

Measurement-Based Cache Representativeness on Multipath Programs

Suzana Milutinovic
Universitat Politècnica
de Catalunya and BSC

Jaume Abella, Enrico Mezzetti
Barcelona Supercomputing
Center (BSC)

Francisco J. Cazorla
BSC and IIIA-CSIC

ABSTRACT

Autonomous vehicles in embedded real-time systems increase critical software size and complexity whose performance needs are covered with high-performance hardware features like caches, which however hampers obtaining WCET estimates that hold valid for all program execution paths. This requires assessing that all cache layouts have been properly factored in the WCET process. For measurement-based timing analysis, the most common analysis method, we provide a solution to achieve cache representativeness and full path coverage: we create a *modified* program for analysis purposes where cache impact is upper-bounded across any path, and derive the minimum number of runs required to capture in the test campaign cache layouts resulting in high execution times.

CCS CONCEPTS

•Computer systems organization → Real-time systems;

KEYWORDS

Cache, randomization, multipath programs, probabilistic analysis

1 INTRODUCTION

Most new services in real-time products (e.g. cars) are totally or partially provided by software, making it one of the most important factors to increase competitive edge [11]. The increasing complexity of software, to manage huge and diverse types of data and for decision making, heavily complicates timing validation and verification. This is compounded by the use of performance-improving hardware features (e.g. caches) to cover unprecedented critical software's performance needs, e.g. expected to increase by 100x in the next years in the automotive domain [34]. Increased hardware and software complexity hampers the confidence that can be put on WCET estimates derived by measurement-based timing analysis (MBTA), the most used timing analysis technique in critical real-time embedded systems [37]: the complex control-flow graph of software seriously complicates providing input data (vectors) that exercise all program paths; whereas caches create huge execution time variability since the way in which applications' objects are mapped in memory – on which users have decreasing control – determines their cache layout and application's performance.

The increased variability in the execution time of a task, whose distribution can have arbitrary variance and shape, has motivated several research works to use statistical techniques to derive bounds to execution time distributions. This has resulted in a variant of MBTA referred to as Measurement-based probabilistic timing analysis (MBPTA) [2]. MBPTA delivers a probabilistic WCET (pWCET) function (see Figure 1(a)) upper-bounding the execution time distribution (pETd) of the program at any exceeding probability. For instance, in Figure 1(a) the probability that execution time exceeds 25 ms is at most 10^{-10} . MBPTA has been applied to avionics [35] and space [9] case studies and its impact on certification has also been addressed [33]. MBPTA has been complemented with solutions that inject randomization in program's timing behavior to relieve the user from controlling those *jittery resources* increasing program's execution time variability. This is achieved by hardware techniques (e.g. random arbitration policies and random placement/re-placement techniques) that are now part of a commercial product for the space domain [5]; and software techniques that work at the compiler/linker level or source code level [15].

Randomization makes that all potential behaviors that a given jittery resource (e.g. caches) can have are naturally (and randomly) explored in every new test, enabling the derivation of probabilistic guarantees. As a result, the user is relieved from controlling memory mapping, which is arguably complex (in fact infeasible as software complexity grows) [20]. As a side effect, the execution of a program on a randomized platform results in a probabilistic execution time distribution (pETd), see dashed line in Figure 1(a).

Time-aware Address Conflict (TAC) [24] works on top of MBPTA to factor in cache layouts building on randomization properties. TAC guarantees that the number of execution time measurements fed to MBPTA is large enough to observe the effect of all relevant cache placements, whose probability of occurrence may be sufficiently low not to be observed in the default number of runs of MBPTA [1]. If the number of runs satisfies this criteria, measurements are regarded as *cache-layout representative*. However, TAC has only been demonstrated on single-path analysis.

Path Upper-Bounding (PUB) builds an *upper-bounding program* [14] to automatically extend path coverage beyond the subset of paths exercised with user's input vectors, effectively relieving the user from the enumeration of all paths. Nonetheless, the required minimum number of runs required for factoring in cache representativeness for a reliable application of MBPTA has not been explored in the scope of automatic full-path coverage, as provided by PUB.

Overall, no existing MBPTA solution attains cache representativeness and full path coverage simultaneously. We cover this gap by proposing the first approach to determine the minimum number of runs required in MBPTA-compliant platforms for MBPTA techniques delivering full path coverage automatically. In particular, this paper (1) devises a reliable combined application of PUB and TAC mechanisms by analyzing the relation between the memory

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

DAC '18, San Francisco, CA, USA

© 2018 ACM. 978-1-4503-5700-5/18/06...\$15.00

DOI: 10.1145/3195970.3196075

address sequences in the original program and its modified (*pubbed*) version, thus identifying key properties that guide the joint application of PUB and TAC. This work also (2) provides evidence of the reliability of the approach, and (3) evaluates its cost on the Mälardalen benchmark suite.

2 BACKGROUND

MBPTA [2] applies Extreme Value Theory (EVT) [6, 16] on a set of execution time measurements, which must meet certain statistical properties (e.g. independence and identical distribution), and determines the best set of maxima values of the sample to be used to estimate the pWCET.

TAC. EVT can only estimate the combined effects of events observed in its input set (execution time values). For instance, some cache placements may produce arbitrarily higher execution times than others when the capacity of one or more cache sets is exceeded, and accesses to addresses mapping to those sets are interleaved with long reuse distances (e.g. in a round-robin fashion). EVT is unable to predict those events (and combinations thereof) when they are not observed in the execution time measurements passed as input [1]. TAC [24] analyzes the address sequence of the program under analysis, and determines the particular groups of addresses with larger impact on execution time when placed together in a set. Then, by correlating their impact with their estimated probability of occurrence, TAC determines the number of execution time measurements needed to guarantee that they are observed with a sufficiently high probability. For instance, the number of runs is sized so that the probability of missing such events is lower than 10^{-9} , thus in line with the most stringent fault probabilities allowed for hardware components.

PUB builds a modified version of the program (P_{pub}) such that any path of P_{pub} exhibits an execution time distribution that upper-bounds those of all paths in the original program P_{orig} .

$$\forall i, j \in paths(P_{orig}) : F(P_{orig}^i(t)) \geq F(P_{pub}^j(t)) \quad (1)$$

That is, the accumulated probability ($F(x)$) of any path i of the original program for any execution time t is equal or higher than for any path j of the *pubbed* version of the program. Note that the extended program is used only to generate the pWCET at analysis, while at deployment time the original unmodified program is used.

PUB extends the source code of the original program with a compiler pass that inflates the code of each branch of conditional constructs (e.g. if-then-else, switch) with instructions and memory accesses so that each branch in the *pubbed* code includes equivalent cache access patterns in all of the branches of the original conditional construct¹. By applying this concept recursively starting from the innermost conditional constructs, operations and memory patterns can be upper-bounded. Interestingly, PUB inserts the needed memory accesses with functionally-innocuous operations (e.g. loading data into a read-only register). PUB builds on input vectors triggering the highest loop bounds and all basic blocks (but not all paths), which is a common requirement of safety standards for code coverage. PUB relieves end users from having to control memory mapping, values operated and paths traversed, which are much less obvious from code inspection.

¹For clarity we refer to the sequence of addresses of one path, regardless of whether they are instructions or data since the reasoning provided in this paper applies to both.

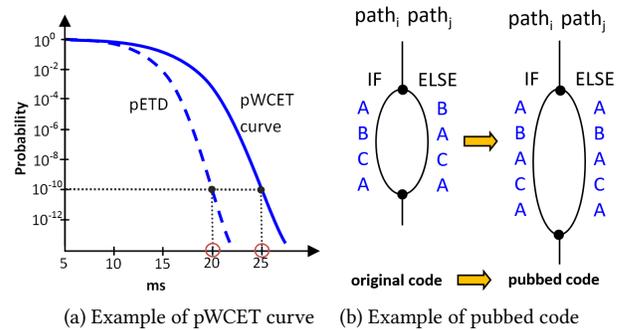


Figure 1: Basic concepts on MBPTA and PUB

PUB relies on a key property of time-randomized caches that does not hold for time-deterministic caches: given a memory access sequence, adding any memory access in any position necessarily causes a worse probabilistic execution time distribution [14]. For instance, given an if-then-else where the ‘if’ branch contains the memory access sequence $M_{if} = \{ABCA\}$ and the ‘else’ branch $M_{else} = \{BACA\}$, the sequence $M_{pub} = \{ABACA\}$ is an upper-bound for both (see Figure 1(b)). If we compare M_{if} and M_{pub} , the first two accesses (A and B) will have the same probabilistic behavior. If the third access in M_{pub} is a hit (A), cache contents are not altered and the rest of the sequence is identical to the remaining part of M_{if} . Hence, M_{pub} increases execution time by 1 hit. Conversely, if the third access is a miss, the fifth access (A again) has higher probability to hit. Thus, M_{pub} will experience 4 misses in the first 4 accesses and a likely-hit in the last access. Instead, M_{if} will experience at most 4 misses, hence having a shorter execution time than M_{pub} . We refer the reader to the PUB technique for further proofs, details on instruction cache specifics and memory alignment issues [14]. Note that, this method is not compatible with time-deterministic caches [14] since, for instance, Least-Recently Used (LRU) replacement policies may lead to lower execution time by inserting accesses. E.g. in a 2-way cache the sequence $\{ABCA\}$ experiences 4 misses (all accesses), whereas $\{ABACA\}$ experiences only 3 (second and third occurrence of A are hits).

3 ACHIEVING FULL PATH COVERAGE AND CACHE REPRESENTATIVENESS

3.1 Difficulties integrating PUB and TAC

In order to reach full path coverage, PUB extended programs result in address sequences that highly likely differ from that of the original program. This complicates reasoning on TAC, i.e. on the number of runs to ensure that random cache layouts causing high execution times are captured. Intuitively, since PUB adds cache accesses to the original program, the number of runs required for TAC reduces, since the more the addresses in a program the more likely they conflict in the same cache set, and hence the lower the number of runs to randomly hit one of those cases. However, this is not the case. To illustrate so, let M_{orig}^j be the address sequence of path j . Let $ins(M, x)$ be an operator to insert an address x in a sequence M , which can be done in multiple ways, as long as the same ordering of address accesses in the original path is preserved. The sequence of addresses of a path j in the *pubbed* program (M_{pub}^j) is the result of inserting several addresses, $n_j \geq 0$, in M_{orig}^j .

$$\forall j \in \text{paths}(P) : M_{pub}^j = \text{ins}(\dots(\text{ins}(\text{ins}(M_{orig}^j, x_1), x_2)\dots), x_{n_j}) \quad (2)$$

No relationship can be established between the number of runs required to achieve representativeness in M_{orig}^j and M_{pub}^j . That is, $R_{TAC}(M_{orig}^j)$ can be higher/lower than $R_{TAC}(M_{pub}^j)$, where $R_{TAC}(M)$ is the minimum number of runs required for address sequence M as determined by TAC. This is shown with two examples.

3.1.1 $R_{TAC}(M_{orig}^j) < R_{TAC}(M_{pub}^j)$. Let us assume a program with two paths, P_{orig}^1 and P_{orig}^2 with address sequences $M_{orig}^1 = \{ABCA\}^{1000}$ and $M_{orig}^2 = \{ADEA\}^{1000}$, and a cache with $S = 8$ cache sets and $W = 4$ ways. The exponent of the address sequences represents the number of times that those address sequences repeat. PUB application tries to minimize the number of addresses inserted to minimize the impact in the pWCET estimate. Hence, a potential result of applying PUB could be as follows:

$$\begin{aligned} M_{pub}^1 &= M_{pub}^2 = (\text{ins}(\text{ins}(M_{orig}^1, D), E))^{1000} = \\ &(\text{ins}(\text{ins}(M_{orig}^2, B), C))^{1000} = \{ABCDEA\}^{1000} \end{aligned} \quad (3)$$

Since both M_{orig}^1 and M_{orig}^2 have 3 different addresses each, none of them can exceed the space in a cache set (4 ways), i.e. they cannot generate a cache layout resulting in high execution time. As a result, TAC does not impose a minimum number of runs higher than that required by the standard application of MBPTA. Note that, for instance, in the case of M_{orig}^1 addresses A , B and C will end up fitting in a cache set after, potentially, few random replacements [24].

Instead, M_{pub}^1 (and M_{pub}^2) has 5 different addresses, so it exceeds the space in a cache set. For instance, if addresses A , B , C , D and E are randomly placed in the same set, they will experience at least 1000 misses due to the address not present in the cache set in each of the 1000 traversals of the address sequence. Hence, TAC determines that a sufficient number of runs is needed to guarantee that this sequence is not observed with a maximum configurable probability. The probability of observing such a sequence is exactly $(1/S)^4 = 0.000244$, so the probability of not observing it in R runs is $(1 - 0.000244)^R$. If such probability needs to be lower than 10^{-9} to be regarded as negligible w.r.t. the corresponding safety standard [24], then $R > 84875$. Hence, PUB address sequences may require a higher minimum number of runs than the original ones.

3.1.2 $R_{TAC}(M_{orig}^j) > R_{TAC}(M_{pub}^j)$. Let us now consider that $M_{orig}^1 = \{ABCDEA\}^{1000}$ and $M_{orig}^2 = \{ABCDFA\}^{1000}$. In this case, and building upon the results obtained in the example before, each original path would require $R > 84875$ runs. M_{pub}^1 (and M_{pub}^2) would be $\{ABCDEFDA\}^{1000}$, thus including 6 different addresses. Abrupt cache miss counts would require 5 out of the 6 addresses to be placed in the same set, whose probability is $(1/S)^4 \cdot 6 = 0.00146$. Therefore, $R > 14138$. Hence, PUB address sequences may require a lower minimum number of runs than the original sequences.

3.2 Sensible Application of PUB and TAC

Since no relation can be established between the minimum number of runs required by the original program and its *pubbed* version,

our method relates both using only the pWCET and their execution time distributions. To that end we build on the following reasoning.

Observation 1: *The probabilistic execution time distribution (pETd) obtained for any given path of the pubbed program is an upper-bound to the pETd of any path of the original program [14].*

This observation is formalized in Equation 1. Note that the pETd refers to those that would be obtained by running the corresponding program paths an infinite number of times. Thus, while this observation would generally hold also for the statistical samples obtained from the pETd, it can only be guaranteed to hold for the reference distributions due to the statistical nature of samples.

Observation 2: *The pWCET curve obtained from the sample of a pETd is (probabilistically) an upper-bound for the execution time distribution that has been sampled [2].*

Under the appropriate setup for execution time collection, e.g. ensuring independence across experiments, and sources of randomization injected at hardware or software level, it has been shown that pWCET estimates upper-bound pETd [2, 27].

Corollary 1: *From Observations 1 and 2, it follows that the pWCET curve obtained from a sample of any given path of a pubbed program upper-bounds the pETd of any path of the original program.*

From Corollary 1, we make the following key observation:

Observation 3: *Any path choice from the pubbed program is equally valid to derive a minimum number of runs and a reliable pWCET distribution that upper-bounds the pETd of all paths in the original program.*

For instance, by applying TAC on an arbitrary path P_{pub}^j we obtain that the minimum number of runs needed for cache representativeness is R_{pub}^j . Since $pWCET(P_{pub}^j)$, when it is derived from a sample of at least R_{pub}^j runs, is an upper-bound for the pETd of any path of the original program, $pWCET(P_{pub}^j)$ achieves both *full path upper-bounding and cache representativeness*.

Some observations must be made to complete the view on the approach taken to use PUB and TAC appropriately.

Observation 4: *For two different paths j and k of a pubbed program, TAC results in R_{pub}^j and R_{pub}^k minimum number of runs, respectively. In general, no relation can be established between R_{pub}^j and R_{pub}^k since the insertion of the missing addresses in each path with $\text{ins}(M, x)$ may not lead to identical address sequences.*

For instance, if $M_{orig}^1 = \{ABA\}$ and $M_{orig}^2 = \{ACA\}$, by applying PUB we could obtain $M_{pub}^1 = \{ABCA\}$ and $M_{pub}^2 = \{ACBA\}$. In the general case, address sequences can be arbitrarily different, thus leading to different results when applying TAC. This is analogous to the scenario when relating the number of runs of the original and *pubbed* paths, described in Section 3.1.

Nevertheless, building on previous appreciation, even if a different *pubbed* path is used, the pWCET remains reliable and representative w.r.t. all paths in the original program. Building on the reasoning above, we also note that:

Observation 5: *The pWCET estimates obtained for P_{pub}^j and P_{pub}^k cannot be related to each other. In other words, potentially, the pWCET estimate for path P_{pub}^j can be above or below that of P_{pub}^k for the full range of probabilities or only for a subset of them. In any case, this is irrelevant given that both pWCET estimates are reliable and representative upper-bounds of all paths of the original program.*

Corollary 2: *Since every single pWCET estimate for each pubbed path is a reliable and representative upper-bound for the pETd of all*

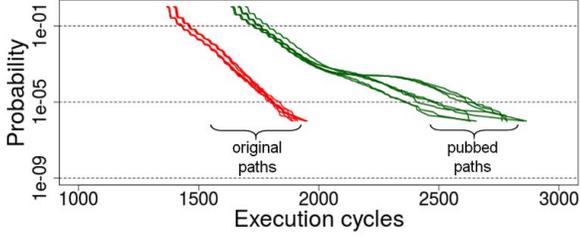


Figure 2: ECCDF for bs’s original paths and pubbed paths.

paths of the original program, we can take as pWCET for any given exceedance threshold the lowest value across all pubbed paths.

Corollary 2 builds upon the fact that, the pWCET estimates of all paths are equally reliable and representative. Differences only relate to pessimism, so we can trade analysis cost for tightness by analyzing an increased number of paths of the *pubbed* program. While only one is strictly needed, the larger the number of paths analyzed, the higher the chances of obtaining tighter pWCET estimates. Note, however, that no guarantee exists on whether tighter pWCET estimates will be obtained since it could turn to be that the lowest one across the available paths is the first one we obtain.

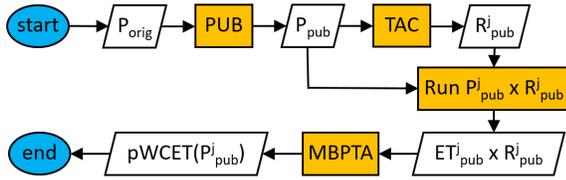


Figure 3: Overall application process of PUB and TAC.

Overall, the application process of PUB and TAC comprises the steps shown in Figure 3.² Starting from the original program P_{orig} we apply PUB to generate the pubbed program P_{pub} . By default, we choose an arbitrary input vector provided by the user, which executes path j , to collect its address sequence. Optionally, we may explore more paths to search for tighter pWCETs, as explained before. We analyze the address sequence with TAC to derive the number of measurements required, R^j_{pub} . We execute the pubbed program with the same input vector as many times as dictated by TAC, thus producing a sample of execution times, ET^j_{pub} . Finally, we apply MBPTA on the resulting sample to obtain a pWCET estimate that reliably upper-bounds any path of the original program, under any layout of objects in memory/cache.

3.3 A Practical Example

We illustrate how our method works with an example based on the binary search (bs) benchmark from the Mälardalen suite [10]. Details on the hardware setup are provided later in Section 4. Bs is a multipath program whose input determines the number of loop iterations and the actual path traversed. For the sake of illustration we stick to its default input to set the loop bounds (15 integer elements), and explore the different type of input vectors (related to the degree of ordering of the elements to sort) that lead to the maximum number of iterations. In particular, 8 different cases lead to different paths triggering the maximum number of iterations.

²We omit details on PUB and TAC implementation due to space constraints and refer an interested reader to original works [14, 24].

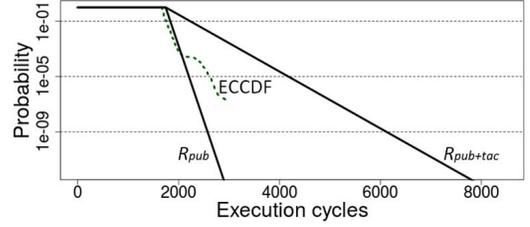


Figure 4: pWCET for bs v_9 with R_{pub} and R_{p+t} runs.

The first experiment provides evidence for Corollary 1: we collect 1,000,000 execution times for each of the 8 original paths characterized by the maximum number of iterations, before and after applying PUB. Their Empirical Complementary Cumulative Distribution Function (ECCDF) is shown in Figure 2. As it can be seen, each *pubbed* path upper-bounds *all* original paths. Hence, when applying MBPTA to any pubbed program, resulting pWCET curves upper-bound all original paths. For instance, the highest observed execution time across all paths is below 2,000 cycles, whereas the lowest pWCET estimate across all *pubbed* paths at an exceedance probability of 10^{-6} per run (so at the same probability as the highest execution time, $1/R = 10^{-6}$) is 2,297 cycles (for path v_9).

In a second experiment, we compute the pWCET curve for input v_9 , with $R_{pub} = 1,000$ and $R_{pub+tac} = 70,000$ (as computed by TAC), see Figure 4 that also shows the ECCDF with 6,000,000 runs of the path triggered with v_9 (dashed green line). A sample of $R_{pub} = 1,000$ runs does not capture the ‘knee’ (abrupt variation) in the ECCDF, so when used with MBPTA it fails to capture the abrupt change. This change is caused by a cache placement with high impact on execution time that occurs with low probability. With $R_{pub+tac} = R_{p+t} = 70,000$ runs the impact of that cache placement is observed and the resulting pWCET upper-bounds it. In general, for all inputs (v_i), see Table 2, TAC requires more runs than PUB to account for conflicting cache placements.

Table 1: BS. Execution Time (in thousands). Table 2: Runs (in thousands) for PUB, PUB+TAC and MBPTA.

	Runs		pWCET@ 10^{-12}	
	R_{pub}	R_{p+t}	PUB	P+T
v1	1	40	3,212	4,125
v3	2	20	3,149	4,432
v5	50	50	6,712	6,712
v7	20	20	4,317	4,317
v9	1	70	2,850	7,571
v11	1	8	3,455	4,003
v13	1	80	3,026	7,377
v15	6	40	2,995	3,694

	Runs		
	R_{orig}	R_{pub}	R_{p+t}
bs	1	1	40
cnt	10	2	70
fir	6	9	600
janne	3	1	200
crc	3	5	10
edn	1	1	70
i.sort	40	40	80
jfdc	2	2	50
m.mult	200	200	200
fdct	8	8	8
ns	3	3	500

4 EVALUATION

We model a pipelined in-order processor with first level instruction (IL1) and data (DL1) caches analogous to that in [24]. L1 caches are 4KB 2-way 32B/line, and implement random placement and replacement policies [15]. The content of cache memories is flushed before each run of a program. We conduct the evaluation using the Mälardalen [10] benchmark suite with default input sets, considering them representative of the worst case for loop bounds. Some of the analyzed benchmarks are single-path, so the execution time

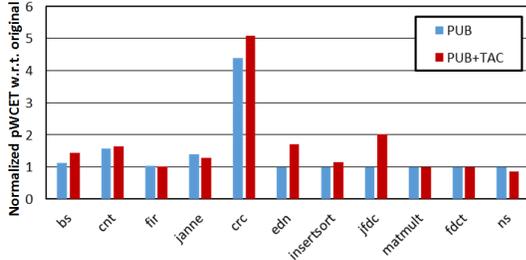


Figure 5: pWCET estimates of PUB and PUB+TAC w.r.t. the original pWCET with user-provided input sets.

variability is due to hardware effects, while for others execution time varies depending on input values. Applying PUB+TAC took approximately 15 minutes per benchmark on average.

4.1 Representative Number of Runs

First, we have collected the number of runs required for MBPTA convergence on the *pubbed* version of the programs, R_{pub} , and the number of runs on those same *pubbed* versions as reported by TAC, $R_{pub+tac}$. Results are shown in Table 2. As shown, despite in some cases the number of runs does not increase, TAC often requires $R_{pub+tac} > R_{pub}$ to attain representativeness. In practice, R_{pub} runs often provide enough measurements from the tail of the execution time distribution, but not with enough confidence. Moreover, in some cases, as shown later, using R_{pub} runs instead of $R_{pub+tac}$ may lead to pWCET estimates that do not account for some events that occur with significant probability. For completeness, we also report the number of runs required on the original version of the programs, applying neither TAC nor PUB, so only determined by MBPTA (R_{orig}). As shown, $R_{pub+tac} > R_{orig}$, although there is no specific relation between $R_{pub+tac}$ (or R_{pub}) and R_{orig} , since they correspond to different programs in practice.

4.2 pWCET Estimates

For some benchmarks, the particular input data available leads to the worst-case path (or they are simply single path). Hence, for those benchmarks we can determine that any difference between the pWCET obtained with and without PUB is fully accountable to overestimation introduced to attain full path coverage. On the remaining benchmarks, instead, it is not possible to clearly tell apart how much of the pWCET increase corresponds to unobserved (worst) paths and how much is plain overestimation.

Some of the benchmarks evaluated are single path (the 6 rightmost ones in Figure 5): *edn*, *insertsort*, *jfdc*, *matmult*, *fdct* and *ns*. Others (the 4 leftmost ones in Figure 5) have multiple paths, but the default input vector used already triggers the worst-case path: *bs*, *cnt*, *fir* and *janne*. Finally, for *crc* we are unable to identify the worst-case path and, based on code inspection, we are highly confident that the default input does not trigger it.

Figure 5 shows the pWCET increase caused by the application of PUB and PUB+TAC w.r.t. the direct application of MBPTA with neither PUB nor TAC. First, we observe that the application of PUB increases pWCET estimates between 4% and 59% for the 4 leftmost benchmarks. In this case, since we know that we already exercised the worst-case path, we can tell this increase is pessimism due to PUB. For the 6 rightmost benchmarks PUB has no impact as they are

single-path and hence, PUB application is innocuous. Finally, in the case of *crc* PUB leads to a 4.4x higher pWCET estimate (340% above that without PUB). While code inspection reveals that this increase captures unobserved paths, it is hard to tell apart how much of this increase is needed to attain path coverage and how much is pessimism incurred by PUB. However, this is the general situation we can expect in real applications: the worst-case path cannot be determined a priori and PUB accounts for it automatically.

When applying TAC on top of PUB, we observe a variety of different behaviors: (1) in most cases (e.g. *bs* and *fir*) pWCET variation is relatively small. This occurs because R_{pub} runs already included execution time measurements for all relevant cache placements. Hence, variations are mostly caused by random variations in the execution time sample used. Note that this may lead to either higher or lower pWCET estimates. (2) In some cases, TAC accounts for cache placements likely not observed otherwise, such as for *edn* and *jfdc*. (3) Finally, in the case of *ns*, the pWCET estimate decreases by 15% with PUB+TAC w.r.t. PUB only. In this case we realize that execution times observed are already close to the maximum execution time possible in practice. Moreover, the application of TAC increases the number of runs from 3,000 to 500,000 to guarantee that high execution times are observed sufficiently. As a consequence, the set of highest execution times obtained with $R_{pub+tac}$ is relatively more homogeneous, since an increasing number of runs makes execution times approach the maximum possible value asymptotically. As a result, MBPTA delivers a tighter bound and hence, the pWCET estimate decreases by 15%.

5 RELATED WORK

Several WCET analysis techniques have been proposed in the last decades [37], each of them with its pros and cons. While static timing analysis is still (and will continue to be) the preferred technique to deal with relatively simple high-criticality systems, MBTA approaches are the most effective means to analyze cutting-edge systems, exploiting high-performance hardware features. Notably MBTA is the dominant practice in the automotive domain, and it has been successfully applied to the highest criticality software, e.g. DAL-A software in avionics [17], to meet functional safety standards requirements [12, 30]. MBPTA is a probabilistic variant of MBTA that has been receiving increasing attention since early 2000's [3, 8]. We stick to the standard notion of WCET (pWCET), identifying worst-case timing bounds for tasks in isolation, disregarding inter-task and inter-core effects: pWCET estimates computed with MBPTA hold valid by construction for multicores if hardware shared resources are MBPTA-compliant [35], or if contention in shared resources is accounted for a posteriori [7] and monitored during operation [25, 26].

The vast majority of works considering MBPTA build on top of EVT. Some of them analyze the implications of upper-bounding execution times with distributions potentially more pessimistic than exponential tails [19], using dependent data [31], and obtaining pWCET estimates in the presence of discrete and/or dependent data [18]. However, only few works analyze the relationship existing between the pWCET estimates obtained during analysis and the timing behavior during operation [2]. Guidelines for a correct application of MBPTA and potential issues are discussed in [13, 23, 32]. Statistical aspects in the application of EVT are explored in [13, 32] where EVT parameters selection and confidence intervals are the

main concerns. A more informed application of EVT to the software domain is advocated instead in [23], restricting EVT to using exponential tail distribution [2] on the account that it has been argued to be the most stable (and always overapproximating) distribution to model worst-case execution of real-time software [2, 27]. A universally recognized issue in the application of EVT is related to identifying an exhaustively representative input sample [13, 32, 38]: our work directly responds to this challenge by synthetically deriving a path-upperbounding version of the target program and defining the minimum number of observations to be collected on such version.

Achieving full path coverage in MBTA has been studied from different angles. Some works attempt to reduce programs to a single execution path building on constant-time predicated instructions [28], but its requirements and cost are difficult to leverage with industrial practice. Other approaches, instead, use genetic algorithms and model checking to generate input vectors to achieve full path coverage [4, 36], but confidence on the results is limited due to their heuristic nature. In the context of MBPTA, only PUB [14] and EPC [38] have been proposed to achieve full path coverage. PUB, upon which this work builds, modifies the source code to produce a program whose timing upper-bounds that of the original program. EPC, instead, operates on fine-grain measurements at basic block level to discount the benefits of specific path traversals and obtain path-independent measurements.

Some MBPTA works derive the minimum number of runs to obtain representative pWCET estimates. This concern was pointed out in [1, 29] and conveniently addressed in [1, 21] with some limitations related to the access sequences. Extensions to arbitrarily complex access sequences have been presented with limiting computational cost first [22], and with affordable costs later, as in the case of TAC [24]. However, to the best of our knowledge, our paper is the first attempt to simultaneously attack the full path coverage and representativeness problems in the context of MBPTA.

6 CONCLUSIONS

In this paper we tackle the issues of achieving full path coverage and representativeness against potential cache layout scenarios that may appear with low-probability. We demonstrate that both objectives can be achieved simultaneously, supporting our proposed method with empirical evidence. We show that the execution time distribution of any path of a *pubbed* program upper-bounds the execution time distributions of all paths in the original program. Then, we also show that by collecting a sufficient number of execution time measurements of any *pubbed* path, the pWCET estimate upper-bounds all paths in the original program under all cache layouts occurring with relevant probabilities. As a result we simultaneously deliver both full path coverage and cache representativeness.

ACKNOWLEDGMENTS

This work has been partially funded by the Spanish Ministry of Science and Innovation under grant TIN2015-65316-P and the HiPEAC Network of Excellence. The Ministry of Economy and Competitiveness partially supported Suzana Milutinovic under FPI grant (BES-2016-077561), Jaume Abella under Ramon y Cajal postdoctoral fellowship (RYC-2013-14717) and Enrico Mezzetti under Juan de la Cierva-Incorporación postdoctoral fellowship (IJCI-2016-27396).

REFERENCES

- [1] J. Abella et al. 2014. Heart of Gold: Making the Improbable Happen to Extend Coverage in Probabilistic Timing Analysis. In *ECRTS*.
- [2] J. Abella et al. 2017. Measurement-Based Worst-Case Execution Time Estimation Using the Coefficient of Variation. *ACM TODAES*, 22, 4 (June 2017).
- [3] G. Bernat et al. 2002. WCET Analysis of Probabilistic Hard Real-Time System. In *RTSS*.
- [4] S. Bunte et al. 2011. Improving the Confidence in Measurement-Based Timing Analysis. In *ISORC*.
- [5] Cobham Gaisler. 2016. LEON3 Processor (Probabilistic platform). <http://www.gaisler.com/index.php/products/processors/leon3>. (2016).
- [6] S. Coles. 2001. *An Introduction to Statistical Modeling of Extreme Values*. Springer.
- [7] E. Diaz et al. 2017. MC2: Multicore and Cache Analysis via Deterministic and Probabilistic Jitter Bounding. In *Ada Europe*.
- [8] S. Edgar and A. Burns. 2001. Statistical Analysis of WCET for Scheduling. In *RTSS*.
- [9] M. Fernandez et al. 2017. Probabilistic Timing Analysis on Time-Randomized Platforms for the Space Domain. In *DATE*.
- [10] J. Gustafsson et al. 2010. The Mälardalen WCET Benchmarks-Past, Present and Future. In *WCET Workshop*.
- [11] E. Heymann. 2017. *The digital car. More revenue, more competition, more cooperation*. Technical Report. Deutsche Bank Research, Frankfurt am Main Germany.
- [12] International Organization for Standardization. 2009. *ISO/DIS 26262. Road Vehicles – Functional Safety*.
- [13] S. Jimenez et al. 2017. Open Challenges for Probabilistic Measurement-Based Worst-Case Execution Time. *IEEE Embedded Systems Letters* (2017).
- [14] L. Kosmidis et al. 2014. PUB: Path Upper-Bounding for Measurement-Based Probabilistic Timing Analysis. In *ECRTS*.
- [15] L. Kosmidis et al. 2016. Fitting processor architectures for measurement-based probabilistic timing analysis. *Microprocessors and Microsystems* 47 (2016), 287 – 302.
- [16] S. Kotz et al. 2000. *Extreme value distributions: theory and applications*. World Scientific, 185 pages.
- [17] S. Law and I. Bate. 2016. Achieving Appropriate Test Coverage for Reliable Measurement-Based Timing Analysis. In *28th ECRTS*.
- [18] G. Lima and I. Bate. 2017. Valid Application of EVT in Timing Analysis by Randomising Execution Time Measurements. In *RTAS*.
- [19] G. Lima et al. 2016. Extreme Value Theory for Estimating Task Execution Time Bounds: A Careful Look. In *ECRTS*.
- [20] E. Mezzetti and T. Vardanega. 2013. A rapid cache-aware procedure positioning optimization to favor incremental development. In *19th RTAS*.
- [21] E. Mezzetti et al. 2015. Randomized Caches Can Be Pretty Useful to Hard Real-Time Systems. *LITES* 2, 1 (2015).
- [22] S. Milutinovic et al. 2016. Modelling Probabilistic Cache Representativeness in the Presence of Arbitrary Access Patterns. In *ISORC*.
- [23] S. Milutinovic et al. 2017. On uses of Extreme Value Theory fit for industrial-quality WCET analysis. In *SIES*.
- [24] S. Milutinovic et al. 2017. Software Time Reliability in the Presence of Cache Memories. In *Ada-Europe*.
- [25] Jan Nowotzsch and et al. 2014. Monitoring and WCET analysis in COTS multi-core-SoC-based mixed-criticality systems. In *DATE*.
- [26] J. Nowotzsch et al. 2014. Multi-core Interference-Sensitive WCET Analysis Leveraging Runtime Resource Capacity Enforcement. In *ECRTS*.
- [27] K. Palma et al. 2017. On Using GEV or Gumbel Models when Applying EVT for Probabilistic WCET Estimation. In *RTSS*.
- [28] P. Puschner. 2003. The single-path approach towards WCET-analysable software. In *International Conference on Industrial Technology*.
- [29] J. Reineke. 2014. Randomized Caches Considered Harmful in Hard Real-Time Systems. *LITES* 1, 1 (2014).
- [30] RTCA and EUROCAE. 1992. *DO-178B / ED-12B, Software Considerations in Airborne Systems and Equipment Certification*.
- [31] L. Santinelli et al. 2014. On the Sustainability of the Extreme Value Theory for WCET Estimation. In *WCET Workshop*.
- [32] L. Santinelli et al. 2017. Revising Measurement-Based Probabilistic Timing Analysis. In *RTAS*.
- [33] Z. Stephenson et al. 2013. Supporting Industrial Use of Probabilistic Timing Analysis with Explicit Argumentation. In *INDIN*.
- [34] <https://www.arm.com/about/newsroom/arm-expects-vehicle-compute-performance-to-increase-100x-in-next-decade.php>. 2015. *ARM Expects Vehicle Compute Performance to Increase 100x in Next Decade*.
- [35] F. Wartel et al. 2015. Timing Analysis of an Avionics Case Study on Complex Hardware/Software Platforms. In *DATE*.
- [36] I. Wenzel et al. 2005. Automatic timing model generation by CFG partitioning and model checking. In *DATE*.
- [37] R. Wilhelm et al. 2008. The worst-case execution-time problem overview of methods and survey of tools. *ACM TECS* 7 (May 2008), 1–53. Issue 3.
- [38] M. Ziccardi et al. 2015. EPC: Extended Path Coverage for Measurement-Based Probabilistic Timing Analysis. In *RTSS*.