Memory-Processor Co-Scheduling in Fixed Priority Systems

Alessandra Melani[1], Marko Bertogna[2], Vincenzo Bonifaci[3], Alberto Marchetti-Spaccamela[4], Giorgio Buttazzo[1]

[1] Scuola Superiore Sant'Anna - Pisa, Italy - {alessandra.melani, g.buttazzo}@sssup.it
[2] Università di Modena e Reggio Emilia - Modena, Italy - marko.bertogna@unimore.it
[3] Istituto di Analisi dei Sistemi ed Informatica, CNR - Roma, Italy - vincenzo.bonifaci@iasi.cnr.it
[4] Università di Roma "La Sapienza" - Roma, Italy - alberto@dis.uniroma1.it

ABSTRACT

A major obstacle towards the adoption of multi-core platforms for real-time systems is given by the difficulties in characterizing the interference due to memory contention. The simple fact that multiple cores may simultaneously access shared memory and communication resources introduces a significant pessimism in the timing and schedulability analysis. To counter this problem, predictable execution models have been proposed splitting task executions into two consecutive phases: a memory phase in which the required instruction 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 it also allows a more efficient (and predictable) pipelining of memory and execution phases through proper co-scheduling algorithms.

In this paper, we take a further step towards the design of smart co-scheduling algorithms for sporadic real-time tasks complying with the M/C (memory-computation) model. We provide a theoretical framework that aims at tightly characterizing the schedulability improvement obtainable with the adopted M/C task model on a single-core systems. We identify a tight critical instant for M/C tasks scheduled with fixed priority, providing an exact response-time analysis with pseudo-polynomial complexity. We show in our experiments that a significant schedulability improvement may be obtained with respect to classic execution models, placing an important building block towards the design of more efficient partitioned multi-core systems.

Keywords

Predictable execution models, Memory-aware scheduling, Co-scheduling algorithms, Schedulability analysis, Real-time systems

RTNS 2015 November 4 - 6 2015, Lille, France

1. INTRODUCTION

Classic real-time scheduling techniques are being challenged by the advent of many-core platforms integrating a large number of computing units on a single chip. As the core-count increases, the number of computing units becomes comparable to the number of tasks in the system, making the scheduling of the processing bandwidth less of a problem. However, another problem is jeopardizing the predictable exploitation of the huge computing power offered by these systems: the relatively slow access to memory and communication resources. In order to keep the computing units fed with fresh tasks to execute, it is necessary to overcome memory and communication bottlenecks, guaranteeing a proper provisioning of new data and instructions to each core. Therefore, *the scarce resource of interest to schedule is no more, or at least not only, the processing power, but memory bandwidth.*

The real-time community already identified the need for new scheduling algorithms and execution models that may simplify the derivation of tighter timing and schedulability analyses on multi-core platforms. A major effort in this sense is represented by the PREM scheduling framework introduced in [32]. Within this framework, tasks are split into different phases: a memory phase in which the task pre-fetches the required instruction and data from memory and/or I/O devices (M-phase), and an execution phase in which the task executes without needing to access shared memory and communication devices (C-phase). 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. In this paper, we will focus on the simpler model where each task is composed of just one memory phase followed by an execution phase. To distinguish this model from the generality of the task models adopted within the PREM framework, we will denote it as *M/C task model.*

The advantage of pre-fetching execution models is that they enable an easier characterization of the timing interference, decoupling memory and execution interferences into two distinct phases. During an M-phase, each task may experience only contention delays due to shared memory accesses. Conversely, during a C-phase, each task may be interfered only by the C-phases of other tasks concurrently scheduled on the same core. This allows computing the worst-case execution time (WCET) of the C-phase of each task using classic timing and schedulability analysis tools

developed for single core architectures, without needing to take into account memory contention and concurrent bus requests from other cores, but considering each core in isolation.

Execution models based on pre-fetching techniques are more amenable to timing analysis and have at least two other important advantages:

- 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].
- 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 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 were based on heuristic approaches and pessimistic schedulability analyses [32, 8, 43, 40, 3, 4]. Despite this pessimism, they showed a significant potential improvement with respect to classic (i.e., non pre-fetching) execution models. In this work, we aim at providing a more formal characterization of the schedulability of M/C task systems, identifying critical instant scenarios leading to worst-case response-times, and providing an exact schedulability test to evaluate the theoretical improvement achievable with the adopted task model.

For this purpose, we will tackle the schedulability problem from a theoretical side, placing an important building block towards the definition of a framework that is able to fully exploit the potential of co-scheduling techniques. We will make assumptions that aim at simplifying the presented proofs and the exposition of the main theoretical concepts, focusing on the principal aspects of the schedulability problem. In particular, we will assume a preemptive fixed-priority scheduler, with each task having the same (static) priority for both memory and computation phases, and no preemption overhead. Postponing a discussion on the adopted assumptions to Section 3, we here focus on the definition of this simple schedulability problem: given a set of sporadic M/C tasks, identify whether it is schedulable with a given priority assignment.

At first sight, one may think that the 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, (i) preemptive EDF is not an optimal scheduling algorithm for M/C task systems; and (ii) the synchronous arrival of all tasks, 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 [22] to characterize the schedulability of M/C task systems.

The above considerations motivated us to analyze in more detail the scheduling and schedulability problems of M/C task systems, investigating better algorithms and tests to be able to fully collect the potential of pre-fetching execution models for real-time applications.

Contributions of the paper. This paper aims at establishing the theoretical background to better understand the schedulability of M/C task systems. In particular, it will provide the following contributions for a configuration with a single core and single memory channel:

- A proof that EDF is not an optimal scheduling algorithm for M/C task systems, but it has a lower speedup bound of 2 from an optimal scheduler.
- The definition of a tight critical instant for M/C task systems scheduled with fixed-priority, and the relative proof that no other task release configuration may produce a larger response-time.
- An exact response-time analysis for M/C systems scheduled with fixed-priority, leading to a necessary and sufficient schedulability test.

Then, we will characterize the schedulability improvement obtainable in a single-core/single-memory setting by means of extensive simulations using randomly generated workloads, highlighting which systems are more likely to benefit, and to which extent, from pre-fetching execution models.

Organization of the paper. The remainder of the paper is organized as follows. The next section presents the related work on predictable execution models and related co-scheduling algorithms. Section 3 presents the adopted task model and the assumptions made throughout the paper, discussing their validity. Section 4 presents multiple schedulability results for single core systems. Experimental results are provided in Section 5 to validate the effectiveness of the proposed techniques.

2. RELATED WORK

Pre-fetching techniques are widely adopted in the embedded and high-performance computing domain for different complementary reasons. As shown in [25], these techniques allow improving the cache (or scratchpad) locality reducing *average* execution times. When coupled with double buffering techniques, they also allow hiding the memory latency by executing a pre-fetched task while pre-fetching the context of another one [27]. 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.

[1]See, e.g., Texas Instrument Keystone II [1], where a 20x speedup can be obtained exploiting the burst read features of the integrated DMA engines.

This latter target is pursued in a seminal paper by Pellizzoni et al. [32] through the definition of the Predictable Execution Model (PREM). As explained in the introduction, such a model splits the execution of critical tasks into two distinct phases: a memory phase where the task context is pre-fetched into local memory, and an execution phase where the task executes with no memory contention. 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. An automatic tool for code refactoring is presented in [26], 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], etc.) leveraging offloading directives that explicitly distinguish between shared and private data items, and that allow data/instruction prefetching [27].

An orthogonal approach to increase the cache locality and improve the predictability of memory accesses is using cache locking [34, 39] or partitioning [5] techniques. Scratchpad memory allocation has been considered in [16] for single task scenarios. In [42], the Carousel mechanism was proposed for dynamic scratchpad management in a multitasking system scheduled with RM. However, both approaches stall the CPU while loading tasks to scratchpad, and, therefore, do not take advantage of the overlapping of memory and execution phases.

To allow the simultaneous execution of memory and execution phases, a dynamic scratchpad management technique has been proposed in [41]. In [8], different scheduling algorithms for PREM-compliant task systems are compared with a simulation-based approach. In [43], a TDMA-based scheduling algorithm is proposed for PREM tasks on a multicore platform. In [40], a schedulability analysis is presented for non-preemptable PREM tasks on a single core or a partitioned multicore system. In [3, 4], the scheduling and schedulability problems for globally scheduled PREM tasks are addressed. All these works are based on heuristic scheduling approaches and only sufficient schedulability analyses relying on pessimistic upper-bounds on the worst-case interference experienced by each task.

To 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 [38, 36, 30, 31, 33]. 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 [38, 30, 29] and EDF-based systems [36, 31, 33]. 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 [30]. Alternative sufficient analyses for real-time distributed systems include the use of per-stage deadlines [28], real-time calculus [37], timed automata [23], compositional analysis [17], and delay composition algebra [20, 19]. The latter approach seems to provide the best trade-off between schedulability performance and complexity, and we will use it as a term of comparison 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 machine, the problem has a polynomial solution [21]; however, if at least one stage consists of two or more machines, then the problem becomes strongly NP-hard [18]. For this reason, several heuristics have been proposed [13]. Recently, a PTAS has also been proposed in [35].

3. SYSTEM MODEL

We consider a set $\mathcal{T}$ of n periodic and sporadic real-time tasks $\tau_1, \ldots, \tau_n$. 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 will call *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., $R_k = \max_{j \in \tau_k}(f_k^j - r_k^j)$. We denote with R_k^M the worst-case response-

[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

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 computational phase of τ_k, i.e., from the end of the memory phase until the completion of the C-phase.

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 [41], 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.

To simplify the model, potential 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, automotive and avionic control systems, the output of the computation phase is typically restricted to a few actuation operations or detection signals. Moreover, as explained in [32, 40], potential write-back phases may be easily combined with the (read) M-phase of the subsequently scheduled instance.

One last remark concerns the preemption overhead, which is neglected in our scheduling model. This assumption is commonly adopted in the real-time literature to simplify the analysis, allowing the derivation of exact schedulability tests, optimal scheduling algorithms and, in general, a cleaner understanding of the scheduling problem. However, the impact of preemptions on the system schedulability should be carefully analyzed before applying the theoretical results to a practical use case. In particular, this assumption becomes less valid when task footprints are comparable to the size of the local cache/scraptchpad memory, in which case a preempting task may evict a significant amount of useful memory blocks to a preempted task, leading to a considerable preemption delay. In this paper, we assume the local memory be sufficiently large to allow neglecting the preemption overhead. However, we underline that this assumption has been introduced to simplify the derivation of the theoretical background required for a sound schedulability analysis of pre-fetching execution models. As done with classic scheduling models, such an analysis will then be leveraged to integrate preemption overhead in future works. We are currently working on two alternative strategies: (i) factorizing the memory penalty required by a preempted task to restart its memory phase when resuming after a preemption; and (ii) integrating the presented schedulability analysis with the limited preemption framework [12]. This last approach seems particularly promising to significantly limit, or even avoid, the preemption overhead by encapsulating consecutive M/C phases within a non-preemption region.

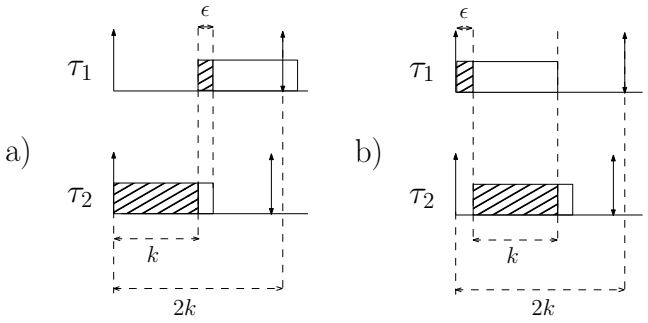

Figure 1: EDF is not optimal for arbitrary collections of M/C jobs (striped blocks are M-phases).

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 [15], it is provably not optimal for M/C jobs, as shown in the following example.

Example 1. *Consider a system composed of two jobs: a job J_1 with a memory phase $M_1 = \epsilon$, a computation phase $C_1 = k$, and a deadline $D_1 = 2k$; and a job J_2 with a memory phase $M_2 = k$, a computation phase $C_2 = \epsilon$, and a deadline $D_2 = 2k - \epsilon$. Figure 1 inset a) shows the schedule with EDF: job J_1 is given lower priority than J_2, resulting in a response-time of $2k + \epsilon$ for J_1, missing its deadline. Instead, in inset b), job J_1 is given higher priority than J_2, so that both jobs meet their deadlines.*

By taking $k >> \epsilon$, the above example also shows that *EDF is far from an optimal scheduling algorithm by a speedup factor of at least 2.*

Another result that is no more valid for the M/C model concerns the concept of critical instant for periodic and sporadic tasks. A critical instant is a particular release configuration that leads to the largest possible response-time for a given task. For regular (independent) task instances, a critical instant is given by the synchronous release of all tasks, with jobs released as soon as possible, i.e., with consecutive task instances separated by their minimum inter-arrival time [24]. The following example shows that this is no more true for M/C task systems.

Example 2. *Consider a system composed of two periodic or sporadic tasks: a task τ_1 with a memory phase $M_1 = 0$, a computation phase $C_1 = 2$, and a deadline $D_1 = 2$; and a task τ_2 with a memory phase $M_2 = 2$, a computation 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[4] will immediately start executing the computation phase of τ_1, completing right before its deadline; meanwhile, τ_2 executes its memory 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.*

[4]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).

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 1 (from [22]). *For classic sporadic task systems (where each task τ_i has only one single execution phase of worst-case length E_i) with constrained deadlines scheduled with fixed-priority, the worst-case response-time of a task τ_k can be computed by finding R_k from the following iterative relation, starting with $R_k = E_k$:*

$$R_k \leftarrow \sum_{j \leq k} \left\lceil \frac{R_k}{T_j} \right\rceil E_j. \quad (1)$$

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 [38, 36, 30, 31, 33], 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 a tight response-time analysis for the considered setting.

4.1 Critical Instant

The problem in deriving a tight 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 (2)), 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 viceversa. 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:

$$R_k = R_k^M + R_k^C. \quad (2)$$

We hereafter 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 the 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, according to the following definitions.

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.*

The following lemma proves that there is at most one dual-interfering job per higher priority task.

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. Since, due to the constrained deadline model, each higher priority task τ_i has at most one job ready at time ϕ_k, the theorem follows. □

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 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 2 (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 $\tau_i, 1 \leq i < k$ 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.

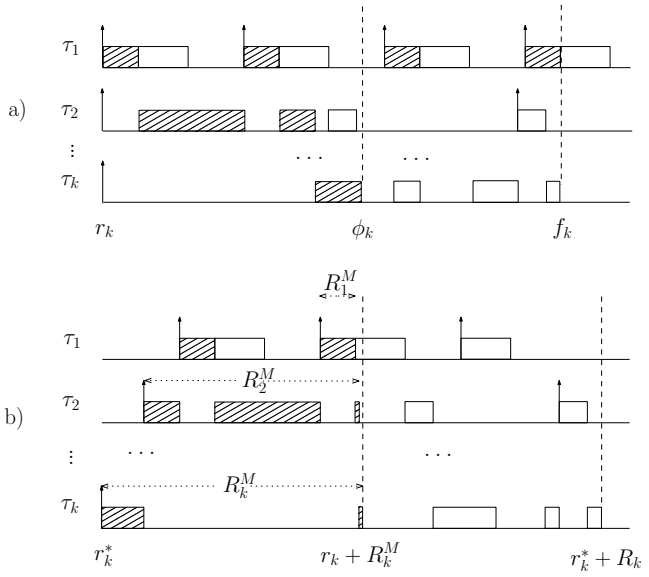

Figure 2: Synchronous release (a) and critical instant configuration (b).

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

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 [24]. 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 2.*

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 2, 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, proving the lemma. □

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 in [11]. 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 2.

4.2 Exact Response-Time Analysis

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

Theorem 3. *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 $R_k^M = M_k$:*

$$R_k^M \leftarrow \sum_{i \leq k} \left\lceil \frac{R_k^M}{T_i} \right\rceil M_i, \tag{3}$$

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

$$R_k^C \leftarrow C_k + \sum_{i<k} \left\lceil \frac{R_k^C + R_i^M}{T_i} \right\rceil C_i. \tag{4}$$

Proof. Consider the critical instant configuration of Theorem 2. 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 (2).

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 (3).

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 [7], where the worst-case response-time of the M-phase behaves as a release

jitter for the C-phase[5]. 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 2). 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:

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

By simplifying the terms, Equation (4) follows, proving the theorem. □

A simple (necessary and sufficient) schedulability test can be found by checking whether the worst-case response-time R_k computed with Theorem 3 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.

We note that the test of Theorem 3 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 (4) has to be limited to higher priority tasks assigned to the same core of the considered task τ_k, while Equation (3) is still extended to all higher priority tasks. While a deeper analysis of partitioned multi-core systems is left as a future work, we would like to highlight that such systems are likely to magnify the performance improvements obtainable with the adopted M/C scheduling model, avoiding unpredictable memory contentions due to simultaneous accesses to shared memory, and exploiting a harmonized pipelining of memory and execution phases. We are currently working on smart partitioning strategies based on the exact analysis developed in this paper.

5. EXPERIMENTAL RESULTS

To provide an experimental characterization of the performance improvement that may be obtained 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.

The tests compared have been implemented in MATLAB®, and the code is fully available online [2].

5.1 Task-set generation

The generation of each task τ_k, $k \in \{1, \ldots, n\}$, is performed as follows:

- the worst-case computation time C_k is uniformly selected in the interval $[10, 1000]$;
- the worst-case memory access time M_k is then computed as $\lfloor f_{mc} C_k \rfloor$, where $f_{mc} \stackrel{\text{def}}{=} M_k/C_k$ is the *memory-to-computation ratio*;
- the task utilization $u_k^M + u_k^C$ is generated using UUniFast [10];
- the period T_k is then calculated as $\left\lceil \frac{M_k + C_k}{u_k^M + u_k^C} \right\rceil$;
- the relative deadline D_k is uniformly selected in the interval $[M_k + C_k, T_k]$.

Task priorities are assigned according to the Deadline Monotonic (DM) ordering. Although DM is not an optimal priority assignment for M/C task systems (Example 1 can be identically used for the DM case), it seems to be the best available option. In fact, no optimal priority assignment algorithm is known so far for M/C task systems. Moreover, it is possible to show that Audsley's algorithm [6] is not directly applicable, due to the dependence of the response-time of each task on the relative priority ordering of higher-priority tasks (see Equation (4)). As observed in [14], such a property forbids the applicability of Audsley's bottom up priority assignment. We plan to further investigate the problem of priority assignment for M/C task systems in a future work.

In our experiments, three different schedulability tests based on response-time analysis have been compared:

- the exact test of Theorem 3, referred to as RTA-MC;
- the test in [20] based on delay composition (RTA-DC), restricted to the sub-case of fixed-priority scheduling and two-stage jobs;
- the response-time analysis for classic sequential task systems given by Theorem 1, referred to as RTA, taking $E_k = M_k + C_k$ as total execution time of the task.

5.2 Evaluation of Experiments

In the first set of experiments, we varied the total utilization of the task-set U_τ from 0.1 to 1.5, generating 10000 task-sets for each value on the x-axis. Figure 3 shows the results with $n = 8$ tasks, and a memory-to-computation ratio $f_{mc} = 0.5$. As it can be seen, RTA-MC outperforms RTA, especially for high values of U_τ, confirming that the pipelining of memory and execution phases is highly beneficial in terms of schedulability. For task-set utilizations close to 0.9, the M/C model allows scheduling 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 having $U_\tau > 1$, which is obviously not possible using the classic RTA approach. The performance

[5]Note that the correspondence between the analysis for M/C tasks and that for classic tasks with release jitter does not trivially descend from the models, but from the critical instant configuration identified in Section 4.1 which is shown to *jointly* maximize the worst-case response-time of the M-phase (i.e., the maximum jitter) and that of the C-phase.

of RTA-DC, however, 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.

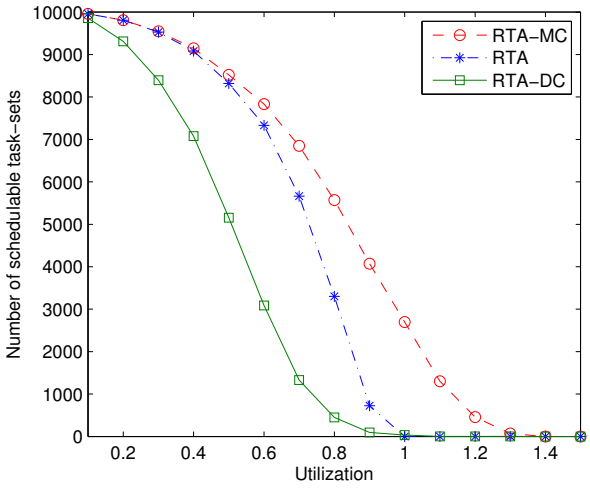

Figure 3: Experiments varying $U_{\mathcal{T}}$, with $f_{mc} = 0.5$ and $n = 8$.

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* [9], which allows reducing a tridimensional plot to a bidimensional one. In particular, let $S(p, u) \in [0, 1]$ denote the schedulability ratio for a given parameter p and utilization value u, picked in a set $\mathcal{U} = \{0.1, 0.2, \ldots, 1\}$ of evenly-spaced utilization levels. Then, the weighted schedulability $W(p)$ can be defined as

$$W(p) = \frac{\sum_{u \in \mathcal{U}} u \cdot S(U, p)}{\sum_{u \in \mathcal{U}} u}.$$

Figure 4 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 4 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.

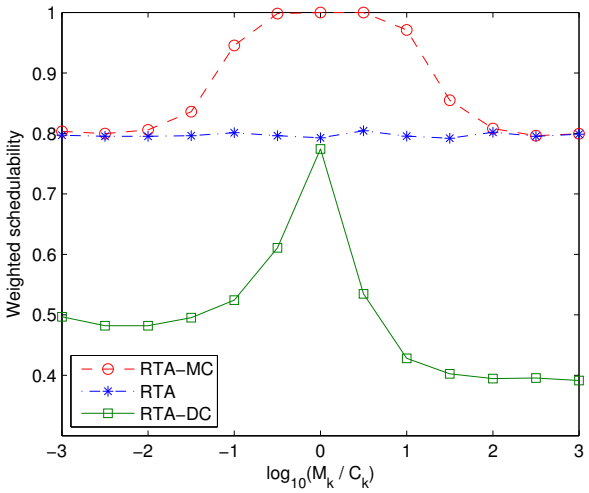

Figure 4: Experiments varying f_{mc}, with $n = 8$, and implicit deadlines.

In the third set of experiments, we varied the number of tasks n in the interval $[2, 30]$, with $f_{mc} = 0.1$. Figure 5 illustrates the results for the implicit deadline case, while Figure 6 refers to the constrained deadline case. 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 7 better clarifies how the performance of the tests varies depending on the deadline model. Here, we set $n = 8$ and $f_{mc} = 0.1$, varying 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 5 and 6.

6. CONCLUSIONS AND FUTURE WORKS

Processor-centric scheduling techniques where to accommodate memory interferences are no longer able to exploit the immense computing power offered by many-core platforms. 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, and identified a tight critical instant configuration that leads to the largest possible response-time of M/C tasks scheduled with fixed-priority. Based on this result, we developed a response-time analysis that allows exactly comput-

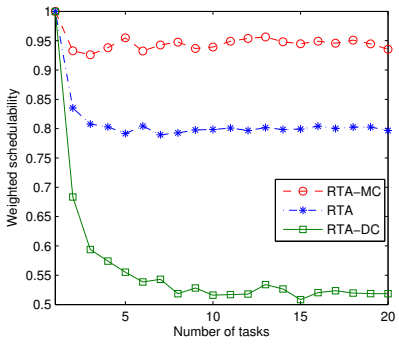

Figure 5: Experiments varying n, with $f_{mc} = 0.1$ and implicit deadlines.

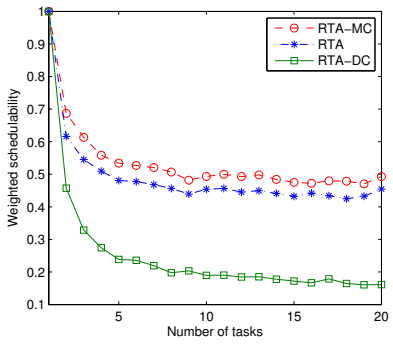

Figure 6: Experiments varying n, with $f_{mc} = 0.1$ and constrained deadlines.

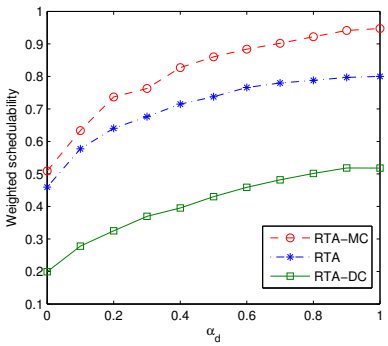

Figure 7: Experiments varying α_d, with $f_{mc} = 0.1$, and $n = 8$.

ing the worst-case response-time of each task, with a reduced (pseudo-polynomial) complexity. Finally, we showed by extensive simulations that significant performance improvements may be obtained leveraging a pipelined execution of memory and execution phases, efficiently hiding the memory latency and improving the 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. In future works, we plan to better study the impact of pre-fetching techniques to 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 while its context has already been loaded to local memory. Finally, we aim at tackling different problems remained open in this paper, like the derivation of optimal scheduling algorithms and priority assignments, a generalization to multi-phase tasks, and the extension to partitioned and global multiprocessor scheduling.

7. REFERENCES

[1] *Texas Instruments. The 66AK2H12 Keystone II Processor*. http://www.ti.com/product/66AK2H12.

[2] *A MATLAB® implementation of schedulability tests for memory-processor co-scheduling in fixed priority systems.* http://retis.sssup.it/∼al.melani/downloads/MC.zip, 2015.

[3] A. Alhammad and R. Pellizzoni. Schedulability analysis of global memory-predictable scheduling. In *International Conference on Embedded Software (EMSOFT 2014)*, New Delhi, India, October 12-17, 2014.

[4] A. Alhammad, S. Wasly, and R. Pellizzoni. Memory efficient global scheduling of real-time tasks. In *21st IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2015)*, Seattle, WA, USA, April 13-16, 2015.

[5] S. Altmeyer, R. Douma, W. Lunniss, and R. Davis. Evaluation of cache partitioning for hard real-time systems. In *26th Euromicro Conference on Real-Time Systems (ECRTS 2014)*, Madrid, Spain, July 8-11, 2014.

[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, September 1993.

[8] S. Bak, G. Yao, R. Pellizzoni, and M. Caccamo. Memory-aware scheduling of multicore task sets for real-time systems. In *18th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2012)*, Seoul, Korea, August 19-22, 2012.

[9] A. Bastoni, B. Brandenburg, and J. Anderson. Cache-related preemption and migration delays: empirical approximation and impact on schedulability. In *6th International Workshop on Operating Systems for Embedded Real-Time Applications (OSPERT 2010)*, Brussels, Belgium, July 6, 2010.

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

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

[12] 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.

[13] 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.

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

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

74, 1974.

[16] J. Deverge and I. Puaut. Wcet-directed dynamic scratchpad memory allocation of data. In *19th Euromicro Conference on Real-Time Systems (ECRTS 2007)*, Pisa, Italy, July 4-6, 2007.

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

[18] 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.

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

[20] P. Jayachandran and T. Abdelzaher. A delay composition theorem for real-time pipelines. In *19th Euromicro Conference on Real-Time Systems (ECRTS 2007)*, Pisa, Italy, July 4-6, 2007.

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

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

[23] J. Krakora, L. Waszniowski, P. Pisa, and Z. Hanzalek. Timed automata model for component-based real-time systems. In *5th IEEE International Workshop on Factory Communication Systems (WFCS 2004)*, Vienna, Austria, September 22-24, 2004.

[24] 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, January 1973.

[25] 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 *36th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2003)*, San Diego, CA, USA, December 3-5, 2003.

[26] R. Mancuso, R. Dudko, and M. Caccamo. Light-prem: Automated software refactoring for predictable execution on cots embedded system. In *20th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2014)*, Chongqing, China, August 20-22, 2014.

[27] 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, February 2012.

[28] M. D. Natale and J. A. Stankovic. Dynamic end-to-end guarantees in distributed real time systems. In *15th IEEE Real-Time Systems Symposium (RTSS 1994)*, San Juan, Puerto Rico, December 7-9, 1994.

[29] J. C. Palencia and M. G. Harbour. Exploiting precedence relations in the schedulability analysis of distributed real-time systems. In *20th IEEE Real-Time Systems Symposium (RTSS 1999)*, Phoenix, AZ, USA, December 1-3, 1999.

[30] J. C. Palencia and M. G. Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *19th IEEE Real-Time Systems Symposium (RTSS 1998)*, Madrid, Spain, December 2-4, 1998.

[31] J. C. Palencia and M. G. Harbour. Offset-based response time analysis of distributed systems scheduled under EDF. In *15th Euromicro Conference on Real-Time Systems (ECRTS 2003)*, Porto, Portugal, July 2-4, 2003.

[32] R. Pellizzoni, E. Betti, S. Bak, J. Criswell, M. Caccamo, and R. Kegley. A predictable execution model for COTS-based embedded systems. In *17th IEEE Real-Time and Embedded Applications Symposium (RTAS 2011)*, Chicago, IL, USA, April 11-14, 2011.

[33] 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, March 2007.

[34] I. Puaut and C. Pais. Scratchpad memories vs locked caches in hard real-time systems: a quantitative comparison. In *Design, Automation and Test in Europe Conference and Exhibition (DATE 2007)*, Nice, France, April 16-20, 2007.

[35] 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.

[36] M. Spuri. Holistic analysis for deadline scheduled real-time distributed systems. Technical report rr-2873, INRIA, France, April 1996.

[37] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *IEEE International Symposium on Circuits and Systems (ISCAS 2000)*, Geneva, Switzerland, May 28-31, 2000.

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

[39] B. C. Ward, J. L. Herman, C. J. Kenna, and J. H. Anderson. Making shared caches more predictable on multicore platforms. In *25th Euromicro Conference on Real-Time Systems (ECRTS 2013)*, Paris, France, July 9-12, 2013.

[40] S. Wasly and R. Pellizzoni. Hiding memory latency using fixed priority scheduling. In *20th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2014)*, Berlin, Germany, April 15-17, 2014.

[41] S. Wasly and R. Pellizzoni. A dynamic scratchpad memory unit for predictable real-time embedded systems. In *25th Euromicro Conference on Real-Time Systems (ECRTS 2013)*, Paris, France, July 9-12, 2013.

[42] J. Whitham and N. C. Audsley. Explicit reservation of local memory in a predictable, preemptive multitasking real-time system. In *18th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2012)*, Bejing, China, April 17-19, 2012.

[43] 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, November 2012.