MC2: Multicore and Cache Analysis via Deterministic and Probabilistic Jitter Bounding

Enrique Díaz[1,2], Mikel Fernández[1], Leonidas Kosmidis[1], Enrico Mezzetti[1], Carles Hernandez[1], Jaume Abella[1], and Francisco J. Cazorla[1,3]

[1]Barcelona Supercomputing Center (BSC), Barcelona, Spain
{enrique.diaz, mikel.fernandez, leonidas.kosmidis, enrico.mezzetti, carles.hernandez, jaume.abella, francisco.cazorla}@bsc.es
[2]Universitat Politècnica de Catalunya, Barcelona, Spain
[3]IIIA-CSIC, Bellaterra, Spain

Abstract. In critical domains, reliable software execution is increasingly involving aspects related to the timing dimension. This is due to the advent of high-performance (complex) hardware, used to provide the rising levels of guaranteed performance needed in those domains. Caches and multicores are two of the hardware features that have the potential to significantly reduce WCET estimates, yet they pose new challenges on current-practice measurement-based timing analysis (MBTA) approaches. In this paper we propose *MC2*, a technique for multilevel-cache multicores that combines deterministic and probabilistic jitter-bounding approaches to reliably handle both the variability in execution time generated by caches and the contention in accessing shared hardware resources. We evaluate *MC2* on a COTS quad-core LEON-based board and our initial results show how it effectively captures cache and multicore contention in pWCET estimates with respect to actual observed values.

Keywords: WCET · MBTA · Multicore contention · Probabilistic timing analysis · Jitter bounding

1 Introduction

Computing power needs are steadily increasing in the critical real-time embedded domains, fuelled by the complexity and sheer amount of data a modern on-board software is expected to handle [3, 7, 33]. At hardware level, while high-performance features, such as caches and multicore processors, provide the demanded performance, they also bring about hard-to-model *jitter* (variability) in execution time, which complicates timing validation and verification. This has resulted in an increased attention on timing in safety standards (e.g., ISO26262 [15] in automotive) and support documents (e.g., CAST32-A [8] in aerospace).

MBTA is the dominant timing analysis approach in most real-time domains [34]. MBTA aims at deriving a worst-case execution time (WCET) estimate that holds for the program during *system operation* from the execution time measurements captured during the tests executed at various stages in the *analysis phase*. The quality of the derived WCET estimates lies on the user's ability

The final publication is available at Springer via https://link.springer.com/chapter/10.1007/978-3-319-60588-3_7

to design stressful test scenarios (conditions) that are presumably close to the worst-case conditions that can arise during system operation. The degree of control available to the user, while adequate on simple processor designs, diminishes with the inclusion of complex hardware that challenges: (i) designing worst-case scenarios, e.g., identifying the memory object allocation (code and data) that results in cache set mappings with high impact on execution time, and the worst contention scenarios that the application can suffer in a multicore; and (ii) designing experiments in which bad (pathological) behavior for several resources occurs simultaneously. Overall, despite the user may perform thousands of experiments, there is no guarantee on whether the bad behavior in the sources of jitter (*soj*), like the cache, has been sufficiently captured. This reduces the confidence on the MBTA WCET estimates, which in turn can prevent the use of some high-performance hardware features in critical real-time embedded systems.

Measurement-Based Probabilistic Timing Analysis (MBPTA) [5, 31] is a variant of MBTA that aims at increasing the confidence on WCET estimates. MBPTA, which has been successfully evaluated on several case studies (e.g., [31, 32]), aims at relieving the user from controlling hardware *soj*. Instead, MBPTA makes that their impact on the measurements emerges naturally, reducing user's burden to only controlling the number of runs to perform [20]. To that end, MBPTA implicitly controls the impact of jittery resources on measurements captured at analysis. In particular, some resources are forced to work on their worst latency during analysis (upperbounding), hence ensuring measurements conservatively capture their impact. The latency of other resources is instead randomized so that their execution times at analysis vary according to a probabilistic execution time distribution that can be used to upperbound the latencies during operation.

In this paper we propose the *MC2* (multicore and cache) MBPTA approach for the analysis of a Commercial-Off-The-Shelf (COTS) multicore processor equipped with multilevel-caches. While hardware designs have been proposed [17][14] for MBPTA compliance, and some of them have hit pre-silicon (RTL) readiness level [14], analyzing MBPTA applicability on COTS multicore processors is fundamental to favor a fast and widespread adoption of MBPTA. MC2 exposes, in a combined MBPTA-compliant manner, the jitter of caches and multicore contention to the execution time measurements taken at analysis. As a result, the WCET estimates MBPTA generates from those measurements upperbound the impact of both resources on program execution time. MC2 combines two techniques that have been classified as MBPTA compliant: software randomization [18] for cache-jitter management, and delay upperbounding for multicore contention management [16]. For the latter, since multicore contention can lead to very pessimistic WCET estimates [13] when contention bounds are provisioned for the worst possible contention, MC2 provides adaptable WCET estimates that depend on contenders' contention. Our results provides evidence that MC2 effectively captures the impact on execution time - and hence on WCET estimates - of both resources, and provides tight WCET estimates.

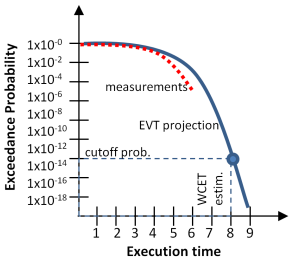

Fig. 1: pWCET example.

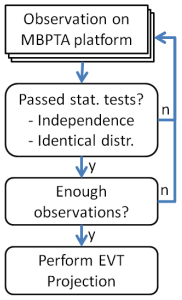

Fig. 2: MBPTA steps.

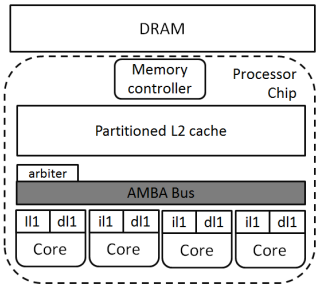

Fig. 3: Reference architecture.

2 Background

When selecting the timing analysis technique to use, industrial users balance the cost-effectiveness of the technique and the evidence that it can provide to satisfy the level of confidence required by the domain-specific standards [1]. MBTA techniques are less rigorous than static analysis methods but, in general, are more attractive because of their cost-effectiveness and major affinity with the consolidated industrial practice. The quality of MBTA's derived WCET estimates relates to the evidence on their coverage of the worst-case conditions. When evidence obtained is sufficient, MBTA can be used for high-integrity software, e.g., DAL-A functions in avionics [22]. In practice, all techniques require user-provided inputs, e.g., worst-case scenarios for measurements for MBTA and hardware timing models for static timing analysis (with hardware documentation potentially being inaccurate or incomplete [1], thus eventually resorting to measurements to reverse engineering the timing model [26]). This makes complex argue about the quality of a WCET figure. In this paper, we focus on MBTA with the intent to increase the confidence that can be placed on the provided WCET estimates.

MBPTA. MBPTA applies Extreme Value Theory [9] (EVT) on execution time observations from the analysis phase to derive the probabilistic WCET (pWCET) distribution that upperbounds program's execution time during operation. MBPTA requires guaranteeing that the observations obtained at analysis capture those events that can impact execution time at operation, and so pWCET estimates [1]. MBPTA, by deploying EVT (see Figure 1), is able to derive the probability that bad behavior of several of the *soj* (whose impact has been captured in the analysis-time runs) are triggered in the same run. Hence, EVT has to be seen as a method to predict pathological combinations of observed events in the analysis-time measurements. In general, EVT cannot predict the appearance of unobserved events since their impact on execution time can be arbitrarily large. To cover this gap, MBPTA builds an argument on representativeness by means of i) either injecting randomization in the timing behavior of certain hardware resources (e.g., caches and buses) so that it is possible to determine the probability of their worst behavior to be captured in the analysis-time measurement

runs; or ii) making resources to work on their worst latency so the analysis time measurements capture their worst timing behavior.

MBPTA application procedure starts by (1) collecting a set of representative observations, see Figure 2. MBPTA then (2) applies some statistical test such as independence and identical distribution tests [5] required for EVT application. Since in a MBPTA-compliant platform these *probabilistic* properties hold by construction[1], in case *statistical* tests are failed, the user is simply asked for more runs until statistical tests – which are subject to false positives/negatives – are passed. (3) MBPTA checks whether the size of the sample is enough to include all relevant events and ensure certain statistical stability of the results. To that end we use the initial findings in [24] and ask the user for more runs until this condition is satisfied. As final step, (4) MBPTA derives an EVT distribution (pWCET estimate) as shown in Figure 1.

Software Randomization. MBPTA handles resources with small jitter (usually in the order of few cycles) by means of upperbounding, i.e., by forcing the resource to operate on its worst latency during analysis time [14]. However, cache resources exhibit high jitter between hit and miss events, especially when these events span across multiple levels of cache. For this reason, timing randomization is used. In particular we use software randomization which, by randomly varying the memory layout between distinct program executions, causes cache events (hits/misses) to have a probabilistic behavior that holds during operation. This allows cache jitter to be properly modelled by MBPTA. In this work we use our custom implementation of TASA (Toolchain Agnostic Software rAndomization)[21, 19], a static variant of software randomization, applied at source-code level. TASA randomizes the position in memory for any memory object in the software under analysis such as functions, stack frames and global data. Moreover, TASA can randomly affect the internal memory layout of several memory objects such as stack frames and structures.

In general, compilers allocate memory objects in the order they are in the source file. Very few compiler options violate this principle, which can be disabled during compilation with small (if any) impact in the compiler performance [21]. TASA, by randomly rearranging the order of declarations for the corresponding objects in the source file, modifies their relative position in the binary. This, in combination with additional random-sized padding in the form of `nop` instructions or unused data, increases the potential difference among binary layouts. When the binary is loaded to main memory for the program execution, the random binary memory layout translates into random main memory mapping and hence, a random cache layout, i.e., memory objects are allocated in random cache sets.

[1] Despite time-randomization, programs might exhibit a degenerate distribution of timing, e.g., having a single or very few different execution times. While extremely rare in practice for real-size programs, the lack of jitter would suggest that the maximum observed execution time could be reasonably used as a precise WCET indicator.

3 Reference Platform

We use a 4-core LEON3 [2] platform implemented on an FPGA. Each LEON3 core implements a 7-stage pipeline and comprises first level instruction (*ic*) and data (*dc*) caches, with the *dc* implementing a write-through no write allocate policy, see Figure 3. An AMBA AHB bus propagates stores, *dc* misses and *ic* misses to the partitioned L2 cache deploing a write-back policy. Requests sent to the bus are not split. Hence, the bus is locked all the time a request accesses the L2. If it misses in L2, the bus is locked until the request is solved in main memory and answered back. Requests are arbitrated in the bus using round-robin which provides time analyzability [12]. Hence, our reference architecture comprises two main hardware shared resources, the bus and the memory, with the bus arbiter controlling the contention in both of them. Our platform also comprises performance monitoring counters (PMCs) from which we track *ic* misses, *dc* misses, store operations and L2 misses, as detailed in Section 4.

In our experiments we consider one task under analysis (*tua* or τ_a) and several (up to three) contender tasks, referred to as $c(\tau_a)$ or τ_b, τ_c and τ_d. τ_a is always a time-critical task for which a WCET estimate is to be derived.

4 Handling Multicore Contention and Cache Jitter

Goals and Challenges. MC2 aims at reliably capturing the impact that multicore contention (handled by the bus arbiter in our reference architecture) and cache jitter have on pWCET estimates. This requires ensuring that the execution time observations collected at analysis capture the impact of the jitter of both. To ease MC2 adoption, this goal has to be achieved under the following restrictions:

1. MBPTA compliance. The proposed technique must be MBPTA-compliant requiring minimum changes to the single-core MBPTA timing analysis approach, which has already been evaluated with several industrial case studies [31].
2. pWCET estimates should be time composable, so that they are independent of the load contenders put on resources. Time composability enables incremental integration of applications, performing timing analysis of applications mostly in isolation, without the need of regression tests. Time composability also allows updating functionality during system operation without the need of analysing the entire task set, but just those tasks that are updated.
3. The information required by MC2 from the tasks should be obtained via PMCs to facilitate its applicability to real hardware.
4. WCET estimates should be obtained as early as possible in the design process to facilitate incremental software integration [23] (ideally during unit testing) and should hold across integrations for incremental verification purposes.

Overall Process. MC2 process starts by running the software-randomized task under analysis (τ_a) in isolation, see Figure 4. This exposes the impact of cache jitter to the observed execution time (oet^i) in each run r^i. As a side effect, since the hit/miss pattern of τ_a changes across runs (due to software-randomization),

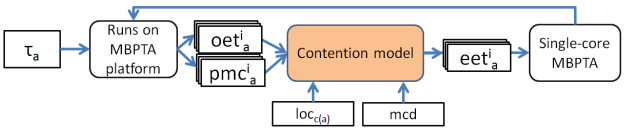

Fig. 4: Schematic view of the proposed pTC contention model.

its number of accesses to the bus and the memory also varies. Hence, τ_a has an access distribution to cache/memory rather than a single value (with small variations) as it would be the case if τ_a had not been time randomized.

MC2 also factors in the maximum contention delay (mcd) for each request type and the level of contention generated by τ_a's contenders ($c(\tau_a)$), which is refereed as $loc_{c(a)}$. Δ_{cont} in Equation (1) captures both mcd and $loc_{c(a)}$. By feeding MBPTA with enlarged execution times (eet), MC2 provides MBPTA with representative information on the impact of cache and multicore contention.

$$eet^i = oet^i + \Delta_{cont} \tag{1}$$

Detailed Explanation. MC2 builds upon the following assumptions and inputs:

1. *τ_a's observed execution times* (oet^i_a) in each run r^i_a of τ_a's in isolation.
2. *τ_a's number of requests* (pmc^i_a) obtained with PMC readings in each run r^i_a of τ_a's in isolation. The particular counters are discussed later.
3. *Worst-case request overlap assumptions*: MC2 assumes that contenders' requests align in the worst possible manner with each τ_a request causing maximum impact on τ_a's execution time. While this assumption is pessimistic, it relieves the end user from modelling the particular cycle when requests occurs, which would be an overly expensive effort, and would only be doable after integration. Instead, assuming mcd delay for each contenders' request brings some pessimism but allows MC2 enable WCET estimates during unit testing to favor incremental integration [23]. This is in contrast to the number of requests that can be derived during unit testing and do not change (for our architecture) at integration, i.e., the number of requests of a task to the bus/memory is not affected by its contenders.
4. *User-provided contender's level of contention* ($loc_{c(a)}$): MC2 factors in the contention (i.e., number of requests) of $c(\tau_a)$. To that end, we follow two models. The first one, called fully Time Composable (fTC), assumes each contender task makes as many requests of the longest duration as total number of requests generated by τ_a. The fTC models results in fully time-composable estimates, but at the cost of over-estimation. To reduce the latter, a second model, called partially Time Composable (pTC), is adjustable to the expected level of contention of the contenders (i.e., its number of requests and their type). The pTC model derives the WCET estimate for τ_a in isolation under a given level of contention of its contenders. At integration time, composability can be assessed by simply checking that the contention level of the particular contenders is smaller than the level assumed at analysis. Both models are detailed in Section 5 and Section 6 respectively.

Table 1: Request types and their latency in our reference board.

Type	mcd	Description
sh	$l^{sh} = 1$	L2 st hit
lh	$l^{lh} = 8$	L2 ld hit in L2
lmc	$l^{lmc} = 28$	L2 ld clean miss
smc	$l^{smc} = 28$	L2 st clean miss
lmd	$l^{lmd} = 31$	L2 ld dirty miss
smd	$l^{smd} = 31$	L2 st dirty miss

Table 2: PMCs available in the reference processor.

Name	Description
pmc^{icm}	Bus reads caused by ic misses
pmc^{dcm}	Bus reads caused by dc misses
pmc^{st}	Writes to L2
pmc^{m}	Misses in the L2

Request Characteristics. MC2 requires information about the types of requests to the bus, with emphasis on those having different usage of the bus, and the maximum time each request holds the bus (mcd).

From processor manuals, we identify six types of request to the bus (see Table 1): load/store requests that hit/miss in L2, and for the case of misses, since the L2 is write-back, request evicting and not evicting dirty data. The former are called dirty misses and the latter clean misses.

The mcd for each request, see the second column of Table 1, is the time interval (measured in cycles) since a request is granted access to the bus until it relinquishes the bus. Hence, mcd is the maximum contention that a request of each type can incur on other requests. We have derived mcd values empirically following the process described in [16]. The general approach consists in generating small benchmarks that generate a single-type of requests (e.g., load hits in L2) and architect experiments so tight bounds to request latencies can be derived.

5 fTC contention Model

fTC derives a WCET estimate that is an upper bound to the slowdown τ_a can suffer regardless of the load its contender tasks put on the shared resources. This requires the model to pessimistically assume that the number of contenders equals $Nc-1$ where Nc is the number of cores – four in our platform. Further, the model assumes that for every τ_a request its contenders have one request of the worst type, i.e., causing the longest contention on it that in our architecture corresponds to lmd and smd (indistinctly referred to as xmd). Hence, fTC assumes that each request of τ_a is delayed 31 $cycles = l^{smd} = l^{lmd}$ by each contenders' request. Overall, fTC builds a set of enlarged execution time observations as shown in Equation (2), where n_a^i is the total number of request that τ_a performs in run r^i.

$$eet_a^i = oet_a^i + \Delta_{cont}^{i,fTC} = oet_a^i + \left[n_a^i \times (Nc-1) \times l^{xmd}\right] \quad (2)$$

6 pTC model

The fTC model may result in noticeably pessimistic WCET estimates. The partially Time Composable (pTC) model presented in this section trades time

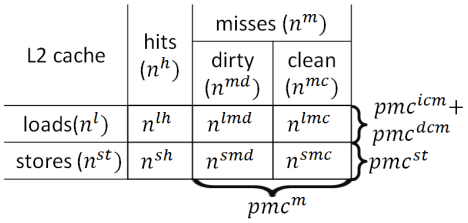

Fig. 5: Events and PMCs

	τ_b requests conflicting with τ_a requests	τ_a unpaired requests
before pairing		n_a^i
Pairing md	$\check{c}^{md} = \min(n_a, \hat{n}_b^{md})$	$n_a^{ii} = \max(0, n_a^i, -\check{c}^{md})$
Pairing mc	$\check{c}^{mc} = \min(n_a^{ii}, \check{n}_b^{mc})$	$n_a^{iii} = \max(0, n_a^{ii} - \check{c}^{mc})$
Pairing lh	$\hat{c}^{lh} = \min(n_a^{iii}, \hat{n}_b^{lh})$	$n_a^{iiii} = \max(0, n_a^{iii} - \hat{c}^{lh})$
Pairing sh	$\check{c}^{sh} = \min(n_a^{iiii}, \check{n}_b^{sh})$	$n_a^{iiii} = \max(0, n_a^{iiii} - \check{c}^{sh})$
after pairing		n_a^{iiii}

Fig. 6: Pairing steps

composability to tighten WCET estimates. With pTC [12], the user can yet enjoy benefits of incremental integration with small effort to assess time composability. The pTC model, instead of assuming $Nc-1$ contenders, takes the actual number of running τ_a contenders. Also, unlike fTC, pTC tracks the number of requests of each type. This offers a powerful solution to tighten WCET estimates with a reasonable low impact on time composability. pTC assumes that an upper bound to the number of contenders' request of each type can be derived.

The pTC model derives the impact that the number of requests for each contender task τ_b can cause on τ_a. Ideally, we would like to have a PMC for the number of requests of each type made by the task. We refer to that ideal counter as n^{xxx} where xxx corresponds to one of the types in Table 1. However, in the target platform there is not a specific set of PMCs measuring those values as shown in Table 2, which lists the relevant PMCs we used.

We performed an analysis of the relation we derive among the events needed by the pTC model and the PMCs in the architecture (pmc^{yyy}) as shown in Figure 5: the number of loads to L2 (n^l) matches the number of misses to the dc and the ic ($pmc^{dcm} + pmc^{icm}$); the number of stores matches pmc^{st}; and the number of misses pmc^m cover those caused by clean and dirty evictions ($pmc^m = n^{lmc} + n^{smc} + n^{lmd} + n^{smd}$). Further, the number of stores n^{st} matches $pmc^{st} = n^{sh} + n^{smd} + n^{smc}$.

With the existing PMC we approximate the number of requests of each type made by each contender task. In doing so, we take into account the request latency so that the resulting impact that τ_b causes on τ_a derived with PMCs is an upperbound to the actual one we would derive if we had the ideal PMCs. The approach consists in upper bounding high-latency requests first, which in our architecture are dirty misses (lmd and smd).

Bounding Dirty Misses: The number of L2 misses evicting a dirty line is upper bounded by the minimum between the number of stores ($n^{st} = pmc^{st}$) that cause lines to be dirty and the number of L2 misses ($n^m = pmc^m$) that evict cache lines, see left part of Equation (3).

$$\boxed{\hat{n}^{md} = min(pmc^m, pmc^{st})} \quad \rightarrow \quad \boxed{\check{n}^{mc} = pmc^m - \hat{n}^{md}} \tag{3}$$

Since $n^m = n^{md} + n^{mc}$, the approximation in Equation (3) may result in assuming that some misses generate dirty evictions while, in reality, they do not,

thus introducing some pessimism. In particular, it results in a lower bound to the number of clean misses ($\check{n}^{mc}$), see the right side of Equation (3).

Bounding Load Hits: The number of loads that hit in cache is upper bounded by the minimum between the number of hits (n^h) and the number of loads (n^l) to the L2, see Equation (4). They are respectively computed from PMCs as follows:

The number of loads performed to the L2 cache (n^l) equals the number of misses in the *dc* and *ic*, i.e., $n^l = pmc^{icm} + pmc^{dcm}$. Note that the number of loads to the L2 includes hits and misses (dirty and clean), i.e., $n^l = n^{lh} + n^{lmc} + n^{lmd}$.

The number of hits in L2, n^h is obtained with existing PMCs as $n^h = (pmc^{icm} + pmc^{dcm} + pmc^{st}) - pmc^m$, that is, the number of read/write accesses to the L2, which include load misses in the *ic* and the *dc* plus stores (due to the write-through policy of the *dc* cache), minus the number of L2 misses. Note that pmc^m does only count the number of direct misses. More specifically, it does not count the number of memory accesses due to write backs.

$$\boxed{\hat{n}^{lh} = min(n^h, n^l)} \quad \rightarrow \quad \boxed{\check{n}^{sh} = n^h - \hat{n}^{lh}} \tag{4}$$

Since $n^h = n^{lh} + n^{sh}$, a lower bound to the number of store hits is derived as shown in the right side of Equation (4).

Bounding Contention: Once bounds to τ_b accesses have been computed, the pTC model assumes that requests from τ_b delay τ_a requests by their respective *mcd*. This is implemented by iteratively "pairing" each request from a run r^i of τ_a with one request of τ_b from worst to best latency, see Figure 6.

1. First, the number of requests from task τ_b of type miss dirty ($\hat{n}^{md}$), i.e the type with highest *mcd*, that contend with requests of τ_a, n^i_a, is given by: $\hat{c}^{md} = min(n^i_a, \hat{n}^{md}_b)$. Hence the number of unpaired requests from τ_a is $n'^i_a = max(0, n^i_a - \hat{c}^{md}_b)$ requests of τ_a unpaired.
2. Those n'^i_a requests contend with $\check{n}^{mc}$ (second most impacting type) requests of τ_b: $\check{c}^{mc} = min(n'^i_a, \check{n}^{mc}_b)$. This results in $n''_a = max(0, n'^i_a - \check{c}^{mc})$ τ_a's unpaired requests.
3. Those n''^i_a requests contend with $\hat{n}^{lh}_b$ (third most impacting type) requests of τ_b: $\hat{c}^{lh} = min(n''^i_a, \hat{n}^{lh}_b)$ with $n'''_a = max(0, n''^i_a - \hat{c}^{lh})$ requests unpaired.
4. Finally, the n'''^i_a remaining τ_a's requests contend with $\check{n}^{sh}_b$ (fourth most impacting type) requests of τ_b: $\check{c}^{sh} = min(n'''^i_a, \check{n}^{sh}_b)$. With the remaining τ_a's request $n''''^i_a = max(0, n'''^i_a - \check{c}^{sh})$ not contending with any request of τ_b.

The obtained pTC contention is the result of assuming that each of these contentions among τ_a and its contender τ_b are aligned in the worst way, causing a contention delay as long as each τ_b request (see Equation (5)). This process is repeated for the other potential contender tasks τ_c and τ_d. The overall pTC contention bound is given by Equation (6).

$$\Delta^{i,pTC}_{\tau_b \rightarrow \tau_a} = (\hat{c}^{md} \times l^{md}) + (\check{c}^{mc} \times l^{mc}) + (\hat{c}^{lh} \times l^{lh}) + (\check{c}^{sh} \times l^{sh}) \tag{5}$$

$$\Delta^{i,pTC}_{cont} = \Delta^{i,pTC}_{\tau_b \rightarrow \tau_a} + \Delta^{i,pTC}_{\tau_c \rightarrow \tau_a} + \Delta^{i,pTC}_{\tau_d \rightarrow \tau_a} \tag{6}$$

Other Considerations. The fTC model has the advantage of breaking the dependence between scheduling and WCET. In an exact model, the WCET figure to be used depends on the schedule of tasks, which creates a circular dependence as WCET is also an input for deriving a feasible schedule. This issue has been initially tackled by an iterative approach to simultaneously attack WCET and scheduling [11]. With fTC the WCET estimate is not affected by the load contender tasks put on the shared resources, and hence it does not depend on task scheduling. However, this comes at the cost of inflated WCET estimates.

It is worth noting that the principles of the presented model does not only apply to the studied processor but also to other multicore processors. In order to adapt the model, it is necessary to understand the type of events using the shared resource and their duration. The quality of the results, however, depends on the PMC support available and the accuracy it guarantees in tracking the desired events, see Figure 5. As part of our current work we are extending the model to multicore processors in other domains, e.g., automotive.

7 Experimental Results

We first demonstrate our combined approach on a synthetic application and then with benchmarks of the EEMBC Automotive suite [28].

Hardware Setup. We used an FPGA implementation of the LEON3 [2] platform, as introduced in Section 3. Each core comprises separate 16KB 4-way set-associative L1 caches for instruction and data, with write-through, no write allocate policy. The cache hierarchy is complemented by a shared 128KB 4-way unified L2 cache, with write-back policy. An AMBA AHB bus provides connections among private caches, the L2 and the DRAM memory controller. In our setting, we configured the L2 to be partitioned among cores (contention is still to be suffered on bus accesses), so each core has a 32KB direct-mapped L2.

MBPTA Setup. We applied MBPTA to the target program, considering 10^{-12} as the pWCET exceedance probability threshold of interest. We collected 3,000 runs to meet the representativeness requirements, as determined by the ReVS method [24]. The obtained set of observations successfully passed the statistical independence and identical distribution tests, prerequisites to the application of EVT, and allowed MBPTA to converge on a pWCET distribution.

7.1 Synthetic Application

Our synthetic application resembles an "aggressive" program uniformly accessing the shared bus for 30% of its execution time. It consists of several functions that are sequentially accessed within a loop a total of one hundred times. Each function comprises a variable number of instructions, performing a mixture of purely arithmetic and read/write operations.

Our empirical evaluation proceeds through two incremental steps. First, we assess the effectiveness of software randomization in enabling MBPTA to capture intra-core cache jitter. To that end, we execute and analyze our program in

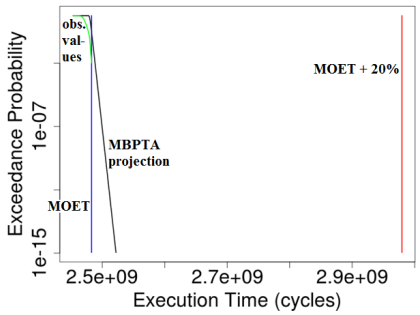

Fig. 7: Results of TASA and MBPTA on a single-core setup.

isolation (i.e., no contention at all), in a simple single-core setup. Second, we show how the results in isolation can be complemented with the analysis of inter-core contention jitter. We therefore assess our analytical model, combining cache and contention jitter, against representative execution scenarios where all cores in the system are concurrently enabled.

Capturing Cache Jitter. We exploited TASA to enable MBPTA to capture the execution time variability incurred by caches. To this extent, we analyzed our application in a single-core setup, guaranteeing complete isolation. Figure 7 reports the pWCET distribution computed by MBPTA for the target program. Observed values are upperbounded by the MBPTA projection and pWCET result is obtained by selecting the value of the projection at the 10^{-12} exceedance threshold. In this case the pWCET distribution is particularly close to the maximum observed execution time (MOET). It is worth noting that, since plain observed values do not provide any worst-case guarantee, it is common (though pretty unscientific) industrial practice to resort to a *fudge factor* to account for unknown factors. This factor is typically in the order of magnitude of 20% of the MOET. Notably, the pWCET computed with MBPTA is not only much tighter than the 20% margin, but also comes with scientific reasoning.

Multicore Contention. The MC2 approach extends single-core pWCET estimates by capturing the effect of inter-core contention through a contention model based on PMCs. In order to assess the precision of our analytical model, we performed a set of experiments on representative execution scenarios where all cores in the system are concurrently enabled and compared against the results of our analytical (fTC and pTC) models. In our setting and platform, the theoretical worst-case inter-core contention suffered by an application corresponds to the fTC scenario where all bus access requests are triggered one cycle after the reserved slot and all other cores already have pending requests, each one incurring the latency of a L2 dirty-miss. While being fully time-composable, this scenario can be extremely pessimistic in practice as it can only occur under extremely bad and rare overlapping of bus requests and cache miss patterns.

fTC. We consider first the fTC contention model as it is used as a reference for the pTC one. Our application, τ_a, is executed under two different scenarios

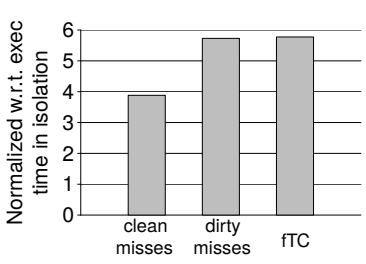

Fig. 8: Execution time when $c(\tau_a)$ create clean/dirty misses and fTC.

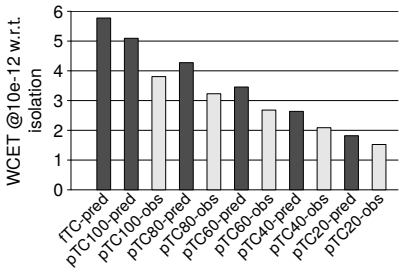

Fig. 9: Result of the pTC under different load scenarios generated by the contenders

of contention: (1) against three stressing kernels performing loads that miss in L2 (i.e., clean misses) and (2) against three stressing kernels performing stores in L2 overwriting data (i.e., dirty misses). Figure 8 shows the execution time of τ_a under the two scenarios of contention and the bound derived with the fTC model. Values are normalized against the MOET from baseline observations (i.e., no contention). As expected the model is accurate when the execution conditions are matching the fTC assumptions (i.e., worst request latencies and alignment). In practice, contenders will not generate those overly-conflictive scenarios. The pTC model cures the pessimism coming from the worst-case latency assumption.

pTC. To compare the accuracy of the pTC model and how it adapts to contenders' load on the bus, we run our application τ_a against three copies of a benchmark that performs a variable number of bus accesses depending on the configuration, which we express as a percentage of τ_a accesses. Figure 9 compares the observed execution times against the predictions of the pTC model. Results from applying the fTC are included as well for the sake of comparison. As expected, the fTC model yields pessimistic pWCET estimates. Conversely, we observe that the pTC model computes pWCET estimates decrease in parallel with the load put by contenders on the shared bus. Note that the difference among fTC and $pTC - 100\%$ is that the former assumes that all requests contribute the worst-case latency (dirty misses), whereas the latter accounts for the actual type of requests of the contenders. For any value of p, e.g., $pTC - 40\%$, the derived $pWCET@10^{-12}$ with pTC tightly upperbounds the actual observed value.

All in all this synthetic evaluation confirms that the MC2 method effectively captures both cache and multicore contention into pWCET estimates that are analytically reliable and tightly upperbounding the observed values.

7.2 EEMBC

To further evaluate our approach we applied MC2 on the EEMBC automotive benchmarks [28]. In particular, we analyzed `a2time`, `cacheb`, `idctrn`, `iirflt`, `puwmod`, and `tblook` on the same platform. Figure 10 reports, for each benchmark, MOET in both singlecore and multicore scenarios, and the results of the fTC and pTC models. For the pTC model contention was generated by deploying

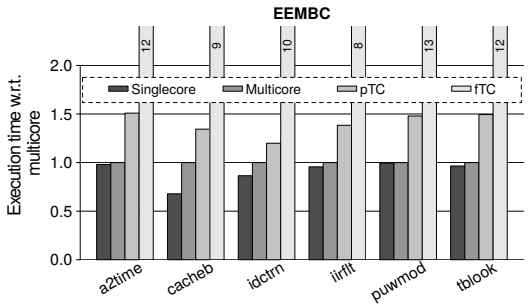

Fig. 10: Results of EEMBCs against 3 copies of themselves

three copies of the benchmark itself. All results are normalized with respect to the multicore MOET.

First we observe that fTC WCET estimate are in general extremely high (∼11x). This is explained by the fact that fTC model assumes not only the worst-case alignment scenario but also the worst-case latencies for each contender access, which is generally unrealistic. The pTC model, instead, provides quite good results, with all values below 1.5x the multicore MOET. The only pessimism in the pTC model comes from its conservative assumptions on the alignment of requests. pTC estimates provides a good compromise between tightness and flexibility: a further reduction in pessimism cannot be had without exact knowledge on how bus accesses interleave, which is not flexible and typically too difficult to derive.

8 Related Work

Several approaches have been proposed to account for inter-core contention by computing an upper bound to the delay a task or application may suffer [10]. Some of those approaches require extending classic timing analysis framework to account for the effect of shared resources [4], but they are generally unsustainable owing to the entailed computational complexity. Other approaches suggest a separate (compositional) analysis approach [29, 30, 6]. They propose a separate analysis for contention and, frequently, rely on splitting tasks into sub-tasks or phases so that worst-case alignment in (typically) TDMA-based arbiters can be reasonably computed. Assuming that tasks can be split into phases allows refining the analysis model and reducing the overall pessimism; however, this assumption is quite application-dependent and cannot be generalized. Moreover, the above approaches typically rely on insightful information on all the applications in the system and a preliminary static analysis step to characterize the pattern of memory accesses. Conversely, the contention analysis approach we rely on limits the pessimism while at the same time making no assumption on how memory accesses are distributed. Our model only requires support for PMCs, which is often available (though at variable extent) in COTS platforms.

Other approaches make use of specific hardware and/or RTOS mechanisms to enforce precomputed bounds to the maximum contention caused/suffered at run time [27, 25]. While interesting, those approaches do rely on domain-specific and custom run-time hardware mechanisms that are not typically available, and yield results that are only valid under the specific task set and system configuration. Our approach, instead, derives bounds on the inter-core contention that are at the same time realistic and only partially dependent on the co-runners characteristics, as a first step towards enabling incremental development and qualification.

The use of PMCs to model contention and derive an upper bound to multicore contention delays has been originally introduced in [16], where the analytical model for fTC and pTC is tailored to the NGMP platform. In this work, we readapt the same concept to the MBPTA framework and combines the contention model in [16] (adapted to the LEON3) with software randomization to provide holistic pWCET bounds, accounting for both cache jitter and contention effects.

9 Conclusions

We have proposed MC2, a technique for COTS multilevel-cache multicores that derives WCET estimates factoring in the jitter generated by caches and multicore contention. To that end, each measurement fed in input to MBPTA systematically accounts for the impact of both resources, effectively enabling MBPTA to factor them in when deriving pWCET estimates. Our results on a COTS platform confirm that MC2 effectively captures the impact of both multi-level cache variability and inter-core contention in realistic WCET estimates, that tightly upperbound observed values.

Acknowledgments. This work has been partially supported by the Spanish Ministry of Economy and Competitiveness (MINECO) under grant TIN2015-65316-P and the HiPEAC Network of Excellence. Jaume Abella has been partially supported by the MINECO under Ramon y Cajal postdoctoral fellowship number RYC-2013-14717. Carles Hernández is jointly funded by the MINECO and FEDER funds through grant TIN2014-60404-JIN.

References

1. J. Abella, C. Hernandez, E. Quinones, F. J. Cazorla, P. R. Conmy, M. Azkarate-askasua, J. Perez, E. Mezzetti, and T. Vardanega. Wcet analysis methods: Pitfalls and challenges on their trustworthiness. In *Industrial Embedded Systems (SIES), 2015 10th IEEE International Symposium on*, pages 1–10. IEEE, 2015.
2. Areroflex Gaisler. Leon3 Processor. 2008.06.16. `http://www.gaisler.com/index.php/products/processors/leon3`.
3. D. Buttle. Real-time in the prime-time, ETAS GmbH, Germany. In *Keynote talk at 24th Euromicro Conference on Real-Time Systems, Pisa, Italy, July*, 2012.
4. S. Chattopadhyay, L. K. Chong, A. Roychoudhury, T. Kelter, P. Marwedel, and H. Falk. A unified wcet analysis framework for multi-core platforms. In *2012 IEEE 18th Real Time and Embedded Technology and Applications Symposium*, pages 99–108, April 2012.

5. L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzetti, E. Quinones, and F. J. Cazorla. Measurement-based probabilistic timing analysis for multi-path programs. In *Real-Time Systems (ECRTS), 2012 24th Euromicro Conference on*, pages 91–101. IEEE, 2012.
6. D. Dasari, V. Nelis, and B. Akesson. A framework for memory contention analysis in multi-core platforms. *Real-Time Systems*, 52(3):272–322, 2016.
7. G. Edelin. Embedded systems at THALES: the Artemis challenges for an industrial group. In *Lecture at ARTIST Summer School*, Autrans, France, 2009.
8. Federal Aviation Administration, Certification Authorities Software Team (CAST). *CAST-32A Multi-core Processors*, 2016.
9. W. Feller. *An Introduction to Probability Theory and Its Applications*. John Willer and Sons, 1968.
10. G. Fernandez, J. Abella, E. Quiñones, C. Rochange, T. Vardanega, and F. J. Cazorla. Contention in multicore hardware shared resources: Understanding of the state of the art. In *OASIcs-OpenAccess Series in Informatics*, volume 39. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2014.
11. G. Fernandez, J. Abella, E. Quiones, L. Fossati, M. Zulianello, T. Vardanega, and F. J. Cazorla. Seeking time-composable partitions of tasks for cots multicore processors. In *2015 IEEE 18th International Symposium on Real-Time Distributed Computing*, pages 208–217, April 2015.
12. G. Fernandez, J. Jalle, J. Abella, E. Quiñones, T. Vardanega, and F. J. Cazorla. Resource usage templates and signatures for cots multicore processors. In *Proceedings of the 52nd Annual Design Automation Conference*, DAC '15, 2015.
13. M. Fernández, R. Gioiosa, E. Quiñones, L. Fossati, M. Zulianello, and F. J. Cazorla. Assessing the suitability of the ngmp multi-core processor in the space domain. In *Proceedings of the Tenth ACM International Conference on Embedded Software*, EMSOFT '12, pages 175–184, New York, NY, USA, 2012. ACM.
14. C. Hernandez, J. Abella, A. Gianarro, J. Andersson, and F. J. Cazorla. Random modulo: A new processor cache design for real-time critical systems. In *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2016.
15. International Organization for Standardization. *ISO/DIS 26262. Road Vehicles – Functional Safety*. ISO, Geneva, Switzerland, 2009.
16. J. Jalle, M. Fernandez, J. Abella, J. Andersson, M. Patte, L. Fossati, M. Zulianello, and F. J. Cazorla. Bounding resource contention interference in the next-generation microprocessor (ngmp). In *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*, 2016.
17. L. Kosmidis, J. Abella, E. Quiñones, and F. J. Cazorla. A cache design for probabilistically analysable real-time systems. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '13, 2013.
18. L. Kosmidis, C. Curtsinger, E. Quiñones, J. Abella, E. Berger, and F. J. Cazorla. Probabilistic timing analysis on conventional cache designs. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '13, 2013.
19. 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.
20. L. Kosmidis, E. Quiones, J. Abella, T. Vardanega, I. Broster, and F. J. Cazorla. Measurement-based probabilistic timing analysis and its impact on processor architecture. In *Euromicro Conference on Digital System Design*, Aug 2014.

21. L. Kosmidis, R. Vargas, D. Morales, E. Quiones, J. Abella, and F. J. Cazorla. Tasa: Toolchain-agnostic static software randomisation for critical real-time systems. In *International Conference on Computer-Aided Design*, pages 1–8, Nov 2016.
22. S. Law and I. Bate. Achieving appropriate test coverage for reliable measurement-based timing analysis. In *2016 28th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 189–199, July 2016.
23. E. Mezzetti and T. Vardanega. A rapid cache-aware procedure positioning optimization to favor incremental development. In *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 107–116, April 2013.
24. S. Milutinovic, J. Abella, and F. J. Cazorla. Modelling probabilistic cache representativeness in the presence of arbitrary access patterns. In *International Symposium on Real-Time Distributed Computing (ISORC)*, pages 142–149, May 2016.
25. J. Nowotsch, M. Paulitsch, D. Bhler, H. Theiling, S. Wegener, and M. Schmidt. Multi-core Interference-Sensitive WCET Analysis Leveraging Runtime Resource Capacity Enforcement. In *Euromicro Conference on Real-Time Systems*, July 2014.
26. J. Nowotsch, M. Paulitsch, A. Henrichsen, W. Pongratz, and A. Schacht. Monitoring and wcet analysis in cots multi-core-soc-based mixed-criticality systems. In *Conference on Design, Automation & Test in Europe (DATE)*, 2014.
27. M. Paolieri, E. Quiñones, F. J. Cazorla, G. Bernat, and M. Valero. Hardware support forWCET analysis of hard real-time multicore systems. In *36th Annual International Symposium on Computer Architecture*, ISCA, 2009.
28. J. A. Poovey, T. M. Conte, M. Levy, and S. Gal-On. A benchmark characterization of the eembc benchmark suite. *IEEE Micro*, 29(5):18–29, Sept. 2009.
29. S. Schliecker, M. Negrean, G. Nicolescu, P. Paulin, and R. Ernst. Reliable performance analysis of a multicore multithreaded system-on-chip. In *6th international conference on Hardware/Software codesign and system synthesis*.
30. A. Schranzhofer, J.-J. Chen, and L. Thiele. Timing analysis for tdma arbitration in resource sharing systems. In *16th IEEE Real-Time and Embedded Technology and Applications Symposium*, RTAS, 2010.
31. F. Wartel, L. Kosmidis, A. Gogonel, A. Baldovin, Z. Stephenson, B. Triquet, E. Quiñones, C. Lo, E. Mezzetti, I. Broster, J. Abella, L. Cucu-Grosjean, T. Vardanega, and F. J. Cazorla. Timing analysis of an avionics case study on complex hardware/software platforms. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, pages 397–402, 2015.
32. F. Wartel, L. Kosmidis, C. Lo, B. Triquet, E. Quiones, J. Abella, A. Gogonel, A. Baldovin, E. Mezzetti, L. Cucu, T. Vardanega, and F. J. Cazorla. Measurement-based probabilistic timing analysis: Lessons from an integrated-modular avionics case study. In *Int. Symposium on Industrial Embedded Systems*, June 2013.
33. A. West. NASA Study on Flight Software Complexity. Final Report. Technical report, NASA Excellence Program, 2009.
34. R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström. The worst-case execution-time problem: Overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.*, 7(3):36:1–36:53, May 2008.