Mixed-criticality Scheduling with Dynamic Memory Bandwidth Regulation

Muhammad Ali Awan*, Konstantinos Bletsas*, Pedro F. Souto†, Benny Akesson‡, Eduardo Tovar*
*CISTER Research Centre and ISEP/IPP, Porto, Portugal
†University of Porto, FEUP-Faculty of Engineering and CISTER Research Centre, Porto, Portugal
‡ESI (TNO), Eindhoven, the Netherlands

***Abstract*—Mixed-criticality multicore system design must often guarantee both safety and high performance. Memory bandwidth regulation among different cores can be a useful tool for guaranteeing safety, as it mitigates the interference when accessing main memory. The use of mode changes and system models like Vestal's can help provide both safety, for critical functions, and scheduling performance, by efficiently utilising the platform. This work therefore combines per-core memory access regulation with the well-established Vestal model and improves 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 memory-regulation-aware and stall-aware schedulability analysis for such systems, based on AMC-max. By comparison, the state-of-the-art offered no option of dynamic adjustment of core budgets, and only offered regulation-aware schedulability analysis based on AMC-rtb, which is inherently more pessimistic. Finally, 3) we consider different task assignment and bandwidth allocation heuristics, to assess the improvement from the dynamic memory budgets and new analysis. Our results show improvements in schedulability ratio of up to 9.1% over the state-of-the-art.**

***Keywords*-mixed-criticality; dynamic memory bandwidth;**

I. Introduction

In *mixed criticality systems* (MCSs), functions of different criticality use the same hardware resources (e.g., cores, interconnects and memories). Since unpredictable interference from lower-criticality can be disastrous, this is traditionally avoided through rigid performance isolation [1], which can be inefficient in utilising the platform. In response, new scheduling theory, such as Vestal's model [2] and its variants, tries to enable better performance guarantees without compromising safety. However, in multicores, resource sharing between cores, if unaccounted for, can cause unpredicable timing behavior [3]. With respect to main memory, different cores contending for the memory can cause *stalling* of execution on the core. Such stalling must often be kept low and must be upper-bounded, on grounds of safety and certifiability.

Memory contention can be mitigated by using *regulation*, which is the approach of the Single-Core Equivalence (SCE) framework [4]. Each core gets a periodically-replenished *memory access budget*. If 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. Recently [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 (ordered lowest to highest). 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 utilisation, without compromising safety.

Compared to [6], this work brings three new contributions. First, it permits the cores' memory access budgets to be dynamically adjusted at mode change, matching their different requirements in each mode, for better schedulability. Secondly, the memory-regulation-aware schedulability test devised as part of this work is based on AMC-max [8], a more exact but more complex test, compared to AMC-rtb used in [6]. Finally, experiments with synthetic tasks and different task assignment and memory budget allocation heuristics, explore the improvement from the two first contributions.

II. Related Work

The most established variant [7] of Vestal's model assumes that each task has 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*, set to the lowest 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 testing relies on fixed-priority worst-case response time (WCRT) analysis, using the appropriate task WCETs. Two 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 fairly well. AMC-IA [10] slightly outperforms AMC-max.

Many works exist on mitigating interference on shared resources and integrating its effects to the schedulability analysis [4], [5], [11]–[16]. Yao et al. [5] offer stall-aware WCRT

analysis for a platform where all cores undergo memory regulation. However, that work was criticality-agnostic or assumed (as in [16]) critical and non-critical tasks do not use the same core. In comparison, in [6], we ported the regulation scheme from [5] to Vestal's model, which allows both critical and non-critical tasks on the same core, for efficiency, without compromising the schedulability of the critical tasks and system safety. The present work further (i) integrates the regulation-awareness to a more accurate, but more computationally intensive, schedulability test (AMC-max) and (ii) generalises the model by allowing the cores' memory access budgets to be dynamically adjusted at mode change, according to their different needs in each mode. This dynamic reallocation of resources at mode change, is something we also explored with cache resources in [17].

III. System model

Platform and memory regulation: We assume a platform with m identical cores that access the main memory via a single shared memory controller. A core can have multiple outstanding memory requests. Most of our assumptions are inspired by the SCE framework [4] – specifically 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. As in [5], we assume each memory access takes a constant time of L. Memory accesses are regulated by software (e.g., [18]) or hardware.

Each core i is assigned a pair of memory access budgets (Q_i^L, Q_i^H), one budget for each mode of operation. 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. Any core exceeding its access budget, in L-mode or H-mode, is stalled until the start of the next regulation period. The regulation periods are synchronised on all cores. The sum of the cores' memory budgets does not exceed the available memory bandwidth in the L-mode ($\sum_i \frac{Q_i^L}{P} \leq 1$) and the H-mode ($\sum_i \frac{Q_i^H}{P} \leq 1$). As in [5], CPU computation and memory accesses do not overlap in time.

Task model: We assume a set $\tau = \{\tau_1, \tau_2, \cdots, \tau_n\}$ of n independent mixed-criticality sporadic tasks. Each task has a relative deadline D_i, a minimum inter-arrival time T_i. and a criticality level κ_i. The tasks are partitioned to the cores offline; migrations are disallowed. Tasks on each core are premptively scheduled with fixed priorities. 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. However, in case of overrun of either its $C_i^{e|L}$ or its $C_i^{m|L}$ 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, resetting the cores' memory budgets to their Q_i^H value is delayed till the start of the subsequent $(r+1)^{th}$ regulation period, denoted as time $s' \geq s$. Hence, H-jobs executing in the r^{th} regulation period after the mode switch 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 the $(r+1)^{th}$ regulation period). H-WCETs need not be specified for the L-tasks. The system is schedulable if no 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. Stall Analysis

In L-mode, Yao's stall analysis [5] applies directly. A memory access can stall its core either (i) because of regulation (i.e., if the core's memory budget is exhausted), hence called a *regulation stall*, or (ii) because of concurrent memory accesses by other cores (a *contention stall*). Yao et al. bound the total stall of a task τ_i (omitting the core and task indexes for clarity) as follows: Let $b = \frac{Q}{P}$ (the core's *bandwidth*) and $r = \frac{C^m}{C}$ (the task's *stall ratio*). If $b \leq \frac{1}{m}$, then

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

If $b > \frac{1}{m}$ and $r = \frac{C^m}{C} < \frac{1-b}{(m-1)b}$, then

$$stall = (P - Q) + (m-1) \cdot Q \quad (2)$$

If $b > \frac{1}{m}$ and $r = \frac{C^m}{C} \geq \frac{1-b}{(m-1)b}$, then

$$stall = \begin{cases} (1+K_1)(P-Q) + r_1 & if\ C \leq (1+K_1)Q \\ \left(1 + \frac{C}{Q}\right)(P-Q) + r_2 & \text{otherwise} \end{cases} \quad (3)$$

$$\text{where, } K_1 = \left\lfloor \frac{C^e}{Q - RBS} \right\rfloor,\ RBS = \frac{P-Q}{m-1}$$

$$r_1 = \min\{P - Q, (m-1)(C^m - K_1 \cdot RBS)\}$$

$$r_2 = \min\{P - Q, (m-1)(C \% Q)\}$$

All 3 cases consider an initial regulation stall of $(P - Q)$, occuring in the worst case if a task is scheduled to run, but the core's budget is already exhausted. Since we are in the L-mode, $Q = Q^L$. The above equations from [5] assume that a single task is running on the core, so it is never preeempted: at any time, it is either executing, accessing memory or stalled. In Section V, we state how this is relaxed, with a composite task modelling multiple tasks; additional intuition in [19].

We now analyse the stall incurred by an H-task upon a mode switch, assuming that no other tasks are running on the same core. The stall expression developed will be used for response time analysis, as in [5]. Like Yao et al., we use L, the memory access latency, as the unit of time for all times, including P and Q. Therefore, we may refer to memory access time, C^m, as the "number of memory accesses".

Let s be the mode switch instant. To simplify the mathematical expressions, 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 independently bound 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 V-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 computation time in L-mode (resp. H-mode), i.e. before (resp. after) mode switch. Then:

$$C^{m|H} = C^{m|\ell} + C^{m|h} \tag{4}$$

$$C^{e|H} = C^{e|\ell} + C^{e|h} \tag{5}$$

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

$$ub(Stall^h) = single(C^m = ub(C^{m|h}), \quad C^e = ub(C^{e|h}), Q = Q^H) \tag{6}$$

where $single()$ is the worst-case stall according to [5], ignoring the initial regulation stall 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 (6) 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 (4) and (5), and Yao's analysis shows that the stall is non-decreasing with both C^m and C^e.

So, the challenge is to compute expressions for the lower bounds. Since a non-stalled task must be either computing or accessing memory, we use the following lower bounds:

$$lb(C^{e|\ell}) = max(0, s - ub(C^{m|\ell}) - ub(Stall^\ell)) \tag{7}$$

$$lb(C^{m|\ell}) = max(0, s - ub(C^{e|\ell}) - ub(Stall^\ell)) \tag{8}$$

These are safe but pessimistic, especially $lb(C^{m|\ell})$, since 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:

$$ub(C^{e|\ell}) = min(s, C^{e|L}) \tag{9}$$

$$ub(C^{m|\ell}) = min\left(\frac{s}{P} \cdot Q^L, C^{m|L}\right) \tag{10}$$

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 $Stall^\ell$. Moreover, the maximum stall in each regulation period is $P - Q$, hence:

$$ub(Stall^\ell) = min\Big(\frac{s}{P} \cdot (P - Q^L), \quad single(C^e = ub(C^{e|\ell}), C^m = ub(C^{m|\ell}), Q = Q^L)\Big) \tag{11}$$

where $ub(C^{e|\ell})$ and $ub(C^{m|\ell})$ are given by (9) and (10).

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 $ub(Stall^e ll)$ and $ub(Stall^h)$ are given by (11) and (6).

V. Schedulability Analysis

We integrate the memory regulation related stalls in the AMC-max scheme by considering 3 cases: L-mode steady operation, H-mode steady operation and mode-switch operation.

Schedulability in L-mode is tested using Yao's analysis [5], i.e., standard WCRT analysis with an added stall term, computed for a synthetic/composite task comprising all jobs running in the response time window of task τ_i under analysis:

$$R_i^{L(k+1)} = C_i^L + \sum_{j \in hp(i)} \left\lceil \frac{R_i^{L(k)}}{T_j} \right\rceil C_j^L + Stall(R_i^{L(k)}) \tag{12}$$

$$C_{comp}^{m|L} = C_i^{m|L} + \sum_{j \in hp(i)} \left\lceil \frac{R_i^{L(k)}}{T_j} \right\rceil C_j^{m|L} \tag{13}$$

$$C_{comp}^{e|L} = C_i^{e|L} + \sum_{j \in hp(i)} \left\lceil \frac{R_i^{L(k)}}{T_j} \right\rceil C_j^{e|L} \tag{14}$$

where $R_i^{L(k)}$ denotes the response time value after k iterations, and $Stall(R_i^{L(k)})$ is the memory stall, computed using Yao's

analysis [5], with $Q = Q^L$ (omitting core index) and a composite task in L-mode with parameters $C^m = C^{m|L}_{comp}$ and $C^e = C^{e|L}_{comp}$. In the first iteration ($k = 0$), $R_i^{L(k)}$ is initialised with the solution of Recurrence (12) without the stall term.

The worst-case response time in steady H-mode is computed similarly. The differences being that the composite task models the interference of only H-tasks whose priority at least that of τ_i, and the stall term is computed using Q^H rather than Q^L.

To upper-bound the response time of an H-task τ_i when a mode switch occurs s time units after its release, we extend standard AMC-max from [8] by adding a stall term:

$$R_i^{s(k+1)} = C_i^H + \sum_{j \in hpL(i)} \left(\left\lfloor \frac{s}{T_j} \right\rfloor + 1 \right) C_j^L \qquad (15)$$
$$+ \sum_{j \in hpH(i)} \left\{ M(j, s, R_i^{s(k)}) C_j^H + \left(\left\lceil \frac{R_i^{s(k)}}{T_j} \right\rceil - M(j, s, R_i^{s(k)}) \right) C_j^L \right\} + Stall(s, R_i^{s(k)})$$
$$R_i^H = \max(R_i^s), \quad \forall s \in \{0, 1, \ldots, R_i^L\} \qquad (16)$$

where $M(j, s, t) = \min \left\{ \left\lceil \frac{t - s - (T_k - D_k)}{T_k} \right\rceil + 1, \left\lceil \frac{t}{T_k} \right\rceil \right\}$ upper-bounds the number of jobs of τ_j in the interval(s, t).

As in [8], (15) 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. As in [8], we take the maximum of the response times for all values of s. 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.

We next derive the stall term using the concept of equivalent composite/synthetic task both for static, i.e., unmodified 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 can use AMC-max analysis to derive the $C^m = C^{m|H}_{comp}, C^e = C^{e|H}_{comp}$ parameters of the composite task as:

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

Because $Q = Q^L = Q^H$, Yao's single-criticality stall expression [5] is applicable and therefore we use:

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

to compute the stall, which is then used in Recurrence (15).

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 IV. 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 IV 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, as 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 upper-bounds as $M(k, s, R_i^s)$ the number of interfering H-jobs in H-mode, and then subtracts it from the upper bound of the number of interfering H-jobs in the response time window. The number of interfering H-jobs in L-mode may hence be underestimated, thereby leading to an unsafe estimate of the stall term in (15). For 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 applies. 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. 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 higher contention if:

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

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-mode, independently of their criticality:

$$ub(C^{m|\ell}) = C_i^{m|L} + \sum_{j \in hp(i)} \left(\left\lfloor \frac{s}{T_j} \right\rfloor + 1 \right) C_j^{m|L} \quad (17)$$

$$ub(C^{e|\ell}) = C_i^{e|L} + \sum_{j \in hp(i)} \left(\left\lfloor \frac{s}{T_j} \right\rfloor + 1 \right) C_j^{e|L} \quad (18)$$

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 $M(k, s, R_i^s)$ as upper bound for the number of interfering H-jobs in H-mode. Therefore, the parameters of the equivalent synthetic task are:

$$ub(C^{m|h}) = C_i^{m|H} - lb(C_i^{m|\ell}) + \sum_{j \in hpH(i)} M(k, s, R_i^s) C_j^{m|H} \quad (19)$$

$$ub(C^{e|h}) = C_i^{e|H} - lb(C_i^{e|\ell}) + \sum_{j \in hpH(i)} M(k, s, R_i^s) C_j^{e|H} \quad (20)$$

Applying single-criticality stall analysis [5] to a composite task with the parameters given by (19) and (20) 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, we assumed that the mode switch instant s occurs at regulation period boundaries, i.e. that it coincides with budget change instant s', therefore the stall analysis assumes that, upon mode switch, all H-jobs execute with Q^H in H-mode. If this is not the case, then some H-jobs may execute with Q^L during the (s, s') interval, despite already being 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.

VI. Bandwidth Allocation and Task Assignment

We devised the following task-to-core assignment and memory bandwidth allocation heuristics to explore the benefit of our analysis and the use dynamic memory budgets. We used Audsley's task priority assignment algorithm [20], 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:

Stage 1: This stage considers static memory budgets across the mode switch and uses stall-aware AMC-max analysis (from Section V) for feasibility testing. 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:

Stage 2: Using sensitivity analysis, we "trim off" from each core, any memory bandwidth that 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 Stage 1. The objective of this trimming stage is to pick a pair of budgets (Q_i^L, Q_i^H) for that core, such that it remains schedulable and Expression (21) is maximised:

$$\underbrace{(Q_i^{stage-1} - Q_i^L)}_{L-budget\ trimming} + \underbrace{(Q_i^{stage-1} - Q_i^H)}_{H-budget\ trimming} \quad (21)$$

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 (21).

Stage 3: We now try to assign any remaining tasks using Memory-Fit. In attempted assigments, 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. After every successful assignment, the additional memory is trimmed off again similarly, for reuse.

B. Single-Step

This heuristic is similar to the third stage of the previous heuristic. It assigns tasks to cores using Memory-Fit and the new stall-aware AMC-max analysis. After each assignment, the memory bandwidth is trimmed off from each mode of

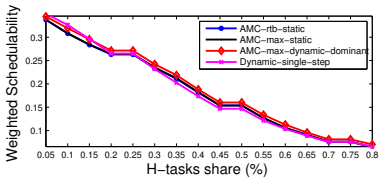

Fig. 1

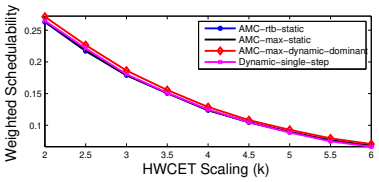

Fig. 2

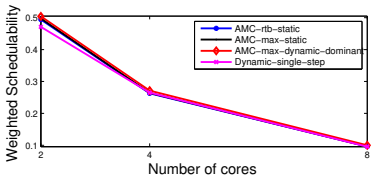

Fig. 3

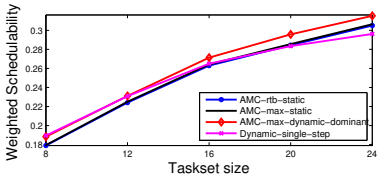

Fig. 4

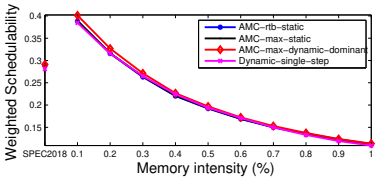

Fig. 5

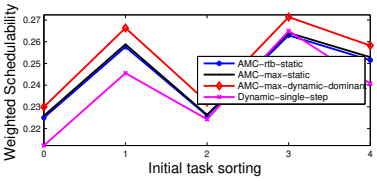

Fig. 6

operation to be used by the next task. In the trimming process, there may be multiple feasible budgets pairs but pair minimising (21) is chosen. More details on trimming in [19].

VII. Evaluation

We implemented the analysis and allocation heuristics in a Java tool [21] and generated synthetic workloads as follows: 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 [22], [23] is used to generate the L-mode task utilisations (u_i^L) in an unbiased way. Then, the L-WCET of a task (C_i^L) is equal to $T_i \cdot u_i^L$. 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.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 computation.

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 [24]. 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, tasks are indexed in descending order of U_i^L. In our experiments, this performs better than descending order of $(\kappa_i,\ D_i)$, $(\kappa_i,\ U_i^L)$, D_i or $C_i^{m|\kappa_i}/T_i$. Other default parameters are: $m = 4$, $k = 2$, $n = 16$, 40% H-tasks, $L = 0.04$ μsec.

The compared heuristics are: (i) **AMC-max-dynamic-dominant**: The first heuristic from Section VI. (ii) **AMC-max-static**: This corresponds to the output of the first-stage of AMC-max-dynamic-dominant. It benefits from the new analysis, but not from the dynamic memory budgets across the mode switch. (iii) **Dynamic-single-step**: The other heuristic in Section VI. (iv) **AMC-rtb-static**: The state-of-the-art from [6], for potentially uneven core budgets, but static across the mode switch. It uses Memory-Fit and stall-aware AMC-rtb analysis. For additional, worse-performing heuristics, see [19].

Due to space constraints, in each plot we vary a single parameter; the rest are set to the defaults. We also condense the results into plots of *weighted schedulability* [25], [26] (Fig. 1-6). Surprisingly, AMC-rtb-static and AMC-max-static perform quite well – a testament to their task-to-core allocation heuristic (Memory-Fit). AMC-max-dynamic-dominant, which uses Memory-Fit plus dynamic memory budgets dominates AMC-max-static, which in turn outperforms AMC-rtb-static. We observed an absolute difference of 9% in the pure schedulability ratios of AMC-max-dynamic-dominant vs AMC-max-static and 1.2% in weighted schedulability ratio. Similarly, maximum achieved absolute difference in schedulability ratios of AMC-max-dynamic-dominant and AMC-rtb is 9.1%, and 1.3% in terms of weighted schedulability. Dynamic-single-step outperforms AMC-rtb-static and AMC-max-static in most cases, but it loses out to AMC-max-dynamic-dominant, because of the inefficiency of the greedy budget trimming.

In Figure 6, values of 0 to 4 represent a sorting of the task sets in descending order of $(\kappa_i,\ D_i)$, $(\kappa_i,\ U_i^L)$, (D_i), (U_i^L) and $(\frac{C_i^{\kappa_i}}{T_i})$, respectively. In our experiments, (U_i^L) performs best.

VIII. Conclusions

This work 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. 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. We consider this as one more step towards predictable mixed-criticality systems with good scheduling performance.

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).

REFERENCES

[1] A. Burns and R. I. Davis, "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, B. Akesson, V. Nlis, M. A. Awan, and S. M. Petters, "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, M. Caccamo, R. Mancuso, J.-E. Kim, M.-K. Yoon, R. Pellizzoni, H. Yun, R. Kegley, D. Perlman, G. Arundale et al., "Single core equivalent virtual machines for hard realtime computing on multicore processors," Univ. of Illinois at Urbana Champaign, Tech. Rep., 2014.

[5] G. Yao, H. Yun, Z. P. Wu, R. Pellizzoni, M. Caccamo, and L. Sha, "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, P. Souto, K. Bletsas, B. Akesson, and E. Tovar, "Mixed-criticality scheduling with memory bandwidth regulation," in Proceedings of the 55th ACM/IEEE Conference on Design Automation Conference, March 2018.

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

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

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

[10] H.-M. Huang, C. Gill, and C. Lu, "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, G. Giannopoulou, R. Pellizzoni, Z. Wu, and N. Stoimenov, "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, D. de Niz, B. Andersson, M. Klein, O. Mutlu, and R. Rajkumar, "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, B. C. Ward, N. Kim, and J. H. Anderson, "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] M. Chisholm, N. Kim, S. Tang, N. Otterness, J. H. Anderson, F. D. Smith, and D. E. Porter, "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, O. M. Duarte, H. Karray, K. Mueller, D. Muench, and J. Nowotsch, "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, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha, "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, K. Bletsas, P. F. Souto, B. Akesson, and E. Tovar, "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, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha, "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. A. Awan, K. Bletsas, P. F. Souto, B. Akesson, and E. Tovar, "Mixed-criticality scheduling with dynamic memory bandwidth regulation (long version)," CISTER Research Center ISEP/IPP, Porto, Portugal, Technical Report, 2018, http://www.cister.isep.ipp.pt/docs/.

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

[21] B. Nikolic, M. A. Awan, and S. M. Petters, "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.

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

[23] R. I. Davis and A. Burns, "Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems," in Proceedings of the 30th IEEE Real-Time Systems Symposium, 2009, pp. 398–409.

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

[25] A. Bastoni, B. Brandenburg, and J. Anderson, "Cache-related preemption and migration delays: Empirical approximation and impact on schedulability," Proceedings of OSPERT, pp. 33–44, 2010.

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