Milutinovic *et al. EURASIP Journal on Embedded Systems* (2017) 2017:28
DOI 10.1186/s13639-017-0076-8

EURASIP Journal on Embedded Systems

RESEARCH Open Access

On the assessment of probabilistic WCET estimates reliability for arbitrary programs

Suzana Milutinovic[1,2*], Jaume Abella[1] and Francisco J. Cazorla[1,3]

Abstract

Measurement-Based Probabilistic Timing Analysis (MBPTA) has been shown to be an industrially viable method to estimate the Worst-Case Execution Time (WCET) of real-time programs running on processors including several high-performance features. MBPTA requires hardware/software support so that program's execution time, and so its WCET, has a probabilistic behaviour and can be modelled with probabilistic and statistic methods. MBPTA also requires that those events with high impact on execution time are properly captured in the (*R*) runs made at analysis time. Thus, a *representativeness* argument is needed to provide evidence that those events have been captured. This paper addresses the MBPTA representativeness problems caused by set-associative caches and presents a novel representativeness validation method (ReVS) for cache placement. Building on cache simulation, ReVS explores the probability and impact (miss count) of those cache placements that can occur during operation. ReVS determines the number of runs *R*′, which can be higher than *R*, such that those cache placements with the highest impact are effectively observed in the analysis runs, and hence, MBPTA can be reliably applied to estimate the WCET.

Keywords: Real time, WCET, Probabilistic timing analysis, Verification, Safety

1 Introduction

The validation and verification (V&V) process for critical real-time systems requires collecting *sufficient evidence* that critical functions will execute correctly and timely. In this context, the term *sufficient evidence* relates to the corresponding functional safety standard and the integrity level of the task analysed. Timing V&V, the focus of this paper, comprises estimating the Worst-Case Execution Time (WCET) of tasks with appropriate methods and tools and providing evidence that they can be scheduled into their allocated time budgets. In industrial environments, several factors determine the WCET analysis tool/technique to use. First, achieving enough confidence in WCET estimates according to the relevant safety standards (e.g. ARP4761 in the avionics domain [1] and ISO26262 in the automotive domain [2]). Second, obtaining WCET estimates as tight as possible so tasks can be successfully scheduled while minimising the amount of hardware resources required. And third, keeping the overheads incurred to apply the timing analysis technique as low as possible to keep the competitive edge.

The increasing complexity of the software and hardware used in critical real-time systems affects all three factors and challenges state-of-the-art methods and practices for WCET estimation. In this paper, we focus on Measurement-Based Timing Analysis (MBTA), the most used technique across domains such as automotive, railway, space and avionics [3]; and that is applied to the highest criticality software, e.g. DAL-A software in avionics [4]. MBTA usually captures the high watermark execution time and adds to it an engineering margin to account for the unknown. The reliability of this margin depends on user's ability to create test scenarios representative of those that can occur during system operation. This, in turn, builds on user's experience and control of those elements impacting application's execution time. The latter is challenged by the presence of complex hardware/software with massive interactions among components with non-obvious impact on timing, ultimately decreasing the confidence on MBTA's derived WCET estimates.

*Correspondence: suzana.milutinovic@bsc.es
[1]Barcelona Supercomputing Center (BSC), Barcelona, Spain
Full list of author information is available at the end of the article

Measurement-Based Probabilistic Timing Analysis (MBPTA) [5] is a probabilistic variant of MBTA that aims at keeping MBTA's low cost/benefit ratio while increasing guarantees on WCET estimates in the presence of complex hardware. To that end, MBPTA combines probabilistic/statistical timing analysis and two techniques to control jitter: the randomisation of the timing behaviour of some (hardware and software) resources and forcing some resources to work in their worst latency. MBPTA derives probabilistic WCET (pWCET) estimates, a distribution that expresses the maximum probability with which one instance of the program can exceed a given execution time bound.

MBPTA deploys Extreme Value Theory [6, 7] (EVT) to build a pWCET distribution (curve) based on a sample with a limited number of observations (runs), e.g. $R = 1000$, collected during the analysis phase. MBPTA requires that some sources of execution time variation (jitter) are randomised [8] (e.g. cache placement) so that, if enough runs are performed, the impact of their jitter in execution time is captured. This principle emanates from probabilistic and statistics theory, where a random variable can be modelled based on a sample of observations with increasing confidence and accuracy as the size of the sample grows. For MBPTA, the platform designed together with the measurement collection method makes the worst-case timing behaviour of the task under analysis be described by a random variable.

Determining the number of runs required by MBPTA (R) is challenged by the use of random placement in caches. Set-associative (and direct-mapped) Time-Randomised Caches (TRC) [9] deploy random placement, which makes each address to be mapped to a random and independent set across program runs. Therefore, each run results in 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 [10]. This fact becomes an issue for MBPTA when those *cache placements of interest* occur with a sufficiently high probability to be deemed as relevant by the corresponding safety standard (e.g. above 10^{-9}), but sufficiently low not to be observed in the measurements at analysis time (e.g. below 10^{-3}) [10–12]. In these cases, MBPTA could not capture the impact of this event on the program's execution time. Thus, evidence that those cache placements of interest are sufficiently represented in the measurements passed as input to EVT is needed to have enough confidence in MBPTA results.

So far, only the Heart of Gold (HoG) [10] method and its extensions [13, 14] have been proposed to tackle this problem. However, those solutions only work 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 fashion. However, this is not the general case since access patterns can be arbitrarily complex and irregular.

Contribution. We present *Representativeness Validation by Simulation (ReVS)*, a method *valid for arbitrary cache access patterns* to assess whether pWCET estimates obtained with MBPTA—for a given number of runs—are reliable. Otherwise, ReVS provides means to determine the number of extra runs needed.

In particular, we make the following contributions:

1. We present a method based on cache simulations to explore the space of cache random placements and determine those ones leading to the highest execution times at different exceedance probability thresholds. In particular, we identify their probability of occurrence and their impact in terms of miss count for instruction and data caches. By applying MBPTA on the R miss counts collected from the program by means of simulation, we derive a probabilistic Worst-Case Miss Count (pWCMC) distribution—an upper-bound of the miss count distribution of the program under analysis[1].
2. If the pWCMC distribution does not upperbound the worst cache-placement scenarios, ReVS increases the number of runs iteratively until a value R' so that the pWCMC distribution successfully upper-bounds those scenarios. At that point, the execution time observations with R' runs can be used to derive a pWCET estimate that reliably upperbounds the impact of the worst cache-placement scenarios.
3. ReVS determines R' based on the analysis of the most accessed addresses in the program, whose number is limited based on the affordable computational cost. In order to understand the impact of dismissing the least frequently accessed addresses, we provide a qualitative analysis together with a quantitative assessment by comparing the results of ReVS for different number of addresses considered.
4. We evaluate ReVS using the Embedded Microprocessor Benchmark Consortium (EEMBC) automotive suite [15]. Our results show that, differently to the default application of MBPTA, ReVS allows increasing confidence up to a given user-defined threshold (e.g. 10^{-9}) by increasing the number of runs whenever needed.

Overall, ReVS allows controlling the confidence level of pWCET estimates in the presence of caches. Deploying ReVS is of prominent importance since MBPTA

has already been successfully assessed in the context of some industrial case studies [16] and time-randomised cache (TRC) has been already prototyped into field-programmable gate arrays (FPGAs) [17].

2 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 Fig. 1. Figure 1a shows the probability distribution function (PDF) of the execution times collected from $R = 1000$ runs of a synthetic program running on a MBPTA-compliant platform [8]. The corresponding cumulative distribution function (CDF) and the complementary CDF (1-CDF) are depicted in Fig. 1b in logarithmic scale. With R observations (execution time measurements), one could accurately estimate the pWCET at an exceedance probability of $1/R$ at most. Since much smaller probabilities are needed in the context of critical real-time systems, EVT is used to estimate the function that describes the rightmost tail of the execution time distribution. Figure 1c shows the result of applying EVT to estimate the pWCET distribution in our example. The dashed line corresponds to the 1-CDF for the 1000 measurements collected, whereas the continuous line corresponds to the pWCET distribution.

MBPTA requires that execution conditions for tests performed at analysis time lead to execution times that match or upperbound those during system operation [8]. To that end, a reliable MBPTA application requires a representativeness step [10]. Such step is intended to provide evidence that analysis time observations capture the impact of the events that can arise during operation and have a significant impact on execution time and so, on the pWCET. These events are called *events of interest*, which we refer to as *cache placements of interest* for the case of the cache. To reach this goal, MBPTA-compliant platforms either (i) randomise the timing behaviour of certain hardware resources (e.g. caches [9]) so that each potential behaviour occurs with a probability or (ii) make resources to work on their worst latency during the analysis phase [8]. Both techniques, randomisation and upperbounding, are applied so that the execution time distribution during analysis upperbounds the one during operation. In building its representativeness argument, MBPTA considers two probabilities, as shown in Fig. 1d.

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. $P_{\text{cff}} = 10^{-9}$). Such cutoff probability relates to what the corresponding functional safety standard describes as reasonable or unreasonable risk. Based on the hazard analysis and risk assessment of the particular functionality implemented by the task, one can determine an appropriate probability threshold (P_{cff}). For instance, if $P_{\text{cff}} = 10^{-9}$ and a given event occurs with 0.9 probability, the probability of not observing it in ten trials would be 10^{-10} and hence irrelevant in this context. In other words, the risk of missing this event with ten trials is not unreasonable.

P_{obs}. Relevant events, whose probability is above P_{cff}, need to be accounted for pWCET estimation. This requires that their effect is captured in the measurements collected at analysis time (see Fig. 1d). However, given a 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 Fig. 1d, determines the lowest probability of occurrence of an event such that the probability of not observing it in the analysis time measurements is below the cutoff probability, P_{cff}. 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 $R = 1000$ runs, we can guarantee that, if $P_{\text{event}} \geq 0.021$, 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

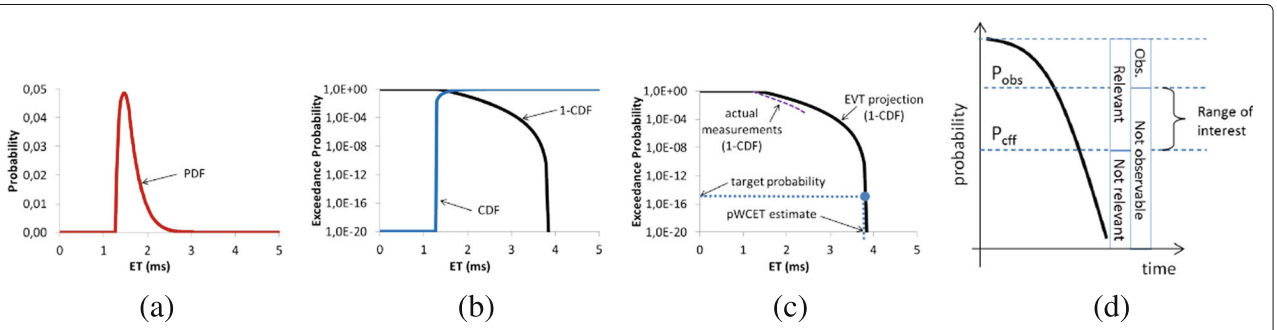

Fig. 1 Synthetic program PDF, CDF, 1-CDF and pWCET curve. **a** Probability distribution fuction (PDF). **b** CDF and 1-CDF (logarithmic scale). **c** Example of pWCET curve. **d** Probability ranges of interest

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 $R = 1000$) is $P_{\text{event}} \in [10^{-9}, 0.021]$. Authors in [10] identified that random cache placement events can be in that range and, hence, can affect the representativeness of MBPTA pWCET estimates.

2.1 Cache-related representativeness challenges

TRC implement random placement with a hardware module that maps addresses to set randomly and independently. The module hashes the address being accessed with a random number to compute the (random) set where the address is placed [9]. The random number 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.

HoG [10] tackles representativeness issues of cache-related events for TRC, which were also identified in [11, 12]. HoG identifies the cache-related events of interest affecting execution time and determines their probability to occur. In particular, authors in HoG [10] 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. HoG has been improved to provide precise probabilities rather than approximations [13]; further, some initial works also consider software time-randomised caches rather than only hardware time-randomised ones [14]. Still, those works build upon the same assumption as HoG: the impact of all addresses in execution time is homogeneous.

We make the observation 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 such a cache placement causes an abrupt increase of the execution time depends on the access pattern for those addresses.

HoG, as well as our proposal, ReVS, operates at cache line address granularity since cache lines are allocated and evicted atomically, so different addresses belonging to the same cache line address are regarded as the same address for the application of ReVS. In the rest of the paper, we use "address" and "cache line address" interchangeably to refer to cache line address.

2.2 Problem statement

We introduce the problem addressed by ReVS 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 ReVS. We resort to the notation defined in Table 1.

Let $\mathcal{Q}_1 = \{ABABABABAB\}$ be a sequence of memory accesses, whose unique (cache line) 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 an S-set direct-mapped cache, the probability that A and B are randomly mapped to the same set is given by $P_{\text{event}} = S \times \left(\frac{1}{S}\right)^U$, 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, $P(s_A = s_B) = P_{\text{event}}$, is given by $\overline{P_{\text{event}}(R)} = (1 - P_{\text{event}})^R$. For $R = 1000$, a typical value used for MBPTA, the two rows corresponding to $|@(\mathcal{Q}_1)| = 2$ in Fig. 2 show P_{event} and $\overline{P_{\text{event}}(R)}$ for different values of S representative of typical L1 and L2 caches in real-time systems. Conflictive cache-mapping scenarios are those where $P_{\text{event}} \in [10^{-9}, 0.021]$ (for $R = 1,000$), so that the event can occur with a non-negligible probability during operation, and

Table 1 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 $\mathcal{Q}_i$
$\lvert@(\mathcal{Q}_i)\rvert$	Number of addresses in $\mathcal{Q}_i$
aC_i	One combination of addresses from $@(\mathcal{Q}_i)$
$\lvert aC_i\rvert$	Address count in (i.e. cardinality of) aC_i

		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. ($P_{obs} = 0.021$)

there is a non-negligible probability of missing this event in the measurements taken at analysis time. We observe that the larger the cache is, 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 $S \geq 64$ (grey 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 [10] assumes that all addresses have the same impact, so it will determine P_{event} as the probability of *any two addresses* (i.e. *AB*, *AC*, *AD*, *BC*, *BD* and *CD*) to be mapped in the same set, plus the probability of three addresses to be mapped in the same set (i.e. *ABC*, *ABD*, *BCD*), plus the probability of the four addresses to be mapped in the same set (i.e. *ABCD*). This will lead to the values in the two rows corresponding to $|@(\mathcal{Q}_2)| = 4$ in Fig. 2. However, the true cache placement of interest occurs only when A and B are mapped in the same set (*AB*, *ABC*, *ABD*, *ABCD*). In that case, all accesses are misses and otherwise there will be exactly four misses (cold misses for the four different addresses accessed). Hence, in this case, 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, which is not in the range of interest since it is higher than P_{obs} (0.021). In reality, it is 4×10^{-3}, which falls in the range of interest: $[10^{-9}, 0.021]$. In this scenario, more runs are required to provide enough confidence in capturing the event of interest in the measurements, but HoG fails to identify this situation.

Overall, *providing evidence* that those cache mappings where at least $W + 1$ addresses compete for the same cache set have been observed with a sufficiently high probability increases evidence on whether cache jitter is captured with MBPTA. This requires being aware of the actual access pattern of the program under analysis so that only those cache placements producing a high impact on execution time are considered. If the probability of missing any such cache placement is too high, the number of runs needed at analysis needs to be increased so that confidence on observing those placements is high enough.

3 ReVS: a high-level description

The Representativeness Validation by Simulation (ReVS) method identifies the (conflictive) sets of (cache line) addresses, aC_i, with high impact on execution time when they are randomly mapped to the same cache set. ReVS also tightly upperbounds the probability of occurrence of those scenarios and assesses whether the pWCET distribution derived with MBPTA upperbounds their impact. The validation is performed in the miss count domain rather than in the execution time domain, and it is applied for each cache memory individually (i.e. instruction and data caches). ReVS relies on miss counts correlating with execution time. While this is usually the case since cache misses have been shown to be one of the major contributors to programs' execution time, we perform a quantitative assessment for our reference processor architecture (Section 6). Whenever this is not the case, then, the impact of cache misses can be disregarded since jitter due to other resources is much larger, and therefore, ReVS would not be needed. In that case, the default number of runs R would suffice for a reliable application of MBPTA since, as pointed out in [10]; so far, only cache placement events have been shown to challenge MBPTA reliability. Still, as long as cache misses are one of the main sources of jitter, the use of ReVS is mandatory for a reliable application of MBPTA.

3.1 ReVS main steps

ReVS includes the following steps:

1. Due to the computational cost of ReVS, only the U most accessed (cache line) addresses are kept in the address trace (from which ReVS considers all potential combinations). Part of our future work consists of considering, in a first step, all (cache line) 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. This will allow considering arbitrarily large and complex programs.
2. For each combination of addresses regarded as conflictive—and also for each group of combinations—ReVS (i) determines its probability

and (ii) performs cache simulations in which conflictive addresses are mapped to the same cache set. The probability (obtained analytically) and miss count information (obtained through simulation) allow ReVS identifying those conflictive aC_i leading to conflictive cache placements that must be upperbounded. ReVS 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. ReVS also performs cache simulations in which all addresses are randomly mapped and applies MBPTA with a default number of runs R. ReVS generates a probabilistic worst-case miss-count (pWCMC) with these miss measurements. By validating whether the pWCMC distribution obtained upperbounds all conflictive cache set mappings (i.e. miss count and probability pairs), ReVS 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.

3.2 ReVS process

ReVS process is illustrated in Fig. 3. The solid curve represents the pWCMC estimate generated from the 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}. Their grey counterparts are those below P_{cff}, which is discarded by ReVS since their probability of occurrence is deemed as irrelevant. Stars are those aC_i (and their combinations) whose miss counts are upperbounded by the pWCMC, while the miss counts of the aC_i marked with crosses are not. In this case, ReVS requires increasing the number of runs, from R to R', such that the impact of those aC_i is properly upperbounded. As shown, the resulting pWCMC curve with R' runs upperbounds the impact of all aC_i. Therefore, the pWCET estimate obtained with R' runs upperbounds the timing impact of all cache placements of interest with sufficient confidence.

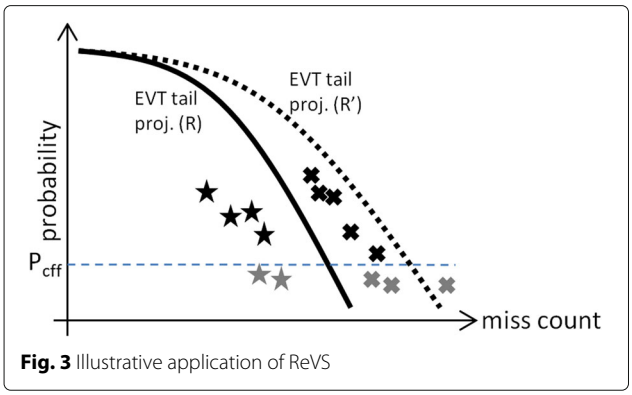

Fig. 3 Illustrative application of ReVS

3.3 An illustrative example

Let us assume a loop that contains the following sequence of accesses $\mathcal{Q}_1 = \{ABCDECDECDECDEFG\}$, so that it repeats. In this scenario, there are 35 different aC_i with cardinality 3, $\{ABC, ABD, ABE, \ldots\}$. 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 $< (C, D, E) >$ and any other addresses are mapped into the same set (e.g. $aC_j = \{C, D, F\}$). The lowest step in terms of 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 ReVS needs to capture is the probability and impact of each step. ReVS will consider incrementally only one aC_i, e.g. $\{C, D, E\}$, then combinations 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 set of combinations and obtaining the corresponding $<$ probability, impact $>$ pairs. Each of these pairs will be compared against the pWCMC distribution as illustrated in Fig. 3. This step will be repeated for all cardinalities $|aC_i|$ in the range $[W + 1, U]$.

4 ReVS detailed steps

In this section, we describe in detail ReVS' main application steps.

4.1 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. The number of those address combinations is computed as shown in Eq. 1, with $U = |@(\mathcal{Q}_i)|$. Note that the generated aC_i has cardinalities in the range $[W + 1, U]$.

$$N_{aCi}^{\mathcal{Q}_i} = \sum_{k=W+1}^{U} \binom{U}{k} \tag{1}$$

Ideally, we would like to consider all addresses in the program under analysis, U, i.e. the number of unique addresses in $\mathcal{Q}_i$. However, computation costs to generate all combinations and simulate them in the cache conflict simulator may limit the actual number of addresses that can be considered to be up to U', where $U' < U$. This may

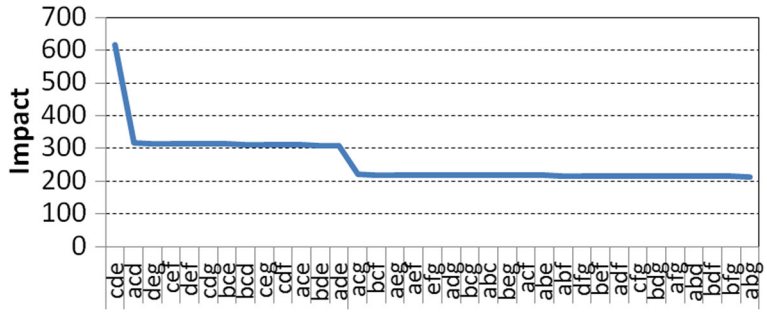

Fig. 4 Impact (miss count) of different aC_i

have an effect on the minimum number of runs R' provided by ReVS. How the address count U' impacts R', and so the confidence of the pWCET estimates, is discussed in detail later in Section 5.

4.2 aC_i Impact and probability

Probability. The probability of a given combination of addresses aC_i to be mapped in the same set is shown in Eq. 2. The probability of one address to be mapped in a specific set is $1/S$, and so the probability of mapping $|aC_i|$ addresses in a specific set is $(1/S)^{|aC_i|}$. Since there are S sets in cache, this probability needs to be multiplied by S.

$$\text{prob}_{\text{same-set}}(aC_i) = S \times (1/S)^{|aC_i|} \tag{2}$$

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[2]. In our experiments, we assume $M = 1000$, 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. While this paper relies on random placement as well as random replacement, the latter—although convenient—it is not strictly needed. Instead, other replacement policies could be used (e.g. LRU or pseudo-LRU). This could change the impact of the different address combinations. However, ReVS would be applied exactly in the same way. Studying the impact on R' and the pWCET estimates of other replacement policies is beyond the scope of this work.

4.3 Combined aC_i impact and probability

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 $|aC_i| = |aC_j|$. Their individual probabilities are $P(aC_i) = P(aC_j) = S \times \left(\frac{1}{S}\right)^{|aC_i|}$, but the probability of having *exactly* one of them is $P(aC_i \cup aC_j) = P(aC_i) + P(aC_j) - P(aC_i \cap aC_j)$, whereas the impact will be the same. In general, determining the impact and probability of joint scenarios is challenging.

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. Note that, $P(aC_i)$ and $P(aC_j)$ are not mutually exclusive in general because addresses may repeat across sets, thus leading to arbitrary intersections for each group. We address this issue by upperbounding such union of probabilities as their addition. Note that this choice may 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.

In very extreme cases, some $<$ impact, prob $>$ pairs could be set with such a high probability that the pWCMC never upperbounds them, thus leading to a failure to pass ReVS. However, false step passes cannot occur. Note also that pairs are never disregarded even if they reach P_{obs} (0.021). However, in practice, we would need extremely tiny caches and large address counts to reach P_{obs}.

Impact. The impact of having any two aC_i or aC_j is obtained as the average of their impacts, since either 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 is captured 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. Conceptually, this can be implemented sorting the different aC_i from the highest to lowest impact, and selecting those two combinations with the highest impact, then the three with the highest impact and so on and so forth. For instance, if we have the following four combinations $aC_1 = \{A, B, C\}$, $aC_2 = \{C, D, E\}$, $aC_3 = \{F, G, H\}$ and $aC_4 = \{A, D, J\}$ for $K = 3$, with their respective $<$ impact, prob $>$ pairs $< 100, 0.0001 >$, $< 70, 0.0001 >$, $< 90, 0.0001 >$ and $< 80, 0.0001 >$, we would sort them and obtain: aC_1, aC_3, aC_4, aC_2. aC_1 in isolation has already been considered as an individual combination. We now consider groups of two, three and four combinations. The group of two combinations includes aC_1 and aC_3. Its $<$ impact, prob $>$ pair would be $< 95, 0.0002 >$, thus reflecting the average impact and the added probabilities. The group of three combinations includes aC_1, aC_3 and aC_4 and would be represented with the pair $< 90, 0.0003 >$. The group with four combinations includes all of them and is represented with the pair $< 85, 0.0004 >$. Note that there is no way to select groups of two, three or four combinations with the highest average impact than the ones chosen, and their probabilities would be exactly the same since all combinations with the same number of addresses have identical probabilities. This step delivers a list of pairs ($<$ impact, prob $>$) that must be upperbounded by the pWCMC curve.

4.4 Validation against pWCMC

The final step consists in collecting the miss counts for R runs without enforcing any specific placement, so that all addresses are mapped randomly. Then, MBPTA is applied to obtain the pWCMC curve. Those R runs can be performed, for instance, in the same simulator where the $<$ impact, prob $>$ pairs have been obtained. Finally, those pairs are compared against the pWCMC curve.

4.4.1 Outcomes of the validation

Different scenarios may arise for the set of $<$ impact, prob $>$ pairs when assessing them against the pWCMC.

- **Step passed**. If all pairs $<$ impact, prob $>$ are upperbounded by the pWCMC curve, or the curve falls within their confidence interval; then, it can be argued that R runs account for all relevant cache placements. Similarly to any statistical approach, there is some chance that the actual impact of a particular cache placement is larger than estimated simply because it is above the confidence interval estimated. In this case, we make the following considerations: (1) Safety functional standards accept confidence levels of 99% even for the highest safety integrity levels. For instance, the verification of hardware design in terms of single-point fault metric in the context of ISO26262 in the automotive domain sets the coverage threshold at 99% for the highest Automotive Safety Integrity Level (ASIL D) [2] in clause 8.4.5. (2) The probability of being above is very low ($<$ 1% due to the 99% confidence). Due to the Gaussian distribution produced by the Monte-Carlo experiments, the probability could only be above with decreasing probabilities. Therefore, the actual impact is very unlikely to be above and, if it was, it should be naturally very close to the confidence interval estimated. Therefore, the evidence obtained with this process is in line with industrial practice since pWCMC reliability is proven to be probabilistically high.
- **Step failed**. If the pWCMC is below the confidence interval for at least one pair, ReVS asks for more runs. However, there is a risk of having a false positive despite the number of runs already suffices to upperbound all pairs. For instance, the Monte-Carlo experiment may produce, by chance, a particular cache placement with very high impact but that occurs with very low probability. Such placement, if observed, may shift the confidence interval towards higher impact values, thus making the pWCMC to be below the confidence interval for this pair. In this case, the cost of the false positive relates to asking the end user for more execution time measurements of his program, but it does not decrease the reliability of the method.

4.4.2 Determining the number of runs

ReVS starts an iterative process by setting the value of R' to the number of runs required by MBPTA [5] (R). If more runs are required (i.e. pWCMC does not upperbound all pairs), we increase the number of runs by $\Delta_R = 10$. As the number of runs R' increases, we also increase Δ_R accordingly for efficiency. That is, we make $\Delta_R = 100$ when $R' > 1000$ runs, $\Delta_R = 1000$ for $R' > 10,000$ runs, and so on and so forth. Whenever a value of R' is found such that the pWCMC curve upperbounds all pairs, then we explore the interval in steps of $\Delta_R = 10$ to provide a precise answer, although this last step is not strictly needed.

Whenever several caches are analysed, the number of runs to be performed is the maximum R' across all caches obtained with ReVS. In our case, we have instruction and data cache so, $R' = \max(R'_{\text{dcache}}, R'_{\text{icache}})$. As miss counts in the instruction and data caches are independent events, it is sufficient to observe their set of worst address

placements separately (those leading to high miss counts). EVT is in charge of predicting the impact and probability of the different bad address placements for data and instructions to occur simultaneously. Note that using the maximum R' across all caches may be pessimistic due to several reasons and a lower value for R' could suffice:

- It could be the case that, for instance, $R'_{\text{icache}} < R'_{\text{dcache}}$ and IL1 placements observed with R'_{icache} runs lead to higher pWCET estimates than those DL1 placements observed with R'_{dcache} runs. In that case, $R' = R'_{\text{icache}}$ would suffice. However, building such a proof is complex so we resort to using the maximum R' across all caches for reliability of the method.
- The pWCET value will be chosen at a specific probability threshold (e.g. at 10^{-12} per run). Therefore, it may be completely irrelevant that the pWCET value at higher probabilities (e.g. at 10^{-6} per run) is not a true upperbound as long as all relevant events are conveniently upperbounded with the pWCET selected. For instance, the pWCET at 10^{-6} could be 100,000 cycles with $R < R'$ runs, whereas execution times of 101,000 cycles could occur at this probability. However, if the pWCET selected at 10^{-12} with R runs is 150,000 cycles and the highest execution time that can occur at that probability is 145,000 cycles; then, the pWCET estimate is reliable despite using only R runs.

5 Reliability considerations of ReVS

The computational cost of ReVS prevents us to apply it to all program addresses. We limit our application of ReVS to the $U = 15$ most accessed cache lines (which represent $15 \times 8 = 120$ addresses), which results in a computational cost of around 1.5 h per cache and per benchmark (for the suite used in this work) on a regular laptop. For the benchmarks used in this work (see further details in Section 6), those cache line addresses account for 67% of all the memory accesses. In Figs. 5 and 6, we show, for instructions and data respectively, the percentage of the total program's accesses (assuming 32-byte cache lines), covered by the most accessed cache line addresses. We observe the variable behaviour across benchmarks, especially with respect to instruction access coverage. For some benchmarks, the 15 most accessed addresses are sufficient to achieve very high coverage (e.g. `canrdr`, with $>$ 95% instruction and $>$ 75% data accesses covered), while for others, we observe much lower coverage (e.g. `aifftr` and `aiifft`, with $<$ 20% instruction accesses covered).

Using higher number of addresses (e.g. $U = 20$) increases the computational cost of ReVS exponentially due to the exponential increase in the number of address combinations to explore. For instance, we have determined analytically and empirically that this cost would increase by 30$\times$ when moving from $U = 15$ to $U = 20$. Thus, increasing U would only be affordable decreasing the number of cache simulations per combination (e.g. from 1000 down to 100), which would lead to much wider confidence intervals for the $<$ impact, prob $>$ pairs. This would challenge the usefulness of ReVS since too wide intervals would make almost any pWCMC to fall within the interval, thus failing to identify those cases where the number of runs is too low to capture relevant placements.

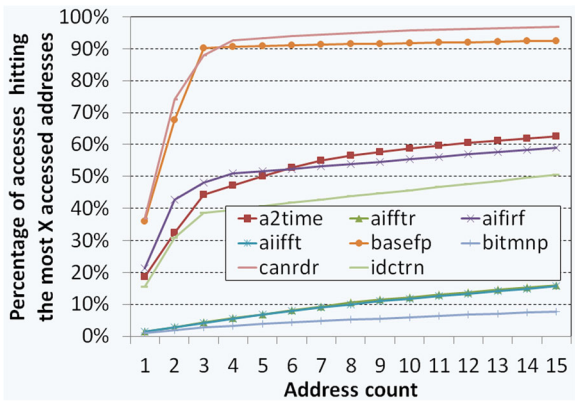

Fig. 5 Instruction accesses coverage

While part of our current work is to reduce the computational requirements of ReVS, in this section, we provide an analysis of the potential impact on confidence of dismissing some addresses.

5.1 Impact of address choice

As a rule of thumb, the most accessed addresses are the ones able to create a higher miss variability with different cache placements and hence higher execution time variability. This is typically the case since the most accessed

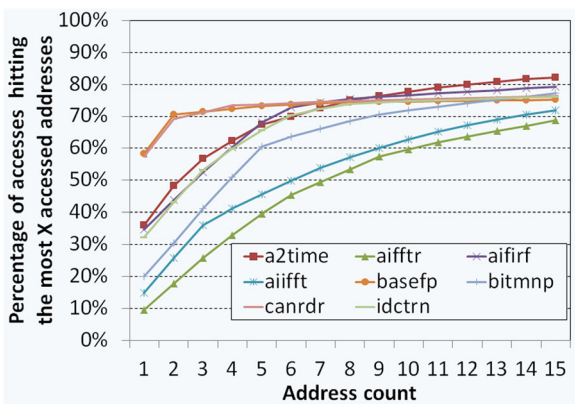

Fig. 6 Data accesses coverage

addresses are the ones with the highest potential to create miss count variations, thus affecting representativeness. This relates to instruction addresses accessed in loops and data accesses with reuse distances long enough not to be mapped into registers. In both cases, this leads to reuse distances often above W, so that some random placements may make each of those addresses cause much higher miss counts than usual. However, there are some known, as well as some potential, exceptions that we review next.

Exception 1. Due to the particular access patterns of the program, the most accessed addresses might have very high hit rates even if placed in the same set as other addresses simply because their reuse distance is pretty short. For instance, in a cache with $W = 4$, a program whose access pattern is $\{ABACADAEAFAGABACADAEAFAG\}$ would lead to very high hit probabilities for A despite the particular cache placement. However, addresses B, C, D, E, F and G are much more sensitive to the particular cache placement since they may lead to high miss rates if five or more addresses are placed in the same set. Hence, although in general higher access counts relate to higher miss counts and so higher sensitivity to the particular cache placement, this cannot be proven true in all cases.

Exception 2. In some cases, the addresses considered may be as relevant as some of the addresses dismissed. This is, for instance, the case of instruction addresses in a loop. Let us assume a loop whose code spans to 20 cache lines. By using $U = 15$, 5 of those 20 addresses will be ignored. In this case, all address combinations with a given address count (e.g. five addresses) have the same impact. However, the number of combinations produced with 15 addresses is lower than the one that would be obtained with 20 addresses. Hence, the corresponding pair $< \text{impact}, \text{prob} >$ with $U = 15$ will have a lower probability than the one obtained with $U = 20$. As a result, for a given R, the resulting pWCMC may be deemed as not reliable for $U = 20$ (thus requesting more runs), whereas it may be deemed as reliable with $U = 15$ (thus not requesting more runs).

5.2 Impact on R'

Using a limited number of unique addresses may affect the final number of runs R' requested to the user. If R' with $U = 15$, referred to as R'_{15}, is equal or higher than R'_{all}, where all stands for all addresses in the program, then our method may be requesting some extra runs above those strictly needed. This increases the burden on the user side but delivers confidence levels equal or higher than the desired ones.

Conversely, it can be the case that $R'_{15} < R'_{\text{all}}$. In this case, some risk exists that those R'_{15} runs do not capture all relevant cache placements with sufficient confidence. Still, even in that case, the pWCET is not necessarily optimistic. Other cache placements or other sources of execution time variability may make MBPTA produced a reliable pWCET curve even if the particular event in the example has not been observed in the measurements collected during the analysis phase, which could occur with a probability higher than required (e.g. 10^{-6} instead of 10^{-9}).

In order to understand the impact on R' of different values of U, in Section 6.4, we perform an analysis of ReVS comparing $U = 15$ and $U = 10$. This allows us to understand what we lose by discarding the 5 least accessed addresses out of the 15 most accessed ones.

6 Experimental results

We model an in-order processor with a memory hierarchy comprising 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 [9]. 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 Autobench 1.1 benchmark suites, representative of some safety-related real-time automotive applications [15]. The average number of lines of code (LoC) for these benchmarks is around 6500. A popular benchmark suite used in academia in this domain, Mälardalen benchmarks [18], has instead only 350 LoC per benchmark. Therefore, due to the higher complexity of EEMBC and their industrial nature, we have used them for the evaluation of this work. We consider the $U = 15$ most accessed addresses for instructions and data for each benchmark that covers on average 67% of the accesses across all benchmarks. Average reuse distances and their standard deviation are shown in Table 2 for the full traces ($U = \infty$) and those with $U = 15$. As shown, there is a wide variety of different behaviour across benchmarks, especially for the DL1, thus stressing the ability of ReVS to determine the number of runs R' needed. In order to analyse the impact of dismissing some addresses, we also consider $U = 10$ and compare it against $U = 15$. In all cases, we start by applying MBPTA with the number of runs R regarded as sufficient by the MBPTA technique for each program [5]. Then, we apply our approach, ReVS, for the instruction and data caches and obtain the number of runs required to pass the validation step R'.

Table 2 Average and standard deviation for the reuse distances in EEMBC benchmarks

	Reuse distance							
	$U = \infty$				$U = 15$			
	IL1		DL1		IL1		DL1	
	μ	σ	μ	σ	μ	σ	μ	σ
`a2time`	7.97	28.94	2.53	8.20	0.57	1.75	1.26	2.45
`aifftr`	3.16	9.05	7.92	71.45	0.34	0.99	0.52	0.77
`aifirf`	0.81	3.52	2.02	9.24	0.55	1.16	0.74	1.22
`aiifft`	3.27	9.29	7.93	73.55	0.34	0.93	0.51	0.75
`basefp`	0.37	2.21	2.81	38.22	0.30	0.71	0.35	0.61
`bitmnp`	9.55	27.49	1.92	4.51	0.56	1.52	1.24	0.85
`canrdr`	0.62	1.77	1.79	11.73	0.58	1.18	0.41	0.72
`idctrn`	0.96	3.78	2.07	14.38	0.29	0.77	0.46	0.72

6.1 Correlating execution time and miss counts

ReVS is required when miss counts impact execution time. 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. 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 [17]. Executions on this processor take much longer than the ones on our simple simulator, whose accuracy has been assessed against the FPGA implementation. However, as shown next, these results prove that modelling execution time mostly with cache behaviour is an extremely accurate proxy. Both, the cache simulator and the FPGA, implement write-back write-allocate policies in DL1. However, the FPGA includes a write buffer for dirty lines evicted, whereas the simulator does not.

Qualitative assessment. First, we perform $R = 1000$ 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 normalise 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 normalised to the differential between the minimum and maximum values observed. Formally, normalised 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:

$$\text{NormMiss}_i = \frac{\text{Miss}_i - \left(\text{MIN}_{j=0}^{R}\text{Miss}_j\right)}{\left(\text{MAX}_{j=0}^{R}\text{Miss}_j\right) - \left(\text{MIN}_{j=0}^{R}\text{Miss}_j\right)} \tag{3}$$

Likewise, we compute NormET_i:

$$\text{NormET}_i = \frac{\text{ET}_i - \left(\text{MIN}_{j=0}^{R}\text{ET}_j\right)}{\left(\text{MAX}_{j=0}^{R}\text{ET}_j\right) - \left(\text{MIN}_{j=0}^{R}\text{ET}_j\right)} \tag{4}$$

where ET_i is the execution time measured in execution i.

NormMiss and NormET for `a2time` and `bitmnp` benchmarks are shown in Fig. 7. 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 to obtain correlation coefficients [19]: the Pearson product-moment correlation coefficient and Spearman's rank correlation coefficient. The Pearson product-moment correlation coefficient measures the linear dependence between two variables. 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. Both

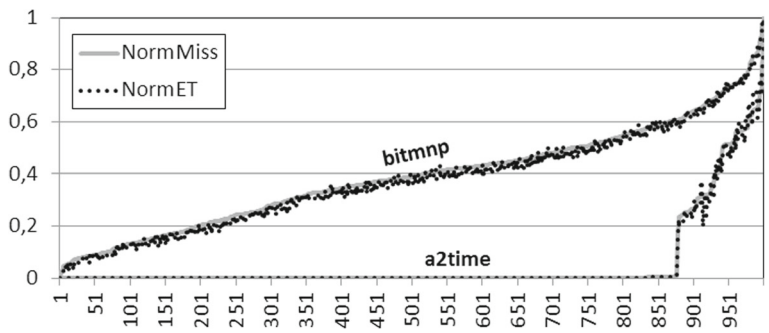

Fig. 7 NormMiss and NormET for `a2time` and `bitmnp` sorted by NormMiss

methods deliver 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. For both methods, we use a 5% significance level (a typical value for this type of tests [20]).

As shown in Table 3, 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). We have further analysed benchmarks with the lowest values and have realised that they experience very low execution time and miss count variations. Thus, other sources of jitter, like those introduced by the store buffer, have a relatively higher impact than for other benchmarks.

6.2 ReVS results: illustrative examples

To illustrate how ReVS 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). A larger number of runs was not collected due to the cost to run those many simulations. Note that performing that number of runs is not required for ReVS application; we just perform them for illustrative purposes in this section.

ReVS passed. Figure 8a shows the result of applying ReVS for the instruction accesses of `bitmnp`. The curves on the left show the $<$ impact, prob $>$ pairs derived with ReVS 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 Fig. 8b, where the ECCDF is reliably upperbounded by the pWCET estimate derived with MBPTA with R runs.

Table 3 The 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

ReVS failed. In the case of `aifirf`, our method detects that the number of runs obtained with MBPTA $R = 4400$ is not enough to provide a reliable pWCET estimate. In Fig. 8c, we observe that the pWCMC curve does not upperbound the $<$ impact, prob $>$ pairs generated by ReVS. As a result in the timing domain, the pWCET estimate derived with R runs does not upperbound the execution time of the program. ReVS requires the number of runs to be increased to $R' = 21,390$. If MBPTA is applied in the timing domain with R' runs, the resulting pWCET estimate is reliable as we can observe in Fig. 8d.

In general, applying MBPTA with R runs instead of R' delivers reliable pWCET estimates. We have corroborated this fact empirically with all the benchmarks and through a number of experiments with different configurations in other works. However, if ReVS is not used, there may be some probability of obtaining an unreliable pWCET upperbound. Still, whether this occurs or not relates to the confidence level obtained with R runs instead of R' as explained later.

6.3 ReVS results: EEMBC automotive

Table 4 summarises the number of runs required by MBPTA (R) and ReVS in the miss domain for both DL1 and IL1 for all benchmarks. For the sake of completeness, we also compare ReVS against different flavours of HoG: its original [10] and improved versions [13], considering the full address trace or only $U = 15$ cache line addresses. The number of runs required by ReVS is the maximum across DL1 and IL1. That is, $R' = \max(R'_{\text{IL1}}, R'_{\text{DL1}})$. By comparing R' and R, we can assess whether the number of runs required by MBPTA in the execution time domain could lead to lower confidence levels than desired, which occurs when $R < R'$ for MBPTA.

We observe that this is the case for all the benchmarks in Table 4 for MBPTA. Regarding HoG, we realise that, for the full address trace, the number of cache line addresses for either data or code is no less than 35 across benchmarks. Regardless of whether we use the original or improved version of HoG, the number of runs required as determined by those methods is upperbounded by the number delivered by MBPTA. If we restrict the traces to $U = 15$, the same conclusion holds since, for instance, the improved version of HoG makes MBPTA start with 209 runs, which is always upperbounded by the minimum number of runs required by MBPTA (300).

Note that these results are not independent of the actual cache setup. For instance, if we used a 4-way 32-set cache instead of a 2-way 64-set cache, then HoG improved would request at least 10,955 runs for $U = 15$, which may not be upperbounded by the minimum number of runs required by MBPTA (300). We can conclude that the number of runs needed to achieve the level of confidence

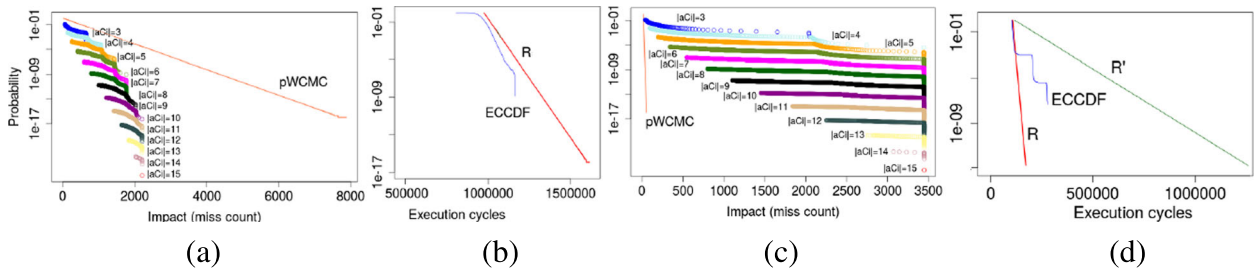

Fig. 8 ReVS applied to the instruction accesses of `bitmnp` and `aifirf`. The analysis is performed for combinations of addresses with increasing cardinality, $|aC_i| \in [W+1, U]$ and pWCET estimates obtained with R and R' runs. **a** pWCMC for `bitmnp`. **b** pWCET for `bitmnp`. **c** pWCMC for `aifirf`. **d** pWCET for `aifirf`

desired is highly sensitive to the actual access patterns. Therefore, ReVS is needed in the general case.

On the other hand, this does not mean that results obtained with less than R' runs are unreliable, but the confidence level had on them is lower than the target confidence level. ReVS keeps the likelihood of missing relevant cache placement scenarios of interest below 10^{-9}, as discussed in Section 2. Instead, if we only use the number of runs, R, determined by MBPTA (HoG), the likelihood of missing those scenarios becomes higher (see the corresponding *likelihood* columns). This decreases the confidence on the results below the levels defined in the corresponding safety standards. Still, it is often the case that relevant scenarios are 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 challenges the development of arguments for certification. Regarding execution time cost, HoG method executes in around 100 ms per cache and benchmark on average (in comparison to 1.5 h for ReVS method). This is expected since HoG neglects access patterns and can model the program as the number of unique addresses. Instead, ReVS is a pattern-aware method that tradeoffs computational cost for accuracy.

6.4 Assessing ReVS reliability

We assess the impact on reliability of analysing a limited number of addresses by comparing the results in terms of R' of applying ReVS considering $U = 10$ (R'_{10}) and $U = 15$ (R'_{15}). Results for R'_{15} are shown in Table 4, whereas results for R'_{10} are shown in Table 5.

The first observation is that either R'_{10} is higher than R'_{15} or, if lower, close to it. In particular, 5 out of 8 benchmarks meet the condition $R'_{10} \geq R'_{15}$. Therefore, the confidence level obtained is equal or higher than the desired one, but the number of runs requested to the user may increase. For instance, the most extreme cases are those of `aifftr` and `aiifft`, where the end user is requested to increase the number of runs by 3× and 5×, respectively, w.r.t. the case where $U = 15$, as well as the case of `canrdr`, where MBPTA required 10,000,000 runs[3]. In those cases where $R'_{10} < R'_{15}$, the difference is between 0.5 and 13.8%. This makes that the confidence level of the pWCET estimates obtained with R'_{10} is slightly lower than desired. In this case, the chance of missing relevant events grows to the range $[1.1 \times 10^{-9}, 1.7 \times 10^{-8}]$. While this is not desirable, still, the likelihood of these unwanted scenarios can be deemed as extremely low.

If we analyse the results in more detail, we realise that R'_{10} is higher or only slightly lower than R'_{15} for the IL1. This relates to the scenarios described in Section 5 for

Table 4 Results for all EEMBC benchmarks

	ReVS				MBPTA / HoG / HoG($U = 15$)	
	R'_{IL1}	R'_{DL1}	R'	likelihood (R')	R	likelihood (R)
`a2time`	58,360	540	58,360	10^{-9}	2650	0.390
`aifftr`	6840	5500	6840	10^{-9}	2200	0.001
`aifirf`	21,390	11,530	21,390	10^{-9}	4400	0.014
`aiifft`	8920	8770	8920	10^{-9}	1900	0.011
`basefp`	82,080	20,010	82,080	10^{-9}	300	0.927
`bitmnp`	4640	3510	4640	10^{-9}	850	0.007
`canrdr`	18,610	7950	18,610	10^{-9}	350	0.677
`idctrn`	65,770	47,700	65,770	10^{-9}	3650	0.317

Table 5 Results for all EEMBC benchmarks [$U = 10$]

	ReVS			
	R'_{IL1}	R'_{DL1}	R'	likelihood (R')
a2time	78,930	1520	78,930	$< 10^{-9}$
aifftr	14,230	19,600	19,600	$< 10^{-9}$
aifirf	25,890	5400	25,890	$< 10^{-9}$
aiifft	18,030	46,230	46,230	$< 10^{-9}$
basefp	72,900	1700	72,900	1.0×10^{-8}
bitmnp	3670	4000	4000	1.7×10^{-8}
canrdr	17,750	10,000,000	10,000,000	$< 10^{-9}$
idctrn	65,460	58,860	65,460	1.1×10^{-9}

the IL1: the number of addresses accessed in a round-robin manner inside the main loop may be higher than U. By using a low value for U, the probability of producing cache placements of interest is lower in the Monte-Carlo experiment and thus, more runs are needed to produce those placements so that the pWCMC curve upperbounds all $<$ impact, prob $>$ pairs. However, when increasing U, the true probability of the relevant placements is higher than assumed by ReVS with low U values. However, a larger U value also increases the chances of randomly placing enough conflictive addresses in the same set and thus triggering cache placements of interest, which leads to pWCMC curves upperbounding all $<$ impact, prob $>$ pairs. Hence, fewer runs are needed to guarantee that those placements are conveniently observed.

Runs needed for the DL1 grow in all cases but for two notable exceptions: `aifirf` and `basefp`. This decrease in R'_{DL1} with $U = 10$ w.r.t. $U = 15$ has no effect on the confidence level achieved since it is masked by the fact that R'_{IL1} is typically higher than R'_{DL1}. However, this is not necessarily always the case and thus, discarding some DL1 addresses might potentially affect the confidence level achieved for the pWCET estimates. For instance, if we compute the probability of missing relevant placements only with R'_{DL1}, then likelihood for R'_{10} would be 6.1×10^{-5} and 0.17 for `aifirf` and `basefp`, respectively.

In summary, using the most accessed addresses of program typically allows achieving the desired confidence level for the pWCET estimates. However, ReVS reliability might be affected in some specific scenarios due to the effects of those addresses dismissed due to the computational cost of the method. Hence, part of our future work consists of extending ReVS to be able to analyse all programs addresses within acceptable computation time bounds.

7 Related work

Literature on WCET estimation is abundant [3]. 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 [5, 16, 21–23]. However, MBPTA may lead to pWCET estimates with lower confidence than desired on top of caches implementing random placement in some particular scenarios [10–12]. Some solutions exist for scenarios where all accessed addresses have the same impact in terms of execution time [10]. However, access patterns of programs may be arbitrary, since addresses are accessed with different frequencies and with arbitrary interleaving. In this paper, we tackle this issue by proposing—as an extension of the conference paper in [24]—a validation step, ReVS, able to test whether the confidence had on the WCET estimates obtained with MBPTA is sufficiently high. If it is not, ReVS increases the number of runs needed until the validation step is passed.

So far, in the real-time domain, EVT has been applied only to execution times [5, 23], whereas 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 [25]. Results show that MBPTA provides competitive results with respect to those provided by static timing analysis techniques.

8 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 some risk of not observing all relevant events affecting execution time during analysis runs. Therefore, the confidence had in the pWCET estimates obtained is lower than desired. While this challenge has already been addressed 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 attain the desired confidence in pWCET estimates for arbitrary memory access patterns. Our method, ReVS, identifies the worst miss counts and their probabilities of occurrence and, by means of controlled cache simulations, tests whether the number of measurement runs used for pWCET estimation is high enough to capture all cache placements of interest. Our results illustrate the effectiveness of our method to attain the desired confidence level in the pWCET estimates obtained.

Our future work will focus on reducing the computational cost of ReVS, generalising it towards more complex architectures such as multi-level cache hierarchies, considering other random placement policies such as random modulo or software randomisation, and dealing

with the considerations related to multi-path programs. We foresee reducing the computational cost by using analytical methods to dismiss combinations that cannot produce high miss count variations (e.g. addresses hardly interleave) and creating buckets of addresses whose behaviour is almost identical, so that combinations including one of them need to be explored once rather than exploring each one of them. Work on cache hierarchies, placement policies and multi-path programs will build on the probabilistic nature of cache placement to derive the number of runs R' needed to capture relevant events.

Endnotes

[1] We also provide evidence that execution time and miss counts strongly correlate on the commercial processor prototyped on FPGA used for evaluation purposes.

[2] The *expected value* of a random variable is the average value obtained after infinite repetitions of the experiment. In the case of a finite sample, the expected value is approximated with the average of the observed values.

[3] In fact, due to the difficulties to search for the precise value in our toolchain for big samples, we could only determine that the pWCMC was not upperbounded with 1,000,000 runs, but it was with 10,000,000 runs.

Abbreviations

MBPTA: Measurement-Based Probabilistic Timing Analysis; MBTA: Measurement-Based Timing Analysis; PTA: Probabilistic Timing Analysis; ReVS: Representativeness Validation by Simulation; TRC: Time-Randomised Cache; WCET: Worst-Case Execution Time; WCMC: Worst-Case Miss Count

Funding

This work has received funding from the European Community's FP7 program [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.

Authors' contributions

SM, JA and FJC jointly developed the theoretical and practical solutions described in this paper, analysed the results and evolved the techniques until they reached a sufficient degree of maturity. SM took care of all the implementations and collected all the data from the benchmarks. JA and FJC identified the most convenient correlation tests used in Section 5.1 and applied them on the data. Finally, the three authors contributed in similar amounts to the writing and reviewing of the manuscript, as well as to producing the figures and tables. All authors read and approved the final manuscript.

Competing interests

The authors declare that they have no competing interests.

Publisher's Note

Author details

[1]Barcelona Supercomputing Center (BSC), Barcelona, Spain. [2]Universitat Politècnica de Catalunya (UPC), Barcelona, Spain. [3]IIIA-CSIC, Barcelona, Spain.

Received: 15 October 2016 Accepted: 27 March 2017
Published online: 11 April 2017

References

1. SAE International; ARP4761: Guidelines and methods for conducting the safety assessment process on civil airborne systems and equipment. http://standards.sae.org/arp4761/. Accessed 4 Apr 2016
2. *ISO/DIS 26262. Road Vehicles – Functional Safety*, 1st edn. (International Organization for Standardization, Geneva, 2011)
3. 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, P Stenström, The worst-case execution-time problem—overview of methods and survey of tools. ACM Trans. Embed. Comput. Syst. **7**(3), 36–13653 (2008). doi:10.1145/1347375.1347389
4. S Law, I Bate, in *2016 28th Euromicro Conference on Real-Time Systems (ECRTS)*. Achieving appropriate test coverage for reliable measurement-based timing analysis, (2016), pp. 189–199. doi:10.1109/ECRTS.2016.21
5. L Cucu-Grosjean, L Santinelli, M Houston, C Lo, T Vardanega, L Kosmidis, J Abella, E Mezzetti, E Quiñones, FJ Cazorla, in *2012 24th Euromicro Conference on Real-Time Systems*. Measurement-based probabilistic timing analysis for multi-path programs, (2012), pp. 91–101. doi:10.1109/ECRTS.2012.31
6. W Feller, *An introduction to probability theory and its applications*. (John Wiley and sons, 1967). https://books.google.es/books?id=nLgdAQAAMAAJ
7. S Kotz, S Nadarajah, *Extreme value distributions: theory and applications*. (World Scientific Publications, Singapore, 2000), p. 185
8. L Kosmidis, E Quiñones, J Abella, T Vardanega, I Broster, FJ Cazorla, in *17th Euromicro Conference on Digital System Design (DSD)*. Measurement-based probabilistic timing analysis and its impact on processor architecture, (2014), pp. 401–410. doi:10.1109/DSD.2014.50
9. L Kosmidis, J Abella, E Quiñones, FJ Cazorla, in *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*. A cache design for probabilistically analysable real-time systems, (2013), pp. 513–518. doi:10.7873/DATE.2013.116
10. J Abella, E Quiñones, F Wartel, T Vardanega, FJ Cazorla, in *26th Euromicro Conference on Real-Time Systems*. Heart of gold: making the improbable happen to increase confidence in mbpta, (2014), pp. 255–265. doi:10.1109/ECRTS.2014.33
11. J Reineke, Randomized caches considered harmful in hard real-time systems. Leibniz Trans. Embed. Syst. **1**(1), 03–10313 (2014)
12. E Mezzetti, M Ziccardi, T Vardanega, J Abella, E Quiñones, F Cazorla, Randomized caches can be pretty useful to hard real-time systems. Leibniz Trans. Embed. Syst. **2**(1), 01–10110 (2015)
13. P Benedicte, L Kosmidis, E Quinones, J Abella, FJ Cazorla, in *11th IEEE Symposium on Industrial Embedded Systems (SIES)*. Modelling the confidence of timing analysis for time randomised caches, (2016), pp. 1–8. doi:10.1109/SIES.2016.7509421
14. P Benedicte, L Kosmidis, E Quinones, J Abella, FJ Cazorla, in *2016 IEEE 14th International Conference on Industrial Informatics (INDIN)*. A confidence assessment of WCET estimates for software time randomized caches, (2016), pp. 90–97. doi:10.1109/INDIN.2016.7819140
15. JA Poovey, TM Conte, M Levy, S Gal-On, A Benchmark Characterization of the EEMBC Benchmark Suite. IEEE Micro. **29**(5), 18–29 (2009). doi:10.1109/MM.2009.74
16. F Wartel, L Kosmidis, A Gogonel, A Baldovino, Z Stephenson, B Triquet, E Quiñones, C Lo, E Mezzetta, I Broster, J Abella, L Cucu-Grosjean, T Vardanega, FJ Cazorla, in *Design, Automation Test in Europe Conference Exhibition (DATE)*. Timing analysis of an avionics case study on complex hardware/software platforms, (2015), pp. 397–402. doi:10.7873/DATE.2015.0189
17. C Hernandez, J Abella, FJ Cazorla, J Andersson, A Gianarro, in *20th Data Systems In Aerospace Conference (DASIA)*. Towards making a LEON3 multicore compatible with probabilistic timing analysis (ASD-EUROSPACE, Brussels, 2015)
18. J Gustafsson, A Betts, A Ermedahl, B Lisper, in *10th International Workshop on Worst-Case Execution Time Analysis (WCET 2010)*, ed. by B Lisper. The Mälardalen WCET benchmarks: past, present and future (Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, 2010), pp. 136–146.

http://drops.dagstuhl.de/opus/volltexte/2010/2833. doi:10.4230/OASIcs.WCET.2010.136
19. M Hollander, DA Wolfe, *Nonparametric statistical methods*. (John Wiley & Sons, New York, 1973)
20. RA Fisher, in *Breakthroughs in Statistics: Methodology and Distribution*, ed. by S Kotz, NL Johnson. The arrangement of field experiments (Springer New York, New York, 1992), pp. 82–91. doi:10.1007/978-1-4612-4380-9_8
21. JL Diaz, DF Garcia, K Kim, C-G Lee, LL Bello, JM Lopez, SL Min, O Mirabella, in *23rd IEEE Real-Time Systems Symposium (RTSS)*. Stochastic analysis of periodic real-time systems, (2002), pp. 289–300. doi:10.1109/REAL.2002.1181583
22. G Bernat, A Burns, M Newby, Probabilistic timing analysis: an approach using copulas. J. Embedded Comput. **1**(2), 179–194 (2005)
23. JP Hansen, SA Hissam, GA Moreno, in *9th Intl. Workshop on Worst-Case Execution Time Analysis, WCET 2009, Dublin, Ireland, July 1–3, 2009*. Statistical-based WCET estimation and validation, (2009). http://drops.dagstuhl.de/opus/volltexte/2009/2291
24. S Milutinovic, J Abella, FJ Cazorla, in *2016 IEEE 19th International Symposium on Real-Time Distributed Computing (ISORC)*. Modelling probabilistic cache representativeness in the presence of arbitrary access patterns, (2016), pp. 142-149. doi:10.1109/ISORC.2016.28
25. J Abella, D Hardy, I Puaut, E Quiñones, FJ Cazorla, in *26th Euromicro Conference on Real-Time Systems*. On the comparison of deterministic and probabilistic wcet estimation techniques, (2014), pp. 266–275. doi:10.1109/ECRTS.2014.16