Technical Report

Mixed-criticality Scheduling with Dynamic Memory Bandwidth Regulation

Muhammad Ali Awan*

Konstantinos Bletsas*

Pedro F. Souto

Benny Åkesson*

Eduardo Tovar*

*CISTER Research Centre

CISTER-TR-180603

2018/08/28

Mixed-criticality Scheduling with Dynamic Memory Bandwidth Regulation

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

*CISTER Research Centre

Polytechnic Institute of Porto (ISEP-IPP)

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, ksbs@isep.ipp.pt, kbake@isep.ipp.pt, emt@isep.ipp.pt

http://www.cister.isep.ipp.pt

Abstract

Mixed-criticality multicore system design must often provide both safety guarantees and high performance. Memory bandwidth regulation among different cores can be a useful tool for providing safety guarantees as it mitigates the interference when accessing main memory. The use of mode changes and system models such as those of Vestal can help provide both safety, for critical functions, and scheduling performance, by efficiently utilising the platform. In this work, we therefore combine per-core memory access regulation with the well established Vestal model and improve on the state-of-the-art in two respects. 1) we allow the memory access budgets of the cores to be dynamically adjusted, when the system undergoes a mode change, reflecting the different needs in each mode, for better schedulability. 2) we devise a memory-regulation-aware and stall-aware schedulability analysis for such systems, based on the well-known AMC-max technique. By comparison, the state-of-the-art did not offer the option of dynamic adjustment of core budgets, and only offered regulation-aware schedulability analysis based on AMC-rtb, which is inherently more pessimistic. As an additional contribution, 3) we consider different task assignment and bandwidth allocation heuristics, in experiments with synthetic task sets, to assess the improvement from using dynamic memory budgets and the new analysis. In our results, we have observed an improvement in schedulability ratio up to 9.1% over the state-of-the-art algorithm.

Mixed-criticality Scheduling with Dynamic Memory Bandwidth Regulation (Long Version)

Muhammad Ali Awan*, Konstantinos Bletsas*, Pedro F. Souto†, Benny Akesson‡, Eduardo Tovar*
*CISTER/INESC-TEC and ISEP/IPP, Porto, Portugal
†University of Porto, FEUP-Faculty of Engineering and CISTER/INESC-TEC, Porto, Portugal
‡Embedded Systems Innovation, Eindhoven, the Netherlands

***Abstract*—Mixed-criticality multicore system design must often provide both safety guarantees and high performance. Memory bandwidth regulation among different cores can be a useful tool for providing safety guarantees as it mitigates the interference when accessing main memory. The use of mode changes and system models such as those of Vestal can help provide both safety, for critical functions, and scheduling performance, by efficiently utilising the platform. In this work, we therefore combine per-core memory access regulation with the well-established Vestal model and improve on the state-of-the-art in two respects. 1) we allow the memory access budgets of the cores to be dynamically adjusted, when the system undergoes a mode change, reflecting the different needs in each mode, for better schedulability. 2) we devise a memory-regulation-aware and stall-aware schedulability analysis for such systems, based on the well-known AMC-max technique. By comparison, the state-of-the-art did not offer the option of dynamic adjustment of core budgets, and only offered regulation-aware schedulability analysis based on AMC-rtb, which is inherently more pessimistic. As an additional contribution, 3) we consider different task assignment and bandwidth allocation heuristics, in experiments with synthetic task sets, to assess the improvement from using dynamic memory budgets and the new analysis. In our results, we have observed an improvement in schedulability ratio up to 9.1% over the state-of-the-art algorithm.**

I. Introduction

Many real-time embedded systems (e.g., in the automotive, avionics, railway and aerospace domains) are *mixed criticality systems* (MCSs), meaning that functions of different criticalities use the same hardware resources, such as cores, interconnects and memories. A deadline miss by a high-criticality task may be disastrous, so lower-criticality tasks must not interfere unpredictably. This is traditionally avoided through rigid performance isolation [1, Footnote 1], which can be inefficient in utilising the platform.

However, increasingly often, offering high performance (within the size, weight, power and cost constraints of the system) is also important for MCSs. This motivated a move towards multicore platforms, to obtain that performance. In parallel, the real-time research community is providing the scheduling theory, such as Vestal's model [2] and its variants, to enable better performance guarantees on a given platform, without compromising safety. However, the move to multi-cores, introduces resource sharing between cores, which, if unaccounted for, can cause unpredicable timing behavior [3]. This is most evident for main memory: Different cores contending for the memory can lead to *stalling* of execution on the core. Such stalling must often be kept low and must be always upper-bounded, on grounds of safety and certifiability.

Memory contention can be mitigated by using *regulation*, which is the approach taken under the Single-Core Equivalence (SCE) framework [4]. Each core gets an associated periodically-replenished *memory access budget*. When a core attempts to issue more memory accesses than its budget, it gets temporarily stalled, until the next replenishment. This allows the worst-case memory stall per task to be upper-bounded and incorporated into the schedulability test. Yao et al [5] did this for single-criticality systems and in recent work [6], we incorporated their contributions to the schedulability theory of mixed-criticality systems conforming to the most established variant [7] of Vestal's model.

This model views the system operation as different modes (two or more, ordered from lowest to highest), and the set of tasks present in each mode is a subset of those present in the next-lowest mode. Different worst-case task execution times (WCETs) are assumed for the same task in each mode it is a part of, with corresponding degrees of confidence. This allows less rigorous (less costly) WCET estimation for lower-criticality tasks and higher system utilisation, without compromising safety.

This work targets memory-regulated multicores conforming to the same model as [6], but it brings three new contributions. First, it considers a more general scheduling and memory regulation arrangement, wherein the cores' memory access budgets are dynamically adjusted at mode change. This permits the memory budgets to be better reapportioned according to the cores' different requirements in each mode, in order to improve schedulability. Secondly, the memory-regulation-aware schedulability test devised as part of this work is based on AMC-max [8, Section IV.C], whereas in [6], it was based on AMC-rtb [8, Section IV.B]. AMC-max is a tighter (i.e., less pessimistic) test than AMC-rtb, but more complex, conceptually and computationally. Finally, we explore, in experiments with synthetic tasks sets and different task assignment and memory budget allocation heuristics, the schedulability improvement from the two first contributions.

The rest of this paper is structured as follows. Next (in Section 2), we provide additional background and discuss related work. In Section 3, we formally define our system model. In Section 4, we discuss in more detail the relevant state-of-the-art in Adaptive Mixed Criticality (AMC) scheduling and analysis for memory-bandwidth-regulated systems that we build upon. Section 5 provides our new stall analysis, for the class of considered systems. Section 6 discusses our new mixed-criticality schedulability analysis, and how the worst-case stall terms derived using our stall analysis are integrated into it. In Section 7, we propose different heuristics for task-to-core assignment and memory bandwidth allocation to the cores. Section 8 evaluates these heuristics and assesses the schedulability improvement from the use of dynamic memory access budgets over static ones. Section 9 concludes the paper.

II. Related Work

In its most established variant [7], Vestal's model assumes an ordered set of criticality levels. Each task has a period, a deadline, a (design) criticality level and a set of WCETs – one for every criticality level not exceeding its own and non-decreasing with respect to the latter. For this model, Baruah et al. [8] devised *Adaptive Mixed Criticality* (AMC) scheduling and the notion of run-time *system criticality level*, initialised to the lowest task criticality at startup. If a task exceeds its WCET for the system's current criticality level, the system stops all tasks with criticality equal to that level and increments its criticality level. Schedulability analysis relies on fixed-priority worst-case response time (WCRT) analysis, using the appropriate task WCETs. Two such schedulability tests are presented in [8]: AMC-rtb and the tighter, but more complex, AMC-max. Fleming and Burns [9] extended AMC to an arbitrary number of criticality levels and showed that AMC-rtb approximates AMC-max reasonably well. The later AMC-IA test [10] slightly outperforms AMC-max. Other works improve on AMC-max via a slightly different preemption model.

Many works exist on mechanisms for mitigating the interference on shared resources and on integrating the effects of such interference to the schedulability analysis [4], [5], [11]–[16]. Yun et al. [16] analysed the response time of critical tasks scheduled on a single core by regulating the memory access rates of cores running non-critical tasks only. This idea was generalised in [5] by regulating all cores and through a corresponding schedulability analysis that finds the response time of all tasks. However, unlike Vestal's model, in [5] critical and non-critical tasks cannot co-exist on the same core, which is potentially inefficient in terms of resource usage. In comparison, in [6], we ported the regulation scheme from [5] to a standard Vestal model, which is more general and allows both critical and non-critical tasks on the same core, for efficient resource use without compromising the schedulability of the critical tasks and system safety. In the present work, we go well beyond that by (i) integrating the regulation-awareness to a more accurate, but more computationally, intensive schedulability test (AMC-max) and (ii) generalising the model to allow for the cores' memory access budgets to be dynamically adjusted at mode change, in order to better serve their needs in different modes. This dynamic reallocation of resources at mode change, for better performance, is something we have also explored with partitioned cache resources in [17].

III. System model

A. Platform and memory regulation model

We assume a multicore platform composed of m identical physical cores that access the main memory via a single shared memory controller. Each core can have multiple outstanding memory requests. The prefetchers and speculative units are disabled. Most of the assumptions made in this work are inspired by the Single-core-equivalence framework [4], more specifically from Yao et al. [5]. The combined policy of both the memory controller and its interconnect is round-robin [5], [18]. The last-level cache is either private or partitioned to each core. Like Yao et al. [5], each memory access is assumed to take a constant time of L. The accesses to main memory are regulated by a software mechanism such as MemGuard [18] or in hardware.

Each core i is assigned a pair of memory access budgets (Q_i^L, Q_i^H), one for each mode of operation, respectively. These are the maximum number of memory accesses allowed in each regulation period of length P. The relation between Q_i^L and Q_i^H is arbitrary, i.e., either $Q_i^L \leq Q_i^H$ or $Q_i^L > Q_i^H$. This budget pair (Q_i^L, Q_i^H) is set at design time for each core i and budgets may differ across cores. The budget enforcement mechanism ensures that a core exceeding its given limit, in L-mode or H-mode, is stalled until the start of the next regulation period. The regulation periods are synchronised on all cores. It is ensured that the sum of the memory budgets assigned to all cores does not exceed the available memory bandwidth at any time instant in L-mode ($\sum_i \frac{Q_i^L}{P} \leq 1$) and H-mode ($\sum_i \frac{Q_i^H}{P} \leq 1$). Like Yao at el. [5], we assume CPU computation and memory accesses do not overlap in time.

B. Task model

We assume a task-set τ composed of n independent mixed criticality sporadic tasks ($\tau = \{\tau_1, \tau_2, \cdots, \tau_n\}$). Each task has a relative deadline D_i, a minimum inter-arrival time T_i and a criticality level κ_i. The task-set is partitioned on a given multicore platform offline and migrations are not allowed at run-time. Tasks on each core are scheduled with a preemptive fixed-priority scheduling algorithm. We consider the same Vestal mixed-criticality model as Baruah and Burns [8], which views the system operation as different modes, whereby only tasks of certain criticality or higher execute. For each task, different WCET estimates are assumed with different confidence in their safety. For simplicity, we assume only two modes of operation (L-mode and H-mode). The L-mode WCET estimate (L-WCET) of a task τ_i is denoted as C_i^L and its safe, but pessimistic, H-mode WCET estimate (H-WCET) is denoted as $C_i^H \geq C_i^L$. The values are computed in isolation on a core assuming no interference from other cores on the shared memory controller and its interconnect. The WCET estimates (both C_i^L and C_i^H) can be further subdivided into two parts: (a) CPU computation and (b) memory access time. Hence, $C_i^L = C_i^{e|L} + C_i^{m|L}$ and $C_i^H = C_i^{e|H} + C_i^{m|H}$.

The system boots in L-mode with each core initialised with its Q_i^L memory budget and remains in L-mode as long as no job (on any core) exceeds its $C_i^{e|L}$ or its $C_i^{m|L}$. However, in case of overrun of either $C_i^{e|L}$ or $C_i^{e|m}$ by any task τ_i, all L-tasks are stopped and the system switches to H-mode, where only the H-tasks execute. If the mode switch occurs in the r^{th} regulation period at time instant s, the resetting of the memory budget on each core to their Q_i^H value is delayed till the start of the subsequent regulation period ($r + 1^{th}$). This budget-switch instant is denoted as $s' \geq s$. Hence, H-jobs executing in r^{th} regulation period after the mode switch time instant s on any core i, execute with a memory budget of Q_i^L till the budget-switch instant s' (i.e., the start of subsequent ($r + 1^{th}$) regulation period). The H-WCETs need not be specified for the L-tasks. The system is schedulable if no task deadline is missed in L-mode and no H-task deadline is missed in H-mode (including H-tasks caught in the mode switch). This must be verifiable offline, via schedulability tests that use the respective WCET estimates and memory budgets for each mode.

We also denote by $hpL(i)$ and $hpH(i)$ the sets of L- and H-tasks, respectively, with priority higher than task τ_i. Moreover, $hp(i) = hpL(i) \cup hpH(i)$. Tasks in $hpL(i)$ can execute in L-mode only, whereas tasks in $hpH(i)$ may execute in both L- and H-mode. The response time of a task in L-mode and H-mode is denoted as R_i^L and R_i^H, respectively, and it includes the stall time due to the contention and the memory regulation.

IV. Overview of the State-of-the-art

A. Adaptive Mixed Criticality (AMC) Scheduling

The AMC fixed-priority-based mixed-criticality scheduling algorithm is designed for platforms that can monitor job execution times. It supports any number of criticality levels, but here we consider only two, for simplicity. Tasks are scheduled according to their fixed priority. The system starts in L-mode, with L-WCET estimates assumed for all tasks. Any overrun (excedance of L-WCET estimate) triggers a mode switch that (i) halts L-tasks and (ii) assumes H-WCET estimates henceforth for the H-tasks. The schedulability analysis of AMC builds on standard fixed-priority response time analysis [19], [20]. The schedulability conditions are checked for both modes, L and H, and also for the mode transition interval (which starts at the moment of the mode switch and ends at the earliest idle instant). The schedulability of any task τ_i in L-mode is checked by comparing its deadline D_i to its WCRT R_i^L (Equation 1), computed using the L-WCETs.

$$R_i^L = C_i^L + \sum_{\tau_j \in hp(i)} \left\lceil \frac{R_i^L}{T_j} \right\rceil C_j^L \tag{1}$$

where $hp(i)$ is the set of higher-priority L- and H-tasks.

Likewise, in the steady H-mode, for each H-task τ_i, the WCRT is computed using the H-WCETs of tasks in $hpH(i)$, the set of H-tasks with higher priority than τ_i. Nevertheless, the steady H-mode analysis is subsumed by the transition mode analysis, meaning that if the system is schedulable during the transition mode interval, it is also schedulable in steady H-mode. For the mode transition analysis, Baruah et al. [8] offer two tractable WCRT estimation techniques based on solving recurrence relations. The first one, *AMC-rtb*, avoids enumeration of the instants at which a mode switch may occur by bounding the worst-case interference by higher priority L-tasks separately from that of higher-priority H-tasks. This analysis is straightforward, but pessimistic, since the worst-case interference by L-tasks cannot occur simultaneously with the worst-case interference from H-tasks. The tighter but more elaborate second recurrence, *AMC-max*, considers the key instants when the mode switch may occur and takes the maximum response time obtained for each of these instants. For the H-task under analysis τ_i and any mode switch instant s, AMC-max has the advantage of upper-bounding: (i) the number of jobs by higher-priority H-tasks ($hpH(i)$) that may execute in H-mode after the mode switch with H-WCET estimates and (ii) the number of jobs by higher-priority L- and H-tasks ($hp(i)$) that may execute before the mode switch with L-WCET estimates. The WCRT time of an H-task τ_i is given by (2) [8]:

$$R_i^H = \max(R_i^s), \ \ \forall s \in \{0, R_i^L\} \tag{2}$$

$$R_i^s = C_i^H + \sum_{\tau_j \in hpL(i)} \left(\left\lfloor \frac{s}{T_j} \right\rfloor + 1 \right) C_j^L + \tag{3}$$

$$\sum_{\tau_k \in hpH(i)} \left\{ M(k, s, R_i^s) C_k^H + \left(\left\lceil \frac{t}{T_k} \right\rceil - M(k, s, R_i^s) \right) C_k^L \right\}$$

$$M(k, s, t) = \min \left\{ \left\lceil \frac{t - s - (T_k - D_k)}{T_k} \right\rceil + 1, \left\lceil \frac{t}{T_k} \right\rceil \right\} \tag{4}$$

We use AMC-max in this work due to its property to differentiate between demand happening before and after the mode switch, which in turn allows to efficiently quantify the effect of dynamic memory budgets.

B. Memory Bandwidth Regulation Analysis

Yao et al. [5] provide schedulability analysis with memory stall analysis for single-criticality systems using partitioned fixed-priority scheduling with Rate-Monotonic priority assignment. We summarize this work as it is used in our analysis.

a) Stall Analysis: A memory access resulting from a cache miss can stall its core either (i) because of memory regulation, e.g., if the core's memory budget has been exhausted, referred to as *regulation stall*, or (ii) because of concurrent memory accesses by other cores, referred to as *contention stall*. Yao et al. [5], in their analysis, initially consider (as an assumption later relaxed) that the task under analysis is not preempted – therefore, at any time, it is either executing, accessing memory or stalled. This analysis leads to three worst-case memory access patterns, depending on the core's bandwidth $b = \frac{Q}{P}$ and the task's cache stall ratio $r = \frac{C_i^m}{C_i}$. Due to space limitations, we just briefly describe the different cases/patterns; for details, see [5]. Note that we omit the index of the task under analysis in this subsection for simplicity.

Case 1: $b \leq \frac{1}{m}$ In this case, the worst-case stall occurs when all the memory accesses are clustered, maximizing the regulation stall, and the stall can be upper-bounded by:

$$stall = \begin{cases} \frac{C^m}{Q}(P\text{-}Q)\text{+}(m\text{-}1)Q & \text{if } C^m\%Q\text{=}0 \\ \left\lceil \frac{C^m}{Q} \right\rceil (P\text{-}Q)\text{+}(m\text{-}1)(C^m\%Q) & \text{otherwise} \end{cases} \quad (5)$$

Case 2: $b{>}\frac{1}{m}$ *and* $r{=}\frac{C^m}{C}{<}\frac{1-b}{(m-1)b}$ In this case, the worst-case stall occurs when all memory accesses suffer the maximum contention stall and an upper bound of the stall is given by:

$$stall = (P - Q) + (m - 1) \cdot Q \quad (6)$$

Case 3: $b > \frac{1}{m}$ *and* $r = \frac{C^m}{C} \geq \frac{1-b}{(m-1)b}$ In this case, the density of memory accesses is such that some regulation periods must suffer regulation stalls, and an upper bound of the stall is:

$$stall = \begin{cases} (1 + K_1)(P - Q) + r_1 & \textit{if } C \leq (1 + K_1)Q \\ \left(1 + \frac{C}{Q}\right)(P - Q) + r_2 & \text{otherwise} \end{cases} \quad (7)$$

$$\text{where, } K_1 = \left\lfloor \frac{C^e}{Q - RBS} \right\rfloor, \; RBS = \frac{P - Q}{m - 1}$$
$$r_1 = \min\{P - Q, (m - 1)(C^m - K_1 \cdot RBS)\}$$
$$r_2 = \min\{P - Q, (m - 1)(C\%Q)\}$$

Note that all 3 cases account for an initial regulation stall, of duration $P - Q$, that may occur in the worst case when a task is scheduled to run, but the core has already exhausted its memory budget.

b) Schedulability Analysis: The schedulability analysis in [5] relies on standard fixed-priority response time analysis [20], comparing the response time of each task, in decreasing order of a task's priority, with its deadline. However, the above stall analysis assumes that the task is not preempted. Therefore, to analyse the response time of each task, Yao et al. [5] construct a synthetic task, equivalent to all the activations that occur in the response time window of the task under analysis, using the following recurrence:

$$R_i^{(k+1)} = C_i + \sum_{\tau_j \in hp(i)} \left\lceil \frac{R_i^{(k)}}{T_j} \right\rceil C_j + stall\big(R_i^{(k)}\big) \quad (8)$$

where the stall term, $stall\big(R_i^{(k)}\big)$, is computed using Yao's algorithm with these parameters for the equivalent synthetic/composite task:

$$\begin{cases} C_{Comp}^{m(k)} = C_i^m + \sum_{\tau_j \in hp(i)} \left\lceil \frac{R_i^{(k)}}{T_j} \right\rceil C_j^m \\ C_{Comp}^{e(k)} = C_i^e + \sum_{\tau_j \in hp(i)} \left\lceil \frac{R_i^{(k)}}{T_j} \right\rceil C_j^e \end{cases} \quad (9)$$

The initial estimate of τ_i's response time $R_i^{(0)}$ is computed with a standard response time recurrence, without a stall term.

TABLE I: Symbols used in the analysis

Q^L, Q^H	L- and H-mode memory budgets, respectively
$C^{m\|L}, C^{m\|H}$	L- and H-mode maximum memory access time (equal to the number of memory accesses) respectively
$C^{e\|L}, C^{e\|H}$	L- and H-mode maximum CPU execution, respectively
$C^{m\|\ell}, C^{m\|h}$	memory access time (equal to the number of accesses) before and after mode switch, respectively
$C^{e\|\ell}, C^{e\|h}$	CPU execution before and after mode switch, respectively
$Stall^\ell, Stall^h$	total stall before and after mode switch, respectively
$Stall$	total stall
P	regulation period
m	number of cores
s	mode-switch instant
s'	budget-switch instant
$single()$	worst-case single criticality stall according to Yao et al. analysis [5], ignoring the regulation stall at the beginning of the execution
$lb(X)$	lower bound of parameter X
$ub(X)$	upper bound of parameter X

V. Stall Analysis

In this section we analyse the stall incurred by a H-task upon mode-switch, assuming that there are no other tasks running on the same core, therefore we drop the task index. The stall expression developed in this section will be used for response time analysis, as done by Yao et al. [5]. Because we build on Yao's stall analysis, we use L, the memory access latency, as the unit of time for all times, including P and Q. Therefore, sometimes, we refer to memory access time, C^m, as the number of memory accesses. Table I summarizes the symbols used in this analysis.

Let s be the mode switch instant. To simplify the mathematical expressions, in our analysis, s is measured relative to the beginning of the first regulation period after the release of the H-task under analysis, i.e. after the initial regulation stall, rather than relative to the task release, as done in [8]. Our goal is to upper bound the total stall, independently of the value of s. Our approach is to upper bound the stall in each of the following two phases: 1) before the mode switch, i.e. before s, and 2) after the mode switch, i.e. after s, for all possible values of s.

Let $Stall^\ell$ and $Stall^h$ be the stall in each of these phases, ignoring the initial regulation stall, which is added later. We bound independently the values of the stall in each of these phases, i.e. we compute $ub(Stall^\ell)$ and $ub(Stall^h)$, where $ub(X)$ denotes an upper-bound of parameter X. To compute these bounds we use single-criticality stall analysis by Yao et al. [5] with the appropriate parameter values. For conciseness, in this section and the next, we refer it as *Yao's analysis*.

In this section, we assume that s occurs at a regulation period boundary. Therefore, s is a multiple of P. In Subsection VI-C, we drop this assumption.

Let $C^{m|\ell}$ (resp. $C^{m|h}$) be the memory access time in L-mode (resp. H-mode), i.e. before (resp. after) mode switch, and $C^{e|\ell}$ (resp. $C^{e|h}$) be the memory access time in L-mode (resp. H-mode), i.e. before (resp. after) mode switch. Then, it must be:

$$C^{m|H} = C^{m|\ell} + C^{m|h} \tag{10}$$

$$C^{e|H} = C^{e|\ell} + C^{e|h} \tag{11}$$

To upper bound the stall after the mode switch, we use:

$$ub(Stall^h) = single(C^m = ub(C^{m|h}), C^e = ub(C^{e|h}), Q = Q^H) \tag{12}$$

where $single()$ is the worst-case stall according to Yao analysis, ignoring the regulation stall at the beginning of the execution and

$$ub(C^{m|h}) = C^{m|H} - lb(C^{m|\ell})$$

$$ub(C^{e|h}) = C^{e|H} - lb(C^{e|\ell})$$

where $lb(X)$ denotes a lower-bound for parameter X. I.e., in (12) we use upper-bounds for $C^{m|h}$ and $C^{e|h}$ estimated using lower bounds for $C^{m|\ell}$ and $C^{e|\ell}$, respectively. This is because of (10) and (11), and Yao's analysis shows that the stall is non-decreasing with both C^m and C^e.

So, the challenge is to compute the expressions for the lower bounds. Since when a task is not stalled it must be either computing or accessing memory, we use the following lower bounds:

$$lb(C^{e|\ell}) = max(0, s - ub(C^{m|\ell}) - ub(Stall^\ell)) \tag{13}$$

$$lb(C^{m|\ell}) = max(0, s - ub(C^{e|\ell}) - ub(Stall^\ell)) \tag{14}$$

These expressions are safe but pessimistic, especially for $lb(C^{m|\ell})$, because it is unlikely that $Stall^\ell$ be maximum when $C^{e|\ell}$ is also maximum.

Tight independent upper bounds for $C^{e|\ell}$ and $C^{m|\ell}$ are:

$$ub(C^{e|\ell}) = min(s, C^{e|L}) \tag{15}$$

$$ub(C^{m|\ell}) = min\left(\frac{s}{P} \cdot Q^L, C^{m|L}\right) \tag{16}$$

Note that $\frac{s}{P} \cdot Q^L$ is the value imposed by memory regulation: in L-mode the memory budget is Q^L. Although taken independently these bounds are tight, it may be the case that they cannot both occur simultaneously.

These upper bounds can be used as C^e and C^m, respectively, in Yao's stall analysis to upper bound $Stall^\ell$. Furthermore, we know that the maximum stall in each regulation period is $P - Q$, therefore:

$$ub(Stall^\ell) = min\Big(\frac{s}{P} \cdot (P - Q^L), single(C^e = ub(C^{e|\ell}), C^m = ub(C^{m|\ell}), Q = Q^L)\Big) \tag{17}$$

where $ub(C^{e|\ell})$ and $ub(C^{m|\ell})$ are given by (15) and (16), respectively.

Thus, an upper bound of the stall of a non-preemptable H-task upon mode switch is given by:

$$(P - Q) + ub(Stall^\ell) + ub(Stall^h)$$

where $ub(Stall^ell)$ and $ub(Stall^h)$ are given by (17) and (12), respectively.

VI. Schedulability Analysis

In this section, we integrate the memory regulation related stalls in the AMC-max scheme. Like in AMC-max, we consider 3 cases: L-mode steady operation, H-mode steady operation and mode-switch operation.

Schedulability in steady L-mode is determined by applying Yao et al. [5] analysis, as summarized in Section IV, more specifically in (8) and (9). I.e., we apply standard response time analysis to a synthetic/composite task that comprises all the jobs that run within the response time window of the task under analysis, τ_i, in L-mode:

$$R_i^{L(k+1)} = C_i^L + \sum_{j \in hp(i)} \left\lceil \frac{R_i^{L(k)}}{T_j} \right\rceil C_j^L + Stall(R_i^{L(k)}) \tag{18}$$

$$C_{comp}^{m|L} = C_i^{m|L} + \sum_{j \in hp(i)} \left\lceil \frac{R_i^{L(k)}}{T_j} \right\rceil C_j^{m|L} \tag{19}$$

$$C_{comp}^{e|L} = C_i^{e|L} + \sum_{j \in hp(i)} \left\lceil \frac{R_i^{L(k)}}{T_j} \right\rceil C_j^{e|L} \tag{20}$$

where $R_i^{L(k)}$ denotes the response time value after k iterations, $C^m = C_{comp}^{m|L}$ and $C^e = C_{comp}^{e|L}$ are the parameters of the composite task in L-mode, and $Stall(R_i^{L(k)})$ is the memory stall and is computed using Yao et al. stall analysis [5] for the composite task with $Q = Q^L$ (omitting core index). In the first iteration ($k = 0$), $R_i^{L(k)}$ is initialized with the solution of Recurrence (18) without the stall term.

The worst-case response time in steady H-mode can be computed using similar equations. The differences being that the composite task models the interference of only H-tasks whose priority is equal or higher than τ_i, and the stall term is computed using Q^H rather than Q^L.

To upper-bound the response time of a H-task τ_i when there is a mode switch at time s relative to τ_i's job release, we extend Recurrence (4) for AMC-max from [8] to take into account the effect of the stalls induced by memory regulation. Indeed, this recurrence considers the interference of higher priority L-jobs before the mode switch and the interference of higher priority H-jobs throughout τ_i's response time window, but [8] does not assume memory regulation. To take into account the effect of memory regulation in the response time, we add a stall term $Stall(s, R_i^{s(k)})$, obtaining Recurrence (21). Like in [8], we take the maximum of the response times for all values of s. Also like in [8], there is no need to compute R_i^s for all values of $s < R_i^L$; it is enough to compute it only at the release time of jobs of L-tasks in $\mathbf{hpL}(i)$.

$$R_i^{s(k+1)} = C_i^H + \sum_{j \in hpL(i)} \left(\left\lfloor \frac{s}{T_j} \right\rfloor + 1 \right) C_j^L \tag{21}$$
$$+ \sum_{j \in hpH(i)} \left\{ M(j, s, R_i^{s(k)}) C_j^H \right.$$
$$\left. + \left(\left\lceil \frac{R_i^{s(k)}}{T_j} \right\rceil - M(j, s, R_i^{s(k)}) \right) C_j^L \right\} + Stall(s, R_i^{s(k)})$$
$$R_i^H = \max(R_i^s), \ \ \forall s \in \{0, 1, \dots, R_i^L\} \tag{22}$$

where $M(j, s, t)$ is an upper-bound on the number of active jobs of task j in the (s, t) time interval and is given by (4).

The additional stall term $Stall(s, R_i^{s(k)})$ considers the effect of the memory regulation mechanism in the WCRT of a H-task τ_i and may differ for fixed ($Q^L = Q^H$) and dynamic ($Q^L \neq Q^H$) memory bandwidth allocation policies.

In the following, we derive the stall term using the concept of equivalent composite/synthetic task both for static, i.e. when Q does not change upon mode switch, and for dynamic memory bandwidth policies.

A. *$Stall(s, R_i^s)$ with static memory bandwidth allocation*

In this case, the mode switch affects only the tasks that can execute, not the core's memory bandwidth, i.e. $Q^L = Q^H$. Therefore, we use the AMC-max analysis, more specifically (3), to derive the $C^m = C_{comp}^{m|H}, C^e = C_{comp}^{e|H}$ parameters of the composite task:

$$C_{comp}^{m|H} = C_i^{m|H} + \sum_{j \in hpL(i)} \left(\left\lfloor \frac{s}{T_j} \right\rfloor + 1 \right) C_j^{m|L} \tag{23}$$
$$+ \sum_{k \in hpH(i)} \left\{ M(k, s, R_i^s) C_k^{m|H} \right.$$
$$\left. + \left(\left\lceil \frac{R_i^s}{T_k} \right\rceil - M(k, s, R_i^s) \right) C_k^{m|L} \right\}$$
$$C_{comp}^{e|H} = C_i^{e|H} + \sum_{j \in hpL(i)} \left(\left\lfloor \frac{s}{T_j} \right\rfloor + 1 \right) C_j^{e|L} \tag{24}$$
$$+ \sum_{k \in hpH(i)} \left\{ M(k, s, R_i^s) C_k^{e|H} \right.$$
$$\left. + \left(\left\lceil \frac{R_i^s}{T_k} \right\rceil - M(k, s, R_i^s) \right) C_k^{e|L} \right\}$$

Because $Q = Q^L = Q^H$, single criticality stall expression of Yao et al. [5] is applicable and therefore we use:

$$(P - Q) + single(C^e = C_{Comp}^{e|H}, C^m = C_{Comp}^{m|H}, Q = Q^L) \tag{25}$$

to compute the stall, which is then used in Recurrence (21).

B. *$Stall(s, R_i^s)$ with dynamic memory bandwidth allocation*

In this case, we use the stall analysis for mode switch presented in Section V. However, that analysis can be applied directly only to a task that does not suffer interference from other tasks, i.e. only to the highest priority H-task, if its priority is also higher than that of all the L-tasks. Otherwise we need to use the concept of composite task.

The stall analysis of Section V essentially upper bounds the stalls before and after the budget change of a single non-preemptive H-task, by upper bounding the CPU execution and the memory access times in both phases, i.e. before and after the mode switch. The use of a synthetic task composed of task τ_i and all the tasks with priority higher than τ_i ensures that the synthetic task is non-preemptive. To upper bound the CPU execution and the memory accesses in each phase of the composite task we rely on AMC-max [8].

However, we cannot use that analysis directly, because it does not compute the number of interfering H-jobs in L-mode independently of the number of interfering H-jobs in H-mode. Instead, it uses (4) to upper bound the number of interfering H-jobs in H-mode, which it then subtracts from the upper bound of the number of interfering H-jobs in the response time window. As a result, the number of interfering H-jobs in L-mode estimated may be smaller than the actual number of L-jobs and, therefore, lead to an unsafe estimate of the stall term in Recurrence (21). This is shown below with example.

Example: Assume that $Q^H < Q^L$ and that $Q^L/P < 1/m$. This means that both in L-mode and in H-mode, Case 1 of Yao's stall analysis apply. I.e., that in both modes the worst case stall occurs when the number of regulation stalls is maximum. Consider moving a job, with C^m memory accesses, from H-mode to L-mode, thus increasing the number of memory accesses in L-mode and decreasing the number of memory accesses in H-mode by the same amount. Assume that these C^m memory accesses suffered a contention stall in H-mode, but they lead to one additional regulation stall in L-mode. Thus the reduction in stall in H-mode is, according to Yao's stall analysis, $C^m \cdot (m-1)$, because the maximum contention stall is $m-1$ (L time units). On the other hand the increase in stall in L-mode is $(P - Q^L) - (Q^L - C^m) \cdot (m-1)$, i.e. there is an additional regulation stall but some of the memory accesses that before the move suffered maximum contention stall, now occur in a period with a regulation stall, and therefore there will be a reduction of $Q - C^m$ contention stalls in L-mode. Thus the move of one job from one mode to another will lead to a higher contention if:

$$C^m \cdot (m-1) < (P - Q^L) - (Q^L - C^m) \cdot (m-1)$$
$$\frac{Q^L}{P} < \frac{1}{m}$$

which holds by assumption. Thus, it is possible that moving some job from one mode to another, even with a larger memory budget, will lead to a larger total stall.

Because of this, to ensure safety, we compute the bound of interfering H-jobs in L-mode independently of bound of interfering H-jobs in H-mode, and therefore use the same expression to compute the number of interfering jobs in L-

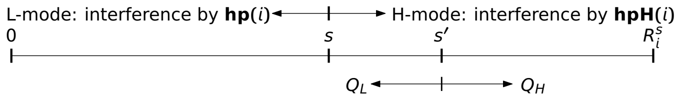

Fig. 1: Transition mode: mode switch may not coincide with a regulation period boundary, i.e. with memory budget change.

mode, independently of their criticality:

$$ub(C^{m|\ell}) = C_i^{m|L} + \sum_{j \in hp(i)} \left(\left\lfloor \frac{s}{T_j} \right\rfloor + 1 \right) C_j^{m|L} \tag{26}$$

$$ub(C^{e|\ell}) = C_i^{e|L} + \sum_{j \in hp(i)} \left(\left\lfloor \frac{s}{T_j} \right\rfloor + 1 \right) C_j^{e|L} \tag{27}$$

Applying single criticality stall analysis [5] to a composite task with these parameters and $Q = Q^L$, yields an upper-bound of the stall in L-mode of task τ_i, ignoring initial regulation stall.

To upper bound the stall in H-mode, we use (4) from AMC-max analysis to upper bound the number of interfering H-jobs in H-mode. Therefore, the parameters of the equivalent synthetic task are:

$$\begin{aligned} ub(C^{m|h}) = {} & C_i^{m|H} - lb(C_i^{m|\ell}) \\ & + \sum_{j \in hpH(i)} M(k, s, R_i^s)\, C_j^{m|H} \end{aligned} \tag{28}$$

$$\begin{aligned} ub(C^{e|h}) = {} & C_i^{e|H} - lb(C_i^{e|\ell}) \\ & + \sum_{j \in hpH(i)} M(k, s, R_i^s)\, C_j^{e|H} \end{aligned} \tag{29}$$

Applying single criticality stall analysis [5] to a composite task with the parameters given by (28) and (29) and $Q = Q^H$, yields an upper-bound of the stall in H-mode of task τ_i.

C. *Dropping the $s = s'$ assumption*

So far, including Section V, we assumed that the mode switch instant, s, occurs at regulation period boundaries, i.e. that it coincides with budget change instant, s', and therefore the stall analysis assumes that, upon mode switch, all H-jobs execute with Q^H in H-mode. However, if, as shown in Fig. 1, this is not the case, with dynamic memory bandwidth allocation, upon mode switch, some H-jobs may execute with Q^L during the (s, s') interval, even though they are already in in H-mode. However, $Stall^\ell$ includes only the stall generated by H-jobs that are released up to s, whereas $Stall^h$ includes the stall of H-jobs that are released after s, but assuming that the memory bandwidth allocated is Q^H. As a result, the total stall computed may be lower than the actual stall. Thus, to ensure that our analysis is safe, we assume that upon mode switch there is a regulation stall of $s' - s$, similar to the initial regulation stall upon job release in Yao's analysis. Clearly this is safe: the stall in that interval cannot be larger.

VII. Memory Bandwidth Allocation and Task-to-core Assignment Heuristics

We devised the following task-to-core assignment and memory bandwidth allocation heuristics to explore the benefit from our analysis and the use dynamic memory budgets. We used Audsley's task priority assignment algorithm [21], even if it is no longer necessarily optimal with an additional stall.

A. *AMC-max Dominant*

We call this heuristic "Dominant" because it initially attempts to assign tasks using static memory budgets across the mode switch and only attempts dynamic budgets if the previous strategy does not succeed. This heuristic has 3 stages:

1) In the first stage, it considers static memory budgets across the mode switch and uses stall-conscious AMC-max analysis for feasibility. The task-to-core assignment is performed via Memory-Fit [17]. That is, the core i that needs the least increase in its Q_i to accommodate a task, is chosen for its assignment. The memory bandwidth requirement of a core is determined using binary search over the available memory bandwidth range. This heuristic allows for uneven memory bandwidth assignment across cores. If a task cannot be assigned on any core, it is set aside for later stages and the next task is considered for allocation. Once no more tasks can be assigned, we enter a second stage:
2) Using sensitivity analysis, we "trim off" from each core, any memory bandwidth which was over-committed, in one of the modes, due to the static memory bandwidth allocation across the mode switch. This binary-search-based sensitivity analysis, mimimises the memory bandwidth in each mode of operation using the stall-aware AMC-max schedulability analysis, with Q_i^L and Q_i^H not necessarily equal. Let $Q_i^{stage-1}$ be the static budget (i.e., same for both modes) for core i after the first stage. The objective of this trimming stage is end up a pair of budgets (Q_i^L, Q_i^H) for that core, such that it remains schedulable and the following expression is maximised:

$$\underbrace{(Q_i^{stage-1} - Q_i^L)}_{L-budget\ trimming} + \underbrace{(Q_i^{stage-1} - Q_i^H)}_{H-budget\ trimming} \tag{30}$$

Since $Q_i^{stage-1}$ is the minimum static budget for which core i is schedulable, there can exist no feasible pair (Q_i^L, Q_i^H) such that $(Q_i^L < Q_i^{stage-1}) \wedge (Q_i^H < Q_i^{stage-1})$. To maintain schedulability, we can either (i) decrease at most of one of the two budgets in the initial pair ($Q_i^{stage-1}$, $Q_i^{stage-1}$) while the other one stays fixed or (ii) even *increase* one of them, if that allows decreasing the other budget by more than the same amount, given the interdependence of L-mode and H-mode memory budgets (Pareto principle). In any case, we have to consider multiple pairs and pick the one that minimises expression (30).

3) In the third stage, the heuristic tries to assign any remaining tasks using Memory-Fit. When trying out a task assigment, the target core's Q_i^L and Q_i^H are provisionally increased by the overall amounts reclaimed, for each mode respectively, from the preceding trimming. Upon a successful assignment, the additional memory is trimmed off again and used for other unassigned tasks in a similar manner.

B. AMC-max Gradient

We call this heuristic "Gradient" because it tends to assign H-tasks to smaller-indexed cores and L-tasks to higher-indexed cores, creating a gradient of sorts. This heuristic also has three stages. In the first stage, the memory bandwidth is evenly divided among cores and kept static across the mode switch. The task-to-core assignment is performed via First-Fit. When assigning H-tasks, the cores are considered in the order $1..m$; for L-tasks, the order is inversed. A task that cannot be assigned is set aside. The second and third stages of this approach are the same as those of AMC-max-dynamic.

C. Single-Step

This heuristic is similar to the third stage of previous two heuristics. It performs task-to-core assignment with Memory-Fit heuristic using the schedulability analysis that allows dynamic memory bandwidth allocation across mode switch. After each assignment, the memory bandwidth is trimmed off from each mode of operation to be used by the next task. In the trimming process, there may be more than one feasible pair of budgets but the final pair is selected based on the criterion mentioned in Equation 30. More details on the trimming process are presented in the Appendix.

VIII. EVALUATION

A. Experimental Setup

We used a Java tool [22] to implement our analysis and allocation heuristics. We performed experiments with synthetic workloads generated by the same tool and controlled by the following parameters.

- Task periods are log-uniform distributed in the $10-100$ msec range. We assume implicit deadlines (D_i=T_i), even if our analysis holds for constrained deadlines ($D_i \leq T_i$).
- Uunifast-discard [23], [24] is used to generate the L-mode task utilisations in an unbiased way. Then, the L-WCET of a task (C_i^L) is the product of its period and its L-mode utilisation.
- For each task, we randomly select the cache stall ratio $r = \frac{C_i^m}{C_i}$ from the SPEC2006 suite [5]. In turn, $C_i^{m|L} = r \cdot C_i^L$ and $C_i^{e|L} = C_i^L - C_i^{m|L}$.
- The fraction of H-tasks in the task set is user-defined.
- An H-task's C_i^H is a linearly scaled up value of its L-WCET with a user-defined factor k [25].

TABLE II: Overview of Parameters

Parameters	Values	Default
H-WCET scaling up factor	$\{2 : 0.5 : 6\}$	2
Number of cores (m)	$\{2, 4, 8\}$	4
Task-set size (n)	$\{8 : 4 : 24\}$	16
Fraction of H-tasks in τ	$\{0.2 : 0.05 : 0.8\}$	0.4
Cache stall ratio limit	$\{SPEC2006, 0.1 : 0.1 : 1\}$	SPEC2006
Inter-arrival time T_i	$10000us$ to $100000us$	N/A
Nominal L-mode utilisation	$\{0.1 : 0.1 : 1\}$	N/A
Memory access time L	$\{0.02us : 0.01us : 0.06us\}$	$0.04us$

- For each H-task, $C_i^{e|H}$ is uniformly distributed over $[C_i^{e|L},\ k \cdot C_i^{e|L}]$ and, in turn, $C_i^{m|H} = C_i^H - C_i^{e|H}$. This reflects an assumption that most of the pessimism in the H-WCET estimates typically comes from reasoning about memory accesses, rather than arithmetic.
- The memory bandwidth is equal to $\frac{P}{L}$.

We generate a task-set for a given target utilisation of $U = y \times m : y \in (0, 1]$. We used different random class objects to generate random periods, utilisations and r. Each random class object is seeded with different odd number and reused in successive replications [26]. For each set of input parameters, we generate 1000 random task-sets. The ordering of the task set also has an impact on the schedulability ratio. By default, each task set is indexed in descending order of U_i^L. In our experiments, this performs better than descending order of (κ_i, D_i), (κ_i, U_i^L), D_i or $C_i^{m|\kappa_i}/T_i$. The aforementioned parameters for different variables are summarised in Table II.

B. Results

We compare the following heuristics.

- **AMC-max-dynamic-dominant**: This is the first heuristic from Section VII.
- **AMC-max-static**: This corresponds to the output of the first-stage of AMC-max-dynamic-dominant. Therefore, it benefits from the new analysis, but not from the benefits of dynamic memory budgets across the mode switch.
- **AMC-max-dynamic-gradient**: This is the second heuristic from Section VII.
- **AMC-max-static-gradient**: This is a static version of AMC-max-dynamic-gradient that stops after its first stage, i.e., it does not benefit from dynamic bandwidth budgets across the mode switch.
- **Dynamic-single-step**: This is the third heuristic from Section VII.
- **AMC-rtb-static**: This heuristic (taken from state-of-the-art [6]) distributes the memory bandwidth unevenly among cores and the memory budgets remain static across the mode switch. It performs the task-to-core allocation via Memory-Fit and it tests the feasibility of the allocations on each core via stall-aware AMC-rtb [17] analysis. It is faster in terms of computation time but more pessimistic in schedulability analysis.

Due to space constraints, in each plot, we vary only single parameter while the rest are set to the defaults from

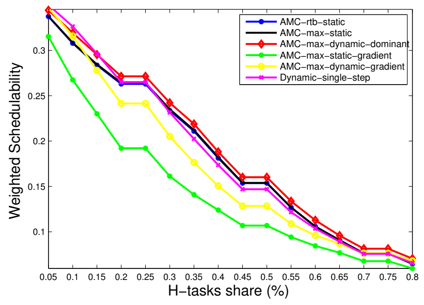

Fig. 2

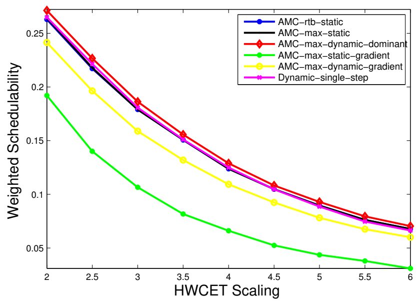

Fig. 3

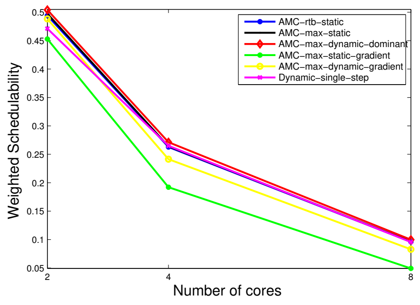

Fig. 4

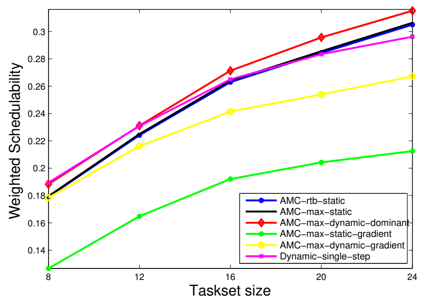

Fig. 5

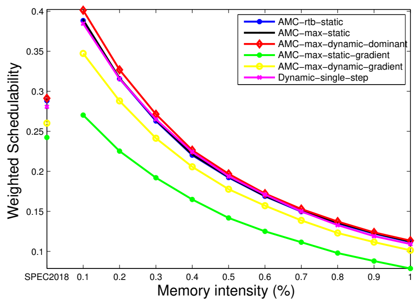

Fig. 6

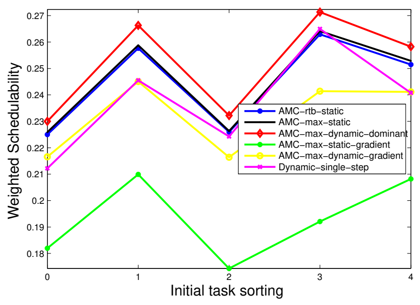

Fig. 7

Table II. For space-efficient presentation, we condense the results into plots of *weighted schedulability (WS)*. This performance metric [27], [28], condenses what would have been three-dimensional plots into two dimensions. It is a weighted average that gives more weight to task-sets with higher utilisation (i.e., supposedly harder to schedule). Let $S_y(\tau, p)$ represent the binary result (0 or 1) of the schedulability test y for a task-set τ with an input parameter p. Then $W_y(p)$, the weighted schedulability for that schedulability test y as a function p, is given by (31), where, $\bar{U}^L(\tau) = \frac{U^L(\tau)}{m}$ is the nominal L-mode utilisation.

$$W_y(p) = \frac{\sum_{\forall\tau} \left(\bar{U}^L(\tau) \cdot S_y(\tau, p)\right)}{\sum_{\forall\tau} \bar{U}^L(\tau)} \qquad (31)$$

Figures 2-7 present all the results. In general, there are two important observations.

1) The performance in terms of schedulability depends on three factors: (a) tightness of the analysis, (b) task-to-core allocation and (c) memory bandwidth allocation heuristics. A right selection of any of these factors can considerably improve the system's schedulability.
2) All heuristics exploiting dynamic memory bandwidth budgets across the mode switch always outperform their corresponding static counterparts.

Surprisingly, AMC-rtb-static and AMC-max-static perform quite well when compared to some heuristics that exploit dynamic memory bandwidth across mode switch. Their superior performance stems from better task-to-core allocation heuristics (Memory-Fit). AMC-max-dynamic-dominant, which uses Memory-Fit along with dynamic memory bandwidth budgets across the mode switch dominates AMC-max-static, which in turn outperforms AMC-rtb-static. We observed an absolute difference of 9% in pure schedulability ratios of AMC-max-dynamic-dominant and MC-max-static and 1.2% in weighted schedulability ratio. Similarly, maximum achieved absolute difference in schedulability ratios of AMC-max-dynamic-dominant and MC-rtb is approximately 9.1%, and 1.3% in terms of weighted schedulability ratio. AMC-max-dynamic-gradient, however, despite using better schedulability analysis, performs worse than AMC-rtb-static and AMC-max-static, due to worse task-to-core allocation heuristics. AMC-max-dynamic-gradient, which dominates AMC-max-static-gradient by design, outperforms the latter by a a big margin due to superior schedulability analysis. Dynamic-single-step outperforms AMC-rtb-static and AMC-max-static in the majority of the cases. However, it loses out to AMC-max-dynamic-dominant, because of the inefficiency of the greedy budget trimming.

The effect of variations in different parameters is discussed below. The increase in the share of H-tasks increases the system demand in H-mode, and hence, decreases the overall schedulability of the system as shown in Figure 2. Similarly, the system demand in H-mode also increases with a higher scaling-up factor for the task H-WCETs and this leads to decreased schedulability of the system (Figure 3). The increase in the number of cores increases the stall factor in the analysis and hence, the scheduability decreases as the number of cores increases (Figure 4). A larger task set size, for the same target utilisation, reduces any fragmentation issues and results in better schedulability (Figure 5). The increase in memory intensity reduces the schedulability due to higher stall values (Figure 6). In Figure 7, values of 0, 1, 2, 3 and 4 represent a sorting of the task sets in descending order of (κ_i, D_i), (κ_i, U_i^L), (D_i), (U_i^L) and ($\frac{C_i^{\kappa_i}}{T_i}$), respectively. In our experiments, (U_i^L) outperforms other sorting orders.

IX. Conclusions

In this work, we improved on the state-of-the-art for memory-regulation-aware mixed-criticality multicore scheduling theory by coming up with tighter AMC-max-based schedulability analysis and the possibility of dynamic adjustment of core memory budgets to the state. We consider this as one more step towards predictable mixed-criticality systems with good scheduling performance. Our experiments with different heuristics using synthetic task sets, showed an improvement of up to 9.1% in terms of pure scheduling ratio over state-of-the-art heuristic. In the future, we also want to experiment with per-task (rather than per-core) memory access budgets, dynamically adjustable at mode change, and compare the performance of the two scheduling arrangements.

Acknowledgements

This work was partially supported by National Funds through FCT (Portuguese Foundation for Science and Technology) within the CISTER Research Unit (CEC/04234), and by FCT and ERDF (European Regional Development Fund) within project nr. 29119 (PREFECT).

Appendix

Here we describe in more detail how the memory bandwidth trimming is performed at each core.

A. In stage 2 of the Dominant and Gradient heuristics

After stage 1, each core i has a pair of equal memory access budgets for both modes, i.e., $(C_i^{stage-1},\ C_i^{stage-1})$. For each core, the trimming process tries out all possible pairs $(C_i^L,\ C_i^H)$ with $C_i^L \in [C_i^{L-min},\ C_i^{stage-1} + Q_{free}^L]$ and $C_i^H \in [C_i^{H-min},\ C_i^{stage-1} + Q_{free}^H]$ to pick the one that minimises Expression 30, restated below:

$$\underbrace{(Q_i^{stage-1} - Q_i^L)}_{L-budget\ trimming} + \underbrace{(Q_i^{stage-1} - Q_i^H)}_{H-budget\ trimming}$$

The above terms correspond to the following quantities:

- C_i^{L-min} is the least value for the L-mode memory access budget (determined through binary-search-based sensitivity analysis) such that the core is schedulable in steady L-mode.
- C_i^{H-min}, respectively, for the steady H-mode.
- Q_{free}^L is the amount of unallocated memory budget for the L-mode; it maybe initially 0 at the end of stage 1 (i.e., at the start of the trimming stage 2) and after each additional core trimming it is updated to $Q_{free}^L := Q_{free}^L + Q_i^{stage-1} - Q_i^L$. This way, we ensure that we never overcommit the bandwidth in L-mode, at any point.
- Similarly for Q_{free}^H.

Under the Gradient heuristic, trimming performs better if cores are considered in descending index order. For simplicity, we use that same order also for the Dominant Heuristic.

B. Under the Single-stage heuristic

This heuristic performs trimming after every task assignment, on the assignment target core i. For trimming, it simply considers all pairs $(C_i^L,\ C_i^H)$ with $C_i^L \in [C_i^{L-min},\ C_i^{stage-1}]$ and $C_i^H \in [C_i^{H-min},\ C_i^{stage-1}]$ (otherwise the solution space would be enormous).

References

[1] A. Burns et al., "A survey of research into mixed criticality systems," ACM Computing Surveys, vol. 50, no. 6, pp. 82:1–82:37, Nov. 2017.

[2] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in Proceedings of the 28th IEEE Real-Time Systems Symposium, 2007.

[3] D. Dasari et al., "Identifying the sources of unpredictability in cots-based multicore systems," in Proceedings of the 8th IEEEInternational Symposium on Industrial Embedded Systems, June 2013, pp. 39–48.

[4] L. Sha et al., "Single core equivalent virtual machines for hard real-time computing on multicore processors," Univ. of Illinois at Urbana Champaign, Tech. Rep., 2014.

[5] G. Yao et al., "Schedulability analysis for memory bandwidth regulated multicore real-time systems," IEEE Transactions on Computers, vol. 65, no. 2, pp. 601–614, Feb 2016.

[6] M. A. Awan et al., "Mixed-criticality scheduling with memory bandwidth regulation," in Proceedings of the 55th ACM/IEEE Conference on Design Automation Conference, March 2018.

[7] S. Baruah et al., "Implementing mixed criticality systems in Ada," in 16th Ada-Europe Conference, 2011, pp. 174–188.

[8] S. K. Baruah et al., "Response-time analysis for mixed criticality systems," in Proceedings of the 32nd IEEE Real-Time Systems Symposium, 2011, pp. 34–43.

[9] T. Fleming et al., "Extending mixed criticality scheduling," in Proc. WMC, RTSS, 2013, pp. 7–12.

[10] H.-M. Huang et al., "Implementation and evaluation of mixed-criticality scheduling approaches for sporadic tasks," ACM Transactions on Embedded Computing Systems, vol. 13, no. 4s, pp. 126:1–126:25, Apr. 2014.

[11] K. Lampka et al., "A formal approach to the WCRT analysis of multicore systems with memory contention under phase-structured task sets," Journal of Real–Time Systems, vol. 50, no. 5, pp. 736–773, Nov 2014.

[12] H. Kim et al., "Bounding memory interference delay in COTS-based multi-core systems," in Proceedings of the 20th IEEE Real-Time and Embedded Technology and Applications Symposium, April 2014, pp. 145–154.

[13] M. Chisholm et al., "Cache sharing and isolation tradeoffs in multicore mixed-criticality systems," in Proceedings of the 36rd IEEE Real-Time Systems Symposium, Dec 2015, pp. 305–316.

[14] ——, "Supporting mode changes while providing hardware isolation in mixed-criticality multicore systems," in Proceedings of the 25th Conference Real-Time and Networked Systems, ser. RTNS '17, 2017, pp. 58–67.

[15] M. Paulitsch et al., "Mixed-criticality embedded systems – a balance ensuring partitioning and performance," in Proceedings of the 18th Euromicro Conference Digital System Design, Aug 2015.

[16] H. Yun et al., "Memory access control in multiprocessor for real-time systems with mixed criticality," in Proceedings of the 24th Euromicro Conference on Real-Time Systems, 2012, pp. 299–308.

[17] M. A. Awan et al., "Mixed-Criticality Scheduling with Dynamic Redistribution of Shared Cache," in Proceedings of the 29th Euromicro Conference on Real-Time Systems, ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 76, 2017, pp. 18:1–18:21.

[18] H. Yun et al., "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, April 2013, pp. 55–64.

[19] M. Joseph et al., "Finding Response Times in a Real-Time System," The Computer Journal, vol. 29, no. 5, pp. 390–395, 1986.

[20] N. Audsley et al., "Applying new scheduling theory to static priority preemptive scheduling," Software Engineering Journal, vol. 8, no. 5, pp. 284–292, 9 1993.

[21] N. C. Audsley, "On priority asignment in fixed priority scheduling," Information Processing Letters, vol. 79, no. 1, pp. 39–44, 2001.

[22] B. Nikolic et al., "SPARTS: Simulator for power aware and real-time systems," in Proceedings of the 8th IEEE International Conference on Embedded Software and Systems. Changsha, China: IEEE, Nov. 2011, pp. 999–1004.

[23] E. Bini et al., "Measuring the performance of schedulability tests," Journal of Real–Time Systems, vol. 30, no. 1-2, pp. 129–154, 2009.

[24] R. I. Davis et al., "Priority assignment for global fixed priority preemptive scheduling in multiprocessor real-time systems," in Proceedings of the 30th IEEE Real-Time Systems Symposium, 2009, pp. 398–409.

[25] A. Burns et al., "Mixed criticality systems-a review," Department of Computer Science, University of York, Tech. Rep, 2013.

[26] R. Jain, The art of computer systems performance analysis - techniques for experimental design, measurement, simulation, and modeling., ser. Wiley professional computing. Wiley, 1991.

[27] A. Bastoni et al., "Cache-related preemption and migration delays: Empirical approximation and impact on schedulability," Proceedings of OSPERT, pp. 33–44, 2010.

[28] A. Burns et al., "Adaptive mixed criticality scheduling with deferred preemption," in Proceedings of the 35rd IEEE Real-Time Systems Symposium, Dec 2014, pp. 21–30.