Memory Bandwidth Regulation for Multiframe Task Sets

Muhammad Ali Awan*, Pedro F. Souto†, Konstantinos Bletsas* , 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*—Timing analysis of safety-critical real-time embedded systems should be free of both optimistic and pessimistic aspects. The multiframe model was devised to eliminate the pessimism in the schedulability analysis of systems with tasks whose worst-case execution times vary from job to job, according to known patterns. However, this model is optimistic and unsafe for multicores with shared memory controllers, since it ignores memory contention, and existing approaches to stall analysis based on memory regulation are very pessimistic if straightforwardly applied. This paper remedies this by adapting existing stall analyses for memory-regulated systems of conventional Liu-and-Layland tasks to the multiframe model. Experimental evaluations with synthetic task sets (and different task and memory budget assignment heuristics) show up to 85% higher scheduling success ratio for our analysis, compared to the frame-agnostic analysis, enabling higher platform utilisation without compromising safety. We also explore implementation aspects, such as how to speed up the analysis and how to trade off accuracy with tractability.**

***Keywords*-memory access regulation; multiframe task model;**

I. Introduction

Modern embedded systems are ever more computationally intensive. This motivates a shift to commercial-off-the-shelf (COTS) multicore platforms, which offer significant advantages in terms of raw computing power, energy consumption, dimension and weight over single-cores. The transition to multicores brings the sharing of resources, such as cache, main memory and I/O devices among cores, which reduces platform costs, but also makes the temporal behaviour of applications harder to analyse. To meet the stringent timeliness requirements in safety-critical domains (e.g, avionics, automotive, health, automation and space), designers must therefore integrate the effects of interference on shared resources into the timing analysis [1], [2]. For each application domain, standards ([3]–[6]) codify the requirements to be met (including temporal ones) to achieve certifiability. In avionics, the certification authorities have also issued a position paper [7] with guidelines for meeting certification requirements on multicore platforms which specifically discusses interference on shared resources.

This work was partially supported by National Funds through FCT/MCTES (Portuguese Foundation for Science and Technology), within the CISTER Research Unit (UID/CEC/04234); also by the Operational Competitiveness Programme and Internationalization (COMPETE 2020) under the PT2020 Partnership Agreement, through the European Regional Development Fund (ERDF), and by national funds through the FCT, within project POCI-01-0145-FEDER-029119 (PREFECT).

Among the approaches devised for dealing with that issue, the Single-Core Equivalence (SCE) framework [8] provides several software-based mechanisms to mitigate the interference on shared cache, main memory and I/O devices. The cache is partitioned among the tasks via Colored Lockdown [9]. The memory bandwidth is regulated among cores by MemGuard [10], which assigns to each core a memory access budget, for use within a *regulation period*. Any core exceeding its allocated bandwidth is stalled until the start of next regulation period, when memory budgets are replenished. Existing techniques [11], [12] for computing (and integrating to the schedulability test) the worst-case memory stalls under such regulation assume a single worst-case execution time (WCET) estimate per task, decomposable into worst-case net processor computation and worst-case total memory access time. However, sometimes such a simplistic characterisation of task worst-case timing behaviour can cause platform under-utilisation and over-engineering.

Namely, many computing tasks actually have very different worst-case execution requirements from one instance (job) to another, but that variation follows a repetitive pattern. Modelling every job of such a task in the schedulability analysis with the maximum WCET of all jobs is pessimistic [13]. Such tasks are best modelled by the *multiframe task* model [14] – a generalisation of Liu and Layland's task model that captures the WCET variation pattern. However, the existing analysis for that task model ignores the contention for shared memory controllers and buses, so it is unsafe for safety-critical systems. Conversely, the memory stall analysis techniques mentioned earlier cannot be directly applied to multiframe tasks. Discarding frame information in order to apply them is pessimistic and may lead to the system be deemed unschedulable or increase system costs. Hence, new analysis is needed.

In response, our paper brings the following four contributions: (1) The worst-case memory stall with a memory regulation mechanism is derived on a shared memory controller and bus for a multiframe task model in a partitioned multicore platform setting. (2) The derived memory stall is integrated to new stall-aware schedulability analysis for the multiframe task model. Implementation aspects of the new analysis and tractability optimisations, at the cost of some pessimism, are also elaborated. (3) Five memory bandwidth and task-to-core assignment heuristics are proposed to achieve better

schedulability success ratio in the multiframe task-model. (4) An experimental evaluation demonstrates that our analysis achieves up to 85.1% of absolute difference in schedulability success ratio over single-frame task model and stall analysis techniques for task sets with different characteristics.

Next in this paper, Section II discusses the state-of-the-art in memory-stall-aware scheduling for single-frame tasks and in stall-oblivious schedulability analysis of multiframe tasks. The system model used in this work is presented in Section III. Some existing results serving as background for our analysis are discussed in Section IV. Section V presents the techniques for computing the worst-case memory stall and its integration into the analysis for fixed-priority multiframe task scheduling. Implementation aspects, such as speeding up the analysis and trading off accuracy with tractability are also discussed. Five task-to-core and memory bandwidth assignment heuristics are introduced in Section VI. The experimental evaluation of the proposed analysis and heuristics is presented in Section VII. Section VIII concludes.

II. Related work

A. Regulation of memory accesses

Several techniques for mitigating memory interference on shared channels in multicores [2], [10], [15]–[17] implement periodic servers in software to manage the memory budgets of the cores. Performance monitoring counters track the memory accesses issued by the cores. Any core exceeding its assigned memory budget is stalled until the start of the next memory budget cycle. We assume the same software-based memory-budget enforcement mechanism. Implementation-related issues of such mechanisms are discussed in [18].

Such regulation does mitigate memory interference but it also introduces new stalls that invalidate the existing schedulability analyses, unless the latter are adapted to account for them. Some efforts in this direction target partitioned fixed-priority preemptive scheduling algorithms [11], [15] and hierarchical scheduling [17]. The latter work is a server-based scheduling mechanism that provides isolation between independent applications scheduled and uses fixed-priority scheduling inside each server. Mancuso et al. [11], under their SCE framework [19], target fixed-priority-scheduled partitioned multicores. The periodic software-based memory regulation mechanism MemGuard [10] ensures that each core gets an equal share of memory bandwidth in each regulation interval (or period) and stalls until the end of the regulation period if that budget is depleted. The resulting memory regulation stalls are integrated into the schedulability analysis [11].

Yao et al. [20] and Pellizzoni and Yun [21] generalised the previous analysis [11] by allowing uneven memory budget assignment to cores, for greater efficiency when the cores memory access requirements are too diverse. Even so, in those works, the stall analysis for any core is agnostic to the memory budgets assigned to the other cores. This motivated Mancuso et al. [22] to explicitly consider the known memory access budgets of other cores for greater accuracy. Agrawal et al. [23] recently also proposed a dynamic memory bandwidth assignment mechanism that varies the budgets over time, based on the application requirements and formulated schedulability analysis for this arrangement. Awan et al. [24] improved on the SCE model [11] by considering periodic server-based scheduling with EDF and uneven per-server memory budgets assuming upper and lower bounds on the access time of a single memory transaction. The stall time analysis considers per-core memory budgets that are variable at run time and can provide inter-server isolation on the same core. An ILP [24] can find a budget assignment that satisfies the timing requirements of the tasks. Awan et al. [25] derived worst-case memory stall analysis for a memory-regulated multicore with two memory controllers and integrated the corresponding stall terms into the schedulability analysis for fixed-priority partitioned scheduling. Other related works integrate the memory regulation stall and effect of cache redistribution into the schedulability analysis of mixed-criticality systems [26]–[28].

B. Multiframe task model

The multiframe task model by Mok and Chen [14] generalises Liu and Layland's task model. Under this model, the WCETs of successive jobs by the same task can vary, according to a repeating pattern. For example, if a given task's "frame size" is N, there are up to N distinct WCETs for its jobs, with the k^{th}, $(k+N)^{th}$, $(k+2N)^{th}$, ... job ("frame") characterised by the same WCET. This is useful for tasks whose execution time varies greatly from job to another subject to a known pattern, as, e.g., in MP3 [29] or MPEG video frame decoding [30]. By leveraging such information, the analysis in [14] is much less pessimistic than using the highest WCET over all frames as the WCET of every job under Liu and Layland's classic analysis. Baruah et al. [13] considered the actual frame pattern, rather than an accumulatively monotonic reordered pattern as in [14], and produced more accurate analysis for the rate-monotonic scheduling of multiframe tasks. Extensions to the model [31] allow additional task attributes, other than the WCET, to differ among frames.

Memory-stall-aware schedulability analysis has not yet been formulated for multiframe task systems. However, since stall-oblivious analysis is unsafe even for Liu-and-Layland task systems, this trivially also holds for multiframe task systems, which are a generalisation of the former. Conversely, while stall-aware analysis exists for systems with a single per-task WCET estimate, disregarding information of frame WCETs in order to apply such analysis would be pessimistic (in the same way that modelling a multiframe task as a Liu-and-Layland task is), which may make the workload unschedulable on a given system or risks increasing system cost. In this paper, we formulate worst-case stall analysis for multiframe tasks under per-core memory access regulation, and integrate the stall terms into the schedulability analysis for the system.

III. System model

Platform and memory regulation: Our assumptions are mostly inspired by the SCE framework [19] – specifically [12]. In a multicore platform, K identical cores access main memory via a single shared memory controller. Cores can have multiple outstanding memory requests. Prefetchers and speculative units are disabled. The combined policy of both the memory controller and its interconnect is round-robin [10], [12]. The caches are either private to each core or partitioned among cores. Memory accesses are regulated by software (e.g., Memguard [10]) or hardware. As in [12], each memory access takes a constant time of L. Performance monitoring counters count the last-level cache misses by each core.

Memory accesses are regulated as follows. Each core k is assigned a memory access budget Q_k. This is the maximum number of memory accesses allowed in each *regulation period* of length P. As in [12], we measure Q_k and P in terms of L. These budgets are assigned at design time and may differ for each core. A core exceeding its allocated memory access budget is stalled until the start of the next regulation period. Regulation periods on all cores are synchronized. The memory bandwidth share of core k is $b_k = \frac{Q_k}{P}$. The sum of the core budgets does not exceed the available memory bandwidth of the system, i.e., $\sum_{k=0}^{K} \frac{Q_k}{P} \leq 1$. As in [12], CPU computation and memory accesses do not overlap in time. Unlike e.g. [32], [33], our model is agnostic with respect to the points in time when memory accesses may occur within a task activation, and hence imposes no particular programming model.

Task model: We assume a multiframe (MF) task-model, in which a task generates jobs of varying execution requirements with a regular pattern. More specifically, a task $\tau_i = (\hat{C}_i, D_i, T_i)$ is characterised by a vector of worst-case execution times $\hat{C}_i$, a relative deadline D_i, and a minimum inter-arrival time of T_i between any two consecutive jobs. The vector $\hat{C}_i = (C_i^0, C_i^1, \ldots, C_i^{F_i-1})$ provides the WCET of F_i consecutive jobs (or frames) that repeat every F_i jobs. The $f^{th}(f \geq 1)$ job of τ_i is denoted as τ_i^f and has an execution requirement of $C_i^{((f-1) \bmod F_i)}$. A set of F_i consecutive jobs of task τ_i is referred to as a *superframe* of τ_i.

A task-set $\tau = \{\tau_1, \tau_2, \cdots, \tau_\ell\}$ is composed of ℓ sporadic tasks. These tasks are statically partitioned to the cores (no migration). Tasks assigned to each core are scheduled with preemptive fixed-priority scheduling. Each task is assigned a distinct priority, e.g. using the deadline monotonic priority assignment algorithm [34]. The WCETs of all jobs in a superframe are computed in isolation on a core assuming no interference from other cores on the shared memory controller and its interconnect, e.g., using techniques and tools in [35]. Because we assume that CPU computation and memory accesses do not overlap in time, every element C_i^j of the $\hat{C}_i$ vector is decomposed into two parts: (a) CPU computation time, $C_i^{j|e}$, and (b) memory access time, $C_i^{j|m}$, and $C_i^j = C_i^{j|e} + C_i^{j|m}$. Therefore, we refer to the WCET of the $f^{th}(f \geq 1)$ job as the pair $\left(C_i^{((f-1) mod\, F_i)|e}, C_i^{((f-1) mod\, F_i)|m}\right)$. As for P, we express all timing parameter of tasks, including, $C_i^{f|e}$ and $C_i^{f|m}$, with $0 \leq f < F_i$, in time units of duration L, the memory access latency.

Algorithm 1 Non-preemptive Task Worst-Case Stall

Input: System and task parameters: C^m, C^e, P, Q, K
Output: Worst-case stall for this task

```
if b = Q/P < 1/K then                          ▷ Case 1
    Compute stall as Equation (1)
else if r ≤ (1−b)/(b·(K−1)) then               ▷ Case 2
    Compute stall as Equation (2)
else                                           ▷ Case 3
    Compute stall as Equation (3)
```

IV. Background

A. Memory-stall analysis for the single-frame task model

Our proposed schedulability analysis integrates the memory regulation stall based on the principles of Yao's stall analysis [12]. This section provides the background on this analysis, as it will be used in Section V. In Yao's stall analysis, 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*). Let $b = \frac{Q}{P}$ (the core's *bandwidth* share or ratio) and $r = \frac{C^m}{C^m + C^e}$ (the task's *stall ratio*), then Yao et al. bound the total stall of a task τ_i (omitting the core and task indices for clarity), as given in Algorithm 1.

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

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

$$stall = \begin{cases} (1 + A_1)(P - Q) + r_1 & if\ C \leq (1 + A_1)Q \\ \left(1 + \frac{C}{Q}\right)(P - Q) + r_2 & \text{otherwise} \end{cases} \quad (3)$$

$$\text{where, } A_1 = \left\lfloor \frac{C^e}{Q - RBS} \right\rfloor,\ RBS = \frac{P - Q}{K - 1}$$

$$r_1 = \min\{P - Q, (K - 1)(C^m - A_1 \cdot RBS)\}$$

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

All three cases consider an initial regulation stall of $(P - Q)$, occurring in the worst case if a task is scheduled to run, but the core's budget is already exhausted. The above equations from [12] assume that a single task is running on the core, so it is never preempted. At any time t, it is either executing, accessing memory, or stalled. The worst-case stall suffered by a task scheduled with other tasks via fixed-priority scheduling on a given core is computed by applying a single-task analysis on a "synthetic task". In addition to the task under analysis, it contains the CPU computation and the memory accesses of all higher priority jobs that may arrive during the response

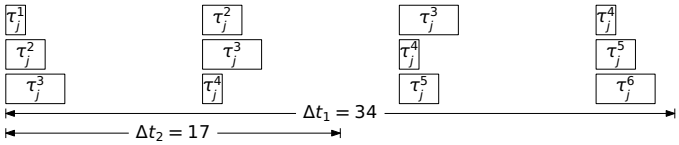

Fig. 1: Different job sequences of a task $\tau_j = ((1,0),(1,1),(2,1)),10,10)$, depending on their phasing.

time of the task under analysis. Yao et al. [12] integrates this worst-case memory stall for memory-regulated single-controller multicore architectures into the classical worst-case response time (WCRT) equation, as given in Equation (4).

$$R_i^{k+1} = C_i + IH(R_i^k) + Stall(R_i^k, \tau_i) \tag{4}$$

In (4), the WCRT for the single-frame task model (i.e., single WCET per task) has three components: i) the WCET of the task under analysis, ii) the interference from the higher priority workload $IH(R_i^k)$, and iii) the worst-case memory stall on a memory-regulated multicore platform $Stall(R_i^k, \tau_i)$. In Section V, we state how a synthetic task is modelled for multiple tasks in the context of a multiframe task-model.

B. Task interference in a multiframe task-model

The schedulability analysis for the multiframe task-model proposed by Baruah et al. [13] does not include the effect of memory stall. To compute the memory-stall oblivious response time in the multiframe task-model, Baruah et al. [13] defined functions $g_i(n)$ in Equation (5) and $G_i(t)$ in Equation (6) to compute the interference from higher priority tasks. The former computes the maximum cumulative execution demand of any sequence of n $(1 \leq n \leq F_i)$ jobs (or frames) of τ_i:

$$g_i(n) = \max_{j=0}^{F_i-1} \sum_{q=j}^{j+n-1} C_i^{q \bmod F_i} \tag{5}$$

The maximum cumulative execution demand $g_i(n)$ is hence the maximum sum of n successive jobs' WCET for τ_i, starting from any job in a superframe and wrapping around, if necessary. $G_i(t)$ uses $g_i(n)$ to compute τ_i's maximum execution demand in any interval of duration t:

$$G_i(t) = q \cdot g_i(F_i) + g_i(r) \tag{6}$$

where, $q = \lfloor \frac{\lceil \frac{t}{T_i} \rceil}{F_i} \rfloor$ and $r = \lceil \frac{t}{T_i} \rceil - (q \cdot F_i)$. Hence, for a task τ_i, the total interference from the higher-priority tasks $IH(t, \tau_i)$ for any time interval of t is given in Equation (7).

$$IH(t, \tau_i) = \sum_{j \in hp(i)} G_j(t) \tag{7}$$

V. Schedulability Analysis

The memory-aware WCRT analysis of Yao [12] cannot be directly applied to multiframe tasks because in this model each frame has its own WCET. This has two main consequences.

First, the analysis must ensure that every frame of every task is schedulable. If some frame of a task is not schedulable, then the task is not schedulable. Second, to upper-bound the interference by higher-priority tasks and the stall in each iteration of the worst-case response time (WCRT) recurrence of a frame, the analysis may need to consider more than just one sequence of jobs per higher-priority task. In the single-frame model, the worst-case behaviour of every job of a task is always the same. Therefore, to upper-bound the interference of a higher-priority task, it is sufficient to compute an upper-bound on the number of its jobs that may arrive in the response time window under consideration. In contrast, in the multiframe model, because different frames have different worst case behaviours, to determine an upper bound on the interference of a higher priority task, we may need to consider also the phasing of the sequence of interfering jobs in the response time window. Figure 1 illustrates 3 job sequences corresponding to different phasings of task $\tau = (((1,0),(1,1),(2,1)),10,10)$ with a superframe of 3 jobs. Over a time window of 34 time units, the job sequence that starts with frame τ^3 leads to more interference than the other two job sequences. On the other hand, for a time window of 17 time units, the sequence that starts with frame τ^2 is the worst. Generally, for a higher priority task τ_j with a superframe of F_j frames, there may be F_j different interfering job sequences for each response time window, depending on the first frame, and each of these sequences may lead to a different worst-case response time of the task under analysis.

For the memory-oblivious model, because the WCET is a scalar, [36] provides a simple and efficient method to determine the worst-case sequence of jobs per interfering task. In our model, the WCET is a pair, i.e. it is decomposed in a worst-case CPU computation time, $C_i^{f|e}$, and a worst-case memory access time, $C_i^{f|m}$, and these are not interchangeable because they affect the stall differently, see Algorithm 1. Thus it appears that it is not possible to compose the analysis in [36] with Yao's stall analysis.

A. Generalization of Yao's WCRT analysis for multiframe

To take into account that each frame in a multiframe task may have a different response time, instead of computing a single response time, R_i, per task τ_i, the analysis computes the response time, R_i^f, of every frame f in a superframe of task τ_i. Therefore the response time of multiframe task τ_i is:

$$R_i = max_{f=0}^{F_i-1} R_i^f \tag{8}$$

To take into account the phasing of the sequences of interfering jobs, in each iteration of the WCRT recurrence, instead of computing a single value, R_i^{k+1}, the analysis computes several values for the response time, one for each combination of the phases of the different higher priority tasks, and takes the maximum of these values. Next, we describe this procedure in detail.

As we have argued earlier, see Figure 1, for each task τ_j with priority higher than that of the task τ_i under analysis, and a given response time window, there are F_j job

sequences, where each job executes for its WCET. Thus, in a given response time window, the worst-case response time will occur for some combination of these job sequences, one sequence per higher priority task τ_j. Formally, let $S_{i,j}^{f|k}$ denote the set of F_j sequences of worst-case jobs of task τ_j in the response time window, $R_i^{f|k}$, computed in the k-th iteration of the WCRT recurrence for frame τ_i^f. For example, consider Figure 1, and assume that $R_i^{f|k} = 17$, then $S_{i,j}^{f|k} = \{(\tau_j^1, \tau_j^2), (\tau_j^2, \tau_j^3), (\tau_j^3, \tau_j^1)\}$. Each tuple in the Cartesian product of these S-sets:

$$\Theta_i^{f|k} = \prod_{j \in hp(i)} S_{i,j}^{f|k} \tag{9}$$

may lead to the worst-case response time in iteration $k+1$. Thus, in iteration $k+1$, we compute the worst-case response time for each tuple, $\theta \in \Theta_i^{f|k}$, using (10), and take the maximum of these values as the value $R_i^{f|k+1}$, see (11), to be used in the following iteration:

$$R_i^{f|k+1}(\theta) = C_i^f + IH(\theta) + stall(\tau_i^f, \theta) \tag{10}$$

$$R_i^{f|k+1} = max_{\theta_\ell \in \Theta_i^{f|k}} R_i^{f|k+1}(\theta_\ell) \tag{11}$$

where $IH(\theta)$ is just the sum of the WCET of all the jobs in the job sequences of tuple θ; and $stall(\tau_i^f, \theta)$ is the stall computed using Yao's stall analysis, see Algorithm 1, for a synthetic task composed of one job of the frame under analysis, τ_i^f, and of all the jobs in the job sequences of tuple θ.

We now provide the expressions to compute $IH(\theta)$ and the synthetic task parameters for Algorithm 1. Let σ_j be a sequence of jobs of task τ_j, and define:

$$(C_{\sigma_j}^e, C_{\sigma_j}^m) = \left(\sum_{k \in \sigma_j} C^e(k), \sum_{k \in \sigma_j} C^m(k) \right) \tag{12}$$

where $k \in \sigma_j$ denotes that job k is in sequence σ_j, and $C^e(k)$ and $C^m(k)$ denote the CPU execution time and memory access time of job k, respectively. Likewise, we define:

$$(C_\theta^e, C_\theta^m) = \left(\sum_{\sigma_j \in \theta} C_{\sigma_j}^e, \sum_{\sigma_j \in \theta} C_{\sigma_j}^m \right) \tag{13}$$

where $\sigma_j \in \theta$ denotes that job sequence σ_j is one of the sequences of tuple θ. Thus:

$$IH(\theta) = C_\theta = C_\theta^e + C_\theta^m = \sum_{\sigma_j \in \theta} \left(C_{\sigma_j}^e + C_{\sigma_j}^m \right) \tag{14}$$

Finally, the C^e and C^m parameters of the synthetic task for Algorithm 1 are given by:

$$(C^e, C^m) = (C_i^{f|e} + C_\theta^e, C_i^{f|m} + C_\theta^m) \tag{15}$$

In the first iteration of the WCRT recurrence (11), we use $R_i^{f,0}$, the WCRT obtained by applying the stall-oblivious analysis for multiframe tasks of Baruah et al. [13], to compute $S_{i,j}^{f|0}$ for all higher-priority tasks τ_j, and therefore $\Theta_i^{f|0}$.

B. Reducing the computational cost of the analysis

The analysis of Section V-A may be computationally very costly. For each task, it computes the WCRT of each frame of a superframe using a recurrence (8). That is, the total number of recurrences is $\sum_{i=0}^{n-1} F_i$, and the cost of each recurrence's iteration of the frames of task τ_i is proportional to $\prod_{j \in hp(i)} F_j$, by the definition of $\Theta_i^{j|k}$ (9). However, it is often possible to reduce the computational cost of the analysis, both the number of recurrences and the number of tuples that need to be considered in each iteration of a recurrence.

We consider the reduction in the number of recurrences first. For example, for task τ_j shown in Figure 1, both elements of the WCET of frame τ_j^3, $(2, 1)$, are larger or equal than the corresponding elements of the WCET of the other frames in superframe $(\tau_j^1, \tau_j^2, \tau_j^3)$. Therefore, to determine whether task τ_j is schedulable, it suffices to check if frame τ_j^3 is schedulable: if frame τ_j^3 is schedulable, so are τ_j^1 and τ_j^2, because they have a "smaller" WCET and the analysis is sustainable; if τ_j^3 is not schedulable, then τ_j is not schedulable and so it is irrelevant whether τ_j^1 or τ_j^2 are schedulable.

Formally, we define the $\leq$ partial-order:

Definition 1: Consider a set of pairs (C^e, C^m), where C^e is CPU computation time and C^m is memory access time. We define the partial-order $\leq$ among the elements of this set as:

$$(C_k^e, C_k^m) \leq (C_\ell^e, C_\ell^m) \Leftrightarrow C_k^e \leq C_\ell^e \wedge C_k^m \leq C_\ell^m$$

Consider frames τ_i^j and τ_i^k of task τ_i. Let us abuse the notation and write $\tau_i^j \leq \tau_i^k$ iff

$$(C_i^{j|e}, C_i^{j|m}) \leq (C_i^{k|e}, C_i^{k|m})$$

So, if $\tau_i^j \leq \tau_i^k$ and $j \neq k$, then we need not compute the WCRT of τ_i^j. Thus, we can reduce the set of frames of a task whose schedulability must be checked to the subset of these frames whose WCET is maximal under the $\leq$ partial-order. (From partially ordered sets' theory, an element is maximal, if there is no other element in the set that is "larger" than it.) Of course, if two frames have the same WCET, i.e. (C^e, C^m), we need to check the schedulability of only one of them.

Using the $\leq$ partial-order, one may also reduce the amount of computation in each iteration of a frame's WCRT recurrence. Now, rather than the $(C_i^{f|e}, C_i^{f|m})$ pairs corresponding to a given frame of a task, we consider the $(C_{\sigma_i}^e, C_{\sigma_i}^m)$ pairs corresponding to a job sequence, see (12). As before, we abuse the notation and write $\sigma_i \leq \sigma_j$ iff

$$(C_{\sigma^i}^e, C_{\sigma^i}^m) \leq (C_{\sigma^j}^e, C_{\sigma^j}^m)$$

In the analysis of the previous section, in iteration $k+1$ of a frame's WCRT recurrence, (11), we consider the set $S_{i,j}^{f|k}$ of all the F_j sequences of jobs with WCET of higher-priority task j that may fit in $R_i^{f|k}$. However, if $\sigma_j^m, \sigma_j^\ell \in S_{i,j}^{f|k} \wedge \sigma_j^m \leq \sigma_j^\ell$ and $m \neq \ell$, then there is no need to analyse the tuples with sequence σ_j^m: if frame τ_i^f is schedulable when subjected

to the interference of a tuple θ with σ_j^ℓ, then it will also be schedulable when subjected to the interference of the tuple obtained from θ by replacing sequence σ_j^ℓ with σ_j^m. Let $M_{i,j}^{f|k}$ be the set of maximal WCET of the job sequences in $S_{i,j}^{f|k}$ under the $\leq$ partial-order. The number of sequence tuples we need to analyse in the $k+1$ iteration can be reduced to:

$$\prod_{j \in hp(i)} |M_{i,j}^{f|k}|$$

where $|U|$ denotes the number of elements in set U.

One may further reduce the number of tuples that need to be analysed in each iteration of a frame's WCRT recurrence, by defining the $\leq$ partial-order over the $(C_{\theta_i}^e, C_{\theta_i}^m)$ pairs corresponding to tuples, see (13). However, once θ has been computed, the computation of $R_i^{n|k+1}(\theta)$ is comparable to the cost required to determine whether θ is upper-bounded by another job-sequence tuple in $\Theta_i^{f|k}$. Therefore our experiments (Section VII), forego this additional reduction step.

C. Trading-off tightness of the analysis for computation time

The reduction in the amount of computation time afforded by the use of the "maximal sets" in each step of the WCRT recurrence is case dependent. In the best case, i.e. if $M_{i,j}^{f|k}$ has only one element, the reduction per higher priority task τ_j, is by a factor of F_j, whereas in the worst case, i.e. each job sequence is maximal, there is no reduction.

When this reduction is not sufficient to make the analysis feasible, we can trade-off the tightness of the analysis for computation time. As observed in the previous subsection, to reduce the computational cost of the analysis, we need to reduce the number of tuples in each iteration of the WCRT recurrence without computing all tuples. Thus, the idea is to reduce the number of job sequences of higher priority tasks below the number of maximals, by adding sequences that upper-bound the elements in the "maximal sets". For example, assume that $M_{i,j}^{f|k} = \{(3,1),(1,3)\}$. If we add a sequence with a WCET of $(3,3)$, then in iteration $k+1$ of the WCRT recurrence, (11), it suffices to consider a single sequence for task τ_j rather than 2, thus reducing to half the computation cost in this iteration. These are virtual sequences in the sense that they do not correspond to job sequences that can occur in the response time window of the task frame under analysis. But this is not an issue. The WCRT recurrence requires the knowledge of only the WCET of the tuples, i.e. the (C_θ^e, C_θ^m), not the knowledge of the job sequences that lead to that WCET, and the WCET of a tuple can be computed from the WCET of the sequences that compose it, see (13).

Formally, let $U_{i,j}^{f|k}$ be a set of (C^e, C^m) pairs, and $V_{i,j}^{f|k}$ be the set of maximals of $M_{i,j}^{f|k} \cup U_{i,j}^{f|k}$ under the $\leq$ partial order; then it is safe to use $V_{i,j}^{f|k}$ instead of $M_{i,j}^{f|k}$ to generate $\Theta_i^{f,k}$ used in the WCRT recurrence (10) and (11). If $|V_{i,j}^{f|k}| < |M_{i,j}^{f|k}|$ we can reduce the computational cost of iteration $k+1$ of the WCRT recurrence.

A generic algorithm to generate $U_{i,j}^{f|k}$ from $M_{i,j}^{f|k}$ is to partition $M_{i,j}^{f|k}$ in subsets, and for each of these subsets compute the (C^e, C^m) that upper-bounds its elements under the $\leq$ partial-order, by taking the maximum of the CPU computation time and the maximum of the memory access time of all elements in that subset. In this algorithm, $V_{i,j}^{f|k}$ is equal to $U_{i,j}^{f|k}$. Different algorithms can be obtained from this generic algorithm depending on 1) the number of subsets of the partition; and 2) the way these subsets are generated.

The specifics of these algorithms are crucial to ensure that the schedulability test terminates, as we argue next. Assume the algorithm bounds the number of maximals in each iteration to the same value. Furthermore, assume that in iteration k, one needs to reduce the number of maximals, i.e. $U_{i,j}^{f|(k-1)} \neq \emptyset$, whereas in iteration $k+1$ there is no such need, $U_{i,j}^{f|k} = \emptyset$. Because reducing the number of maximals as described in the previous paragraph adds pessimism, it may be the case that $R_i^{f|(k+1)} < R_i^{f|k}$, and that from then on the response time estimate "oscillates" between these two values.

Clearly, termination is ensured if, for each interfering task, the maximals of the WCET pairs used in one iteration of the WCRT recurrence (11) upper bound the maximals used in the previous iteration. Formally, if $V_{i,j}^{f|k}$ is the set of maximals of $V_{i,j}^{f|k} \cup V_{i,j}^{f|k-1}$ under the $\leq$ partial-order for all $k > 1$, then the WCRT recurrence (11) terminates. There are many ways to satisfy this condition. A simple approach, suggested by this observation, is to set $U_{i,j}^{f|k}$ to the set of elements of $V_{i,j}^{f|k-1}$ that is not upper bounded by $M_{i,j}^{f|k}$.

The addition of (C^e, C^m) pairs to reduce the number of maximals leads to some extra pessimism that can be easily eliminated by using (C, C^e, C^m) triples. Consider, e.g., two job sequences, σ_1 and σ_2, of a given task with WCET $(C_{\sigma_1}^e, C_{\sigma_1}^m)$ and $(C_{\sigma_2}^e, C_{\sigma_2}^m)$, respectively, such that $C_{\sigma_1}^e > C_{\sigma_2}^e$ and $C_{\sigma_1}^m < C_{\sigma_2}^m$. If the $(C_{\sigma_1}^e, C_{\sigma_2}^m)$ pair is added to reduce the number of tuples, as described in this section, the computation of the IH term, (14), of the WCRT recurrence (10), uses $C_{\sigma_1}^e + C_{\sigma_2}^m$. However this value is larger than $max(C_{\sigma_1}^e + C_{\sigma_1}^m, C_{\sigma_2}^e + C_{\sigma_2}^m)$, which clearly upperbounds the interference, without stall, caused by any of these two job sequences. Thus, to mitigate some of the pessimism introduced by the use of "U-sets", we can modify the analysis so that the M, U and V are sets of $(C_\sigma, C_\sigma^e, C_\sigma^m)$ triples instead of the (C_σ^e, C_σ^m) pairs, the $\leq$ partial-order is extended to triples in the obvious way, and for the computation of the IH term, instead of (14) we use:

$$IH(\theta) = \sum_{\sigma_j \in \theta} C_{\sigma_j} \tag{16}$$

In the experiments in Section VII, we use a particular instance of the analysis described in this section, where we consider the singleton partition of each $M_{i,j}^{f|k}$ and therefore each $U_{i,j}^{f|k}$ has only one (C, C^e, C^m) triple with each component being determined by taking the maximum of the

corresponding component of all the triples in $M_{i,j}^{f|k}$:

$$C = max\{C_\ell : (C_\ell, C_\ell^e, C_\ell^m) \in M_{i,j}^{f|k}\}$$
$$C^e = max\{C_\ell^e : (C_\ell, C_\ell^e, C_\ell^m) \in M_{i,j}^{f|k}\}$$
$$C^m = max\{C_\ell^m : (C_\ell, C_\ell^e, C_\ell^m) \in M_{i,j}^{f|k}\}$$

We call this analysis the "fast analysis", because it reduces the most the computational cost of each iteration of the WCRT recurrence (11). On the other hand, it leads to the least tight upper-bound of the worst-case response time.

D. Implementation details

The description of the analysis may have given the impression that it is hard to implement because of the need to compute the WCET of the job sequences in the $S_{i,j}^{f|K}$ sets, but actually the implementation is not much different from that required by Barauah et. al. [13]. We keep one $F_i \times F_i$ matrix, W_i, per task τ_i. Element $W_i[j,k]$ of the W_i matrix is the WCET (the (C^e, C^m) pair) of the sequence of jobs of τ_i that starts with frame $j\,(0 \leq j < F_i)$ and has length $k\,(0 \leq k < F_i)$ (all elements of the first column are the pair $(0,0)$). Furthermore, we use a vector S, with one element per task τ_i; element $S[i]$ of S is the WCET of τ_i's superframe. Let:

$$\left\lceil \frac{R_i^{f|k}}{T_j} \right\rceil = F_j \times q^k + r^k \wedge 0 \leq r^k < F_j - 1$$

where q^k and r^k are non-negative integers, be the maximum number of arrivals of jobs of τ_j in $R_i^{f|k}$, the response time window of τ_i^f computed in iteration k of the WCRT recurrence (11). Then the set of the WCET of the job sequences in the $S_{i,j}^{f|k}$ set, is the set of (C^e, C^m) pairs, such that:

$$(C^e, C^m) = R_j[\ell, r^k] + q^k \times S[j], \ell = 0 \ldots (F_i - 1)$$

where we treat (C^e, C^m) pairs as vectors and use ordinary vector addition and multiplication of a vector by a scalar.

VI. Memory bandwidth allocation and task-to-core assignment heuristics

Five heuristics are devised for task-to-core and memory bandwidth assignment, to explore the benefits of our memory-aware schedulability analysis of multiframe tasks.

Even first-fit (Even-FF): All cores get an equal share of the memory bandwidth, i.e., $b_k = \frac{1}{m}, \forall k$. Subject to this memory bandwidth assignment, the tasks are assigned to cores using first-fit bin-packing.

Uneven first-fit (Uneven-FF): The heuristic assigns the tasks in at most two rounds. In each round, it uses first-fit for the unassigned tasks. It either terminates successfully, with all tasks assigned or fails, when some task in Round 2 cannot be assigned. Round 1 uses first-fit **assuming even budgets** for all cores. If some task cannot be assigned, it is set aside for Round 2, and the next task is considered. If Round 1 ends and there exist unassigned tasks, we trim-off the superfluous bandwidth from each core, using binary-search-based sensitivity analysis [37], [38]. This leaves each core with the minimum budget sufficient for the schedulability of the tasks assigned to it. In Round 2, the leftover tasks are tested for assignment on each core using first-fit, by adding all the reclaimed memory bandwidth to the target core's budget. Upon successful assignment, that core's budget is retrimmed using sensitivity analysis, with the reclaimed bandwidth used for the next task. Both first-fit-based heuristics (Even-FF and Uneven-FF) are designed to efficiently utilise the processing capacity of the platform.

Memory fit (MF): This heuristic prioritises the efficient use of memory bandwidth. Initially, each core is assigned a memory bandwidth of zero, i.e., b_k=0, $\forall k$, and the available bandwidth (i.e., not yet assigned to any core) is Q. For each task, we compute, for each core, the required increase in memory bandwidth if we added that task to the set of tasks already assigned to that core. A task τ_i is assigned to the core k that requires the smallest increase to its memory bandwidth b_k. Again, we use the binary search approach described in [37], [38] to compute this increase. This additional bandwidth requirement is subtracted from the available memory bandwidth. The heuristic terminates successfully when all tasks have been allocated, or else, if the available bandwidth becomes negative, the task set is not schedulable using this heuristic.

Memory density worst-fit (MDWF): This heuristic assigns tasks using worst-fit bin-packing based on the memory density of a core. The memory density of a task τ_i is defined as $MD(\tau_i) \equiv \sum_{f=0}^{F_i-1} \frac{C_i^{f|m}}{D_i}$. It corresponds to the average, over all frames, of the per-frame worst-case rate of memory accesses, over a job activation window. In turn, the memory density of a core k is defined as

$$MD_k \equiv \begin{cases} 0 & \text{if} \quad \tau(k) = \emptyset \\ \sum_{\tau_i \in \tau(k)} MD(\tau_i) & \text{otherwise} \end{cases} \tag{17}$$

where $\tau(k)$ is the set of multiframe tasks assigned to core k.

We now describe the heuristic. Initially, no core has any assigned task; the tasks are assigned one after the other. To assign task τ_i to a core, the cores are first sorted in non-decreasing memory density order. Next, the heuristic tentatively assigns task τ_i to each core, in order, and tests the schedulability of the core's resulting task set, assuming that all the available memory budget is added to the core's budget. Among all feasible target cores, the task is assigned to the one for whose memory density post-assignment would be the lowest. Then, the excess memory bandwidth assigned to the core is trimmed, to be made available for other tasks, if any. If a task is not schedulable in any core, then the taskset is not schedulable with this heuristic. Note that because initially, MD_k=0, $\forall k$, the first K tasks are assigned each to one of the K empty cores. This heuristic tries to balance the number of memory accesses originating from each core. Worst-fit bin-

packing heuristic has the inherent ability to balance the cores based on the target metric – in this case, memory density.

Total density worst-fit (TDWF): This heuristic is analogous to MDWF, but instead of memory density it attempts to balance using Worst-Fit the total execution density (i.e., computation plus memory accesses). The expectation is that it could perform better in systems whose workload is CPU intensive.

Let $\tau(k)$ be the set of multiframe tasks assigned to core k. We define the total density, TD_k, of core k, as follows:

$$TD_k \equiv \begin{cases} 0 & \text{if} \quad \tau(k) = \emptyset \\ \sum\limits_{\tau_i \in \tau(k)} \dfrac{\sum_{f=0}^{F_i-1} C_i^f}{F_i D_i} & \text{otherwise} \end{cases} \tag{18}$$

VII. Evaluation

A. Experimental setup

A Java tool was developed to implement the proposed analysis for multiframe task-model that incorporates the memory regulation stalls and to evaluate the scheduling performance of different budget assignment heuristics. This tool has two modules. The first module generates the synthetic workload (tasks with frames) with given input parameters for the specified multicore platform. The second module performs the schedulability analysis with the proposed task-to-core assignment and memory bandwidth allocation heuristics.

task set generation: Task periods are generated with a log-uniform distribution in 10 millisecond to 1 second range. Implicit-deadlines are assumed in this evaluation, though the analysis holds for the constrained deadline model. Initially, we generate the first frame of each task[1]. The given target utilisation is distributed among tasks with a UUnifast-discard algorithm [39]. This algorithm takes number of processors, task set size and target utilisation as an input and provides the distribution of utilisation among tasks. The WCET of the first frame of each task is $C_i^1 = T_i \times U_i$. The number of frames for each task are selected randomly within a range of $[1, \alpha]$, where α is an user-defined integer parameter. The WCET of the other frames of a tasks τ_i is generated with uniform distribution within $[\beta \times C_i^1, C_i^1]$, where $\beta \in (0, 1]$ is a user-defined bound in WCET variation of task's frames. A user-defined memory intensity parameter $\gamma \in [0, 1]$ is used to compute the memory access time and CPU computation time for each frame of a task. For a frame τ_i^f, $C_i^{f|m}$ is randomly selected within $[0, \gamma \times C_i^f]$ and $C_i^{f|e} = C_i^f - C_i^{f|m}$. The tasks are given unique priorities based on deadline monotonic priority assignment policy [34]. Each frame inherits the priority of the task.

The target utilisation is varied within a range of $(0, 1]$. We defined different random class objects to generate minimum inter-arrival time, utilisation, WCET of each frame, number of frames and memory accesses of each frame. Each random object is seeded with a different odd number and reused in successive replications [40]. For each set of input parameters, we generate 1000 random task sets. Each memory access is assumed to take 40 nanoseconds [12]. The regulation period length is assumed to be 100 microseconds. The parameters used in our experiments are summarised in Table I. The triple given in this table corresponds to {minimum : increment granularity : maximum} values of a parameter. Please note that default value of $\gamma = 0.5$ tends to generate more CPU bound workload. Similarly, the default value of $\beta = 0.2$ provides wider range to choose the frame's utilisation. This allows us to demonstrate that our proposed analysis can harness the benefits from this variation. Nevertheless, the effect of memory intensive workload and restricted variation in Frame's utilisation on our proposed analysis are also explored.

[1]For convenience and without loss of generality (since shift-rotation of the order of the frames results in an equivalent multiframe task), this is the frame with the greatest WCET.

TABLE I: Overview of Parameters

Parameters	Values	Default val.
Number of cores (K)	$\{2, 4, 8, 12\}$	4
task set size (ℓ)	$\{8, 12, 16, 20\}$	16
Bound on number frames (α)	$\{3 : 1 : 8\}$	6
Frame WCET variation (β)	$\{0.1 : 0.1 : 0.9\}$	0.1
Memory intensity (γ)	$\{0.1 : 0.1 : 0.9\}$	0.5
Regulation period (P)	$100\mu s$	$100\mu s$
Memory accesses time	$40ns$	$40ns$
Inter-arrival time (T_i)	$10ms$ to $1s$	N/A
Nominal utilisation	$\{0.1 : 0.05 : 1\}$	N/A

B. Results

Instead of providing plots in terms of scheduling success ratio for all heuristics, (i.e., the fraction of task sets deemed schedulable under the respective schedulability test), we provide *weighted schedulability* plots due to space limitation. This performance metric [41], [42] condenses a three-dimensional plots into two dimensions. It is a weighted average that gives more weight to task sets with higher utilisation that are supposedly harder to schedule. Using notation from [42], let $S_y(\tau, p)$ represent the result (0 or 1) of the schedulability test y for a given task set τ with an input parameter p. Then $W_y(p)$, the weighted schedulability for that test y as a function p, is $W_y(p) = \sum_{\forall \tau} (U(\tau) \cdot S_y(\tau, p)) \,/\, \sum_{\forall \tau} U(\tau)$. Here, $U(\tau)$ is the system utilisation for a given task set of τ.

We compare all the heuristics presented in Section VI. Each heuristic is tested with both the tight schedulability analysis, see Section V-B, and the fast schedulabily analysis, see Section V-C. These represent the two extremes in a range of possible analyses described in Section V; as implied by their names the "tight analysis" computes the tightest worst-case response time upper-bound, whereas the "fast analysis" does the fastest computation of an upper-bound at the expense of its tightness. To distinguish between the two, we add to the name of the heuristic a suffix, "-TA", for the tight analysis, and "-FA", for the fast analysis. In general, the tight analysis has approximately similar scaling up effect over

the fast analysis across all heuristics. Hence, to reduce the number of lines in the plots, only memory fit is plotted with a fast schedulability analysis (denoted as MF-FA). The naive analysis, which transforms each multiframe task into a single frame task with parameters $C_i^m = \max_{f=0}^{F_i-1} C_i^{f|m}$ and $C_i^e = \max_{f=0}^{F_i-1} C_i^{f|e}$ and $C_i = C_i^m + C_i^e$ and uses Yao's response time analysis, is used a baseline in our experiments. Furthermore, in the baseline, tasks are assigned using memory-fit, because in the evaluation of the single frame task-model in [27], memory-fit performed better than other heuristics. It is denoted as Yao-MF in all figures. The difference in weighted schedulability among density-based heuristics (total density, memory density) is very small, hence, only single heuristic of TDWF-TA is used to represent their performance.

Figure 2 presents the weighted schedulability for the different values of β (bound on the WCET variation of task's frames). The increase in β increases the execution demand of tasks' frames. Hence, the weighted schedulability decreases with an increase in β. The MF-TA efficiently utilises the memory bandwidth in those task set instances where it is the performance bottleneck and hence, performs better than other heuristics. The TDWF-TA heuristic balances the core utilisation based on total utilisation. At larger values of β the utilisation of the system increases and it becomes harder to balance the cores, consequently, the performance of TDWF-TA degrades compared to other heuristics. As expected, the Uneven-FF heuristic performs better than the Even-FF heuristic due to an additional task-to-core allocation loop and uneven memory bandwidth assignment. The Yao-MF heuristic performs worst when compared to proposed heuristics, due to the inherent pessimism in the construction of the equivalent Liu and Layland task-model. One important observation is that MF-FA performs better than TDWF-TA, Uneven-FF-TA and Even-FF-TA, even though it uses the fast schedulability analysis, which is more pessimistic. This shows that the effect of the task-to-core allocation and memory assignment may dominate that of the schedulability analysis used.

The weighted schedulability for different task set sizes is shown in Figure 3. More low-utilisation tasks are easier to schedule than fewer high-utilisation tasks due to bin-packing fragmentation. Hence, the weighted schedulability of the heuristics increases with the task set size for the same utilisation. The difference between Even-FF-TA and Uneven-FF-TA decreases with larger task set sizes. A greater task set size tends to result in more unscheduled tasks for the second phase of Uneven-FF-TA, which are harder to schedule with the remaining trimmed memory bandwidth. The unified task-to-core assignment and memory bandwidth allocation approach of TDWF-TA scales well with task set size.

The initial order of the task set plays an important role in the assignment, as shown in Figure 4. Tasks sorted in non-increasing order of total density ($\sum_{f=0}^{F_i-1} \frac{C_i^f}{D_i}$) performs better than non-increasing order of deadlines and non-increasing order of memory density ($\sum_{f=0}^{F_i-1} \frac{C_i^{f|m}}{D_i}$). The non-increasing total density sorting tends to assign high density tasks initially and low density tasks subsequently, making it easier for all the heuristics to find a successful assignment of tasks. This benefits more task-to-core heuristics that use first-fit bin-packing heuristics (Even-FF-TA and Uneven-FF-TA) than TDWF, which uses a task-to-core heuristic based on worst-fit.

TABLE II: Maximum absolute difference in schedulability success ratio of all heuristics from baseline Yao-MF in different experiments.

Heuristic	β	TSS	TO	α	K	γ
Even-FF-TA	69.5%	71.7%	67.1%	70.1%	74.5%	**76%**
Uneven-FF-TA	71.4%	72%	71.7%	73.4%	76.1%	**76.8%**
TDWF	70.4%	74.6%	70.4%	78.5%	74.2%	**80.9%**
MF-TA	76.2%	84.3%	77.7%	83.2%	83.9%	**85.1%**
MF-FA	75.3%	83.4%	76.9%	82.2%	82.9%	**84.9%**

The effect of the number of frames on all heuristics is presented in Figure 5. Except Yao-MF, other heuristics are essentially insensitive to the number of frames. By increasing the number of frames, we increase the possibility of a frame with either more memory access or more CPU computation. In the case of Yao-MF, this leads to an equivalent task with higher resource demands, and hence its performance degrades with an increase in the number of frames.

Figure 6 shows the weighted schedulability for different number of cores. In this set of experiments, the number of tasks is equal to four times the number of cores and the utilisation of the task set is proportional to the number of cores, so that the normalised utilisation is independent of the number of cores. The increase in the number of cores increases the memory stall as memory contention becomes much higher with more cores. Hence, the weighted schedulability of all heuristics decreases as number of cores increases. Please note that the memory bandwidth stays the same in this experiment and does not scale with the number of cores.

The increase in the memory intensity parameter, γ, increases the number of memory accesses. This leads to a higher memory stall and a reduction in weighted schedulability of all heuristics, as shown in Figure 7.

To provide a better sense of comparison among heuristics, Table II provides, for each experiment, the maximum absolute difference in the schedulability success ratio of all heuristics over Yao-MF in each experiment. In Table II, TSS and TO stand for the task set size and the initial task order experiments, respectively. Bold values in each row show the maximum observed difference between each heuristic and Yao-MF. In the best case with our experimental setup, we achieved up to 85.1% of absolute difference in schedulability success ratio when compared to baseline Yao-MF.

Finally, we performed experiments to analyse the running time of the proposed heuristics. We used a hardware platform with 16 GB RAM, 8 i7-4710MQ cores with maximum frequency of 2.5 GHz and running Linux Mint 18.3 Sylvia operating system. For the default parameters, Figure 8 presents

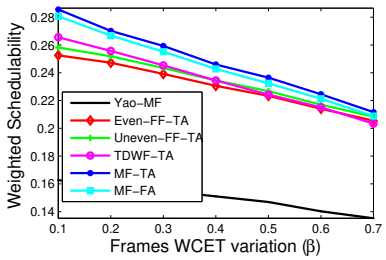

Fig. 2: WCET variation of task's frame

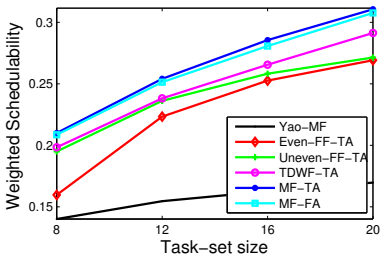

Fig. 3: Variation in task set size

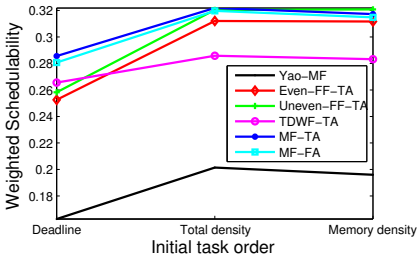

Fig. 4: Different task set sorting orders

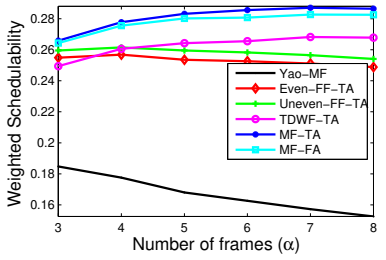

Fig. 5: Varying upper bounds on number of frames

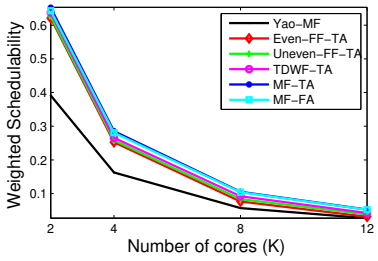

Fig. 6: Platforms with different number of cores

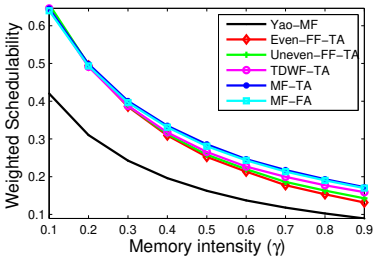

Fig. 7: Variation in memory intensity parameter

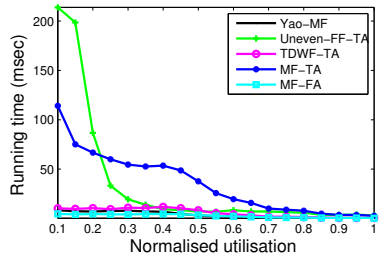

Fig. 8: Average running time against different utilisation values

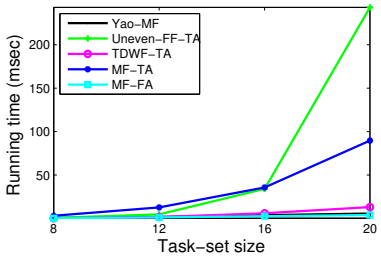

Fig. 9: Average running time vs. different task set sizes

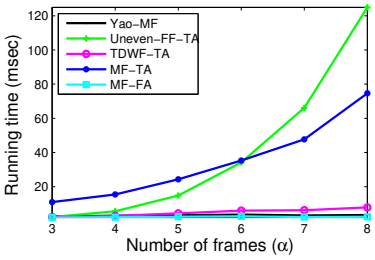

Fig. 10: Average running time for different upper bounds on No. of frames

the average running time of heuristics (per task set analysed) in msec for different task set utilisation values. In general, Uneven-FF-TA and MF-TA consume more time when compared to other heuristics. The latter checks the feasibility of each task on every core to efficiently utilise the memory bandwidth and hence, consumes more time. Another important observation is that even though the MF-TA is a significant improvement over the exhaustive analysis (more than 4 orders of magnitude for the default parameters in terms of tuples, i.e., Equation (10)), its average running time is in a reasonable range of 100 msec and achieves up to 3.7% of absolute (non-weighted) difference in schedulability success ratio when compared to MF-FA, the latter is more than one order of magnitude (11x) faster. The running time of all heuristics decreases with increasing utilisation due to increasingly many task sets failing early.

Figures 9 and 10 show the average running time of the heuristics for different task set sizes and α values, respectively. In these experiments, the average running time is computed across all task set utilisation values. As expected, the running times of the heuristics increase with larger task set sizes and with more frames. Note that Uneven-FF-TA "overtakes" the other heuristics in long running time with larger task set sizes and more frames due to the additional computation in the second phase of the heuristic. So, MF-FA is almost as good as MF-TA, but scales much better in terms of computation time.

VIII. Conclusions

This paper proposes a stall-aware schedulability analysis for multiframe task sets on multicores systems with memory access regulation. The multiframe task model accurately represents the worst-case processing requirements of tasks whose execution times vary according to a known pattern and this promotes efficient platform utilisation, for certain workloads such as media decoders. Meanwhile, memory regulation and its stall analysis improve timing predictability, which is crucial in the design of critical systems. Our experiments with synthetic task sets confirm improvement in scheduling success ratio for our approach of up to 85%(absolute), compared to the frame-agnostic state-of-the-art stall-aware analysis. We also proposed multiple optimisations for the running time of our (anyhow tractable) analysis, achieving approximately 11-fold speedup with little sacrifice of accuracy (no more than 3.7% drop in schedulability ratio).

REFERENCES

[1] 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.

[2] J. Nowotsch et al., "Multi-core interference-sensitive WCET analysis leveraging runtime resource capacity enforcement," in Proceedings of the 26th Euromicro Conference on Real-Time Systems, July 2014, pp. 109–118.

[3] Avionics Application Software Standard Interface, Part 1, Required Services, ARINC SPECIFICATION 653P1-3 ed., AERONAUTICAL RADIO, INC., Nov. 2010.

[4] ISO, "Road vehicles – Functional safety," 2011.

[5] RTCA, Inc., RTCA/DO-178C. U.S. Dept. of Transportation, Federal Aviation Administration, 2012.

[6] IEC, "Functional safety of electrical/electronic/programmable electronic safety-related systems," 2010.

[7] "Certification authorities software team (CAST), position paper (CAST-32A) multicore processors," Certification authorities in North and South America, Europe, and Asia, November 2016.

[8] L. Sha et al., "Single core equivalent virtual machines for hard real—time computing on multicore processors," University of Illinois, Tech. Rep., 2014.

[9] R. Mancuso et al., "Real-time cache management framework for multi-core architectures," in Proceedings of the 19th IEEE Real-Time and Embedded Technology and Applications Symposium, 2013, pp. 45–54.

[10] 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.

[11] R. Mancuso et al., "WCET(m) estimation in multi-core systems using single core equivalence," in Proceedings of the 27th Euromicro Conference on Real-Time Systems, July 2015, pp. 174–183.

[12] 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.

[13] S. K. Baruah et al., "Static-priority scheduling of multiframe tasks," in Proceedings of the 11th Euromicro Conference on Real-Time Systems, June 1999, pp. 38–45.

[14] A. K. Mok et al., "A multiframe model for real-time tasks," IEEE Transactions on Software Engineering, vol. 23, no. 10, pp. 635–645, Oct 1997.

[15] 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.

[16] J. Flodin et al., "Dynamic budgeting for settling DRAM contention of co-running hard and soft real-time tasks," in Proceedings of the 9th IEEEInternational Symposium on Industrial Embedded Systems, June 2014, pp. 151–159.

[17] M. Behnam et al., "Multi-core composability in the face of memory-bus contention," ACM SIGBED Review, vol. 10, no. 3, pp. 35–42, 2013.

[18] R. Inam et al., "The multi-resource server for predictable execution on multi-core platforms," in Proceedings of the 20th IEEE Real-Time and Embedded Technology and Applications Symposium, 2014.

[19] 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.

[20] H. Yun et al., "Memory bandwidth management for efficient performance isolation in multi-core platforms," IEEE Transactions on Computers, vol. 65, no. 2, pp. 562–576, Feb 2016.

[21] R. Pellizzoni et al., "Memory servers for multicore systems," in Proceedings of the 22nd IEEE Real-Time and Embedded Technology and Applications Symposium, 2016, pp. 97–108.

[22] R. Mancuso et al., "WCET Derivation under Single Core Equivalence with Explicit Memory Budget Assignment," in Proceedings of the 29th Euromicro Conference on Real-Time Systems, ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 76. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017, pp. 3:1–3:23.

[23] A. Agrawal et al., "Analysis of dynamic memory bandwidth regulation in multi-core real-time systems," in Proceedings of the 39rd IEEE Real-Time Systems Symposium, 2018, pp. 230–241.

[24] M. A. Awan et al., "Uneven memory regulation for scheduling IMA applications on multi-core platforms," Journal of Real–Time Systems, vol. 55, no. 2, pp. 248–292, Apr 2019.

[25] ——, "Worst-case stall analysis for multicore architectures with two memory controllers," in Proceedings of the 30th Euromicro Conference on Real-Time Systems, ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 106, 2018, pp. 2:1–2:22.

[26] ——, "Mixed-criticality scheduling with memory bandwidth regulation," in Proceedings of the 55th ACM/IEEE Conference on Design Automation Conference, March 2018.

[27] ——, "Mixed-criticality scheduling with dynamic memory bandwidth regulation," in Proceedings of the 24th IEEE Conference on Embedded and Real-Time Computing and Applications, 2018.

[28] ——, "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.

[29] B. Theelen et al., "Performance model checking scenario-aware dataflow," in International Conference on Formal Modeling and Analysis of Timed Systems, 2011.

[30] D. Le Gall, "Mpeg: A video compression standard for multimedia applications," Communications of the ACM, vol. 34, no. 4, pp. 46–58, Apr. 1991.

[31] S. Baruah et al., "Generalized multiframe tasks," Journal of Real–Time Systems, vol. 17, no. 1, pp. 5–22, Jul 1999.

[32] R. Pellizzoni et al., "A predictable execution model for cots-based embedded systems," in Proceedings of the 17th IEEE Real-Time and Embedded Technology and Applications Symposium, April 2011, pp. 269–279.

[33] G. Yao et al., "Memory-centric scheduling for multicore hard real-time systems," Journal of Real–Time Systems, vol. 48, no. 6, pp. 681–715, Nov 2012.

[34] J. Y.-T. Leung et al., "On the complexity of fixed-priority scheduling of periodic, real-time tasks," Performance Evaluation, vol. 2, no. 4, pp. 237 – 250, 1982.

[35] R. Wilhelm et al., "The worst-case execution-time problem—overview of methods and survey of tools," ACM Transactions on Embedded Computing Systems, vol. 7, no. 3, pp. 36:1–36:53, May 2008.

[36] S. Baruah, "The non-preemptive scheduling of periodic tasks upon multiprocessors," Journal of Real–Time Systems, vol. 32, no. 1-2, pp. 9–20, 2006.

[37] M. A. Awan et al., "Semi-partitioned mixed-criticality scheduling," in Proceedings of the 30th International Conference Architecture of Computing Systems, 2017, pp. 205–218.

[38] P. B. Sousa et al., "Unified overhead-aware schedulability analysis for slot-based task-splitting," Journal of Real–Time Systems, vol. 50, no. 5-6, pp. 680–735, 2014.

[39] R. I. Davis et al., "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.

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

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

[42] 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.