Modelling Probabilistic Cache Representativeness in the Presence of Arbitrary Access Patterns

Suzana Milutinovic[†,‡], Jaume Abella[†], Francisco J. Cazorla[⋆,†]
[†] Barcelona Supercomputing Center (BSC). Barcelona, Spain
[‡] Universitat Politècnica de Catalunya (UPC). Barcelona, Spain
[⋆] Spanish National Research Council (IIIA-CSIC). Barcelona, Spain

***Abstract*—Measurement-Based Probabilistic Timing Analysis (MBPTA) is a promising powerful industry-friendly method to derive worst-case execution time (WCET) estimates as needed for critical real-time embedded systems. MBPTA performs several (R) runs of the program on the target platform collecting the execution times in each run. MBPTA builds a probabilistic *representativeness* argument on whether those events with high impact on execution time, such as cache misses, arise on the runs made at analysis time so that their impact on execution time is captured. So far only events occurring in cache memories have been shown to challenge providing such representativeness argument.**

In this context, this paper introduces a representativeness validation method (*RVS*) to assess the probabilistic representativeness of MBPTA's execution time observations in terms of cache behaviour. *RVS* resorts to cache simulation to predict worst-case miss scenarios that can appear during the deployment phase. *RVS* also constructs a *probabilistic Worst-Case Miss Count* curve based on the miss-counts captured in the R runs. If that curve upperbounds the impact of the predicted cache worst-case scenarios, R is deemed as a sufficient number of runs for which pWCET estimates can be reliably derived. Otherwise, the user is requested to perform more runs until all cache scenarios of interest are captured.

I. Introduction

Validation and verification of critical real-time systems require providing evidence that system functions will perform correctly and timely. Timing verification is performed by means of timing analysis methods that estimate the worst-case execution time (WCET) of tasks. WCET estimates, which need to be reliable according to the level of confidence defined in the relevant safety standards (i.e. ARP4761 in the avionics domain [12]), must also be as tight as possible so that tasks can be properly scheduled minimising the amount of hardware resources required. However, the increasing complexity of the software and hardware used in critical real-time systems challenges state-of-the-art methods and practices for WCET estimation [3].

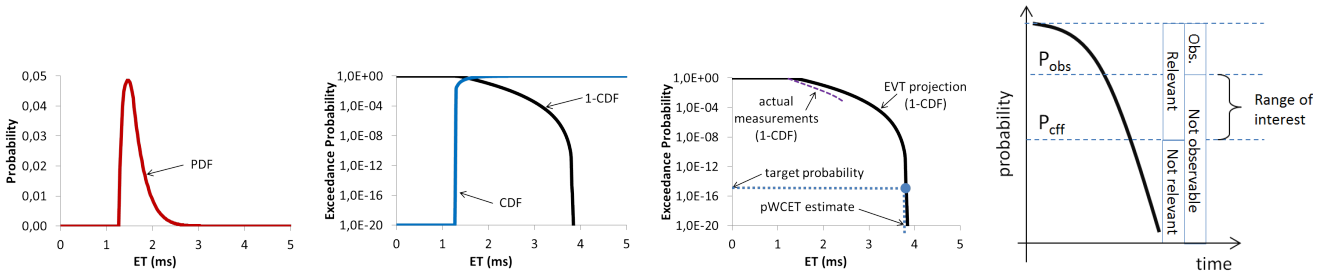
Measurement-Based Probabilistic Timing Analysis (MBPTA) [6] derives probabilistic WCET (pWCET) estimates in the presence of high-performance hardware, e.g. comprising caches. pWCET distributions express the maximum probability with which one instance of the program can exceed a given execution time bound. The WCET value chosen is the one whose exceedance probability is deemed sufficiently low in relation to the integrity level of the functionality being analysed w.r.t. the corresponding safety standard. For instance, in the case of avionics, DAL-A software must not exceed a failure rate of at most 10^{-9} per hour of operation [12].

MBPTA deploys Extreme Value Theory [9], [15] (EVT) to build the pWCET distribution (curve) based on a sample with a limited number of (runs) observations, e.g., $R = 1,000$, collected during the analysis phase. MBPTA requires that the conditions under which measurements are collected during the analysis phase lead to equal or worse timing behaviour than those conditions that can arise at operation [5]. To that end, in MBPTA-compliant processor architectures some sources of execution time variation (jitter) are randomised [5] (e.g., cache placement) so that, if enough runs are performed, the impact of the jitter of those resources in execution time is captured.

In the processor architectures studied so far, the execution time observations used as input for MBPTA capture with high probability the impact of all timing events produced by time-randomised hardware resources except for time-randomised caches (*TRc*) [13]. In particular, set-associative *TRc* deploy random placement with which in each run addresses are randomly mapped to cache sets, defining a random *cache (set) placement*. The execution time of those runs in which the number of addresses (randomly) mapped to a cache set exceeds its associativity (W) can be significantly higher than when this is not the case. The problem arises when those *cache placements of interest* occur with a sufficiently high probability to be deemed as relevant by the corresponding safety standard, but sufficiently low not to be observed in the measurements at analysis time [1], [18], [8]. For instance, for a program accessing 5 addresses, the probability that all of them are randomly mapped to the same set in a 32-set 4-way cache is $10^{-6} \approx (1/32)^4$, hence, of relevance in avionics and automotive. If $R = 1,000$ runs are performed – a typical value used by MBPTA – the probability that at least one run captures the execution time impact of the cache placement of interest where the five addresses are mapped to the same set is very low ($\approx 10^{-3}$), and hence, highly likely not to be captured by MBPTA. As a result MBPTA would fail to upperbound program's execution time. Thus, a reliable MBPTA application requires that those cache placements of interest are sufficiently represented in the measurements passed as input to EVT.

So far only the HoG [1] method has been proposed to attack this problem. However, HoG only works for programs for which the impact on execution time of mapping any subset, bigger than W, of program addresses to a given set is the same. This is in general only the case when program's addresses are accessed mostly in a round-robin

 DOI 10.1109/ISORC.2016.28



(a) Probability distribution function (PDF)

(b) CDF and 1-CDF (logarithmic scale)

(c) Example of pWCET curve

(d) Probability ranges of interest

Fig. 1. Synthetic program PDF, CDF, 1-CDF and pWCET curve.

fashion. However, this is not the general case since access patterns can be arbitrarily complex and irregular.

Contribution. We present *representativeness validation by simulation (RVS)*, a method *valid for any cache access pattern* to assess whether pWCET estimates obtained with MBPTA – for a given number of runs – are reliable. Otherwise *RVS* provides means to determine the number of extra runs needed to regain confidence on MBPTA results. In particular, we make the following contributions:

(1) We present a method based on space exploration searching and cache simulations to determine the worst (random) cache placements, their occurrence probability and impact in terms of miss counts for instruction and data caches.

From the R runs performed by MBPTA, we collect miss counts and derive a probabilistic Worst-Case Miss Count (pWCMC) distribution, a probabilistic prediction of the expected miss counts and their probability based on the information collected in the R runs.

If the pWCMC distribution does not upperbound the worst cache-placement scenarios, they are deemed as not reliably captured in the R runs. In such a case, *RVS* increases the number of runs iteratively until a value R' so that the pWCMC distribution successfully upperbounds those scenarios. At that point, the execution time observations of the R' runs can be used to derive a pWCET estimate that reliably upperbounds the impact of the worst cache-placement scenarios.

(2) We evaluate *RVS* using the EEMBC automotive suite [17]. Our results show that whenever *RVS* requests more runs than those used by MBPTA (i.e $R' > R$), the pWCET curve that would have been obtained with MBPTA's default number of runs (R) might be unreliable and the pWCET curve obtained with the *RVS*'s number of runs (R') can be regarded as reliable. Hence, this makes *RVS* a fundamental step to maintain confidence on MBPTA results. Overall, *RVS* helps increasing MBPTA confidence on obtained pWCET estimates in the presence of caches. Deploying *RVS* is of prominent importance since MBPTA has already been successfully assessed in the context of some industrial case studies [19], [20] and TRc have been already prototyped into FPGAs [11].

II. Input-data representativeness under MBPTA

MBPTA delivers a pWCET distribution function that describes the highest probability (e.g., 10^{-15}) at which one instance of a program may exceed the corresponding execution time bound. This is better understood with the example in Figure 1. Figure 1(a) shows the probability distribution function (PDF) of the execution times collected from $R = 1,000$ runs of a synthetic program running on a MBPTA-compliant platform [14], [5]. The corresponding cumulative distribution function (CDF) and the complementary CDF (1-CDF) are depicted in Figure 1(b) in logarithmic scale. With R observations (execution time measurements), one could estimate the pWCET at an exceedance probability of $1/R$ at most. Since much smaller probabilities are needed in the context of safety-relevant systems, EVT is used to estimate the function that describes the rightmost tail of the execution time distribution. For our example, Figure 1(c) shows the result of applying EVT to estimate the pWCET distribution. The dashed line corresponds to the 1-CDF for the 1,000 measurements collected, whereas the continuous line corresponds to the pWCET distribution.

MBPTA requires that analysis time conditions either match or upperbound the timing impact of those conditions at operation [14], [5] so that by construction the pWCET distribution obtained upperbounds the one at operation. The correct application of MBPTA involves a representativeness step [5] such that *evidence is provided on the fact that analysis time observations capture the impact of those events that can arise at operation and significantly impact execution time and so, pWCET*. These are called *events of interest*, which we refer to as *cache placements of interest* for the case of the cache. MBPTA i) either injects randomisation in the timing behaviour of certain hardware resources (e.g. caches and buses) so that it is possible to determine the probability of their worst behaviour to be captured in the analysis-time measurement runs; or ii) makes resources to work on their worst latency so that the analysis time measurements capture the worst timing behaviour that those resources may have during operation [14].

In building its representativeness argument MBPTA considers two probabilities, as shown in Figure 1(d).

P_{cff}. For random events, MBPTA defines *representativeness* as the requirement by which the impact of any *relevant event* affecting execution time is properly upperbounded at analysis time. Relevant events are those occurring with a probability above a cutoff probability (e.g. $P_{cff} = 10^{-9}$), which is determined by the integrity level of the task and the probability of failure allowed under such integrity level as dictated by the corresponding functional safety standards in the domain (e.g., DAL-A software must not exceed a failure

TABLE I
BASIC NOTATION.

S	Number of sets in cache
W	Number of ways in cache
cl_s	Size in bytes of a cache line
$@_A$ or A	Address assigned to a memory object
$\mathcal{Q}_i$	Sequence of accesses to cache
$@(\mathcal{Q}_i)$	Unique (non-repeated) addresses in $\mathcal{Q}_i$
$\|@(\mathcal{Q}_i)\|$	Number of addresses in $\mathcal{Q}_i$
aC_i	One combination of addresses from $@(\mathcal{Q}_i)$
$\|aC_i\|$	Address count in (i.e cardinality of) aC_i

rate of at most 10^{-9} per hour of operation [12]).

$\mathbf{P_{obs}}$. While those events occurring with overly low probability become irrelevant for pWCET estimation purposes, events occurring with higher probability need to be accounted for, and this requires that their effect is captured in the measurements taken at analysis time (see Figure 1(d)). However, with the number of runs R carried out at analysis, only events with a relatively high probability can be observed in the measurement runs. P_{obs} as presented in Figure 1(d), determines the lowest probability of occurrence of an event such that the probability of not observing it in the analysis time measurements is below a cutoff probability, e.g. 10^{-9}. P_{obs} is a function of the probability of occurrence per run of the event, P_{event}, and the number of runs R (observations) collected by MBPTA at analysis time. For instance, for a cutoff probability of 10^{-9} and $R = 1,000$ runs, we can guarantee that if $P_{event} \geq 0.021$ the event will not be observed with a probability smaller than 10^{-9}, that is, $10^{-9} \geq (1 - 0.021)^{1000}$. It also follows that with a higher number of runs, events with lower probability can be captured.

Overall, the range of probabilities in which relevant events are unlikely to be observed (for $R = 1,000$) is $P_{event} \in [10^{-9}, 0.021]$.

A. Cache-related representativeness challenges

TRc implement random placement by deploying a hardware module to randomly map addresses to sets. The module hashes the address being accessed with a random number (RII) to compute the (random) set where the address is placed [13]. RII remains constant during the program execution so that an address is placed in the same set during the whole execution, but it is randomly changed across executions so that the particular set where an address is placed is also random and independent of the placement for the other addresses across executions. Thus, the probability of any two addresses to be placed in the same set is $1/S$ where S is the number of cache sets.

Heart of Gold (HoG) [1] tackles representativeness issues of cache related events for *TRc*, which were also identified in [18], [8]. In particular, HoG tackles the scenario in which the execution times, obtained from a MBPTA-compliant architecture deploying *TRc* [13], causes MBPTA to yield optimistic pWCET estimates. HoG identifies the cache-related events of interest affecting execution time and determines their probability to occur. In particular, authors in HoG [1] notice that the number of addresses competing for a set is the critical parameter affecting execution time noticeably: whenever up to W addresses are mapped into the same set, those lines end up fitting in the cache set regardless of their access pattern. This occurs because after some random evictions each address can be stored in a different cache line in the set, thus not causing further misses. Conversely, if more than W cache line addresses compete for the cache set space, then they do not fit and evictions will occur often. This scenario is the *cache placement of interest*.

However, HoG relies on the assumption that *the impact of all addresses in execution time is homogeneous*, which happens, for instance, in access sequences in which addresses are accessed in a round robin fashion.

Observation: We make the key appreciation that having more than W addresses mapped to the same set is a necessary condition to trigger a cache placement of interest, but it is not sufficient. Whether that cache placement actually causes an abrupt change on execution time, *depends on the access pattern for those addresses*.

B. Problem statement

We introduce the problem addressed by *RVS* with two illustrative examples. For simplifying the discussion in this section we focus on direct-mapped caches, though in the rest of the paper our focus are set-associative caches. In the first example, the number of misses generated when a subset of addresses is mapped to a set is the same regardless of the particular addresses chosen – as assumed by HoG. In the second example, different conflicting addresses (i.e. addresses mapped to the same set) produce different miss counts – as addressed by *RVS*. We resort to the notation defined in Table I.

Let $\mathcal{Q}_1 = \{ABABABABAB\}$ be a sequence of memory accesses, whose unique addresses are $@(\mathcal{Q}_1) = \{A, B\}$ with $U = |@(\mathcal{Q}_1)| = 2$. Such a sequence may happen when A and B are accessed inside a loop body. For a S-set direct-mapped cache, the probability that, when A and B are randomly mapped to a set, they conflict in the same set is given by $P_{event} = S \times \left(\frac{1}{S}\right)^U$, so $1/S$ in this case. The probability that in the R measurement runs taken at analysis – in each of which a new random set is given to A and B – there is no run in which both are mapped to the same set, $P(s_A = s_B) = P_{event}$, is given by $\overline{P_{event}(R)} = (1 - P_{event})^R$. For $R = 1,000$, a typical value used for MBPTA, the two rows corresponding to $|@(\mathcal{Q}_1)| = 2$ in Figure 2 show P_{event} and $\overline{P_{event}(R)}$ for different values of S representative of typical L1 and L2 caches in real-time systems. Conflictive cache-mapping scenarios are those where $P_{event} \in [10^{-9}, 0.021]$ (for $R = 1,000$), so that the event can occur with a non-negligible probability at operation, and $P_{event} < P_{obs}$, so there is a non-negligible probability of missing this event in the measurements taken at analysis time. We observe that the larger the cache the lower the probability of A and B to conflict in the same set (P_{event}), with MBPTA likely missing the impact of this event when $S \geq 64$ (gray cells).

Let $\mathcal{Q}_2 = (ABABABABABCD)$ be another sequence with $@(\mathcal{Q}_2) = \{A, B, C, D\}$ and $U = 4$. $\mathcal{Q}_2$ may occur when A and B are accessed in a loop and C and D after the loop. HoG [1] assumes that all addresses have the same impact, so it will determine P_{event} as the probability of

		Number of sets (S)									
		8	16	32	64	128	256	512	1024	2048	4096
|@(Q1)|=2	P_{event}	0.125	0.063	0.031	0.016	8E-03	4E-03	2E-03	1E-03	5E-04	2E-04
	$\overline{P_{event}(R)}$	1E-58	9E-29	2E-14	1E-07	4E-04	0.020	0.142	0.376	0.614	0.783
|@(Q2)|=4	P_{event}	0.590	0.333	0.177	0.091	0.046	0.023	0.012	6E-03	3E-03	1E-03
	$\overline{P_{event}(R)}$	9E-388	6E-177	3E-85	3E-42	3E-21	6E-11	8E-06	3E-03	0.053	0.231

Fig. 2. P_{event} and $\overline{P_{event}(R)}$ as a function of S. ($P_{obs} = 0.021$)

any two addresses (e.g AB, AC, AD, BC, BD or CD) to be mapped in the same set (P_{event} should also include the case when 3 or 4 addresses are mapped to the same set). This will lead to the values in the two rows corresponding to $|@(\mathcal{Q}_2)| = 4$ in Figure 2. However, the true cache placement of interest occurs only when A and B are mapped in the same set (AB). In that case, all accesses are misses and otherwise there will be exactly 4 misses (cold misses for the 4 different addresses accessed). Thus, HoG fails to determine P_{event} for $\mathcal{Q}_2$. As a result, for instance, for $S = 256$ HoG determines that the probability of the cache placement of interest is 0.023 – not in the range of interest since it is higher than $0.021 = P_{obs}$– while in reality is $4{\cdot}10^{-3}$, which is in the range $[10^{-9}, 0.021]$. In this scenario more runs are required to provide confidence that the event of interest is captured in the measurements, but HoG fails to capture this situation.

Overall, increasing the confidence on MBPTA results requires *detecting* those cache mappings where $W + 1$ addresses compete for the same cache set with a sufficiently high probability. However, it is important to be *aware of the actual access pattern of the program under analysis* so that only those cache mappings producing a high impact on execution time are considered. In that case, an *action* is required, for instance by increasing the number of runs so a probabilistic argument can be build on the fact that those cache mappings are captured in the measurement runs.

III. RVS

The Representativeness Validation by Simulation (*RVS*) method relies on identifying the *conflictive combinations (set) of addresses*, aC_i so that if they are randomly mapped to the same cache set they lead to cache (set) mapping scenarios with high impact on execution time. *RVS* also estimates (upperbounds) the probability of occurrence of those scenarios and assesses whether the pWCET distribution derived with MBPTA truly upperbounds the impact of those scenarios. The validation is performed in the miss count domain rather than in the execution time domain, and it is applied for all cache memories individually (i.e. instruction and data caches). *RVS* relies on the assumption that miss counts highly correlate with execution time. This is usually the case since cache misses have been shown to be one of the major contributors to programs' execution time. Yet we perform a quantitative assessment of this fact for our reference processor architecture (Section IV). *RVS* includes the following steps:

(1) *RVS* considers all combinations of the most accessed addresses. Part of our future work consists of considering, in a first step, all addresses and quickly discarding those combinations that cannot be the most conflictive ones, i.e those that if mapped to the same set cause a low impact on execution time.

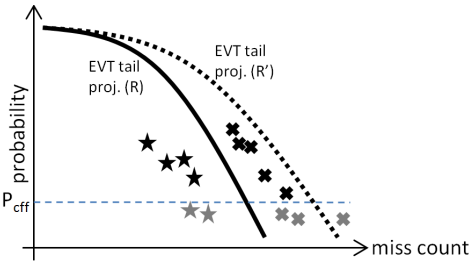


Fig. 3. Illustrative application of *RVS*.

(2) For each combination of addresses regarded as conflictive – and also for each group of combinations – *RVS* (i) determines its probability and (ii) performs a cache simulation in which conflictive addresses are mapped to the same cache set resulting in a number of misses. The probability and miss count information helps *RVS* identifying those conflictive aC_i leading to bad cache mappings that must be upperbounded. *RVS* uses a light-weight cache simulator for *TRc* to estimate the number of misses when a given aC_i is mapped in the same cache set, where $|aC_i| > W$.

(3) *RVS* also performs cache simulations in which all addresses are randomly mapped and applies MBPTA with a default number of runs R. From this information *RVS* generates a probabilistic worst-case miss-count (pWCMC). By validating whether the pWCMC distribution obtained upperbounds all bad cache-set mappings (i.e miss count and probability pairs), *RVS* determines whether the number of runs R used by MBPTA suffices. If this is not the case, more runs are performed until the validation step is passed with $R' \geq R$ runs. Whenever it is passed, the number of runs R' is the minimum number of execution time measurements that MBPTA needs to use.

This is better illustrated in Figure 3, in which the solid curve represents the pWCMC estimate generated from miss counts obtained from R runs and the black stars and black crosses represent the miss counts obtained for all aC_i – and their combinations – whose probability of occurrence is above P_{cff}. Meanwhile their gray counter-parts are those below P_{cff}, which are discarded by *RVS* since their probability of occurrence is deemed as negligible. Stars are those aC_i (and their combinations) whose miss count (i.e. impact) is covered by the pWCMC, while the miss counts of the aC_i marked with crosses are not. In this example scenario, *RVS* requires the user to increase the number of runs from R to R' such that the impact of those aC_i is properly upperbounded. We also observe that when increasing the number of runs to R' runs (with $R' > R$) the resulting pWCMC curve captures the impact of all aC_i. Overall, this results in an increased number of runs R' for which the obtained pWCMC estimate reliably upperbounds the miss count of all aC_i and therefore, the pWCET estimate obtained with R' runs also upperbounds their timing impact.

In the following subsections we describe in detail the steps to apply *RVS*.

A. Generating Combinations of Conflictive Addresses

Let $\mathcal{Q}_i$ be the sequence of accesses under analysis. In theory all aC_i such that $|aC_i| > W$ need to be properly

upperbounded. Those address combinations can be generated from $@(\mathcal{Q}_i)$, and their number is:

$$\sum_{k=W+1}^{U} \binom{U}{k} \quad (1)$$

where $U = |@(\mathcal{Q}_i)|$. Note that the generated aC_i have cardinalities in the range $[W+1, U]$. The key element in this first approach for *RVS* is that we consider U, i.e. the number of unique addresses in $\mathcal{Q}_i$, to be low enough so that all aC_i can be generated and evaluated with the cache conflict simulator. While it is computational intensive, this exhaustive approach brings confidence on the fact that all relevant sets of addresses have been considered. If a lower number of unique addresses $U' < U$ needs to be considered, then we will select those with highest access counts in $\mathcal{Q}_i$.

B. aC_i Impact and Probability

Probability. The probability of a given combination of addresses aC_i to be mapped in the same set is given by:

$$S \times (1/S)^{|aC_i|} \quad (2)$$

Impact. The impact is obtained by performing a Monte-Carlo experiment where each observation is a cache simulation. In each simulation all the addresses in aC_i are forced to be mapped to exactly one random set. The other addresses in $\mathcal{Q}_i$ are mapped randomly. The number of observations (M) needs to be sufficiently high so that the impact of the random mapping of addresses not in aC_i is captured. The impact, i.e. miss count in our case, that is produced for the aC_i is the average miss count under all M mappings. In our experiments we assume $M = 1,000$, which provides a confidence interval of $\pm 2\%$ with 99% confidence. The inputs for the cache conflict simulator include (i) the sequence of cache lines accessed; (ii) aC_i, whose addresses are mapped to the same (random) set, while the rest of the addresses in $\mathcal{Q}_i$ are mapped randomly; and (iii) the cache configuration.

C. Combined aC_i Impact and Probability

Ultimately, *RVS* focuses on determining whether miss count - probability pairs ($< impact, prob >$) are captured by the pWCMC curve: if two combinations of addresses aC_i and aC_j lead to the same miss count impact, the probability of that miss count impact is the union of the probabilities of both combinations of addresses, since when *any* of the two combinations are mapped to the same set, they lead to that miss count. Hence, in addition to considering each combination of addresses (aC_i) in isolation, *it is also needed to determine the joint probability of several* aC_i. For instance, let us consider an example where aC_i and aC_j have the same impact and $|aC_i| = |aC_j|$. The probability of having one of them is $P(aC_i) = P(aC_j) = S \times \left(\frac{1}{S}\right)^{|aC_i|}$, but the probability of having any of them is $P(aC_i \cup aC_j) = P(aC_i) + P(aC_j) - P(aC_i \cap aC_j)$, which is obviously higher than any of their individual probabilities, and the impact will be the same. However, determining the impact and probability of joint scenarios is challenging.

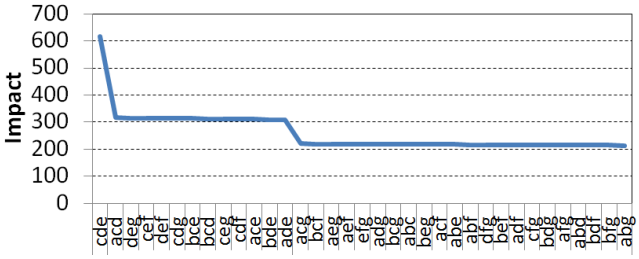


Fig. 4. Impact (miss count) of different aC_i.

Probability. Determining the total probability for the union of any arbitrary number of aC_i is overly complex in practice because we should be able to compute the intersections of each pair of aC_i, each group of three, four, and so on and so forth. Moreover, $P(aC_i)$ and $P(aC_j)$ are not independent in general because addresses may repeat across sets, thus leading to arbitrary intersections for each group. We address this issue by upperbounding union of probabilities as their addition. Note that upperbounding probabilities will lead to an increased risk of not passing the validation step because miss count and probability pairs will be more likely to be above the pWCMC. This, however, may imply collecting more runs than needed but will not lead to false step passes.

Impact. The impact of having any aC_i is obtained as the average of their impacts, since any of them can occur individually with the same probability. Note that individual probabilities for all of them have already been considered, and the case of having aC_i and aC_j simultaneously in the same set has already been considered when analysing those aC_h with larger cardinality such that $|aC_h| = |aC_i \cup aC_j|$.

For each cardinality in the range $[W+1, U]$ we analyse the combined impact of those aC_i with higher individual impact. Since, aC_i with the same cardinality have *exactly* the same probability (see Equation 2), considering those aC_i with highest impact leads to an average impact equal or higher than for any other aC_i group and the same probabilities. As a result of this step we obtain a list of pairs ($< impact, prob >$) describing the impact in misses and the probability of those cache mapping scenarios to be compared against the *pWCMC* curve.

Example. Let us assume the sequence of accesses $\mathcal{Q}_1 = \{ABCDECDECDECDEFG\}$ that repeats since those addresses are in an outer loop with CDE accessed in an inner loop. In this scenario there are 35 different aC_i with cardinality 3, $\{ABC, ABD, ABE, ...\}$. Figure 4 shows the impact when the addresses in each aC_i (shown in the X axis) are forced to be mapped in the same set (in a direct-mapped cache) and the rest are mapped randomly. We observe that $aC_i = \{C, D, E\}$ generates the highest impact. The second step occurs when two addresses of (C, D and E) and another address are mapped into the same set (e.g. $aC_j = \{C, D, F\}$). The lowest step in the impact occurs when only one or none of the three most repeated addresses is in the address combination (e.g. $aC_k = \{C, F, G\}$). Intuitively, what *RVS* needs to capture is the probability – and impact – of each step. As explained, *RVS* will consider incrementally only one aC_i e.g. $\{C, D, E\}$, then combi-

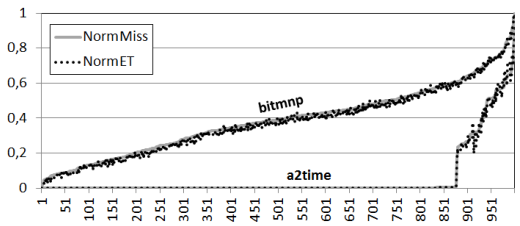


Fig. 5. $NormMiss$ and $NormET$ for `a2time` and `bitmnp` sorted by $NormMiss$.

nations of aC_i, for instance the case where $\{C, D, E\}$ or $\{A, C, D\}$ occur, then $\{C, D, E\}$, $\{A, C, D\}$ or $\{D, E, G\}$, and so on and so forth, thus always considering the worst case for any address set count. Each of these groups will be compared against the pWCMC distribution as illustrated in Figure 3. This step will be repeated for all cardinalities $|aC_i|$ in the range $[W + 1, U]$.

D. Validation against pWCMC

The final step in the application of *RVS* consists in deriving the pWCMC curve by applying EVT to the miss count readings obtained in the R runs performed. Note that, instead of using the miss counts collected from the runs on the target platform, we can use those obtained with the cache simulator since their distribution is the same. In this case, in each simulation we process the sequence of accesses of the program but we do not enforce any group of addresses to be mapped together. Instead, all of them are mapped randomly. Then we generate the pWCMC distribution and compare it against all $< impact, prob >$ obtained before. If any such pair exceeds the pWCMC distribution, we regard the pWCMC as optimistic[1]. In that case we need to increase the number of runs and apply MBPTA again until the pWCMC properly upperbounds all pairs, which will occur eventually since those runs will include measurements with those address combinations placed in the same set when $(1 - P(aC_i))^{R'}$ approaches P_{cff}.

In our case we start this iterative process by setting the value of R' to the number of runs required by MBPTA [6] (R). If more runs are required, we increase the number of runs by $\Delta_R = 10$. Whenever several caches are analysed, the number of runs to be performed is the maximum R' across all caches obtained with *RVS*. In our case we have instruction and data cache so, $R' = max(R'_{dcache}, R'_{icache})$.

IV. Experimental Results

We model an in-order processor with a memory hierarchy composed of first level 4KB 2-way set-associative 32B/line data (DL1) and instruction (IL1) caches and main memory. Both set-associative caches implement random placement and replacement [13]. The latency of an instruction depends on whether the access hits or misses in the instruction cache: a hit has 1-cycle latency and a miss has 100-cycle latency. The memory operations access the data cache so they can last 1 or 100 cycles depending on whether they miss or not. The remaining operations have a fixed execution latency (e.g. integer additions take 1 cycle).

We evaluate several EEMBC automotive benchmarks representative of some safety-related real-time automotive applications [17]. We consider the U = 15 most accessed addresses for instructions and data for each benchmark that covers on average 66.85% of the accesses across all benchmarks. In all cases we start by applying MBPTA with the number of runs R regarded as sufficient by the MBPTA technique for each program [6]. Then we apply our approach, *RVS*, for the instruction and data caches, and obtain the number of runs required to pass the validation step R'.

A. Correlating Execution Time and Miss counts

RVS requires that miss counts and execution times are strongly correlated. While this is generally the case since misses in cache lead to slow off-chip accesses, we perform a quantitative assessment of this fact: we first illustrate such correlation visually for some benchmarks and then we evaluate quantitatively such correlation for the whole set of EEMBC automotive benchmarks. For that purpose we use an FPGA implementation of an in-order processor implementing random placement and replacement caches [11]. Executions on this processor take much longer than the ones on our simple simulator but, as shown next, prove that modelling execution time mostly with cache behavior is an extremely accurate proxy.

Qualitative assessment. First, we perform $R = 1,000$ runs for each benchmark collecting both their execution times and their total number of cache misses (DL1 and IL1 misses). In order to correlate the variation of both metrics we normalize them: for each benchmark we subtract the minimum execution time (miss count) from the execution time (miss count) observed in each experiment. This differential is normalized to the differential between the minimum and maximum values observed. Formally, normalized misses for a given execution i, referred to as $NormMiss_i$, are obtained as follows, where $Miss_i$ stands for the number of misses measured in execution i:

$$NormMiss_i = \frac{Miss_i - \left(MIN_{j=0}^{R} Miss_j\right)}{\left(MAX_{j=0}^{R} Miss_j\right) - \left(MIN_{j=0}^{R} Miss_j\right)} \quad (3)$$

Likewise, we compute $NormET_i$:

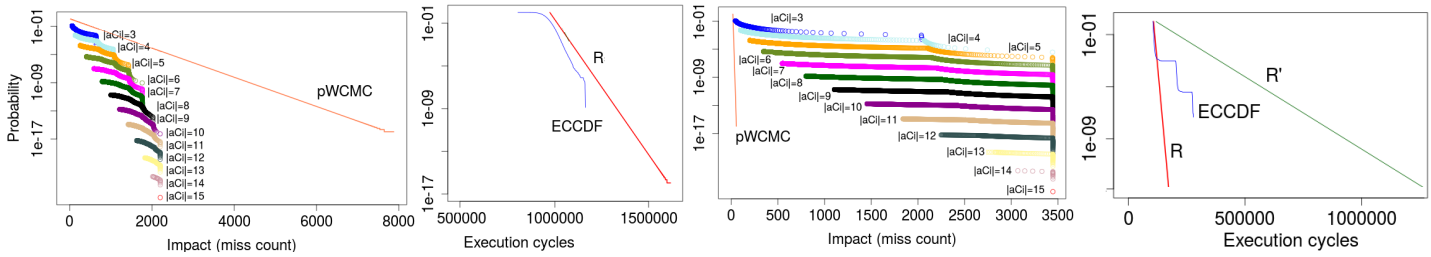
$$NormET_i = \frac{ET_i - \left(MIN_{j=0}^{R} ET_j\right)}{\left(MAX_{j=0}^{R} ET_j\right) - \left(MIN_{j=0}^{R} ET_j\right)} \quad (4)$$

where ET_i is the execution time measured in execution i.

$NormMiss$ and $NormET$ for `a2time` and `bitmnp` benchmarks are shown in Figure 5. As shown, both metrics overlap almost completely. Only some discrepancies are observed for `a2time` due to the effects of the store buffer. However, the average deviation of one metric w.r.t. the other is 0.4% and 1.5% for `a2time` and `bitmnp` respectively.

Quantitative correlation. In order to assess the correlation between miss counts and execution times quantitatively, we have used two different correlation methods

[1]The impact of a pair is obtained within a confidence interval as explained before. Thus, if the pWCMC distribution is within the confidence interval, we cannot reject the hypothesis of such distribution being reliable.



(a) pWCMC for `bitmnp` (b) pWCET for `bitmnp` (c) pWCMC for `aifirf` (d) pWCET for `aifirf`

Fig. 6. *RVS* applied to the instruction accesses of `bitmnp` and `aifirf` (the analysis is performed for combinations of addresses with increasing cardinality, $|aC_i| \in [W+1, U]$) and pWCET estimates obtained with R and R' runs.

to obtain correlation coefficients [16]: Pearson product-moment correlation coefficient and Spearman's rank correlation coefficient. The Pearson product-moment correlation coefficient measures the linear dependence between two variables. Pearson's method delivers as output a value in the range [-1,1], where 1 indicates total positive correlation, 0 no correlation and -1 total negative correlation. In our case we expect values close to 1, meaning that there is a linear positive correlation between execution times and miss counts. Spearman's rank correlation coefficient measures the statistical dependence between two variables by assessing to what extent those variables can be modelled using a monotonic function. As for the Pearson's method, the output value is in the range [-1,1], where 1 indicates total positive correlation, 0 no correlation and -1 total negative correlation. Again, we also expect values close to 1. For both methods we use a 5% significance level (a typical value for this type of tests).

As shown in Table II, all benchmarks obtain very high values for these tests, so miss counts and execution times are highly correlated and such correlation is highly linear (high values for Pearson's test). As shown, in some cases values are not as high as in other cases (although still very high). For instance, this is the case of `aiifft`. We have further analyzed benchmarks with the lowest values and have realized that they experience very low execution time and miss count variations. Thus, other sources of execution time variation, like those introduced by the store buffer, have a relatively higher impact.

TABLE II
PEARSON AND SPEARMAN CORRELATION COEFFICIENTS FOR $NormMiss$ AND $NormET$.

	Pearson	**Spearman**
`a2time`	0.997	0.933
`aifftr`	0.918	0.911
`aifirf`	0.960	0.956
`aiifft`	0.923	0.913
`basefp`	0.999	0.998
`bitmnp`	0.998	0.998
`canrdr`	0.974	0.973
`idctrn`	0.950	0.951

B. RVS results: Illustrative Examples

To illustrate how *RVS* works we present results for one EEMBC Automotive benchmark passing the validation step with R runs (`bitmnp`) and for one requiring extra runs (`aifirf`). For the purpose of this experiment, we perform ten million runs to compute the actual distribution of misses, referred to as ECCDF (Empirical Complementary CDF or 1-CDF) in the following figures. A larger number of runs was not collected due to the cost to run that many simulations. Note that performing that number of runs is not required for *RVS* application, we just perform them for illustrative purposes in this section.

***RVS* passed**. Figure 6(a) shows the result of applying *RVS* for the instruction accesses of `bitmnp`. The curves on the left show the $< impact, prob >$ pairs derived with *RVS* for each cardinality $|aC_i| \in [W+1, U]$. It can be observed that all $< impact, prob >$ pairs are below the pWCMC curve, thus meaning that the number of runs R suffices for a reliable application of MBPTA for this benchmark. This is corroborated in Figure 6(b), where the ECCDF is reliably upperbounded by the pWCET estimate derived with MBPTA with R runs.

***RVS* failed**. In the case of `aifirf`, our method detects that the number of runs obtained with MBPTA $R = 4,400$ is not enough to provide a reliable pWCET estimate. In Figure 6(c) we observe that the pWCMC curve does not upperbound the $< impact, prob >$ pairs generated by *RVS*. As a result in the timing domain, the pWCET estimate derived with R runs does not capture the execution time of the program that is dominated by the misses in cache. *RVS* requires the number of runs to be increased to $R' = 21,390$. If MBPTA is applied in the timing domain on the observations made on R' runs the resulting pWCET estimate is reliable as we can observe in Figure 6(d).

C. RVS results: EEMBC Automotive

Table III summarises for all benchmarks the number of runs required by MBPTA (R) and *RVS* in the miss domain for both DL1 and IL1. The number of runs required by *RVS* is the maximum of the value for first level caches, i.e. DL1 and IL1. That is, $R' = max(R'_{IL1}, R'_{DL1})$. By comparing R' and R we can assess whether the number of runs required by MBPTA in the execution time domain could lead to optimistic pWCET distributions, which occurs when $R < R'$.

We observe that this is the case for all the benchmarks in Table III. This does not mean that results obtained with less than R' are necessarily incorrect, but potentially incorrect. *RVS* keeps the likelihood of missing relevant cache placement scenarios of interest below 10^{-9}, as discussed in Section II. Instead, if we only use the number of runs, R, determined by MBPTA, the likelihood of missing those scenarios becomes much higher (see last column). This decreases the confidence on the results below the levels

TABLE III
RESULTS FOR ALL EEMBC BENCHMARKS.

	RVS				MBPTA	
	R'_{IL1}	R'_{DL1}	R'	likelihood(R')	R	likelihood(R)
a2time	58,360	540	58,360	10^{-9}	2,650	0.390
aifftr	6,840	5,500	6,840	10^{-9}	2,200	0.001
aifirf	21,390	11,530	21,390	10^{-9}	4,400	0.014
aiifft	390	8,770	8,770	10^{-9}	1,900	0.011
basefp	82,080	20,010	82,080	10^{-9}	300	0.927
bitmnp	4,640	3,510	4,640	10^{-9}	850	0.007
canrdr	18,610	7,950	18,610	10^{-9}	350	0.677
idctrn	65,770	47,700	65,770	10^{-9}	3,650	0.317

defined in the corresponding safety standards. Still it can be the case that sometimes relevant scenarios can be observed and, whenever they are not, their effect may be superseded by other processor effects. Although this may result in pWCET estimates truly upperbounding program's execution time, the lack of evidence on this prevents developing sufficient arguments for certification.

V. RELATED WORK

There is a plethora of methods for the estimation of WCET in the real-time domain [21]. Recently MBPTA has emerged as an alternative to obtain WCET estimates with high confidence and to apply industrial practice for complex software running on top of complex hardware [7], [4], [10], [6], [19], [20]. However, it has been shown that MBPTA may lead to optimistic WCET estimates on top of caches implementing random placement in some particular scenarios [1], [18], [8]. Solutions for scenarios where all accessed addresses have the same impact in terms of execution time have been proposed [1]. However, access patterns of programs do not necessarily match such constraint and are typically arbitrary, since addresses are accessed with different frequencies and with arbitrary interleaving. In this paper we tackle this issue by proposing a validation step, *RVS*, able to test whether the WCET estimates obtained with MBPTA are reliable and, if they are not, to increase the number of runs needed until the validation step is passed.

It is worth nothing that so far in the real-time domain EVT has been applied only to execution times [10], [6], [5], while in other domains EVT has been applied to measure flow floods, stock min/max values, etc. In this respect, this paper makes the contribution of extending the use of EVT to other metrics in the real-time domain, in particular to miss counts.

An initial comparison between MBPTA and Static Timing Analysis, which is out of the scope of this paper, has been already performed [2]. Results show that MBPTA provides competitive results with respect to those provided by Static Timing Analysis Techniques.

VI. CONCLUSIONS

MBPTA uses EVT to estimate the pWCET of programs. Some events affecting execution time significantly may occur with a probability sufficiently low so that they may not be observed during the analysis phase. This leads to EVT lacking enough information and, hence, producing optimistic pWCET distributions. While this issue has been solved for programs with homogeneously accessed addresses, access patterns are arbitrary in the general case.

In this paper we introduce a validation step for MBPTA needed to claim reliability of pWCET estimates for arbitrary memory access patterns. Our method, *RVS*, differently to other methods that only work in the execution time domain, performs a validation step in the domain of miss counts. *RVS* identifies the worst miss counts and their probabilities of occurrence and, by means of controlled cache simulations, tests whether pWCET estimates can be regarded as reliable. Our results illustrate the effectiveness of our method. Our future work will focus on reducing the computational cost of *RVS* and generalising it towards more complex architectures.

ACKNOWLEDGMENTS

This work has received funding from the European Community's FP7 programme [FP7/2007-2013] under grant agreement 611085 (PROXIMA, www.proxima-project.eu). Support was also provided by the Ministry of Science and Technology of Spain under contract 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.

REFERENCES

[1] J. Abella et al. Heart of Gold: Making the improbable happen to extend coverage in probabilistic timing analysis. In *ECRTS*, 2014.
[2] J. Abella et al. On the comparison of deterministic and probabilistic WCET estimation techniques. In *ECRTS*, 2014.
[3] J. Abella et al. WCET analysis methods: Pitfalls and challenges on their trustworthiness. In *SIES*, 2015.
[4] G. Bernat and M. Newby. Probabilistic WCET analysis, an approach using copulas. *Journal of Embedded Computing*, 2006.
[5] F.J. Cazorla et al. Upper-bounding program execution time with extreme value theory. In *WCET Workshop*, 2013.
[6] L. Cucu-Grosjean et al. Measurement-based probabilistic timing analysis for multi-path programs. In *ECRTS*, 2012.
[7] J.L Díaz, D.F. Garcia, K. Kim, C.G. Lee, L.L. Bello, López J.M., and O. Mirabella. Stochastic analysis of periodic real-time systems. In *the 23rd IEEE Real-Time Systems Symposium (RTSS02)*, 2002.
[8] Enrico Mezzetti et al. Randomized caches can be pretty useful to hard real-time systems. *LITES*, 2(1), 2015.
[9] W. Feller. *An introduction to Probability Theory and Its Applications*. 1996.
[10] J. Hansen et al. Statistical-based WCET estimation and validation. In *WCET Analysis workshop*, 2009.
[11] C. Hernandez et al. Towards making a LEON3 multicore compatible with probabilistic timing analysis. In *DASIA*, 2015.
[12] SAE International. ARP4761: Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment.
[13] L. Kosmidis et al. A cache design for probabilistically analysable real-time systems. In *DATE*, 2013.
[14] L. Kosmidis et al. Probabilistic timing analysis and its impact on processor architecture. In *DSD*, 2014.
[15] S. Kotz et al. *Extreme value distributions: theory and applications*. World Scientific, 2000.
[16] D.A. Wolfe M. Hollander. *Nonparametric statistical methods*. 1973.
[17] Jason Poovey. *Characterization of the EEMBC Benchmark Suite*. North Carolina State University, 2007.
[18] J. Reineke. Randomized caches considered harmful in hard real-time systems. *LITES*, 1(1), 2014.
[19] F. Wartel et al. Measurement-based probabilistic timing analysis: Lessons from an integrated-modular avionics case study. In *SIES*, 2013.
[20] F. Wartel et al. Timing analysis of an avionics case study on complex hardware/software platforms. In *SIES*, 2013.
[21] R. Wilhelm et al. The worst-case execution time problem: overview of methods and survey of tools. *ACM TECS*, 7(3):1–53, 2008.