2017-06-28

WCET derivation under single core equivalence with explicit memory budget assignme

WCET Derivation Under Single Core Equivalence With Explicit Memory Budget Assignment*

Renato Mancuso[1], Rodolfo Pellizzoni[2], Neriman Tokcan[3], and Marco Caccamo[4]

1 **University of Illinois at Urbana-Champaign, Urbana, IL, USA**
`rmancus2@illinois.edu`
2 **University of Waterloo, Waterloo, Canada**
`rpellizz@uwaterloo.ca`
3 **University of Illinois at Urbana-Champaign, Urbana, IL, USA**
`tokcan2@illinois.edu`
4 **University of Illinois at Urbana-Champaign, Urbana, IL, USA**
`mcaccamo@illinois.edu`

Abstract

In the last decade there has been a steady uptrend in the popularity of embedded multi-core platforms. This represents a turning point in the theory and implementation of real-time systems. From a real-time standpoint, however, the extensive sharing of hardware resources (e.g. caches, DRAM subsystem, I/O channels) represents a major source of unpredictability. Budget-based memory regulation (throttling) has been extensively studied to enforce a strict partitioning of the DRAM subsystem's bandwidth. The common approach to analyze a task under memory bandwidth regulation is to consider the budget of the core where the task is executing, and assume the worst-case about the remaining cores' budgets.

In this work, we propose a novel analysis strategy to derive the WCET of a task under memory bandwidth regulation that takes into account the exact distribution of memory budgets to cores. In this sense, the proposed analysis represents a generalization of approaches that consider (i) even budget distribution across cores; and (ii) uneven but unknown (except for the core under analysis) budget assignment. By exploiting the additional piece of information, we show that it is possible to derive a more accurate WCET estimation. Our evaluations highlight that the proposed technique can reduce overestimation by 30% in average, and up to 60%, compared to the state of the art.

1998 ACM Subject Classification C.3 Real-Time and Embedded Systems

Keywords and phrases real-time multicore, WCET, single-core equivalence, DRAM management, certification

Digital Object Identifier 10.4230/LIPIcs.ECRTS.2017.3

1 Introduction

Multi-core system-on-chip (SoC) are mainstream products. In multi-core platforms, applications can concurrently access shared hardware resources, such as: DRAM, memory bus(es), shared cache(s), and I/O channels. From a real-time perspective, the extensive sharing of

* The material presented in this paper is based upon work supported by the National Science Foundation (NSF) under grant numbers CNS-1302563 and CNS-1646383, NSERC DG 402369-2011 and CMC Microsystems. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the NSF and other sponsors.

29th Euromicro Conference on Real-Time Systems (ECRTS 2017).
Editor: Marko Bertogna; Article No. 3; pp. 3:1–3:23
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

hardware resources shakes a fundamental assumption of traditional real-time theory, i.e. that the composition of individually analyzed real-time tasks could be used to reason about the system as a whole. Many factors affect the composability of multi-core systems. There is a large consensus, however, that unregulated contention at the level of memory resources (caches, buses, DRAM banks) represents a major source of unpredictability. In fact, when one or more of these resources creates a bottleneck, it causes severe inter-core interference. When a given shared resource becomes a bottleneck with respect to overall system's activity, the worst-case execution time (WCET) of a task on a core can be significantly impacted. Clearly, without imposing a deterministic regulation strategy to cope with high memory resource utilization, interference-induced delays can be hard to model, forcing WCET and schedulability analysis to be more pessimistic.

In recent years, there has been a downtrend in the production and availability of high-performance single-core chips. On the other hand, avionic and automotive industries possess a large base of certified software developed for single-core chips. As such, an *en-masse* migration of industrial players in the safety-critical systems domain are due to a migration to multi-core SoCs. Unfortunately, due to inter-core interference, schedulability analysis results derived for single-core systems cannot be reused when migrating to multi-core platforms. As a part of our previous work, we proposed Single Core Equivalence technology (SCE) [21]. Under SCE, access to shared memory resources are strictly partitioned using a set of OS-level techniques. SCE exploits budget-based memory bandwidth regulation to cope with concurrent activity of tasks on the DRAM subsystem. In our previous work [21, 28], we considered even memory budget assignment across cores, and proposed an analytic model of SCE to estimate the WCET(m) of a task under analysis given the knowledge of its behavior in isolation. In this work, we derive the WCET analysis by making the following extensions: (i) we relax the assumption that the main memory controller bandwidth is evenly distributed among the different cores; and (ii) we consider the exact distribution of memory budgets to cores, and derive a more accurate WCET estimation. Hence, in this paper, we discuss how to calculate the WCET based on the specific budget assignment $\mathcal{Q}$ known at system design time. Only the m active cores have a non-zero budget assignment in $\mathcal{Q}$. It follows that the new WCET($\mathcal{Q}$) implicitly depends on m, but also takes into account the specific setup of a per-core memory bandwidth regulation mechanism. We detail the derivation of WCET($\mathcal{Q}$) for a real-time task given a knowledge of its behavior in isolation. This is the first work to derive a WCET analysis with *uneven* and *explicitly known* memory budget distribution for tasks that run on top of a performance isolation framework for COTS multi-core systems. Thereby, this work makes the following contributions:

- Analysis of WCET for tasks under budget-based memory bandwidth regulation with explicit per-core budget assignments;
- Extension of response-time analysis to incorporate the new WCET derivation for tasks that are scheduled synchronously (bound) or asynchronously (unbound) with respect to the memory regulation period;
- Simulation-based evaluation to compare the derived analysis with exact (brute-force) analysis and state-of-the-art analysis for uneven memory budgets, as proposed in [37].

The rest of the paper is organized as follows. Section 2 provides an overview of the related work. We first briefly review the key components of the SCE framework in Section 3. Next, we discuss the system model and assumptions in Section 4. Section 5 details the proposed WCET($\mathcal{Q}$) analysis, while scheduling considerations are discussed in Section 6. Simulation-based evaluation is presented in Section 7. The paper concludes in Section 8.

2 Related Work

The problem of inter-core interference in multi-core architectures is well known and has been largely studied in literature. As a result, several different ways of approaching the new challenges introduced by multi-core platforms have been proposed. Static analysis has been widely adopted to model the behavior of shared caches in [35, 17, 12] and shared memory buses [26, 14]. A system-wise static analysis for an abstracted multi-core platform that takes into account CPU pipelines, a shared cache and a main memory bus has been proposed in [6], on top of which a unified WCET framework has been formulated [5]. Static analysis certainly represents a promising approach to the problem of determining the WCET of tasks in a multi-core system. Nonetheless, at the current level of refinement, strong assumptions are needed to carry out static analysis, e.g. a LRU cache policy, TDMA-based buses, or the presence of a fixed workload in the system.

A different approach consists in designing multi- and many-core architectures to implement deterministic resource sharing schemes to provide better guarantees on the WCET of real-time tasks. The precision timed (PRET) architecture [9, 4] embeds runtime control and deadline enforcement at the level of processor instruction set while proposing a set of hardware modifications to achieve good performance without sacrificing predictability. In the context of PRET, Reineke et al. proposed a PRET DRAM controller [25] that prevents contention at the level of memory controller by partitioning the physical address space. Specific hardware support in the memory subsystem was also proposed in [22] to lower the pessimism in WCET estimation. Predator, a predictable DRAM controller that internally implements bandwidth regulators was proposed in [2]. Similarly, predictability is enhanced in the MERASA project [32] by reducing inter-core interference at the hardware level.

This work targets commercially available systems, and effectively improves the analysis of a software-based memory bandwidth regulation technique, namely MemGuard [40, 41]. The most related work in this area concerns the design and analysis of protocols for access to shared (memory) resources. Time division multiple access (TDMA) was proposed in [26] as an arbitration protocol for shares buses. A timing analysis for TDMA arbitration was presented in [27]. By using code re-factoring, the PRedictable Execution Model (PREM) [23, 20] allows high-level co-scheduling of clusters of memory requests and CPU execution. PREM has been extended to multi-core systems in [36], and a similar model was adopted in [30, 33, 31] in the context of scratchpad-based platforms. The memory regulation technique (MemGuard) that we analyze belongs to a class of budget-based memory regulation techniques [39]. The key insight is that to each task is assigned a server, so that a fraction of the shared resource's available bandwidth is reserved. CPU resource sharing under periodic resource reservation was considered to derive hierarchical scheduling analysis [29, 8].

Our work has a number of similarities with [3], as both propose a way to estimate a task's WCET on multi-core platforms with shared DRAM subsystem. Three main differences however set the works apart. First, in [3] it is assumed that the exact behavior of a task is known in the form of a trace of executed instructions and memory references. Conversely we hereby assume that only the number of DRAM memory accesses is known, and then derive the arrangement of memory accesses and execution that leads to the worst-case. Second, [3] considers a non-arbitrated DRAM subsystem, while we explicitly account for the effect of memory bandwidth regulation. Third, unlike [3], we only focus on the problem of extending the WCET experimentally calculated in isolation (single-core case) to the multi-core case.

The work in [24] introduced a memory server for multicore systems that can provide better predictability by controlling at a fine granularity internal parameters of the DRAM subsystem. WCET analysis for the considered bandwidth regulation technique was originally proposed in the context of SCE [21, 28], under the assumption that equal budget is assigned to each core. Yao et al. in [37] proposed an analysis for MemGuard considering *uneven* bandwidth assignment. In [37, 24], however the analysis is carried out assuming that only the budget of the core under analysis is known, while the budgets assigned to other cores are not. Conversely, we relax this assumption and show that major improvements in terms of WCET calculation can be obtained.

3 Background about SCE Components

In this section, we briefly introduce few background concepts about each of the integrated resource management techniques comprising SCE: Colored Lockdown for shared cache management; MemGuard for DRAM bandwidth reservation; PALLOC for DRAM bank partitioning.

3.1 Colored Lockdown – Cache Assignment

SCE leverages Colored Lockdown [19] to mitigate inter-core interference at the cache level by allocating (locking) application memory areas in last-level cache. Colored Lockdown involves two main stages: an offline profiling stage; and an online cache allocation stage. During the offline stage, the task is analyzed to build a *profile*. The generated profile is effectively a list of memory pages ranked by access frequency. During the online stage, the most frequently accessed (hot) pages in the profile, up to a cutoff threshold, are locked in cache. By varying the cutoff threshold from 0 to the number of entries of the profile[1], it is possible to derive a *progressive lockdown curve* (PLC) [21]. The PLC plots the WCET of a task as function of the number of hot memory pages allocated in cache. In other words, if x is the number of profile pages to allocate for an application, the output of PLC(x) contains two pieces of information: (1) a corresponding value of WCET C for the task running in isolation (single-core scenario); and (2) a residual maximum number of cache misses μ, corresponding to accesses to all those profile pages not allocated in cache. As we show in Section 4, the C and the μ parameters obtained at this step represent the starting point to derive the value of WCET under memory bandwidth regulation with m active cores. Finally, note that the PLC could also be derived using static analysis tools.

3.2 MemGuard – Memory Bandwidth Partitioning

Similarly to shared caches, DRAM memory is one of the main sources of inter-core interference. To improve isolation, SCE uses MemGuard [40]. The goal of MemGuard is to provide bandwidth reservation on a per-core basis. MemGuard uses a series of per-core regulators that are responsible for monitoring and enforcing the memory bandwidth allocation. Each regulator monitors the amount of DRAM transactions performed by each core (or alternatively, the number of last-level cache misses) via hardware-specific performance monitoring capabilities. By considering the worst-case latency L_{max} for a single memory request to be serviced, it is possible to derive a worst-case (guaranteed) bandwidth at which the memory subsystem can operate. MemGuard operates as follows: it is configured to enforce the bandwidth assignment at a given period P. Based on L_{max}, it is possible to compute a **total budget**

[1] Or up to the maximum number of pages that can be locked in cache.

in terms of number of memory transactions that can be globally performed during P. This parameter, namely Q, can be computed as $Q = \frac{P}{L_{max}}$. From the global budget, **per-core budgets** Q_i can be assigned arbitrarily, as long as their total does not exceed Q_i. At the beginning of each period, MemGuard configures the hardware performance counters to trigger an event when any of the cores exceeds its Q_i threshold of completed DRAM memory requests. To enforce the strict bandwidth assignment, upon reception of a budget-exhausted event, MemGuard idles the associated core. Any idled core resumes its activity at the beginning of the next replenishment period P. The length of the regulation period P is a system-wide parameter and should be much smaller than the minimal application task period. In our current implementation, $P = 1$ ms, matching also the OS scheduler tick interval. MemGuard also offers different schemes to share reserved yet unused memory bandwidth across cores to achieve significant average-case performance improvements. In this paper, we are only concerned with strict bandwidth assignment, while additional details on bandwidth reclaiming and sharing mechanisms can be found in [40].

3.3 PALLOC – DRAM Bank Partitioning

The DRAM structure is organized into ranks, banks, rows and columns [13]. Whenever a given row is accessed in a bank, subsequent accesses on the same row (row-hits) can be serviced with a small latency. Conversely, if a subsequent access requires data in a different row (row-miss), a significant increase in the latency is introduced. Different banks of the same DRAM chip can satisfy requests in parallel [38, 15]. In a multi-core scenario, several cores can potentially access the same DRAM bank. In this case, (i) the row-miss ratio of a task can increase as multiple cores access the same bank; and (ii) requests originated by the core under analysis can be re-ordered after other cores' requests, introducing additional delay [15, 24]. To mitigate inter-core interference at the level of DRAM banks, *private banking* can be used. Under private banking, non overlapping sets of DRAM banks are assigned to different cores. SCE uses a DRAM bank-aware OS-level memory allocator, namely PALLOC [38], which allows system designers to assign specific DRAM banks to cores (or applications) and to enforce private banking. This way, tasks running in parallel do not collide on DRAM banks and do not suffer inter-core conflicts at this level, as long as there is a sufficient number of banks to accommodate them. A detailed discussion on how PALLOC works can be found in [38].

4 System Model and Assumptions

In this section, we discuss the considered system model as well as the assumptions under which the analysis is performed. Table 1 summarizes the list of parameters that will be used throughout the paper to calculate the WCET($\mathcal{Q}$) of a task in a system where m represents the number of active cores and $\mathcal{Q} = \{Q_1, \ldots, Q_m\}$ represent the budget assignment to the cores. The budgets in $\mathcal{Q}$ are **sorted** in ascending order. The parameter P represents the budget replenishment period of MemGuard such that the memory access budget for each core i will be restored to Q_i every P time units. Q_i represents the number of memory accesses that a given core i is allowed to perform within each MemGuard period of length P.

The parameter C captures the WCET of the considered task running in isolation once the last-level cache assignment for the task has been determined using Colored Lockdown. Under this scenario, the maximum value of residual cache misses μ can be obtained. The values of C and μ can be derived by using either static analysis or an experimental approach. Static analysis can be used whenever a micro-architectural model for the considered platform

Table 1 Summary of parameters for SCE response-time analysis.

Param.	**Interpretation**
m	Number of active cores in the system
P	MemGuard budget replenishment period
C	WCET for the considered task in isolation
μ	Number of residual misses after last-level cache assignment
E	Execution-only time in slots of length L_{max}
L_{min}	Minimum amount of time for a single memory request
L_{max}	Maximum amount of time for a single memory request
Q_i	Maximum number of memory requests for core i over P
Q	Maximum number of memory requests that can be globally performed in P

is available [7, 1]. On single-core systems, it is common industrial practice to experimentally derive the value of WCET. Hence, C could be derived by reusing the same practices on a multi-core platform by idling all but one cores. Existing tools that adopt this approach are part of the industry practice toward WCET determination on single-core platforms [34, 16].

Prefetchers, branch predictors, and speculative execution units are assumed to be disabled. Thanks to private banking, the DRAM requests from a core under analysis are never re-ordered after a group of requests from a different core. Each core is allowed to have more than one outstanding memory request, and we assume that the DRAM controller globally implements a round-robin[2] scheduling policy. In other words, read (write) memory requests reach the DRAM controller. Requests are then dispatched to DRAM banks, and their responses from the banks are forwarded on the bus in a round-robin scheme (single memory server). We assume that read and write requests are treated equally. We also assume that the maximum time to complete a memory transaction L_{max} is also the maximum delay introduced by other cores' individual requests. Note that this is not true in general, as the latter value can be significantly smaller than L_{max}. Albeit significant improvements can be obtained on the final WCET estimation, assuming a delay value different from L_{max} complicates the mathematical formulation without fundamentally impacting the general approach. For this reason, we leave this discussion as part of our future work. The best-case memory access latency is captured by the parameter L_{min}. It follows that the time to perform a single transaction is bounded between L_{min} and L_{max}.

It is important to distinguish between time spent in memory and time spent for pure execution on the CPU C^e. For in-order processors, where each memory request is blocking, there is no overlapping in time between memory and computation. Hence, a safe upper-bound on C^e can be computed as: $C^e = C - \mu \cdot L_{min}$. For out-of-order processors, the overlapping could range from 0 to $\mu \cdot L_{max}$. Hence $C^e = C$ is always a safe upper-bound, but it could be significantly improved using static analysis to reduce the pessimism on the amount of computation/memory overlapping. Since the granularity at which we conduct our analysis is L_{max}, we will often consider pure computation time (no memory) in slots of length L_{max}. The resulting slotted computation time E can be derived as $E = \left\lceil \frac{C^e}{L_{max}} \right\rceil$.

We do not address the problem of contention at the level of shared cache bus and controller. The cache bus is normally on-chip and features a much higher bandwidth

[2] Many modern COTS multi-core chips (e.g. Freescale MPC56xx and MPC57xx chip families) designed for safety-critical operations typically implement a round-robin policy on the main memory controller arbiter.

compared to the main memory bus. Contention at this level has been found to affect some modern platforms [10]. However, due to its high bandwidth capabilities, cache buses are normally able to sustain, without hitting the saturation point, the simultaneous activity of several CPUs. According to our benchmarks, on the Freescale P4080 platform used for our evaluations, contention at this level produces no visible effects up to 4 active cores and yields negligible slowdowns with 8 active cores.

Additionally, COTS architectures are often not time-composable between CPU pipeline and cache hierarchy [18]. This means that under certain circumstances, a local reduction of execution time (e.g. a cache hit) can produce a global increment in the execution time and vice versa, determining what is known as timing anomaly. If an experimental approach is used, the effect of timing anomalies is largely embedded in the experimentally derived WCET. This is because, thanks to Colored Lockdown, there is no variation in the sets of accesses that hit/miss in cache. Nonetheless, recent studies [11] suggest that the timing effect arising from timing anomalies can be statically analyzed and accounted at design time without significantly pessimistic overestimation.

In this work, we assume that per-core budgets Q_i are assigned at design-time to each core and are thus explicitly known. Moreover, we assume that the total bandwidth assignment does not exceed the guaranteed bandwidth. In other words, it must hold that:

$$\sum_{i=1}^{m} Q_i = \frac{P}{L_{max}} \; . \tag{1}$$

The analysis performed in the next sections will be done on each core i individually. We will indicate parameters that are core-specific with the i subscript (e.g. Q_i). To avoid overloading the notation, we will omit any index on per-task parameters (e.g. C, μ).

5 Worst-Case Derivation

In this section, we discuss how WCET derivation can be performed using the assumptions made in Section 4.

5.1 Memory/CPU Configurations

A distinctive aspect of our analysis is that we explicitly consider the effect that bandwidth regulation has on the behavior of the cores. For instance, suppose that our core under analysis (Core 2) has budget $Q_2 = 2$. Now consider a task running on core 2 that wants to perform 2 memory accesses back-to-back. Now, assume that there is only one more core in the system (Core 1), with budget $Q_1 = 1$. Thanks to the regulation mechanisms, Core 1 can interfere with only one of the two memory requests performed by Core 2, because Core 1 will be regulated after interfering once.

In order to study the worst-case execution time of tasks under analysis, it is fundamental to understand all the possible worst-case memory access patterns within a single MemGuard period P. Since we consider round-robin arbitration, the possible memory access patterns within a single regulation period can be derived combining (i) round-robin arbitration of resources from different cores; and (ii) the effect of regulation. In order to explain how this can be achieved, let us consider an example setup.

In our example, we consider $m = 4$ cores and a period of length $P = 2\ ms$. Assuming a value of $L_{max} = 0.2\ ms$, the number of memory transactions that can be performed at the guaranteed bandwidth within a regulation period is: $Q = \frac{P}{L_{max}} = 10$. Let us assume that

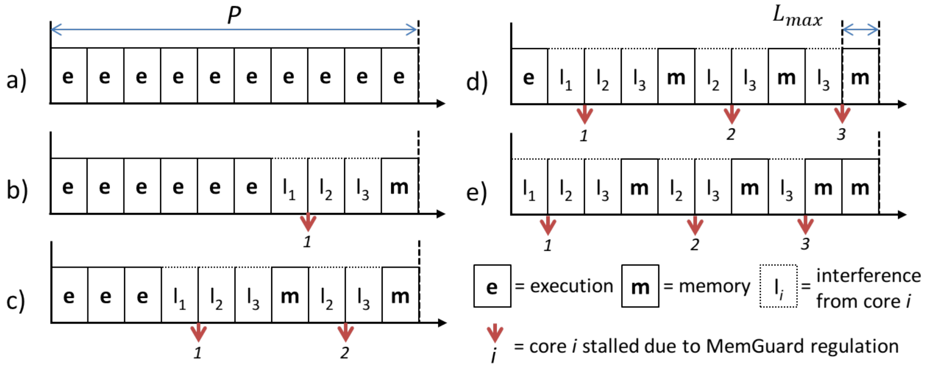
P
L_{max}
a) e e e e e e e e e e
b) e e e e e e I_1 I_2 I_3 m
1
c) e e e I_1 I_2 I_3 m I_2 I_3 m
1 2
d) e I_1 I_2 I_3 m I_2 I_3 m I_3 m
1 2 3
e) I_1 I_2 I_3 m I_2 I_3 m I_3 m m
1 2 3
e = execution
m = memory
I_i = interference from core *i*
i = core *i* stalled due to MemGuard regulation

Figure 1 Possible memory access patterns within a regulation period P for Core 4 with budget assignment $\mathcal{Q} = \{1, 2, 3, 4\}$.

the bandwidth assignment to cores is the following: $Q_1 = 1, Q_2 = 2, Q_3 = 3$ and $Q_4 = 4$. In order to visualize the possible patterns for Core $i = 4$, consider Figure 1.

In Figure 1a, a generic task running on core 4 only performs execution, hence, it suffers no interference from other cores. For this reason, the task will execute for an amount of time that corresponds to the length of 10 memory transactions of length L_{max}. In order to carry out our analysis, we consider time spent for execution (no memory) at a granularity that is useful for our calculations. For this reason, we consider execution as if it progresses in *slots* of length L_{max}. In this sense, the pattern in Figure 1a can be interpreted as if the task under analysis performs 10 *execution slots* of size L_{max}.

In Figure 1b, the task under analysis performs one memory transaction. Since memory transactions are satisfied following a round-robin policy, in the worst case the task will suffer interference on that single transaction from all the other cores. The corresponding pattern consists of 6 execution slots and a single memory transaction. From the point of view of Core 4 (core under analysis), The rest of the time in P is wasted due to the activity of other cores (contention).

In Figure 1c, the task under analysis performs 2 memory transactions. However, since $Q_1 = 1$, Core 1 will only interfere with either the first or the second transaction before being regulated by MemGuard. Hence, 3 cores will interfere on one memory transaction, but only 2 cores (Core 2 and 3) will interfere on the other memory transaction. Similarly, in Figure 1d, the same reasoning applies. Moreover, after the second memory transaction, Core 2 is regulated as well, leaving only Core 3 to interfere on the third transaction of the core under analysis. Finally, in Figure 1e, it can be seen that after three memory transactions, all the cores except the core under analysis are regulated, allowing an interference-free memory transaction (two consecutive m-slots in the figure). Each of the patterns in Figure 1 captures a different worst-case given that a task wants to perform a certain number of memory requests, say $M \in \{0, \ldots, Q_i\}$ within P. Three observations can be made to clarify why each pattern captures the worst-case, given that the number M of memory requests to be performed in a given regulation period is known:

1. each request takes L_{max};
2. unless the other cores are regulated, they always interfere with the memory transactions of the core under analysis;
3. the amount of execution performed by the core is the minimum under any scenario that allows M memory requests to be performed within P. Here, the minimum is considered because the final objective is to maximize the number of periods across which computation is performed.

Given a known budget assignment $\mathcal{Q}$ that meets the constraint in Equation 1, it is possible to derive all the possible (worst-case) patterns of memory and computation as in Figure 1. We refer to such set of possible patterns as memory/computation configurations, or simply as **M/C configurations**. A single M/C configuration can be expressed as a pair of the form $\langle M, C \rangle$, where M represents the number of performed memory transactions while C represents the number of executions slots performed within the considered regulation period.

The set Z_i of all the possible M/C configurations for a task running on a given core i can be constructed as follows. First, we define the *cumulative interference* operator $|Q|_h$. This operator considers a given bandwidth-to-core assignment of the form $\mathcal{Q} = \{Q_1, \ldots, Q_m\}$ and returns the number of cores whose bandwidth assignment is greater or equal than the threshold h. For instance, consider the example in Figure 1. In this case, $|Q|_1 = 4, |Q|_2 = 3, |Q|_3 = 2$ and $|Q|_4 = 1$. Next, we construct Z_i as follows:

$$Z_i = \bigcup_{h=0}^{Q_i-1} \{\langle M_h, C_h \rangle\} \cup \{\langle Q_i, 0 \rangle\}, \tag{2}$$

where $M_h = h$, $C_0 = Q$ and $C_h = Q - \sum_{j=1}^{h} |Q|_j$. Hence, the set Z_4 derived for the system presented in Figure 1 will be: $Z_4 = \{\langle 0, 10 \rangle, \langle 1, 6 \rangle, \langle 2, 3 \rangle, \langle 3, 1 \rangle, \langle 4, 0 \rangle\}$. Note that the quantity $Q - C_h - M_h = \sum_{j=1}^{h} |Q|_j - M_h$ captures the maximum interference suffered by a task on a given pattern with index h in the ordered set Z_i. Clearly, the trend of M_h is linear as one more memory access is considered as the index h increases. Conversely, the trend of C_h depends on the amount of interference suffered by the core and the regulation, as we discuss below.

5.2 Objective Formulation

Once the M/C configurations Z_i for a given core i are known, they apply to all the tasks scheduled on the considered core. Next, we reason on a single task's parameters to formulate the objective for the worst-case execution time calculation, i.e. WCET($\mathcal{Q}$). The work in [37] analyzed the WCET of a task under memory bandwidth regulation by calculating a *stall* term to be added to the task WCET obtained in isolation. In this paper, we adopt a different strategy. Instead of computing a *stall* term, we derive the maximum number of regulation periods required to complete a given task. The problem of deriving the worst-case execution time of a task WCET($\mathcal{Q}$) can be thought as the problem of finding the longest memory access pattern, given the parameters $\mathcal{Q}$, E, μ and the index of the considered core i.

From the way they are constructed, M/C configurations are in a discrete domain. **However, the problem of finding the maximum number of regulation periods in the discrete domain becomes a combinatorial problem. Instead, we reason in a continuous domain and use Theorem 4 to show that what obtained in the continuous domain represents an upper-bound on the value of WCET($\mathcal{Q}$)**. In other words, by only reasoning on **continuous** M/C curves, we show how to calculate an upper bound on the maximum number of regulation periods through which μ memory requests and E slots of computation can span.

▶ **Definition 1.** We define a sequence of regulation periods as *memory access pattern.* A memory access pattern, spanning through L regulation periods, is structured in the following way: the first $L - 1$ periods consume memory and computation according to some $\langle M_h, C_h \rangle \in Z_i$. Each of the $L - 1$ periods can use a different $\langle M_h, C_h \rangle$ element. During the L-th period, any leftover computation/memory is performed. In fact, for the last period we

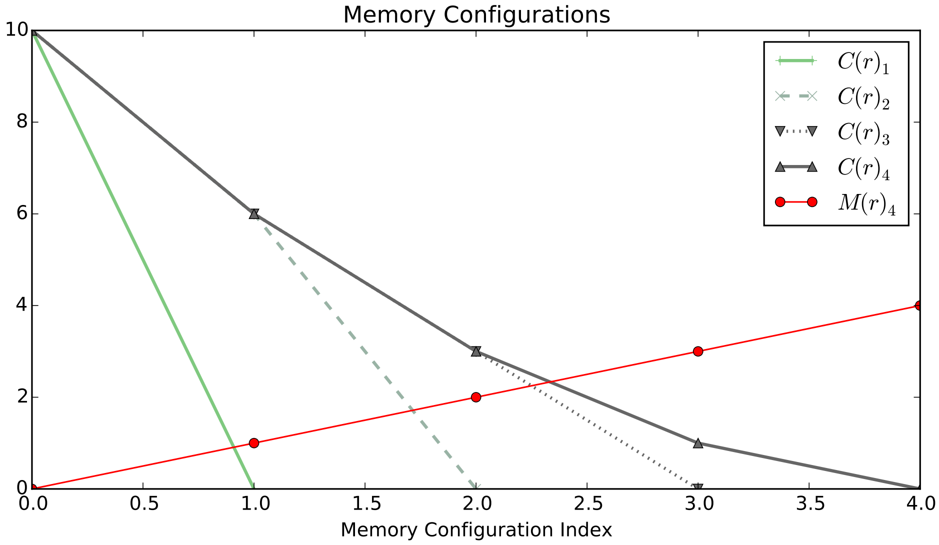

Figure 2 Trend of $M(r) = r$ and $C(r)$. Note the convex trend of the connecting line between the C_h components for $C(r)_4$.

can consider any element in Z_i with some C_h and M_h components that are both larger than the computation and memory leftover, respectively.

▶ **Definition 2.** We define the length of a memory access pattern as the number L of regulation periods that compose the pattern.

The following transformation is used to convert the problem from the discrete domain into the continuous domain. Specifically, we consider each component of each element of Z_i as a point along a *computation consumption curve* $C(r)$ and a *memory consumption curve* $M(r)$ for C_h and M_h components, respectively, with $r \in \mathbb{R}$. For both the curves, the domain depends on the MemGuard budget assignment for the core under analysis. Specifically, $r \in [0, Q_i]$. Since the memory consumption trend is always linear, $M(r)$ can be simply defined as the connecting line among M_h components, or simply $M(r) = r$. Conversely, $C(r)$ is defined as follows:

$$C(r) = \begin{cases} C_r & \text{if } \langle M_r, C_r \rangle \in Z_i \\ C_h + (C_{h+1} - C_h)(r - h) & \text{if } C_h < r < C_{h+1} : \langle M_h, C_h \rangle, \langle M_{h+1}, C_{h+1} \rangle \in Z_i \end{cases} \tag{3}$$

In other words, $C(r)$ is defined as the connecting line between the components C_h in Z_i, as depicted in Figure 2. In the figure, we consider $m = 4$ cores and a fixed budget assignment $\mathcal{Q} = \{1, 2, 3, 4\}$, and show the resulting $C(r)$ curves for each core, labeled as $C(r)_1, \ldots, C(r)_4$. In the figure, only $M(r)_4$ is depicted, since it is similar on all the cores and only varies in the length of its domain.

Consider $C(r)_4$ and $M(r)_4$ depicted in Figure 2. In this example, the two functions are piece-wise continuous and **convex curves**. Note that $M(r)_i$ always exhibits a linear trend, hence it is always convex. $C(r)_i$ has a convex trend when we consider the core(s) i with the highest budget assignment in $\mathcal{Q}$. Highest-budget cores will never suffer regulation. This sufficient condition can be simply expressed as $Q_i = \max\{\mathcal{Q}\}$. If this condition is not satisfied, $C(r)$ could be still convex. It is the case of $C(r)_3$ in Figure 2. Nonetheless, in general, this property cannot be assumed if $Q_i \neq \max\{\mathcal{Q}\}$. $C(r)_2$ in Figure 2 falls in the latter case. If both functions are convex, it means that in a single period, a linear increase in execution slots results in a more-than-linear decrease in memory requests, and vice-versa. We first discuss how an upper-bound on the worst-case execution time can be derived in case of convexity. Next, we discuss how non-convexity can be handled in Section 5.4.

5.3 Max Length for Memory Access Pattern (convex case)

We now focus on identifying the worst-case access pattern under the hypothesis of convex M/C configurations. Let us now introduce the quantity $P(r)$ as the upper-bound on the length L of the memory access pattern when performing $C(r)$ slots of computation and $M(r)$ memory requests per period. $P(r, E, \mu)$ is defined according to Lemma 3. Note that $P(r, E, \mu)$ depends on three parameters. Since E and μ are constant for a given task, we abuse the notation and only denote this function (and successive derivations) as $P(r)$.

▶ **Lemma 3.** *Consider a point $r \in [0, Q_i]$ in the domain of continuous M/C curves. When $C(r)$ slots of computation and $M(r)$ memory transactions are performed during each regulation period, the resulting length of the memory access pattern is $P(r)$:*

$$P(r) = \begin{cases} P^e(r) & \text{if } \frac{E}{C(r)} < \frac{\mu}{M(r)} \\ P^m(r) & \text{if } \frac{E}{C(r)} > \frac{\mu}{M(r)} \\ \max\{P^e(r), P^m(r)\} & \text{if } \frac{E}{C(r)} = \frac{\mu}{M(r)} \end{cases} \tag{4}$$

where $P^e(r)$ and $P^m(r)$ are defined as follows:

$$P^e(r) = \left\lceil \frac{E}{C(r)} \right\rceil + \left\lceil \frac{\mu - M(r)\lceil \frac{E}{C(r)} \rceil}{Q_i} \right\rceil^+ \tag{5a}$$

$$P^m(r) = \left\lceil \frac{\mu}{M(r)} \right\rceil + \left\lceil \frac{E - C(r)\lceil \frac{\mu}{M(r)} \rceil}{Q} \right\rceil^+ \tag{5b}$$

and the operator $\lceil f(x) \rceil^+ = \max\{\lceil f(x) \rceil, 0\}$.

Proof. Let us first focus on the condition between the first two cases. $E/C(r)$ represents the (decimal) number of regulation periods to complete the amount of computation slots E. Similarly, $\mu/M(r)$ expresses the number of regulation periods to perform all μ memory transactions. The first case in Equation 4 represents the case in which E computation slots are completed before μ memory transactions are performed. The second case captures the opposite case.

Case $E/C(r) < \mu/M(r)$: In this case, execution is completed within the first $P_{init} = \lceil E/C(r) \rceil$ regulation periods. After that point, only memory requests are left to be performed. However, *some* memory requests have been performed in the initial P_{init} periods. This amount is at most $M(r) \cdot P_{init}$. If the quantity $\mu - M(r) \cdot P_{init}$ is positive, then this represents the amount of memory transactions that are left to be performed. Since execution slots have been completed, the memory transactions after the first P_{init} periods will be performed back-to-back. Since core i under analysis is subject to regulation with budget Q_i, it will take at least $P_{left} = \left\lceil \frac{\mu - M(r) \cdot P_{init}}{Q_i} \right\rceil$ to complete the leftover memory requests (if any) with budget Q_i. Equation 5a is obtained by summing $P_{init} + P_{left}$.

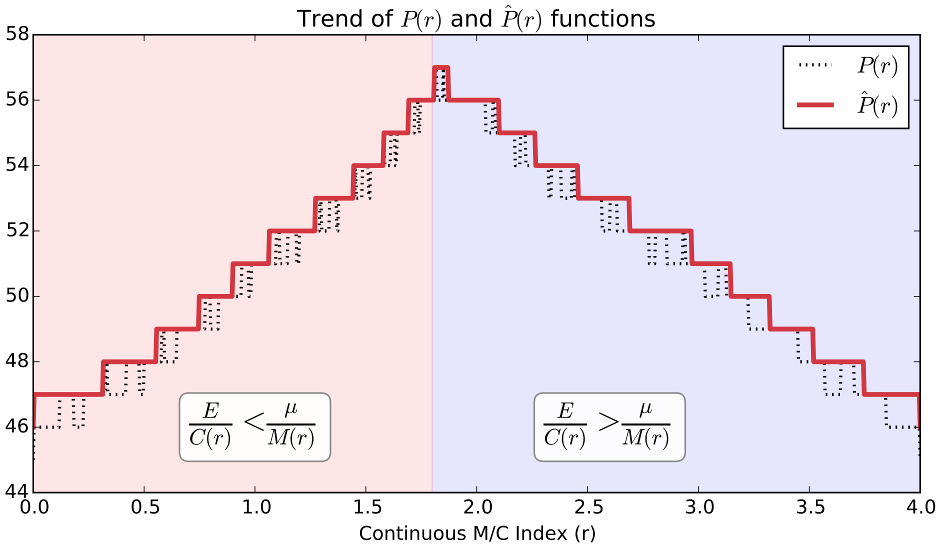

Figure 3 Trend of $P(r)$, $\hat{P}(r)$ for a system with $m = 4$, $\mathcal{Q} = \{1, 2, 3, 4\}$, and a task on core $i = 4$ with parameters $E = 200, \mu = 100$.

Case $E/C(r) > \mu/M(r)$: This case is analogous to the first case, by swapping the role of memory and computation. In fact, in this case $P_{init} = \lceil \mu/M(r) \rceil$ is the number of periods to entirely perform μ memory requests at a per-period rate of $M(r)$. The leftover computation is $E - C(r) \cdot P_{init}$, if this quantity is positive. Once memory has been entirely performed, the remaining regulation periods are entirely filled with computation since no regulation is suffered. Hence, Q execution slots per period will be performed after P_{init}. The leftover (if any) computation is completed in $P_{left} = \left\lceil \frac{E-C(r) \cdot P_{init}}{Q} \right\rceil$. The sum of $P_{init} + P_{left}$ provides the length of the memory access pattern in this case and is equivalent to Equation 5b.

Case $E/C(r) = \mu/M(r)$: Finally, in this point, the value of $P(r)$ is simply the maximum between Equation 5a and 5b. ◀

To find an upper-bound on WCET($\mathcal{Q}$), we first find the value of r that maximizes $P(r)$. Since $C(r)$ is defined as a piece-wise linear curve, an easy way to reason independently on each segment and find the value of r on each segment that maximizes the function. In order to find the maximum value of $P(r)$, we reason on each of the individual segments of the $C(r)$ curve individually. In practice, this can be significantly optimized by considering that the changes of slope in $C(r)$ is less than or equal to $i - 1$. Due to space constraints, we do not discuss possible optimization. Consider an arbitrary (integer) value of $h \in \{0, \dots, Q_i - 1\}$. Clearly $M(r) = r$. We can also write $C(r)$ in the domain $r \in [h, h + 1]$ as:

$$C(r) = C_h + (C_{h+1} - C_h) \cdot (r - h) = \beta r + \gamma, \tag{6}$$

where $\gamma = \alpha - h\beta$, $\alpha = C_h$, and $\beta = (C_{h+1} - C_h)$. Note that α, β and γ are all constant in the considered segment of $C(r)$. Furthermore, note that $\gamma \geq 0$ and $\beta \leq 0$.

Let us focus on the case where the second term of $P^e(r)$ and $P^m(r)$ is non-zero. By expanding Equation 4 in the considered segment, we can rewrite the two terms of $P(r)$, i.e. $P^e(r)$ and $P^m(r)$, locally as[3]:

$$P^e(r) = \left\lceil \left\lceil \frac{E}{\beta r + \gamma} \right\rceil \left(1 - \frac{r}{Q_i}\right) + \frac{\mu}{Q_i} \right\rceil \tag{7a}$$

$$P^m(r) = \left\lceil \left\lceil \frac{\mu}{r} \right\rceil \left(1 - \frac{\beta r + \gamma}{Q}\right) + \frac{E}{Q} \right\rceil \tag{7b}$$

Furthermore, since it always holds that $\lceil f(x) \rceil \leq f(x) + 1$, it is possible to upper-bound each term in $P(r)$ and remove the inner ceiling by considering $\hat{P^e}(r)$ and $\hat{P^m}(r)$ defined as follows:

$$\hat{P^e}(r) = \begin{cases} \left\lceil \frac{E}{\beta r + \gamma} \right\rceil & \text{if } \left\lceil \frac{E}{C(r)} \right\rceil > \frac{\mu}{M(r)} \\ \left\lceil \left(\frac{E}{\beta r + \gamma} + 1\right)\left(1 - \frac{r}{Q_i}\right) + \frac{\mu}{Q_i} \right\rceil & \text{otherwise} \end{cases} \tag{8a}$$

$$\hat{P^m}(r) = \begin{cases} \left\lceil \frac{\mu}{r} \right\rceil & \text{if } \left\lceil \frac{\mu}{M(r)} \right\rceil > \frac{E}{C(r)} \\ \left\lceil \left(\frac{\mu}{r} + 1\right)\left(1 - \frac{\beta r + \gamma}{Q}\right) + \frac{E}{Q} \right\rceil & \text{otherwise} \end{cases} \tag{8b}$$

Note that Equation 8a (resp., Equation 8b) has two cases. The first case is used when the second term of Equation 5a (resp., Equation 5b) is zero; otherwise, the second case is used with the simplification performed in Equation 7a (resp., Equation 7b) and following upper-bounding. By using $\hat{P^e}(r)$ and $\hat{P^m}(r)$, we define $\hat{P}(r) \geq P(r)$ as:

$$\hat{P}(r) = \begin{cases} \hat{P^e}(r) & \text{if } \frac{E}{\beta r + \gamma} < \frac{\mu}{r} \\ \hat{P^m}(r) & \text{if } \frac{E}{\beta r + \gamma} > \frac{\mu}{r} \\ \max\{\hat{P^e}(r), \hat{P^m}(r)\} & \text{if } \frac{E}{\beta r + \gamma} = \frac{\mu}{r} \end{cases} \tag{9}$$

Note that $\hat{P}(r)$ is a function with one variable (r). Moreover, the value r^* that maximizes $\hat{P}(r)$ can be found by reasoning without the outer ceiling in $\hat{P^e}(r)$ and $\hat{P^m}(r)$. Within each segment, it is possible to find its critical points using the first-derivative test. The first two terms of Equation 9 are used according to a condition on r that can be rewritten as:

$$\frac{E}{\beta r + \gamma} < \frac{\mu}{r} \implies r < \frac{\mu\gamma}{E - \mu\beta}. \tag{10}$$

The value r_{sw} where the condition in Equation 10 changes, represents a first critical point:

$$r_{sw} = \frac{\mu\gamma}{E - \mu\beta}. \tag{11}$$

Whenever Equation 10 is satisfied, the first term of Equation 9 is considered; otherwise the second term of the equation is used. Let us reason on these two terms of $\hat{P}(r)$ separately. For the first term, the values of r that constitute critical points are:

$$r_{1,1} = -\frac{\sqrt{-E\beta Q_i - E\gamma} + \gamma}{\beta}; \quad r_{1,2} = \frac{\sqrt{-E\beta Q_i - E\gamma} - \gamma}{\beta}. \tag{12}$$

[3] We rely on the property that $\lceil \lceil x \rceil + y \rceil = \lceil x \rceil + \lceil y \rceil$, with $x, y \geq 0$.

Note that $r_{1,1}$ and $r_{1,2}$ are real numbers only when $E\gamma \leq -E\beta Q_i$. The functions $\hat{P^e}(r)$ and $\hat{P^m}(r)$ have a point where the derivative may not exist in r_2 and r_3, respectively:

$$r_2 = -\frac{\gamma}{\beta}; \quad r_3 = \sqrt{\frac{\mu(Q-\gamma)}{\beta}}, \tag{13}$$

under the condition that $\gamma \geq Q$. Finally, the function has a point where the derivative may not exist in $r_4 = 0$.

Recall that $\hat{P}(r)$ is defined over the closed interval $[h, h+1]$, and it has a different expression as the selected segment $h \in \{0, \ldots, Q_i - 1\}$ changes. The boundaries of the interval constitute additional test-points for the maximum. It follows that, for a given segment h, the maximum L^*_h of $P(r)$ can be found as:

$$L^*_h = \max_{r \in \{h, h+1, r_{sw}, r_{1,1}, r_{1,2}, r_2, r_3, r_4\}} \{\hat{P}(r)\}. \tag{14}$$

Clearly, the points $r_{1,1}$, $r_{1,2}$ and r_2 do not need to be evaluated if they do not satisfy Equation 10, or they lie outside the interval $(h, h+1)$; similarly, r_3 and r_4 do not need to be tested if they satisfy Equation 10, or they lie outside the range $(h, h+1)$. Moreover, $r_{1,1}$, $r_{1,2}$ and/or r_3 need to be removed from the set of test-points if they are not real numbers. Conversely, r_{sw} is always evaluated. Finally, an upper-bound on the global maximum L^* of $P(r)$ can be found as follows:

$$L^* = \max_{h \in \{0, \ldots, Q_i - 1\}} \{L^*_h\}. \tag{15}$$

Equation 15 not only provides an upper-bound on the length of the worst-case memory access pattern in the continuous case; but also the rate r^* such that the worst-case memory access pattern can be constructed by a sequence of identical L^* regulation periods, during which $C(r^*)$ (resp., $M(r^*)$) units of computation (resp., memory requests) are performed. Theorem 4 allows us to use the obtained result in case of memory access patterns where each regulation period consumes resources according to discrete M/C configurations.

▶ **Theorem 4.** *Consider a task τ that performs E units of computation and μ memory transactions. Consider the maximum length L^*_d of any memory access pattern of τ constructed using discrete and convex M/C configurations. An upper bound on L^*_d is given by the maximum length L^*_c of a pattern computed using $P(r)$ (see Equation (4)) for a task τ' that performs $E + Q$ units of computation and $\mu + Q_i$ memory transactions, with continuous and convex M/C configuration.*

Proof. The theorem follows from the fact that $C(r)$ and $M(r)$ are convex curves. Let us assume that the longest memory access pattern for τ constructed using a sequence of discrete M/C configurations has length L^*_d. Consider the structure of a task that performs E units of computation and μ memory transactions. It is a sequence of L^*_d regulation periods. Let us use a_h (non-negative integers) to count how many times the element $\langle C_h, M_h \rangle \in Z_i$ appears in the sequence. We have this relation:

$$E \leq E_{tot} = a_0 C_0 + \ldots + a_{Q_i} C_{Q_i} \leq E + Q \tag{16a}$$

$$\mu \leq M_{tot} = a_0 M_0 + \ldots + a_{Q_i} M_{Q_i} \leq \mu + Q_i \tag{16b}$$

where $a_0 \ldots a_{Q_i}$ are integer coefficients that represent how many times a certain (discrete) M/C configuration appears in the pattern. Recall that, by Definition 1, any M/C configuration can be considered in the last regulation period, as long as the leftovers in computation and memory are both equal or smaller than the considered last-period configuration. This choice does not violate the conditions $E_{tot} \leq E + Q$ and $M_{tot} \leq \mu + Q_i$. It follows that:

$$\sum_{i=0}^{Q_i} a_i = L_d^*. \tag{17}$$

Consider a transformation that translates the non-repeating pattern composed by a sequence of discrete M/C configurations into a repeating pattern with continuous M/C configurations. In the transformed pattern, in each period we perform $C(r_c)$ and $M(r_c)$ units of computation and memory respectively, where r_c is selected as follows:

$$r_c = \frac{\sum_{i=0}^{Q_i} a_i \cdot i}{L_d^*}. \tag{18}$$

Consider the amount of computation performed in each period:

$$C\left(\frac{\sum_{i=0}^{Q_i} a_i \cdot i}{L_d^*}\right) = C\left(\sum_{i=0}^{Q_i} \frac{a_i \cdot i}{\sum_{i=0}^{Q_i} a_i}\right). \tag{19}$$

Recall that the definition of a convex function $f(x)$ imposes that:

$$\lambda_i \in \mathbb{N} \text{ s.t. } \sum_i \lambda_i = 1 \implies f\left(\sum_i \lambda_i x_i\right) \leq \sum_i \lambda_i f(x_i). \tag{20}$$

Further, note that:

$$\sum_{i=0}^{Q_i} \frac{a_i}{\sum_{i=0}^{Q_i} a_i} = \frac{a_0}{a_0 + \ldots + a_{Q_i}} + \ldots + \frac{a_{Q_i}}{a_0 + \ldots + a_{Q_i}} = 1. \tag{21}$$

Since $C(r)$ is convex, it holds that:

$$C\left(\sum_{i=0}^{Q_i} \frac{a_i \cdot i}{\sum_{i=0}^{Q_i} a_i}\right) \leq \sum_{i=0}^{Q_i} \frac{a_i \cdot C(i)}{\sum_{i=0}^{Q_i} a_i} \implies C(r_c) \leq \frac{E_{tot}}{L_d^*}. \tag{22}$$

From this result, it follows that:

$$L_d^* \leq \frac{E_{tot}}{C(r_c)} \leq \frac{E + Q}{C(r_c)}, \tag{23}$$

and, by repeating the convexity considerations, that:

$$L_d^* \leq \frac{M_{tot}}{M(r_c)} \leq \frac{\mu + Q_i}{M(r_c)}. \tag{24}$$

Recall that $\hat{P}(r)$ is actually defined with three parameters: $\hat{P}(r, E, \mu)$, and that $\hat{P}(r, E, \mu)$ can only increase when both the values of E and μ are increased. Take r^* as the value that maximizes $\hat{P}(r^*, E + Q, \mu + Q_i)$, it must hold that:

$$L_c^* = \hat{P}(r^*, E + Q, \mu + Q_i) \geq \hat{P}(r_c, E + Q, \mu + Q_i) \tag{25}$$

Moreover, it follows directly from from Equations 5a and 5b that:

$$\hat{P}(r_c, E + Q, \mu + Q_i) \geq \frac{E + Q}{C(r_c)} \text{ if } \frac{E + Q}{C(r_c)} < \frac{\mu + Q_i}{M(r_c)} \tag{26a}$$

$$\hat{P}(r_c, E + Q, \mu + Q_i) \geq \frac{\mu + Q_i}{M(r_c)} \text{ if } \frac{E + Q}{C(r_c)} > \frac{\mu + Q_i}{M(r_c)} \tag{26b}$$

In case of equality $(E+Q)/C(r_c) = (\mu+Q_i)/M(r_c)$, Equations 26a and 26 are both satisfied. Due to what expressed in Equations 23-26, it follows that $L_c^* \geq L_d^*$, which proves the theorem. ◀

The following corollary can be used to formalize the calculation of WCET($\mathcal{Q}$) based on Theorem 4.

▶ **Corollary 5.** *Consider a task τ that performs E slots of computations and μ memory transactions on core i. Then, an upper-bound on WCET($\mathcal{Q}$) can be computed as:*

$$WCET(\mathcal{Q}) = P \cdot \hat{P}(r^*, E+Q, \mu+Q_i), \tag{27}$$

i.e., by finding the value r^* that maximizes $\hat{P}(r, E+Q, \mu+Q_i)$.

Proof. The proof simply follows from Theorem 4. Note that WCET($\mathcal{Q}$) is expressed in time by multiplying the maximum number of regulation periods $\hat{P}(r^*, E+Q, \mu+Q_i)$ by the length of each period P. ◀

5.4 Max Length for Memory Access Pattern (non-convex case)

Whenever the core under analysis is not one of the cores with the highest memory budget assignment, i.e. when $Q_i \neq \max\{\mathcal{Q}\}$, the curve $C(r)$ may not be convex, but it will be convex until the (Q_i-1)-th component in Z_i.

In order to upper-bound the WCET($\mathcal{Q}$) in this case, we use the following observation. Suppose that the length of the worst-case (longest) memory access pattern is L^*. In this pattern, $L^* - k$ are periods without regulation, where, in each period, computation and memory is consumed according to an element in $Z_i \setminus \langle 0, Q_i \rangle$. The remaining k periods are regulation-only periods, where 0 computation slots are performed and Q_i memory transactions are performed, with k varying between 0 and $\lfloor \frac{\mu}{Q_i} \rfloor$.

Following this observation, an upper-bound on WCET($\mathcal{Q}$) can be calculated as:

$$\max_{k \in [0, \lfloor \frac{\mu}{Q_i} \rfloor]} \{\hat{P}(r_k^*, E+Q, \mu+Q_i-k\cdot Q_i)+k\} = \max_{k \in [0, \lfloor \frac{\mu}{Q_i} \rfloor]} \{\hat{P}(r_k^*, E+Q, \mu+Q_i(1-k))+k\}, \tag{28}$$

where r_k^* is the value of the variable r that maximizes $\hat{P}(r)$ when the amount of computation slots and memory requests to perform is set to $E+Q$ and $\mu+Q_i(1-k)$, respectively.

6 Schedulability Analysis

Once an upper-bound on WCET($\mathcal{Q}$) is derived according to what described in Section 5, it is possible to compute the schedulability of a task-set under fixed-priority scheduling and SCE resource partitioning. First, let us make explicit the parameters with which WCET($\mathcal{Q}$) is invoked, and use the notation WCET($\mathcal{Q}$, C, μ) to indicate an upper bound on the time to perform C execution time units and μ memory transactions under the budget assignment $\mathcal{Q}$. For the purpose of the analysis, we consider a set of implicit-deadline tasks $\tau_1, \ldots, \tau_n$, where each task τ_k is characterized by a period T_k, a WCET in isolation C_k, and a maximum number of residual last-level cache misses after lockdown μ_k. We assume that the scheduling policy is based on tasks' fixed priority assignment (e.g. Rate Monotonic) upon a partitioned multi-core system; this is a common practice used in industrial applications. Next, we consider a level-k busy interval.

We consider two cases: (i) all the tasks' releases and deadlines are aligned with system's tick (*inbound case*); and (ii) tasks' releases and deadlines may not be aligned with system's tick (*outbound case*). In general, since MemGuard regulation period is synchronized with system's tick, it is a good design practice to have inbound tasks. This has two main benefits: first, a tasks is never released when the MemGuard budget has been exhausted by a previously running task; second, it is possible to derive the response time of tasks by individually considering their WCET = WCET($\mathcal{Q}$, C_k, μ_k). This is because tasks are never preempted in the middle of a regulation period.

6.1 Analysis of Inbound Tasks

In order to check the schedulability of inbound tasks, it is enough to reuse typical response-time analysis. Specifically, for a task under analysis τ_k, it is possible to derive its response-time by finding the first h such that $R_k^{(h+1)} = R_k^{(h)}$ or such that $R_k^{(h)} > T_k$, given that:

$$R_k^{(0)} = \mathrm{WCET}(\mathcal{Q}, C_k, \mu_k) \tag{29a}$$

$$R_k^{(h+1)} = \mathrm{WCET}(\mathcal{Q}, C_k, \mu_k) + \sum_{\tau_j \in hp(\tau_k)} \left\lceil \frac{R_k^{(h)}}{T_j} \right\rceil \cdot \mathrm{WCET}(\mathcal{Q}, C_k, \mu_k), \tag{29b}$$

where $hp(\tau_k)$ represents the set of tasks with priority higher than the task under analysis τ_k. Intuitively, if there exist a h such that $R_k^{(h+1)} = R_k^{(h)}$ and $R_k^{(h)} \leq T_k$, the considered task is schedulable and it is non-schedulable otherwise.

6.2 Analysis of Outbound Tasks

The analysis of outbound tasks is slightly more complicated because tasks can not only be activated in the middle of a regulation period, but also preempted within a regulation period. This, in turn, makes the behavior of a task dependent upon other tasks' memory access patterns. For this reason, instead of reasoning on tasks as separate entities, it is easier to consider the overall worst-case memory access pattern of a level-k busy interval as a whole. When we consider a set of outbound tasks, we can effectively merge the execution (and memory accesses) of an instance of the task under analysis with all the interfering instances of higher priority tasks. In this case, $R_k^{(0)}$ and $R_k^{(h+1)}$ can be calculated as follows:

$$R_k^{(0)} = \mathrm{WCET}(\mathcal{Q}, C_k, \mu_k) + B_i. \tag{30a}$$

$$R_k^{(h+1)} = \mathrm{WCET}(\mathcal{Q}, C_{hep(\tau_k)}, \mu_{hep(\tau_k)}) + B_i, \tag{30b}$$

where $hep(\tau_k)$ denotes the set of tasks with priority greater than or equal to task τ_k. The operators $C_{hep(\tau_k)}$ and $\mu_{hep(\tau_k)}$ are defined as follows:

$$C_{hep(\tau_k)} = \sum_{\tau_j \in hep(\tau_k)} \left\lceil \frac{R_k^{(h)}}{T_j} \right\rceil \cdot C_j, \tag{31a}$$

$$\mu_{hep(\tau_k)} = \sum_{\tau_j \in hep(\tau_k)} \left\lceil \frac{R_k^{(h)}}{T_j} \right\rceil \cdot \mu_j. \tag{31b}$$

Finally, the term B_i is a blocking term that represents the maximum amount of time that a task must wait if it is activated immediately after the MemGuard budget has been exhausted by a previously running task. Since core i is stalled once it has consumed its allocated budget Q_i. The term B_i can be calculated as: $B_i = P - Q_i \cdot L_{min}$.

7 Evaluation

In this section, we evaluate the pessimism of the derived WCET($\mathcal{Q}$) bound compared to the exact worst-case memory access pattern found in the discrete case. We also compare the quality of the derived bound with respect to the analysis in [37].

7.1 Budget generation

A key component of our evaluation is the distribution of budgets to cores. The proposed analysis, in fact, exploits the full knowledge about the assigned budgets to account for regulation of interfering cores. In fact, if even budgets are assigned to all the cores, no benefits are observed compared to the method proposed in [21] and [37]. Thus, we introduce a metric of variance of the budget assignment, namely δ. Consider a total budget Q. The distribution of Q to cores follows a linear trend where δ is the slope of the increase. The total budget assigned to all the cores, however, is always less than or equal to Q. For instance, consider a system with $m = 8$ cores and a value of $Q = 100$. When $\delta = 0$, the assignment $\mathcal{Q}_{\delta=0} = \{12, 12, 12, 12, 13, 13, 13, 13\}$. Conversely, for $\delta = 0.035$ we have $\mathcal{Q}_{\delta=0.035} = \{1, 4, 8, 11, 14, 17, 21, 24\}$.

7.2 Analysis of Pessimism

To compare the discussed bound with the theoretical optimum, we have implemented a back-tracking algorithm that uses a brute-force approach to explore all the possible memory access patterns and to find the maximum. Clearly, the algorithm has a large complexity and its runtime explodes quickly with realistic system parameters. For this reason, we compare the proposed analysis and the exact (brute-force) algorithm on system instances with small parameters. We consider a system with $m = 8$ cores, with $L_{max} = 0.1$ ms, $P = 10$ ms, such that $Q = 100$. We inspect the system with $\delta \in [0, 0.035]$, with randomly-distributed values of $E \in [1, 110]$ and $\mu \in [1, 110]$. Each sample consists of 100 different task parameters. Each of the 100 randomly generated tasks is evaluated on all the 8 cores.

Figure 4 depicts the relative overestimation compared to the exact case in the considered δ range. Given a core and a value of δ, the figure reports the average of the overestimation percentage between the exact derivation and the proposed analysis. To increase the readability of the plot, we omit cores 3,4 and 6 that exhibit intermediate trends. Three characteristics can be noted. First, for low values of δ, tasks behave similarly on all the cores, since budget is evenly distributed. Second, the overestimation is high, starting at 80% on the left of the figure. This, however, is an artificial effect because the randomly generated tasks span across few regulation periods. Unfortunately, performing an evaluation of the brute-force algorithm on larger task parameters becomes computationally unfeasible. Third, as we move toward the right of the figure, there is a sharp drop in overestimation for the low-budget cores. This is because, as less budget is assigned, the length of the task increases; hence the overestimation, which remains constant, has a proportionally smaller weight on the task length.

To validate the last claim, i.e. that the overestimation added by our analysis is roughly constant compared to the exact case, consider Figure 5. In this figure, we use the same tasks generated to plot Figure 4. For each core, the bar at 0 reports the number of tasks with the exact same length in both the brute-force algorithm and our analysis; the bar at 1 counts the number of tasks that are longer by 1 regulation period compared to the exact case; and so on. It can be noted that in all the cases, the overestimation does not exceed 5 regulation periods. Moreover, especially, when the cores with intermediate budgets are considered, tasks are mostly overestimated only by 3 regulation periods.

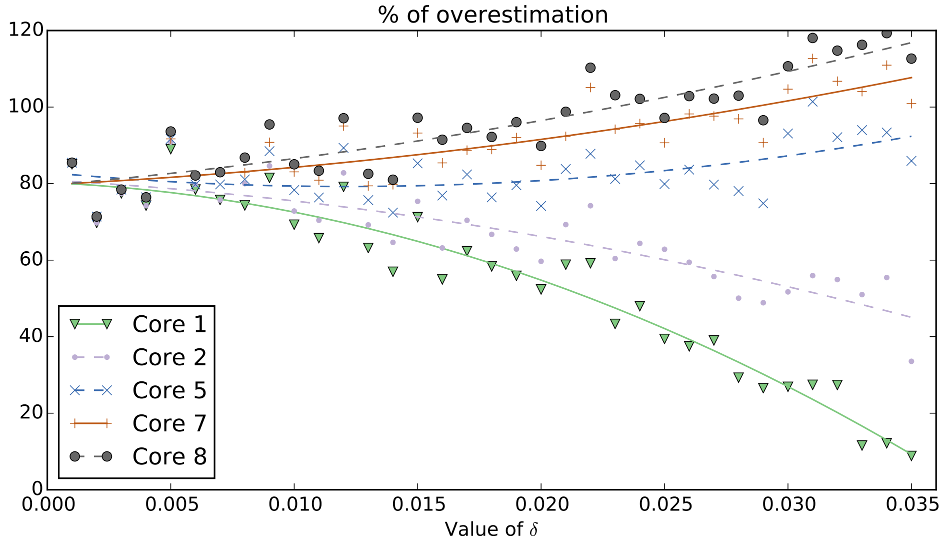

Figure 4 Relative (%) overestimation compared to exact (brute-force) derivation on cores 1, 2, 7 and 8 as a function of δ. Higher y-values correspond to larger overestimation.

7.3 Performance Comparison

In order to understand the benefit of explicitly considering the bandwidth regulation on all the cores, we compare the proposed approach against the analysis in [37]. Both ours and the analysis in [37] share many fundamental assumptions about the system model. The main difference is that the analysis proposed in this work explicitly considers the budget assignment $\mathcal{Q}$ to all the cores. Without this extra piece of information, the analysis in [37] is necessarily more pessimistic, as the worst-case corresponds to the case where the remaining budget is evenly distributed among the remaining cores.

The goal of this experiment is to quantify the gain in terms of WCET derivation. Since the complexity of our analysis is pseudo-polynomial with respect to budgets and task parameters, in this experiment we use system and task parameters that are reflective of a realistic use-case. These parameters are aligned with the evaluation conducted in [21]. We consider $m = 8$, $L_{max} = 4.96 \times 10^{-8}$ s, $P = 1$ ms, such that $Q = 20161$. We inspect the system with $\delta \in [1, 7]$, with randomly-distributed values of $E \in [1, 300000]$ and $\mu \in [1, 200000]$. Once again, each sample consists of 100 different task parameters. Each of the 100 randomly generated tasks is evaluated on all the 8 cores.

The results for this experiment are reported in Figure 6. Three main aspects are relevant to mention. First, for highly differentiated budget assignments (larger values of δ), the proposed algorithm outperforms the analysis in [37], with a reduction of about 30% (left – average), and up to 60% (right – max) in the overestimation of the task's WCET. Second, for lower values of δ, the two algorithms behave almost identically. Third, a slight performance degradation (around 1%) can be observed for high values of δ and low-budget cores. This arises from the fact that in order to upper-bound the WCET, an additional Q and Q_i units of computation and memory, respectively, are added to the task (see Corollary 5). We also believe that different budget assignment schemes, e.g. with exponential increase as opposed to linear, may significantly affect the analysis performances.

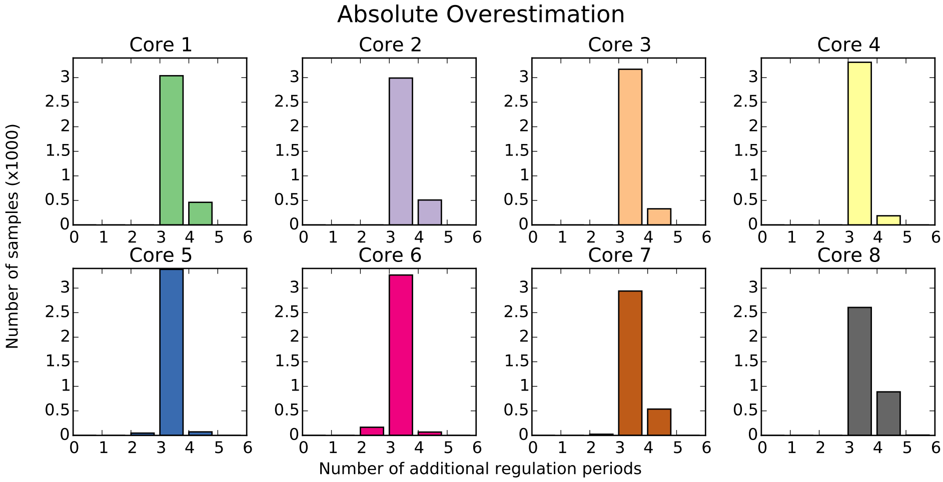

Figure 5 Absolute overestimation, in terms of additional regulation periods, compared to exact (brute-force) derivation on cores 1 to 8.

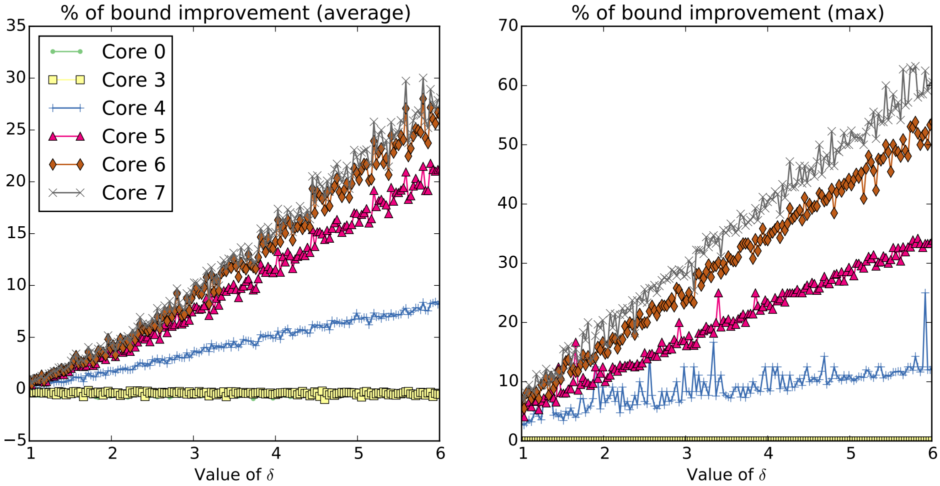

Figure 6 Percentage of WCET improvement over analysis proposed in [37] as a function of δ on cores 1 to 8. Higher y-values means better improvements.

8 Conclusion and Future Work

In this work, we discussed an improved analysis strategy to derive the WCET of a task under memory bandwidth by exploiting exact knowledge of budget-to-core assignments. In this way, we show that it is possible to derive a more accurate WCET estimation, with performance gains that go from 30% in average up to 60%, compared to the state of the art.

As a future work, we plan to validate our analysis on a commercial multi-core platform, and to study how different budget assignment strategies affect the performance of the WCET estimation. Additionally, we plan to extend this work with more in-depth considerations about algorithmic complexity as well as possible optimizations.

References

1 AbsInt. aiT worst-case execution time analyzers, 2014. URL: http://www.absint.com/ait/.

2 B. Akesson, K. Goossens, and M. Ringhofer. Predator: A predictable SDRAM memory controller. In *Proceedings of the 5th IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis*, CODES+ISSS'07, pages 251–256, New York, NY, USA, 2007. ACM. doi:10.1145/1289816.1289877.

3 S. Altmeyer, R. I. Davis, L. Indrusiak, C. Maiza, V. Nelis, and J. Reineke. A generic and compositional framework for multicore response time analysis. In *Proceedings of the 23rd International Conference on Real Time and Networks Systems*, RTNS'15, pages 129–138, New York, NY, USA, 2015. ACM. doi:10.1145/2834848.2834862.

4 D. Bui, E. A. Lee, I. Liu, H. Patel, and J. Reineke. Temporal isolation on multiprocessing architectures. In *Proceedings of the 48th Design Automation Conference*, DAC'11, pages 274–279, New York, NY, USA, 2011. ACM. doi:10.1145/2024724.2024787.

5 S. Chattopadhyay, C. L. Kee, A. Roychoudhury, T. Kelter, P. Marwedel, and H. Falk. A unified wcet analysis framework for multi-core platforms. *ACM Trans. Embed. Comput. Syst.*, 13(4s):124:1–124:29, April 2014. doi:10.1145/2584654.

6 S. Chattopadhyay, A. Roychoudhury, and T. Mitra. Modeling shared cache and bus in multi-cores for timing analysis. In *Proceedings of the 13th International Workshop on Software & Compilers for Embedded Systems*, SCOPES'10, pages 6:1–6:10, New York, NY, USA, 2010. ACM. doi:10.1145/1811212.1811220.

7 P. Cousot. Abstract interpretation based formal methods and future challenges. In *Informatics – 10 Years Back. 10 Years Ahead.*, volume 2000 of *Lecture Notes in Computer Science*, pages 138–156, London, UK, UK, 2001. Springer-Verlag. doi:10.1007/3-540-44577-3_10.

8 R. I. Davis and A. Burns. Resource sharing in hierarchical fixed priority pre-emptive systems. In *2006 27th IEEE International Real-Time Systems Symposium (RTSS'06)*, RTSS'06, pages 257–270, Washington, DC, USA, 2006. IEEE Computer Society. doi:10.1109/RTSS.2006.42.

9 S. A. Edwards and E. A. Lee. The case for the precision timed (PRET) machine. In *44th Annual Design Automation Conference*, DAC'07, pages 264–265, New York, NY, USA, 2007. ACM. doi:10.1145/1278480.1278545.

10 G. Gracioli and A. Fröhlich. On the influence of shared memory contention in real-time multicore applications. In *Proceedings of the IV Brazilian Symposium on Computing Systems Engineering (SBESC)*, Washington, DC, USA, 2014. IEEE Computer Society. doi:10.1109/SBESC.2014.8.

11 S. Hahn, M. Jacobs, and J. Reineke. Enabling compositionality for multicore timing analysis. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, RTNS'16, pages 299–308, New York, NY, USA, 2016. ACM. doi:10.1145/2997465.2997471.

12 D. Hardy, T. Piquet, and I. Puaut. Using bypass to tighten WCET estimates for multi-core processors with shared instruction caches. In *Proceedings of the 2009 30th IEEE Real-Time Systems Symposium*, RTSS'09, pages 68–77, Washington, DC, USA, 2009. IEEE Computer Society. doi:10.1109/RTSS.2009.34.

13 B. Jacob, S. Ng, and D. Wang. *Memory systems: cache, DRAM, disk.* Morgan Kaufmann, 2007.

14 T. Kelter, H. Falk, P. Marwedel, S. Chattopadhyay, and A. Roychoudhury. Bus-aware multicore wcet analysis through tdma offset bounds. In *Real-Time Systems (ECRTS), 2011 23rd Euromicro Conference on*, pages 3–12, July 2011. doi:10.1109/ECRTS.2011.9.

15 H. Kim, D. de Niz, B. Andersson, M. Klein, O. Mutlu, and R. R. Rajkumar. Bounding and reducing memory interference in COTS-based multi-core systems. *Real-Time Systems*, 52(3):356–395, 2016. doi:10.1007/s11241-016-9248-1.

16 L. Kosmidis, E. Quiñones, J. Abella, G. Farrall, F. Wartel, and F. J. Cazorla. Containing timing-related certification cost in automotive systems deploying complex hardware. In *Proceedings of the 51st Annual Design Automation Conference*, DAC'14, pages 22:1–22:6, New York, NY, USA, 2014. ACM. doi:10.1145/2593069.2593112.

17 Y. Li, V. Suhendra, Y. Liang, T. Mitra, and A. Roychoudhury. Timing analysis of concurrent programs running on shared cache multi-cores. *Real-Time Systems*, 48(6):638–680, November 2012. doi:10.1007/s11241-012-9160-2.

18 T. Lundqvist and P. Stenström. Timing anomalies in dynamically scheduled microprocessors. In *Proceedings of the 20th IEEE Real-Time Systems Symposium*, pages 12–, Washington, DC, USA, 1999. IEEE Computer Society. doi:10.1109/REAL.1999.818824.

19 R. Mancuso, R. Dudko, E. Betti, M. Cesati, M. Caccamo, and R. Pellizzoni. Real-time cache management framework for multi-core architectures. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, RTAS'13, pages 45–54, Philadelphia, PA, USA, April 2013. IEEE Computer Society. doi:10.1109/RTAS.2013.6531078.

20 R. Mancuso, R. Dudko, and M. Caccamo. Light-prem: Automated software refactoring for predictable execution on cots embedded systems. In *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2014 IEEE 20th International Conference on*, pages 1–10, Aug 2014. doi:10.1109/RTCSA.2014.6910515.

21 R. Mancuso, R. Pellizzoni, M. Caccamo, L. Sha, and H. Yun. WCET(m) estimation in multi-core systems using single core equivalence. In *Real-Time Systems (ECRTS), 2015 27th Euromicro Conference on*, pages 174–183, July 2015. doi:10.1109/ECRTS.2015.23.

22 M. Paolieri, E. Quiñones, F. J. Cazorla, G. Bernat, and M. Valero. Hardware Support for WCET Analysis of Hard Real-time Multicore Systems. *SIGARCH Comput. Archit. News*, 37(3):57–68, June 2009. doi:10.1145/1555815.1555764.

23 R. Pellizzoni, E. Betti, S. Bak, G. Yao, J. Criswell, M. Caccamo, and R. Kegley. A predictable execution model for COTS-based embedded systems. In *Proceedings of the 2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, RTAS'11, pages 269–279, Washington, DC, USA, 2011. IEEE Computer Society. doi:10.1109/RTAS.2011.33.

24 R. Pellizzoni and H. Yun. Memory servers for multicore systems. In *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 1–12, April 2016.

25 J. Reineke, I. Liu, H.D. Patel, S. Kim, and E.A. Lee. PRET DRAM controller: bank privatization for predictability and temporal isolation. In *Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, CODES+ISSS'11, pages 99–108, New York, NY, USA, 2011. ACM. doi:10.1145/2039370.2039388.

26 J. Rosén, A. Andrei, P. Eles, and Zebo Peng. Bus access optimization for predictable implementation of real-time applications on multiprocessor Systems-on-Chip. In *Proceedings of the 28th IEEE International Real-Time Systems Symposium*, RTSS'07, pages 49–60. IEEE Computer Society, 2007. doi:10.1109/RTSS.2007.13.

27 A. Schranzhofer, J. J. Chen, and L. Thiele. Timing analysis for tdma arbitration in resource sharing systems. In *2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 215–224, April 2010. doi:10.1109/RTAS.2010.24.

28 L. Sha, M. Caccamo, R. Mancuso, J.E. Kim, M.K. Yoon, R. Pellizzoni, H. Yun, R.B. Kegley, D.R. Perlman, G. Arundale, and R. Bradford. Real-time computing on multicore processors. *Computer*, 49(9):69–77, Sept 2016. doi:10.1109/MC.2016.271.

29 I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees. In *RTSS 2003. 24th IEEE Real-Time Systems Symposium, 2003*, pages 2–13, Dec 2003. doi:10.1109/REAL.2003.1253249.

30 R. Tabish, R. Mancuso, S. Wasly, A. Alhammad, S. S. Phatak, R. Pellizzoni, and M. Caccamo. A real-time scratchpad-centric OS for multi-core embedded systems. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2016 IEEE 22th*, April 2016. doi:10.1109/RTAS.2016.7461321.

31 R. Tabish, R. Mancuso, S. Wasly, S. S. Phatak, R. Pellizzoni, and M. Caccamo. A reliable and predictable scratchpad-centric OS for multi-core embedded systems. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2017 IEEE 23th*, April 2017.

32 T. Ungerer, F. Cazorla, P. Sainrat, G. Bernat, Z. Petrov, C. Rochange, E. Quinones, M. Gerdes, M. Paolieri, J. Wolf, H. Casse, S. Uhrig, I. Guliashvili, M. Houston, F. Kluge, S. Metzlaff, and J. Mische. MERASA: Multicore execution of hard real-time applications supporting analyzability. *IEEE Micro*, 30(5):66–75, 2010. doi:10.1109/MM.2010.78.

33 S. Wasly and R. Pellizzoni. A dynamic scratchpad memory unit for predictable real-time embedded systems. In *Real-Time Systems (ECRTS), 2013 25th Euromicro Conference on*, pages 183–192. IEEE, 2013. doi:10.1109/ECRTS.2013.28.

34 I. Wenzel, R. Kirner, B. Rieder, and P. Puschner. Measurement-based worst-case execution time analysis. In *Third IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems (SEUS'05)*, pages 7–10, May 2005. doi:10.1109/SEUS.2005.12.

35 J. Yan and W. Zhang. WCET analysis for multi-core processors with shared L2 instruction caches. In *Proceedings of the 2008 IEEE Real-Time and Embedded Technology and Applications Symposium*, RTAS'08, pages 80–89, Washington, DC, USA, 2008. IEEE Computer Society. doi:10.1109/RTAS.2008.6.

36 G. Yao, R. Pellizzoni, S. Bak, E. Betti, and M. Caccamo. Memory-centric scheduling for multicore hard real-time systems. *Real-Time Systems*, 48(6):681–715, November 2012. doi:10.1007/s11241-012-9158-9.

37 G. Yao, H. Yun, Z. P. Wu, R. Pellizzoni, M. Caccamo, and L. Sha. Schedulability analysis for memory bandwidth regulated multicore real-time systems. *IEEE Transactions on Computers*, 65(2):601–614, February 2016. doi:10.1109/TC.2015.2425874.

38 H. Yun, R. Mancuso, Z.-P. Wu, and R. Pellizzoni. PALLOC: DRAM Bank-Aware Memory Allocator for Performance Isolation on Multicore Platforms. In *Proceedings of the IEEE Intl. Conference on Real-Time and Embedded Technology and Applications Symposium (RTAS)*, Berlin, Germany, April 2014. doi:10.1109/RTAS.2014.6925999.

39 H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha. Memory access control in multiprocessor for real-time systems with mixed criticality. In *Proceedings of the 2012 24th Euromicro Conference on Real-Time Systems*, ECRTS'12, pages 299–308, Washington, DC, USA, 2012. IEEE Computer Society. doi:10.1109/ECRTS.2012.32.

40 H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha. MemGuard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms. In *Proceedings of the 2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, RTAS'13, pages 55–64. IEEE Computer Society, 2013. doi:10.1109/RTAS.2013.6531079.

41 H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha. Memory bandwidth management for efficient performance isolation in multi-core platforms. *IEEE Transactions on Computers*, 65(2):562–576, February 2016. doi:10.1109/TC.2015.2425889.