the cores and the servers, respectively, as summarized in Table 4.

The solution space is defined by two sets of constraints, in addition to the one described above to represent the dependence of X_k on K_k . The first set ensures that

⁶ For simplicity, we use equally spaced samples in a given interval. However, our approach does not depend on this assumption and will work with any set of input samples (integers) within a given interval.

the scheduling of the servers and their bandwidth allocation is such that at any quantum of a time slot the memory bandwidth demand does not exceed the system’s memory bandwidth. The second set ensures that all servers are allocated their execution budget, X_k , i.e. a set of consecutive quanta, on a single core.

The first set of constraints is composed of constraints:

$$\sum_{\forall j} \sum_{\forall k} q_{ijk} K_k \leq K, \forall i \tag{10}$$

Essentially, this set of constraints states that, for any time quantum, the sum of the memory bandwidth allocated to each core ($\sum_{\forall k} q_{ijk} K_k$) does not exceed the bandwidth of the memory system.

Although the use of discrete time simplifies the first set of constraints, ensuring that each server is allocated its execution budget becomes less intuitive. First, we define binary variables f_{ik} to identify the first and l_{ik} to identify the last quantum of each server;

$$f_{ik} = \begin{cases} 1, & \text{if } i \text{ is the first execution quantum of server } k \\ 0, & \text{otherwise} \end{cases}$$

$$l_{ik} = \begin{cases} 1, & \text{if } i \text{ is the last execution quantum of server } k \\ 0, & \text{otherwise} \end{cases}$$

and add the following constraints.

$$\sum_{\forall i} f_{ik} = 1, \forall k \tag{11}$$

$$\sum_{\forall i} l_{ik} = 1, \forall k \tag{12}$$

$$\sum_{\forall i} (i \times l_{ik}) - \sum_{\forall i} (i \times f_{ik}) + 1 = X_k, \forall k \tag{13}$$

These constraints ensure that each server has only one first quantum (11) and one last quantum (12), and that the number of quanta between them is equal to the server’s execution budget (13). Since a quantum should not be allocated to a server before its first quantum or after its last quantum, we add these additional constraints.

$$f_{ik} = 1 \Rightarrow \sum_{\ell=0}^{i-1} q_{\ell jk} = 0, \forall i, j, k \tag{14}$$

$$l_{ik} = 1 \Rightarrow \sum_{\ell=i+1}^{Q-1} q_{\ell jk} = 0, \forall i, j, k \tag{15}$$

Finally, we need to ensure that each server is mapped only to one core and that it is allocated its execution budget on that core. Thus, we add yet another set of binary variables, c_{jk} :

$$c_{jk} = \begin{cases} 1, & \text{if server } k \text{ is mapped to core } j \\ 0, & \text{otherwise} \end{cases}$$

and the following set of constraints:

$$\sum_{\forall j} c_{jk} = 1, \forall k \tag{16}$$

$$c_{jk} = 1 \Rightarrow \sum_{\forall i} q_{ijk} = X_k, \forall j, k \tag{17}$$

$$\sum_{\forall i} \sum_{\forall j} q_{ijk} = X_k, \forall k \tag{18}$$

$$\sum_{\forall k} q_{ijk} \leq 1, \forall i, j \tag{19}$$

In this set of constraints, (16) ensures that a server is mapped to just one core. Constraint (17) ensures that the number of time quanta allocated for a server on the core where it is mapped is equal to that server’s execution budget. Constraints (17) and (18) together ensure that a server is not allocated any time quanta on any core other than the one it is mapped to. Finally, (19) disallows sharing among servers of any time quanta on any core. In other words, a single time quantum of a core cannot be allocated to more than one server.

Table 5 summarizes all constraints for quick reference. Note that, constraints (10), (14), (15) and (17) are not linear. The first because it comprises products of decision variables, whereas the remainder because they include an implication. Thus, strictly, this formulation is not a MILP. However, there are well known transformations (Coelho 2013; Grant 2015) to linearise these kinds of non-linear constraints. We chose to present the non-linear versions above for the sake of clarity and ease of readability. In Appendix B, we present the linearisation of these constraints.

8 Evaluation

For evaluation, we compare the schedulability of our proposed approach with the even bandwidth allocation⁷ described in Mancuso et al. (2015). We also evaluate the effect of memory and time quantization in the ILP model both on the schedulability and on the time needed to find a schedule, if one exists.

⁷ The memory bandwidth is equally divided among cores and stall analysis is performed through our proposed approach.

Table 5 List of all constraints with their description

Iterated variables	Expression for the constraint	Description of constraints
$\forall k$	$K_k = \sum_{\forall v} x_{kv} K_{kv}$	
$\forall k$	$X_k = \sum_{\forall v} x_{kv} X_{kv}$	Select a single budget pair (K_{kv}, X_{kv}) , from a set of input budget pairs
$\forall k$	$\sum_{\forall v} x_{kv} = 1$	
$\forall i$	$\sum_{\forall j} \sum_{\forall k} q_{ijk} K_k \leq K$	For any time quantum, the sum of memory budgets allocated to each core does not exceed the total memory bandwidth
$\forall k$	$\sum_{\forall i} f_{ik} = 1$	Each server has only one first time quantum
$\forall k$	$\sum_{\forall i} l_{ik} = 1$	Each server has only one last time quantum
$\forall k$	$\sum_{\forall i} (i \times l_{ik}) - \sum_{\forall i} (i \times f_{ik}) + 1 = X_k$	For each server, the number of time quanta between the first and the last quantum (including first and last) is equal to the server’s execution budget
$\forall i, j, k$	$f_{ik} = 1 \Rightarrow \sum_{\ell=0}^{i-1} q_{\ell jk} = 0$	A server is not allocated any time quantum before first quantum
$\forall i, j, k$	$l_{ik} = 1 \Rightarrow \sum_{\ell=i+1}^{Q-1} q_{\ell jk} = 0$	A server is not allocated any time quantum after last quantum
$\forall k$	$\sum_{\forall j} c_{jk} = 1$	A server is mapped to just one core
$\forall j, k$	$c_{jk} = 1 \Rightarrow \sum_{\forall i} q_{ijk} = X_k$	A server is allocated its execution budget on the server it is mapped to
$\forall k$	$\sum_{\forall i} \sum_{\forall j} q_{ijk} = X_k$	Each server is allocated to at most one core (this is ensured together with the previous constraint)
$\forall i, j$	$\sum_{\forall k} q_{ijk} \leq 1$	A single time quantum cannot be allocated to more than one server

8.1 Experimental setup

For this evaluation, we developed a two-module Java tool. Its source code can be found at Awan (2016). The first module generates a synthetic task-set, which it uses to construct the IMA applications or servers, and computes the server budgets. The second module generates an ILP model for these servers on a multi-core platform and invokes an ILP solver to map the servers on the cores and to generate a cyclic schedule for each core, as shown in Fig. 1.

The task-set is generated for a given target normalised utilisation (in isolation) using the UUnifast-discard algorithm (Bini and Buttazzo 2009; Davis and Burns 2009) to allow unbiased distribution of the target utilisation. We limit the utilisation of

individual tasks, u_i , to 0.5, by halving the utilisation value output by UUnifast-discard, to avoid too many infeasible task-sets.⁸ Task periods are generated with a log-uniform distribution in the range 20–200 ms. The deadline is set equal to the period, even though our analysis holds for arbitrary deadlines. A task's WCET C_i is derived by multiplying its utilisation u_i and period T_i .

The residual memory accesses of a task are computed by multiplying its C_i with a uniformly distributed random number generated in the range $[0.2\mathcal{E}\alpha, 1.8\mathcal{E}\alpha]$. The “referenced ratio” $\mathcal{E} = 7.97\mu\text{s}^{-1}$ is empirically computed from the data of the tracking applications used by Mancuso et al. (2015)⁹. It means that, on average, those applications generate up to $C_i \cdot \mathcal{E}$ memory accesses per job. The *memory intensity factor*, α , is a variable that affords us some control on the memory bus bandwidth utilisation: the higher α the larger that utilisation.

The generated tasks are distributed to applications in a round-robin fashion to construct a set of applications. To compute the applications' execution budgets, we define a *memory quantization factor* β . For each application, we compute a set of execution budgets corresponding to β different values of the memory budget that are equidistant in $(0, K]$. The memory subsystem parameters, taken from Mancuso et al. (2015), are: $P = 1000\mu\text{s}$, $L_{\min} = 0.0238\mu\text{s}$, $L_{\max} = 0.0497\mu\text{s}$, $K = 20132$.

For each set of input parameters, we generate 100 task sets and compute their respective budgets. We use independent pseudo-random generators for the utilisations, minimum inter-arrival time/deadlines and residual memory accesses, and reuse the seeds in successive replications.

The second module implements the ILP formulation for an input task-set on IBM ILOG CPLEX v12.6.3 and interfaces it with the Java tool using Concert technology. We have control over the number Q of quanta per server period, allowing us to trade off computation time with pessimism. The server period is set to $S = \left\lfloor \frac{\min_{\forall \tau_i \in \tau} \min(T_i, D_i)}{Q} \right\rfloor * Q$. The module can be set to either find an optimal solution, by specifying an appropriate objective function, for example with respect to memory resources, or stop at any feasible solution that satisfies all constraints of the ILP model.

In all experiments, the defaults for the number of cores (m), number of applications (N), number of tasks, target utilisation (without interference), α , β and Q , are 4, $2 \times m$, $2 \times N$, 0.3, 1, 50 and 15, respectively. We used a 2.2 GHz 6-core/12-thread Intel Xeon E5-2420 v2 system with 32 GB of RAM to run all experiments presented below. We have also summarized all the parameters in Table 6.

8.2 Experimental results

The first experiment compares the schedulability ratio of our uneven (general) memory bandwidth allocation policy with that of the even policy. The latter divides the available

⁸ For the parameter set values used in our experiments, most applications with tasks whose utilisation is greater than 0.5 cannot be scheduled on a single core, even when allocated the full memory bandwidth.

⁹ The progressive lockdown curve of the tracking application in Mancuso et al. (2015) shows that 18 locked-down pages offer the best trade-off against the WCET in isolation. The ratio of residual memory accesses (1067882) to the WCET in isolation (133, 989.029 μs) at this point gives $\mathcal{E} = 7.97\mu\text{s}^{-1}$.

Table 6 Overview of parameters

Parameters	Values	Default
Inter-arrival time T_i	20–200 ms	N/A
Referenced ratio (\mathcal{E})	$7.97\mu s^{-1}$	$7.97\mu s^{-1}$
Memory intensity factor (α)	{0.25 : 0.25 : 2}	1
Number of cores (m)	{2, 4}	4
Number of applications/servers	$\{2 \times m\}$	8
Task-set size	{2, 4} per server	2 per server
Memory quantization factor (β)	{20, 40, 50, 60, 80}	50
Quanta (Q)	{5, 10, 15, 20}	15
Utilisation	$\{0.3 \times m\}$	1.2

bandwidth equally to all cores. The applications inside a server are scheduled with the EDF scheduling policy irrespective of whether the server is assigned an even or uneven memory bandwidth. Separate instances of the ILP are run for the two policies (even and uneven), to decide the allocation of servers to cores and, in the case of the uneven policy, also the allocation of bandwidth to the servers. Figures 6 and 7 present the results for platforms composed of two and four cores, respectively. Assuming a balanced platform, we have halved the memory bandwidth of the two core platform. This leads to the following parameters being altered for a two core platform: $L_{min} = 0.0477\mu s$, $L_{max} = 0.0993\mu s$, $K = 10066$. The line plots in Figs. 6 and 7 show the schedulability ratio against different values of memory intensity factor (α). In both cases, the uneven policy leads to improved schedulability over the even policy. For extreme values of α in the four core platform, there is no major difference between the two policies: if the bandwidth utilisation is low (i.e., small α), both policies can schedule most applications, whereas if the bandwidth utilisation is high, both policies are unable to schedule most applications. However, in two core platform, the schedulability ratio of even and uneven approaches tend to depart with an increase in memory intensity factor. *In our simulations, we have significant improvements over the even policy.* For $\alpha = 1$ and $m = 2$, 99% of the task sets are found feasible with uneven policy vs. 77% with even policy. Similarly, for $\alpha = 1$ and $m = 4$, 61% of the task sets are found feasible with uneven policy vs. 37% with even policy.

Figures 6 and 7 also show the distribution of the time required to find the feasible solutions for the uneven policy via a boxplot drawn with MatLab (MathWorks Box plots documentation. <http://www.mathworks.com/help/stats/box-plots.html>) for platforms that consist of two cores and four cores, respectively. The red '+' show the outliers. All times presented are the wall-times reported by CPLEX. The time to compute the feasible solution is negligible for the two core platform as the search space for the ILP formulation is very small. In all cases, the system was able to find the feasible solution within two and half seconds. For a four core platform (Fig. 7), in the majority of cases, the ILP solver needs a few minutes (less than twenty minutes) to find a feasible solution except one outlier (not shown in Fig. 7) that takes less than three hours to complete. The computation time increases with the memory intensity factor, as it becomes harder to meet the memory bandwidth and computation time constraints.

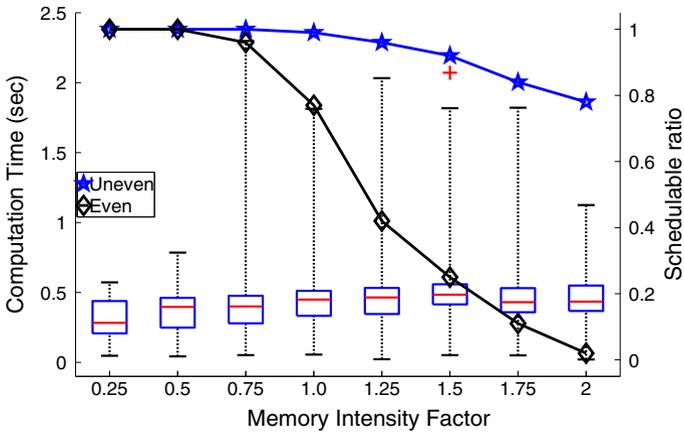


Fig. 6 Effect of the memory intensity factor α on schedulability and the distribution of time to find feasible solutions for uneven policy ($\beta = 50$, $Q = 15$, $m = 2$ and 2 tasks per server)

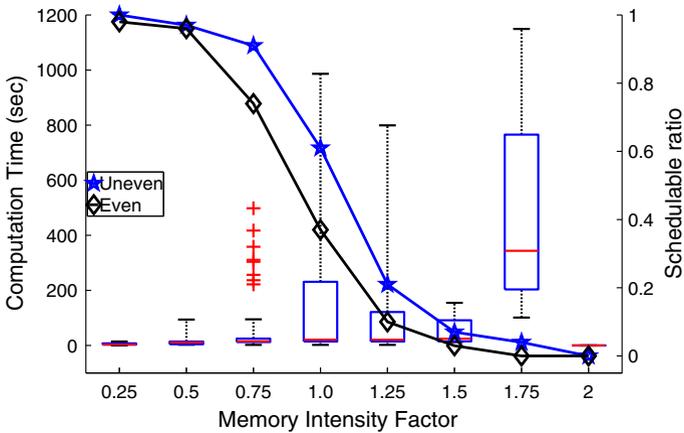


Fig. 7 Effect of the memory intensity factor α on schedulability and the distribution of time to find feasible solutions for uneven policy ($\beta = 50$, $Q = 15$, $m = 4$ and 2 tasks per server)

However, after a given value of α (1 in our experiments), the solver efficiently prunes the search space and the time to find a feasible solution starts to decrease. It spikes again at $\alpha = 1.75$ though.

These results show that the uneven policy improves the schedulability with respect to the even policy. Furthermore, they show that our ILP model is practical for platforms with up to 4 cores. However, it is likely not to be computationally tractable for platforms with 8 cores, and hence heuristics may be required. An alternative, is to explore the specifics of each platform and use a hybrid approach. For example, in the case of platforms that use two memory controllers, such as the Freescale P4080, one might partition the cores in two sets of 4 cores and map each controller to a different set of 4 cores. Furthermore, the set of IMA applications might be partitioned in two, using some heuristic, and a feasible schedule for each one of these sets of IMA applications

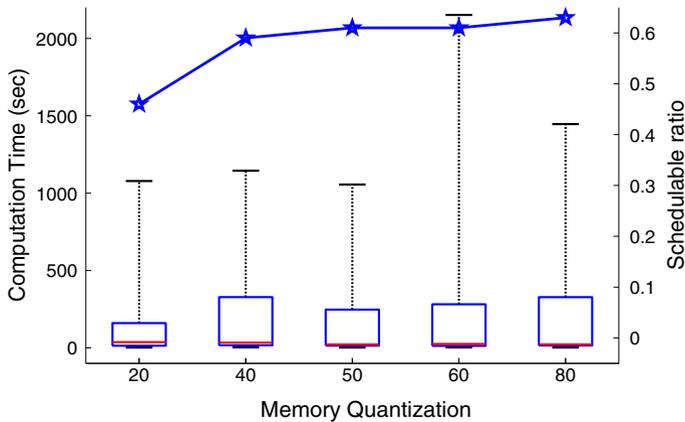


Fig. 8 Effect of the memory quantization parameter β on schedulability and the distribution of time to find feasible solutions ($\alpha = 1$, $Q = 15$, $m = 4$ and 2 tasks per server)

could be found using the ILP model. By the way, there is no indication that platforms with more than 4 cores per memory controller will become the norm.

We next evaluate the effect of quantization of the server budgets on the schedulability and on the time to find a feasible solution for a platform with four cores. Figure 8 considers the memory quantization parameter β (x-axis). As seen, the schedulability ratio (right y-axis) increases with finer quantization granularity. For $\beta = 20$, the schedulability is relatively low because the low resolution of the memory bandwidth assignment leads to some over-allocation of this resource, which is only partially compensated for by the possibility of a lower memory stall (and consequently a lower WCET) afforded by the over-allocation of memory bandwidth. As β increases, and the memory bandwidth over-allocation drops, the schedulability improves rapidly, but then levels off at around 60%. We conjecture that for this value the schedulability ratio approaches the schedulability ratio without memory quantization. The time to find a feasible solution is also in the order of a few minutes, except for some outliers, which still take less than one hour. *The boxplots show no tractability issues as β increases, despite the increase in the search space.*

Figure 9 shows the effects of time quantization, i.e. the number of quanta Q per server period (x-axis) for a platform composed of four cores. The schedulability ratio (right y-axis) increases with Q , like it did with β in Fig. 8. Indeed, as Q increases, the granularity of the servers' execution budgets becomes finer, and the over-allocation is reduced. However, this comes at a cost of more computing time to find a feasible solution, as shown by the boxplots. The increase in the time to find a solution is clearly super-linear on the value of Q , but the time needed is still below two and half hour per instance for the parameters' ranges considered. The rate of increase of the schedulability is much smaller for $Q > 10$ than for lower values, but the plot shows no levelling-off, for the range of values of Q considered. *Hence, the schedulability ratio along the computation time increases with an increase in Q .* The fact that declaring a task-set infeasible may take more than ten times longer (see Fig. 10) and that we are running 100 repetitions for each set of parameter values, prevented us from showing

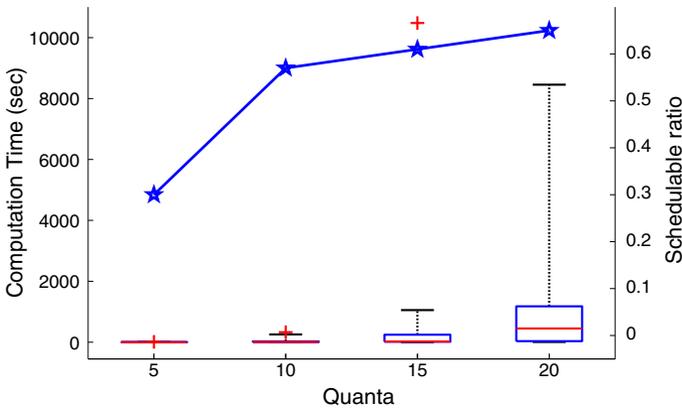


Fig. 9 Effect of Q (number of quanta per server period) on the schedulability ratio and the distribution of time to find feasible solutions ($\alpha = 1, \beta = 50, m = 4$ and 2 tasks per server)

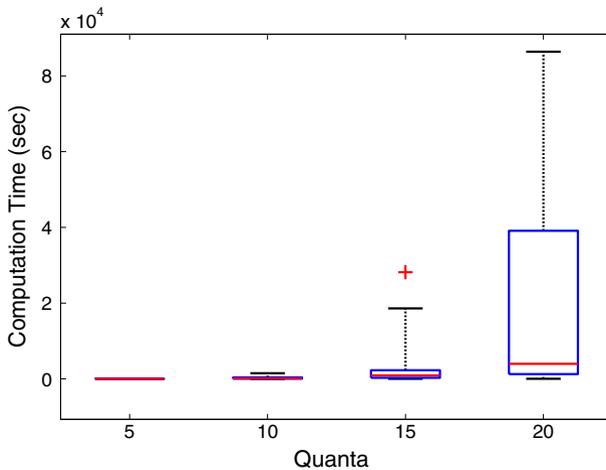


Fig. 10 Effect of Q (number of quanta per server period) on the distribution of time to find infeasible solutions ($\alpha = 1, \beta = 50, m = 4$ and 2 tasks per server)

results for higher Q values. *In practice, where one needs a feasible solution for one particular set of servers, higher Q values are still tractable and it may be worth to trade-off the time to find a solution for the utilisation of the system resources.*

We also present the distribution of the time to find *infeasible* solutions for all experiments in Figs. 10, 11, 12 and 13. The dependence on the different factors is similar to that of the time required to find a feasible solution. The main difference is that the ILP solver takes much longer to declare a task-set infeasible, because it must explore the entire search space rather than stop at the first solution.

Finally, we have performed another set of experiments in which we increases the number of tasks per server to 4. In contrast to previous case (2 tasks per server), doubling the number of tasks per server increases the memory accesses in each server. This

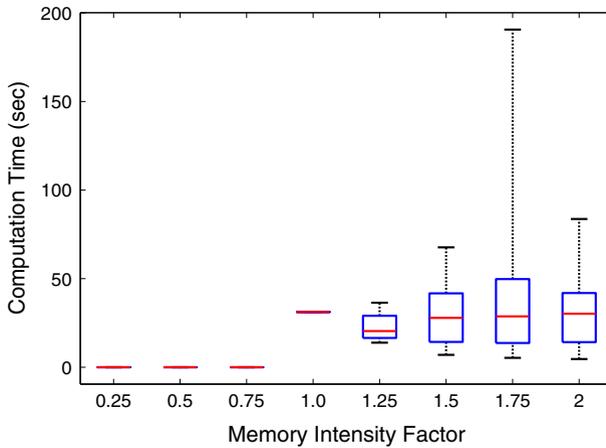


Fig. 11 Effect of the memory intensity factor α on the distribution of time to find infeasible solutions for uneven policy ($\beta = 50$, $Q = 15$, $m = 2$ and 2 tasks per server)

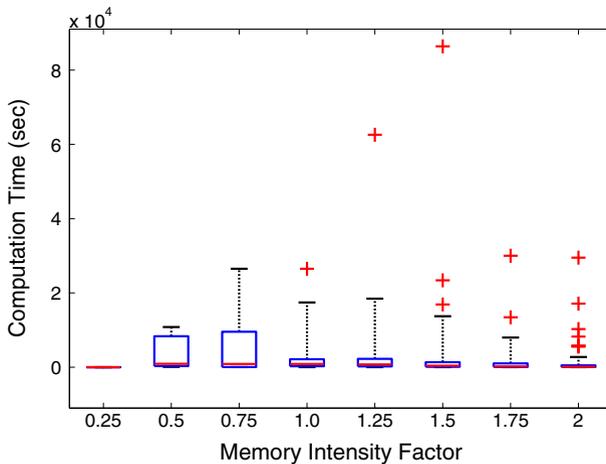


Fig. 12 Effect of the memory intensity factor α on the distribution of time to find infeasible solutions for uneven policy ($\beta = 50$, $Q = 15$, $m = 4$ and 2 tasks per server)

increase affects the schedulability of the system as it becomes harder for ILP to find the feasible solution with increased memory bandwidth requirement. Consequently, the time to compute the feasible solutions on average also increases.

The effect of variation in memory intensity factor on schedulability analysis and computation time of feasible solutions is presented in Figs. 14 and 15 for 2 and 4 cores, respectively. It is evident that the schedulability ratio decreases and the computation time increases due to increase in overall memory bandwidth requirements. The effect of decrease in schedulability is more prominent in 2 cores case as the possibilities of packing the servers is limited. The effect of variation in memory quantization (Fig. 16) and quanta (Fig. 17) on schedulability ratio and computation time of feasible solutions

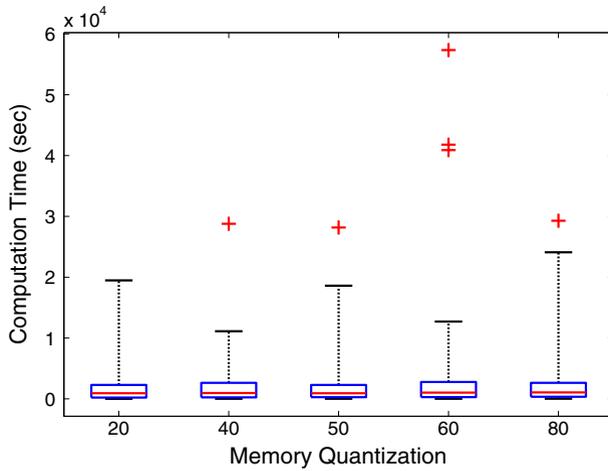


Fig. 13 Effect of the memory quantization parameter β on the distribution of time to find infeasible solutions ($\alpha = 1, Q = 15, m = 4$ and 2 tasks per server)

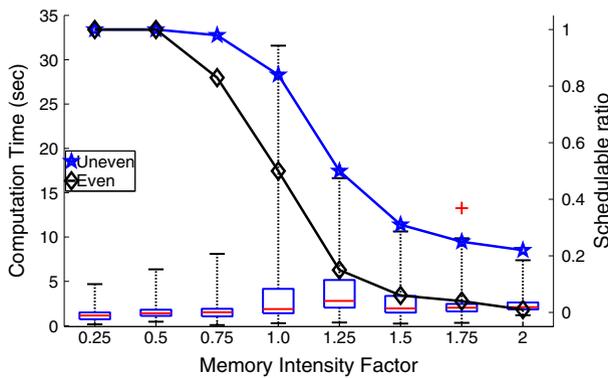


Fig. 14 Effect of the memory intensity factor α on schedulability and the distribution of time to find feasible solutions for uneven policy ($\beta = 50, Q = 15, m = 2$ and 4 tasks per server)

in this experiment shows similar trends except that former is scaled down and latter is scaled up due to the aforementioned reasons. The computation time of the infeasible solutions also scales up in this set of experiments.

9 Conclusions and future work

Sharing memory efficiently among cores in a multi-core platform while ensuring isolation among applications is of utmost importance for the adoption of multi-cores in real-time safety-critical application domains, such as avionics.

This work improves on the state-of-the-art by extending the memory bandwidth reservation mechanism proposed in the scope of Single Core Equivalence (SCE) to

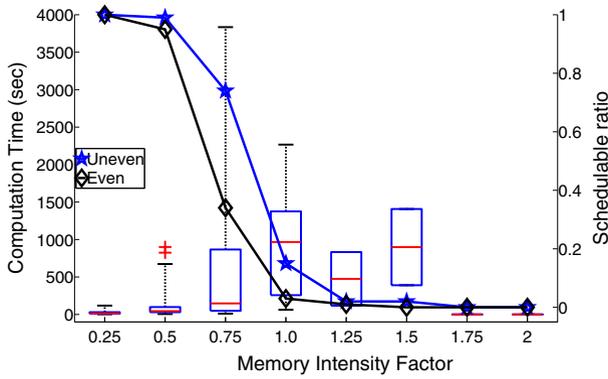


Fig. 15 Effect of the memory intensity factor α on schedulability and the distribution of time to find feasible solutions for uneven policy ($\beta = 50, Q = 15, m = 4$ and 4 tasks per server)

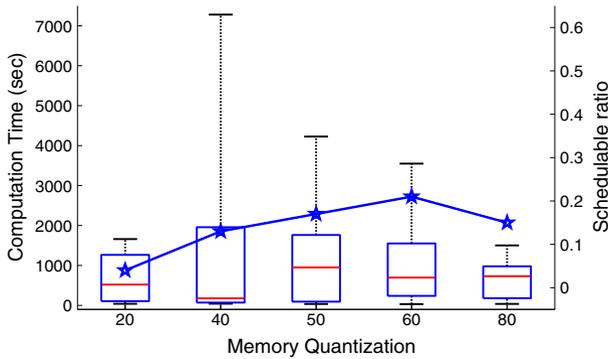


Fig. 16 Effect of the memory quantization parameter β on schedulability and the distribution of time to find feasible solutions ($\alpha = 1, Q = 15, m = 4$ and 4 tasks per server)

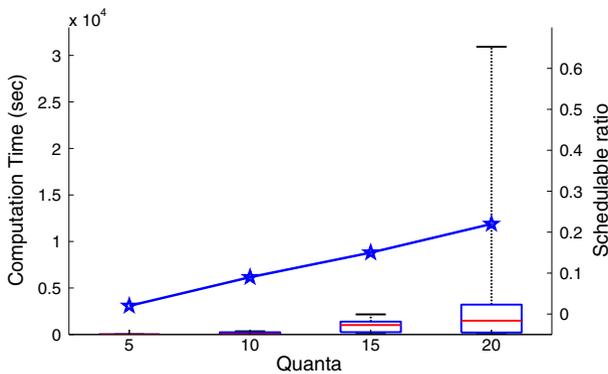


Fig. 17 Effect of Q (number of quanta per server period) on the schedulability ratio and the distribution of time to find feasible solutions ($\alpha = 1, \beta = 50, m = 4$ and 4 tasks per server)

allow uneven memory bandwidth among cores, and variable bandwidth at run time on each core. We formulate new stall time analysis for this memory management arrangement and incorporate it into a schedulability analysis for a server-based approach, which manages both the cores and the memory bus in an integrated way. We assume EDF as the internal server policy, but our approach is easily adaptable to fixed-priority scheduling as well.

Another main contribution of this paper is an ILP model that we developed in order to find a feasible mapping of servers to cores, if one exists, while satisfying the platform's resource capacities. This ILP model relies on our analysis for sizing of the servers' execution and memory budgets. It assumes a single memory controller, but it can be easily generalised for the case of multiple memory controllers partitioned among sets of cores.

Experiments with the ILP model confirm the tractability of the approach for platforms with up to 4 cores and show that the uneven and per-server memory bandwidth allocation can significantly improve the schedulability of sets of applications with realistic resource demand when compared with an even memory bandwidth allocation.

In the future, we would also like to extend our approach to other bus arbitration policies common in COTS platforms such as FR-FCFS.

Acknowledgements This work was partially supported by National Funds through FCT/MCTES (Portuguese Foundation for Science and Technology) within the CISTER Research Unit (CEC/04234).

Proofs of Lemmas in Sect. 5.2

In this section we present the proofs of Lemmas 5 and 6 (For notation, see Table 7.)

Table 7 List of symbols used in the stall term analysis in Sect. 5.2

Symbol	Description
$a_{\bar{r}}$	No. of memory accesses in periods without regulation stall, when the number of regulation stalls is maximum
$a_{\bar{r}0}$	No. of memory accesses in periods both without regulation stall and with less than a_0 memory accesses.
r_0	Number of regulation periods without regulation stalls with at least a_0 memory accesses
r_r	Number of regulation periods with regulation stalls with at least a_0 memory accesses
r_{max}	Maximum number of regulation periods in an execution of a job of task i
$r_{\bar{r}max}$	Maximum number of regulation periods without regulation stall
Δ_0	Additional stall upon the a_0 th access on a regulation period P
Δ_r	Average additional stall per memory access after the $(a_0 - 1)$ th access in a regulation period with a regulation stall

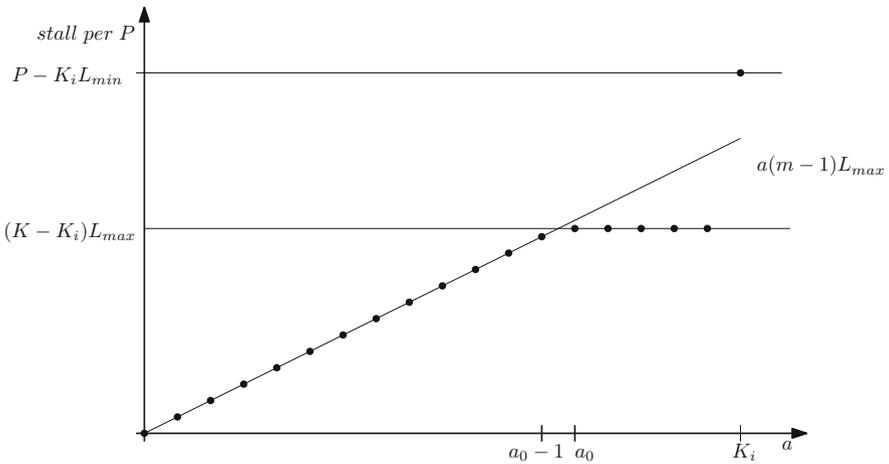


Fig. 18 Stall time in a regulation period as a function of the number of memory accesses, when $P - K_i L_{min} \geq K_i(m - 1)L_{max}$

Lemma 9 Let $\mu_i \leq K_i \cdot r_{max}$. If $P - K_i L_{min} \geq K_i(m - 1)L_{max}$, then the worst-case stall time of a job of task i that executes for up to r_{max} regulation periods, ignoring any regulation stall upon its first memory access, is upper bounded by:

$$\begin{aligned}
 stall_i^r(K_i, r_{max}) = & \left\lfloor \frac{\mu_i}{K_i} \right\rfloor (P - K_i L_{min}) + r_0(K - K_i)L_{max} \\
 & + \min(a_{\bar{r}}, (r_{\bar{r}max} - r_0)(a_0 - 1))(m - 1)L_{max} \quad (20)
 \end{aligned}$$

where:

$$\begin{aligned}
 a_0 &= \left\lceil \frac{K - K_i}{m - 1} \right\rceil \\
 a_{\bar{r}} &= \mu_i - \left\lfloor \frac{\mu_i}{K_i} \right\rfloor K_i \\
 r_0 &= \begin{cases} 0 & \text{if } K_i \leq a_0 \\ \max(\min(a_{\bar{r}} - (a_0 - 1)r_{\bar{r}max}, r_{\bar{r}max}), 0) & \text{otherwise} \end{cases} \\
 r_{\bar{r}max} &= r_{max} - \left\lfloor \frac{\mu_i}{K_i} \right\rfloor \quad (21)
 \end{aligned}$$

Proof If $P - K_i L_{min} \geq K_i(m - 1)L_{max}$, then the worst case occurs for the maximum number of regulation stalls as illustrated by Fig. 18, which plots the stall per regulation period as a function of the number of memory accesses in that period. Note that, by definition of K and given that $L_{min} \leq L_{max}$, it is always the case that $P - K_i L_{min} \geq (K - K_i)L_{max}$.

The maximum number of periods with regulation stalls in any job of task i is $\lfloor \frac{\mu_i}{K_i} \rfloor$. Let r_{max} be the maximum number of regulation periods over which the job can execute. Therefore, when the number of regulation stalls is maximum, (1) the maximum number of periods without regulation stalls is: $r_{\bar{r}max} = r_{max} - \lfloor \frac{\mu_i}{K_i} \rfloor$, if $\mu_i \leq K_i \cdot r_{max}$; and 2) the number of memory accesses in periods without regulation stalls is: $a_{\bar{r}} = \mu_i - \lfloor \frac{\mu_i}{K_i} \rfloor K_i$.

These $a_{\bar{r}}$ accesses will be distributed over several regulation periods. Let r_0 be the number of these regulation periods with at least a_0 memory accesses. Then, by Lemma 1 and Observation 1, the total contention stall time in all periods without regulation stalls is:

$$r_0(K - K_i)L_{max} + \left(a_{\bar{r}} - \sum_{j=1}^{r_0} a_j \right) (m - 1)L_{max} \tag{22}$$

where a_j is the number of memory accesses in regulation period j with at least a_0 memory accesses. To determine the maximum of (22), we consider two cases, depending on whether a_0 is smaller or larger than K_i . Note that, it cannot be $a_0 = K_i$, since in this case, we would have a regulation stall upon the a_0 th access, and therefore all these accesses would be in periods with regulation stalls.

case 1 If $a_0 > K_i$, then the value of (22) is maximum when r_0 is 0.

Indeed, if $a_0 > K_i$ then the core stalls before it performs a_0 accesses. Thus, (22) becomes:

$$\begin{aligned} & a_{\bar{r}}(m - 1)L_{max} && \text{by } r_0 = 0 \\ & = \min(a_{\bar{r}}, (r_{\bar{r}max} - r_0)(a_0 - 1))(m - 1)L_{max} && \text{by } a_0 > K_i \text{ and } r_0 = 0 \end{aligned}$$

Indeed:

$$\begin{aligned} & (r_{\bar{r}max} - r_0)(a_0 - 1) \\ & = r_{\bar{r}}(a_0 - 1) && \text{by } r_0 = 0 \\ & \geq r_{\bar{r}}K_i && \text{by } a_0 > K_i \\ & > a_{\bar{r}} \end{aligned}$$

since, otherwise, there would be at least one regulation stall in the $r_{\bar{r}}$ regulation periods without regulation stalls. Thus, (20) provides an upper bound on the worst-case contention stall time.

Case 2 If $a_0 < K_i$. We consider two further cases depending on the number of memory accesses in periods without regulation stalls.

Case 2.1 If $a_{\bar{r}} \leq r_{\bar{r}max}(a_0 - 1)$, then it is possible to distribute all the $a_{\bar{r}}$ memory over the $r_{\bar{r}}$ regulation periods in such a way that there is less than a_0 memory accesses in each regulation period, which by Lemma 1 leads to the worst-case contention stall

time. Thus, $r_0 = 0$, and therefore, by the same arguments as in Case 1, (20) provides an upper bound on the worst-case contention stall time.

Note also that in this case, $a_{\bar{r}} \leq r_{\bar{r}max}(a_0 - 1)$, the first argument of the $min()$ function in the second case of (21) is negative, and therefore, (21) takes value 0, as it should.

Case 2.2 If $a_{\bar{r}} > r_{\bar{r}max}(a_0 - 1)$, then there must be regulation periods without regulation stalls and with at least a_0 memory accesses.

In this case, the value of (22) is maximum when all regulation periods have at least $a_0 - 1$ memory accesses and we maximize the value of r_0 . Indeed, as illustrated in Fig. 18, in regulation periods without regulation stalls and at least a_0 memory accesses, additional memory accesses do not increase the stall time upper bound. On the other hand, the total stall time on a regulation period increases upon the a_0 th access.

Thus, in this case, the worst case occurs when each memory access above $(a_0 - 1)r_{\bar{r}max}$ occurs in a different regulation period of the $r_{\bar{r}max}$ regulation periods without regulation stalls. However, there are only $r_{\bar{r}max}$ of these regulation periods, hence the second argument of the $min()$ function in the second case of (21). Furthermore, in this case, $a_{\bar{r}} > r_{\bar{r}max}(a_0 - 1)$, the $min()$ function takes a positive value, and therefore r_0 takes the value of the $min()$ function, as it should.

Finally, the first factor of the second term of (22), that is the number of accesses in periods with less than a_0 memory accesses is given by $(r_{\bar{r}max} - r_0)(a_0 - 1)$, which is smaller than $a_{\bar{r}}$ in this case, $a_{\bar{r}} > r_{\bar{r}max}(a_0 - 1)$, since $r_0 \geq 0$. Therefore, (20) provides an upper bound for the worst-case stall-time also in this case. \square

Lemma 10 *Let $\mu_i \leq K_i \cdot r_{max}$. If $P - K_i L_{min} < K_i(m - 1)L_{max}$, the worst-case stall time of a job of task i that executes for up to r_{max} regulation periods, ignoring any regulation stall upon its first memory access, is upper bounded by:*

$$stall_i^c(K_i, r_{max}) = \begin{cases} \mu_i(m - 1)L_{max} & \text{if } \mu_i \leq r_{max}a_{\bar{r}0} \\ (r_{max} - r_0 - r_r)a_{\bar{r}0}(m - 1)L_{max} & \\ \quad + r_0(K - K_i)L_{max} & \\ \quad + r_r(P - K_i L_{min}) & \text{otherwise} \end{cases} \tag{23}$$

where:

$$a_0 = \left\lceil \frac{K - K_i}{m - 1} \right\rceil$$

$$a_{\bar{r}0} = \min(K_i, a_0) - 1$$

$$\Delta_0 = (K - K_i)L_{max} - (a_0 - 1)(m - 1)L_{max}$$

$$\Delta_r = \frac{(P - K_i L_{min}) - (a_0 - 1)(m - 1)L_{max}}{K_i - (a_0 - 1)}$$

$$r_r = \begin{cases} \max(0, \mu_i - r_{max}(K_i - 1)) & \text{if } K_i < a_0 \text{ or } (K_i > a_0 \text{ and } \Delta_0 > \Delta_r) \\ \max\left(0, \left\lfloor \frac{\mu_i - r_{max}(a_0 - 1)}{K_i - (a_0 - 1)} \right\rfloor\right) & \text{if } K_i > a_0 \text{ and } \Delta_0 \leq \Delta_r \\ 0 & \text{if } K_i = a_0 \end{cases}$$

$$r_0 = \begin{cases} 0 & \text{if } K_i < a_0 \\ \max(0, \min(\mu_i - r_{max}(a_0 - 1), r_{max} - r_r)) & \text{if } K_i > a_0 \text{ and } \Delta_0 > \Delta_r \\ \max(0, \min(\mu_i - (r_{max} - r_r)(a_0 - 1) - r_r K_i, r_{max} - r_r)) & \text{if } (K_i > a_0 \text{ and } \Delta_0 \leq \Delta_r) \text{ or } K_i = a_0 \end{cases}$$

Proof If $P - K_i L_{min} < K_i(m - 1)L_{max}$, then the worst-case scenario occurs when the number of contention stalls outside periods with 1) regulation stalls or, by Lemma 1, 2) at least a_0 memory accesses, is maximum.

The maximum number of memory accesses *per regulation period* outside periods with (1) or (2) is given by $a_{r0} = \min(K_i, a_0) - 1$.

Case 1 $\mu_i \leq r_{max} a_{r0}$: In this case, all memory accesses may occur outside periods with (1) or (2) and, by Observation 1, an upper bound of the worst-case stall time is:

$$\mu_i(m - 1)L_{max} \tag{24}$$

case 2 $\mu_i > r_{max} a_{r0}$: In this case, at least one period will have either a regulation stall or at least a_0 memory accesses. Let r_r and r_0 be the number of periods of each, respectively. Then, by Observation 1 and by Lemmas 3 and 1, an upper bound of the stall time is:

$$(r_{max} - r_0 - r_r)a_{r0}(m - 1)L_{max} + r_0(K - K_i)L_{max} + r_r(P - K_i L_{min}) \tag{25}$$

To complete the proof, we need to derive the values of r_0 and r_r . We do that by case analysis, in which we use the following two parameters:

$$\begin{aligned} \Delta_0 &= (K - K_i)L_{max} - (a_0 - 1)(m - 1)L_{max} \\ \Delta_r &= \frac{(P - K_i L_{min}) - (a_0 - 1)(m - 1)L_{max}}{K_i - (a_0 - 1)} \end{aligned}$$

Δ_0 denotes the additional stall time upon the a_0 th memory access, whereas Δ_r is the average stall time per memory access after the $a_0 - 1$ th, when core i has a regulation stall. Figure 19 illustrates the meaning of these parameters.

We consider three cases, depending on the relative values of K_i and a_0 , and we divide one of these cases in two subcases, depending on the relative values of Δ_0 and Δ_r .

Case 2.1 $K_i < a_0$: If $K_i < a_0$ then the core will stall before it performs a_0 memory accesses. Therefore r_0 is always 0, and, by $P - K_i L_{min} < K_i(m - 1)L_{max}$, (25) will be maximum when r_r is minimum. This occurs when all memory accesses are spread by the r_{max} periods in such a way that a regulation stall will occur only when all r_{max} periods have $(K_i - 1)$ memory accesses. Thus, $r_r = \max(0, \mu_i - r_{max}(K_i - 1))$, if $\mu_i \leq K_i(\lceil D_i/P \rceil + 1)$.

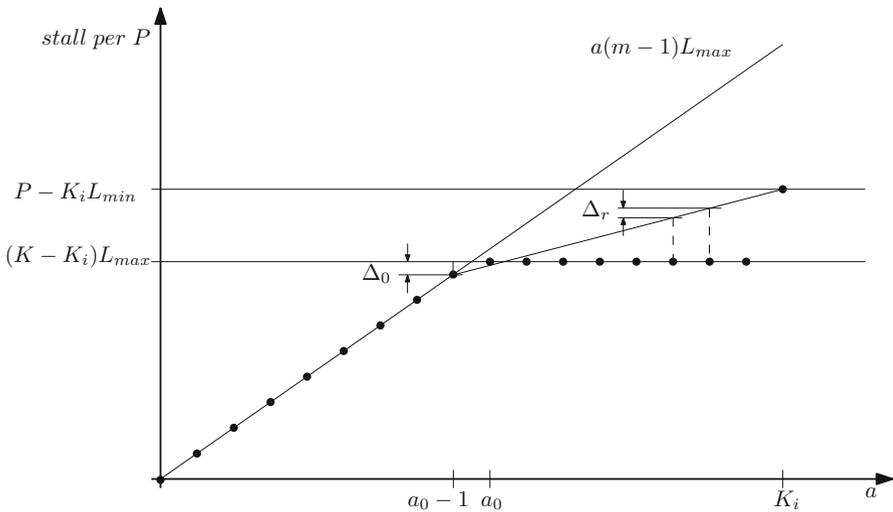


Fig. 19 Stall time in a regulation period as a function of the number of memory accesses, when $\Delta_0 > \Delta_r$

Case 2.2 $K_i > a_0$: By $P - K_i L_{min} < K_i(m - 1)L_{max}$ and by Lemma 1, (25) will be maximum when each regulation period has at least $(a_0 - 1)$ accesses and either (1) r_0 is maximum, or (2) r_r is maximum. This is shown next, using a case analysis depending on the relative values of the parameters Δ_0 and Δ_r defined above.

Case 2.2.2 $\Delta_0 > \Delta_r$: This case is illustrated by Fig. 19, which plots the per regulation period stall as a function of memory accesses per period. By the definition of Δ_0 and Δ_r , the worst case occurs when the accesses above $r_{max}(a_0 - 1)$ are spread over the r_{max} regulation periods so as to maximize r_0 . In this case, r_0 is given by:

$$r_0 = \max(0, \min(\mu_i - r_{max}(a_0 - 1), r_{max} - r_r) \tag{26}$$

Indeed, r_0 cannot be lower than 0, and cannot be larger than $r_{max} - r_r$.

To ensure that r_0 is maximum, r_r must be minimum and, if $\mu_i \leq K_i(\lceil D_i/P \rceil + 1)$, is given by:

$$r_r = \max(0, \mu_i - r_{max}(K_i - 1)) \tag{27}$$

Indeed, r_r cannot always be 0. Once a job performs $K_i - 1$ memory accesses in every regulation period, each additional memory access leads to one additional regulation stall, thus increasing r_r by one.

Note that although Fig. 19 illustrates the case of $P - K_i L_{min} > (K - K_i)L_{max}$, by definition of Δ_0 and Δ_r , $\Delta_0 > \Delta_r$ also when $P - K_i L_{min} = (K - K_i)L_{max}$ and $K_i > a_0$. (Note that, as shown in the proof of Lemma 3, $P - K_i L_{min} \geq (K - K_i)L_{max}$.)

Case 2.2.1 $\Delta_0 \leq \Delta_r$: This case is illustrated in Fig. 20, which plots the per regulation period stall as a function of memory accesses per period. By the definition of Δ_0 and

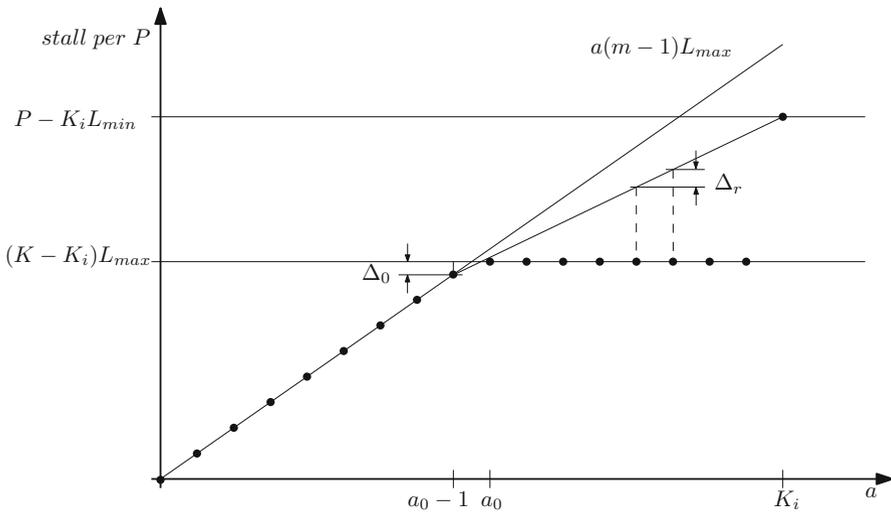


Fig. 20 Stall time in a regulation period as a function of the number of memory accesses, when $\Delta_0 < \Delta_r$

Δ_r , the worst case occurs when the accesses above $r_{max}(a_0 - 1)$ are distributed over the r_{max} regulation periods so as to maximise r_r . Thus, if $\mu_i \leq K_i(\lceil D_i/P \rceil + 1)$, the maximum number of regulation periods with regulation stalls is:

$$r_r = \max\left(0, \left\lfloor \frac{\mu_i - r_{max}(a_0 - 1)}{K_i - (a_0 - 1)} \right\rfloor\right) \tag{28}$$

When all regulation periods have at least $(a_0 - 1)$ memory accesses and r_r is maximum (as given by (28)), (25) will be maximum, if r_0 is also maximised. This means, if there are additional memory accesses that are not enough to cause one additional regulation stall, by Lemma 1, the stall will be maximum if these additional memory accesses are spread over as many regulation periods as possible, rather than clustered into a single regulation period. Thus, if $\mu_i \leq K_i(\lceil D_i/P \rceil + 1)$, the value of r_0 that maximises (25) is:

$$r_0 = \max(0, \min(\mu_i - (r_{max} - r_r)(a_0 - 1) - r_r K_i, r_{max} - r_r)) \tag{29}$$

Case 2.3 $K_i = a_0$: In this case, every access above $r_{max}(a_0 - 1)$ both causes a regulation stall and is the a_0 th access of one regulation period. Thus, for each of these regulation periods, we have two upper bounds on the per regulation period stall: $(K - K_i)L_{max}$, by Lemma 1, and $P - K_iL_{min}$, by Lemma 3. Since we want the tightest upper bound of the total stall, we must use the smallest of these values. Nevertheless, we can still use (25), as for the other cases, as long as we set r_0 and r_r to the appropriate values, derived below.

As shown in Case 2 of Lemma 3, $(K - K_i)L_{max} \leq P - K_iL_{min}$. Therefore, we consider each additional access as leading to the a_0 th access, i.e. $r_r = 0$ and $r_0 = \max(0, \mu_i - r_{max}(a_0 - 1))$, if $\mu_i \leq K_i(\lceil D_i/P \rceil + 1)$.

Note that for $r_r = 0$, the expression of r_0 in Case 2.2, see (29), reduces to the expression of r_0 in this case. Therefore, we also use (29) for this case in the formulation of this lemma, thus avoiding to introduce yet another case in the expression of r_0 . □

Linearisation of constraints

In this section, we present a linearisation of constraints (10), (14), (15) and (17), thus making the ILP model presented in Sect. 7 strictly linear.

$$\sum_j \sum_{\forall k} q_{ijk} K_k \leq K, \forall i \tag{10}$$

or,

$$\sum_j \sum_{\forall k} z_{ijk} \leq K, \forall i$$

$$K_k - K(1 - q_{ijk}) \leq z_{ijk}, \forall i, j, k$$

$$z_{ijk} \geq 0, \forall i, j, k$$

$$z_{ijk} \leq K_k, \forall i, j, k$$

$$z_{ijk} \leq K q_{ijk}, \forall i, j, k$$

$$z_{ijk} \in [0, K]$$

$$X_k \in [0, Q]$$

$$K_k \in [0, K]$$

The z_{ijk} decision variables are the artefacts of the linearisation of (10)—each product $q_{ijk} K_k$ is replaced by a variable z_{ijk} —and therefore it is not defined earlier. The linearisation of constraints (14) and (15) are presented as follows.

$$f_{ik} = 1 \Rightarrow \sum_{\ell=0}^{i-1} q_{\ell jk} = 0, \forall i, j, k \tag{14}$$

or,

$$(i \times f_{ik}) + \sum_{\ell=0}^{i-1} q_{\ell jk} \leq i, \forall i, j, k$$

$$l_{ik} = 1 \Rightarrow \sum_{\ell=i+1}^{Q-1} q_{\ell jk} = 0, \forall i, j, k \tag{15}$$

or,

$$(Q - i - 1)l_{ik} + \sum_{\ell=i+1}^{Q-1} q_{\ell jk} \leq Q - i - 1, \forall i, j, k$$

Similarly, we present the linearisation of (17). Similar to previous case (10), the b_{ijk} decision variables results from the linearisation of (17) and denote the each product $q_{ijk} c_{jk}$.

$$c_{jk} = 1 \Rightarrow \sum_{\forall i} q_{ijk} = X_k, \quad \forall j, k \quad (17)$$

$$\text{or, } \sum_{\forall j} \sum_{\forall i} q_{ijk} c_{jk} = X_k, \quad \forall k$$

$$\text{or, } \sum_{\forall j} \sum_{\forall i} b_{ijk} = X_k, \quad \forall k,$$

$$b_{ijk} \leq c_{jk}, \quad \forall i, j, k$$

$$b_{ijk} \leq q_{ijk}, \quad \forall i, j, k$$

$$b_{ijk} \geq c_{jk} + q_{ijk} - 1, \quad \forall i, j, k$$

References

- Avionics Application Software Standard Interface, Part 1, Required Services, ARINC SPECIFICATION653P1-3 ed., AERONAUTICAL RADIO, INC. (2010)
- Awan MA (2016) Source code. <http://webpages.cister.isep.ipp.pt/~maan/MemoryReservation.zip>
- Baruah S, Burns A (2006) Sustainable scheduling analysis. In: Proceedings of the 27th IEEE real-time systems symposium. pp 159–168
- Baruah S, Mok A, Rosier L (1990) Preemptively scheduling hard-real-time sporadic tasks on one processor. In: Proceedings of the 11th IEEE real-time systems symposium
- Behnam M, Inam R, Nolte T, Sjödin M (2013) Multi-core composability in the face of memory-bus contention. ACM SIGBED Rev. 10(3):35–42
- Bini E, Buttazzo G (2009) Measuring the performance of schedulability tests. J Real-Time Syst 30(1–2):129–154
- Brandenburg B (2011) Scheduling and locking in multiprocessor real-time operating systems. Ph.D. Dissertation, Department of Computer Science University of North Carolina at Chapel Hill
- Certification authorities software team (cast), position paper (cast-32) multicore processors, Certification authorities in North and South America, Europe, and Asia (2014)
- Coelho PLC (2013) Linearization of the product of two variables. <http://www.leandro-coelho.com/linearization-product-variables/>
- Davis RI, Burns A (2009) Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. In: Proceedings of the 30th IEEE Real-Time Systems Symposium. pp 398–409
- Flodin J, Lampka K, Yi W, (2014) Dynamic budgeting for settling dram contention of co-running hard and soft real-time tasks. In: 2014 9th IEEE International Symposium on Industrial Embedded Systems (SIES). pp 151–159
- Grant M (2015) How to linearize this constraint a summation of a product of a integer with a binary. <http://math.stackexchange.com/questions/1328817/how-to-linearize-this-constraint-a-summation-of-a-product-of-a-integer-with-a-bi>
- Inam R, Mahmud N, Behnam M, Nolte T, Sjödin M (2014) Multi-core composability in the face of memory-bus contention. In: Applications symposium
- Liu J (2000) Real-time systems. Prentice Hall, New Jersey
- Mancuso R, Dudko R, Betti E, Cesati M, Caccamo M, Pellizzoni R (2013) Real-time cache management framework for multi-core architectures. In: 19th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS). pp 45–54
- Mancuso R, Pellizzoni R, Caccamo M, Sha L, Yun H (2015) WCET(m) estimation in multi-core systems using single core equivalence. In: Proceedings of the 27th Euromicro conference on real-time systems. pp 174–183
- Nowotzsch J, Paulitsch M, Buhler D, Theiling H, Wegener S, Schmidt M, (July, (2014) Multi-core interference-sensitive WCET analysis leveraging runtime resource capacity enforcement. In: 2014 26th Euromicro conference on IEEE real-time systems (ECRTS). pp 109–118

- Pellizzoni R, Yun H (2016) Memory servers for multicore systems. In: 2016 IEEE real-time and embedded technology and applications symposium (RTAS). pp 97–108
- RTCA, Inc. (2005) Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations. U.S. Dept. of Transportation, Federal Aviation Administration
- RTCA, Inc. (2012) RTCA/DO-178C. U.S. Dept. of Transportation, Federal Aviation Administration
- RTCA, Inc. (2012) RTCA/DO-254. U.S. Dept. of Transportation, Federal Aviation Administration
- Sha L, Caccamo M, Mancuso R, Kim J-E, Yoon M-K, Pellizzoni R, Yun H, Kegley R, Perlman D, Arundale G, Richard B et al (2014) Single core equivalent virtual machines for hard realtime computing on multicore processors. Univ. Tech. Rep, Urbana Champaign
- Sousa PB, Bletsas K, Tovar E, Souto P, Åkesson B (2014) Unified overhead-aware schedulability analysis for slot-based task-splitting. *J Real Time Syst* 50(5–6):680–735
- Souto P, Sousa P, Davis R, Bletsas K, Tovar E (2015) Overhead-aware schedulability evaluation of semi-partitioned real-time schedulers. In: Proceedings of the 21st IEEE conference on embedded and real-time computing and applications. pp 110–121
- Yao G, Yun H, Wu ZP, Pellizzoni R, Caccamo M, Sha L (2016) Schedulability analysis for memory bandwidth regulated multicore real-time systems. *IEEE Trans Comput* 65(2):601–614
- Yun H, Yao G, Pellizzoni R, Caccamo M, Sha L (2012) Memory access control in multiprocessor for real-time systems with mixed criticality. In: 2012 24th Euromicro IEEE conference on real-time systems (ECRTS). pp 299–308
- Yun H, Yao G, Pellizzoni R, Caccamo M, Sha L (2013) Memguard: memory bandwidth reservation system for efficient performance isolation in multi-core platforms. In: Proceedings of the 19th IEEE real-time and embedded technology and applications symposium, pp 55–64
- Yun H, Mancuso R, Wu Z-P, Pellizzoni R (2014) PALLOC: DRAM bank-aware memory allocator for performance isolation on multicore platforms. In: Proceedings of the 20th IEEE real-time and embedded technology and applications symposium. pp 155–166

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Muhammad Ali Awan received his Bachelor's Degree in Computer Engineering from National University of Science and Technology (NUST), Pakistan, in 2005. He completed his Master's Degree in 2007 from Royal Institute of Technology (KTH), Sweden, in System on Chip Design with a focus on Digital System Design and Embedded Systems. He worked as Lecturer in NUST from 2007 to 2008. Afterwards, he joined IMEC Belgium as a researcher for two years and focused on High Level Memory Management issues. He joined CISTER Research Center in 2010 and enrolled as a Ph.D. candidate in University of Porto, Portugal. During his four years of Ph.D. program, he participated in a research on "Real-Time Power Management on Partitioned Multicores". In 2014, he received his Ph.D. Degree with distinction from University Porto, Portugal under the supervision of Stefan M. Petters. Currently, he is working on the design, implementation and performance analysis of the safety-critical systems on a variety of hardware platforms. His research

interests include real-time systems, multicore scheduling, mixed-criticality systems, safety-critical systems, energy-aware scheduling, heterogeneous multicore architecture design and exploration, power modelling and resource-aware system optimizations.



Pedro F. Souto received a Licenciatura in Electrical Engineering from the University of Porto in 1986, and a Ph.D. in Computer Science from the Stony Brook University, USA, in 1999. He is an Assistant Professor at the Department of Informatics Engineering of the Faculty of Engineering of the University of Porto, and a member of CISTER Research Center. His research interests include distributed systems, fault-tolerance, real-time systems and multiprocessing.



Benny Akesson received his M.Sc. degree at Lund Institute of Technology, Sweden in 2005 and a Ph.D. from Eindhoven University of Technology, the Netherlands in 2010. Since then, he has been employed as a Researcher at Eindhoven University of Technology, Czech Technical University in Prague, and CISTER/INESC TEC Research Unit in Porto. Currently, he is working as a Research Fellow at ESI (TNO) in Eindhoven. His research interest is design methodologies for cyber-physical systems, in particular platform architectures, performance modeling, and domain-specific languages. He has published more than 60 peer-reviewed conference papers and journal articles, as well as two books about memory controllers for real-time embedded systems.



Konstantinos Bletsas has a Degree in Electronic and Computer Engineering (2002) from the Technical University of Crete (Chania, Greece) and a Ph.D. in Computer Science (2007) from the University of York (UK). Since 2007, he is a researcher at the CISTER Research Centre (Porto, Portugal). His focus is on the design and timing analysis of real-time scheduling algorithms for multiprocessor and/or mixed-criticality systems.



Eduardo Tovar has received the Licentiate, M.Sc. and Ph.D. degrees in electrical and computer engineering from the University of Porto, Porto, Portugal, in 1990, 1995 and 1999, respectively. Currently he is a Professor in the Computer Engineering Department at the School of Engineering (ISEP) of Polytechnic Institute of Porto (IPP), where he is also engaged in research on real-time distributed systems, wireless sensor networks, multiprocessor systems, cyber-physical systems and industrial communication systems. He heads the CISTER Research Unit, an internationally renowned research centre focusing on RTD in real-time and embedded computing systems. He is deeply engaged in research on real-time distributed systems, multiprocessor systems, cyber-physical systems and industrial communication systems. He is currently the Vice-chair of ACM SIGBED (ACM Special Interest Group on Embedded Computing Systems) and was for 5 years, until December 2015, member of the Executive Committee of the IEEE Technical Committee on Real-Time Systems (TC-RTS). Since 1991

he authored or co-authored more than 150 scientific and technical papers in the area of real-time and embedded computing systems, with emphasis on multiprocessor systems and distributed embedded systems. He has been consistently participating in top-rated scientific events as member of the Program Committee, as Program Chair or as General Chair. Notably he has been program chair/co-chair for ECRTS 2005, IEEE RTCSA 2010, IEEE RTAS 2013 or IEEE RTCSA 2016, all in the area of real time computing systems. He has also been program chair/co-chair of other key scientific events in the area of architectures for computing systems and cyber-physical systems as is the case of ARCS 2014 or the ACM/IEEE ICCPS 2016 or in the area of industrial communications (IEEE WFCS 2014). He is General Co-Chair of the CPSWeek 2018.

Affiliations

Muhammad Ali Awan¹  · Pedro F. Souto²  · Benny Akesson³  ·
Konstantinos Bletsas¹  · Eduardo Tovar¹ 

Pedro F. Souto
pfs@fe.up.pt

Benny Akesson
benny.akesson@tno.nl

Konstantinos Bletsas
ksbs@isep.ipp.pt

Eduardo Tovar
emt@isep.ipp.pt

- ¹ CISTER Research Centre and ISEP, Polytechnic Institute of Porto, Porto, Portugal
- ² FEUP-Faculty of Engineering and CISTER Research Centre, University of Porto, Porto, Portugal
- ³ ESI (TNO), Eindhoven, The Netherlands