PACO: Fast Average-Performance Estimation for Time-Randomized Caches

Suzana Milutinovic[†,‡], Eduardo Quiñones[‡], Jaume Abella[‡], Francisco J. Cazorla[‡,⋆]

[†]Universitat Politècnica de Catalunya, Spain [‡]Barcelona Supercomputing Center, Spain
[⋆]Spanish National Research Council (IIIA-CSIC), Spain

ABSTRACT

Probabilistic timing analysis is a powerful approach to derive worst-case execution time (WCET) estimates, as needed in safety-critical systems, in the presence of high-performance hardware features (e.g., caches). To that end, the timing behavior of certain hardware resources, such as caches, is randomized. Time-randomized (TR) caches allow deriving hit/miss probabilities for each access and probabilistic WCET estimates for the overall program.

However, the analysis of the average performance of TR caches, which is needed for lowly-critical high-performance tasks in mixed-criticality environments, has been neglected. So far, average performance of a TR cache can only be analyzed through simulation, whose accuracy strongly depends on carrying a large number of simulations. In this paper we address this challenge by proposing PACO, an accurate analytical approach to estimate cache hit/miss probabilities of full applications, parts of them and individual cache accesses at low cost for a wide variety of TR cache hierarchies and setups.

1. INTRODUCTION

In safety-critical real-time embedded systems the worst-case execution time (WCET) of tasks needs to be estimated. As safety-related software becomes more complex to provide higher functionality, hardware platforms also become more complex to provide the levels of performance required. Such complexity challenges obtaining trustworthy and tight WCET estimates [28]. Probabilistic timing analysis (PTA) [6, 12, 7, 8, 26, 9, 4, 27] has emerged as a way to achieve predictability through probabilistic means instead of through deterministic means, as most timing analyses do [2].

Cache memories are one of the resources introducing higher complexity in WCET analysis, which has made them object of intense study [19, 10, 20, 11, 17, 13, 5]. In this line, a study conducted for the European Space Agency [5] shows the difficulties of using caches since small program changes that lead to different memory layouts can trigger pathological cache behavior which were called cache risk patterns. The complexity in the analysis of the cache lies on the fact that caches are stateful resources so whether an access hits or misses depends on: i) the sequence of accesses; and ii) the location of memory objects (which easily changes across different runs), since it determines their cache set placement.

Time-randomized (TR) caches [15, 16] break the dependence between memory allocation and cache placement such that the hit/miss probability of an access is not affected by the particular memory address accessed, as needed by PTA. TR caches have been shown to be competitive in terms of worst-case performance with respect to standard time-deterministic caches deploying modulo placement and least-recently-used replacement [2]. A wide variety of TR cache configurations and hierarchies have been shown analyzable in the context of measurement-based PTA (MBPTA) [8, 26]. Those configurations include multi-level caches; direct-mapped, fully-associative and set-associative caches; shared caches for instructions and data; write-through and copy-back write policies; etc.

Some current real-time mixed-criticality systems comprise both real-time and high-performance applications, which makes average performance also critical. For instance, in the space domain it is well accepted that systems will be dual-criticality [21] with control applications requiring real-time guarantees, hence designed to meet requirements in the worst case; and high-performance payload applications requiring high average performance.

Since caches have very high impact on average performance, a fast evaluation of different cache setups becomes essential in the design of a system. However, exploring the cache design space in a tractable manner with an increasing number of interacting parameters remains an open problem due to the high number of detailed simulations required. The problem exacerbates in the context of TR caches. This occurs because for non-randomized caches one run may suffice to determine their average performance under a given design, but for TR caches several runs are required to obtain a representative execution time distribution for each cache design point. This is particularly critical in early processor design stages to (1) design the cache architecture of a given processor for real-time systems; (2) evaluate how reference applications behave on that cache setup; and (3) tune applications to be run on that architecture.

We respond to this challenge by proposing an approach for fast evaluation of TR caches, which we call **P**robabilistic **A**nalytic **C**ache m**O**delling (PACO). Given a set of reference programs from which we extract a trace of instruction and data memory accesses, PACO derives tight estimates of cache miss probabilities, P_{miss}[1]. We propose tight approximation formulas, which we implement in PACO, to estimate P_{miss} for every access in the program as a way to understand the performance of programs running on top of TR caches, thus removing the need for carrying out a large number of time-consuming simulations. PACO builds upon the formu-

DAC '15, June 07 - 11, 2015, San Francisco, CA, USA

http://dx.doi.org/10.1145/2744769.2744886

[1]The probability of hit (P_{hit}) is given by $P_{hit} = 1 - P_{miss}$.

las used in the context of MBPTA [15, 16], whose purpose was simply illustrating MBPTA compliance of those cache designs, rather than being used for SPTA [9, 4]. PACO extends formulas to a wide variety of cache setups and improves their accuracy. The main contributions of this paper are as follows:
(1) We assess the accuracy of existing formulas [15, 16] approximating P_{miss} for several cache hierarchies and configurations. We do so by comparing the approximated probabilities obtained analytically and the more accurate probabilities obtained with a very large number of simulations.
(2) We identify some sources of inaccuracy for a number of formulas due to their inability to capture dependencies across random events in the context of TR caches.
(3) We deliver some new approximations for some of those formulas proving that higher accuracy can be achieved.

PACO has an overall inaccuracy below 2.6% across all cache configurations. PACO's execution time is comparable to that of running only 10 simulations per cache design-point, much less than needed to obtain stable (representative) average performance estimates for TR caches.

2. MISS PROBABILITY IN TR CACHES

Probabilistic WCET estimates (pWCET) in the context of PTA [6, 12, 7, 8, 26, 9, 4, 27], have an associated exceedance probability. While pWCET estimates can be exceeded, PTA allows deriving them for probabilities that are low-enough for the application domain (e.g., in the region of 10^{-15} per activation) [26].

Under PTA, instructions have a probabilistic timing behavior represented with an Execution Time Profile (ETP) $ETP(\mathcal{I}_i) = <\vec{t_i}, \vec{p_i}>$. $\vec{t_i} = (t_i^1, t_i^2, ..., t_i^{N_i})$ enumerates all possible latencies that the instruction can take. $\vec{p_i} = (p_i^1, p_i^2, ..., p_i^{N_i})$, with $\sum_{j=1}^{N_i} p_i^j = 1$, lists the associated probability of occurrence. For instance, the ETP of a memory operation in a simple single-level cache hierarchy is as follows:
$ETPmemop = < (l_{hit}, l_{miss}), (p_{hit}, p_{miss}) >$, where l_{hit} and l_{miss} are the hit and miss latencies respectively and p_{hit} and p_{miss} their corresponding probabilities.

P_{miss} for TR caches has been studied from different angles in each of the two main PTA strands: static (SPTA) and measurement-based (MBPTA). It is worth noting that MBPTA only needs probabilities to exist (i.e. cache accesses need to have a probabilistic nature), but does not need to determine them. To that end, in the context of MBPTA only *approximation formulas* to P_{miss} have been given. Instead SPTA [7, 9, 4] needs those probabilities to apply *convolution* across ETPs of instructions. For SPTA *upper-bound formulas* to P_{miss} have been derived.

SPTA. SPTA derives ETPs for instructions at a high-abstraction level, e.g. an instruction in the object code. Since SPTA is intended to provide a WCET distribution upper-bounding the actual execution time distribution of the program, it needs P_{miss} used during timing analysis to match or upper-bound the real probability of miss once the system is deployed. Moreover, convolution operator used in SPTA requires independence across ETPs to be applied.

Without loss of generality, we assume that each address corresponds to a different cache line. We use capital letters, e.g. A, to refer to (cache line) addresses. Whenever a subindex is added, e.g A_i, it refers to the i-th access to address A. The superindex is the absolute access count number in the considered sequence. For instance in our *reference sequence*: $(A_{j-1}, B_1^1, B_2^2, C_1^3 ..., F_1^k, A_j)$ we focus on deriving P_{miss} for a given access A_j based on the accesses carried out since the last access to A, A_{j-1}. We generically refer to the i-th access between A_{j-1} and A_j as X^i. For instance, X^3 corresponds to C_1^3, i.e. the first access to address C.

For a fully-associative cache with W ways, for the reference address sequence in which no access X^l, with $1 \leq l \leq k$, accesses cache line A, the following upper-bound can be used [4]:

$$P_{miss_{A_j}}(W) = \begin{cases} 1 - \left(\frac{W-1}{W}\right)^k & \text{if } k < W \\ 1 & \text{otherwise} \end{cases} \quad (1)$$

It has been shown that P_{miss} approximation formulas for MBPTA [29, 15, 16], despite being exact for some specific access sequences and upper-bounds for some others, do not provide the independence across ETPs needed by SPTA. Thus, they may lead to optimistic – so non-trustworthy – WCET estimates in the context of SPTA. As a result, SPTA relies on its own set of upper-bound probabilities that provide independence across ETPs (instructions). For instance, SPTA requires for each cache access an estimate that upper-bounds its miss probability regardless of whether previous accesses hit/miss in cache, as it is the case of Equation 1.

While upper-bound formulas to P_{miss} are interesting from a WCET perspective, they are of little interest from an average case perspective as they can be inordinately pessimistic with respect to the average case.

MBPTA. MBPTA takes as input a collection of end-to-end observed execution times to derive pWCET estimates for a program. MBPTA applies Extreme Value Theory (EVT) [18], a well-known statistical method, to produce an execution time distribution that upper bounds the one of the actual program at deployment. Then, the execution time value of that distribution matching the acceptable exceedance threshold is chosen as the pWCET estimate.

MBPTA *approximation formulas* to P_{miss}, which can approximate P_{miss} *from above* or *from below,* are used as a way to illustrate the probabilistic nature of the events occurring in TR cache organizations. For instance, for our reference sequence the miss probability on a fully-associative cache where no constraint is placed on the miss probability of X^l (where $1 \leq l \leq k$), is approximated as follows:

$$P_{miss_{A_j}}(W) = 1 - \left(\frac{W-1}{W}\right)^{\sum_{l=1}^{k} P_{miss_{X^l}}} \quad (2)$$

However, applications of P_{miss} approximation formulas to measure average performance and, more importantly, their accuracy, have not been studied yet. In this paper we cover this gap by proposing PACO, which relies on those approximations and improves them to estimate P_{miss} for instructions, code sequences and full programs.

3. PACO: APPROXIMATING P_{MISS}

Next we describe the cache organizations considered to evaluate PACO and the formulas used to approximate P_{miss}.

3.1 TR cache setup

In a TR cache two main elements are randomized: the replacement and the placement. We use Evict-on-Miss (EoM) as Random Replacement (RR) policy, under which, on the

event of a miss in a given set, a victim line in that set is randomly selected to be evicted. Random placement (RP) [15] uses a random number (RN), generated either by hardware or software, and the address being accessed as its inputs. A hash function combines both and provides a unique and constant cache set (mapping) for the address along the execution. If the RN changes, the cache set in which the address is mapped changes as well, so cache contents must be flushed for consistency purposes. Changing the RN at program execution boundaries reduces flushing overhead. The RP policy proposed in [15] ensures that, given a memory address and a set of RNs, the probability of mapping such address to any given cache set is the same and independent from other addresses.

We consider first level instruction (IL1) and data (DL1) caches and a unified second level cache (UL2). We analyze copy-back (CB) and write-through (WT) caches, and 3 different configurations for the associativity: (F) fully-associative, (D) direct-mapped and (S) set-associative (4-way in the evaluation section). Thus, there are 6 different cache types: (CBF), (CBD), (CBS), (WTF), (WTD), (WTS). We start our analysis with fully-associative and direct-mapped caches in which only the random-replacement and the random-placement policies are respectively used. Finally, we focus on set-associative caches that deploy both random placement and replacement.

3.2 Copy-back Fully-associative Caches (CBF)

$\mathbf{P_{miss}}$ for DL1 and IL1 are called P^{DL1}_{miss} and P^{IL1}_{miss} respectively. For a copy-back setup P_{miss} is as shown in Equation 2, in which P_{miss} for a given access depends on the number of accesses, and their associated probability, between A_j and the previous access to the same line A_{j-1}.

In the case of the DL1 only memory operations (mop^l), i.e. loads and stores, access the DL1. Hence, for the DL1, X^l in Equation 2 represents all memory operations between A_{j-1} and A_j, and so we have Equation 3 where X^l is replaced by mop^l. W_{DL1} is the number of ways in the DL1.

$$P^{DL1}_{miss_{A_j}}(W_{DL1}) = 1 - \left(\frac{W_{DL1}-1}{W_{DL1}}\right)^{\sum_{l=1}^{k} P^{DL1}_{miss_{mop^l}}} \quad (3)$$

For the IL1 X^l stands for all instructions between A_{j-1} and A_j, i.e. $inst^l$.

$$P^{IL1}_{miss_{A_j}}(W_{IL1}) = 1 - \left(\frac{W_{IL1}-1}{W_{IL1}}\right)^{\sum_{l=1}^{k} P^{IL1}_{miss_{inst^l}}} \quad (4)$$

On every miss of an access X^l between two accesses to the same line, A_{j-1} and A_j, a random eviction is carried out. On every eviction the probability of not evicting A_j is $(W_{DL1}-1)/W_{DL1}$ for DL1 (IL1 is analogous). The exponent in Equation 3 accumulates the miss probability of all accesses between A_{j-1} and A_j. This formula approximates P_{miss} based on the expected number of evictions produced by all accesses occurred since the previous access to A. Note that for the first access A_1 we have that $P_{miss_{A_1}} = 1$.

$\mathbf{P_{miss}}$ for UL2 considers the number of evictions produced between A_{j-1} and A_j, since it determines the number of misses in UL2: NM_{UL2}. A data access misses in the UL2 if it misses in the DL1 first, which occurs with probability $P^{DL1}_{miss_{X^l}}$ and it also misses in the UL2, which occurs with probability $P^{UL2}_{miss_{X^l}}$. The latter probability is computed as shown in Equation 5. NM_{UL2} is also affected by the number of misses in the IL1 that also miss in UL2.

$$NM_{UL2} = \sum_{l=1}^{k} \left[\left(P^{DL1}_{miss_{X^l}} \times P^{UL2}_{miss_{X^l}} \right) + \left(P^{IL1}_{miss_{X^l}} \times P^{UL2}_{miss_{X^l}} \right) \right]$$

Overall the UL2 miss probability for A_j is given by:

$$P^{UL2}_{miss_{A_j}}(W_{UL2}) = 1 - \left(\frac{W_{UL2}-1}{W_{UL2}}\right)^{NM_{UL2}} \quad (5)$$

Note that this formula relies on approximated P_{miss} values in DL1 and IL1, thus accumulating inaccuracies in a multiplicative way. Another (smaller) source of inaccuracy for this approximation is the fact that, in pipelined processors, instruction and data accesses are not aligned because a data access can suffer evictions from younger instruction accesses (so accesses after A_j) that reach UL2 early in the pipeline, and because an instruction access can suffer evictions from older data accesses (so accesses before A_{j-1}) that reach UL2 later in the pipeline. For instance, in the sequence $(B_1A_1B_2B_3A_2B_4)$ a data access of A_2 could suffer an eviction from the instruction access of B_4 and an instruction access of A_1 could be evicted by a data access of B_1.

3.3 Copy-back Direct-Mapped Caches (CBD)

$\mathbf{P_{miss}}$ for DL1 and IL1. While any given cache line A can be evicted by any new line fetched from memory in a fully-associative cache, only lines placed in the same set as A can evict it. Indeed, any such line will evict A in a direct-mapped cache. Random placement leads to a probability of $\frac{1}{S}$ of two cache lines to be placed in the same set given S cache sets. Thus, given the same access sequence as before, $(A_{j-1}, X^1, ..., X^k, A_j)$, where A_{j-1} and A_j correspond to accesses to the same cache line, and no X^l (where $1 \leq l \leq k$) accesses the same cache line as A_j, the probability of A_j to miss in cache is as follows:

$$P^{xL1}_{miss_{A_j}}(S_{xL1}) = 1 - \left(\frac{S_{xL1}-1}{S_{xL1}}\right)^{q} \quad (6)$$

Where xL1 stands for either DL1 or IL1 since the same equation is valid for both caches. q_x stands for the number of unique (i.e. non-repeated) cache lines among all X^l for either DL1 (q_D) or IL1 (q_I). This is so because repeated addresses access always the same set so either all of them cannot evict A or all of them would evict it [15].

$\mathbf{P_{miss}}$ for UL2. Instead, for the UL2, P_{miss} is approximated as follows:

$$P^{UL2}_{miss_{A_j}}(S_{UL2}) = P^{xL1}_{miss_{A_j}}(S_{xL1}) \times \left(1 - \left(\frac{S_{UL2}-1}{S_{UL2}}\right)^{q_I+q_D}\right) \quad (7)$$

where q_I and q_D are the number of unique instruction and data addresses respectively accessed by all instructions in between the one accessing A_{j-1} and the one accessing A_j.

3.4 Copy-back Set-associative Caches (CBS)

$\mathbf{P_{miss}}$ for DL1 and IL1. P_{miss} values in direct-mapped and fully-associative caches are independent given that P_{miss} depends on unique addresses in the former and on previous P_{miss} values in the latter. As a result, probability of both events to occur can be obtained by multiplying their respective probabilities [15].

$$P_{miss_{A_j}}(W,S) = \left(1-\left(\frac{W-1}{W}\right)^{\sum_{l=1}^{k} P_{miss_{X^l}}}\right) \times \left(1-\left(\frac{S-1}{S}\right)^{q}\right) \quad (8)$$

Equation 8 is the product of equations 2 and 6, meaning that an access is a miss in cache if any X^l accessed the same set (second part of the equation) and it randomly evicted A in that set (first part of the equation).

In this equation we identify a source of inaccuracy due to the fact that the first part considers *all* evictions occurred in between A_{j-1} and A_j when, instead, it should only consider those occurring in the same cache set. Therefore, as random placement is intended to distribute randomly and evenly addresses across the different cache sets, we propose dividing by S, the number of sets, the exponent of the first part:

$$P_{miss_{A_j}}(W,S) = \left(1-\left(\frac{W-1}{W}\right)^{\frac{\sum_{l=1}^{k} P_{miss_{X^l}}}{S}}\right) \times \left(1-\left(\frac{S-1}{S}\right)^{q}\right) \quad (9)$$

$\mathbf{P_{miss}}$ UL2. In the case of UL2, P_{miss} for access A_j in our reference sequence is as follows, where $xL1$ represents the cache accessed by A_j, that is, $IL1$ or $DL1$:

$$P^{UL2}_{miss_{A_j}}(W_{UL2},S_{UL2}) = P^{xL1}_{miss_{A_j}}(W_{xL1},S_{xL1}) \times P^{UL2-only}_{miss_{A_j}} \quad (10)$$

$P^{UL2-only}_{miss_{A_j}}$ is the miss probability for A_j as if it accessed UL2 directly (omitting xL1). It would be as equation 9, but using W_{UL2} and S_{UL2} instead of W and S respectively, and q_I and q_D instead of q, as in equation 7.

3.5 Write-through Caches (WTx)

The case of write-through caches is analogous to that of copy-back ones with the following differences:

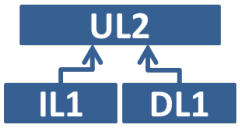
- P^{DL1}_{miss} for DL1 accesses of store instructions is irrelevant from a performance perspective as those accesses are forwarded to UL2 anyway.
- As we assume that UL2 is always copy-back, P^{UL2}_{miss} must consider those accesses caused by IL1 misses, DL1 load misses and *all* DL1 store accesses (regardless of whether they hit or miss).

Other than that, those approximations used for copy-back caches remain valid for write-through ones. We do not provide them explicitly due to space constraints.

3.6 Multiple Addresses per Cache Line

When the addressable unit is smaller than a cache line, accesses to different addresses can be mapped to the same cache line. This has no impact on our previous formulation. For instance, let us assume the sequence $(A_{j-1}, B^1_1, C^2_1, D^3_1, E^4_1 ..., F^k_1, A_j)$, in which B and C go to the same line.

We can simply abstract this sequence as $(A_{j-1}, B^1_1, B^2_2, D^3_1, E^4_1 ..., F^k_1, A_j)$, hence considering that the access to C corresponds to another access to B. This allows us applying the same formulation as above to compute P_{miss}.



	IL1	DL1	UL2
1 level	Read only	Copy Back	-
2 levels	Read only	Write Through	Copy Back

Figure 1: Cache hierarchy and setups considered.

4. EVALUATION

This section evaluates the accuracy of PACO to estimate P_{miss} (and so P_{hit}) probabilities. For that purpose, we compare PACO against simulation where 100,000 simulations are used to obtain figures highly accurate for the leftmost decimal digits of the different probabilities.

We consider two cache setups, *1-level* and *2-level*.
– Under 1-level, only the first level instruction (IL1) and data (DL1) TR caches are used. In this setup DL1 is copy back and the IL1 is read-only.
– 2-level also includes a unified second level (UL2) TR cache which is accessed in case of miss in IL1 or DL1, see Figure 1. This is the most complex hierarchy shown in [16] and conclusions can be extrapolated to larger hierarchies with third or even fourth level caches. In this setup, IL1 is read-only, DL1 is write-through and UL2 is copy-back. DL1 is no-write-allocate, so store misses do not fetch new data to DL1. All store instructions reach UL2 regardless of whether they hit in DL1.

In our reference cache setup, DL1 and IL1 are 8 KB in size and have 32-byte lines. UL2 is 64 KB and has 32-byte lines. We consider direct-mapped, fully-associative and 4-way set-associative caches. Caches are non-inclusive, hence imposing no constraint on whether contents in IL1 or DL1 must or must not be in UL2. Differences in the behavior w.r.t. other inclusion policies have been shown to be rather small [16].

The evaluation has been conducted on the EEMBC Autobench benchmark suite [22], which is a well-known suite reflecting the current real-world demand of some automotive embedded systems. Address traces for PACO and simulation measurements have been collected using the reference input provided together with the benchmark suite. If the analysis needs to be performed for multiple input sets, such analysis can be performed individually for each input set and combined analogously as for the case of running simulations.

Results are reported in terms of the following figures:

- *Per-access evaluation.* For each access in the program we compute the *absolute* difference between the probabilities provided by PACO and those obtained through simulation. For instance if $P^{sim}_{miss} = 10.5\%$ and $P^{PACO}_{miss} = 11.5\%$ the difference is 1%[2]. We then obtain the average and standard deviation of those values across each benchmark for each one of the caches (DL1, IL1 and UL2) in all those 6 setups described in Section 3. Per-benchmark results are averaged thus giving each benchmark the same weight.
- *Per-program evaluation.* Users may be interested in analyzing probabilities at a much coarser granularity than per-access. Therefore, it may be interesting estimating average probabilities for a given program.

[2]Alternatively, this could be expressed in *percentage points* (pp). A *pp* is the unit for the arithmetic difference of two percentages. E.g. going from 1% to 9% is an 8 pp increase. In this paper we decided not to use pp.

Table 1: Per-access P_{miss} accuracy. Avg stands for average and Std for standard deviation.

Cache setup	DL1		IL1		UL2	
	Avg	Std	Avg	Std	Avg	Std
CB-FA	0.02%	0.04%	0.02%	0.04%	N/A	N/A
CB-DM	0.02%	0.03%	0.07%	0.07%	N/A	N/A
CB-SA	1.02%	1.74%	2.59%	3.02%	N/A	N/A
WT-FA	0.01%	0.02%	0.02%	0.04%	0.26%	1.33%
WT-DM	0.01%	0.03%	0.07%	0.07%	0.93%	1.31%
WT-SA	0.54%	1.28%	2.59%	3.02%	2.33%	5.11%

Table 2: Per-program P_{miss} accuracy.

Cache setup	DL1	IL1	UL2
	Average	Average	Average
CB-FA	0.00%	0.00%	N/A
CB-DM	0.00%	0.02%	N/A
CB-SA	0.68%	2.49%	N/A
WT-FA	0.00%	0.00%	0.17%
WT-DM	0.00%	0.02%	0.88%
WT-SA	0.36%	2.49%	2.21%

Thus, we also report per-program results as follows. We compute the actual difference *(not absolute)* between the probabilities provided by PACO and those obtained through simulation. We average those values across each benchmark for each one of the caches in all setups. This provides the actual inaccuracy per program for each cache in each cache organization. Then, we compute the absolute values for each program and report the average difference for each cache in each scenario.

Let us introduce a simple example to illustrate the difference between per-access and per-program results. Let us assume a single cache and 4 accesses whose P_{miss} through simulation is 0.2, 0.1, 0.1, 0.3 respectively and 0.25, 0.1, 0.2, 0.25 through PACO. In this case, per-access average P_{miss} error is 0.05 $\left(\frac{0.05+0+0.1+0.05}{4}\right)$ whereas per-program accuracy error is 0.025 $\left(\frac{0.05+0+0.1-0.05}{4}\right)$. As shown, per-program error can only be lower because errors can cancel out. In the example, underestimation for the 4^{th} access partially offsets the overestimation for the 1^{st} and 3^{rd} accesses.

Finally, we also report results in terms of computational cost. We compare the implementation of PACO w.r.t. the simple cache simulator implemented as baseline. Both of them have been coded from scratch following usual programming guidelines, compiled analogously and no specific code optimization has been applied. Execution times have been obtained on top of a Xeon Dual-Core 5148 operating at 2.33 GHz with 12 GB of DRAM.

4.1 Per-access Results

As shown in Table 1, P_{miss} estimates obtained with PACO are highly accurate for all fully-associative (FA) and direct-mapped (DM) setups with an average difference of 0.03% for DL1 and IL1 caches implementing copy-back (CB) or write-through (WT) policies. Results for DL1 and IL1 still offer good accuracy for set-associative (SA) caches although less than for FA and DM setups, thus indicating that there is potential for improvement of the model. Results for the UL2 cache are less accurate as they accumulate P_{miss} inaccuracies in DL1/IL1 caches determining how many UL2 accesses occur, on top of the UL2 cache inaccuracy itself.

Table 3: Absolute P_{miss} values.

Cache setup	DL1	IL1	UL2
	Average	Average	Average
CB-FA	2.04%	0.95%	N/A
CB-DM	7.85%	1.66%	N/A
CB-SA	2.06%	1.05%	N/A
WT-FA	10.97%	0.95%	13.86%
WT-DM	12.63%	1.66%	12.01%
WT-SA	10.90%	1.05%	12.42%

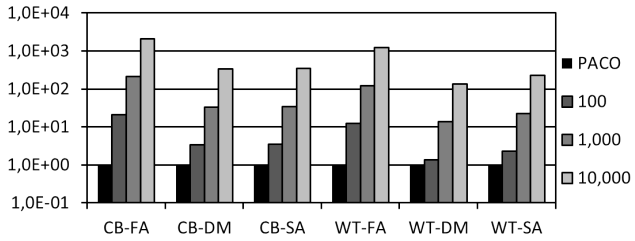


Figure 2: Execution time of simulations normalized w.r.t. PACO.

Note that IL1 results for CB and WT policies are identical as all IL1 accesses are read accesses, and so the write policy has no effect.

4.2 Per-program Results

As shown in Table 2, average P_{miss} estimates obtained with PACO for full programs are more accurate than per-access ones as inaccuracies cancel out to some degree. Again, accuracy for DM and FA caches is much higher than for SA ones. In fact, results for DM and FA DL1 and IL1 caches only show negligible inaccuracy.

For the sake of reference, absolute miss rates for the full programs obtained with the simulator are shown in Table 3, thus illustrating that absolute inaccuracies are relatively low except for some SA caches.

4.3 Execution Time Cost

We have also compared the cost of executing our model w.r.t. the cost of simulating cache behavior, which we regard as the only alternative to obtain results at the same granularity as PACO, so per-access and per-cache memory for any target cache setup. Results are shown in Figure 2. As shown, PACO has a cost similar to that of running 31 simulations on average (between 4 and 74 simulations for different setups), so always lower than that of performing 100 simulations. The relative cost of PACO for DM and SA caches, the ones with higher cost, can be further reduced if multiple cache setups need to be evaluated. This is so because most computation time of PACO is spent computing the unique address reuse distance (q in Equation 6), which needs to be computed only once regardless of the number of cache setups to be evaluated. Conversely, the cost of simulation grows linearly with the number of cache setups. For instance, if we evaluated 100 different cache setups, the average cost of PACO would be as low as that of 4 simulations per setup, thus 25 times lower than using 100 simulations per setup.

In summary, our per-access and per-program results show that PACO inaccuracy is within 0.7% of P_{miss} obtained with 100,000 simulations on average. Thus, PACO provides high

accuracy for P_{miss} with low execution time requirements.

5. RELATED WORK

Cache modeling is a prolific topic with plenty of models devised for deterministic cache policies such as modulo placement and least recently used (LRU) replacement among others [3, 25, 14]. However, to the best of our knowledge, only simulation has been proven feasible to model TR caches used in the context of PTA. PACO is the first attempt towards fast and accurate modeling of TR caches.

PTA technology has evolved during the last decade [6, 12, 7, 8, 26, 9, 4, 27, 28]. Recently, some authors have raised concerns in the use of TR caches in the real-time domain [24]. In particular it is shown that TR caches may produce some risky events not captured by MBPTA. However, it has been shown that those events can be identified for a trustworthy application of MBPTA [18, 1], thus supporting the importance of TR caches and so the need for PACO. In fact, MBPTA has already been evaluated with real avionics case studies [26, 27] and it is in the process of being evaluated on a number of real platforms with a number of case studies [23].

Thus, TR caches will be deployed soon and their average timing behavior needs to be characterized conveniently. PACO covers this gap.

6. CONCLUSIONS AND FUTURE WORK

Time-randomized caches have been deeply studied from a WCET perspective, but there is a lack of efficient ways to estimate the hit/miss probabilities as this can only be done with large number of simulations. In this paper we introduce PACO to efficiently estimate hit and miss probabilities for a wide variety of cache setups and organizations. Our results show that PACO obtains an accuracy within 2.6% across different setups and caches with low computational cost.

As part of our future work we plan to extend our evaluation to a wider variety of cache setups, find more accurate approximations for set-associative caches and extend our model to approximate the probability of evicting dirty lines. We also plan to optimize the implementation of PACO for a further execution time cost reduction.

Acknowledgments

The research leading to these results has received funding from the European Community's FP7 under the PROXIMA Project, grant agreement no 611085. This work has also been partially supported by the Spanish Ministry of Science and Innovation under grant TIN2012-34557, the HiPEAC Network of Excellence, and COST Action IC1202: Timing Analysis On Code-Level (TACLe). Jaume Abella has been partially supported by the Ministry of Economy and Competitiveness under Ramon y Cajal postdoctoral fellowship number RYC-2013-14717.

7. REFERENCES

[1] J. Abella et al. Heart of gold: Making the improbable happen to extend coverage in probabilistic timing analysis. In *ECRTS*, 2014.

[2] J. Abella et al. On the comparison of deterministic and probabilistic WCET estimation techniques. In *ECRTS*, 2014.

[3] A. Agarwal et al. An analytical cache model. *ACM Trans. Comput. Syst.*, 7(2), May 1989.

[4] S. Altmeyer and R. I. Davis. On the correctness, optimality and precision of static probabilistic timing analysis. In *DATE*, 2014.

[5] G. Bernat, A. Colin, and J. Esteves. Considerations on the LEON cache effects on the timing analysis of on-board applications. In *DASIA*, 2007.

[6] G. Bernat, A. Colin, and S.M. Petters. WCET analysis of probabilistic hard real-time systems. In *RTSS*, 2002.

[7] F. J. Cazorla et al. PROARTIS: Probabilistically analyzable real-time systems. *ACM Trans. on Embedded Computing Systems*, 12(2s), 2013.

[8] L. Cucu-Grosjean et al. Measurement-based probabilistic timing analysis for multi-path programs. In *ECRTS*, 2012.

[9] R.I. Davis et al. Analysis of probabilistic cache related pre-emption delays. In *ECRTS*, 2013.

[10] C. Ferdinand and R. Wilhelm. Fast and Efficient Cache Behavior Prediction for Real-Time Systems. *Real-Time System Journal*, XVII:131–181, 1999.

[11] C. Ferdinand et al. Reliable and precise WCET determination for a real-life processor. In *EMSOFT*, 2001.

[12] J. Hansen, S. Hissam, and G. A. Moreno. Statistical-based WCET estimation and validation. In *WCET Workshop*, 2009.

[13] D. Hardy and I. Puaut. WCET analysis of multi-level non-inclusive set-associative instruction caches. In *RTSS*, 2008.

[14] P.J. Joseph et al. A predictive performance model for superscalar processors. In *MICRO*, 2006.

[15] L. Kosmidis et al. A cache design for probabilistically analysable real-time systems. In *DATE*, 2013.

[16] L. Kosmidis et al. Multi-level unified caches for probabilistically time analysable real-time systems. In *RTSS*, 2013.

[17] B. Lesage, D. Hardy, and I. Puaut. WCET analysis of multi-level set-associative data caches. In *WCET Workshop*, 2009.

[18] E. Mezzetti et al. Randomized caches can be pretty useful to hard real-time systems. *LITES*, 2(1), 2015.

[19] F. Mueller. Predicting instruction cache behavior. *Language, Compilers and Tools for Real-Time Systems*, 1994.

[20] F. Mueller. Timing analysis for instruction caches. *Real-Time Systems Journal - Special issue on worst-case execution-time analysis*, 2000.

[21] M. Patte and V. Lefftz. System impact of distributed multi core systems. Technical Report ESTEC Contract 4200023100, European Space Agency, 2011.

[22] J. Poovey. *Characterization of the EEMBC Benchmark Suite*. North Carolina State University, 2007.

[23] PROXIMA. *EU-FP7 Project: www.proxima-project.eu.*

[24] J. Reineke. Randomized caches considered harmful in hard real-time systems. *LITES*, 1(1), 2014.

[25] G.E. Suh et al. Analytical cache models with applications to cache partitioning. In *ICS*, 2001.

[26] F. Wartel et al. Measurement-based probabilistic timing analysis: Lessons from an integrated-modular avionics case study. In *SIES*, 2013.

[27] F. Wartel et al. Timing analysis of an avionics case study on complex hardware/software platforms. In *DATE*, 2015.

[28] R. Wilhelm et al. The worst-case execution time problem: overview of methods and survey of tools. *Trans. on Embedded Comp. Systems*, 7(3):1–53, 2008.

[29] S. Zhou. An efficient simulation algorithm for cache of random replacement policy. In *NPC*, 2010.