Considerations on Combining Vestal's Mixed-criticality Task Model and the Predictable Execution Model (PREM) for Real-time Systems

Ishfaq Hussain[1], Muhammad Ali Awan[1], Konstantinos Bletsas[1], Pedro F. Souto[2] and Eduardo Tovar[1]

[1] *CISTER Research Centre and ISEP/IPP, Porto, Portugal*
[2] *University of Porto, FEUP-Faculty of Engineering and CISTER Research Centre, Porto, Portugal*

Abstract
In the design of critical real-time embedded systems, predictability in timing behavior and in system resource usage is necessary. Vestal's mixed-criticality task model and the Predictable Execution Model (PREM) help achieve these objectives in different ways. Under Vestal's model, multiple worst-case execution time (WCET) estimates are considered for each task, with corresponding degree of confidence, and associated with a different criticality level. The schedulability analysis can derive the appropriate timing safety guarantees for each task without using more conservative estimates than needed, thereby avoiding overengineering. The adaptive variant of Vestal's model also allows for system modes, with some tasks idled at mode change and more conservative WCET estimates thereafter assumed for remaining tasks. Meanwhile, the 2-phase PREM model, via compiler support, first fetches from memory (into the cache) all the locations that a task will access, and only subsequently proceeds with computation. This removes a lot of the uncertainty in WCET estimation stemming from the cache state and memory access delays, leading to better predictability and tighter WCET estimates. Vestal's model and the PREM model, however, were independently conceived, and never combined. In this work, we explore different possibilities about how these two models could be combined. We focus on the semantics of multiple (static or probabilistic) per-task estimates of processor computation time and number of memory accesses, how these can be derived, the associated compiler and O/S support required, and the implications for timing analysis.

Keywords
Real time systems, mixed criticality model, Predictable execution model.

1. Introduction

The criticality of a computing task reflects the severity of the consequences of its failure; and a deadline miss is a form of failure. Higher-criticality tasks are developed according to stricter methodologies, require stronger safety guarantees and, traditionally, ran on

CERCIRAS WS01: 1st Workshop on Connecting Education and Research Communities for an Innovative Resource Aware Society

hussa@isep.ipp.pt (I. Hussain); awa@isep.ipp.pt (M. A. Awan); Koble@isep.ipp.pt (K. Bletsas); pfs@fe.up.pt (P. F. Souto); emt@isep.ipp.pt (E. Tovar)
http://www.cister.isep.ipp.pt/people/ishfaq_hussain// (I. Hussain)
0000-0002-4470-1744 (I. Hussain)

separate hardware from lower-criticality tasks. However, size, weight and cost concerns, currently motivate (e.g., in automotive or avionics systems) *mixed-criticality* systems, where tasks of different criticality can coexist on the same platform. Often, commercial-off-the-shelf (COTS) hardware is used, which offers good performance for its cost but is not very timing-predictable. Researchers try to deal with that in different ways, two of which are Vestal's mixed-criticality task model and the Predictable Execution Model (PREM). The present work highlights alternatives ways for combining those two models.

Vestal's model [1] uses multiple WCET estimates for the same task, with different degrees of confidence, associated with a corresponding criticality level. It avoids having to assume very pessimistic WCET estimates for all tasks, in schedulability analysis. In this paper, we consider its adaptive mode-based variant [2]. Under that model, every task has a criticality level and a set of WCET estimates – one for every criticality level up to its own (and progressively more conservative). The system is initially in the lowest mode, with all tasks present (and their WCETs for the lowest-criticality level assumed). If any task exceeds its WCET estimate for the system's current mode, then all tasks with criticality matching the mode's level are dispensed with, and the system switches to the next-highest mode and, for each remaining task, its next-highest WCET is assumed.

The **Predictable Execution Model (PREM)** is a way of dealing with intercore interference on accessing shared resources. PREM tasks are structured as sequences of scheduling intervals, which are of two types: *predictable* or *compatible*. A predictable interval is non-preemptible and it has a memory phase, followed by a computation phase. In the memory phase, all the data required during the computation phase is prefetched into the cache. This ensures that the computation phase is cache-miss-free and removes the need to analyse the cache state during WCET analysis, and the resulting pessimism.

In this work, we explore different ways in which PREM and the adaptive mode-based mixed-criticality model could be combined. For each alternative, we summarise the semantics, challenges and implications, in terms of analysis, compiler and O/S support.

2. Related Work

Vestal's paper [1] was the first work which considered multiple WCET estimates per task and co-scheduling of different-criticality tasks on the same platform. Building on that, Baruah et al. [2] proposed adaptive mixed-criticality scheduling (AMC). It involves system modes, with corresponding WCET estimates assumed for the tasks, and mode changes triggered by WCET estimate overruns. AMC was later extended to arbitrary deadlines [3] and multiframe tasks [4, 5]. For a survey of all related works, see [6].

Deployment of real-time systems (single- or mixed-criticality) on COTS hardware gives rise to predictability concerns, as different tasks access shared resources (i.e., caches, buses and memory). There exist different approaches in the literature that try to alleviate and/or upper-bound such interference, e.g., via regulation-based arbitration [7, 8, 9, 10], use of locks [11], partitioning of shared resources or through code refactoring [12, 13, 14]. The Predictable Execution Model (PREM) [12], in particular, requires task code structured as distinct memory or computation phases. Besides the original 2-phase model (memory-computation), there also exists the 3-phase model (memory-computation-memory) [15]. In the 2-phase model, both reads and writes are combined and performed at the beginning of a scheduling interval and the compiler ensures that that the computation phases will execute without cache misses. In the 3-phase model, reads and writes are further split into two phases. Our work considers the 2-phase mode, which was initially presented for single cores [12] and later extended for multicores [14].

3. System model

3.1. Task Model

Consider a set τ of N independent sporadic tasks, i.e., $\tau = \{\tau_0, \tau_1, \tau_2, \cdots, \tau_{N-1}\}$. For a task τ_i, κ_i, T_i and $D_i \leq T_i$ denote its criticality, minimum inter-arrival time and deadline, respectively. Each task is a sequence of non-preemptive predictable scheduling intervals, as in PREM [12]. For simplicity, we assume just two criticality levels, high (H) and low (L), i.e., $\kappa_i \in \{L, H\}$. Let N_i denote the number of scheduling intervals of a task τ_i and $E_i = \{E_{i,0}, E_{i,1}, \cdots, E_{i,N_i-1}\}$ the set of its scheduling intervals themselves. Each scheduling interval can be modelled by two parameters: the (worst-case) number of memory accesses performed in its memory phase and the (worst-case) length of its execution phase. However, in this work, we consider multiple estimates of each of those two parameters, with corresponding degrees of confidence associated with different criticality levels. For example, measurement-based techniques can be used to infer probably (but not provably) safe estimates whereas static analysis can be used for the highest degree of confidence. Therefore, we have L- and H-estimates, for the same quantity.

We assume that each memory access (cache miss) has a fixed latency. This allows us to simplify the notation, by using that latency as the unit of time. Then, an interval $E_{i,j}$ can be modeled as $\{\mu^L_{i,j}, \mu^H_{i,j}, C^{L|e}_{i,j}, C^{H|e}_{i,j}\}$; the first two scalars are estimates of memory accesses in the memory phase and the latter two are WCET estimates for the execution phase. (Different semantics for the pairs $\{\mu^L_{i,j}, \mu^H_{i,j}\}$ are explored in Section 4.)

3.2. Hardware Platform

Consider a multicore platform composed of K identical cores $\{P_0, P_1, \cdots, P_{K-1}\}$. The main memory is accessed through a single shared memory controller and interconnect, whose combined scheduling policy is round-robin, as in [7, 9]. The outer-level cache is partitioned or private to each core and same for the inner-level cache(s). Performance measuring counters (PMCs) are used to count the number of memory accesses.

4. Scheduling model

Analogously to existing works based on Vestal's adaptive mixed-criticality model, the idea is for the system to undergo mode switch whenever some L-estimate by some scheduling interval (e.g., $\mu^L_{i,j}$ or $C^{L|e}_{i,j}$) is exceeded. However, we have not yet defined how L- and

H-estimates relate to each other, nor what happens at mode switch. Different conceivable options for that exist. We next briefly examine four alternative models.

For better illustration and comparison, we do so via an example task set, consisting of four tasks (see Table 1). Two of these tasks are high-criticality (τ_2, τ_3) while the other two (τ_0, τ_1) of low-criticality. We consider a dual-core platform, with 3 tasks (τ_1 to τ_3) assigned to core P_1 and τ_0 assigned to P_0. On core P_1, task τ_3 has the lowest priority and τ_1 the highest one. The trace of execution for different PREM-MCS scheduling models is depicted in Figures 1 to 5. For easier illustration, we assume that memory accesses from different cores are served by the memory controller in a round-robin manner. (This is not part of the PREM-MCS model, which is agnostic w.r.t. the scheduling policy of the memory controller. We just had to assume one such policy, when drawing the schedules.)

Definition 4.1 (Transition scheduling interval). *If a mode switch is triggered by an overrunning task τ_m during the execution of scheduling interval $E_{i,j}$ of task τ_i, then $E_{i,j}$ is a transition scheduling interval.*

Definition 4.2 (Transition job). *A job with a transition scheduling interval.*

Definition 4.3 (Compute-interfered scheduling interval). *A transition interval that is in (or at the start of) its processor computation phase at the time of a mode switch.*

Definition 4.4 (Memory-interfered scheduling interval:). *A transition interval that is in (or at the start of) its memory phase at the time of a mode switch.*

4.1. PREM-MCS T-model

This variant uses, for a scheduling interval of an H-task, a single memory access estimate in both modes (i.e., $\mu^L_{i,j} = \mu^H_{i,j}$). This is conservatively derived, therefore it cannot be exceeded, triggering a mode switch. A mode switch can only be triggered by a processor execution overrun. The mode switch semantics are:

- Initially the system is in L-mode.
- A mode switch is triggered if any scheduling interval overruns its $C^{L|e}_{i,j}$.
- At the mode switch, all L-tasks are dropped and, for H-tasks, their $C^{H|e}_{i,j}$ estimates are henceforth assumed, for the transition scheduling interval, subsequent scheduling intervals of transition jobs and all future jobs.

As shown in Figure 1, as soon as the mode switch is triggered by $\tau_{0,1}$: $\tau_{2,1}$ (2nd job of task τ_2) executing on P_1 and all subsequently-executed scheduling intervals of all tasks, execute with more conservative H-mode estimates. This model is straightforward and requires no changes to the existing PREM compiler and its timing analysis could be based on that in [2] (for non-multiframe adaptive mixed-criticality tasks) with minimal changes. However, it passes on the opportunity to use different memory access estimates in each mode, instead always only using conservative estimates.

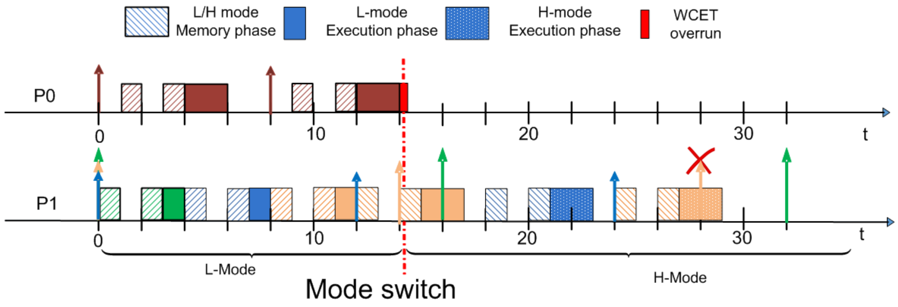

Figure 1: Schedule of example task set as per PREM-MCS T-model

Table 1
Task set for example

Task	Crit.	intervals (L-estimates)		intervals (H-estimates)	$D_i = T_i$	Priority	Core
		Case1	Case2-4				
τ_0	L	{2, 2}	{2, 2}	{-}	8	N/A	P_0
τ_1	L	{2, 1}	{2,1}	{-}	16	Highest	P_1
τ_2	H	{2,1}	{1,1}	{2, 2}	12	Middle	P_1
τ_3	H	{{2,1},{2,1}}	{{1,1},{1,1}}	{{2,2},{2,2}}	14	Lowest	P_1

4.2. PREM-MCS K-model

This variant uses less conservative estimates for both memory accesses and computation in the L-mode ($\mu^L_{i,j} \leq \mu^H_{i,j}$ and $C^{e,L}_{i,j} \leq C^{e,H}_{i,j}$). In either mode, as in PREM, the computation phase cannot incur cache misses; the memory phase is engineered to fetch all the locations needed into the cache, be they many or few. The mode switch semantics are:

- Initially the system is in L-mode, with corresponding estimates assumed for the tasks.
- A mode switch can be triggered by either memory access or computation overrun, by some task's scheduling interval.
- At mode switch, L-tasks are dropped and H-estimates are hitherto assumed for H-task scheduling intervals. For a transit interval $E_{i,j}$:
  - If $E^k_{i,j}$ is memory-interfered, then H-estimates are assumed for it and for subsequent intervals for this and all future jobs. This is depicted in Figure 2.
  - Analogously if $E^k_{i,j}$ is compute-interfered, with one difference: Since the memory phase of $E^k_{i,j}$ has already completed without violating its L-estimate, for the transition interval we consider $\mu^L_{i,j}$ memory accesses. This scenario is depicted in Figure 3.

This model can offer improved schedulability, compared to previous one (T). Probabilistic worst-case analysis techniques could be used to obtain the L-estimates. Static worst-case analysis techniques would only be used for H-estimates (safe but pessimistic). The existing PREM compiler [16] can be used without any changes to derive PREM-compliant scheduling intervals: Different jobs may issue different number of accesses

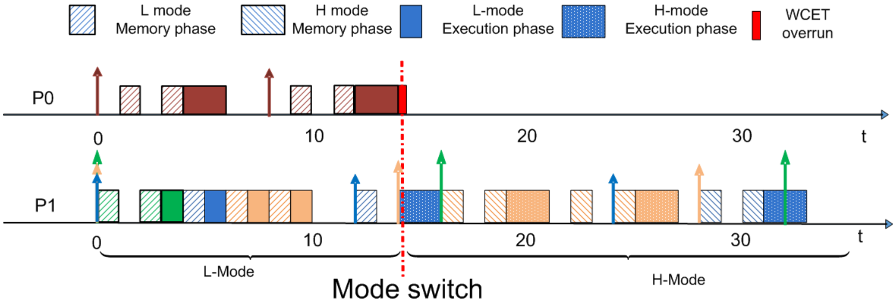

Figure 2: Case1: Schedule of example task set as per PREM-MCS K-model

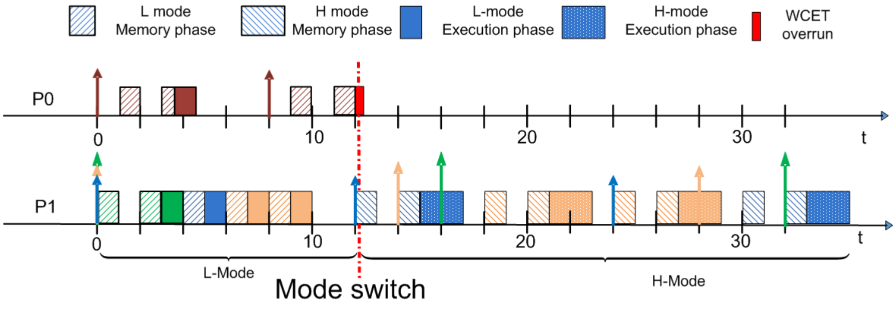

Figure 3: Case2: Schedule of example task set as per PREM-MCS K-model

(e.g., depending on run-time conditions), but would cover all locations needed to ensure no cache miss by the computation phase. The computation phase can still overrun its L-WCET due to e.g., a rare control flow. The schedulability analysis would be based on [2] with minor changes, as for the PREM-MCS T-model, just with the added consideration of whether the transition interval was memory- or compute-interfered (to avoid pessimism).

4.3. PREM-MCS P-model

The difference from the previous variant (K) is that the assumption/requirement of no cache misses during the computation phase is relaxed. In the L-mode (i.e., as long as the L-estimates are not overrun), this is still guaranteed by design. However, in the H-mode, under this variant, cache misses are possible. Figure 4 shows the execution pattern of the example task set as per the PREM-MCS P-model. The cache misses in the computation phase after the mode switch are represented as boxes with grid fill pattern.

This model violates the core assumption of PREM that computation phases are cache-miss-free. Contention therefore arises from memory accesses generated during the computation phase of H-tasks. If only few cache misses can occur, this can be managed via hardware servers, as in [17], to manage their effect. Software-based memory regulation can also be used [7] and/or even round-robin serving of accesses by different cores by the memory controller. This facilitates analytically upper-bounding the memory stalls.

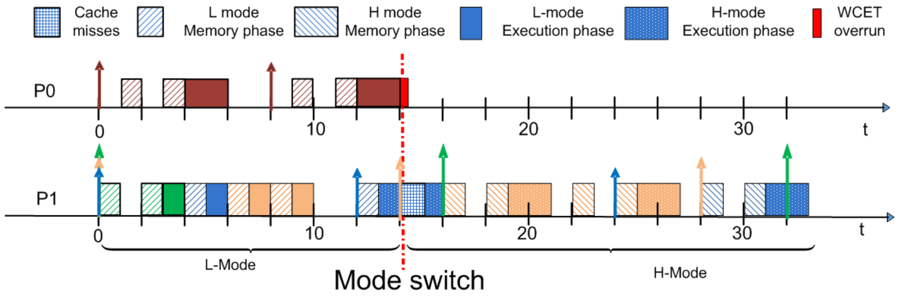

Figure 4: Schedule of example task set as per PREM-MCS P-model

4.4. PREM-MCS A-model

The distinguishing feature of this variant is that, for compute-interfered transition intervals, the computation phase is interrupted at the moment of mode changes, and a supplementary memory phase (with $\mu^H_{i,j} - \mu^L_{i,j}$ accesses) is executed; afterwards, the computation phase resumes from where it was interrupted. This is a significant departure from some of the assumptions in PREM.

This arrangement seeks to optimise for the common case: instead of prefetching memory locations that a task will rarely, if ever, need, only do this reactively, when indications arise that there is a chance of needing them. A mode switch triggered by another task can be such an indication. The case of the task itself triggering the mode switch, by exceeding its computation phase L-WCET merits some discussion.

In the general case, the computation phase of a scheduling interval $E_{i,j}$ overruning its $C^L_{i,j}$ might be purely down to control flow, and will not necessarily indicate that an access to a memory location other the $\mu^L_{i,j}$ locations prefetched by the corresponding memory phase is imminent. Similarly, if the computation phase of $E_{i,j}$ needs to access a memory location other than the $\mu^L_{i,j}$ locations prefetched by the corresponding memory phase, this would not necessarily be preceded by an exceedance of its $C^L_{i,j}$ estimate. In any case, the mode change semantics might be justified out of abundant caution.

Another possible way to justify the arrangement is by selecting the $\mu^L_{i,j}$ memory locations accessed during the memory phase and the computation phase L-mode WCET $C^L_{i,j}$ in conjunction such that (verifiably, by offline static analysis), under any control flow, no access to a memory location other than those $\mu^L_{i,j}$ occurs, unless (previously, in the same control flow) there is an overrun of the $C^L_{i,j}$ computation phase WCET estimate. Then, at mode switch, the supplementary memory phase fetches $\mu^H_{i,j} - \mu^L_{i,j}$ additional locations, in case they are needed; and this ensures the compuation phase will be cache-miss-free, in any case.

These semantics require significant changes both to the existing PREM compiler and scheduler. Under the PREM model, it is assumed that no system calls or interrupt service routines are served during predictable intervals. (These are only served under compatible intervals – the other type of scheduling intervals that we don't consider here.) However, for a transition interval, an interrupt-based method is required, to immediately initiate

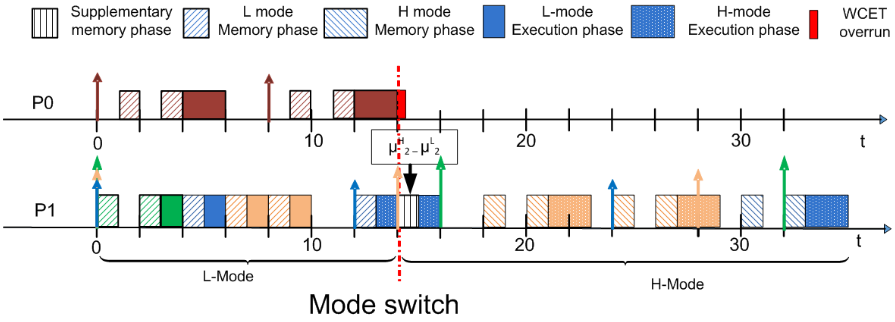

Figure 5: Schedule of example task set as per PREM-MCS A-model

the supplementary memory phase and to subsequently return control to the computation phase, at the point of interruption.

Some of the potential implications, in terms of complexity, on the static, offline timing WCET analysis have been mentioned above. From the schedulability analysis perspective, the interference and stall from the additional memory phase need to be quantified and incoropated into the schedulability analysis. The specifics will differ, depending, e.g., on whether there is memory access regulation [7], TDMA-based memory access [18] or DMA-enabled parallel memory accesses [19], but they can be complex. However, this model is quite flexible and it has a potential for improved schedulability.

In the example of Figure 5, an additional memory phase (with $\mu^H_{2,2} - \mu^L_{2,2}$ = 2 - 1 = 1 access) takes place during the computation phase of $\tau_{2,2}$.

5. Conclusion

Achieving both predictability and efficient resource utilisation for real-time systems deployed on COTS multicore platforms is challenging because of shared resources. In this work, we have proposed the combination of Predictable Execution model (PREM) and the Adaptive Mixed-Criticality model (AMC), as a promising way of dealing with such challenges. We outlined four possible ways of combining the semantics of those two models, especially regarding mode changes. Each of those has different advantages and implications, w.r.t. compiler support, run-time support and offline timing analysis.

This work is a first step in combining PREM and adaptive mixed-criticality scheduling. As a next step, weighing all the options, we will settle on the appropriate model semantics, and work on the corresponding schedulability analysis.

Acknowledgements

This work was partially supported by National Funds through FCT/MCTES (Portuguese Foundation for Science and Technology), within the CISTER Research Unit (UIDP/UIDB/04234/2020); 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 PREFECT (POCI-01-0145-FEDER-029119); by FCT through the European Social Fund (ESF) and the Regional Operational Programme (ROP) Norte 2020, under grant 2020.08045.BD. This work is partially supported by Connecting Education and Research Communities for an Innovative Resource Aware Society (CERCIRAS) COST Action CA19135 funded by European Cooperation in Science and Technology (COST) Association.

References

[1] S. Vestal, Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance, in: Proc. 28th RTSS, 2007, pp. 239–243.

[2] S. K. Baruah, A. Burns, R. I. Davis, Response-time analysis for mixed criticality systems, in: Proc. 32th RTSS, 2011, pp. 34–43.

[3] A. Burns, R. I. Davis, Response time analysis for mixed criticality systems with arbitrary deadlines, in: Proc. 5th Int. Workshop on Mixed Criticality Systems (WMC), 2017.

[4] I. Hussain, M. A. Awan, P. F. Souto, K. Bletsas, B. Akesson, E. Tovar, Response time analysis of multiframe mixed-criticality systems, in: Proc. 27th RTNS, 2019, pp. 8–18.

[5] I. Hussain, M. A. Awan, P. F. Souto, K. Bletsas, B. Akesson, E. Tovar, Response time analysis of multiframe mixed-criticality systems with arbitrary deadlines, Real-Time Systems 57 (2021).

[6] A. Burns, R. Davis, Mixed criticality systems – a review (12th ed.), Technical Report, Department of Computer Science, University of York, 2019.

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

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

[9] G. Yao, H. Yun, Z. P. Wu, R. Pellizzoni, M. Caccamo, L. Sha, Schedulability analysis for memory bandwidth regulated multicore real-time systems, IEEE Transactions on Computers 65 (2016) 601–614.

[10] M. A. Awan, K. Bletsas, P. F. Souto, B. Akesson, E. Tovar, Mixed-criticality scheduling with dynamic memory bandwidth regulation, in: Proc. 24th RTCSA, 2018, pp. 111–117.

[11] H. Yun, S. Gondi, S. Biswas, Protecting memory-performance critical sections in soft real-time applications, arXiv preprint arXiv:1502.02287 (2015).

[12] R. Pellizzoni, E. Betti, S. Bak, G. Yao, J. Criswell, M. Caccamo, R. Kegley, A predictable execution model for COTS-based embedded systems, in: Proc. 17th RTAS, 2011, pp. 269–279.

[13] A. Schranzhofer, J. Chen, L. Thiele, Timing analysis for tdma arbitration in resource sharing systems, in: Proc. 16th RTAS, 2010, pp. 215–224.

[14] A. Alhammad, R. Pellizzoni, Schedulability analysis of global memory-predictable scheduling, in: Proc. 14th EMSOFT, 2014, pp. 20:1–20:10.

[15] J. M. Rivas, J. Goossens, X. Poczekajlo, A. Paolillo, Implementation of memory centric scheduling for COTS multi-core real-time systems, in: Proc. 31st ECRTS, 2019, pp. 7:21–7:23.

[16] B. Forsberg, M. Solieri, M. Bertogna, L. Benini, A. Marongiu, The predictable execution model in practice: Compiling real applications for cots hardware, ACM Trans. on Embedded Computing Systems (TECS) 20 (2021) 1–25.

[17] R. Pellizzoni, M. Caccamo, Impact of peripheral-processor interference on WCET analysis of real-time embedded systems, IEEE Trans. on Computers 59 (2010) 400–415.

[18] G. Yao, R. Pellizzoni, S. Bak, E. Betti, M. Caccamo, Memory-centric scheduling for multicore hard real-time systems, Real-Time Systems 48 (2012) 681–715.

[19] A. Melani, M. Bertogna, R. I. Davis, V. Bonifaci, A. Marchetti-Spaccamela, G. Buttazzo, Exact response time analysis for fixed priority memory-processor co-scheduling, IEEE Transactions on Computers 66 (2017) 631–646.