Mixed-criticality Scheduling with Memory Bandwidth Regulation

Muhammad Ali Awan*, Pedro F. Souto†, Konstantinos Bletsas*, 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

***Abstract*—Mixed-criticality (MC) multicore system design must reconcile safety guarantees and high performance. The interference among cores on shared resources in such systems leads to unpredictable temporal behaviour. Memory bandwidth regulation among different cores can be a useful tool to mitigate the interference when accessing main memory. However, for mixed-criticality systems conforming to the (well-established) Vestal model, the existing schedulability analyses are oblivious to memory stalling effects, including stalls from memory bandwidth regulation. This makes it unsafe. In this paper, we address this issue by formulating a schedulability analysis for mixed-criticality fixed-priority-scheduled multicore systems using per-core memory access regulation. We also propose multiple heuristics for memory bandwidth allocation and task-to-core assignment. We implement our analysis and heuristics in a tool and evaluate them, performance-wise, through extensive experiments. Our experiments show that stall-oblivious schedulability analysis may be optimistic due to contention on shared memory resources.**

I. Introduction

There is a recent trend of mixed-criticality systems (MCSs) in many real-time embedded domains (automotive, avionics, aerospace), where functions of different criticalities use the same hardware resources, such as cores, interconnect and memories [1]. A deadline miss by a high-criticality task can be disastrous, so lower-criticality tasks must not interfere unpredictably. This can be achieved through performance isolation [1]. However, too rigid isolation wastes processing resources, so designers want (i) efficient use of processing capacity and (ii) schedulability guarantees for all tasks under typical conditions subject to (iii) ensured schedulability of high-criticality tasks in all cases. This is the philosophy of Vestal's model [2], [3], which views the system operation as different modes, whereby only tasks of a certain criticality or above execute. Different worst-case task execution times (WCETs) are assumed for the same task in each mode it can be a part of, with corresponding degrees of confidence. This permits less rigorous (less costly) WCET estimation for low-criticality tasks, without compromising safety. Vestal's model is the basis of most scheduling theory for MCSs [4].

Scheduling techniques for Vestal's model and their analyses can benefit from being combined with techniques intended to make the platform both more realistic *and* more predictable and amenable to analysis by incorporating the architectural details. A major such effort, albeit criticality-agnostic, is the Single-Core Equivalence (SCE) framework [5], which regulates access to different shared platform resources in order to make contention patterns predictable/amenable to analysis.

In this work, we import SCE's per-core memory bandwidth regulation mechanism into the schedulability analysis of a multicore MCS with static task-to-core-partitioning, scheduled under Fixed Priorities, i.e., the Adaptive Mixed-Criticality (AMC) scheduling algorithm [6]. For such a system, we formulate stall-aware schedulability analysis, (Section IV) extending the existing stall-oblivious AMC-rtb test. We then formulate various heuristics for (uneven, in the general case) memory bandwidth allocation to cores and for task-to-core assignment (Section V). Finally, we experimentally evaluate the scheduling performance of these heuristics, using the new schedulability test (Section VI). We establish that many task sets deemed feasible by stall-oblivious tests may be infeasible.

II. Related Work

In its classic form [2] [3], Vestal's versatile and well-established model for uniprocessor MCSs, assumes an ordered set of criticality levels. Each task has a period, a deadline, a (design) criticality level and a set of WCETs – one per criticality level and non-decreasing with respect to the latter. For this model, Baruah et al. [6] devised 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 [6]: AMC-rtb and the tighter, but more complex, AMC-max. Fleming and Burns [7] extended AMC to an arbitrary number of criticality levels and showed that AMC-rtb approximates AMC-max reasonably well. The later AMC-IA test [8] 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 [9], [10] and on integrating the effects of such interference in the schedulability analysis [1], [5], [11]–[13]. Yun et al. [13] analysed the response time of

Work partially supported by National Funds through FCT (Portuguese Foundation for Science and Technology) and co-financed by ERDF (European Regional Development Fund) under Portugal2020 program, within the CISTER Research Unit (CEC/04234).

978-3-9819263-0-9/DATE18/©2018 EDAA

critical tasks scheduled on a single core by regulating the memory access rates of cores running non-critical tasks only. This idea was generalized [12] by regulating all cores and through a corresponding schedulability analysis that finds the response time of all tasks. However, unlike Vestal's model, in [12] critical and non-critical tasks cannot co-exist on the same core, which is potentially inefficient in terms of resource usage. In comparison, in this paper, we port the regulation scheme from [12] 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.

III. Task Model and Overview of AMC Scheduling

Next, we formalise our MC task model and give an overview of the AMC-rtb schedulability analysis. Afterwards, we introduce the memory-regulation-related aspects of the system and discuss their implications on the schedulability analysis.

A. Task Model

Consider a platform with m identical cores and a set τ of n MC sporadic tasks (τ_1 to τ_n). Each task has a relative deadline D_i, a minimum interarrival time T_i and a criticality level κ_i, which is either low (L) or high (H). The tasks are partitioned to the cores offline and do not migrate at run-time.

We assume the same Vestal MC model as Baruah and Burns [6], which views the system operation as different modes, whereby only tasks of a certain criticality or higher execute. Different WCETs estimates are assumed for each task in different modes, with different confidence in their safety. In our case, with two modes (L and H), the L-mode WCET estimate (or L-WCET, for short) of a task τ_i is denoted as C_i^L and its (safe but pessimistic) H-WCET estimate as $C_i^H \geq C_i^L$. The system boots in L-mode and remains in L-mode as long as no task exceeds its L-WCET; but if that happens, then all L-tasks are stopped and the system switches to H-mode, where only the H-tasks execute. It follows that 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 transition). This must be verifiable offline, via schedulability tests that use the respective WCET estimates for each mode. Next, we discuss such tests, assuming fixed-priority scheduling, which in the context of Vestal's model, becomes AMC.

B. Adaptive Mixed Criticality (AMC) Scheduling

AMC is a fixed-priority-based mixed-criticality scheduling algorithm for platforms that can monitor for how long jobs have been executing. It supports multiple criticality levels, but here we only consider two, in accordance with our task model.

Under AMC, at any time, tasks are scheduled according to their fixed priority. The only implications of a mode change (triggered by some job exceeding its L-WCET estimate) are that (i) L-tasks are halted and (ii) different WCET estimates are henceforth assumed for the H-tasks. Therefore, AMC schedulability analysis relies on standard fixed-priority response time analysis [14], [15] and checks the schedulability in both operating modes, L and H, as well as during the mode transition interval. The latter starts at the moment of the mode switch and ends at the earliest idle instant. Checking the schedulability in L-mode is a straightforward application of standard WCRT analysis. For each task τ_i, we compare its deadline to its WCRT, computed using the L-WCETs:

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

where $hp(i)$ is the set of tasks with higher priority than τ_i. Likewise, for the steady H-mode, for each H-task τ_i, one would compute the WCRT using the H-mode WCETs of τ_i and all H-tasks with higher priority than τ_i. However, this steady H-mode analysis is subsumed by the analysis for the mode transition interval. If the system is schedulable during the transition, it will also be schedulable in steady H-mode.

For H-tasks caught in a mode transition, Baruah et al. [6] offer two tractable WCRT estimation techniques, based on solving recurrence relations. The first recurrence, *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 by higher-priority H-tasks. The key observation leveraged is that no L-task can execute beyond the L-mode response time of the task under analysis, R_i^L, because the mode change must occur before that instant. As for the interference by H-tasks, it is larger, the earlier the mode switch occurs. Therefore, it is upper-bounded assuming that the mode switch occurs immediately after the release of the task under analysis. Thus, an upper bound for the WCRT of a job by an H-task τ_i, caught in a mode change, is provided by R_i^*, the solution to recurrence:

$$R_i^* = C_i^H + \sum_{\tau_j \in hpH(i)} \left\lceil \frac{R_i^*}{T_j} \right\rceil C_j^H + \sum_{\tau_\ell \in hpL(i)} \left\lceil \frac{R_i^L}{T_\ell} \right\rceil C_\ell^L \quad (2)$$

where $hpL(i)$ and $hpH(i)$ are the sets of L-tasks and H-tasks, respectively, with higher priority than τ_i.

AMC-rtb analysis is straightforward, but pessimistic, since the worst-case interference by L-tasks cannot occur simultaneously with the worst-case interference by H-tasks. The tighter but more elaborate AMC-max analysis [6] considers the key instants when the mode switch may occur and takes the maximum response time obtained for each of these instants. For simplicity, we use AMC-rtb rather than AMC-max, even though our approach can also be adapted to the latter.

C. Memory Access Regulation Model

The above discussion is oblivious to aspects related to memory accesses (to caches or main memory) and any regulation thereof. To integrate such aspects, we mostly adopt the SCE [5] assumptions, more specifically those of [12]:

We assume that all cores access main memory via a single shared memory controller. The combined policy of both the memory controller and its interconnect is round-robin [9], [12]. The last-level cache is either private or partitioned to each core. Like Yao et al. [12], we assume that each memory access takes a constant time L and that access to main memory is regulated (by the Memguard software mechanism [9] or in hardware). Specifically, each core i has a *memory access budget* Q_i, which is the maximum allowed memory access time within a *regulation period* of length P. These budgets are set at design time and may be uneven across cores. The budget enforcement semantics are that a core i that consumes its memory access budget, Q_i, within a regulation period is *stalled* until the start of the next regulation period. Regulation periods on all cores are synchronised. The *memory bandwidth share* of core i is $b_i = Q_i/P$. By design, $\sum_i b_i \leq 1$, i.e. the bandwidth is not overcommitted. Finally, Yao et al. [12] assume that CPU computation and memory access do not overlap in time; therefore, the WCET of task τ_i is $C_i = C_i^m + C_i^e$, where C_i^m is the memory access time and C_i^e the CPU computation time.

D. Memory Bandwidth Regulation Analysis

For the above memory regulation model, Yao et al. provide schedulability analysis in [12] for *single-criticality* systems using partitioned fixed-priority scheduling, with priorities assigned according to the Rate-Monotonic policy. We summarize that analysis below and adapt it to a MC model in Section IV.

a) Stall Analysis: When performing a memory access, as a result of a cache miss, the core may be stalled 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*. The analysis in [12] assumes that a task is not preempted and therefore it is executing, accessing memory or is stalled at any time. As a result, three worst-case memory access patterns, depending on the core's bandwidth b as well as on the task's cache stall ratio $r = \frac{C_i^m}{C_i}$ are identified. Because of space limitations, we just briefly describe the different cases/patterns, for details please check [12]. Note that we omit the index of a 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:

$$stall = \begin{cases} \frac{C^m}{Q}(P\text{-}Q)\text{+}(m\text{-}1)Q & \text{if } C^m\%Q\text{=}0 \\ \left\lceil \frac{C^m}{Q} \right\rceil (P\text{-}Q)\text{+}(m\text{-}1)(C^m\%Q) & \text{otherwise} \end{cases} \quad (3)$$

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:

$$stall = (P - Q) + (m - 1) \cdot Q \quad (4)$$

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:

$$stall = \begin{cases} (1 + K_1)(P - Q) + r_1 & if\ C \leq (1 + K_1)Q \\ \left(1 + \frac{C}{Q}\right)(P - Q) + r_2 & \text{otherwise} \end{cases} \quad (5)$$

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

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

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

Algorithm 1 from [12] summarizes the computation of an upper bound on the worst-case stall of a non-preemptive task and is presented here to facilitate the understanding of the approach we are proposing for MCSs.

Algorithm 1 Non-preemptive Task Worst-Case Stall

Input: System and task parameters: C, C^m, C^e, r, P, Q, m
Output: Worst-case stall for this task
1: **if** $b = \frac{Q}{P} < \frac{1}{m}$ **then** compute stall as Eq. (3)
2: **else if** $r = \frac{C^m}{C} \leq \frac{1-b}{b \cdot (m-1)}$ **then** compute stall as Eq. (4)
3: **else** compute stall as Eq. (5) **end if**

b) Schedulability Analysis: The schedulability analysis in [12] relies on standard fixed-priority response time analysis [15], 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. [12] consider a synthetic task equivalent to all the activations that occur in the response time window of the task under analysis and uses the following recurrence:

$$R_i^{(k+1)} = C_i + \sum_{\tau_j \in hp(i)} \left\lceil \frac{R_i^{(k)}}{T_j} \right\rceil C_j + stall\big(R_i^{(k)}\big) \quad (6)$$

where the stall term, $stall\big(R_i^{(k)}\big)$, is computed using Algorithm 1 with these parameters for the equivalent synthetic task:

$$\begin{cases} C^{m(k)} = C_i^m + \sum_{\tau_j \in hp(i)} \left\lceil \frac{R_i^{(k)}}{T_j} \right\rceil C_j^m \\ C^{e(k)} = C_i^e + \sum_{\tau_j \in hp(i)} \left\lceil \frac{R_i^{(k)}}{T_j} \right\rceil C_j^e \end{cases} \quad (7)$$

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

IV. Stall-aware AMC-rtb Analysis

In this section, we extend the well-known AMC-rtb [6] MC schedulability analysis to also consider memory budgets.

A. Implications of Memory Regulation to the Task Model

Section III-A defines L-mode (C_i^L) and H-mode (C_i^H) WCET estimates for the tasks in the respective modes. As a first step towards bringing this mixed-criticality model in line with the analysis by Yao et al [12], we henceforth distinguish between worst-case CPU computation time ($C_j^{e|\bullet}$) and worst-case memory access time ($C_j^{m|\bullet}$), in each mode,

and define $C_i^L = C_j^{e|L} + C_j^{m|L}$ and $C_i^H = C_j^{e|H} + C_j^{m|H}$. Thus, in accordance with Vestal's principles, each H-task has two (C^e, C^m) pairs: $(C^{e|L}, C^{m|L})$ for the L-mode, and $(C^{e|H}, C^{m|H})$ for the H-mode, the latter being more conservative. On the other hand, there are fewer tasks contending for the memory in H-mode (since the L-tasks are dropped). Our memory-regulation-aware MC schedulability analysis, in the next section, leverages this fact to reduce pessimism.

B. Analysis

We now incorporate the stall terms derived for each mode by the analysis in [12] to the respective WCRT equations.

1) L-mode Analysis: In L-mode, AMC behaves like standard fixed-priority scheduling. Therefore, we can apply Yao et al. [12] analysis described in III-D, using the L-WCETs for both the L-tasks and the H-tasks. More specifically, we perform the analysis in decreasing priority order, and to compute the response time of task i we use recurrence:

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

where the $stall(R_i^{L(k)})$ is computed using Algorithm 1 from [12]. Furthermore, $R_i^{L(0)}$ is computed with standard response time recurrence without any stall term. If the WCRT of some task exceeds its deadline, the task set is unschedulable.

2) Mode Transition Analysis: We now merge the regulation-stall-aware analysis from [12] with the analysis of Baruah et al. [6] for mode switching. For simplicity, we consider the AMC-rtb analysis, although this approach could be applied to the other analyses from [6].

Upon a mode switch under AMC, L-tasks are halted and afterwards only H-tasks run. Therefore, an upper-bound on the WCRT of an H-task τ_i, assuming that the mode switch occurs before its completion can be given by R_i^*, the solution to (2). To take into account the regulation-induced stall, we add the stall term obtaining the following recurrence:

$$\begin{aligned} R_i^{*(k+1)} =& C_i^H + \sum_{\tau_j \in hpH(i)} \left\lceil \frac{R_i^{*(k)}}{T_j} \right\rceil C_j^H + \sum_{\tau_\ell \in hpL(i)} \left\lceil \frac{R_i^L}{T_\ell} \right\rceil C_\ell^L \\ &+ stall(R_i^{*(k)}) \end{aligned} \quad (9)$$

$$\begin{cases} C^{m(k+1)} = C_i^{m|H} + \sum_{\tau_j \in hp(i)} \left\lceil \frac{R_i^{*(k)}}{T_j} \right\rceil C_j^{m|\kappa_j} \\ C^{e(k+1)} = C_i^{e|H} + \sum_{\tau_j \in hp(i)} \left\lceil \frac{R_i^{*(k)}}{T_j} \right\rceil C_j^{e|\kappa_j} \end{cases} \quad (10)$$

Like for Recurrence (8), we compute the stall term with Algorithm 1 using an equivalent "synthetic task" that comprises the demand of both C^e and C^m of all the activations within the response time window. The synthetic task parameters C^e and C^m can be computed with Recurrence (10). Recurrences (9) and (10) can be initialized with R_i^L computed through Recurrence (8). In line with the assumptions made by AMC-rtb, we use $C^{e|L}$ and $C^{m|L}$ for all activations of L-tasks, and $C^{e|H}$ and $C^{m|H}$ for all activations of H-tasks.

The computation of the stall term via (9) is pessimistic. We use $R_i^{*(k)}$ as a stall term input to compute the parameters of the synthetic tasks via Recurrence (10). However, L-tasks are idled after the mode change, hence, all the activations of hpL(τ_i) tasks should be considered only within the R_i^L interval. In contrast, all activations of hpH(τ_i) tasks must be considered for the entire $R_i^{*(k)}$ interval. We refine the stall term for Recurrence (9) and denote it as $stall(R_i^L, R_i^{*(k)})$ with two parameters. The synthetic task parameters for $stall(R_i^L, R_i^{*(k)})$ are computed with Recurrence (11). The rest of the procedure remains the same, the stall is computed with Algorithm 1 and Recurrence (9) is initialised with R_i^L.

$$\begin{cases} C^{m(k+1)} = C_i^{m|H} + \sum_{\tau_j \in hpL(i)} \left\lceil \frac{R_i^L}{T_j} \right\rceil C_j^{m|L} + \sum_{\tau_\ell \in hpH(i)} \left\lceil \frac{R_i^{*(k)}}{T_\ell} \right\rceil C_\ell^{m|H} \\ C^{e(k+1)} = C_i^{e|H} + \sum_{\tau_j \in hpL(i)} \left\lceil \frac{R_i^L}{T_j} \right\rceil C_j^{e|L} + \sum_{\tau_\ell \in hpH(i)} \left\lceil \frac{R_i^{*(k)}}{T_\ell} \right\rceil C_\ell^{e|H} \end{cases} \quad (11)$$

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

We propose five heuristics for allocating memory bandwidth and tasks to the cores, and evaluate them in terms of system schedulabity. Like AMC, we use Audsley's algorithm [16] to assign task priorities, even though it is no longer necessarily optimal in the presence of stalls.

1) Even: All cores get the same share of the total memory bandwidth. Subject to this, the task-to-core assignment is performed using first-fit bin-packing.

2) Uneven: Initially, this heuristic also distributes the bandwidth evenly to the cores and employs first-fit for task assignment. However, instead of declaring failure whenever a task does not fit on any core, it sets that task aside, and moves on to consider the next task. Any tasks that remain unassigned, after considering all tasks are handled in-order as follows: Each core's bandwidth is "trimmed" to the minimum value that preserves schedulability, via sensitivity analysis (specifically, binary interval search). Let the total reclaimed bandwidth from all cores be B. A second round of first-fit tries to assign the remaining tasks, assuming that the bandwidth of the target core i is increased by B. Upon successfully assigning such a task, we trim anew the target cores's memory budget via sensitivity analysis, adjust the available reclaimed budget and move on to the next task in a similar manner.

3) Greedy-fit: Initially, the total memory bandwidth is assigned to the first core and the task-set is iterated over once (in a given order) to assign the maximum number of tasks to this core; if one task does not fit, we try the next one. Afterwards, the spare bandwidth on this core is reclaimed using sensitivity analysis, and is fully assigned to next core. This continues until all tasks are assigned or the cores run out.

4) Humble-fit: Similar to greedy-fit, except that when a task assignment fails, we move to the next core (attempting no more task assignments on the current one).

5) Memory-fit: Initially, $b_i = 0$ for every core i. Each task is assigned to the core i that requires the least increase to its b_i to accommodate it, subject to existing task assignments.

Obviously, we expect "Uneven" to outperform "Even" schedulability-wise, but run slower, as it explores larger solution space. "Greedy-fit" and "Humble-fit" aggressively optimise for the available core processing capacity, possibly at the cost of memory bandwidth. Conversely, "Memory-fit" optimises for bandwidth instead, possibly at the cost of processing resource usage. Hence, all these alternative techniques sample the solution space in different ways.

VI. Evaluation

1) Experimental Setup: We used a Java tool to generate synthetic workloads with the following control parameters. Task periods are generated with a log-uniform distribution in the range $10ms - 100ms$. We assume implicit deadlines ($D_i = T_i$), even if our analysis holds for constrained deadlines ($D_i \leq T_i$). The L-mode utilisation of each task is generated using UUnifast-discard [17], [18] for unbiased distribution of utilisation values. Task L-WCETs are derived by multiplying the task period with its L-mode utilisation. H-WCETs are computed by multiplying the L-WCET with a user-defined factor [4]. The fraction of H-tasks in the task-set is a user-specified parameter. The cache stall ratio $r = \frac{C_i^m}{C_i}$ of each task is randomly chosen from the SPEC2006 suite [12]; in turn, $C_i^{m|L} = r \times C_i^L$ and $C_i^{m|H} = r \times C_i^H$. Consequently, $C_i^{e|L} = C_i^L - C_j^{m|L}$ and $C_i^{e|H} = C_i^H - C_i^{m|H}$.

Each task-set is generated for a given target utilisation of $U = x \times m : x \in (0, 1]$. We created different objects of a random class, each seeded with different odd integers and reused in successive replication [19], to generate random values for periods, utilisations and r. For each set of input parameters, we generate 1000 random task-sets. Each task-set is indexed[1] in descending order of $\frac{C_i^{m|\kappa_i}}{T_i}$, except for Memory-fit that performs better with descending order of U_i^L. Table I summarises the parameters used in our experiments.

2) Results: We compare all heuristics from Section V along with one heuristic (*"AMC-rtb-FF"*) that uses *necessary* schedulability tests on each core. "*AMC-rtb-FF*" tests the feasibility assuming zero stalling, using standard stall-oblivious AMC-rtb analysis [6] and first-fit task allocation. It gauges the schedulability drop in stall-aware schedulability analysis due to memory contention.

Due to space constraints, in each plot we vary only one parameter; the rest conform to the defaults from Table I. Since the number of plots would still be too high to accommodate,

[1] In our experiment, we found that this ordering performs better than descending (κ_i, D_i), (κ_i, U_i^L), U_i^L, D_i or $C_i^{m|L}/T_i$.

TABLE I: Overview of Parameters

Parameters	Values	Default
H-WCET scaling up factor	$\{2 : 0.5 : 6\}$	2
Number of cores (m)	$\{2, 4, 8, 16\}$	4
Task-set size (n)	$\{8 : 4 : 32, 64\}$	16
Fraction of H-tasks in τ	$\{0.2 : 0.05 : 0.8\}$	0.4
Regulation period (P)	$\{1us, 10us, 100us, 1ms\}$	$100us$
Cache stall ratio limit	$\{0.1 : 0.1 : 1\}$	SPEC2006
Inter-arrival time T_i	$10ms$ to $100ms$	N/A
Nominal L-mode utilisation	$\{0.1 : 0.01 : 1\}$	N/A

we further reduce the number of plots instead of plotting outright the scheduling success ratio for each parameter combination, we condense this information into plots of *weighted schedulability (WS)*. This performance metric [20], [21], 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 (12), where, $\bar{U}^L(\tau) = \frac{U^L(\tau)}{m}$ is the nominal L-mode utilisation.

$$W_y(p) = \frac{\sum_{\forall \tau} \left(\bar{U}^L(\tau) \cdot S_y(\tau, p) \right)}{\sum_{\forall \tau} \bar{U}^L(\tau)} \quad (12)$$

A bigger fraction of H-tasks or a greater H-WCET scaling-up factor raises the H-mode demand, leading to lower WS as seen in Figures 1 and 2, respectively. A larger number of cores increases the contention and hence, decreases the WS (Figure 3). This experiment assumes a task-set size of $4 \cdot m$, instead of the default 16. A greater task-set size decreases the bin-packing fragmentation and hence leads to better WS for most heuristics (see Figure 4). To analyse the effect of memory intensity, we randomly select a task's r uniformly over an interval $(0, z]$. Values of z from 0.1 to 1 are used in different experiments. For reference, we also plot results for when the tasks randomly get their r values from the SPEC2006 benchmark suite. Greater memory intensity results in higher stall-related overheads and lower WS (Figure 5).

Figure 6 presents the *unweighted* schedulability success ratio for default parameters but for $m = 2$ because exhaustive search (added in this experiment) was intractable with $m = 4$. We can see from this and all other figures that Memory-fit, Uneven, Even, Humble-fit and Greedy-fit is the descending ordered list w.r.t. schedulability ratio. Greedy-fit and Humble-fit perform worse than the Uneven and Even approaches due to their too aggressive optimisation w.r.t. the use of processing resources at the cost of available memory bandwidth. Indeed, Humble-fit is less aggressive in that respect, which is why it performs better compared to Greedy-fit. On the other hand, Memory-fit, which optimises the use of memory bandwidth, performs better than Uneven. This means, for the given set of parameters, memory bandwidth is the scarcest resource. AMC-rtb-FF excludes the overheads of memory stall and hence, effectively serves as a ceiling on schedulability success ratio. In Figure 6, the "Exhaustive" algorithm checks the system's feasibility using our analysis, after considering all possible allocations of tasks to cores. The efficiency of the proposed

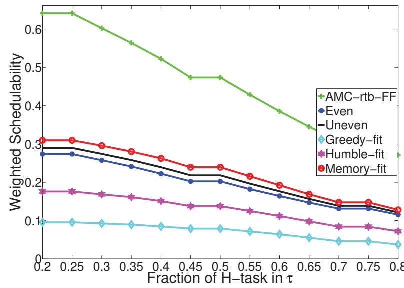

Fig. 1

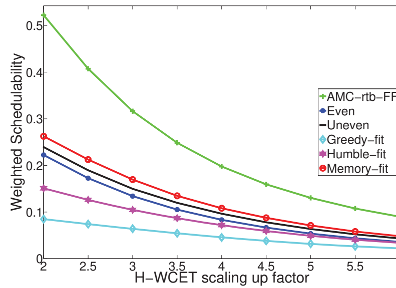

Fig. 2

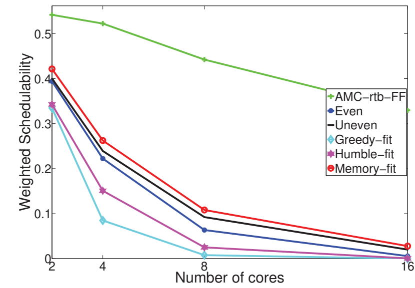

Fig. 3

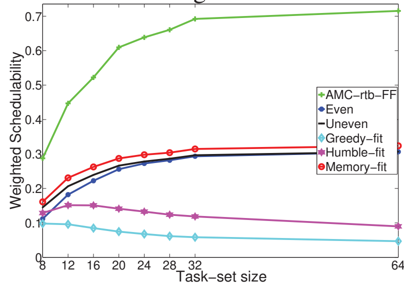

Fig. 4

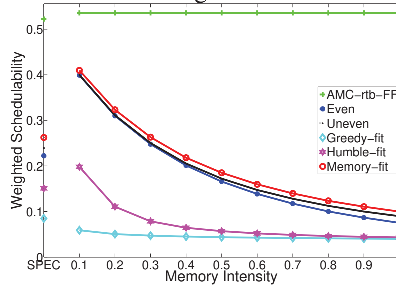

Fig. 5

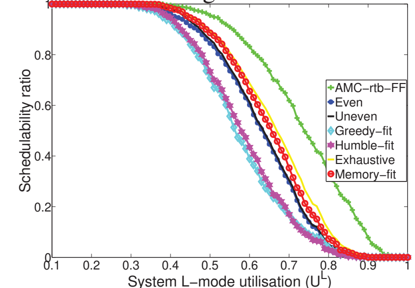

Fig. 6

heuristics is evident from their very small difference from the Exhaustive solution. Conversely, the wide gap between AMC-rtb-FF and the stall-aware Exhaustive algorithm shows that many task-sets deemed feasible by AMC-rtb-FF are not feasible with stall-aware schedulability tests and hence, underlines the significance of stall-awareness. Nevertheless, AMC-rtb-FF is not entirely oblivious to memory demand as task-set sorting is still based on $\frac{C_i^{m|\kappa_i}}{T_i}$, hence, there is a smaller dip in Figure 5 at 0. By construction, Greedy-fit and Humble-fit do not benefit from an increase in task-set size despite gains of low bin-packing fragmentation (Figure 4).

VII. Conclusions

We proposed task-to-core partitioning, memory bandwidth distribution and regulation-aware (stall-cognisant) schedulability analysis for MC fixed-priority-scheduled multicore systems. Combining Vestal's model with techniques intended to make the multicore platform more realistic and predictable, improves confidence in it and leads to an accurate and hence, safer schedulability analysis. Our results show that stall-aware analysis can eliminate false positives in schedulability testing and that for better schedulability it is important to optimize the memory bandwidth allocation and task-to-core assignment heuristic for the scarcest resource in the system. In the future, we intend to dynamically vary the memory bandwidth distribution at mode switch to further improve system schedulability.

References

[1] 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 18th DSD, Aug 2015.

[2] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in 28th RTSS, 2007.

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

[4] A. Burns and R. Davis, "Mixed criticality systems-a review," Department of Computer Science, University of York, Tech. Rep, 2013.

[5] 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.

[6] S. K. Baruah, A. Burns, and R. I. Davis, "Response-time analysis for mixed criticality systems," in 32nd RTSS, 2011, pp. 34–43.

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

[8] H.-M. Huang, C. Gill, and C. Lu, "Implementation and evaluation of mixed-criticality scheduling approaches for sporadic tasks," Trans. Emb. Comput. Syst., vol. 13, no. 4s, pp. 126:1–126:25, Apr 2014.

[9] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha, "Memguard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms," in 19th RTAS, April 2013, pp. 55–64.

[10] 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 20th RTAS, April 2014, pp. 145–154.

[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," J. Real–Time Syst., vol. 50, no. 5, pp. 736–773, Nov 2014.

[12] G. Yao, H. Yun, Z. P. Wu, R. Pellizzoni, M. Caccamo, and L. Sha, "Schedulability analysis for memory bandwidth regulated multicore real-time systems," Trans. Computers, vol. 65, no. 2, pp. 601–614, Feb 2016.

[13] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha, "Memory access control in multiprocessor for real-time systems with mixed criticality," in 24th ECRTS, 2012, pp. 299–308.

[14] M. Joseph and P. Pandya, "Finding Response Times in a Real-Time System," The Comp. J., vol. 29, no. 5, pp. 390–395, 1986.

[15] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. Wellings, "Applying new scheduling theory to static priority preemptive scheduling," Software Engineering Journal, vol. 8, no. 5, pp. 284–292, 9 1993.

[16] N. C. Audsley, "On priority assignment in fixed priority scheduling," Inform. Processing Lett., vol. 79, no. 1, pp. 39–44, 2001.

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

[18] R. I. Davis and A. Burns, "Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems," in 30th RTSS, 2009, pp. 398–409.

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

[20] 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.

[21] A. Burns and R. Davis, "Adaptive mixed criticality scheduling with deferred preemption," in 35rd RTSS, Dec 2014, pp. 21–30.