On the Comparison of Deterministic and Probabilistic WCET Estimation Techniques

Jaume Abella[†], Damien Hardy[*], Isabelle Puaut[*], Eduardo Quiñones[†], Francisco J. Cazorla[†,‡]
[*]University of Rennes I, IRISA, [†]Barcelona Supercomputing Center, [‡]Spanish National Research Council (IIIA-CSIC)

***Abstract*—Timing validation is a critical step in the design of real-time systems, that requires the estimation of Worst-Case Execution Times (WCET) for tasks. A number of different methods have been proposed, such as Static Deterministic Timing Analysis (SDTA). The advent of Probabilistic Timing Analysis, both Measurement-Based (MBPTA) and Static Probabilistic Timing Analyses (SPTA), offers different design points between the tightness of WCET estimates, hardware that can be analyzed and the information needed from the user to carry out the analysis. The lack of comparison among those techniques makes complex the selection of the most appropriate one for a given system.**

This paper makes a first attempt towards comparing comprehensively SDTA, SPTA and MBPTA, qualitatively and quantitatively, under different cache configurations implementing LRU and random replacement. We identify strengths and limitations of each technique depending on the characteristics of the program under analysis and the hardware platform, thus providing users with guidance on which approach to choose depending on their target application and hardware platform.

I. Introduction

Timing analysis of real-time software comprises deriving an upper-bound of the execution time for each sequential software unit, and finding a feasible schedule of those software units. Several timing analysis techniques have been proposed to obtain Worst-Case Execution Time (WCET) estimates of applications running on a particular hardware [31]. Desired properties of a WCET estimate are its *tightness* (the estimated WCET is close to the actual one) and its *safety* (the estimated WCET is larger than any possible execution time).

There are several ways to classify existing timing analysis techniques. In this paper we classify them into two classes of approaches: *deterministic* timing analysis techniques (DTA), that produce a single WCET estimate, and *probabilistic* timing analysis techniques (PTA) that produce multiple WCET estimates with associated exceedance probabilities. Both approaches have their static (SDTA, SPTA) and measurement-based (MBDTA, MBPTA) variants. A commonality of all timing analysis approaches is that they pursue the same goal, namely providing an upper bound on tasks execution time. With that aim, DTA and PTA make different recommendations on hardware and software designs that help providing tight WCET estimates. A key differentiating element between both approaches resides on the fact that DTA tries to reach time predictability advocating for hardware and software designs that are deterministic in their execution time to produce *provably safe* WCET estimates. Meanwhile, most PTA techniques advocate for hardware and software designs that have a randomized timing behavior, to produce WCET estimates that can be exceeded with a given *probability*.

While there have been proposals for both DTA and PTA techniques, to our knowledge no effort has been done to compare DTA and PTA. Several factors explain this lack of comparison:

- Until very recently, PTA techniques were still on their infancy comparatively to the more mature DTA, that are already used in the development of real products.
- Each class of approach performs better on a different hardware design, and has different requirements on software, which makes the definition of fair comparison conditions hard to achieve. For instance, DTA techniques perform better on time-deterministic caches, whereas PTA is defined for time-randomized ones.

In this paper we cover that gap by making for the first time a comparison between some DTA and PTA techniques in the context of architectures with an instruction cache. More precisely, our contributions are as follows:

- We perform an extensive *qualitative* evaluation of the hardware and software requirements of three timing analysis methods: one DTA, static deterministic timing analysis (SDTA) and two PTA, referred to as Static Probabilistic Timing Analysis (SPTA) and Measurement-Based Probabilistic Timing Analysis (MBPTA). This allows to identify the strengths and limitations of each technique depending on the characteristics of the program under analysis and the hardware platform. This provides guidance to the system designers on which approach to choose depending on their target applications and hardware platforms.
- We define a common environment on which all methods are applicable, and on this (necessarily partial) common applicability denominator, we perform a *quantitative comparison* of the estimated WCETs provided by all studied methods.
- We study and compare the *sensitivity* of the studied methods to cache parameters (associativity degree, cache line size, cache size).

It is worth mentioning that our objective in this paper is not to defend one particular timing analysis method at the expense of others. Rather, we pay special attention to identifying the hardware and software designs for which each method performs best. We also pay special attention to selecting a fair hardware/software framework in which DTA and PTA can be compared. To the best of our knowledge, our work is the first attempt to provide a fair qualitative and quantitative comparison of methods too often considered as antagonists.

The rest of this paper is organized as follows. In Section II

we provide an overview of the WCET estimation techniques that will be compared throughout the paper. Section III compares the methods in qualitative terms. Section IV is devoted to the quantitative comparison of the WCET estimates obtained with each technique. Section V presents some related work. Finally, Section VI draws the main conclusions of this work.

II. Overview of the WCET estimation techniques

This section provides an overview of the timing analysis techniques that will be compared to each other later in the paper. For each class of technique, the selected technique is representative of the current state-of-the-art. Before describing the techniques themselves, assumptions made on the hardware that will be used in the quantitative comparison are given.

A. Hardware assumptions

As a first-step towards a more general timing model for the architecture, it is assumed for the pipeline that each instruction takes a fixed known number of cycles to be executed, avoiding the occurrence of timing anomalies. In our processor design the only source of timing variation comes from the instruction cache, which is accessed by each instruction during the fetch stage. A single-level of instruction cache (L1 cache) is assumed. While this architecture is rather simple, it facilitates the interpretation of the results, and thus, the comparison across the different timing analyses.

Cache contents are organized into fixed-size cache lines. Many different designs can be found in existing processors and in the literature. Among those, the most common ones due to their functional, timing and power characteristics include: (i) direct-mapped caches, (ii) fully-associative caches and (iii) set-associative caches.

Direct-mapped caches (DM for short) are arranged in a way such that each address has a unique possible location in the cache. The placement function (generally modulo function) determines that location.

Fully-associative caches (FA for short) instead do not use any placement function. Any line fetched from memory can be potentially stored in any physical line in the cache. Therefore FA caches require a replacement function to decide which cache line to replace when a new line is fetched and all cache lines are occupied. Amongst the most common replacement functions one can find least-recently used (LRU) and random policies, such as Evict-On-Miss (EoM). When using EoM replacement, on the event of a miss, EoM randomly selects a victim line to be evicted.

Finally, set-associative caches (SA for short) combine both approaches, by implementing a number of sets (as in a DM cache) with a number of cache lines in each set (as in a FA cache). Therefore, SA caches implement both placement and replacement functions.

B. Static Deterministic Timing Analysis

Static Deterministic Timing Analysis (SDTA) techniques construct a cycle-accurate model of the system that is fed with a mathematical representation of the code to derive a safe upper-bound of the WCET [31]. SDTA techniques require detailed knowledge of the underlying hardware. For instance, in the case of cache analysis, the replacement policy (e.g., LRU) and the placement policy (e.g., modulo) used by the hardware need to be known.

SDTA methods are generally divided into two steps, commonly named *low-level* analysis and *high-level* analysis. The *low-level* analysis is used to account for the processor micro-architecture while the *high-level* analysis determines the longest execution path among all possible flows in a program. A number of static analysis methods have been designed in the last two decades at both levels (see [31] for an extensive survey of methods and tools). Concerning the low-level analysis, methods have been designed to estimate WCETs in architectures equipped with caches [28], [23], [13], [15] or pipelines [10], [20].

The SDTA technique used in this paper is a mature technique implemented in several tools [31]. The two main steps for WCET estimation are briefly described hereafter. The reader is referred to [31] and [28] for further details on SDTA.

1) Low-level analysis for caches: The contribution of instruction caches to the WCET is determined by associating a Cache Hit/Miss Classification (CHMC) to every memory reference. The CHMC we use, defined in [28], represents the worst-case behavior of each memory reference with respect to the instruction cache:

- *always-hit* (AH): the reference will always result in a cache hit,
- *first-miss* (FM): the reference could neither be classified as hit nor as miss the first time it occurs but will always result in cache hits afterwards,
- *always-miss* (AM): the reference will always result in a cache miss,
- *not-classified* (NC): the reference could neither be classified as hit nor as miss.

CHMCs are obtained by applying the static analysis technique described in [28], based on abstract interpretation. When using this technique, three analyses that operate on the program control flow graph are defined:

- the *Must* analysis determines if a memory block is *always* present in the cache at a given program point: if so, the reference CHMC is *always-hit*,
- the *Persistence* analysis determines if a memory block will not be evicted after it has been loaded; the CHMC of such references is *first-miss*,
- the *May* analysis determines if a memory block may be in the cache at a given point: if not, the reference CHMC is *always-miss*. Otherwise, if present neither in the Must analysis nor in the Persistence analysis the reference CHMC is *not classified*.

Each of these three analyses computes an *Abstract Cache State* (ACS) at every program point until a fixpoint is reached. The semantics of abstract cache states depends on the considered analysis. For instance, a block in the *Must* ACS at a given point is guaranteed to be in the cache at that point.

This technique has been originally defined for LRU replacement policy [28]. It was extended later to account for other replacement policies, including EoM replacement [26], but for EoM it is known to result in pessimistic WCET estimates.

Regarding the placement policy, SDTA methods defined so far implicitly assume modulo placement. Random placement could also be supported by assuming that accesses are mapped to the same set. This would obviously lead to pessimistic WCET estimates.

2) High-level analysis: The most prevalent high-level analysis technique, named IPET for *Implicit Path Enumeration Technique* is based on an Integer Linear Programming (ILP) formulation of the WCET calculation problem [21]. This formulation reflects the program structure and the possible execution flows using a set of linear constraints. An upper bound of the program's WCET is obtained by considering all the basic blocks of the program and maximizing the following objective function:

$$\sum_{i \in BasicBlocks} T_i * f_i \tag{1}$$

T_i (constant in the ILP problem) is the timing information of basic block i. T_i integrates the effects of caches, and is computed using the CHMC of its references and the cache and memory latencies. f_i (variables in the ILP system, to be instantiated by the ILP solver) correspond to the number of times basic block i is executed. The value of the objective function provided by the ILP solver is a safe upper-bound of all the possible execution times.

C. Probabilistic Timing Analysis

Probabilistic Timing Analysis (PTA) [4], [14], [5], [8], [29], [9], [2] provides WCET estimates with an associated probability of exceedance, called probabilistic WCET estimates (pWCET). Despite the fact that a pWCET estimate can be exceeded with a given probability, thus leading to a timing failure, PTA aims at obtaining pWCET estimates for arbitrarily low probabilities. As a consequence, even if any pWCET may in principle be exceeded, this can only happen with a given probability, which can be determined at a level low enough for the application domain (e.g. in the region of 10^{-15} per activation).

Under PTA, the probabilistic timing behavior of an operation[1] can be represented with an Execution Time Profile (ETP). An ETP is expressed by the pair: <timing vector , probability vector>. The former enumerates all the possible latencies that an operation may incur. The latter, for each latency in the timing vector, lists the associated probability of occurrence. Hence, for an operation $\mathcal{O}_i$ we have $ETP(\mathcal{O}_i) =< \vec{t_i}, \vec{p_i} >$ where $\vec{t_i} = \{t_i^1, t_i^2, ..., t_i^{N_i}, \}$ and $\vec{p_i} = \{p_i^1, p_i^2, ..., p_i^{N_i}, \}$, with $\sum_{j=1}^{N_i} p_i^j = 1$. For instance, in the case of a memory operation in a simple single-level cache hierarchy, its ETP 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} are the probabilities of occurrence of a hit and a miss respectively. It is worth noting that an ETP is the implementation of a random variable, in which the ETP's timing vector represents the values that the random variable can take and the ETP's probability vector the associated probabilities.

PTA can be implemented either in a static (SPTA) or measurement-based (MBPTA) manner.

1) Static Probabilistic Timing Analysis: In SPTA, execution time probability distributions for individual operations are determined statically from a model of the system.

Along a given path, assuming that the probabilities for the execution times of each instruction are independent, SPTA is performed by deploying the convolution ($\otimes$) of the discrete probability distributions (ETPs) that describe the execution time for each instruction along that path. The final outcome is a probability distribution representing the timing behavior of the entire execution path.

More formally, if $\mathcal{X}$ and $\mathcal{Y}$ denote the random variables that describe the execution time of two instructions x and y, the convolution $\mathcal{Z} = \mathcal{X} \otimes \mathcal{Y}$ is defined as follows: $P\{\mathcal{Z} = z\} = \sum_{k=0}^{k=+\infty} P\{\mathcal{X} = k\} P\{\mathcal{Y} = z - k\}$. For instance if an instruction x is known to execute in 1 cycle with a probability of 0.9 and to execute in 10 cycles with a probability of 0.1 and an instruction y has an equal probability of 0.5 to execute in 2 or 10 cycles, we have:

$$\mathcal{Z} = \mathcal{X} \otimes \mathcal{Y} = \begin{pmatrix} 1 & 10 \\ 0.9 & 0.1 \end{pmatrix} \otimes \begin{pmatrix} 2 & 10 \\ 0.5 & 0.5 \end{pmatrix}$$

$$= \begin{pmatrix} 3 & 11 & 12 & 20 \\ 0.45 & 0.45 & 0.05 & 0.05 \end{pmatrix}$$

$$=< \{3, 11, 12, 20\}, \{0.45, 0.45, 0.05, 0.05\} >= ETP_{x \otimes y}$$

For every static instruction, i.e. instruction in the executable of the program, and for each context of execution, SPTA assumes that the ETP of each instruction is not affected by the execution of previous instructions. However, the time-randomized cache designs developed so far for PTA create an intrinsic dependence among the hit probability of an access ($Phit$) and the outcome of previously executed accesses to cache [5], [9]. This dependence is broken by creating a lower bound function to the hit probability of every instruction to make it independent from previous accesses [2].

As defined in [9], assuming a N-way set-associative cache implementing EoM random replacement, the lower bound to the hit probability of an access to address A_j in the sequence $A_i B_1 B_2 ... B_k A_j$, where A_i and A_j access the same cache line, and where all B_i access a cache line different from A_j but mapped to the same set, is given by:

$$\hat{P}_{hit}^{EoM} = \begin{cases} \left(\frac{N-1}{N}\right)^k & \text{if } k < N \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

The lower bound to the hit probability is obtained by considering that all accesses between A_i and A_j result in a cache miss, leading to a probability of eviction of $\frac{1}{N}$ for each access. This is a worst-case assumption since each cache miss can potentially evict the cache line containing A_j, whereas

[1] An operation is any executable component, at the finest granularity, it represents an access to a resource and at the highest, the entire program. In between, we can find instructions (which may access several resources), basic blocks, functions, etc.

[2] Since $P_{miss} = 1 - P_{hit}$, having a lower bound to P_{hit} is equivalent to having an upper bound to P_{miss}

cache hits cannot. However, as noted in [2], if $k \geq N$, the worst case occurs when N accesses to different cache lines out of the k accesses are hits, as this means that those N accesses occupy all cache lines and thus, there is no space left for A_j. In that case, in order to be safe, it is assumed that A_j cannot hit in cache.

2) Measurement-Based Probabilistic Timing Analysis: MBPTA [14], [8], [29] derives pWCET estimates for a program based on a collection of end-to-end observed execution times on a MBPTA-compliant platform [8] for which an execution time profile can be derived for each (dynamic) instruction. MBPTA applies Extreme Value Theory (EVT) [19], a well-known statistical method that, based on the complementary cumulative distribution function (CCDF) of the observed execution times, provides the probability that the execution time of a given run of a program exceeds a threshold pWCET.

The correct utilization of EVT imposes that the execution times observed on the different runs can be described with independent and identically distributed (i.i.d.) random variables [8]. However, it is noted that the fact that the execution times obtained when running a software on top of a processor are i.i.d. does not make such processor MBPTA compliant. This is so because MBPTA has its own requirements in addition to those imposed by EVT [6]. For instance, at analysis time, one could randomly collect N execution times from the execution of a program on a conventional time-deterministic – and so non-MBPTA-compliant – hardware platform. The collected execution times would be i.i.d. due to random picking, enabling the application of EVT. However, the exceedance distribution would be representative only of the execution conditions captured at analysis time, which lack evidence to upper-bound the deployment-time behavior of the program.

It has been shown that the applicability of MBPTA [8], [1] rests on the key assumption that all sources of execution time variation in the system must either be statically (i.e. deterministically) or probabilistically upper-bounded so that latencies experienced at analysis time match or upper-bound those at deploy time[3]. In our target processor, at core level all instructions have a fixed (deterministic) latency and hence, are MBPTA compliant. The only element of the execution of an instruction varying its latency is the access to cache.

In [1] it is shown how, by having ETPs at the level of dynamic instructions[4], MBPTA requirements are fulfilled. In other words, having ETP at the level of dynamic instructions allows 1) the required i.i.d. properties to emerge at the level of end-to-end measurement runs; and 2) ensuring that all sources of execution time variation in the system can be statically (i.e. deterministically) or probabilistically upper-bounded.

At the cache level it means that every dynamic access to the cache must be characterized by an ETP, i.e. a probability of hit and miss. It is worth mentioning that *MBPTA requires no derivation of an upperbound probability of miss for cache accesses*. This is so because MBPTA, unlike SPTA, does not derive pWCET estimates by convolution, but based on observations. MBPTA requires that cache accesses have an associated probability of hit/miss. The formula presented in [16] represents an approximation to the actual probability of hit of an access to a random placement random replacement cache. Its purpose is showing that cache hits/misses have a probabilistic nature, which is enough for the applicability of MBPTA. Following the same philosophy, approximation formulas for the hit/miss probability are shown in [17] for multi-level random caches.

[3]Note that MBPTA as described in [8] relies on the fact that the user provides those paths that upper-bound the paths of interest for the WCET.

[4]A dynamic instruction an individual occurrence of a given instruction in a given program path.

TABLE I
TIMING ANALYSIS TECHNIQUES AND THE PLACEMENT AND REPLACEMENT POLICIES THAT THEY CAN ANALYZE

Policy		SDTA	SPTA	MBPTA
Placement	Replacement			
TD	TD	yes	no	no
TD	TR	pessimistic	yes	yes
TR	TR	very pessimistic	no	yes

III. QUALITATIVE COMPARISON OF METHODS

A fair comparison of the SDTA, SPTA and MBPTA timing analysis techniques is complex to carry out, because each technique is suited to certain different hardware and software characteristics. In this section we pave the way to a fair comparison of these timing analysis techniques.

A. Suitability to cache setups

We start with a qualitative comparison of the different timing analysis techniques under different set-associative cache setups. Table I lists the three timing analysis techniques evaluated in this paper and whether they can tightly analyze different combinations of placement and replacement policies. In the table TD stands for time-deterministic placement (e.g., Modulo) and replacement (e.g., LRU) policies while TR means time-randomized placement (e.g., [16]) and replacement (e.g., EoM). The combination of TR placement and TD replacement is not present in the table since it provides no advantage over the other combinations and to our best knowledge no one has considered it so far.

- *Fully deterministic cache designs*. SDTA can analyze such cache designs tightly, specially for LRU cache replacement [26]. MBPTA and SPTA cannot be applied on such cache designs since they require some degree of randomization.
- *Partially randomized caches*. When introducing randomization in the replacement policy, SDTA is still able to analyze such designs but at the cost of pessimistic WCET estimates since only one cache line per set can be considered during the analysis whatever the associativity of the cache. SPTA can analyze the cache designs that enable deriving a lower bound hit probability for every access to the cache, such as EoM replacement [5], [9], [2]. MBPTA can also analyze those cache designs for which each access has a probability of hit/miss. The use of deterministic placement makes that SPTA pWCET estimates are only valid for the memory mapping assumed in the analysis. That is, SPTA computes the probability of

hit for the accesses going to the same cache set, which is determined by the particular memory addresses in which the program code (and data) is allocated in memory. The same principle applies to MBPTA when deployed on partially randomized caches.

- *Fully randomized caches*. When the cache design is fully randomized (randomized placement and replacement), SDTA can be extended to analyze such designs. However, the resulting WCET estimates will be overly pessimistic since it would capture only the spatial locality. To the best of our knowledge, SPTA methods so far have not been extended to randomized placement [16]. For MBPTA, this kind of design is fully analyzable by construction. Unlike partially randomized caches, fully randomized caches break the dependence between the actual addresses of a given memory object (e.g. data or code) and the cache set in which it is allocated, so pWCET estimates obtained with MBPTA remain valid even if the memory mapping of the program changes at deployment time.

In summary, each technique carries with it its sets of hardware designs that can be analyzed tightly. For instance, to produce accurate WCET estimates, SDTA is intrinsically dependent on the fact that the hardware is time-deterministic. Similarly, PTA techniques demand time-randomized hardware. Therefore, to be as fair as possible, we select as cache design the common applicability denominator of all methods. We use modulo-placement set-associative caches and fully-associative caches and assume that programs are always allocated in the same positions in memory.

B. Suitability to multi-path programs

Multi-path programs may affect each of the analysis in a different way. For SDTA, multi-path code is safely supported, but on the other hand may lead to pessimistic WCET estimates since infeasible paths may be considered during the analysis.

For MBPTA, multi-path code leads to a path coverage issue: MBPTA only provides a safe pWCET for those paths that are exercised during the timing analysis phase. No bound can be provided for paths that are not traversed. Hence, in order to obtain a safe pWCET, MBPTA would need to exercise the worst-case scenario during the measurement step, which is hard to achieve. Recent advances on MBPTA show different ways to overcome this limitation [18].

The most mature SPTA methods, like the one presented in section II-C1, estimate pWCETs on a per-path basis, and thus are perfectly suited to single-path programs. The first method to consider multiple paths was recently sketched in [9].

In our goal of performing a fair comparison of mature enough techniques, only single-path programs are considered in the quantitative comparison in section IV.

C. Sensitivity to lack of address information

Determining the run-time addresses of all memory references statically is complex, because many such addresses are computed at run-time (references to arrays, stack-allocated data, calls to libraries whose code is not available, etc.). Hence the sensitivity of the different timing analysis techniques may be a differentiating element in their performance.

SDTA is highly sensitive to imprecise address information. In the event of an access whose address cannot be determined statically, SDTA to be safe not only generally assumes a miss (except when spatial locality can be captured), but also assumes that any possibly referenced address will be accessed.

SPTA, because it has to compute maximum reuse distances between accesses to the same cache line, is also sensitive to the lack of address information. In case an address is unknown, SPTA has to assume that it can be mapped to any set, which effectively increases the reuse distance of the other accesses.

MBPTA is insensitive by construction to the lack of address information. If TD placement is used, MBPTA needs that the location of data/code of the program in memory observed during analysis time, remains during deployment time.

In the quantitative comparison in section IV, we use programs where all referenced addresses are known statically. The quantitative study of the sensitivity to missing address information is left for future work.

IV. Quantitative comparison of methods

In this section, we give a quantitative comparison of the WCET estimates computed by SPTA, SDTA and MBPTA analysis methods on a set of benchmarks.

A. Cache setup

In order to make a fair comparison of the studied timing analysis techniques, in this section we select realistic cache structures and latencies, through an exploration of the latency/energy trade-offs. We select cache designs with similar and realistic energy budgets.

We assume a processor implemented in 45nm technology, using low-power transistors, operating at 1V voltage and 1.2GHz frequency. Also, we show cache energy per access normalized with respect to the 2-way set-associative (SA) cache. Absolute values in pico seconds (*ps*) and nano Joules (*nJ*) are also shown. In all cases we consider a 1KB cache with 32 bytes line size. We have corroborated that the observed trends hold across different associativities for other line and cache size values within reasonable ranges (16-64 bytes per line, 1-4KB cache size).

To reduce energy overheads of fully-associative caches, *FA* caches may implement a technique called way halting [33]. Way halting trades off power for latency and it is especially suitable for FA caches. In particular, *FA-wa* (fully-associative way-halting) caches split tags into several parts and perform tag comparisons serially in a way that the following parts of the tag are only compared if previous parts matched. The number of tag partitions combinations is, in general, inordinately huge. Thus, we consider partitioning tags into 2 and 3 parts, as they provide very high power reductions at the expense of low cache hit and miss latency increase. In order to compute energy per access we assume that bits in each partition are selected in a way that given N bits only $\frac{1}{2^N}$ of the tags match on average. Assuming 1KB 32B/line caches and 32-bit addresses, minimum energy per access without exceeding a latency of 2 cycles is achieved with 5-7-15 tag partitions for 3 partitions. Note that 5 address bits are not part of the tag as they are used as offset inside the cache lines.

TABLE II
LATENCY & ENERGY PER ACCESS (E/A) FOR DIFFERENT CACHE SETUPS

	Latency		E/A	
	cycles	ps	norm	nJ
SA-2	1	811	1.00	1.63
SA-4	1	800	1.10	1.79
SA-8	1	800	1.27	2.06
FA	1	447	2.18	3.54
FAwa	2	1116	0.47	0.76

Cache latencies and energy per access results are shown in Table II. We considered 2-way, 4-way and 8-way SA caches, as well as two FA caches: *FA* and *FAwa*. Energy per access is shown for read accesses only, although write accesses show similar trends. Results have been obtained with the CACTI tool [24], which is an accurate timing, power and area model for cache memories.

As shown, latency remains roughly constant across all SA configurations. FA latency is lower than SA one as the architecture of these designs allows tag read and comparison to occur simultaneously due to the use of content-addressable memory (CAM) cells instead of conventional static random access memory (SRAM) ones plus comparators. This is particularly true when few tags need to be compared (only 32 in our configuration). Although not shown, delay quickly increases as the number of tags does. For instance, in our example *FA* latency is 55% that of a *SA-2* one, but for a 4KB 16B/line cache it is 80%.

Conversely, energy per access grows moderately as associativity increases for SA configurations, but it is huge for the regular FA cache. FAwa decreases energy cost being its energy per access even lower than that for SA caches[5].

To stay in a given power envelope, unless otherwise stated, two reference cache configurations will be used in the following, both with a 1KB instruction cache and 32B cache lines: SA-4 (associativity of 4, latency of 1), and FAwa (fully-associative, latency of 2).

B. Experimental setup

The benchmarks used during the experiments are described in Table III. The first six ones come from the Mälardalen WCET benchmark suite[6], while the last one was a modification of the *aifirf* EEMBC automotive benchmark[7] to make it single-path.

All experiments were conducted on C codes compiled with gcc 4.1 with no optimization into MIPS R2000/R3000 binary code. The default linker memory layout is used: functions are represented in memory sequentially, and no alignment directive is used. The experimental setup was as follows:

- Numbers for the SDTA method come from the Heptane timing analyzer [31], that implements the static analysis technique described in section II-B.

[5]Analogously as for latency, FAwa caches are much more sensitive to the number of cache lines than SA caches. Thus, in a 1KB 32B/line setup energy for FAwa is 47% that of a SA-2 cache, but in a 4KB 16B/line cache it is 93%.

[6]http://www.mrtc.mdh.se/projects/wcet/benchmarks.html

[7]http://www.eembc.org/

TABLE III
CHARACTERISTICS OF ANALYZED CODES

Name	Description	Binary code size (bytes)
sqrt	Square root function implemented by Taylor series	520
fdct	Fast Discrete Cosine Transform	3368
jfdctint	Discrete-cosine transformation on a 8x8 pixel block	3000
minver	Inversion of floating point matrix	4472
matmult	Matrix multiplication	848
edn	Finite Impulse Response (FIR) filter calculations	4708
aifirf	Single-path modification inverse fast fourier transform	6920

- Numbers from the SPTA analysis method were obtained by applying the calculation formulas given in section II-C1 on traces of addresses obtained using a MIPS instruction-set simulator. To keep analysis time as low as possible, we integrated into the tool the uniform re-sampling method as defined in [22]. The method keeps at most 100,000 timing values in each ETP. For every convolution, it uniformly selects the points to be kept in the new distribution. By construction of the re-sampling method, the new re-sampled distribution is safe.
- Numbers from the MBPTA analysis are obtained from 1000 runs of each benchmark on the MIPS simulator, which showed to be enough for MBPTA. The Wald-Wolfowitz independence test and the Kolmogorov-Smirnov two-sample identical distribution test are performed as described in [8] to ensure that timing measurements from the different experiments are independent and identically distributed (i.i.d). In the results presented hereafter, either the statistical tests passed (with a significance level of $\alpha = 0.05$), or all timing measurements were identical, which fulfills the requirements to apply extreme value theory.

For all methods, a memory latency of 70 cycles is considered.

C. Exceedance functions for reference benchmarks and cache configurations

To give a first idea of the shape of the complementary cumulative distribution function (CCDF) calculated by the probabilistic methods, Figure 1 gives the exceedance distribution function for the WCET obtained with MBPTA and SPTA. The figure also shows the (single) WCET estimate computed by the SDTA method, for LRU and EoM, as well as measured execution times, for which, in the case of EoM, we performed 100,000 runs. The results are given for the two reference cache configurations SA-4 and FAwa. Results are given for benchmark *minver*, which has the most legible curves and is representative of the average behavior of all benchmarks.

We observe that SDTA estimates the behavior of LRU caches very tightly. However, when used to analyze EoM caches, as explained in section III, the analysis becomes very pessimistic, and the degree of pessimism largely increases with the cache associativity. This is due to the fact that SDTA-EoM virtually reduces the cache to a 1-way cache, which is

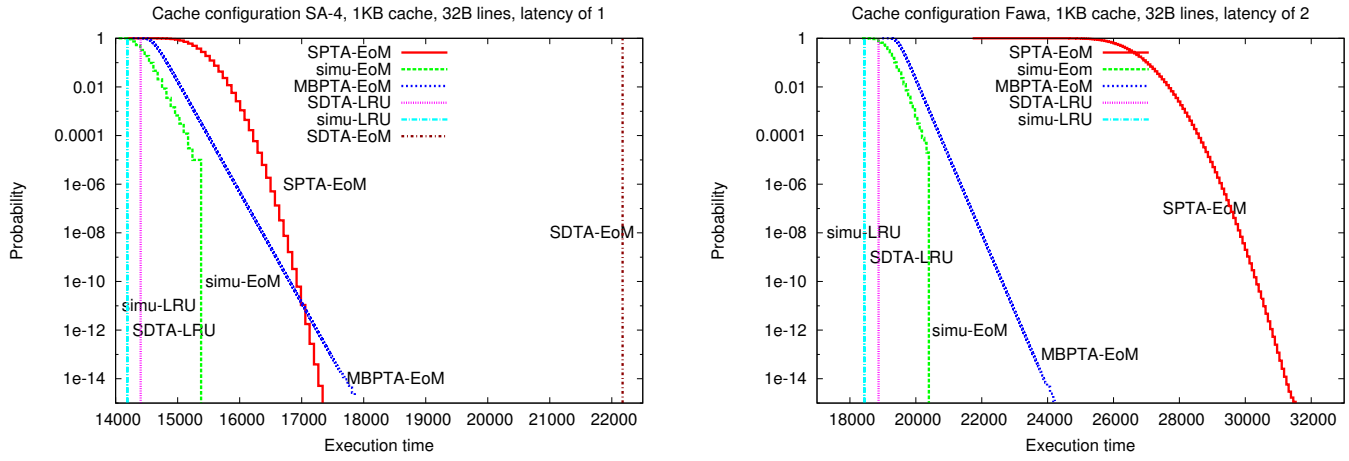


Fig. 1. pWCET distributions for the *minver* benchmark for cache configuration SA-4 (left) and FAwa (right)

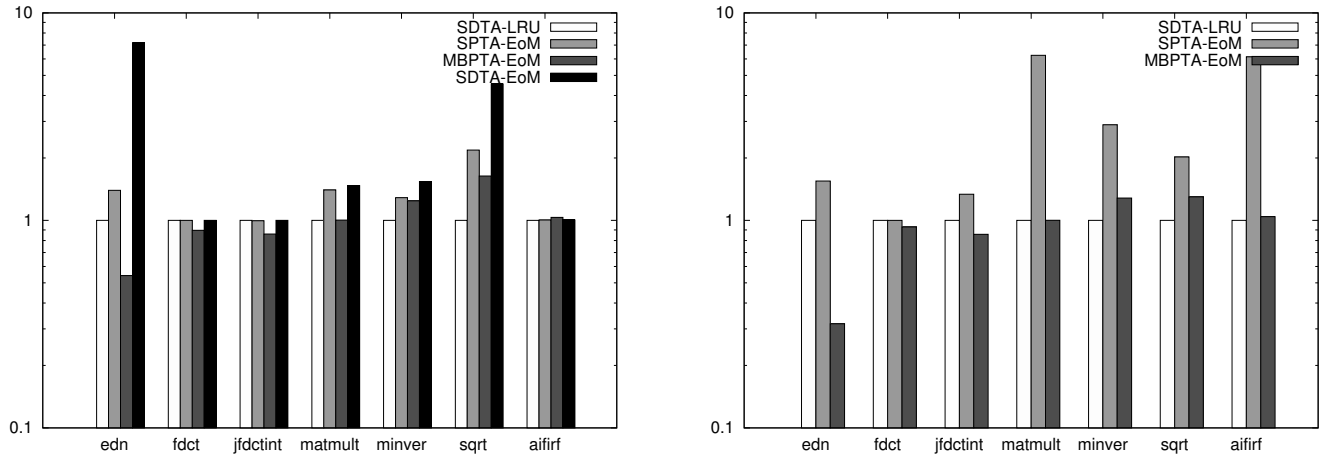


Fig. 2. pWCET distributions for all benchmarks, for a probability of 10^{-15}, for cache configuration SA-4 (left) and FAwa (right)

equivalent to $\frac{1}{4}$ of the total cache size for configuration SA-4 and one cache line for FAwa.

SPTA and MBPTA, in the case of an application whose working set fit in the cache like *minver*, provide WCET estimates between SDTA-LRU and SDTA-EoM. pWCETs for both SPTA and MBPTA are observed to be strictly higher than observed execution times.

D. Comparison of methods for a probability of 10^{-15} for the reference cache configurations

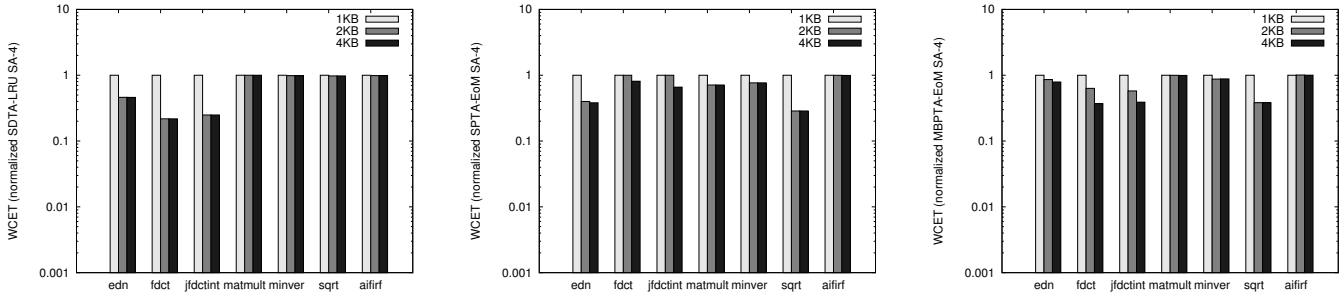
To further compare the different methods, we now fix the exceedance probability to 10^{-15} per run[8]. Figure 2 gives the WCETs for SDTA-LRU, SPTA-EoM, MBPTA-EoM and SDTA-EoM for the two reference cache configurations. Values in the figure are normalized with respect to the WCETs obtained with SDTA-LRU for the studied cache configuration.

1) Results for benchmarks with a working set smaller than cache size: For the benchmarks in this category, *matmult, minver, sqrt* and *aifirf*; their code fits very comfortably in cache. As a result they suffer only cold misses for the LRU cache. Meanwhile, with the EoM cache, there is a non zero probability of replacements to occur.

SPTA-EoM may increase the pWCET noticeably w.r.t. SDTA-LRU, especially for fully-associative caches. The reason for this lies on the fact that modulo placement and LRU provide typically good performance for the common case. Thus, as code cache lines neither exceed cache associativity in the set-associative setup, nor in the fully-associative one, SDTA-LRU provides tight WCET estimates.

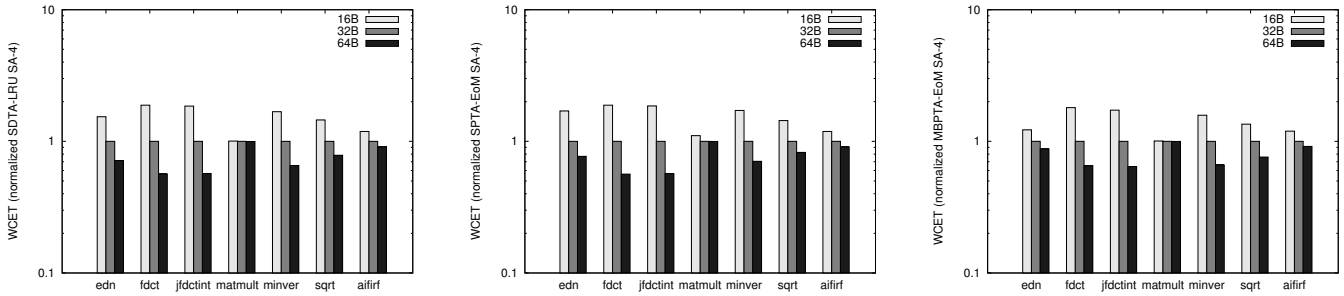
With fully-associative caches, SPTA-EoM accounts for combinations of cache replacements occurring with low – but still meaningful – probability, thus harming pWCET estimates noticeably. With set-associative caches this effect is mitigated by the fact that modulo placement maps cache lines accessed across different sets, thus preventing many replacements from occurring as any access can only potentially evict lines in each own set. Therefore, fewer undesirable replacement can occur, thus making SPTA-EoM more competitive than for fully-associative caches in our experimental setup.

[8]In avionics systems an exceedance probability of 10^{-15} per run has been shown to keep the timing failure rate below 10^{-9} per hour of operation, as needed by the highest integrity levels in DO-178B [27].



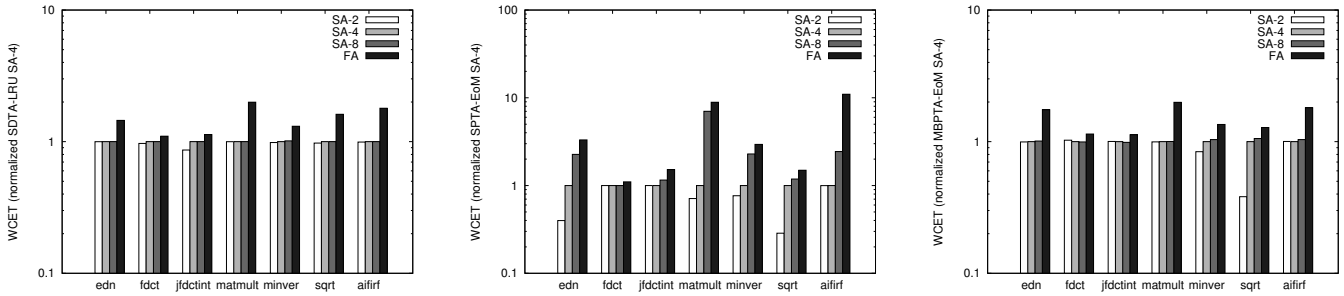
(a) SDTA (b) SPTA (c) MBPTA

Fig. 3. Sensitivity to cache size (reference 1KB cache, 32B lines, SA-4, latency 1)



(a) SDTA (b) SPTA (c) MBPTA

Fig. 4. Sensitivity to line size (reference 1KB cache, 32B lines, SA-4, latency 1)



(a) SDTA (b) SPTA (c) MBPTA

Fig. 5. Sensitivity to associativity (reference 1KB cache, 32B lines, SA-4, latency 1)

pWCET estimates derived with MBPTA-EoM are higher than the ones computed by SDTA-LRU. There are only slightly higher for *matmult* and *aifirf*, and significantly higher for the other benchmarks. In the case of MBPTA-EoM, pWCET estimates are higher because MBPTA accounts for the higher execution time of EoM as compared with LRU, plus some pessimism we attribute to the use of Extreme Value Theory.

2) Results for benchmarks with a working set larger than cache size: In the particular case where the program's working set either does not fit or it is close to use all cache space (benchmarks *edn*, *fdct, and jfdctint*), LRU caches offer poor performance. Thus, SDTA-LRU delivers high WCET estimates regardless of its tightness. For instance, if 5 cache lines placed into the same cache set of a 4-way cache are accessed in a round-robin manner, all of them would miss. Conversely, in a EoM cache some accesses would hit due to the non-null survivability delivered by random replacement.

In such situations, MBPTA-EoM always delivers lower pWCET estimates than SDTA-LRU. The epitome of this scenario is *edn*, whose pWCET estimates for MBPTA-EoM are largely below those for SDTA-LRU.

In comparison, SPTA-EoM for SA-4 caches, produces pWCETs which are close, but always higher to those produced by SDTA-LRU. We attribute this to the pessimism introduced by Equation 2. Improving Equation 2 to be less pessimistic is an area for future work. For FA caches, as explained later, SPTA-EoM pessimism augments largely with the cache associativity degree.

E. Sensitivity to cache parameters

In this section, we study the sensitivity of each method to cache parameters (cache size, line size, associativity). We vary each of these three parameters one at a time, and for each method (e.g., SPTA) compare WCETs with the ones

obtained with the same method on reference configuration SA-4 (1KB cache, 32B cache lines, cache latency of 1 cycle, 4-way set associative cache). For SPTA and MBPTA we set the exceedance threshold to 10^{-15} per run.

a) Impact of cache size: Sensitivity of SDTA, SPTA and MBPTA to cache size is studied in Figure 3. All methods react similarly to an increase of the cache size. When the cache size increases, the WCET (resp. pWCET) decreases, up to an application-dependent point when the code's working set entirely fits in the cache.

b) Impact of cache line size: Sensitivity of SDTA, SPTA and MBPTA to cache line size is studied in Figure 4. We observe that an increase of the line size results in a decrease of the obtained WCET (resp. pWCET). This is because with larger cache lines, the total number of loads from memory is decreased. One may notice that a fixed latency has been assumed for all cache setups regardless of the cache line size. We do not consider this as a bias in the evaluation, because if multiple transfers are needed to fill a cache line, it is always possible to transfer first the part needed despite of needing extra cycles to transfer the rest of the cache line, so memory latency can be regarded as constant across different cache line sizes.

c) Impact of associativity: Sensitivity of SDTA, SPTA and MBPTA to the cache associativity is studied in Figure 5. A first observation is that for FA caches, the obtained WCET (resp. pWCET) is for all methods higher than for a 8-way SA cache. This is because, as discussed in Section IV-A, we selected the FAwa configuration for FA caches, that has a latency of two cycles, compared to the one cycle latency of the other cache configurations.

Another observation is that all techniques but SPTA compute WCETs (resp. pWCETs) that do not depend much on the cache associativity for a given cache size. We attribute the small variations to the different mappings of code to sets, that differ slightly between cache configurations. In contrast, the pWCETs computed by SPTA degrade significantly when the cache associativity increases. This is inherent to equation 2, which overestimates the number of misses to ensure the independence between accesses. The larger the cache associativity (and thus the number of conflicts for a given cache set), the larger the overestimation.

V. Related Work

Several works have been published in the domain of WCET estimation for real-time tasks. They fall in the categories of *probabilistic* (PTA) or *deterministic* (DTA) methods, which can be further divided into *static* (SPTA, SDTA) or *measurement-based* (MBPTA, MBDTA) methods.

Related work on SDTA. Many SDTA techniques have been designed in the last decades (see [31] for a survey). Among them, the mostly used WCET computation technique, called IPET (Implicit Path Enumeration Technique) [21], estimates the WCET through the resolution of an Integer Linear Programming (ILP) problem constraining the execution frequencies of the program's basic blocks.

SDTA needs a low-level analysis phase to determine the worst-case timing behavior of the micro-architectural components: pipelines and out-of-order execution [11], [20], [10], branch predictors [7], [3] and caches. Regarding cache memories, two main classes of approaches have been proposed: *static cache simulation* [23], based on dataflow analysis, and the methods described in [12], [28], based on abstract interpretation. Both classes of methods provide a classification of the outcome of every memory reference in the worst-case execution scenario. These methods were originally designed for code only, and for a Least Recently Used (LRU) replacement policy. They have been later extended to other replacement policies [26], data [25], [13] and unified caches [30], and caches hierarchies [15].

Related work on SPTA. Existing SPTA techniques have been designed for architectures including time-randomized architectural elements, so far mainly caches [4], [5], [9], [2]. There have been two replacement policies deployed in Time Randomized caches. Evict on Access (EoA) and Evict on Miss (EoM). With EoA on every access to a cache set, a line is randomly evicted from that set. With EoM, on the event of a miss in a given set, a victim is randomly selected from that set. Both replacement policies are SPTA compliant. Another commonality among EoM and EoA is that the probability of hit of every access, under both replacement policies, depends on the history of outcomes of previous accesses. SPTA requires lowerbounds for the hit probability, and hence upper-bounding the miss probability of every access. For EoA such lower bound for P_{hit} was done in [5], while for EoM in [9], [2]. In [9] authors focus on the effect of preemptions on time-randomized instruction caches and sketch a first method for analyzing multi-path programs. In [2] authors for the most relevant accesses derive the complete probability tree to determine the actual hit/miss probability of each access to a time randomized cache, while for the rest of the accesses use a less-time-consuming but more pessimistic model based on Equation 2.

Related work on MBPTA. MBPTA [14], [8], [29] works by processing the execution time observations of the program under analysis on an MBPTA-compliant processor. In [1] it is shown the properties required on an MBPTA-compliant processor: at the cache level it is required that every dynamic memory access has an associated probability of miss. Note that while SPTA requires deriving or upper-bounding the probability of miss, MBPTA requires it only to exist. In that line of work, in [16] authors propose a random placement policy that enables the use of MBPTA on set associative caches without requiring the analysis tool to know the address accessed by the program, knowledge that cannot be obtained for industrial-size programs. The hit and miss probabilities shown in [16] for single-level caches and in [17] are meant to show the probabilistic nature of cache accesses and do not bound the hit/miss probability. Hence, those formulas cannot be used in the context of SPTA.

It is worth mentioning that MBPTA bases on a branch of statistics called Extreme Value Theory (EVT) [19]. EVT can be applied to time-deterministic architectures, e.g., processors with caches deploying LRU replacement [32]. However, MBPTA significantly reduces the requirement on the user in terms of the control required on the experimental environment when collecting the end-to-end observations feeding EVT such

that the obtained results are sound [6].

VI. Conclusions

WCET estimation for tasks in safety critical real-time systems is a complex process, especially in the presence of cache memories needed for performance. Different methods have been devised for that purpose including SDTA, SPTA and MBPTA, offering different tradeoffs in terms of hardware required and quality of the WCET estimates.

This paper has performed a comprehensive qualitative comparison showing when each technique is particularly suitable in terms of hardware and software (types of cache memories that can be tightly analyzed, sensitivity to lack of knowledge for referenced addresses, behavior in the presence of multiple-path programs). Methods were also compared quantitatively on a common environment that all methods support. Overall, our conclusions supported by quantitative data help users to better choose the timing analysis method to use for their particular applications and hardware platforms.

SPTA was shown to be more pessimistic than MBPTA. Addressing this issue, as well as extending our approach to multi-path programs, is the subject of our ongoing work.

Acknowledgments

The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under the PROXIMA Project (www.proxima-project.eu), 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). The authors acknowledge Robert I. Davis and Suzana Milutinovic for their contribution to this work.

References

[1] J. Abella et al. Measurement-based probabilistic timing analysis and i.i.d property. White Paper., 2013. http://www.proartis-project.eu/publications/MBPTA-white-paper.

[2] S. Altmeyer and R. I. Davis. On the correctness, optimality and precision of static probabilistic timing analysis. In *Proceedings of Design Automation and Test Europe (DATE)*, 2014.

[3] I. Bate and R. Reutemann. Worst-case execution time analysis for dynamic branch predictors. In *Proceedings of Euromicro Conference on Real-Time Systems (ECRTS)*, 2004.

[4] G. Bernat, A. Colin, and S.M. Petters. WCET analysis of probabilistic hard real-time systems. In *Proceedings of Real-Time Systems Symposium (RTSS)*, 2002.

[5] F. J. Cazorla, E. Quinones, T. Vardanega, L. Cucu-Grosjean, B. Triquet, G. Bernat, E. Berger, J. Abella, F. Wartel, M. Houston, L. Santinelli, L. Kosmidis, C. Lo, and D. Maxim. PROARTIS: Probabilistically analyzable real-time systems. *ACM Transactions on Embedded Computing Systems*, 12(2s), 2013.

[6] F.J. Cazorla, T. Vardanega, E. Quinones, and J. Abella. Upper-bounding program execution time with extreme value theory. In *Proceedings of International Workshop on Worst-Case Execution Time Analysis (WCET)*, 2013.

[7] A. Colin and I. Puaut. Worst case execution time analysis for a processor with branch prediction. *Real-Time Systems Journal*, 18(2-3):249–274, May 2000.

[8] L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzeti, E. Quinones, and F.J. Cazorla. Measurement-based probabilistic timing analysis for multi-path programs. In *Proceedings of Euromicro Conference on Real-Time Systems (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 *Proceedings of Euromicro Conference on Real-Time Systems (ECRTS)*, July 2013.

[10] J. Engblom. *Processor pipelines and static worst-case execution time analysis*. PhD thesis, Uppsala University, 2002.

[11] J. Engblom and A. Ermedahl. Pipeline timing analysis using a trace-driven simulator. In *Proceedings of Conference on Real-Time Computing and Applications Symposium (RTCSA)*, December 1999.

[12] C. Ferdinand. *Cache Behavior Prediction for Real-Time Systems*. PhD thesis, Saarland University, 1997.

[13] C. Ferdinand and R. Wilhelm. On predicting data cache behavior for real-time systems. In *Proceedings of the Workshop on Languages, Compilers, and Tools for Embedded Systems (LCTES)*, 1998.

[14] J. Hansen, S. Hissam, and G. A. Moreno. Statistical-based wcet estimation and validation. In *Proceedings of International Workshop on Worst-Case Execution Time Analysis (WCET)*, 2009.

[15] D. Hardy and I. Puaut. WCET analysis of instruction cache hierarchies. *J. Syst. Archit.*, 57(7):677–694, August 2011.

[16] L. Kosmidis, J. Abella, E. Quinones, and F. J. Cazorla. A cache design for probabilistically analysable real-time systems. In *Proceedings of Design Automation and Test Europe (DATE)*, 2013.

[17] L. Kosmidis, J. Abella, E. Quinones, and F.J. Cazorla. Multi-level unified caches for probabilistically time analysable real-time systems. In *RTSS*, 2013.

[18] L. Kosmidis, J. Abella, F. Wartel, E. Quinones, A. Colin, and F. J. Cazorla. PUB: Path upper-bounding for measurement-based probabilistic timing analysis. In *Proceedings of Euromicro Conference on Real-Time Systems (ECRTS)*, 2014.

[19] S. Kotz and S. Nadarajah. *Extreme value distributions: theory and applications*. World Scientific, 2000.

[20] X. Li, A. Roychoudhury, and T. Mitra. Modeling out-of-order processors for WCET estimation. *Real Time Systems Journal*, 34(3), November 2006.

[21] Y.-T. S. Li and S. Malik. Performance analysis of embedded software using implicit path enumeration. In *Proceedings of the Workshop on Languages, Compilers, and Tools for Embedded Systems (LCTES)*, pages 88–98, New York, NY, USA, November 1995.

[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 *Proceedings of International Conference on Real-Time and Network Systems (RTNS)*, 2012.

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

[24] N. Muralimanohar, R. Balasubramonian, and N.P. Jouppi. CACTI 6.0: A tool to understand large caches. *HP Tech Report HPL-2009-85*, 2009.

[25] H. Ramaprasad and F. Mueller. Bounding worst-case data cache behavior by analytically deriving cache reference patterns. In *Proceedings of Real Time on Embedded Technology and Applications Symposium (RTAS)*, 2005.

[26] J. Reineke, D. Grund, C. Berg, and R. Wilhelm. Timing predictability of cache replacement policies. *Real-Time Systems Journal*, 37:99–122, November 2007.

[27] RTCA and EUROCAE. *DO-178B / ED-12B, Software Considerations in Airborne Systems and Equipment Certification*, 1992.

[28] H. Theiling, C. Ferdinand, and R. Wilhelm. Fast and precise WCET prediction by separated cache and path analyses. *Real-Time Systems Journal*, 18(2-3):157–179, 2000.

[29] 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 *Proceedings of Symposium on Industrial Embedded Systems (SIES)*, 2013.

[30] R. T. White, F. Mueller, C. A. Healy, D. B. Whalley, and M. G. Harmon. Timing analysis for data and wrap-around fill caches. *Real-Time Systems Journal*, 17(2-3):209–233, 1999.

[31] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström. The worst-case execution time problem: overview of methods and survey of tools. *Trans. on Embedded Computing Systems*, 7(3):1–53, 2008.

[32] L. Yue, I. Bate, T. Nolte, and L. Cucu-Grosjean. A new way about using statistical analysis of worst-case execution times. *ACM SIGBED Review*, 8(3):11–14, September 2011.

[33] C. Zhang, F. Vahid, J. Yang, and W. Najjar. A way-halting cache for low-energy high-performance systems. *ACM Transactions on Architecture and Code Optimization (TACO)*, 2(1):34–54, March 2005.