Exact Response Time Analysis for Fixed Priority Memory-Processor Co-Scheduling

Alessandra Melani, Marko Bertogna, Robert Davis, Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, Giorgio Buttazzo

Exact Response Time Analysis for Fixed Priority Memory-Processor Co-scheduling

Alessandra Melani, Marko Bertogna, Robert I. Davis,
Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, and Giorgio Buttazzo

Abstract—Recent technological advances have led to an increasing gap between memory and processor performance, since memory bandwidth is progressing at a much slower pace than processor bandwidth. Pre-fetching techniques are traditionally used to bridge this gap and achieve high processor utilization while tolerating high memory latencies. Following this trend, new computational models have been proposed to split task execution in two consecutive phases: a memory phase in which the required instructions and data are pre-fetched to local memory (M-phase), and an execution phase in which the task is executed with no memory contention (C-phase). Decoupling memory and execution phases not only simplifies the timing analysis, but also allows a more efficient (and predictable) pipelining of memory and execution phases through proper co-scheduling algorithms. This paper takes a further step towards the design of smart co-scheduling algorithms for sporadic real-time tasks complying with the memory-computation (M/C) model, by proposing a theoretical framework aimed at tightly characterizing the schedulability improvement obtainable with the adopted M/C task model on single-core systems. In particular, a critical instant is identified for M/C tasks scheduled with fixed priority and an exact response time analysis with pseudo-polynomial complexity is provided. Then, we investigate the problem of priority assignment for M/C tasks, showing that a necessary condition to achieve optimality is to allow different priorities for the two phases. Our experiments show that the proposed techniques provide a significant schedulability improvement with respect to classic execution models, placing an important building block towards the design of more efficient partitioned multi-core systems.

Index Terms—Co-Scheduling, Response Time Analysis, Schedulability Analysis, Real-Time Systems.

1 Introduction

One of the major obstacles to improve the performance of current computing systems is the growing divergence between processor speed and memory speed, as the relatively slow access to memory and communication resources poses a serious limitation on the exploitation of processing power [24]. To feed the CPU with tasks ready to execute, memory and communication bottlenecks must be overcome by guaranteeing a proper provisioning of new data and instructions. Therefore, in addition to the processor, memory becomes another crucial resource of interest to be scheduled.

The real-time community has already identified the need for new scheduling algorithms and execution models allowing an efficient exploitation of the computing power and the derivation of tighter schedulability bounds both on single- and multi-core platforms. A major effort in this sense is represented by the PREM scheduling framework [44]. In this framework, tasks consist of different phases: a memory phase (M-phase) in which the task pre-fetches the required instructions and data from memory and/or I/O devices, and an execution phase (C-phase) in which the task executes without needing to access shared memory and communication devices. Depending on the model variants, tasks may have an additional memory phase to store the computed data back to memory, and/or they may be composed of multiple consecutive memory-execution frames.

This paper focuses on a simpler model, referred to as the *M/C task model*, where tasks include a memory phase followed by an execution phase. The advantage of pre-fetching execution models is that they decouple memory and execution phases, so that different phases of different tasks may overlap in time: during an M-phase, a task may only suffer interference from other shared memory accesses, whereas during a C-phase, a task is subject to interference only from the C-phases of other tasks. Execution models based on pre-fetching techniques are more amenable to timing analysis and have at least two fundamental advantages: (i) By grouping together all memory accesses, it is possible to better exploit burst read/write features (either DMA- or cache-based) for simultaneously loading/storing multiple memory locations in a back-to-back fashion, i.e., without needing to pay the full memory latency for each required instruction/data. This is particularly important for architectures featuring powerful DMA engines[1]; (ii) The coarser granularity of the memory and execution phases may be leveraged to devise smart co-scheduling algorithms that are able to reduce the overall response time by overlapping

- *A. Melani and G. Buttazzo are with the TeCIP Institute, Scuola Superiore Sant'Anna, Pisa, Italy. E-mail: {alessandra.melani, g.buttazzo}@sssup.it.*
- *M. Bertogna is with the University of Modena and Reggio-Emilia, Modena, Italy. E-mail: marko.bertogna@unimore.it.*
- *R.I. Davis is with the University of York, York, England, UK, and INRIA-Paris, Paris, France. E-mail: rob.davis@york.ac.uk.*
- *V. Bonifaci is with Istituto di Analisi dei Sistemi ed Informatica, CNR, Rome, Italy. E-mail: vincenzo.bonifaci@iasi.cnr.it.*
- *A. Marchetti-Spaccamela is with Sapienza University of Rome, Rome, Italy. E-mail: alberto@dis.uniroma1.it.*

1. See, e.g., Texas Instrument Keystone II (http://www.ti.com/product/66AK2H12), where a 20x speedup can be obtained exploiting the burst read features of the integrated DMA engines.

M/C phases. Since the two phases act on separate resources (bus and shared memory vs. processing elements), it is possible to hide memory latencies by properly orchestrating the access to processing and memory resources. In particular, this latter possibility will be thoroughly analyzed in this paper, identifying the possible schedulability improvement that can be obtained by leveraging the pipelined execution of memory and execution phases of different tasks. Previous related works adopting similar execution models showed a significant improvement with respect to classic (i.e., non pre-fetching) execution models, although they were based on heuristic approaches and pessimistic schedulability analyses [1], [2], [9], [44], [58], [60].

This work proposes an exact characterization of the schedulability of M/C task-sets on a single core, identifying critical instant scenarios that lead to worst-case response times under fixed priority preemptive scheduling and providing a necessary and sufficient schedulability test for the adopted task model. At a first sight, one may think that existing results for classic sporadic task systems (i.e., with tasks having just one phase) may be easily adapted to the M/C task model. Indeed, the M/C model trivially reduces to the classic sporadic task model when one of the two phases is negligible for all tasks. However, when this is not true, the simple fact that the memory and execution phases of different tasks may run in parallel invalidates most of the well-known results for classic preemptive task systems. In particular, it will be shown that (i) preemptive Earliest Deadline First (EDF) is not an optimal scheduling algorithm for M/C task systems (its exact speedup factor is 2); and (ii) the synchronous tasks activation with minimum inter-arrival separation among consecutive task instances does not represent a critical instant for M/C task systems, i.e., there may exist other release configurations that lead to a higher response time. The last observation is particularly detrimental to the schedulability analysis, because it prevents using the classic response time analysis [29] to characterize the schedulability of M/C task systems. The considerations stated above motivated us to investigate better algorithms and schedulability tests to fully exploit the potential of pre-fetching execution models for real-time applications, providing an important building block for the design of smart co-scheduling techniques. Interestingly, the theoretical results introduced in the paper can be also applied in the case of distributed transactions with two execution stages, for which only sufficient schedulability conditions with a tractable complexity have been derived so far [41], [42], [45], [52], [55].

Contributions of the paper. This paper establishes the theoretical background for addressing the schedulability analysis of M/C task systems. In particular, it provides the following contributions for a configuration with a single core and single memory channel: (i) EDF algorithm is proved to be not optimal for scheduling M/C task systems. In particular, a theoretical characterization in terms of speedup factor is provided; (ii) A critical instant is defined for M/C task systems scheduled with fixed-priority, proving that no other task release configuration may produce a larger response time; (iii) An exact response time analysis is derived for M/C systems scheduled with fixed priority, leading to a necessary and sufficient schedulability test; (iv) The paper proves that Deadline Monotonic (DM) is not an optimal priority assignment strategy and that Ausley's optimal priority assignment (OPA) algorithm cannot be used in conjunction with the exact test proposed in this paper; (v) Approximate schedulability tests are derived to enable the applicability of the OPA algorithm; (vi) The exact test proposed in this paper is generalized to consider different priorities for the M- and C-phase of each task, and possible priority assignment strategies are discussed for this more general case. Finally, the schedulability improvement obtainable in a single-core/single-memory setting is characterized by means of extensive evaluations using randomly generated workloads, identifying the systems that are more likely to benefit, and to which extent, from pre-fetching execution models, and the priority assignment strategies that perform better for the proposed M/C scheduling model.

This work is an extended version of the RTNS 2015 paper on *Memory-Processor Co-Scheduling in Fixed Priority Systems* [38]. The main extensions include Theorem 2, Section 5 and Section 6.3.

2 Related Work

The study of the interplay between memory access and CPU utilization is not novel in the real-time computing literature [47], [50]. More recently, there has been significant interest in the research community in addressing shared resource contention in multicore processors (see e.g. [61] and references therein). To better characterize the interference due to memory contention, new execution models have been proposed in the literature making use of pre-fetching techniques, which are widely adopted in the embedded and high-performance computing domain for different complementary reasons. Rosen et al. [48] investigated these techniques in the context of worst-case execution time computation and bus access optimization. Lu et al. [34], showed that pre-fetching techniques can improve the cache (or scratchpad) locality reducing the *average* execution times. They also allow hiding the memory latency by executing a pre-fetched task while pre-fetching the context of another one [36], and, most importantly, they allow predictably computing, bounding and mastering the memory interference due to concurrent accesses to shared memory by multiple tasks/cores, simplifying the computation of worst-case execution times. This latter issue has been addressed by Pellizzoni et al. [44] through the definition of the Predictable Execution Model (PREM). The work was focused on cache-based management of PREM-compatible tasks, showing how to enforce a predictable scheduling of memory and computing resources. It also showed how to automatically re-factor the task code at compile time, provided a set of restrictions is satisfied. Such restrictions are in line with those typically imposed by state-of-the-art tools for static timing analysis, and are also applicable to our M/C model, making it suitable only for certain types of software. An automatic tool for code re-factoring is presented by Mancuso et al. [35], making the adoption of the M/C model transparent to the programmer. Alternatively, M/C-compliant code may be written using programming models commonly adopted for heterogeneous computing systems

(e.g., OpenCL[2], OpenMP[3]) leveraging offloading directives that explicitly distinguish between shared and private data items, and that allow data/instruction pre-fetching [36]. An orthogonal approach to increase the cache locality and improve the predictability of memory accesses is using cache locking [46], [56] or partitioning [4] techniques. Scratchpad memory allocation has been considered by Deverge and Puaut [21] for single task scenarios. Whitham and Audsley [59] proposed the Carousel mechanism for dynamic scratchpad management in a multitasking system scheduled with Rate Monotonic (RM). However, both approaches stall the CPU while loading tasks to scratchpad, and, therefore, do not take advantage of the overlapping of M/C-phases.

To allow the simultaneous execution of memory and execution phases, a dynamic scratchpad management technique has been proposed by Wasly and Pellizzoni [57]. Bak et al. [9] used a simulation-based approach to compare different scheduling algorithms for PREM tasks. Yao et al. [60] proposed a TDMA-based scheduling algorithm for PREM tasks on a multicore platform. Wasly and Pellizzoni [58] presented a schedulability analysis for non-preemptable PREM tasks on single-core and partitioned multicore systems. The scheduling and schedulability problems for globally scheduled PREM tasks were also addressed [1], [2]. All these works are based on heuristic scheduling approaches and sufficient schedulability analyses.

To the best of our knowledge, no exact schedulability test is available for the considered task model. Some similarities may be found with the real-time distributed computing problem, where chains of tasks (also called pipelines or transactions) are executed on different processing nodes so that end-to-end deadlines are guaranteed [41], [42], [45], [52], [55]. The M/C phases considered in our paper may be seen as the precedence-constrained tasks composing a transaction in the distributed computing setting, each one executing on a different machine. For this problem, holistic response time analyses have been proposed for fixed-priority systems [42], [43], [55] and EDF-based systems [41], [45], [52]. Release jitters and offsets are introduced to account for the delayed release of precedence-constrained tasks of a transaction. These approaches typically imply a high complexity due to the difficulties in finding a critical instant scenario. For this reason, most works aim at providing only sufficient schedulability conditions, while existing exact analyses have an exponential complexity [42]. Alternative sufficient analyses for real-time distributed systems include the use of per-stage deadlines [40], real-time calculus [53], timed automata [31], compositional analysis [23], and delay composition algebra [26], [27]. The latter approach seems to provide the best trade-off between schedulability performance and complexity, and we will use it as a reference to evaluate the performance of our analysis.

Finally, the M/C scheduling can be considered a special case of the *flow shop problem* that has been studied by the combinatorial optimization community for its interest in production scheduling. The problem considers a two-stage processing facility and a collection of independent jobs, each comprising two tasks to be processed in order, one per stage. Differently from our setting, all jobs are initially available, and the objective is to minimize the makespan. If each stage consists of a single resource, the problem has a polynomial solution [28]; however, if at least one stage consists of two or more resources, then the problem becomes strongly NP-hard [25]. For this reason, several heuristics have been proposed [15]. Recently, a polynomial-time approximation scheme (PTAS) has also been proposed [49]. The optimality results derived by Johnson [28] have been extended by Melani et al. [39] to address the problem of selecting the optimal operating speed for the two resources.

2. Khronos Group, The OpenCL 1.1 Specifications, 2010: http://www.khronos.org/registry/cl/specs/opencl-1.1.pdf

3. OpenMP Application Program Interface v4, 2011: http://www.openmp.org/mp-documents/OpenMP3.1.pdf

3 System Model

We consider a set $\mathcal{T}$ of n periodic and sporadic real-time tasks $\tau_1, \ldots, \tau_n$, executing on a single-core platform. Each task τ_i is defined by a worst-case memory access time M_i (i.e., the length of its M-phase), a worst-case computation time C_i (i.e., the length of its C-phase), a relative deadline D_i and a period, or minimum interarrival time, T_i. We assume constrained deadlines, i.e., $D_i \leq T_i, \forall i$. Each task τ_i generates an infinite sequence of jobs, with the first job arriving at any time and successive job-arrivals separated by at least T_i time-units. We denote as r_i^j (resp. f_i^j) the release (resp. finishing) time of the j-th job of task τ_i, and as $d_i^j = r_i^j + D_i$ the absolute deadline of that job. Each job released by τ_i first pre-fetches data and instructions to the local memory, taking at most M_i time-units, and then it can start executing for at most C_i time-units on the processor. For any job of τ_i, we refer to as *M/C point*, the completion time of its M-phase, and denote it as ϕ_i^j. We say that the *M-phase* of a task is *ready* whenever a job of that task has been released but it did not yet complete its M-phase, i.e., before its M/C point. Similarly, we say that a *C-phase* of a task is *ready* whenever a job of that task completed its M-phase, but it did not yet complete its C-phase, i.e., between its M/C point and finishing time. In general, a *job* is *ready* if either its M- or C-phase is ready. No assumption can be made on the data locality of later jobs, but each new job will always have to pre-fetch new data from the memory.

We denote as $u_i^M = M_i/T_i$ (resp. $u_i^C = C_i/T_i$) the *memory* (resp. *computation*) utilization of task τ_i. U^M and U^C denote the total memory and computation utilization, i.e., $U^M = \sum_{\forall i} u_i^M$ and $U^C = \sum_{\forall i} u_i^C$. The overall utilization of the M/C task-set $\mathcal{T}$ is denoted as $U_{\mathcal{T}} = U^M + U^C$.

The *Worst Case Response Time* R_k of task τ_k is the worst-case relative finishing time among all its jobs, i.e., $R_k = \max_{j \text{ is a job of } \tau_k}(f_k^j - r_k^j)$. We denote with R_k^M the worst-case response time of the M-phase of task τ_k, i.e., from the job release until the completion of the M-phase; and with R_k^C the worst-case response time of the C-phase of τ_k, i.e., from the end of the memory phase until the completion of the C-phase. We also assume that each task has a best-case memory-access time and a best-case computation time equal to zero. We assume a preemptive fixed-priority scheduler, where each task has the same priority on the processor and for accessing the memory. Tasks are indexed in decreasing priority order, i.e., task τ_1 being the highest priority one. An M-phase (resp. C-phase) of a higher-priority task can preempt an M-phase (resp. C-phase) of a lower-priority one

at no additional cost. Moreover, there is no interference between M- and C-phases. This can be achieved if the M-phase is mastered by a DMA device, while the C-phase is executed by a processing element. Moreover, as shown in [57], the local memory may be partitioned so that simultaneously executing M/C phases never access the same partition. In this way, the M- and C-phases may overlap since they access different resources (DMA and shared memory on one side, processing element on the other side) and different local memory partitions.

We denote by $hp(k)$ the set of tasks with priority higher than τ_k, while $hep(k)$ indicates the set of tasks with priority higher than or equal to τ_k. To simplify the model, write-back phases following the M- and C-phases are not modeled. We remark that this assumption does not affect the validity of the model: with some exceptions, the number of (shared memory) store operations of typical real-time applications is significantly smaller than the number of read requests. Task instructions do not need to be written back. Data structures, images and input signals to process are also not written back. For applications like image detection, surveillance, and control systems, the output of the C-phase is typically restricted to a few actuation operations or detection signals. Moreover, if tasks are periodic and relative deadlines are implicit, write-back phases can be combined with the (read) M-phase of the subsequent job [44], [58].

We now discuss two simplifying assumptions within the considered system model. First, as with much of the real-time scheduling literature, preemption overheads are neglected. This is typically done to enable the derivation of exact schedulability tests, optimal scheduling algorithms and a clearer understanding of the scheduling problem. However, the impact of preemptions should be carefully analyzed before applying the theoretical results to a practical use case. In particular, when task footprints are comparable to the size of the local memory, a preempting task may evict a significant amount of memory blocks that are useful to a preempted task, leading to a considerable preemption delay [3]. This issue can either be solved by (i) assuming the local memory is sufficiently large to accommodate the footprint of all tasks, (ii) considering a memory penalty for each task resuming after a preemption, or (iii) integrating the presented schedulability analysis within the limited preemption framework [14]. This last approach seems promising to limit, or even avoid, the preemption overhead by encapsulating consecutive M/C phases within a non-preemption region, and is the subject of our ongoing work. The second simplifying assumption is to consider that the M-phase is fully preemptable. Most current hardware implementations do not allow a preemptive DMA behavior; however, preemptable DMA controllers have been advocated to reduce starvation and priority inversions effects [50], [51]. Further, some industrial solutions have enhanced DMA controllers to enable fine-grained control for easier preemption. For example, the Enhanced DMA engine TMS320C645x (EDMA3) by Texas Instruments features an Intermediate Transfer Chaining mechanism for breaking up large transfers into multiple smaller transfers[4]. This approximated preemption support is achieved with an efficient pipelined mechanism that is much more effective than classic cycle-stealing techniques, leading to a significantly smaller overhead. The theoretical analysis for the M/C model presented in this paper, assumes fully preemptive M- and C-phases. The analysis would need to be enhanced to account for limited preemption in the M-phase before it could be used to analyze systems built with today's DMA controllers; again, this is the subject of our ongoing work.

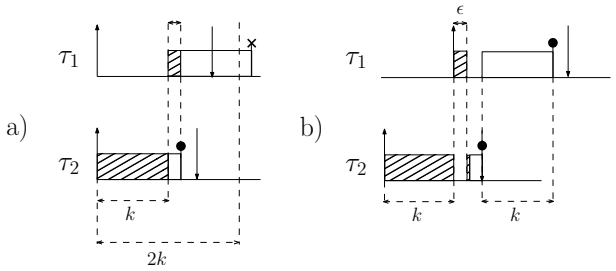

Figure 1: EDF is not optimal for collections of sporadic M/C tasks. In the figure, striped blocks are M-phases, a large dot marker indicates job completion, and a cross indicates job completion but deadline missed.

4 Schedulability Analysis

Before presenting our analysis for M/C task systems, we first show that some established results for classic single-phase systems are not valid for the M/C model. In particular, while EDF is an optimal algorithm for arbitrary collections of regular jobs [20], it is provably not optimal for sporadic M/C jobs, as shown in the following example.

Example 1. *Consider a system composed of two tasks: a task τ_1 with an M-phase $M_1 = \epsilon = 1$, a C-phase $C_1 = k = 10$, a deadline $D_1 = k + 3 = 13$, and a minimum inter-arrival time $T_1 = +\infty$; and a second task τ_2 with an M-phase $M_2 = k = 10$, a C-phase $C_2 = \epsilon = 1$, a deadline $D_2 = k + 2 = 12$, and a minimum inter-arrival time $T_2 = +\infty$. Figure 1a) shows the schedule with EDF: task τ_1 is given lower priority than τ_2, resulting in a response time of $2k + 1 = 21$ for τ_1, missing its deadline. Instead, if the M-phase of τ_1 is given a higher (static) priority than that of τ_2, and conversely the C-phase of τ_2 has higher (static) priority than that of τ_1, we show the critical instance leading to the worst-case response time (see Theorem 4) in Figure 1b). The longest possible response times for the M-phases are 1 for τ_1 and 11 for τ_2. Similarly, the longest possible response times for just the C-phases (from when they are released) are 1 for τ_2 and 11 for τ_1. Therefore, the overall response time for both tasks cannot exceed 12, so that both tasks meet their deadlines.*

We also derive results using a *theoretical method* of comparing the worst-case performance of different scheduling algorithms for the M/C scheduling problem based on a *resource augmentation* metric referred to as the *speedup factor* [30]. Specifically, we derive bounds on the factor by which the speed of the system (affecting both M- and C-phases) needs to be increased to ensure that any task set that is schedulable under some scheduling algorithm $\mathcal{A}$ is guaranteed to be schedulable under another algorithm $\mathcal{B}$. When $\mathcal{A}$ is an optimal algorithm[5], then this speedup factor

4. http://www.ti.com.cn/cn/lit/ug/spru966c/spru966c.pdf

5. The optimal algorithm may be hypothetical or clairvoyant.

provides a measure of the *sub-optimality* of algorithm $\mathcal{B}$. Speedup factors have previously been derived for the classic sporadic task model, comparing fixed priority and EDF scheduling under both preemptive and non-preemptive paradigms [18], [17], [19]. To the best of our knowledge, this is the first time that such measures have been derived for the M/C scheduling problem.

In Example 1, if EDF is used, τ_1 has a response time of $2k+1$, which shows that *EDF is sub-optimal by a speedup factor of at least* $\frac{2k+1}{k+3}$, *which tends to* 2 *as* k *becomes large*. In the following, we prove that 2 is the *exact* speedup factor required by EDF to successfully schedule any sporadic M/C task-set that is schedulable by an optimal algorithm. To this aim, we first recall as a preliminary notion the exact schedulability test for classic sporadic task systems (where each task τ_i has only one single execution phase of worst-case length E_i) scheduled with EDF [10].

Theorem 1 (from [10])**.** *A set of single-phase tasks is schedulable under EDF if and only if, in every time interval, the total processor demand requested by the task-set is no greater than the length of the interval [10]. In other words, a task-set is schedulable under EDF* if and only if*:* $\forall t \geq 0 \quad \sum_{i=1}^{n} DBF_i(t) \leq t$, *where* $DBF_i(t) = \max\left(0, 1 + \left\lfloor \frac{t-D_i}{T_i} \right\rfloor\right) E_i$.

The following theorem establishes the exact speedup factor of EDF for the M/C scheduling model.

Theorem 2. *The exact speedup factor required so that EDF is guaranteed to be able to schedule any sporadic M/C task-set that is schedulable according to an optimal algorithm is 2. Stated otherwise, the sub-optimality of EDF for the M/C scheduling problem is 2.*

Proof. Consider an arbitrary M/C task-set. Let $DBF_M(t)$ be the sum of the demand bound functions for all of the tasks for a time interval of length t considering only the M-phase of each task. Similarly, let $DBF_C(t)$ be the sum of the demand bound functions for all of the tasks for a time interval of length t considering only the C-phase of each task. Recall that the exact schedulability test for sporadic tasks with only a computation phase under EDF requires that $\forall t \geq 0 \; DBF_C(t) \leq t$. It is sufficient for schedulability under EDF that an M/C task-set is schedulable under a serializing version of EDF that does not allow any concurrent access to memory and computation. Hence, the task-set is schedulable under EDF on a system of speed S_1, where

$$S_1 = \max_{\forall t \geq 0} \left(\frac{DBF_M(t) + DBF_C(t)}{t} \right) \leq \max_{\forall t \geq 0} \left(\frac{DBF_M(t)}{t} \right) + \max_{\forall t \geq 0} \left(\frac{DBF_C(t)}{t} \right).$$

Let S_2 denote the right-hand side of the above inequality. Next, we consider an optimal algorithm for the M/C scheduling problem. Necessary conditions for schedulability under an optimal algorithm are that (i) the task set is schedulable considering only the M-phase of each task, ignoring all the C-phases, and (ii) the task set is schedulable considering only the C-phase of each task, ignoring all the M-phases. Since EDF is an optimal single machine scheduling algorithm for sporadic task-sets, it follows that it is necessary for schedulability under an optimal algorithm that the system speed is such that EDF can schedule the M-phases alone, and also that EDF can schedule the C-phases alone. Hence S_3 is a necessary system speed required by an optimal algorithm for the M/C scheduling problem:

$$S_3 = \max_{\forall t \geq 0} \left(\frac{DBF_M(t)}{t}, \frac{DBF_C(t)}{t} \right). \tag{1}$$

Since any given task-set is guaranteed to be schedulable on a system of speed S_2 using EDF, and cannot be schedulable with any (optimal) algorithm on a system of speed lower than S_3, it follows that an upper bound on the speedup factor required by EDF is given by $S = S_2/S_3$. Let $L_M = \max_{\forall t \geq 0} \left(\frac{DBF_M(t)}{t} \right)$ and $L_C = \max_{\forall t \geq 0} \left(\frac{DBF_C(t)}{t} \right)$. It follows that

$$S = \frac{L_M + L_C}{\max(L_M, L_C)}. \tag{2}$$

It is easy to see that the speedup factor S is maximized when L_M and L_C are equal, in which case the upper bound speedup factor is $S = 2$. As we already showed in Example 1 that $S = 2$ is a lower bound, the exact speedup factor required by EDF for the M/C problem is 2. □

Another result that is no longer valid for the M/C model concerns the concept of *critical instant*. A critical instant is a particular release configuration that leads to the largest possible response time under fixed priority scheduling for a given task. For regular (independent) task instances, a critical instant is given by the synchronous release of all tasks, with jobs re-released as soon as possible, i.e., with consecutive task instances separated by their minimum inter-arrival time [33]. The following example shows that this is no longer true for M/C task-sets.

Example 2. *Consider a system composed of two periodic or sporadic tasks: a task* τ_1 *with an M-phase* $M_1 = 0$*, a C-phase* $C_1 = 2$*, and a deadline* $D_1 = 2$*; and a task* τ_2 *with an M-phase* $M_2 = 2$*, a C-phase* $C_2 = 1$*, and a deadline* $D_2 = 3$*. Both tasks have an arbitrarily large period. When both tasks are released synchronously, any work-conserving scheduler*[6] *will immediately start executing the C-phase of* τ_1*, completing right before its deadline; meanwhile,* τ_2 *executes its M-phase, leaving sufficient slack to complete its computing part before its deadline. When instead the release of* τ_1 *is postponed by one time unit, at least one of the tasks will miss its deadline, independently of the adopted scheduling algorithm.*

The above example can be identically used to show that the synchronous periodic release scenario is not a critical instant for sporadic M/C task systems scheduled with fixed-priority. Since the schedulability analysis of classic sporadic task systems scheduled with fixed-priority hinges on the synchronous periodic critical instant, this prevents the adoption of existing results for the considered setting. In particular, the response time analysis for sporadic task sets with constrained deadlines given in the following theorem is not applicable to the M/C model.

Theorem 3 (from [29])**.** *For classic sporadic task systems with constrained deadlines scheduled with fixed-priority, the worst-case*

6. In the M/C model, a scheduler is work-conserving if it never idles a resource (core or memory) whenever there is a ready phase (C- or M-phase, respectively).

response time of a task τ_k can be computed by finding R_k from the following iterative relation, starting with $R_k = E_k$:

$$R_k \leftarrow \sum_{j \in hep(k)} \left\lceil \frac{R_k}{T_j} \right\rceil E_j. \quad (3)$$

The above theorem may be applied to the considered M/C task model as a sufficient test, i.e., to compute an upper-bound on the worst-case response time of an M/C task, using the sum of the memory and computation phases as the worst-case execution time: $E_j = M_j + C_j, \forall \tau_j$. However, this approach is pessimistic since it does not take advantage of the possible overlapping of memory and execution phases in M/C task systems. An alternative approach is using the classic response time analysis to find the worst-case response time of the M-phase (i.e., using $E_j = M_j, \forall \tau_j$), and use this value as a release offset for the corresponding C-phase. This second approach has been adopted in the real-time literature for distributed task systems [41], [42], [45], [52], [55], providing offset-based response time analyses leading to tighter (still, only sufficient) schedulability tests.

In the remainder of this section, we extend the state-of-the-art by providing a necessary and sufficient schedulability test for M/C sporadic task systems with constrained deadlines scheduled with fixed-priority. For this purpose, we identify a new critical instant that leads to the worst-case response time of fixed-priority M/C tasks, and derive an exact response time analysis for the considered setting.

4.1 Critical Instant

The problem in deriving a critical instant for M/C task systems is due to the precedence constraint between the M- and the C-phases. When trying to maximize the overall response time R_k of a task τ_k (see Equation (4)), there may be configurations that maximize the response time R_k^M of the M-phase, but that do not maximize the response time R_k^C of the corresponding C-phase, and vice-versa. Also, the maximum overall response time may theoretically correspond to a configuration that does not maximize either the memory or the computation response time, making it significantly more complex to identify a critical instant scenario. Conversely, if one were able to find a configuration that maximizes both the memory response time and the computation response time, this would automatically give a valid critical instant. Such a configuration would lead to a response time of R_k^M for the M-phase, and of R_k^C for the C-phase. Since the two phases may not overlap, the overall response time of a task τ_k may be easily found as:

$$R_k = R_k^M + R_k^C. \quad (4)$$

Below, we prove that such a configuration indeed exists. To do that, we first introduce a nomenclature to distinguish the different kinds of interfering contributions that each task may experience. We will denote as J_k the job of task τ_k under analysis, dropping the job index to simplify the notation (i.e., the release time of J_k will be denoted as r_k, and its M/C point as ϕ_k), and as τ_i a generic (higher priority) task whose jobs interfere with τ_k. Jobs interfering with J_k may be divided into memory-interfering, processor-interfering and dual-interfering, as follows.

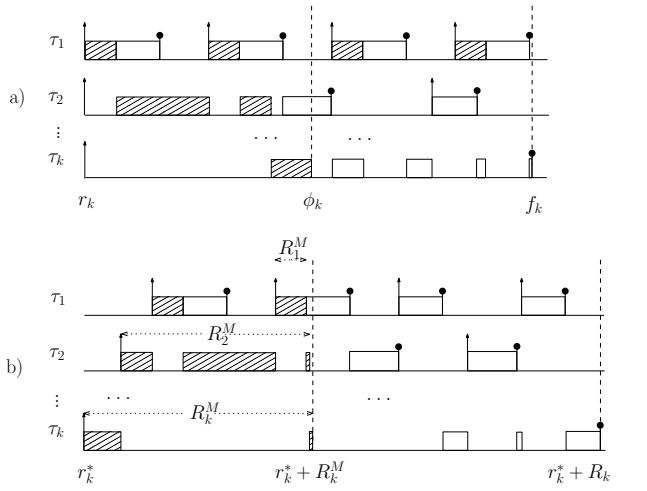

Figure 2: Synchronous release (a) and critical instant configuration (b).

Definition 1. *A job of task τ_i is said to be* M-interfering *(resp.* C-interfering*) with J_k if the M-phase (resp. C-phase) of J_k is ready but it cannot execute while the M-phase (resp. C-phase) of the job of τ_i is executing.*

Definition 2. *A job of task τ_i is said to be* dual-interfering *with J_k if it is both M- and C-interfering with J_k.*

Lemma 1. *Each higher priority task τ_i, $1 \leq i < k$, has at most one dual-interfering job with J_k.*

Proof. The M-phase (resp. C-phase) of job J_k will be interfered only by M-phases (resp. C-phases) of higher priority jobs. Since there is only one M-phase and one C-phase in the considered model, J_k cannot be C-interfered before ϕ_k, and it cannot be M-interfered after ϕ_k. Also, due to the constrained deadline model, each higher priority task τ_i has at most one job ready at time ϕ_k. □

To clarify the nomenclature, consider the example in Figure 2(a), where the synchronous release pattern is assumed for all tasks $\tau_1, \ldots, \tau_k$ in a fixed-priority schedule. With respect to the considered job J_k, the first two jobs of the highest priority task τ_1 are M-interfering jobs, while the latter two are C-interfering jobs. The first job of τ_2 is instead a dual-interfering job, as it interferes with J_k both in memory and CPU. The above example will be used in the following to derive a critical instant configuration for M/C task systems. In particular, we will show that by shifting right all interfering tasks such that they all have a dual-interfering job with M/C point aligned with that of the interfered job J_k, then the response time of J_k is maximized. This result is formally proved in the following theorem.

Theorem 4 (Critical Instant)**.** *The maximum response time of a job J_k of a task τ_k in a fixed-priority M/C system is found when all higher priority tasks $\tau_i, i \in hp(k)$ have:*

1) *a dual-interfering job completing its M-phase an infinitely small amount of time earlier than the M/C point of J_k;*
2) *all jobs released periodically;*
3) *a null M-phase for all (C-interfering) jobs released after the M/C point of J_k.*

Proof. We will prove that under the considered configuration, summarized in Figure 2(b), the response times of both the M- and C-phases of the considered job J_k are individually maximized. We first prove that the response time of the M-phase of J_k is maximized under the considered scenario.

Lemma 2. *The response time of the M-phase of J_k is maximized under the critical instant of Theorem 4.*

Proof. Since there is only one M-phase per job, and it is the first phase to execute (i.e., it does not have any precedence constraint), the problem is similar to the response time analysis of classic (single-phase) systems. By analogy with classic sporadic task systems, the synchronous periodic release pattern (as in Figure 2(a)) maximizes the response time R_k^M of the memory phase of J_k [33]. Under such a configuration, let J_i^* be the last M-interfering job of each higher priority task τ_i. It will be either an M-interfering job (e.g., τ_1), or a dual-interfering job (e.g., τ_2). In either case, the M/C point of J_i^* cannot be later than the M/C point of the interfered job J_k, i.e., $\phi_i^* < \phi_k$. Starting from τ_{k-1} and proceeding in reverse priority order, we now shift right each higher priority task τ_i until the M-phase of its job J_i^* completes an infinitesimal amount of time earlier than ϕ_k (as in Figure 2(b)), the response time of J_k does not change, because, by construction, none of the M-interfering instances exits the window $[r_k, r_k + R_k^M]$ from the right. Note also that no other M-phase may enter the window from the left since R_k^M is already the maximum possible. □

Note that the above lemma can be identically used to show that also the response time of the M-phase of each dual-interfering job J_i^* is maximized under the considered scenario. We now prove that also the response time of the C-phase of J_k is maximized under the considered scenario.

Lemma 3. *The response time of the C-phase of J_k is maximized under the critical instant of Theorem 4.*

Proof. In the critical instant configuration, the M-phase of each dual-interfering job J_i^* has a maximal response time equal to R_i^M. This means that the C-phase of each such job becomes ready at the latest possible instant, i.e., $\phi_i^* = r_i^* + R_i^M$. Moreover, according to the definition of critical instant of Theorem 4, later instances are released as soon as possible, with no M-phase. This means that the largest possible C-phase workload from τ_i is imposed to lower priority C-phases that become ready at ϕ_i^*. Note that, in a single-core system scheduled with fixed-priority, all higher priority C-phase workload will C-interfere with a lower priority C-phase, according to Definition 1. Since the M/C points of all jobs J_i^* are aligned with ϕ_k, J_k will experience the maximum possible C-interference by each higher priority task τ_i. This leads to the worst-case response time R_k^C for the C-phase of J_k. □

Having proved that the response times of both the M- and C-phases of J_k are individually maximized, the theorem follows. □

Note that assuming that the interfering jobs released after ϕ_k may have a null M-phase is not an over-constraining assumption, but it is needed to comply with the notion of "sustainability", as defined by Burns and Baruah [13]. A scheduling algorithm or a schedulability test is defined to be sustainable if any task system determined to be schedulable remains so when it behaves "better" than its worst-case specification; for example, when some of the tasks executes for less than its worst-case execution time. Therefore, the schedulability of the M/C task system has to be ensured also when the M-phase of some of the tasks takes less than M_i, or it is completely skipped, as in the critical instant configuration of Theorem 4.

4.2 Exact Response Time Analysis

Based on the identified critical instant, the following theorem allows computing the exact worst-case response time of each M/C task τ_k.

Theorem 5. *In a fixed-priority system, the worst-case response time of each constrained deadline M/C task τ_k can be computed as $R_k = R_k^M + R_k^C$, where R_k^M is first found from the following iterative relation, starting with $R_k^M = M_k$:*

$$R_k^M \leftarrow \sum_{i \in hep(k)} \left\lceil \frac{R_k^M}{T_i} \right\rceil M_i, \tag{5}$$

and then it is used in the following iterative relation to find R_k^C, starting with $R_k^C = C_k$:

$$R_k^C \leftarrow C_k + \sum_{i \in hp(k)} \left\lceil \frac{R_k^C + R_i^M}{T_i} \right\rceil C_i. \tag{6}$$

Proof. Consider the critical instant configuration of Theorem 4. Since both the M- and C-phase response times of J_k are individually maximized under the considered configuration, the worst-case response time R_k of τ_k can be computed using Equation (4). To compute the worst-case response time R_k^M of the M-phase, we note that it is exactly the same obtained under the synchronous release pattern (see the proof of Lemma 2). Therefore, by analogy with the classic sporadic task model, it can be simply found by the fixed-point iteration of Equation (5).

The worst-case response time R_k^C of the C-phase can instead be found by analogy with the response time analysis for classic sporadic tasks with release jitter (i.e., the maximum deviation of successive task releases from its period) [7], where the worst-case response time of the M-phase behaves as a release jitter for the C-phase. Consider the C-interfering workload produced by the higher priority tasks when (i) the C-phase of the first instance of each task becomes ready with an offset R_i^M, (ii) the M/C point of all the first instances are aligned, and (iii) later instances are released as soon as possible, with no M-phase (see Theorem 4). Under such a configuration, the C-phase response time of J_k can be found by considering the C-interfering contributions from each higher priority task. That is, for each $\tau_i, i < k$, (i) the dual-interfering job J_i^*, and (ii) the remaining interfering instances computed as $\left\lceil \frac{R_k^C - (T_i - R_i^M)}{T_i} \right\rceil$, each contributing for C_i. By adding the worst-case execution time C_k of the task under analysis, we obtain:

$$R_k^C \leftarrow C_k + \sum_{i \in hp(k)} \left(1 + \left\lceil \frac{R_k^C - T_i + R_i^M}{T_i} \right\rceil \right) C_i.$$

By simplifying the terms, Equation (6) follows. □

A simple (necessary and sufficient) schedulability test can be found by checking whether the worst-case response time R_k computed with Theorem 5 is $\leq D_k$, for each task τ_k in the system. Whenever the response time of a task exceeds its deadline, the tests stops, concluding that the task set is not feasible with fixed priority.

Note that Theorem 5 provides a stronger result than the existing analysis for distributed scheduling settings [42], [43], [55], which is able to provide only sufficient schedulability conditions for transactions with dynamic offsets in a tractable time. Instead, the critical instant configuration identified in Section 4.1 is shown to *jointly* maximize the worst-case response times of both the M-phase and the C-phase, providing a necessary and sufficient response time analysis for checking the schedulability of M/C task systems with pseudo-polynomial complexity.[7]

5 The Priority Assignment Problem

In this section, we study the problem of priority assignment for the M/C scheduling model. First (Section 5.1), we show that, as opposed to the classical uniprocessor case [32], Deadline Monotonic (DM) is not an optimal priority ordering. In addition (Section 5.2), we show that the Audsley's Optimal Priority Assignment algorithm (OPA) [6], [8] cannot be used in conjunction with the exact test proposed in Section 4.2. We then propose a sufficient test that is compatible with the OPA algorithm (Section 5.3) and evaluate its performance against the DM priority assignment, which can be applied in combination with our exact test. Finally (Section 5.4), we elaborate on the possibility of assigning different priorities to the M- and C-phase of each task.

5.1 Deadline Monotonic (DM) is not optimal

The following counterexample shows that DM is not an optimal priority assignment for the M/C scheduling model.

Example 3. *Consider a system composed of three tasks with the following parameters:* $\tau_1 = (M_1, C_1, D_1 = T_1) = (9, 1, 20)$, $\tau_2 = (1, 9, 24)$, $\tau_3 = (5, 5, 35)$. *Figure 3a) shows the schedule with DM in the critical instant scenario. In this configuration,* $R_1^M = 9$, $R_1^C = 1$, $R_1 = 10 \leq 20$ *for task* τ_1, *and* $R_2^M = 10$, $R_2^C = 10$, $R_2 = 20 \leq 24$ *for task* τ_2. *However,* τ_3 *misses its deadline, since* $R_3^M = 15$, $R_3^C = 25$, $R_3 = 40 > 35$. *Conversely, all tasks become schedulable in priority order* $[\tau_2, \tau_1, \tau_3]$, *as shown in Figure 3b). In particular, we have:* $R_2^M = 1$, $R_2^C = 9$, $R_2 = 10 \leq 24$ *for task* τ_2; $R_1^M = 10$, $R_1^C = 10$, $R_1 = 20 \leq 20$ *for task* τ_2, *and* $R_3^M = 15$, $R_3^C = 16$, $R_3 = 31 \leq 35$ *for task* τ_3.

Intuitively, DM fails when swapping the priority order of two higher priority tasks changes their offset at the critical instant, such that the interference on a lower-priority task is increased. In the above example, τ_3 is not schedulable with DM, as the execution of its C-phase suffers interference from two instances of each higher-priority task, leading to a deadline miss for τ_3. By swapping the priorities of τ_1 and τ_2, as in Figure 3b), their offsets in the critical instant configuration decrease by one instance the interference of τ_2 on the C-phase of τ_3, which now becomes schedulable. This result suggests how to reason about different priority assignment algorithms. In the context of uniprocessor fixed-priority scheduling, the Optimal Priority Assignment (OPA) algorithm [6], [8] solved the problem of optimally assigning priorities for asynchronous task-sets (i.e., which do not necessarily share a common release time) [5], and for tasks with arbitrary deadlines (i.e., which may be greater than their periods) [54]. In the next section, we reason about the applicability of OPA for M/C task-sets.

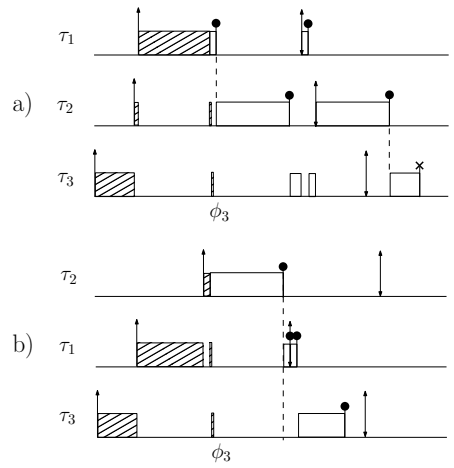

Figure 3: Counterexample on the sub-optimality of DM.

5.2 Applicability of OPA algorithm

In [16], the authors showed that three necessary and sufficient conditions need to be fulfilled to enable the applicability of OPA in combination with a given schedulability test. If these conditions are verified, OPA provides optimal priority assignment with respect to the considered test.

Theorem 6. *The exact RTA-MC test (Theorem 5) is OPA-incompatible.*

Proof. It suffices to show that Condition 1 in [16] does not hold for the RTA-MC test. The response time of the C-phase of any task τ_k (Equation (6)) depends on the response times of the M-phase of higher-priority tasks, which in turn depend on the relative priority ordering of the tasks. For instance, the task set of Example 3 is deemed schedulable by RTA-MC with priority order $[\tau_2, \tau_1, \tau_3]$; however, it becomes unschedulable by switching the order of τ_1 and τ_2, because τ_2 at priority level 2 increases the interference on τ_3, which is then deemed unschedulable, thus Condition 1 from [16] does not hold and so the test is not OPA-compatible. □

5.3 OPA-compatible schedulability tests

The negative result above means that it is interesting to investigate the performance of the OPA algorithm in combination with weaker (i.e., sufficient) schedulability tests. In this section, we derive a sufficient schedulability test that can be used in combination with the OPA algorithm. In this way, we can check whether in our case an optimal priority

7. While a deep analysis of partitioned multi-core systems is left as a future work, note that the test of Theorem 5 can also be adopted for partitioned multi-core systems, where each task is statically assigned to a given core, while all cores share the same main memory. In this case, the sum of Equation (6) has to be limited to higher priority tasks assigned to the same core of the considered task τ_k, while Equation (5) is still extended to all higher priority tasks.

assignment is able to make up for a weaker schedulability test, for example by making comparisons with an exact test assuming DM priority order.

Theorem 7. *For each task τ_k, an upper-bound on its response time can be computed by first finding R_k^M from the following iterative relation, starting with $R_k^M = M_k$:*

$$R_k^M \leftarrow \sum_{i \in hep(k)} \left\lceil \frac{R_k^M}{T_i} \right\rceil M_i, \tag{7}$$

and then using it in the following iterative relation:

$$R_k^C \leftarrow C_k + \sum_{i \in hp(k)} \left\lceil \frac{R_k^C + \min(R_k^M - M_k, D_i - C_i)}{T_i} \right\rceil C_i. \tag{8}$$

Proof. It is enough to show that $(R_k^M - M_k)$ and $(D_i - C_i)$ are both valid upper-bounds on R_i^M. Indeed, by Equation (7) we get:

$$R_k^M - M_k = \sum_{j \in hp(k)} \left\lceil \frac{R_k^M}{T_j} \right\rceil M_j. \tag{9}$$

As the right-hand side is non-negative and $i < k$, it follows that:

$$R_k^M - M_k \geq \sum_{j \in hep(i)} \left\lceil \frac{R_i^M}{T_j} \right\rceil M_j = R_i^M. \tag{10}$$

$D_i - C_i$ is a valid upper-bound on R_i^M as well because by Equation (4) and the schedulability of τ_i it follows that $R_i = R_i^M + R_i^C \leq D_i$, and since $C_i \leq R_i^C$, we conclude that $R_i^M \leq D_i - C_i$. □

In Section 6.3, we will experimentally evaluate the performance of the proposed approaches to priority assignment, along with the exact (Theorem 5) and sufficient (Theorem 7) schedulability tests.

5.4 Different priorities for M- and C-phases

We now add a new degree of freedom to the priority assignment problem by allowing different priorities for the M- and C-phase of each task (so each task has two priorities which are used for all of its jobs). In this new setting, we need to consider different sets of higher-priority tasks for the two phases. In particular, we define $hp^M(k)$ (resp., $hp^C(k)$) as the set of tasks whose M-phase (resp., C-phase) has priority higher than the M-phase (resp., C-phase) of τ_k. Sets $hep^M(k)$ and $hep^C(k)$ are re-defined accordingly. As a preliminary result, we show that the critical instant scenario and, consequently, the exact schedulability analysis described in Section 4 still hold when different static priorities are assigned to the two phases. Theorem 8 generalizes Theorem 4 to encompass this situation.

Theorem 8 (Critical Instant)**.** *The maximum response time of a job J_k of a task τ_k in a fixed-priority M/C system is found when all higher priority tasks $\tau_i, i \in hp^M(k) \cup hp^C(k)$ have:*

1) *a job[8] completing its M-phase an infinitely small amount of time earlier than the M/C point of J_k;*
2) *all jobs released periodically;*
3) *a null M-phase for all (C-interfering) jobs released after the M/C point of J_k.*

8. Contrary to what stated in Theorem 4, this job cannot be referred to as *dual-interfering*. This is only true whenever the interfering task belongs to $hp^M(k) \cap hp^C(k)$.

Proof. For any task τ_i in $hp^M(k) \cup hp^C(k)$, it may be that either (i) $\tau_i \in hp^M(k) \cap hp^C(k)$; or (ii) $\tau_i \in hp^M(k)$ and $\tau_i \notin hp^C(k)$; or (iii) $\tau_i \notin hp^M(k)$ and $\tau_i \in hp^C(k)$.

In case (i), the result of Theorem 4 straightforwardly applies. In case (ii), all jobs of τ_i are M-interfering, but not C-interfering. The configuration of Theorem 8 maximizes the M-interfering contribution on J_k by the result of Lemma 2. Finally, in case (iii), all jobs of τ_i are C-interfering, but not M-interfering. However, if we let a job of τ_i complete its M-phase an infinitely small amount of time earlier than ϕ_k (despite not being M-interfering with J_k), this will cause the C-phase of such a job to be ready at the latest possible instant. As subsequent instances are released as early as possible, the result of Lemma 3 also holds under the critical instant of Theorem 8, concluding the proof. □

Based on the identified critical instant scenario, the following theorem trivially extends the exact schedulability analysis of Theorem 5 to the case of different priorities for the two phases of each M/C task τ_k.

Theorem 9. *In a fixed-priority system, the worst-case response time of each constrained deadline M/C task τ_k can be computed as $R_k = R_k^M + R_k^C$, where R_k^M is first found from the following iterative relation, starting with $R_k^M = M_k$:*

$$R_k^M \leftarrow \sum_{i \in hep^M(k)} \left\lceil \frac{R_k^M}{T_i} \right\rceil M_i, \tag{11}$$

and then it is used in the following iterative relation to find R_k^C, starting with $R_k^C = C_k$:

$$R_k^C \leftarrow C_k + \sum_{i \in hp^C(k)} \left\lceil \frac{R_k^C + R_i^M}{T_i} \right\rceil C_i. \tag{12}$$

We now show that assigning the same priority to the two phases is not optimal, i.e., there are task-sets that are unschedulable when the same priority is assigned to both phases of each task, that become schedulable when different priorities are assigned to the execution phases of some task.

Example 4. *Consider the task-set described in Example 3. We have shown previously that the task-set is schedulable in priority order $[\tau_2, \tau_1, \tau_3]$. Then, if we let $D_1 = T_1 = 19$ (instead of 20), τ_1 fails to meet its deadline and renders the task-set unschedulable. However, if we assign the highest priority to the C-phase of τ_1, with the C-phase of task τ_2 at the second highest priority (leaving the M-phase of task τ_2 still at the highest priority) then the task-set becomes schedulable again. It can be easily checked that in this case, under the critical instant scenario described by Theorem 8, $R_2^M = 1$, $R_2^C = 10$, $R_2 = 11 \leq 24$ for task τ_2; $R_1^M = 10$, $R_1^C = 1$, $R_1 = 11 \leq 19$ for task τ_1, and $R_3^M = 15$, $R_3^C = 16$, $R_3 = 31 \leq 35$ for task τ_3. Hence, all tasks meet their deadlines.*

Effectively, the above example just shows that schedulability can be increased by lifting the assumption of a single static priority assignment to both phases. However, much stronger results can be drawn by rethinking Example 1 (used in Section 4 to show the sub-optimality of EDF) in light of the new results derived in this section. In that

example (Figure 1b)), the priority of task τ_1 changes from high to low when transitioning from its M- to its C-phase, which makes the task-set become schedulable. Additionally, the configuration in Figure 1b) corresponds to the critical instant scenario described by Theorem 8, meaning that in such a configuration both tasks experience their *exact* worst-case response time. It directly follows that Example 1 can be identically used to show that:

Corollary 1. No Fixed Task Priority (FTP) scheduling policy where all jobs of the same task have a single priority used for both M- and C-phases can be optimal;

Corollary 2. No Fixed Job Priority (FJP) scheduling policy where every job has a single priority used for both M- and C-phases can be optimal;

Corollary 3. FTP and FJP scheduling classes have a speedup factor which is lower bounded by 2;

Corollary 4. Dynamic Priority (DP) scheduling, where each job can change priority when transitioning from memory to computation phase, is necessary for optimality.

5.5 Two-phase priority assignment

The above results have demonstrated that a higher schedulability performance can be achieved by allowing different priorities for the M- and C-phase of each task. It now remains to address the problem of how to select priorities for the two phases. The correspondence between the analysis for M/C tasks and that for classic tasks with jitter [7] identified in Section 4.2 allows us to extend the result in [62] to the M/C scheduling model. Specifically, in [62] the authors proved that in the case of jitter, priorities can be optimally assigned according to increasing values of $D_i - J_i$, where J_i denotes the release jitter of task τ_i. This is referred to as Deadline minus Jitter Monotonic priority assignment. In our case, the *release jitter* of the C-phase of task τ_i is given by the response time of its M-phase, i.e., R_i^M. Since the value of R_i^M depends on the relative priority ordering of the M-phases, to get an overall optimal priority assignment for both M- and C-phases it would be required to exhaustively try each combination of priorities for the M-phases, with priorities for the C-phases then assigned according to the corresponding values of $D_i - R_i^M$, until schedulability is verified or all combinations have been tested. To reduce the computational burden of a brute-force search, we propose to allocate an intermediate deadline proportional to the work that must be executed by task τ_i in each phase. Similar priority assignment heuristics have proven effective in the context of multi-stage scheduling (e.g., [22]). More formally, we propose to assign priorities to the two phases as follows: for the M-phases, assign priorities by increasing values of $(D_i \cdot M_i)/(C_i + M_i)$; for the C-phases, assign priorities by increasing values of $D_i - R_i^M$. Experimental evaluation in Section 6 will demonstrate the effectiveness of the proposed heuristic strategy in comparison with a brute-force search of all possible priority orderings.

6 Experimental Results

To provide an experimental characterization of the performance improvement that may be obtained by adopting the M/C task model, we conducted a set of experiments applying the schedulability test proposed in Section 4 to randomly generated M/C workloads scheduled with fixed-priority on a single-core/single-memory setting. We then compared the number of schedulable task-sets detected by our test against classic approaches. Since the test is exact, i.e., necessary and sufficient, the results may be used to infer general properties of M/C sporadic task systems. In particular, we show that our approach efficiently exploits the pipelining of memory and execution phases, determining a significant schedulability improvement with respect to the classic sequential execution model and existing approaches for multi-stage systems. We then evaluate the priority assignment algorithms proposed in Section 5. Specifically, we quantify the schedulability gap between a sub-optimal priority assignment in combination with an exact test and the OPA algorithm in conjunction with a sufficient test. We also evaluate the performance gain that can be attained when different priorities may be assigned to the M- and C-phase of each task. The tests compared have been implemented in MATLAB®, and the code is available online [37].

6.1 Task-set generation

The generation of each task τ_k, $k \in \{1, \ldots, n\}$, is performed as follows: (i) the sum $V_k \stackrel{\text{def}}{=} M_k + C_k$ is uniformly selected in the interval $[10000, 1000000]$; (ii) the worst-case computation time C_k is computed as $\left\lfloor \frac{V_k}{f_{mc}+1} \right\rfloor$, where $f_{mc} \stackrel{\text{def}}{=} M_k/C_k$ is the *memory-to-computation ratio*; (iii) the worst-case memory access time M_k is then computed as $V_k - C_k$; (iv) the task utilization $u_k^M + u_k^C$ is generated using UUnifast [12]; (v) the period T_k is then calculated as $\left\lceil \frac{M_k+C_k}{u_k^M+u_k^C} \right\rceil$; (vi) the relative deadline D_k is uniformly selected in the interval $[M_k + C_k, T_k]$. In our experiments, three different schedulability tests based on response time analysis have been compared: (i) the exact test of Theorem 5, referred to as RTA-MC; (ii) the test in [26] based on delay composition (RTA-DC), restricted to the sub-case of fixed-priority scheduling and two-stage jobs; (iii) the response time analysis for classic sequential task systems given by Theorem 3, referred to as RTA, taking $E_k = M_k + C_k$ as total execution time of the task. In the initial experiments, Deadline Monotonic (DM) priority ordering is used. Priority assignment is further explored in Section 6.3.

6.2 Schedulability results

In the first set of experiments, we varied the total utilization of the task-set $U_{\mathcal{T}}$ from 0.1 to 1.5, generating 10000 task-sets for each value on the x-axis. Figure 4 shows the results with $n = 8$ tasks, and a memory-to-computation ratio f_{mc} selected with log-uniform distribution in the interval $[0.1, 10]$. As can be seen, RTA-MC outperforms RTA, especially for high values of $U_{\mathcal{T}}$, confirming that the pipelining of M- and C-phases is highly beneficial in terms of schedulability. For utilizations close to 0.9, the M/C model admits almost 50% of the generated task-sets, while the performance of classic RTA drops below 10%. As a notable aspect, RTA-MC is also able to schedule task-sets with $U_{\mathcal{T}} > 1$, which is obviously not possible using the RTA approach. This large performance gain is one of the main benefits brought by

the use of a pre-fetching execution model on a single-core platform. The performance of RTA-DC is significantly lower than RTA-MC at all utilization levels, and even lower than RTA, due to the conservative way of estimating the delay incurred by each execution stage.

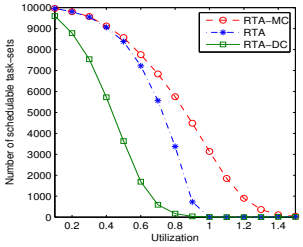

Figure 4: Experiments varying $U_{\mathcal{T}}$, with f_{mc} log-uniformly distributed in $[0.1, 10]$, and $n = 8$.

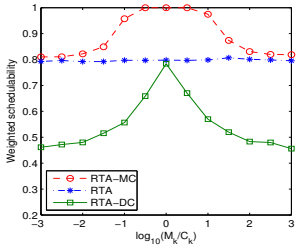

Figure 5: Weighted schedulability as a function of f_{mc}, with $n = 8$, and implicit deadlines.

We conducted other sets of experiments to observe how the schedulability performance varies depending on the value of the memory-to-computation ratio f_{mc} and the number of tasks in the system. Given the large design space to explore, we adopted an aggregate performance metric called *weighted schedulability* (see [11] for a definition).

Figure 5 reports in logarithmic scale the results of weighted schedulability when the observed parameter was the memory-to-computation ratio f_{mc}, varied in the interval $[10^{-3}, 10^{3}]$, with $n = 8$. While the classic RTA test is obviously not affected by variations in the memory-to-computation ratio, the M/C test has a peculiar behavior. Interestingly, when M_k is almost equal to C_k (i.e., f_{mc} is about 1, or, equivalently, $log_{10}(M_k/C_k)$ is around 0), the M/C test admits all the task-sets. When instead the two values are more unbalanced, the performance symmetrically degrades, until asymptotically reaching the performance of the completely sequential RTA. This intuitively means that when the duration of the two phases is comparable, the test can take full advantage of the pipelined execution of M- and C-phases. The RTA-DC test exhibits the same behavior as RTA-MC, since it can also take advantage of such a pipelined execution, but reaches a significantly lower schedulability performance due to the pessimism in the delay estimation. Only for values of f_{mc} close to 1, RTA-DC reaches the performance of the sequential RTA. Although Figure 5 refers to the implicit deadline case, we remark that the same trend is also present in the constrained deadline case, even if less evident due to the reduced slack available.

In the third set of experiments, we varied the number of tasks n in the interval $[2, 30]$, with f_{mc} selected with log-uniform distribution in $[0.1, 10]$. Figure 6 illustrates the results for the implicit deadline case, while Figure 7 refers to the constrained deadline case (i.e., relative deadlines are uniformly selected in the interval $[M_k + C_k, T_k]$). Under the constrained deadline model, the performance of all the tests degrades when n increases. However, when deadlines are implicit, RTA-MC seems to take advantage of the smaller granularity of the tasks (and relative M/C-phases) to obtain a better pipelining of memory and computation, identifying almost all generated task-sets as schedulable. RTA and RTA-DC also reach a constant trend, but are able to schedule a much smaller amount of task-sets (around 80% and 50%, respectively). The experiment in Figure 8 better clarifies how the performance of the tests varies depending on the deadline model. Here, we vary the factor α_d that controls the portion of the interval where the relative deadline can be selected. More specifically, for each value of α_d, the relative deadline D_k of a task τ_k is uniformly chosen in $[(M_k + C_k) + \lceil \alpha_d(T_k - (M_k + C_k)) \rceil, T_k]$. In the extreme case when $\alpha_d = 0$, the relative deadline is uniformly chosen in $[M_k + C_k, T_k]$; when instead $\alpha_d = 1$, all relative deadlines are implicit (i.e., $D_k = T_k$ for all tasks). The results show that by increasing α_d all tests perform significantly better due to the larger slack available. The performance improvement of RTA-MC is however much better, confirming the trend observed in Figures 6 and 7.

6.3 Priority assignment results

6.3.1 *Same priority for the two phases*

We now present the results of our empirical investigation on the priority assignment policies discussed in Section 5. We first focus on the case in which the same priority is assigned to both phases of each task. The priority assignment strategies studied are: (i) Deadline Monotonic (DM) in conjunction with the exact test of Theorem 5 (RTA-DM-EXACT); (ii) Audsley's OPA algorithm in conjunction with the sufficient test of Theorem 7 (RTA-OPA-SUFF); (iii) DM in conjunction with the sufficient test of Theorem 7 (RTA-DM-SUFF); (iv) Brute-force (BF) priority assignment, which exhaustively tries each combination of priorities, in conjunction with the exact test of Theorem 5 (RTA-BF).

Figure 9 illustrates the results of a set of experiments where the number of schedulable task-set has been observed as a function of the task-set utilization (varied in the interval $U_{\mathcal{T}} \in [0.1, 1.5]$ in steps of 0.1). The memory-to-computation ratio has been varied with log-uniform distribution in the interval $f_{mc} \in [0.1, 10]$. For each utilization value, 1000 task-sets have been generated with $n = 8$. In the figures, the performance gap between the exact test and the sufficient test is quantified by the distance between RTA-DM-EXACT and RTA-DM-SUFF. Also, the comparison between RTA-DM-EXACT and RTA-OPA-SUFF shows that in our case an optimal priority assignment is *not* enough to make up for the weaker schedulability test. Interestingly, this is the opposite trend with respect to the global fixed-priority case in a multicore setting [16], where an appropriate (optimal) choice of priority assignment, although combined with a sufficient test, yields larger schedulability than an OPA-incompatible tighter schedulability test combined with DM. We also evaluated the potential performance of the exact test with an exhaustive approach to priority assignment. In particular, the actual schedulability loss due to a sub-optimal priority assignment is given by the comparison between RTA-BF and RTA-DM-EXACT. This gap is extremely small, revealing that, despite being sub-optimal, DM gives extremely good performance in terms of admitted task-sets. Specifically, in this set of experiments only 25 task-sets out of 15000 are deemed schedulable by RTA-BF but not by RTA-DM-EXACT. In addition, the comparison between RTA-BF and RTA-OPA-SUFF shows exactly how much schedulability is lost due to a weaker test but optimal priority assignment.

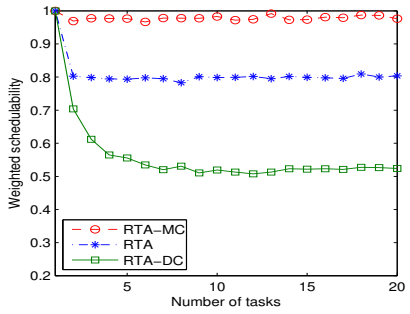

Figure 6: Experiments varying n, with f_{mc} log-uniformly distributed in $[0.1, 10]$, and implicit deadlines.

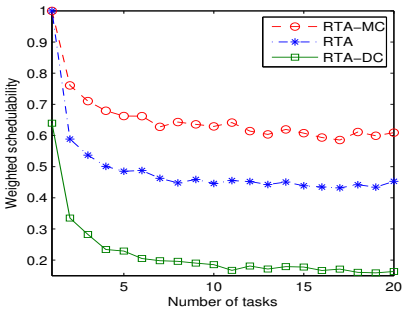

Figure 7: Experiments varying n, with f_{mc} log-uniformly distributed in $[0.1, 10]$, and constrained deadlines.

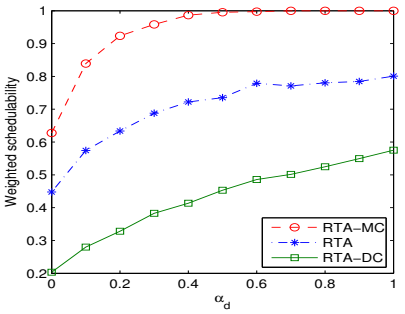

Figure 8: Experiments varying α_d, with f_{mc} log-uniformly distributed in $[0.1, 10]$, and $n = 8$.

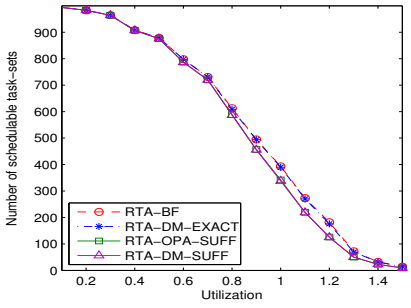

Figure 9: Evaluation of different priority assignments, $n = 8$ and f_{mc} log-uniformly distributed in $[0.1, 10]$.

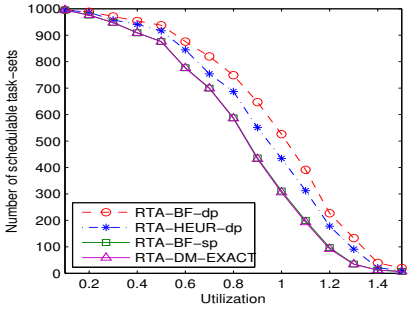

Figure 10: Evaluation of different priority assignments, $n = 8$ and f_{mc} log-uniformly distributed in $[0.1, 10]$.

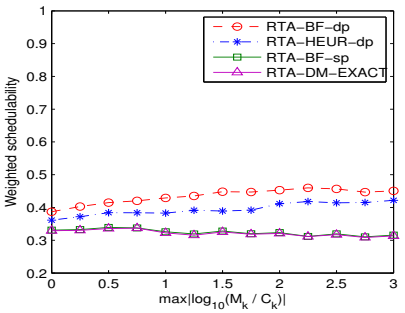

Figure 11: Evaluation of different priority assignments, $n = 8$ and f_{mc} log-uniformly distributed in $[10^{-x}, 10^{x}]$.

6.3.2 *Different priorities for the two phases*

We now extend our evaluation study to the case of different priorities for the two phases of each task, discussed in Sections 5.4 and 5.5. To quantify the schedulability improvement obtainable when different priorities are allowed for the two phases, we measured the number of schedulable task-sets as a function of $U_{\mathcal{T}}$ and examined the following priority assignment strategies: (i) RTA-BF-dp: brute-force priority assignment with two degrees of freedom (which tries each combination of priorities for the first phase and then applies $D_i - R_i^M$ to the second phase, as explained in Section 5.5), in combination with the exact test of Theorem 9; (ii) RTA-HEUR-dp: heuristic priority assignment proposed in Section 5.5 in combination with the exact test of Theorem 9; (iii) RTA-BF-sp: brute-force priority assignment (assuming the same priority for both phases) with the exact test of Theorem 5; (iv) RTA-DM-EXACT: DM over both phases with the exact test of Theorem 5.

Figure 10 reports the results for the case of $n = 8$, f_{mc} selected with log-uniform distribution in $[0.1, 10]$ and $U_{\mathcal{T}}$ varied in the interval $[0.1, 1.5]$ in steps of 0.1. The comparison between RTA-BF-dp and RTA-BF-sp exactly quantifies the schedulability improvement obtainable when (possibly different) priorities are optimally assigned to the two phases of each task. Also, while a negligible schedulability loss is observed between RTA-DM-EXACT and RTA-BF-sp, a more significant gap exists between RTA-HEUR-dp and RTA-BF-dp, mainly due to the sub-optimal priority assignment to the M-phase of each task (see Section 5.5). Specifically, in this experiment 436 task-sets out of 15000 are deemed schedulable by RTA-BF-dp but not by RTA-HEUR-dp. To complement the results in Figure 10, we explore how the performance of the different priority assignment strategies changes with an increasing range of values from which M_k and C_k can be chosen. In particular, we select the value of f_{mc} from a log-uniform distribution in the range $[10^{-x}, 10^{x}]$. The value of x is varied from 0 to 3 in steps of 0.25 to produce the weighted schedulability graph in Figure 11. Observing the distance between the first two curves (RTA-BF-dp and RTA-HEUR-dp) and the second two curves (RTA-BF-sp and RTA-DM-EXACT), we conclude that the improvement allowing different priorities for the two phases slightly increases with increasing x. Intuitively, this is because the need for two priorities per task is most acute when some tasks have a relatively long M-phase and others have a relatively long C-phase, thus allowing the disparate phases to be overlapped.

7 Conclusions and Future Work

In this paper, we took a further step towards the analytical characterization of predictable policies to co-schedule both memory and processing resources, considering sets of sporadic M/C tasks executing on a single-core/single-memory setting. We showed that existing results for classic task models are not applicable to the considered task model. In particular, we showed that no fixed task priority or fixed job priority scheduling algorithm has a speedup factor of less than 2 compared to an optimal algorithm for the M/C scheduling problem. Further we proved that the exact speedup factor for EDF is 2. For fixed priority scheduling, we showed that the synchronous arrival sequence does not necessarily result in the longest response times for M/C tasks, and therefore is not a critical instant. We derived a critical instant configuration and based on this result we

developed a necessary and sufficient schedulability analysis that computes the exact worst-case response time of each task, with a pseudo-polynomial complexity. We also investigated the problem of priority assignment for the M/C scheduling model. We discussed the applicability of existing results and studied the performance of different priority assignment algorithms, considering both the same and distinct priorities for the two phases of each task. We also extended our exact response time analysis to distinct priorities per phase of each task, enabling the derivation of optimality conditions for the M/C scheduling problem. Finally, we showed by extensive experiments that significant performance improvements may be obtained leveraging a pipelined execution of M- and C-phases, efficiently hiding the memory latency and improving schedulability. These results show the great potential of pre-fetching execution models, providing an important building block towards the design of predictable multi-core systems that are able to efficiently harmonize the provisioning of instruction/data to computing units, with a limited memory interference. As future work, we plan to study the impact of pre-fetching techniques on industrial real-time systems, implementing efficient co-scheduling algorithms in platforms featuring multiple cores and memory channels. We expect that further significant improvements may be obtained by exploiting burst read/write features to decrease the length of memory phases of M/C tasks. We also intend to integrate the M/C model with the limited preemption scheduling framework, to avoid a task being preempted once its context has been loaded to local memory. Finally, we aim to tackle different problems that remain open in this paper, such as the generalization to multi-phase tasks, and the extension to partitioned and global multiprocessor scheduling.

ACKNOWLEDGMENTS

This work has been supported by the European Commission through the HERCULES project (H2020/ICT/2015/688860), by the Inria International Chair program and by the ESPRC grant, MCC (EP/K011626/1). EPSRC Research Data Management: No new primary data was created during this study.

REFERENCES

[1] A. Alhammad and R. Pellizzoni. Schedulability analysis of global memory-predictable scheduling. In *EMSOFT*, 2014.

[2] A. Alhammad, S. Wasly, and R. Pellizzoni. Memory efficient global scheduling of real-time tasks. In *RTAS*, 2015.

[3] S. Altmeyer, R. Davis, and C. Maiza. Improved cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems. *Real-Time Systems Journal*, 48(5):499–526, 2012.

[4] S. Altmeyer, R. Douma, W. Lunniss, and R. Davis. On the effectiveness of cache partitioning in hard real-time systems. *Springer Real-Time Systems Journal*, pages 1–46, 2016.

[5] N. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Technical report YCS 164, Dept. Computer Science, University of York, UK, July 1991.

[6] N. Audsley. On priority assignment in fixed priority scheduling. *Information Processing Letters*, 79(1):39–44, 2001.

[7] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8(5):284–292, 1993.

[8] N. Audsley, A. Burns, M. Richardson, and A. Wellings. Hard real-time scheduling: the deadline-monotonic approach. In *RTOSS*, 1991.

[9] S. Bak, G. Yao, R. Pellizzoni, and M. Caccamo. Memory-aware scheduling of multicore task sets for real-time systems. In *RTCSA*, 2012.

[10] S. Baruah, A. K. Mok, and L. E. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *RTSS*, 1990.

[11] A. Bastoni, B. Brandenburg, and J. Anderson. Cache-related preemption and migration delays: empirical approximation and impact on schedulability. In *OSPERT*, 2010.

[12] E. Bini and G. C. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2):129–154, 2005.

[13] A. Burns and S. K. Baruah. Sustainability in real-time scheduling. *Journal of Computing Science and Engineering*, 2(1):74–97, 2008.

[14] G. Buttazzo, M. Bertogna, and G. Yao. Limited preemptive scheduling for real-time systems: a survey. *IEEE Transactions on Industrial Informatics*, 9(1):3–15, 2013.

[15] B. Chen. Analysis of classes of heuristics for scheduling a two-stage flow shop with parallel machines at one stage. *Journal of the Operational Research Society*, 46(2):234–244, 1995.

[16] R. Davis and A. Burns. Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. *Real-Time Systems Journal*, 47(1):1–40, 2010.

[17] R. Davis, A. Burns, S. Baruah, T. Rothvoss, L. George, and O. Gettings. Exact comparison of fixed priority and EDF scheduling based on speedup factors for both pre-emptive and non-pre-emptive paradigms. *Real-Time Systems*, 51(5):566–601, 2015.

[18] R. I. Davis, T. Rothvoß, S. K. Baruah, and A. Burns. Exact quantification of the sub-optimality of uniprocessor fixed priority pre-emptive scheduling. *Real-Time Systems*, 43(3):211–258, 2009.

[19] R. I. Davis, A. Thekkilakattil, O. Gettings, R. Dobrin, and S. Punnekkat. Quantifying the exact sub-optimality of non-preemptive scheduling. In *RTSS*, 2015.

[20] M. Dertouzos. Control robotics: the procedural control of physical processes. *Information Processing*, 74, 1974.

[21] J. Deverge and I. Puaut. WCET-directed dynamic scratchpad memory allocation of data. In *ECRTS*, 2007.

[22] J. Garcia and M. Harbour. Optimized priority assignment for tasks and messages in distributed hard real-time systems. In *PDRTSW*, 1995.

[23] A. Hamann, M. Jersak, K. Richter, and R. Ernst. A framework for modular analysis and exploration of heterogeneous embedded systems. *Real-Time Systems Journal*, 33(1-3):101–137, 2006.

[24] J. Hennessy and D. Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2011.

[25] J. Hoogeveen, J. Lenstra, and B. Veltman. Preemptive scheduling in a two-stage multiprocessor shop is NP-hard. *European J. Oper. Res.*, 89:172–175, 1996.

[26] P. Jayachandran and T. Abdelzaher. A delay composition theorem for real-time pipelines. In *ECRTS*, 2007.

[27] P. Jayachandran and T. Abdelzaher. Reduction-based schedulability analysis of distributed systems with cycles in the task graph. *Real-Time Systems Journal*, 46(1):121–151, 2010.

[28] S. Johnson. Optimal two- and three-stage production schedules with setup times included. *Naval Res. Logist. Q.*, 1:61–68, 1954.

[29] M. Joseph and P. Pandya. Finding response times in a real-time system. *The Computer Journal*, 29(5):390–395, 1986.

[30] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. *Journal of ACM*, 47(4):617–643, July 2000.

[31] J. Krakora, L. Waszniowski, P. Pisa, and Z. Hanzalek. Timed automata model for component-based real-time systems. In *WFCS*, 2004.

[32] J. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic real-time tasks. *Performance Evaluation*, 2(4):237–250, 1982.

[33] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the Association for Computing Machinery*, 20(1):46–61, 1973.

[34] J. Lu, H. Chen, R. Fu, W.-C. Hsu, B. Othmer, P. Yew, and D. Chen. The performance of runtime data cache prefetching in a dynamic optimization system. In *MICRO*, 2003.

[35] R. Mancuso, R. Dudko, and M. Caccamo. Light-prem: Automated software refactoring for predictable execution on cots embedded system. In *RTCSA*, 2014.

[36] A. Marongiu and L. Benini. An openMP compiler for efficient use of distributed scratchpad memory in mpsocs. *IEEE Transactions on Computers*, 61(2):222–236, 2012.

[37] A. Melani. A MATLAB® implementation of schedulability tests for memory-processor co-scheduling in fixed priority systems. http://retis.sssup.it/%7Eal.melani/downloads/MC.zip, 2015.

[38] A. Melani, M. Bertogna, V. Bonifaci, A. Marchetti-Spaccamela, and G. Buttazzo. Memory-processor co-scheduling in fixed priority systems. In *RTNS*, 2015.

[39] A. Melani, R. Mancuso, D. Cullina, M. Caccamo, and L. Thiele. Speed optimization for tasks with two resources. In *DATE*, 2016.

[40] M. D. Natale and J. A. Stankovic. Dynamic end-to-end guarantees in distributed real time systems. In *RTSS*, 1994.

[41] J. C. Palencia and M. Harbour. Offset-based response time analysis of distributed systems scheduled under EDF. In *ECRTS*, 2003.

[42] J. C. Palencia and M. G. Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *RTSS*, 1998.

[43] J. C. Palencia and M. G. Harbour. Exploiting precedence relations in the schedulability analysis of distributed real-time systems. In *RTSS*, 1999.

[44] R. Pellizzoni, E. Betti, S. Bak, J. Criswell, M. Caccamo, and R. Kegley. A predictable execution model for COTS-based embedded systems. In *RTAS*, 2011.

[45] R. Pellizzoni and G. Lipari. Holistic analysis of asynchronous real-time transactions with earliest deadline scheduling. *Journal of Computer and System Sciences*, 73(2):186–206, 2007.

[46] I. Puaut and C. Pais. Scratchpad memories vs locked caches in hard real-time systems: a quantitative comparison. In *DATE*, 2007.

[47] R. Rajkumar, L. Sha, and J. P. Lehoczky. On countering the effects of cycle-stealing in a hard real-time environment. In *RTSS*, 1987.

[48] J. Rosen, A. Andrei, P. Eles, and Z. Peng. Bus access optimization for predictable implementation of real-time applications on multiprocessor systems-on-chip. In *RTSS*, 2007.

[49] P. Schuurman and G. J. Woeginger. A polynomial time approximation scheme for the two-stage multiprocessor flow shop problem. *Theor. Comput. Sci.*, 237(1-2):105–122, 2000.

[50] L. Sha, J. P. Lehoczky, and R. Rajkumar. Solutions for some practical problems in prioritized preemptive scheduling. In *RTSS*, 1986.

[51] B. Sprunt, D. Kirk, and L. Sha. Priority-driven, preemptive I/O controllers for real-time systems. In *ISCA*, 1988.

[52] M. Spuri. Holistic analysis for deadline scheduled real-time distributed systems. Technical report RR-2873, INRIA, 1996.

[53] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *ISCAS*, 2000.

[54] K. Tindell, A. Burns, and A. Wellings. An extendible approach for analyzing fixed priority hard real-time tasks. *Real-Time Systems Journal*, 6(2):133–151, 1994.

[55] K. Tindell and J. Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing and Microprogramming*, 40(2):117–134, 1994.

[56] B. Ward, J. Herman, C. Kenna, and J. Anderson. Making shared caches more predictable on multicore platforms. In *ECRTS*, 2013.

[57] S. Wasly and R. Pellizzoni. A dynamic scratchpad memory unit for predictable real-time embedded systems. In *ECRTS*, 2013.

[58] S. Wasly and R. Pellizzoni. Hiding memory latency using fixed priority scheduling. In *RTAS*, 2014.

[59] J. Whitham and N. C. Audsley. Explicit reservation of local memory in a predictable, preemptive multitasking real-time system. In *RTAS*, 2012.

[60] G. Yao, R. Pellizzoni, S. Bak, E. Betti, and M. Caccamo. Memory-centric scheduling for multicore hard real-time systems. *Real-Time Systems Journal*, 48(6):681–715, 2012.

[61] S. Zhuravlev, J. C. Saez, S. Blagodurov, A. Fedorova, and M. Prieto. Survey of scheduling techniques for addressing shared resources in multicore processors. *ACM Comput. Surv.*, 45(1):4, 2012.

[62] A. Zuhily and A. Burns. Optimal (D - J) monotonic priority assignment. *Information Processing Letters*, 103(6):247–250, 2007.