Heart of Gold[1]: Making the Improbable Happen to Increase Confidence in MBPTA

Jaume Abella[†], Eduardo Quiñones[†], Franck Wartel[*], Tullio Vardanega[Ψ], Francisco J. Cazorla[†,‡]

[†]Barcelona Supercomputing Center (BSC) [‡]Spanish National Research Council (IIIA-CSIC)
[Ψ]University of Padua [*]Airbus SAS France

***Abstract*—Measurement-Based Probabilistic Timing Analysis (MBPTA) has been recently proposed as a viable method to compute probabilistic worst-case execution time (pWCET) bounds for programs with hard real-time constraints.**

As a key trait, MBPTA needs a comparatively small number of observation runs, made on execution platforms to which MBPTA can be applied, to project the tail of the probability of occurrence of worst-case execution time durations of individual programs. In order for the use of MBPTA to fit the bill of industrial-quality development, it is imperative to understand what factors might threaten the trustworthiness of the pWCET computation.

This paper addresses that important question by: (i) identifying the combined characteristics of applications and hardware resources that might lead to optimistic pWCET bounds; (ii) describing why this may occur; and (iii) providing the user with means to detect those cases so that trustworthiness is restored. In particular, we present a method for detecting risk scenarios for time-randomised caches, based on principles that apply to any other time-randomised resource which may challenge the application of MBPTA.

I. Introduction

Probabilistic Timing Analysis (PTA) techniques, and in particular its measurement-based incarnation (MBPTA) [8], [28] have been shown to successfully reduce the information required from the target hardware and software to derive tight probabilistic WCET (pWCET) estimates. MBPTA provides a probability distribution function that upper-bounds the execution time of the program under analysis. MBPTA guarantees that the execution time of a program only exceeds a given threshold with a probability lower than a given target probability (e.g., 10^{-15} per run). The pWCET for a program is defined as the program's execution time bound with its corresponding exceedance probability.

A trustworthy application of MBPTA requires a MBPTA-compliant execution platform [2], which usually comprises a number of hardware or software time-randomised resources. Time randomised resources, which respond with random execution time durations to software use (each accesss to the resource being a random event), cause probabilistic variability in the software timing behaviour, leaving functional behaviour unchanged. One key trait of MBPTA in contrast with other measurement-based timing analysis techniques is that MBPTA does not require observing a high execution time to predict that it may occur. This is so because MBPTA uses Extreme Value Theory (EVT) [21] to predict the effect on timing of the combined occurrence of random events.

In this paper we consider two sources of execution time variation in a MBPTA-compliant platform, namely: *hardware-only induced* random events whose timing behaviour is fully described by the characteristics of the hardware; and *hardware-software induced* random events, whose timing behaviour is determined by the execution of a particular sequence of program instructions that use time-randomised resources.

We show that pWCET estimates obtained with MBPTA are trustworthy so long as sufficient *coverage of the random events* is achieved. Sufficient coverage is obtained when any of the following conditions occurs:

- The input vectors used for the program trigger the longest latency of all events in all time-randomised resources in some runs (e.g., incurring a miss for a cache access with non-zero miss probability). It is not necessary that the worst outcomes of all random events occur in one run. It suffices instead that each such outcome occurs in at least one run, regardless of whether they all occur in the same run or not. The combined effect of the outcomes of all the random events produced by the platform resources is captured by EVT.
- For each resource, only a subset of the random events that they produce yield their longest latency in all runs carried out. However, the longest latency of the remaining events is not higher than that of the observed ones. For instance, in a program execution containing K accesses to a time randomised cache, each being a random event, only L, with $L < K$, cache misses are observed, but the individual latency of the $K - L$ not-observed accesses is not worse than the L that occurred.
- In all program runs, the latencies observed for a given event exhibit similar magnitude, although the worst may not be observed. For instance, 2-, 4- and 6-cycle latencies of a random event are observed but not its worst latency of 8 cycles.

The goal of EVT is to predict the extreme combined effect of the outcomes of random events occurring in a system. EVT however cannot capture the combined outcome of random events that are not observed: the effect that those outcomes can have on the extreme behaviour of the system is missed. MBPTA therefore needs sufficient coverage to be achieved of the outcome of the random events of interest. To this end, we describe the timing characteristics that time-randomised resources should have, and how test software should exercise them so that sufficient coverage is assuredly achieved.

In particular, we show how certain combinations of time-randomised hardware designs and insufficiently informed use of them in software can lead to pathological timing behaviours, to capture which more observation runs than normally required may be needed to trustworthily apply MBPTA.

We first illustrate this phenomenon by building a synthetic program that triggers such behaviour for a particular instance of a time-randomised resource: a cache memory implementing random placement and replacement policies [17]. We then propose *Heart-of-Gold* (HoG for short), a method that allows

[1]The Heart of Gold was the prototype ship for infinitely improbable travel in "The Hitchhiker's Guide to the Galaxy" by Douglas Adams. The ship relied on extremely improbable events to happen to travel faster than light.

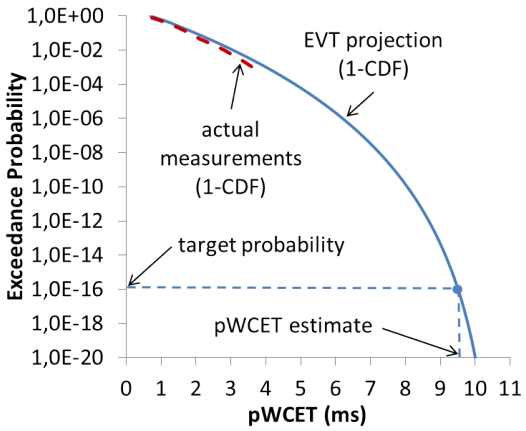

Fig. 1. Example of the 1-CDF and tail projection.

detecting those pathological scenarios by increasing the likelihood of occurrence of low-probability high latencies (should they exist) so that they can be observed and trustworthiness of pWCET prediction can be obtained. While the HoG technique is shown to work for time-randomised caches, the rationale behind it can be applied to other time-randomised resources. However, it is noted that our analysis in Section IV-B shows that other existing time-randomised hardware resources do not produce pathological timing behaviour.

The rest of the paper is organised as follows. Section II reviews MBPTA internals. Section III describes how applications can trigger random events on time-randomised hardware resources. Section IV deepens into the relation between random events and EVT. Sections V reviews some pathological timing behaviours when using random replacement and random placement cache policies. Section VI introduces HoG, our approach to detect pathological timing behaviour. Results are provided in Section VII. Related work is presented in Section VIII. Section IX concludes the paper.

II. MBPTA: Requirements and Application

MBPTA [8], [28] estimates the pWCET for a program from a collection of observed execution times of that program on an MBPTA-compliant platform. MBPTA employs *Extreme Value Theory* (EVT) [21], which computes a trustworthy and tight upper-bound of the tail of the observed execution time distribution, hence providing *pWCET* estimates for arbitrarily low *target probabilities*. Figure 1 shows a hypothetical result of applying EVT to a collection of 1,000 observed execution times. The dashed line represents the complementary cumulative distribution function (1-CDF) derived from the observed execution times. The continuous line represents the projection obtained with EVT.

A. Method Requirements

MBPTA requires certain properties to hold for the collected set of observations:

Independence. The execution time observations passed to MBPTA must be proven independent for EVT to apply. Two random variables are said to be independent if they describe two events such that the occurrence of one event does not have any impact on the occurrence of the other event. Statistical hypothesis testing is used to determine whether the outcomes of a set of executions (a sample of data) lead to a rejection of the null hypothesis, which in this case is that the data are independent, for a pre-specified level of significance. To this end we use the Wald-Wolfowitz test [5], which we describe in Appendix I.

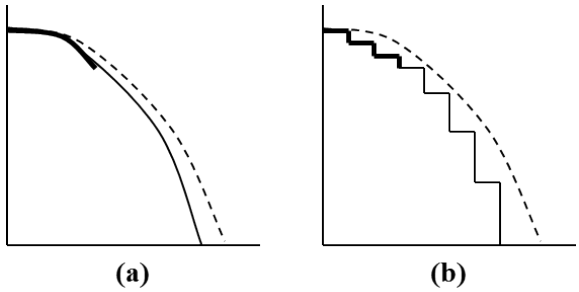

Fig. 2. Examples of EVT projections for different execution time distributions.

Identical distribution. The data input to MBPTA must also be proven identically distributed as EVT requires data to be from the same arbitrary distribution. A set of random variables is said to be identically distributed if all them have the same probability distribution. In the case of MBPTA, execution times must be proven to have identical distribution along time. Analogously to the case of independence, we use statistical testing to determine whether a set of data cannot be consider identically distrusted. To this end we use the Kolmogorov-Smirnov two-sample identical distribution test [11], which we describe in Appendix I.

Curve fitting. The collection of execution times provided as input to EVT must fit a Gumbel distribution [14], [21] to enable the use of MBPTA [8]. This is tested by means of the Exponential Tail (ET) test [13]. Given a set of observations proven i.i.d., the ET test computes an estimator and the bounds of an interval where the estimator must be included. If this is the case, then the Gumbel distribution hypothesis cannot be rejected. Conversely, if the estimator falls out of the interval, the hypothesis is rejected statistically.

Minimum number of runs. Based on the input data, MBPTA provides a Gumbel distribution curve that upper bounds the pWCET of the program under analysis. As such distribution may not be tight, MBPTA uses an iterative process to increase the size of the sample (a.k.a. the number of collected execution times), comparing the Gumbel distribution obtained in each step. When the difference is below a given threshold and this fact occurs consecutively for a given number of steps, it is concluded that including new execution times does not provide further insights, and so, the current Gumbel distribution tightly upper-bounds the pWCET of the program. This iterative process is known to converge since the ET test already proved that the input data fits a Gumbel distribution and so the input data used in each step will eventually describe Gumbel distributions close enough to the actual one.

B. EVT Projection

EVT projection allows bounding from above the tail of the execution time distribution inferred from the sampled execution times. Real probabilistic execution time distributions in the form of 1-CDF are step functions since execution times are discrete in nature and some execution times cannot occur. There is a step from value i to value $i + j$ if execution time durations in the range $(i, i + j)$ cannot occur.

Figure 2 illustrates several 1-CDF examples. Straight thin lines correspond to the real distribution of the execution times, which we would obtain after running an infinite number of times the program of interest. Straight thick lines correspond

to the region of the distribution observed typically with the limited number of runs used by MBPTA. Dashed lines correspond to the EVT projections generated with MBPTA.

Figure 2(a) shows the typical case where steps occur at fine granularity because execution time depends on a large number of random events that cause many different execution times to occur. Measurements capture the shape of the distribution and, thus, MBPTA tightly upper-bounds the tail of the distribution.

In Figure 2(b) larger plateaus occur. However, the random events captured in the observations still have sufficiently high probabilities for high latencies so that they are observed in the few runs required by the "pre-HoG" application of MBPTA. MBPTA is therefore able to produce pWCET estimates that upper-bound the execution time distribution.

Sections III and IV describe: (1) the nature of random events in MBPTA-compliant platforms, identifying the extent to which the software triggering them influences their impact on timing (which determines the size of the steps in the latency and probability distribution shown in Figure 2); and (2) the impact of those random events on the sane application of EVT as part of MBPTA, and how it could be so big to lead to optimistic pWCET estimates.

III. Random Events in a MBPTA-compliant architecture

A trustworthy application of MBPTA requires understanding the system factors that affect the execution time of the program under study: we call them *sources of execution time variability*. MBPTA classifies hardware resources into two groups: jitter-less and jittery [2]. Jitter-less resources are those whose response time is always the same such as in, e.g., most arithmetic logic units. Jittery resources are those whose response time varies across requests. This happens with stateful resources like caches, whose response time for an access depends on the particular access and the state of the cache itself, as well as with stateless resources like a randomly-arbitrated shared bus [16], whose response time depends solely on the random arbitration algorithm.

In MBPTA-compliant hardware, the timing behaviour of jittery resources has to be either upper-bounded or randomised [2] such that the latencies observed for those resources during the analysis phase upper-bound or follow the same distribution as experienced during operation. Whether the timing of a particular resource must be upper-bounded or randomised depends on the trade-off between its hardware design complexity and its potential impact on pWCET.

Next we describe the rationale behind the use of time-randomised resources and how they relate to random events.

A. Time-Randomised Resources for MBPTA

MBPTA uses the execution time measurements taken from end-to-end i.i.d. runs of the program under analysis. It has been shown in [7] that upper-bounding or randomising the timing behaviour of some hardware resources allows removing the dependence on some inputs, which lessens the burden on the user in designing the test runs. On the one hand, for some resources such as caches, upper-bounding is not an option since assuming all accesses to miss would cause exceedingly pessimistic pWCET estimates. On the other hand, measurements are obtained using particular memory placements for objects such as libraries, RTOS code, stack frames, etc., which may change between analysis and operation, and thus invalidate pWCET estimates. Ensuring that the memory object placement during testing represents the one occurring during operation is hard to attain, especially if one must consider all combinations of all potential placements with all other factors [7]. Time-randomised resources greatly help in this case as they allow breaking the dependence between initial conditions and pWCET estimates without incurring excessive pessimism. This benefit has been already shown for placement and replacement policies in cache memories [17], [18] as well as for shared buses [16].

B. From Time-Randomised Resources to Random Events

A *random event* in a MBPTA-compliant architecture corresponds to the access to any hardware/software resource with randomised timing behaviour when used by a particular dynamic instruction[2]. For instance, a random event occurs whenever a particular instruction is fetched by accessing a random-placement random-replacement cache. Random events can be described with a probabilistic distribution function or Execution Time Profile (ETP), which defines the discrete probability distribution function of execution times of an execution component (e.g. an instruction). Thus, for the different execution times of an execution component, an ETP defines their corresponding probabilities. Hence, the timing behaviour of a program/instruction is described by the probability mass function: $(\overrightarrow{l}, \overrightarrow{p}) = (\{l_1, l_2, ..., l_k\}, \{p_1, p_2, ..., p_k\})$, where p_i is the probability of that execution component taking latency l_i, with $\sum_{i=1}^{k} p_i = 1$.

We differentiate two types of random events, namely, hardware-only induced random events and hardware/software induced random events.

HW-only induced random events. In this case, only the hardware determines the ETP of the random event. For instance, in a multicore processor, this is the case of a shared bus to memory if the arbitration policy is random [16], whereby in each arbitration a random core gets access. If every round of arbitration takes L cycles, the ETP of the randomised bus can be described as: $ETP_{rbus} = \{\{0, L, 2L, ...\}, \{p^0_{arb}, p^1_{arb}, p^2_{arb}, ...\}$, where $p^k_{arb} = \left(1 - \frac{1}{N}\right)^k \times \frac{1}{N}$. We observe that the ETP is determined by the maximum number of bus contenders, which solely depends on the hardware configuration. The software running on top of it only determines how many times such a hardware resource is used, but has no effect on its ETP.

HW/SW induced random events. In this case, the ETP is determined by the hardware design together with the use that the software makes of it. This is the case of randomised caches [17]. For instance, assuming a fully-associative cache with random replacement, a cache line is randomly selected for eviction on a miss. Hence the probability of any element in cache to be evicted is $\frac{1}{W}$ on every eviction (a purely hardware dependent factor), with W the number of ways in the cache. However, the number of evictions occurring between two consecutive accesses to the same cache line address depends on which memory addresses the software accesses.

The execution of each access to the random cache is a random event as true probabilities are associated to each

[2]With dynamic instruction we designate an individual occurrence of a given program instruction in a given execution path. With static instruction we designate one of the processor instructions used in the program. Accordingly, any program instruction executed by the processor is a dynamic instruction, regardless of whether the instruction was fetched and executed before. For instance, if the program code has a static instruction i in a loop with M iterations, there will be M different dynamic instructions for i, say i_1, i_2,...,i_M. In this paper we use indistinctly instruction and dynamic instruction.

potential latency [17]. Those probabilities are determined by the behaviour of the hardware and how the software uses it.

While HW-only induced random events are, by construction, independent of each other, this may not be the case for HW/SW induced random events where the outcome of one event may affect the ETP of another event [2]. For instance, in the case of the random-replacement cache described above, the fact that a given access hits or misses will affect the number of evictions, and so the hit/miss probabilities of subsequent accesses.

Next we deepen the relation between the characteristics of random events – their ETP – and EVT.

IV. Random Events and EVT

One of the key differences between MBPTA and other measurement-based timing analysis techniques is that MBPTA does not require observing a high execution time to predict it. That is, EVT gives the user a methodology for predicting the occurrence of non-observed rare events. In this sense, EVT, as applied by MBPTA, focuses on the prediction of the combinations of the highest latencies for a set of random events. However, such prediction will be valid if input data (i.e. the collection of execution times) provides enough *coverage* of the high latencies that may occur. In this section we discuss what "enough coverage" means and why such coverage might not be reached in some particular cases.

A. Coverage of Random Events in MBPTA

As explained in section II-A, the ET test guarantees that MBPTA will converge into a suitable Gumbel distribution upper-bounding the execution time distribution of the program under analysis. The method used to determine the minimum number of runs to feed EVT is iterative and stops collecting further data when convergence is reached. This should happen when the input data provides enough information about the different random events triggered by the program. Enough information (or enough coverage) is attained when any of the following scenarios occurs:

a) The highest latency of all random events is triggered: The highest latency for each random event is observed in any of the execution time observations. Typically only some of those highest latencies will occur in a particular run. For instance, all cache accesses whose miss probability is non-zero effectively miss in at least one run. If this is the case, their impact in execution time as well as their dependences (e.g., a given access A_i misses with higher probability when a previous access B_j misses) will reflect in the execution time observations used by EVT, thus showing the shape of the execution time distribution. The combined effect of those high latencies and their probabilities are then captured by EVT.

For a given random event, whether it can be guaranteed that the probability of its highest latency being observed will be sufficiently high, jointly depends on its occurrence probability and the number of runs. Assume for instance that (1) we want to obtain a pWCET estimate with exceedance probability below 10^{-15} per run (a.k.a. cut-off probability), (2) we collect 500 runs and (3) the highest latency occurs with a probability of 0.1. In this scenario, we will not observe the highest latency with a probability around $10^{-23} \approx ((1-0.1)^{500})$. If there were $1,000$ such events in the program (i.e. per run), the probability of not observing the highest latency for at least one of them in one of the 500 runs would be $10^{-20} \approx (1-(1-10^{-23})^{1000})$. This probability value is negligible, as it falls largely below our target exceedance probability. In such a case we can conclude that the number of runs is enough to provide coverage for this set of random events.

b) The unobserved highest latencies do not exceed the observed ones: It may be the case that the highest latency of some particular random events[3] is not observed. For instance, consider the previous example, assuming that the probability for the highest latency is 0.01. After 500 runs it will not be observed with a probability around 0.0066. If $1,000$ such events occur, the highest latency will not be observed at least for one of them with a probability around 0.999.

However, if the highest latency for all random events is the same (e.g., cache accesses), then it does not matter which particular events exhibit their highest latency but how many of them do so. In particular, we would not observe any such high latency in all runs with a probability around 3.5×10^{-2181}. We would only observe execution times with up to one high latency (so 0 or 1 high latencies per run) with a probability around 5.3×10^{-2176}, and with up to 10 high latencies per run with a probability around 5.5×10^{-2136}. Thus, it can be seen that if the number of such events is large enough (or the probability of the high latencies are high), information about those latencies will be present in the data used by EVT.

c) Observed latencies provide information about unobserved ones: Finally, it can be the case that the highest latencies of all random events are never observed, but observed latencies already provide information about their probabilities. This is the typical case of a shared bus with random arbitration as the one proposed in [16]. Let's assume that there are N contenders to the bus. On every arbitration round the bus randomly picks a contender and grants access to it. The probability that a given contender is not picked after k arbitration rounds is $\left(1-\frac{1}{N}\right)^k$. We observe that this probability decreases but never reaches zero. As a result, the ETP for random events using such a resource is indeed infinite since a core might never be chosen in any arbitration round. However, observed latencies provide information about unobserved ones because latency increases constantly, by as many cycles as the bus latency, as probability decreases by a constant factor. For instance, given a bus shared across 4 cores, the probability of each number of arbitration rounds is as follows: 1 round $\left(\frac{4-1}{4}\right)^0 \times \frac{1}{4} = 0.25$, 2 rounds $\left(\frac{4-1}{4}\right)^1 \times \frac{1}{4} = 0.1875$, 3 rounds $\left(\frac{4-1}{4}\right)^2 \times \frac{1}{4} = 0.1406$, 4 rounds $\left(\frac{4-1}{4}\right)^3 \times \frac{1}{4} = 0.1055$, and so forth. Thus, even if a single random event occurs in each run, few runs will capture the latencies with the highest probabilities, which already show accurately the shape of the whole execution time distribution, so that EVT has enough information.

B. Lack of Coverage

Ultimately, in some pathological scenarios, the achieved coverage may be insufficient. This occurs when none of the previous conditions hold, that is, when: (i) there are just one or few random events; (ii) the probability of their highest latency is extremely low; and (iii) the observed events (latencies) do not provide information about unobserved ones.

Consider for instance a program that triggers 11 random events, 10 of which described by $ETP_1 = \{\{1,2\},\{0.5,0.5\}\}$ and 1 by $ETP_2 = \{\{1,1000\},\{0.9999,0.0001\}\}$. Owing to the random event with ETP_2, any execution time observation

[3]Events whose latency is fixed (e.g., a cache access that always misses) must be disregarded in this analysis as they are not random events but deterministic (constant) ones.

over 500 runs exceeds $1,000$ cycles with probability 0.0001 and ranges between 11 and 21 cycles with probability 0.9999.

In this scenario, the collected observations pass the MBPTA tests and MBPTA converges. Yet, with only 500 runs, it is very unlikely ($1 - 0.9999^{500} = 0.049$) that the highest latency of $1,000$ cycles of the event described by ETP_2 can be observed.

Interestingly, increasing the number of observations increases the likelihood of observing low-probability latencies, but whether such probability is high enough to be confident that high latencies will be observed also depends on how low the probability is for the high-latency event. In the example above, if the number of execution times collected was in the order of $500,000$ instead of 500, the probability of not observing those high execution times would be in the order of 10^{-22}. But even in this case, if the probability of taking $1,000$ cycles was 0.0000001 instead of 0.0001, we would have the same problem even with $500,000$ observations.

EVT projection (1) in Figure 3 describes what would happen in practice in those scenarios where observations do not provide enough coverage. This figure is discussed in Section V. The EVT projection leads to an optimistic pWCET estimate as MBPTA converges before the set of observations provides enough information about the actual shape of the distribution.

After a thorough analysis of the hardware resources for which authors in the literature have proposed time randomisation, we reached the following conclusions:

- The resources that participate in HW-only induced random events can be designed to avoid coverage problems by construction, because they cause no dependence on how the software uses them.
- Only very specific code structures cause coverage problems when used on top of some specific designs of random replacement and/or random placement caches [17], which participate in HW/SW induced random events. In general, if the timing behaviour of the hardware resource depends on how software uses it, it may be the case that particular programs trigger hazardous timing behaviour.

Our analysis of existing times-randomised hardware resources showed that only a few time-randomised cache designs can cause pathological timing behaviour for particular programs. In the remainder of this paper we therefore focus on this hardware resource. Yet, the rationale behind our analysis and the approach we propose to circumvent this problem are general enough to extend to other resources that might trigger similar behaviour.

We present concrete examples that illustrate cases of lack of coverage for random replacement and random placement. We then propose a method to identify those cases, so that optimistic pWCET estimates can be detected and discarded.

V. Pathological Cases in Time-Randomised Caches

Random replacement and random placement policies in caches might lead to pathological timing behaviour in combination with some particular software, causing pWCET estimates to be optimistic. We provide examples of those scenarios and show how they can trigger pathological timing behaviour.

A. Random Replacement Caches

Let us consider a processor design implementing a fully-associative data cache with random replacement [17] and a randomly-arbitrated bus to memory [16], whereas the rest of resources have constant latency. Cache latency is 1 cycle for hits and $1,000$ for misses (plus the bus arbitration in case of a miss). Bus arbitration latency is 1 cycle; the bus is shared across 2 contenders (e.g., 2 cores) and implements random permutations arbitration. With random permutations [16] one round is randomly assigned to each of the N contenders in each window[4]. Each round has a duration of $L = 1$ cycles, the maximum bus cycles that any request may take. In this approach, at every window boundary the arbiter generates a random permutation for the N contenders (cores). This sequence determines the order in which contenders can use the bus. Assuming that requests become ready at random points in time, the probability of a request to wait k rounds to get access to the bus, with $0 \leq k \leq 2N - 2$ is as follows [16]:

$$p^k_{perarb} = \frac{max(N-k,0)}{N^2} + \sum_{i=max(1,N-k)}^{min(N-1,2N-k-1)} \frac{i}{N^3} \qquad (1)$$

In our example, the bus latency is always between 1 and 3 cycles. With this the ETP for a memory operation is as follows $ETP_{memop} = \{1, 1001, 1002, 1003\}\{P_{hit}, P_{miss} \times P^1_{perarb}, P_{miss} \times P^2_{perarb}, P_{miss} \times P^3_{perarb}\}$. The ETP of a core operation is $ETP_{coreop} = \{1\}\{1.0\}$

Listing 1 $add2vectors\ (int\ size,\ int\ *src,\ int\ *dst)$

```
1: int i;
2: for (i = 0; i < size; i++) do
3:     dst[i] = src[i] + dst[i];
4: end for
```

Let us assume the program shown in Listing 1 that represents a function that adds two vectors. Reading $src[i]$ loads one new cache line, and then $dst[i]$ also loads another cache line, that is written back with the result of the addition. Given a N-entry ($N = 256$) fully-associative random-replacement cache, $dst[i]$ evicts the cache line loaded by $src[i]$ with a probability of $\frac{1}{N}$. Further assume that all accesses to $src[i]$ and $dst[i]$ in the loop individually fit in one cache line each. In this scenario, in the first iteration $src[i]$ and $dst[i]$ generate one (cold) miss each. If they both are loaded into different lines, which happens with probability $\frac{N-1}{N}$, then only two misses are observed since in all following iterations $src[i]$ and $dst[i]$ will hit in cache. If in the first iteration $dst[i]$ evicts $src[i]$, in following iterations more misses will be observed. Overall, the probability of experiencing two misses in one run is $P(2\ misses) = \frac{255}{256}$ as at least 2 misses will occur to fetch $src[i]$ and $dst[i]$ the first time. The probability of experiencing 3 or more misses is $P(\geq 3\ misses) = \frac{1}{256}$.

With 300 runs, there is a probability of $\left(\frac{255}{256}\right)^{300} \approx 0.31$ that only two misses are observed. This shows that the cache does not cause variability in the observed execution times. The only variability that would be captured arises from the bus being accessed when $src[i]$ and $dst[i]$ miss in the first iteration.

The application of MBPTA to this scenario would result in the input data passing all tests so that the user would obtain a pWCET estimate without revealing any anomaly. The obtained pWCET estimate would be, however, optimistic because it would account for the observed (low-scale) variability due to the small-scale source of jitter – the bus –, but not the large-scale variability due to the cache jitter, which is unlikely to occur in the limited size sample. This is illustrated in Figure 3 which shows two EVT projections. A first EVT projection called *EVT projection (1)* in which no observation

[4] An *arbitration window* refers to N consecutive bus arbitration rounds.

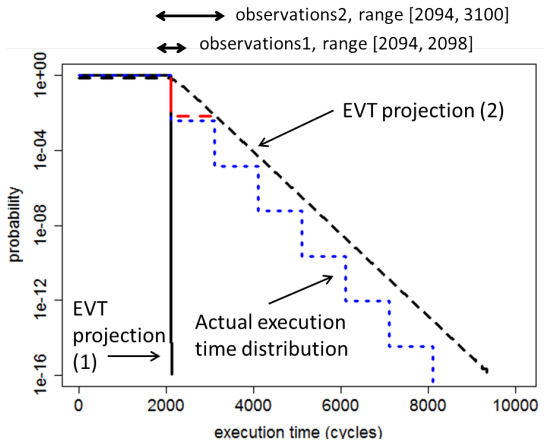

Fig. 3. Examples of a pathological scenario for MBPTA.

is made in which 3 or more misses happen. The $1 - CDF$ of those observations is shown as *observations1*, which spans the interval $[2094, 2098]$ cycles.

A second EVT projection, *EVT projection (2)*, is fed with *observations2*, where 3 misses are observed in at least one run, hence spanning $[2094, 3100]$ cycles. The dotted step line represents the actual 1-CDF of the program. We observe that *EVT projection (1)* is optimistic, i.e. does not upper-bound the actual probabilistic execution time of the program. This is so because *observations1* does not properly capture the timing behaviour of the function. Instead, *observations2* captures the first step of the function due to the fact that at least one run with 3 misses is captured, and so *EVT projection (2)* properly upper-bounds the timing behaviour of the program.

As explained in Section IV, MBPTA needs enough coverage in the execution time observations. This coverage is very likely not granted in this situation, because: (i) the highest latency for some random events is never triggered; (ii) no other random events trigger similar execution time variability; and (iii) the observed latencies do not provide information about unobserved ones.

B. Random Placement Caches

A situation similar to that discussed for random replacement may arise for random placement. The main difference is that the scale of execution time variability due to random-placement caches when set-associativity is very low (e.g., direct-mapped caches) can be inordinately larger than for random-replacement when fully-associative caches are used.

The root of the difference stems from the fact that the line in which a particular address is allocated in a fully-associative or set-associative cache (determined by the replacement policy) changes dynamically every time the line is evicted and then reloaded in cache, during program execution. This is possible because on every access all lines in a given set are scanned to determine whether there is a hit or miss. However, this is not the case for the placement in a set-associative or direct-mapped cache, which assigns a memory address to a given set. On every access, no search is carried out among all sets looking for a piece of data; instead only one set is accessed based on the address bits of the access. Hence, the address-to-set binding provided by the placement policy has to be maintained during program execution.

As a result, while random replacement takes a random decision at each replacement, random placement randomly places each address into a cache set *only once* per run, so that any conflict across different addresses in a given cache set will occur systematically for that run.

Listing 2 $int\ main\ ()$

```
1: int i, tmp, src[4000], dst[4000];
2: for (i = 0; i < 16; i = i + 2) do
3:     dst[i] = src[i] + dst[i];
4:     tmp = tmp + dst[i];
5: end for
6: for (i = 0; i < 4000; i + +) do
7:     dst[3999] = src[3999] + dst[3999];
8:     tmp = tmp + dst[3999]
9: end for
10: return(tmp);
```

Next we introduce a program example (see Listing 2) triggering a behaviour similar to the one shown in Figure 3 when a direct-mapped random-placement cache is used. This particular example is later used in the evaluation section to prove how our *Heart of Gold* technique detects pathological timing behaviour. We refer to this particular program as *corner* program, since it is a corner case.

The *corner* program performs two consecutive traversals of two vectors. In the first one, lines 2-5, different elements in the vectors are accessed. A number of different cache lines from the vectors are fetched. Eventually, the probability of having at least two cache lines mapped into the same set is relatively high. Also, three cache lines and/or multiple pairs of cache lines can conflict in different cache sets. However, any such conflict produces few misses. Thus, the impact in execution time of each conflict is just the latency of a couple of misses. In a second traversal, lines 6-9, few elements of the vectors mapped into 2 cache lines only are accessed a large number of times. The probability of those 2 cache lines to be mapped into the same cache set is very low. However, if they are mapped into the same set they evict each other on each access, thus producing a large number of misses and so, a large execution time increase. Therefore, the structure of this program emulates a source of frequent but low jitter (first traversal) and a source of infrequent but high jitter (second traversal). As shown later, execution times collected for this program pass the independence test but may lead to optimistic pWCET estimates.

VI. Heart of Gold: Detecting Corner Cases

MBPTA requires that the events under consideration, i.e. the observations of end-to-end runs, can be modelled with i.i.d. random variables. The independence and identical distribution tests can detect anomalous timing behaviour of the platform, either in the hardware or the software, thus preventing inappropriate use of MBPTA in deriving pWCET estimates. As any other statistical hypothesis test, the ones used by MBPTA may have *false negatives* with respect to the null-hypothesis, a.k.a type I errors, and *false positives* with respect to the null-hypothesis, a.k.a type II errors. In the former case, data are actually independent (or i.d.) but the test erronously rejects the indepenence hypothesis. In the latter, data are not actually independent (or i.d.) but the test cannot reject the hypothesis that they are. The significance level α, used in each test, determines the probability of false negatives since $\frac{\alpha}{1}$ of the cases data are independent (or i.d.) but the test statistically rejects this hypothesis. However, the fact that a PTA-compliant hardware platform is used guarantees the existence of ETPs [2], whereby the execution times incurred

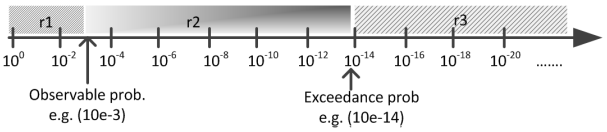

Fig. 4. Range of probabilities of interest when applying MBPTA

by the program under analysis can be described with an i.i.d. random variable, as needed by [8].

Heart of Gold (HoG) addresses the situation in which the execution times, obtained from a PTA-compliant architecture, thus being observations of an i.i.d. random variable, pass the i.i.d. tests but the distribution of the program execution time resembles the one in Figure 3, which causes MBPTA to yield optimistic pWCET estimates. The application of HoG to MBPTA requires identifying two probabilities of interest:

1) The *exceedance probability* ($P_{exceedance}$), with which any instance of program execution may exceed the assigned worst-case bound. The exceedance threshold of interest follows from the applicable safety standard and the rate of activation of the program. For commercial airborne systems at the highest integrity level (DAL-A) for example, the maximum allowed failure rate in a system component is 10^{-9} per hour of operation [1]. If exceeding a pWCET bound was assimilated to a component failure, and the program under analysis was released with a rate of 10^{-2} seconds (i.e. 10^2 activations per second), the pWCET of that task should have an exceedance probability in the $[10^{-14}, 10^{-15}]$ range: $10^{-9}\frac{failures}{hour} / \left(3600 \times 10^2 \frac{task\ activations}{hour}\right)$. An exceedance probability threshold of 10^{-15} should therefore suffice for the highest integrity level.
2) The *observable probability* ($P_{observable}$), which determines the lowest probability of occurrence that an event may have so that with high confidence it can be captured with the given number of observations done by MBPTA. For instance, if MBPTA is applied over $1,000$ observations, the probability of capturing in at least one observation the outcome of one event that happens with probability 10^{-2}, is $1 - 0.99^{1000} = 0.99996$.

These two probabilities define three ranges for any event, as shown in Figure 4. Range *r1* corresponds to the probability interval where a particular outcome for an event is very likely to be observed with the observations done with MBPTA. Range *r2* corresponds to the probability interval where that particular outcome is very unlikely to be observed, but it is still relevant for the correctness (i.e. non-optimism) of the pWCET estimate. Finally, range *r3* corresponds to the probability interval past the cut-off point: anything occurring with such a low probability is regarded as irrelevant in relation with the safety requirements of the system.

A. Rationale behind Heart of Gold

HoG aims at increasing the probability of capturing, in an affordable number of observation runs, the events deemed likely to lead to extreme execution times, which have low probability of occurrence: i.e., the events falling in range *r2*.

In time-randomised caches, high execution times may abruptly occur when the number of different addresses competing for a cache set exceeds the number W of physical lines in the set (a.k.a. associativity). To illustrate this situation we vary the number of addresses, each mapped to a different cache line, which compete for the same set. To this end we built a

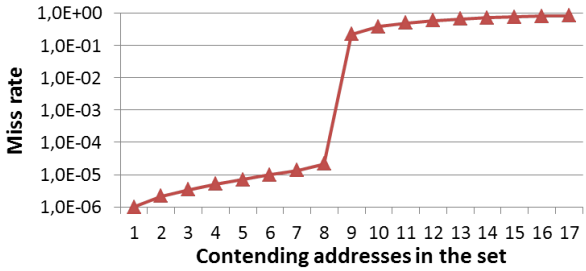

Fig. 5. Miss rates (in logarithmic scale) for different number of addresses accessed in a round-robin fashion competing for a 8-way cache set.

synthetic program that accesses a sequence of x addresses, $a_1, a_2, ...a_x$, 1,000,000 times for $W = 8$ (i.e. as with an 8-way cache). We run this program 1,000 times, collecting the average miss rate for different values of x. Figure 5 shows the miss rate as a function of the relation between x and W. While $x \leq W$ the miss rate grows within a tiny range. At the transition from $x = W$ to $x = W + 1$ an abrupt increase is observed (4 orders of magnitude). From $x = W+1$ onwards the miss rate resumes its modest linear growth. For the sanity of the MBPTA results, it is therefore crucial to determine whether execution times resulting from $W + 1$ addresses competing for the same cache set are included in the observations.

Consider a N-set direct-mapped cache using random placement policy. In that cache, the probability that, for a program accessing M different cache lines, all M lines collide in the same set is given by Equation 2.

$$P^M_{miss} = \left(\frac{1}{N}\right)^M \times N = \left(\frac{1}{N}\right)^{M-1} \quad (2)$$

If we were able to reduce the cache sets by half, then the probability that all M accesses collide in one line would become $\left(\frac{1}{\frac{N}{2}}\right)^{M-1}$, which can also be expressed as $\left(\frac{1}{N}\right)^{M-1} \times 2^{M-1}$, which respresents an increase of 2^{M-1} w.r.t to a cache of N entries. It follows that the probability of M different cache lines to conflict in the same cache set, thus leading to higher execution times, increases by a factor Δ^M_{Pmiss} as shown in Equation 3, where F is the factor by which the cache size is folded (reduced).

$$\Delta^{M,F}_{Pmiss} = F^{M-1} \quad (3)$$

The HoG principle rests on this appreciation to carry out MBPTA experiments on a target platform in which the cache size is reduced by a given folding factor, F. This practice increases the probability of occurrence of those cache-related events that fall in the range *r2* in the full-size cache, but fall in the *r1* range when the reduced cache is used, so that they can be actually observed. In particular, we seek to determine whether the case where at least $W+1$ addresses compete for a set has been observed (which would make it fall in *r1*) or not. If not (which makes it fall in *r2*), then the cache size needs to be decreased until this case is observed.

HoG focuses on set-associative caches as those are the most common cache designs in today's processors, although it can be directly used for direct-mapped caches. In the case of fully-associative caches, instead of decreasing the number of cache sets one should decrease the number of cache lines, but a similar reasoning would work. HoG involves the application of the following steps:

TABLE I
EXAMPLE WITH 3 ADDRESSES (A, B, C) AND 3 SETS ($S = 3$).

ABC,-,-	BC,A,-	A,-,BC	C,AB,-	-,AB,C	-,C,AB
AB,C,-	BC,-,A	B,AC,-	C,A,B	-,AC,B	-,-,ABC
AB,-,C	A,BC,-	B,A,C	C,B,A	-,BC,A	
AC,B,-	A,B,C	B,C,A	C,-,AB	-,A,BC	
AC,-,B	A,C,B	B,-,AC	-,ABC,-	-,B,AC	

Step 1: Setting the Cache Folding Factor F. Some characteristics of random placement related to HoG are the following:

- Random placement randomises the mapping of addresses to sets across runs, but a particular (random) mapping holds for the whole execution. Hence the number of cache line addresses competing in the same cache set solely depends on the number of *unique* cache line addresses U accessing the cache space simultaneously. During this discussion we assume that U corresponds to *all* addresses accessed by the program. The particular value of U to be considered for each program is discussed later in Section VI-B.
- Random placement is particularly devised to map addresses into a number of cache sets, S, where $S = 2^q$ and $q \in \mathbb{N}$; in other words, S must be a power-of-two.

As random placement provides a uniform and random distribution of addresses across sets [17], decreasing the cache size by a factor F (e.g., $F = 2$ means that the cache size is halved) increases the probability of a given set of M different addresses to be mapped in the same set by a factor of F^{M-1}.

Determining whether at least $W + 1$ addresses out of the U under consideration are mapped into the same cache set requires deriving all potential mappings of the U addresses into the S cache sets and the fraction of those mappings in which at least one cache set is allocated $W+1$ addresses. These values can be approximated by means of *weak compositions* theory, where a weak composition of an integer n is a way of writing n as the sum of a sequence of non-negative integers [12]. Here we are interested in all the weak compositions of U made of exactly S parts where at least one element is higher than W and the total number of weak compositions of U is exactly S sets. The latter, under which no limit is put on the values of the parts, is called $CompComb(U, S, -)$. The former, which covers the cases with no part having more than W elements, is called $CompComb(U, S, \leq W)$. The probability of a given address mapping to one of those, with one part greater than W, which we call $P_{extreme}(U, S, W)$, is then obtained as:

$$P_{extreme}(U, S, W) = 1 - \frac{CompComb(U, S, \leq W)}{CompComb(U, S, -)} \quad (4)$$

Obtaining $CompComb(U, S, -)$ and $CompComb(U, S, \leq W)$ can be done with appropriate mathematical/statistical packages or ad-hoc tools, presumably using large-number support, as the computed values are often in the range $[10^{30}, 10^{100}]$. As an example let's consider all combinations for 3 addresses and 3 sets. As shown in Table I, $CompComb(U = 3, S = 3, -) = 27$ and $CompComb(U = 3, S = 3, \leq 1) = 6$, so if $W = 1$, $P_{extreme}(3, 3, 1) = 1 - \frac{6}{27} \approx 0.778$ of the combinations would lead to extreme values. If $W = 2$, then $CompComb(U = 3, S = 3, \leq 2) = 24$ and $P_{extreme}(3, 3, 2) = 1 - \frac{24}{27} \approx 0.111$.

MBPTA makes N_{runs} runs, collecting one execution time observation from each of them. If a given event has a probability P_{event}, after N_{runs} the probability that it does not manifest in any run is $P_{event}^{not\ seen} = 1 - (1 - P_{event})^{N_{runs}}$.

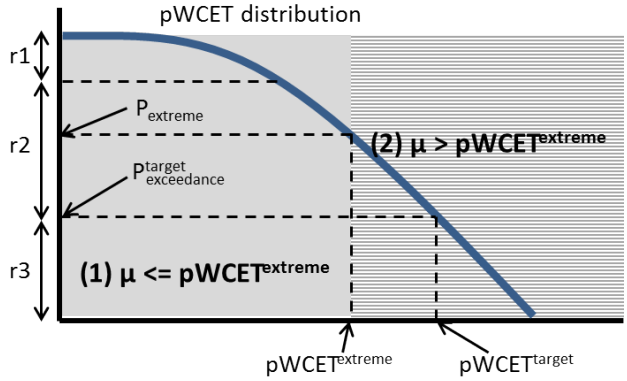

Fig. 6. Casuistic for the different HoG scenarios.

For any event with potentially significant effect on the program execution time, (such as e.g. $W + 1$ addresses being mapped to the same set) we want the probability $P_{event}^{not\ seen}$ of it not being observed in N_{runs} to be smaller than the failure rate for a system component allowed under the applicable safety requirements, e.g., 10^{-9} per hour of operation [1]. In this scenario we can compute the minimum probability so that an event like this one occurs, where $P_{event}^{min} = min(P_{event}) : P_{event}^{not\ seen} < 10^{-9}$. For instance, if $N_{runs} = 300$, the smallest number of runs under MBPTA, any event with probability $P_{event} \geq 0.067$ is not captured in any of the runs with a probability smaller than 10^{-9}. If N_{runs} were 1000, P_{event}^{min} would decrease to 0.02.

HoG focuses on cache related events for which $P_{exceedance} < P_{extreme}(U, S, W) < P_{event}^{min}$. It decreases the cache size by F so that $P_{extreme}(U, \frac{S}{F}, W) \geq P_{event}^{min}$ in order to guarantee that the scenarios of interest (those that may incur high execution time) can be observed.

Step 2: Collecting Results. In order to reduce the number of cache sets by a factor F as determined in the previous step, some simple hardware support is needed. This can be implemented by taking the set identifier delivered with the original cache size, splitting it into parts of $\log_2 \frac{S}{F}$ bits (the last part may have fewer bits, so it can be left-padded with zeros), and XORing those parts to generate the new cache set identifier for the decreased cache.

A meaningful number of runs must be collected using the resized cache memory. Given that collected execution times will be used to determine the average behaviour of the population, few hundreds of runs already provide high confidence. In particular, 1,084 runs provide 95% confidence on the results and an interval width of 5% [23], which are values in accord with common practice in statistics, which provide the same confidence as the i.i.d. tests used by MBPTA. If higher confidence is desired, the number of observations must be increased accordingly.

Step 3: Interpreting the Results. In most of the cases $F = 1$ suffices, as $P_{extreme}(U, S, W) > P_{min}^{event}$, and the process concludes successfully because MBPTA has all the data needed to obtain trustworthy pWCET estimates. We first classify the obtained results based on their average (μ) obtained for the used F and the $pWCET$ distribution obtained for the original cache arrangement. In particular, we are interested in the $pWCET$ obtained at the target exceedance probability $P_{exceedance}^{target}$, denoted $pWCET^{target}$, and the $pWCET$ obtained at the $P_{extreme}(U, S, W)$ exceedance probability,

denoted $pWCET^{extreme}$. Two different scenarios may then be singled out, which are depicted in Figure 6:

1) $\mu \leq pWCET^{extreme}$. The average execution time μ for the decreased cache size is below $pWCET^{extreme}$ meaning that execution times occurring at that probability (those with the extreme behaviour included) are still below the pWCET curve: there is no reason to distrust the $pWCET^{target}$ estimate. This corresponds to region (1) in Figure 6.
2) $\mu > pWCET^{extreme}$. Obtaining an average execution time higher than the $pWCET^{extreme}$ estimate is a clear indication that the execution time distribution may resemble the one in Figure 3 and the pWCET estimate should *not* be trusted. This corresponds to region (2) in Figure 6.

If μ falls in region (2), thus exceeding $pWCET^{extreme}$, it may be the case where the collected execution times include an abnormally high value whose actual probability is very low. For instance, an execution time with probability of occurrence 10^{-3} might well be observed within, say, just 10 runs. If this happens, μ will likely be above $pWCET^{extreme}$. In this case one may increase the volume of data collected with decreased cache size, as extra data can only increase confidence by decreasing the relative weight in the average of very low probability events. If the average execution time (μ) with the increased data, is such that $\mu \leq pWCET^{extreme}$, the pWCET can be trusted. How much extra data should be collected is up to the user to determine, which depends on the level of trust sought in the process.

Our experience with different programs – further confirmed in the following section – shows that $F = 1$ is the general case, which confirms that pWCET estimates obtained with "normal" MBPTA can be trusted. Only a synthetic benchmark running on a peculiar cache arrangement needed higher values for F and, in fact, fell in region (2).

B. Relevant Addresses for HoG

So far we have assumed that U, the relevant addresses for HoG, includes all unique addresses in the program. From Equation 4 it follows that, given a value of W and S, the higher the value of U the higher $P_{extreme}$, resulting in a potentially lower folding factor required to ensure $P_{extreme} \geq P_{event}^{miss}$. Hence, a careful selection of U is required so that it is not overestimated.

Let us illustrate this with an example. We assume a direct mapped data cache and a program containing a loop in which 3 addresses, ABC, are accessed. Further assume that outside the loop, in the prologue and epilogue of the program, addresses DEF are accessed once. In such a program the focus should be on the addresses in the loop, since when two of them are mapped to the same set a conflict will arise on every iteration of the loop, with significant impact on execution time. In fact, the higher the number of iterations the higher the impact of the conflicts among ABC. Interestingly, whether two of DEF are mapped to the same set has no impact on execution time since they are accessed once. Further, if ABC are mapped to different sets, the fact that DEF are mapped to the set where any of them is, also generates no impact on execution time as either they do not evict anything or evict data not to be reused. In this program, the value of U to be considered should be 3.

Overall, a correct determination of U requires understanding the program structure as well as identifying the access patterns in loops and recursive calls. Imbalanced access to the different addresses, as shown above, is a case of interest. In the general case one should analyse reuse distances and access patterns to determine the value of U to consider. It is also the case that HoG may need to be applied separately for different parts of a program, e.g. bottom up in the main program control flow graph, rather than only once for the entire program. Deepening the understanding of this problem is part of our current work.

The programs of interest in most safety-critical domains frequently have a loop-based structure, in which the application code subject to timing analysis is inside a main loop. As a result, accesses are balanced and any particular cache line can be potentially evicted by any other address in the program. This is the case for an avionics application we studied [20] as well as for the automotive benchmark suite used in this paper [24]. Thus, considering U as the set of unique addresses accessed in the main loop and simply excluding those accessed outside the loop, is an accurate approach.

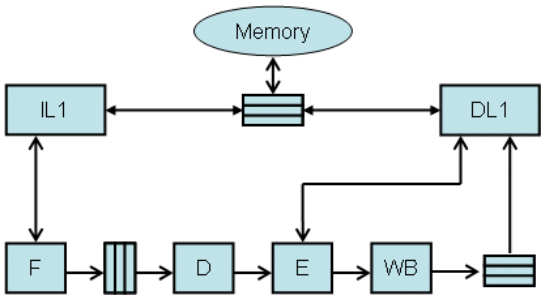

Fig. 7. Processor architecture

VII. Experimental Results

This section evaluates our method to quantify execution time variability in a concrete MBPTA-compliant architecture. First we present the experimental framework. Then we study the execution time variability obtained for a data cache under different processor configurations.

A. Experimental Framework

We use an enhanced version of the SoCLib simulation framework [27] with PowerPC binaries [29] to implement a cycle-accurate 4-stage pipelined in-order core architecture (fetch, decode, execute, write-back). The processor architecture is depicted in Figure 7. The memory hierarchy consists of separate instruction (IL1) and data (DL1) caches, and main memory. Both instruction and data caches are 8KB 8-way set-associative with 16-byte lines. Both caches implement random placement and random replacement.

In our experiments we used the EEMBC Autobench benchmark suite [24], a well-known reference in the automotive domain. We also use the *corner* program, *Listing 2*, which we execute on a 32KB direct-mapped and 16-byte lines DL1 cache and a perfect IL1.

We have collected pWCET estimates for all benchmarks using MBPTA [8] with a target exceedance threshold set at 10^{-15} per run, after the rationale presented in section VI(1).

B. Results

Table II shows $P_{extreme}(U, S, W)$ for all EEMBC benchmarks for both the DL1 and IL1 caches. As shown, for most of them the probability of getting more than W addresses mapped to one set, is close to 1. The lowest probability, 0.566 corresponds to `rspeed` for the DL1, and it is largely above $P_{event}^{min} = 0.067$.

TABLE II
$P_{extreme}(U, S, W)$ FOR DL1 AND IL1.

	DL1		IL1	
Bench.	U	$P_{extreme}(U, S, W)$	U	$P_{extreme}(U, S, W)$
a2time	739	1.000	964	1.000
aifftr	11,706	1.000	1,436	1.000
aifirf	852	1.000	905	1.000
aiifft	11,702	1.000	1,240	1.000
cacheb	587	1.000	1,023	1.000
canrdr	416	1.000	2,166	1.000
iirflt	806	1.000	1,611	1.000
puwmod	167	0.993	791	1.000
rspeed	102	0.566	372	1.000
tblook	1,381	1.000	647	1.000
ttsprk	460	1.000	495	1.000

TABLE III
RESULTS FOR THE *corner* BENCHMARK.

Parameter	**Value**		
F	1	2	4
$P_{extreme}(U, \frac{S}{F}, W)$	0.00098	0.00195	0.00390
μ	186,168	192,430	200,509

In fact we have observed that for typical cache size and associativity pairs for first and second level caches, if $U \geq 0.15 \cdot S \cdot W$, then $P_{extreme}(U, S, W) > P_{event}^{min}$, whereby pWCET estimates are trustworthy. In other words, if U stands for 15% of the cache space, then pWCET estimates are trustworthy as extreme execution times are already included in the data input to MBPTA.

We have also analysed the behaviour of the *corner* benchmark for the DL1. As expected, *Heart of Gold* detects the anomalous behaviour of that program. Results are shown in Table III, where we show $P_{extreme}(U, \frac{S}{F}, W)$ and the average execution time (μ) for different values of F. In all cases, $P_{extreme}(U, \frac{S}{F}, W)$ sits below $P_{event}^{min} = 0.067$, but μ increases for decreasing cache sizes ($F > 1$), rising above the $pWCET^{extreme}$ obtained for full-size cache (189,374 cycles). Hence, decreasing cache size reveals that the pWCET estimate *cannot* be trusted as the scenarios leading to extreme execution times did not occur with the full-size cache.

C. Corollary

While MBPTA applied on top of time-randomised platforms can be regarded as a trustworthy method, our detailed analysis of different cache architectures and illustrative programs exposes a crucial fact: *for any time-randomised platform with HW/SW induced random events, one can devise a program such that a large execution time increase occurs with a probability low enough so that those execution times are unlikely to be included in the sample data provided to MBPTA*. We have performed this analysis for different cache architectures including random-placement and random-replacement set-associative, direct-mapped and fully-associative caches, as well as other cache artifacts described in [6], and for all of them we were able to construct such a program. We may therefore conclude that the *Heart of Gold* technique is a required step to assure that the application of MBPTA to that program will not yield anomalous results and thus the pWCET estimates derived with it are trustworthy.

We have also observed that, in general, if the amount of data competing for the cache space is above 15% of the cache size, pWCET estimates are already trustworthy. If this is not the case, decreasing the cache size in the observation runs suffices to accurately determine whether pWCET estimates are trustworthy. Our experience with real applications [28] has enabled us to understand that timing behaviour that causes hazard to the application of MBPTA is only triggered by programs that only exploit spatial locality in the last cache line fetched (e.g., vector and array traversals), not making any meaningful reuse of data fetched in cache.

VIII. RELATED WORK

A number of timing analysis techniques have been proposed [30]. Static timing analysis (STA) requires plenty of information from the user to derive tight WCET estimates, and may introduce significant pessimism when some information is missing (e.g., memory addresses). Alternatively, measurement-based timing analysis (MBTA) techniques rely on performing extensive testing on the real system with stressful, high-coverage input data, and derive a WCET estimate based on the highest execution time observed adding to it an engineering margin to account for unobserved scenarios. However, determining such margin is challenging in the presence of features producing abrupt timing variations such as cache memories.

PTA has recently emerged as an alternative to conventional timing analysis techniques [6], [8]. PTA aims to reduce the amount of information required to obtain trustworthy and tight WCET estimates. This is particularly true for MBPTA [8], [28], which removes the need for information about many sources of execution time variation [7] such as, for example, the actual addresses accessed by the program under analysis. Further PTA extensions address time composability in either a static [9] or a measurement-based way [20], and to extend/optimise the use of SPTA [3], [22].

MBPTA relies on a smart use of EVT. Other approaches have used EVT for similar problems [4], [10], [15], [25] but on non-PTA-compliant platforms, hence they still require a (comparatively) large amount of information from the user about the sources of execution time variation. Conversely, MBPTA reduces significantly its information need by either upper-bounding latencies for some hardware resources or time-randomising caches [17], [18] (or a software alternative [19]) and buses [16]. Moreover, MBPTA has also recently been used to derive pWCET estimates for faulty cache memories [26].

IX. CONCLUSIONS

The advent of MBPTA as a probabilistic WCET estimation method for critical real-time embedded systems opens the door to obtain trustworthy and tight pWCET upper-bounds at low cost. However, it has been proven that some programs run on top of PTA-compliant hardware platforms may exhibit pathological timing behaviour, which challenges the trustworthiness of MBPTA results.

This paper deconstructs MBPTA and PTA-compliant hardware features so that pathological program behaviour and its consequences can be understood. We show some realistic cases where such behaviour shows up and MBPTA fails to provide pWCET upper-bounds. Finally, we illustrate how those cases where pWCET estimates are optimistic can be identified by means of a new practical, low-cost technique, *Heart of Gold*, thus increasing the trustworthiness of the pWCET estimates obtained with MBPTA.

ACKNOWLEDGMENTS

The work presented in this paper has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement 611085 (PROXIMA,

www.proxima-project.eu). This work has also been partially supported by the Spanish Ministry of Science and Innovation under grant TIN2012-34557 and the HiPEAC Network of Excellence. This joint collaboration was facilitated by the EU COST Action IC1202: Timing Analysis On Code-Level (TACLe). The authors acknowledge Robert I. Davis and Miquel Moretó for their contribution to this work.

REFERENCES

[1] Guidelines and methods for conducting the safety assessment process on civil airborne systems and equipment. *ARP4761*, 2001.
[2] J. Abella, F.J. Cazorla, E. Quinones, and T. Vardanega. Measurement-based probabilistic timing analysis and i.i.d property. White Paper. 2013. http://www.proartis-project.eu/publications/MBPTA-white-paper.
[3] S. Altmeyer and R.I. Davis. On the correctness, optimality and precision of static probabilistic timing analysis. In *DATE*, 2014.
[4] G. Bernat, A. Colin, and S.M. Petters. WCET analysis of probabilistic hard real-time systems. In *RTSS*, 2002.
[5] Bradley. *Distribution-Free Statistical Tests*. Prentice-Hall, 1968.
[6] F.J. Cazorla, E. Quiñones, T. Vardanega, L. Cucu, B. Triquet, G. Bernat, E. Berger, J. Abella, F. Wartel, M. Houston, L. Santinelli, L. Kosmidis, C. Lo, and D. Maxim. PROARTIS: Probabilistically analysable real-time systems. *ACM TECS*, may 2013.
[7] F.J. Cazorla, T. Vardanega, E. Quinones, and J. Abella. Upper-bounding program execution time with extreme value theory. In *WCET workshop*, 2013.
[8] L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzetti, E. Quinones, and F.J. Cazorla. Measurement-based probabilistic timing analysis for multi-path programs. In *the 23rd ECRTS*, 2012.
[9] R.I. Davis, L. Santinelli, S. Altmeyer, C. Maiza, and L. Cucu-Grosjean. Analysis of probabilistic cache related pre-emption delays. In *ECRTS*, 2013.
[10] S. Edgar and A. Burns. Statistical analysis of WCET for scheduling. In *the 22nd IEEE Real-Time Systems Symposium (RTSS01)*, pages 215–225, 2001.
[11] W. Feller. *An introduction to Probability Theory and Its Applications*. 1996.
[12] P. Flajolet and R. Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 2009.
[13] M. Garrido and J. Diebolt. The ET test, a goodness-of-fit test for the distribution tail. In *Methodology, Practice and Inference, second international conference on mathematical methods in reliability*, pages 427–430, 2000.
[14] B.V. Gnedenko. Sur la distribution limite du terme maximum d'une seris aleatoire. *Annals of Mathematics*, 44:423–453, 1943.
[15] J. Hansen, S. Hissam, and G. A. Moreno. Statistical-based wcet estimation and validation. In *the 9th International Workshop on Worst-Case Execution Time (WCET) Analysis*, 2009.
[16] J. Jalle, L. Kosmidis, J. Abella, E. Quinones, and F.J. Cazorla. Bus designs for probabilistic multicore processors. In *DATE*, 2014.
[17] L. Kosmidis, J. Abella, E. Quinones, and F.J. Cazorla. A cache design for probabilistically analysable real-time systems. In *DATE*, 2013.
[18] L. Kosmidis, J. Abella, E. Quinones, and F.J. Cazorla. Multi-level unified caches for probabilistically time analysable real-time systems. In *RTSS*, 2013.
[19] L. Kosmidis, C. Curtsinger, E. Quinones, J. Abella, E. Berger, and F.J. Cazorla. Probabilistic timing analysis on conventional cache designs. In *DATE*, 2013.
[20] L. Kosmidis, E. Quinones, J. Abella, T. Vardanega, and F.J. Cazorla. Achieving timing composability with measurement-based probabilistic timing analysis. In *ISORC*, 2013.
[21] S. Kotz and S. Nadarajah. *Extreme value distributions: theory and applications*. World Scientific, 2000.
[22] D. Maxim, M. Houston, L. Santinelli, G. Bernat, R.I. Davis, and L. Cucu-Grosjean. Re-sampling for statistical timing analysis of real-time systems. In *RTNS*, 2012.
[23] M. McCabe. *Introduction to the Practice of Statistics*. Freeman & Co., 1989.
[24] Jason Poovey. *Characterization of the EEMBC Benchmark Suite*. North Carolina State University, 2007.
[25] P. Radojković, V. Čakarević, M. Moretó, J. Verdú, A. Pajuelo, F.J. Cazorla, M. Nemirovsky, and M. Valero. Optimal task assignment in multithreaded processors: A statistical approach. In *ASPLOS*, 2012.
[26] M. Slijepcevic, L. Kosmidis, J. Abella, E. Quinones, and F.J. Cazorla. DTM: Degraded test mode for fault-aware probabilistic timing analysis. In *ECRTS*, 2013.
[27] SoCLib. http://www.soclib.fr/trac/dev.
[28] F. Wartel, L. Kosmidis, C. Lo, B. Triquet, E. Quinones, J. Abella, A. Gogonel, A. Baldovin, E. Mezzetti, L. Cucu, T. Vardanega, and F.J. Cazorla. Measurement-based probabilistic timing analysis: Lessons from an integrated-modular avionics case study. In *SIES*, 2013.
[29] J. Wetzel, E. Silha, C. May, B. Frey, J. Furukawa, and G. Frazier. *PowerPC User Instruction Set Architecture*. IBM Corporation, 2005.
[30] Wilhelm R. et al. The worst-case execution-time problem overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems*, 7:1–53, May 2008.

APPENDIX I

This appendix provides details about the internals of i.i.d. tests.

A. Wald-Wolfowitz Independence Test

The Wald-Wolfowitz independence test (also known as *runs test*) indicates whether a sequence of binary events are biased or random. In the context of MBPTA, the sequence of binary events to test is built by checking whether a particular execution time of the program under analysis is higher or lower than the median of all observed runs [8]. Events matching exactly the median can be considered higher always, considered lower always, or simply ignored for the generation of the sequence of binary events. Without loss of generality, in the rest of the explanation we assume that they are always considered lower.

Given a sequence of execution times $E_{i=1,..,N}$ and the median $\tilde{E}$ of them we obtain a sequence of N elements $e_{i=1,..,N}$ in the alphabet $\{H, L\}$ (H means higher, L means lower) such that

$$e_i = \begin{cases} H, & E_i > \tilde{E} \\ L, & E_i \leq \tilde{E} \end{cases} \quad (5)$$

The term *run* refers to a sequence of consecutive identical values in the input sequence of the test. For instance, in the sequence $HHHLHHLLHLLH$ there are 7 runs. We refer to the total number of runs as r. The distribution of the number of runs r is approximately normal with [5]:

$$\mu_r = \frac{2 \cdot n_H \cdot n_L}{N} + 1$$
$$\sigma_r = \frac{2 \cdot n_H \cdot n_L \cdot (2 \cdot n_H \cdot n_L - N)}{N^2 \cdot (N-1)}$$

where n_H and n_L correspond to the number of e_i elements that are H or L respectively, so that $N = n_H + n_L$.

Thus, if execution times are truly independent, the following equation holds:

$$Z = \frac{r - \mu_r}{\sigma_r} \rightarrow N(0, 1)$$

Z will be, typically, close to 0. According to [8], the significance level used is 5% so that any value outside of the 95% central region of the distribution is rejected, thus indicating non-independence. This significance level implies that $|Z| < 1.96$ to consider that data are truly independent.

B. Kolmogorov-Smirnov Identical Distribution Test

The two-sample Kolmogorov-Smirnov test is used to prove whether two samples of data correspond to the same distribution. This test computes a parameter called *p-value* (pv) as a function of the absolute maximum distance between the empirical cumulative distribution function (ECDF) of both samples normalised. The p-value is always in the range $[0, 1]$. The higher the distance, the lower the p-value is.

Analogously to the independence test, a significance level of 5% is used [8] so that the null hypothesis is rejected if the p-value is below the significance level: $pv \leq 0.05$ in our case. Conversely, if $pv > 0.05$ the null hypothesis is not rejected and it is assumed that identical distribution is proven.