PROARTIS: Probabilistically Analyzable Real-Time Systems

FRANCISCO J. CAZORLA, Barcelona Supercomputing Center and Spanish National Research Council (IIIA-CSIC)
EDUARDO QUIÑONES, Barcelona Supercomputing Center
TULLIO VARDANEGA, University of Padua
LILIANA CUCU, Institut National de Recherche en Informatique et en Automatique (INRIA)
BENOIT TRIQUET, Airbus France
GUILLEM BERNAT, Rapita Systems
EMERY BERGER, Barcelona Supercomputing Center and University of Massachusetts Amherst
JAUME ABELLA, Barcelona Supercomputing Center
FRANCK WARTEL, Airbus France
MICHAEL HOUSTON, Rapita Systems
LUCA SANTINELLI, Institut National de Recherche en Informatique et en Automatique (INRIA)
LEONIDAS KOSMIDIS, Barcelona Supercomputing Center
CODE LO and DORIN MAXIM, Institut National de Recherche en Informatique et en Automatique (INRIA)

Static timing analysis is the state-of-the-art practice of ascertaining the timing behavior of current-generation real-time embedded systems. The adoption of more complex hardware to respond to the increasing demand for computing power in next-generation systems exacerbates some of the limitations of static timing analysis. In particular, the effort of acquiring (1) detailed information on the hardware to develop an accurate model of its execution latency as well as (2) knowledge of the timing behavior of the program in the presence of varying hardware conditions, such as those dependent on the history of previously executed instructions. We call these problems the timing analysis walls.

In this vision-statement article, we present *probabilistic timing analysis*, a novel approach to the analysis of the timing behavior of next-generation real-time embedded systems. We show how probabilistic timing analysis attacks the timing analysis walls; we then illustrate the mathematical foundations on which this method is based and the challenges we face in the effort of efficiently implementing it. We also present experimental evidence that shows how probabilistic timing analysis reduces the extent of knowledge about the execution platform required to produce probabilistically accurate WCET estimations.

Categories and Subject Descriptors: C.3 [**Computer Systems Organization**]: Special-Purpose and Application-Based Systems—*Real-time and embedded systems*; J.7 [**Computer Applications**]: Computers in Other Systems—*Real time*; B.2.2 [**Algorithmic and Logic Structures**]: Performance Analysis and Design Aids—*Worst-case analysis*

General Terms: Design, Performance

This work was primarily funded by PROARTIS STREP-FP7 European Project under the grant agreement number 249100. E. Quiñones is partially funded by the Spanish Ministry of Science and Innovation under the grant Juan de la Cierva JCI2009-05455. J. Abella has been also supported by the Generalitat of Catalunya under grant Beatriu Pinos 2009 BP-B 00260. E. Berger is partially supported by the National Science Foundation under Grant No. CCF-1012195.
Author's address: F. J. Cazorla; email: francisco.cazorla@bsc.edu.

DOI: http://dx.doi.org/10.1145/2465787.2465796

Additional Key Words and Phrases: Embedded and real-time systems, resource sharing, probablistic real-time systems, worst-case execution time

ACM Reference Format:
Cazorla, F. J., Quiñones, E., Vardanega, T., Cucu, L., Triquet, B., Bernat, G., Berger, E., Abella, J., Wartel, F., Houston, M., Santinelli, L., Kosmidis, L., Lo, C., and Maxim, D. 2013. PROARTIS: Probabilistically analyzable real-time systems. *ACM Trans. Embed. Comput. Syst.* 12, 2s, Article 94 (May 2013), 26 pages.
DOI: http://dx.doi.org/10.1145/2465787.2465796

1. INTRODUCTION

1.1. Problem Statement

The market for critical real-time embedded systems (CRTES) which, among others includes the avionics and automotive sectors, is experiencing an unprecedented growth and is expected to continue to grow steadily for the foreseeable future [Clarke 2011]. Let us, for instance, look at the automotive domain: a state-of-the-art high-end car, which currently embeds up to 70 electronic control units (ECUs), is predicted to embed many more [Charette 2009] to account for the inclusion of such new increasingly sophisticated functions as Advanced Driver Assistance Systems (ADAS). For CRTES of this kind, it is imperative to ensure the timing correctness of system operation: some form of *worst-case execution time* (WCET) analysis is needed to that end.

The competition on functional value, measured in terms of application services delivered per unit of product, faces the CRTES industry with rising demands for greater performance, increased computing power, and stricter cost containment. The latter factor puts pressure on the reduction in the number of processing units and ECUs used in the system, to which industry responds by looking at more powerful processors, with aggressive hardware acceleration features like caches and deep memory hierarchies.

In this evolving scenario, it must be acknowledged that the industrial application of current WCET analysis techniques [Wilhelm et al. 2008], which accounts for a significant proportion of total verification and validation time and cost of system production, yields far-from-perfect results. IBM has, for example, found that 50% of the warranty costs in cars are related to electronics and their embedded software, and that 30% of those costs are related to timing flaws. These instances of incorrect operation cost the industry billions of Euros annually [Charette 2009].

Current state-of-the-art timing analysis techniques can be broadly classified in two complementary strands [Wilhelm et al. 2008]: static timing analysis and measurement-based analysis.

Measurement-based analysis techniques rely on extensive testing performed on the real system under analysis using stressful, high-coverage input data, recording the so-called high watermark execution time, that is, the longest observed execution time, and adding to it an engineering margin to make safety allowances for the unknown. However, the safeness of the engineering margin is extremely difficult (if at all possible) to determine, especially when the system may exhibit discontinuous changes in timing due to pathological cache access patterns or other unanticipated timing behavior.

Static timing analysis techniques rely instead on the construction of a cycle-accurate model of the system and a mathematical representation of the application code which makes it possible to determine the timing behavior on that model. The mathematical representation is then processed with linear programming techniques to determine an upper bound on the WCET, providing stronger guarantees than measurement-based approaches. Static approaches have one important limitation though: they are expensive to carry out owing to the need to acquire exhaustive knowledge of all factors, both hardware and software, that determine the execution history of the program under analysis. Some processor architectures may dramatically increase this cost. Others, possibly subject to intellectual property restrictions or incomplete documentation, may

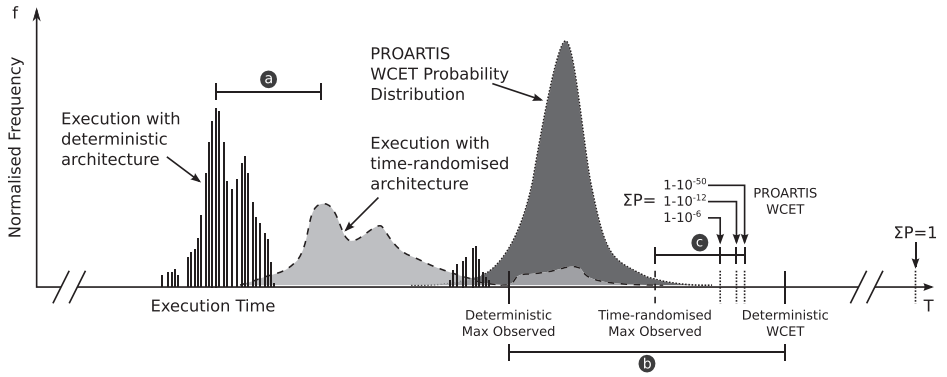


Fig. 1. Execution time distributions for conventional deterministic architectures and a proposed time-randomized architecture, superimposed with the PROARTIS worst-case probability distribution.

even make it altogether impossible, and construction of the timing model must resort to observations.

In order to appreciate the complexity of acquiring complete knowledge of execution history, consider a cache model with a least recently used (LRU) replacement policy. The accuracy in predicting the hit/miss outcome of a memory access depends on knowing the full sequence and addresses of the previous memory accesses made by the program up to the point of interest, in order to build a complete and correct representation of the cache state. Any reduction of the available knowledge, for example, when the addresses of some memory accesses are unknown, leads to a rapid degradation of the tightness of the WCET estimation. In fact, partial knowledge can lead to results as inaccurate as those obtained with no information at all.

1.2. PROARTIS Vision and Approach

Our view is that the cost of acquiring the required knowledge to perform trustworthy analysis can be significantly reduced by adopting a hardware/software architecture whose execution timing behavior eradicates dependence on execution history. One way to achieve this independence is via introducing randomization in the timing behavior of the hardware and possibly of the software (while the functional behavior is left unchanged), coupled with new probabilistic analysis techniques. An example of such hardware is a cache memory in which, in the event of a miss, the victim is randomly selected from any position in the cache. We call this unit of eviction/replacement, *cache entry*. Under this cache configuration, the probability of hit/miss for an access has a small dependence on execution history, in particular, the number of cache misses between the access under consideration and the previous access to the same address. Note that the hit/miss probability is different from the frequency of events. For instance, a memory instruction may have a 50% hit probability if every time it accesses the cache we flip a coin and hit if and only if we get heads. Conversely, if the instruction hits and misses alternately, that instruction does not have a 50% hit probability but a 50% hit frequency. This is because the outcome, and hence the latency, of each access is fully deterministic.

Applying time-randomization techniques has inevitable consequences for the average-case execution time of a program. Figure 1 illustrates the PROARTIS view of execution time; the execution time shift between deterministic and time-randomized architectures is marked (a). In general, we expect the time-randomized profile to shift to the right (to increase) in relation to the deterministic profile, and to spread across a larger range of execution times. However the resulting distribution becomes more

predictable: by decoupling timing events (e.g., cache accesses), they compose into a smooth curve, with a long tail describing execution times which are increasingly unlikely. Dependences between events in deterministic architectures can have an abrupt impact on execution time, producing discontinuities in the possible execution times which are difficult to model with a parametric distribution.

The absolute maximum execution time produced by the PROARTIS analysis will be many times greater than a conventional WCET, as it will correspond with the case where all instructions take the longest possible time to execute (e.g., every cache access is a miss [Quinones et al. 2009]). We expect instead to gain by tightening the gap between observed execution times and worst-case execution bounds using a probabilistic confidence limit. The result of static analysis of deterministic architectures produces a degree of pessimism, where unknown states must be considered to have their worst consequences on the timing of the system. The 'true WCET' lies somewhere in the range marked (b) in Figure 1, between the maximum observed execution time and the WCET bound produced by analysis.

In PROARTIS, the consequences of these unknown states can be considered probabilistically, which allows us to reason about the WCET probabilistically. Techniques from extreme value theory (EVT) are used to construct a worst-case probability distribution. We define worst-case bounds with stated confidence levels, which can be chosen to match the degree of uncertainty present in the rest of the system being analyzed. This will allow a tighter WCET bound to be considered (c).

WCET estimates are computed by considering the execution time at which the cumulative probability (ΣP) exceeds the required level of confidence. These confidence levels are expressed in Figure 1 in terms of the probability of exceeding the WCET threshold in any given run, however this figure should be adjusted based on arrival frequency to determine the probability per hour or per year as necessary.

1.3. Contribution

In this article, we illustrate our vision, which attacks the limits of current timing analysis techniques, using our novel EVT-based probabilistic timing analysis method. We discuss the challenges to be faced at the different layers of the execution stack when adopting our vision, and leave for future work the specification of complete solutions for all of the identified challenges. We argue that our analysis can provide WCET estimations with well-defined confidence bounds for CRTES: this assertion requires that the execution time behavior of the system has specific mathematical properties, in particular *independence and identical distribution*, which we obtain by introducing randomization in the timing behavior at the bottom of the execution stack.

Our main contributions can be summarized as follows.

— We present two approaches to probabilistic timing analysis (PTA). The first statically derives a-priori probabilities from a model of the system: we call it *static PTA* or *SPTA*. The second, measurement-based, derives probabilities from end-to-end runs on the target hardware of the software under study: we call it *MBPTA*. From these runs we collect data about the timing behavior of the application software when executing on our proposed platform with randomized timing behavior.
— For each such approach, we sketch the mathematical foundations on which PTA is based. We further enumerate and begin to attack the challenges we encounter at hardware and software levels upward the execution stack to pursue our vision.
— We then show how SPTA reduces the amount of knowledge needed to achieve tight WCET estimations, and we discuss the benefits that this reduction brings. SPTA requires the reuse distances of every access, which in fact are much easier to obtain than the addresses of memory operations required for conventional static timing

analysis. Tight WCET estimations made with SPTA have less dependence on complete knowledge than conventional methods, and lack of information in the analysis has less impact on the WCET estimations. In fact, WCET estimations provided by SPTA react much more smoothly to the lack of knowledge, with a gradual shift towards a maximum value as knowledge is reduced.

Notably, a probabilistic analysis model is in sync with the aging and reliability behavior of the hardware itself, which common wisdom accepts may fail with a given (though very low) probability. In fact, not only the computing system but any other mechanical parts that form the embedded system also have a distinct probability of failure. In that sense, timing failures can be considered just another type of failure that the system may experience. The objective of the probabilistic timing analysis is to provide WCET estimations that are safe enough for the application, and keep the overall failure rate of the system below the domain-specific threshold of acceptability.

Section 2 provides a background description of how current timing analysis methods work and their limitations. Section 3 explains our probabilistic timing analysis approach and in particular how it addresses the main limitations of current timing analysis techniques. Section 4 provides a comparison of conventional static timing analysis and static probabilistic timing analysis for the cache. Section 5 shows how the probabilistic timing analysis approach fits with future embedded systems design. Section 6 presents related work. Section 7 surveys our main lines of future work. Finally, Section 8 presents the main conclusions of this study.

2. TIMING ANALYSIS ON DETERMINISTIC PLATFORMS

Reliable timing analysis in the form of WCET analysis is a key stage in the design and verification process of real-time systems, and becomes paramount for safety-critical systems. WCET estimates are needed in the development of hard real-time systems to perform schedulability analysis, to ascertain whether the periodic and sporadic tasks meet their timing requirements, to see that the latency in the handling of interrupts falls within limits, and that aperiodic tasks are allowed sufficient time to perform useful work within sufficiently short service windows.

Unfortunately, timing analysis is a complex process, as the variations in execution time experienced by programs may be caused by the characteristics of the software itself, as well as by the hardware platform upon which the program is to run. It therefore follows that all salient characteristics of software and hardware must be thoroughly understood in order to provide meaningful WCET estimations. On the software side, execution time variations may arise from multiple sources, such as varying input sets a program can be asked to operate on, or its layout in memory for both code and data. Similarly, on the hardware side, all features that enhance average performance by exploiting properties of execution history, such as caches, pipelines, and speculation, are potential sources of variation on the execution time.

In the following section we review current state-of-the-art techniques to perform WCET analysis and identify one of the main sources that threatens to make the cost of acquiring the information needed to perform timing analysis unaffordable: the dependence of hardware and software timing behavior on the history of execution.

2.1. A Deterministic Approach to Timing Analysis

Conventional static timing analysis methods require the underlying system to be time deterministic (TD) or time predictable (TP). In a TD system, for a given input set, the sequence of events which will occur in the system is fully known, and so is the time after which the output of an event will be produced. This form of analysis obviously needs a very accurate and detailed model of the system, fully consistent with the causal

dependence in place between all significant events. The accuracy (concerning both overestimation and underestimation) of static timing analysis depend directly on the accuracy of the available system model, which in turn depends on the accuracy of information we can obtain about its actual operation and timing. Conventional analysis can also be applied to TP systems, in which the timing of the events that can occur during execution is not known in advance, but can be bounded. Hence, when the precise timing of an event cannot be determined, pessimistic assumptions can and must be made about its occurrence. This is, for example, the case with cache accesses, which must be considered misses when the memory address issued is not known beforehand, and hence the time of their occurrence cannot be predetermined. Notably, this assumption only holds in the absence of timing anomalies [Lundqvist and Stenstrom 1999; Reineke et al. 2006], whose presence significantly exacerbates the problem.

Unfortunately, with the increase in complexity of next-generation CRTES at both hardware and software level, the extent of knowledge needed, as well as the time, effort, cost, and complexity required to acquire and comprehend all the relevant details, become unaffordable. That is, the large amount of state that is carried by a modern processor leads to combinatorial explosion when trying to enumerate all possible *execution histories*, even for simple pieces of code. Thus, as long as current analysis techniques are unable to scale up to the challenge, increased hardware complexity will cause a significant degradation of the quality of the resulting products.

In summary, industry demands new functionality and higher levels of performance together with reduced cost, weight, and power dissipation, which can only be delivered by advanced hardware features. However, the timing behavior of systems using these advanced hardware features is very hard to deal with by current timing analysis techniques, as the amount of information required is becoming unaffordable.

Next we discuss an example of a hardware component, the cache, whose accurate timing analysis requires an especially large amount of information.

2.2. WCET Dependence on Execution History

Exploiting the execution history of the program, in the form of either temporal or spatial locality, is one of the most common principles of processor design. A typifying example of this strategy is the cache. The use of caches is widespread in general-purpose processors because they can dramatically improve the average application performance by exploiting locality in memory access patterns, reducing access times by up to two orders of magnitude. However, this strategy has an important downside: the execution time of programs and their WCET heavily depend on execution history, which introduces serious complexity. For caches, this dependence shows the level of the cache hierarchy the required data is stored in, and where.

With conventional static analysis approaches, knowledge of the execution history (in addition to knowledge of the target hardware) is required to provide tight WCET estimations. By maintaining the cache state for any point in the program, the analysis can precisely determine whether a memory access will be a hit or a miss. Many research efforts have attempted to obtain predictable cache behavior by applying cache locking (present in PowerPC 440, MPC5554, ARM 940) or deterministic replacement policies, such as least recently used (LRU).

However, to model all possible cache states is extremely costly, as it requires knowledge of all the memory accesses that a program makes. Not having the complete list forces the analysis to make pessimistic assumptions. For example, in a direct-map cache, if a single memory address is not known, the analysis must regard the cache as completely empty, as all cache entries could have potentially been evicted. Memory access times are even harder to predict in the face of complex, multilevel cache hierarchies.

Moreover, deterministic behavior may result in pathological cases that are extremely difficult to predict or detect by testing. A recent study conducted for the European Space Agency shows the difficulties of using caches in CRTES: small program changes that lead to different memory layouts can trigger pathological cache behavior which are called *cache risk patterns*, which are systematic cache misses that lead to large increases in WCET estimation. This may introduce abrupt changes in the WCET when the execution conditions at the time of program deployment are not exactly the same as at the time of analysis.

The difficulty of predicting memory access times in the presence of caches (even though the program may never trigger pathological behaviors) can lead to WCET estimates that are extremely pessimistic, where each unpredictable access is assumed to be a cache miss. This is one of the key sources of pessimism in static WCET analysis.

The level of complexity and the pessimism introduced by cache analysis has led many CRTES to dispense with caches altogether. Hence, despite the fact that the first cache memories appeared in the late sixties, they are still seldomly used in CRTES.

3. PROARTIS: A NEW APPROACH TO TIMING ANALYSIS

The central hypothesis of PROARTIS [2010] is that new advanced hardware features can be used and analyzed effectively in embedded real-time systems with designs that provide time-randomized behavior. The introduction of randomization in the timing behavior of hard-to-analyze resources (e.g., caches) has a twofold objective: (1) it reduces the cost of acquiring the knowledge required to perform trustworthy analysis; (2) it gives the system the properties needed for probabilistic timing analysis by reducing the dependence on execution history. This property, which is discussed in detail in Section 3.1, enables the use of probabilistic analysis techniques in verification arguments for CRTES, proving that the probability of *pathological execution times* is negligible, so that tighter bounds can be obtained. An element of the PROARTIS objectives is to develop a probabilistic timing analysis that will prove that *pathological cases* can only arise with quantifiably negligible probability, rather than trying to eliminate them (which is arguably not possible and could be detrimental to performance). This analysis approach will provide a substantial advance over current methods, including conventional static analysis techniques and testing, which are unlikely to scale up to the size and complexity of next-generation CRTES, the advent of which can hardly be stopped.

The techniques developed in PROARTIS will enable probabilistic guarantees of timing correctness to be derived. For example, if the requirements placed on the reliability of a system indicate that the arrival rate of a timing failure must be less than 10^{-9} per hour of operation, then our analysis techniques translate this reliability requirement into a probabilistic worst-case execution time for the system. Probabilistic analysis provides a continuum of WCETs for different confidence levels. Hence, a system may have a probability of 10^{-9} of exceeding an execution time of 1.5 *ms* over a one-hour period of operation, and arrival rates of 10^{-14}, and 10^{-18} per hour of execution times exceeding 1.6 *ms* and 1.7 *ms*, respectively. The aim of PROARTIS is that for future CRTES (including the hard real-time systems among them), probabilistic guarantees will offer significant advantages over deterministic approaches which attempt to make absolute guarantees, severely limiting the use of advanced hardware features and inevitably offering significantly lower performance guarantees.

Currently, the overall system reliability is expressed in terms of probabilities for hardware failures, memory failures, software faults, and for the system as a whole. PROARTIS extends this notion to timing correctness. We aim to obtain WCET estimations for arbitrarily low probabilities, for example, in the region of 10^{-50} per time unit of operation. To appreciate how small that probability is, consider that extinction-

level events, such as an asteroid hitting the Earth, are estimated to happen about once every 100 million years, hence at an arrival rate of 10^{-16} per second, or 10^{-12} per hour.

3.1. A Probabilistic Approach to Timing Analysis

In general, a probabilistic analysis estimates the chance of future events based on an a-priori model of the probability. For instance, when throwing a die, assuming that the die is not loaded, a probabilistic analysis of the die model would assume that the probability of it falling on any of its faces is 1/6.

Statistical analysis searches for a model or properties when studying some (often large) body of data about observed past events. For instance, on the same preceding example and assuming each face has a unique number, in order for the statistician to check if the die is loaded, he or she would define the hypothesis that 'the probability of each number appearing is the same.' By analyzing a large number of throws, this hypothesis will be accepted or rejected, with a level of confidence based on how closely the observations match the model. In order to avoid confusion, in this article we do not use the word "stochastic" which is often associated with nondeterministic behavior. Although the term is usually described to mean 'based on the theory of probability,' it is often used in other contexts with subtly different meanings. Instead, we refer to probabilistic approaches outlining the nature of our idea which goes beyond the indeterminacy: we consider probability distributions to model the knowledge of processes.

Both probabilistic and statistical analyses associate an event A with a random variable[1] $\mathcal{A}$ describing the probability that such an event occurs. For instance, if A is the event of throwing a die, then $\mathcal{A}$ has the probability function (*PF*) $f_{\mathcal{A}}(\cdot)$ with $f_{\mathcal{A}}(a) = P(\mathcal{A} = a) = \frac{1}{6}$ being the probability to obtain the value $a \in \{1, 2, \cdots, 6\}$ after throwing the die.

The existing probabilistic and statistical analyses for CRTES consider usually either closed form (usually continuous) to describe the distribution of the random variables [Lehoczky 1996], or empirical distributions[2] [Díaz et al. 2002; Navet et al. 2007]. Within PROARTIS, we consider the second option which is more appropriate for measurement-based approaches where the execution traces provide (by grouping the set of values) empirical distributions.

Moreover the existing probabilistic and statistical analyses for CRTES indicate that they require precise hypotheses about the random variables [Griffin and Burns 2010] to be captured in the analysis model. Among many hypotheses, we identify two; *independence* and *identical distribution* of the random variables. We next introduce the definitions of these notions, which we apply in our probabilistic timing analysis.

Definition 1 (*Independent Random Variables*). Two random variables $\mathcal{X}$ and $\mathcal{Y}$ are independent if they describe two events such that the occurrence of one event does not have any impact on the occurrence of the other event.

Definition 1 is equivalent to saying that $\mathcal{X}$ and $\mathcal{Y}$ are independent if and only if

$$P(\mathcal{X} \leq x \text{ and } \mathcal{Y} \leq y) = P(\mathcal{X} \leq x)P(\mathcal{Y} \leq y). \tag{1}$$

[1]In this article we use a calligraphic typeface to denote random variables. Without loss of generality, we consider discrete random variables.

[2]Some authors use the term "density histogram" to design the empirical distributions.

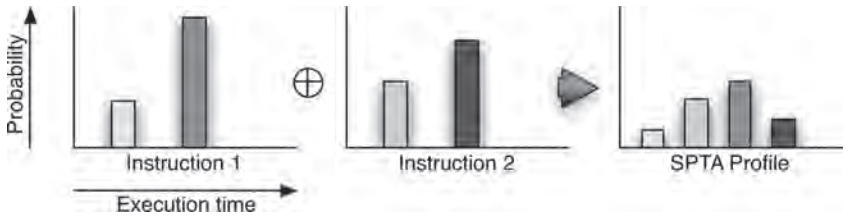


Fig. 2. SPTA uses a convolution to compute a probability distribution for the possible execution times of a sequence of instructions.

Definition 2 (*Identically Distributed Variables*). Two random variables $\mathcal{X}$ and $\mathcal{Y}$ are identically distributed if $\mathcal{X}$ and $\mathcal{Y}$ have the same probability distribution.

Definition 2 is equivalent to saying that two random variables $\mathcal{X}$ and $\mathcal{Y}$ defined on the same space S[3] are identically distributed if for any subset $A \subseteq S$

$$P(\mathcal{X} \in A) = P(\mathcal{Y} \in A). \tag{2}$$

We note that Definition 2 does not imply that two random variables $\mathcal{X}$ and $\mathcal{Y}$, which are identically distributed, are equal.

Definition 3 (*Independent and Identically Distributed Variables*). A sequence of random variables $(\mathcal{X}_n)_n$ is independent and identically distributed (i.i.d.) when any two random variables $\mathcal{X}_i$ and $\mathcal{X}_j$, belonging to the sequence, are independent, $i \neq j$, and have the same distribution function.

A sequence of (fair) die rolls, where each roll is a random variable, is independent and identically distributed as the random variables describe the same event type and the outcomes are obtained from mutually independent events.

Static[4] *Probabilistic Timing Analysis*. In SPTA, execution time probability distributions for individual operations, or components, are determined statically from a model of the processor or software. The design principles of PROARTIS will ensure that it makes sense to derive execution time distributions for these operations, and to make the assumption that the probabilities for the execution time of each instruction are independent. For example, this means that regardless of whether an instruction is actually a cache hit or a miss when executed, the probabilities for later instructions remain the same.

SPTA is performed by calculating the convolution ($\oplus$) of the discrete probability distributions which describe the execution time for each instruction on a CPU; this provides a probability distribution, or execution time profile (ETP), representing the timing behavior of the entire sequence of instructions (see Figure 2).

We have used here a very basic model of the system: the CPU will execute each instruction in a fixed number of cycles, and the only source of timing variation comes from the cache. More complicated modeling is possible, generating a greater number of possible timings for each instruction.

Applying this technique for all instructions in a trace yields a new execution time distribution for the entire sequence of instructions, which includes all possible execution times for that sequence. By choosing a cut-off probability (e.g., 10^{-9}) and measuring where the inverse cumulative distribution function (the exceedance function)

[3]In our case, we suppose S allows the definition of an order relation.

[4]We apply the term *static* to all analyses that derive the a-priori probabilities from an abstraction of the system.

falls below this level, we can read off a WCET with a given confidence that we will not exceed that value.

The convolution of two ETPs requires only one assumption to be made about the input data: that the probability that an instruction causes a cache miss must be independent of the sequence of hits and misses which has occurred due to previous instructions. Note that this does not require that the probability distribution for an instruction is independent of the sequence of preceding instructions, only that the actual outcome of an event (a cache hit or miss) will not affect the calculated probability distribution for following instructions. Dependences between instructions may be modeled by creating profiles for short sequences of instructions which represent all the possible timing interactions between them. We aim to reduce the number of such dependences so that the complexity of combining instruction sequences is reduced.

We now consider a theoretical implementation of a cache that fulfills this requirement, which we refer to as a *time-randomized cache*. It is fully associative and uses an evict-on-access random replacement policy.

The major benefit to be had from this type of cache over conventional ones is that pathological eviction patterns can be avoided by making the probability of any given eviction pattern extremely small. It also reduces the amount of information needed for modeling the cache behavior to only the number of memory accesses, not their locations.

In our time-randomized cache configuration, the probability of evicting a given entry on any single access is $\frac{1}{N}$, where N is the number of cache entries. Zhou [2010] calculates cache-survival probabilities based on the number of cache misses, K, between two consecutive accesses to the same cache entry. With our evict-on-access replacement policy, every access causes an eviction; therefore, we define K to be the number of memory accesses between two consecutive accesses to the same cache entry, including the access for which we are computing K. K is referred to in this article as the *reuse distance* of an address. The value of K must be computed separately for each memory access, as the reuse distance depends on the address.

The formula used in Zhou [2010] does not satisfy our independence requirement: given two consecutive accesses to an address A, the hit probability is lower for the second access if all the intermediate accesses were hits than if they were misses. This is due to conditional probability: if we know that address B was not evicted because we observe a hit, then the chance that we evicted A instead is higher than if we observe a miss. Determining the exact hit probability has exponential cost in the size of K; however, a pessimistic lower bound on hit probability can be computed by applying Equation (3). We assume that there is an upper bound on the amount of information which could be known about the cache contents over the reuse distance. We take the case where all of the addresses accessed are unique and these entries are known to be in the cache for all of the reuse distance, reducing the effective cache size available to A from N to $N-(K-1)$. Therefore, independence is ensured and distributions obtained from this formula may be convolved.

$$P(hit) = \begin{cases} \left(\frac{N-(K-1)-1}{N-(K-1)}\right)^K & \text{if } K < N; \\ 0 & \text{if } K \geq N. \end{cases} \tag{3}$$

Under this cache configuration, the only information that the analysis requires for each memory access is the reuse distance and not the full sequence and addresses of the previous memory accesses, as required when analyzing conventional cache designs such as LRU. Let's assume the sequence of accesses *A B C D A B C A B C* (see Figure 3). We assume that the the cache is initially empty for this example, so the K

Address	Reuse Distance (K)	Hit Prob. (N=32)	Address	Reuse Distance (K)	Hit Prob. (N=32)
(1) A	∞	0.00	(6) B	4	0.87
(2) B	∞	0.00	(7) C	4	0.87
(3) C	∞	0.00	(8) A	3	0.90
(4) D	∞	0.00	(9) B	3	0.90
(5) A	4	0.87	(10) C	3	0.90

Fig. 3. Sequence of memory operations and their reuse distance and hit probability for a 32-entry cache.

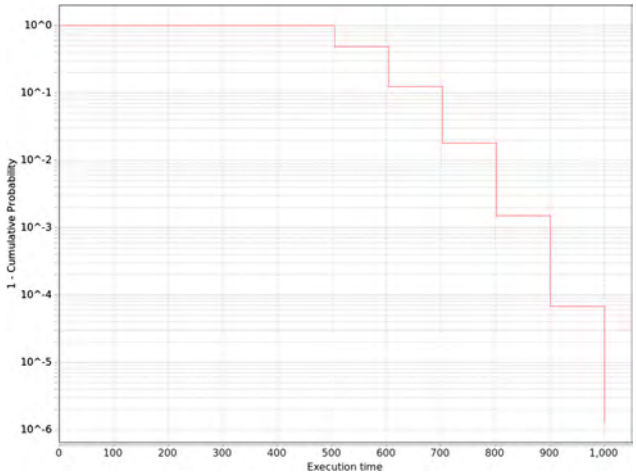


Fig. 4. Example of SPTA exceedance function for a simple ten-instruction trace.

value for the first reference of each address is ∞, and thus its probability of hit is 0 from Equation (3).

To compute the distribution function for the instruction sequence in Figure 3, the hit probability for each access is converted to a two-value distribution, containing the hit and miss probabilities with their associated latencies.

For example, the hit probability for the second access to A in the example (0.87) is translated to a distribution containing an execution time of 1 with probability 0.87, and 100 with probability 0.13. Accesses with zero hit probability create single-valued distributions with an execution time of 100 and probability 1. These distributions are then convolved, and the result can be seen in Figure 4. The chart shows an exceedance function, which is plotted as 1-cumulative probability over the range of execution times. It allows us to read off the probability that the WCET will exceed any given execution time value, or conversely to find a WCET to a given confidence level.

Notice that the maximum execution time is 1,000 cycles: this represents the case where each access is a cache miss.

Measurement-Based[5] *Probabilistic Timing Analysis.* In MBPTA, complete runs of the program under study are made on the target hardware. From these runs, we collect data about the timing behavior of the program when run on low-level software and hardware elements of the PROARTIS platform with randomized timing behavior. This information is used to determine the timing profile (as an execution time frequency distribution) of individual elements of the system, and then these are used as inputs to the timing analysis of the overall system.

[5]We apply the term 'measurement-based' to all analyses that derive a-priori probabilities from a measurement-based model.

For MBPTA, the maximum precision for the probability assigned to a given WCET value is $\frac{1}{number\ of\ runs}$. Hence, one would have to run an impossibly large number of tests to be confident of having observed the longest possible execution time. For instance, an execution time with a 1 in 10^9 probability of occurring will need, on average, one billion test runs to be observed. Theories must be developed so that the required number of experiments (runs) on the target platform is significantly decreased.

Our measurement-based timing analysis method is based on probabilistic techniques and provides an estimation of the WCET of a task or application running as part of a larger system. Using randomization in a system to defeat dependence on execution history is based on the observation that incurring the worst case is a rare event. Therefore any accurate probabilistic timing analysis needs to use the theory of rare events. This theory focuses on the analysis of tails of probability distribution, and it is classically used to study how random processes deviate from their expected value. More precisely, the theory of rare events estimates for an event $\{\mathcal{X} \geq t\}$ the values of t for which the associated probability is very low (e.g., smaller than 10^{-4} per hour).

Two rare events theories fit the WCET estimation problem: the theory of extreme values [Gnedenko 1943] and the theory of large deviations [Lewis and Russel 1997]. Extreme value theory (EVT) provides an estimate for the maximum of a sequence of independent and identically distributed random variables $(\mathcal{X}_i)_n$. Large deviation theory (LDT) instead studies how rare events occur, and it provides an estimate for the sum of a sequence of independent and identically distributed random variables $(\mathcal{X}_i)_n$. LDT relies on the acquired concept that rare events occur in the most likely way.

To the best of our knowledge, EVT is the only rare event theory previously applied to the WCET estimation problem. We provide here an analytical methodology for avoiding the limitations of existing work on EVT (as described in Griffin and Burns [2010]).

Independent and Identically Distributed Variables. Since the analysis models the behavior of one and the same system in the same execution context, the variables are naturally identically distributed. Thus we concentrate our presentation on ensuring that the hypothesis of independence is valid. We identify two means to provide such independence.

— *Statistical methods*. The sequence of dependent variables is transformed into an independent and identically distributed sequence using sampling techniques [Yue et al. 2011]. The obtained independent and identically distributed sequence is not guaranteed to have the same properties as the original sequence.
— *Hardware design*. The architecture guarantees the minimization of dependences among traces. The PROARTIS solution is based on this means.

Continuous Functions. The common utilization of a discrete model within CRTES introduces inaccuracy, including optimistic estimates, if EVT is applied directly on (a set of) discrete values. We identify two methods to circumvent the problem.

— *Block maxima*. The values are grouped and EVT is applied only to the maximum of each group. This method, which Hansen et al. [2009] applied to CRTES, helps EVT provide WCET bounds instead of mere execution time estimation (as was the case of the EVT-based solution proposed in Edgar and Burns [2001]). Coupling the grouping method to a static probabilistic analysis improves the quality of group size.
— *Exceedance threshold*. Only values that are larger than a given threshold are retained. With respect to the block maxima method, this method compares the values to a given value instead of the maximum value of a group. The difficulty with this method stems from the determination of the threshold.

In addition to the limitations described in Griffin and Burns [2010], we identify the overestimation incurred by EVT in case of an insufficient number of runs. Hence the knowledge of a minimum number of runs for a program is required. This number could be obtained by calibrating the results obtained with measurement-based approaches to the results of the static probabilistic timing analysis: when the former match the latter, then the number of measurement runs is sufficient. It follows that measurement-based probabilistic analysis should be built on top of static probabilistic analysis, which thus is the root of the solution. In this article, we therefore focus on static probabilistic timing analysis and leave the formulation of a measurement-based probabilistic analysis approach as future work.

3.2. Challenges at the Platform Level

The angle of attack taken by PROARTIS is to develop an execution platform (hardware and software up to and including the middleware) whose timing behavior has as little dependence as possible on execution history and that enables probabilistic timing analysis, without sacrificing average performance.

3.2.1. Hardware Level. At hardware level, our approach consists in time-randomizing the behavior of hardware components. In adopting this approach, several key questions have to be answered. Do all hardware resources have to be randomized? Can a processor architecture have deterministic and time-randomized resources at the same time? To answer these questions, we classify hardware resources into four types. The main characteristic of resources in terms of timing is whether their latency is fixed or not, whether it impacts WCET noticeably, and whether it can be easily predicted.

— Many resources have a fixed latency. Estimating their impact on timing is trivial in our processor model. For instance, the latency to read a register or to perform an addition is typically fixed, so estimating the latency of an instruction simply requires knowing whether it uses such resources (which is implicit in the instruction itself) or whether all instructions must incur a time penalty as if they were using such resources (which is fixed by the architecture).
— Some specific instructions may have a variable latency when accessing a given resource. However, the difference between the worst-case and the best-case latencies is rather low. In that case (i.e., low variability), the simplest solution would consist of always assuming the worst-case latency to simplify the analysis. The impact in the WCET is negligible. For instance, the latency of a division may vary by a few cycles depending on the value of the input registers to operate and divisions are scarce. Thus, assuming the worst-case latency has very modest impact on the determination of the WCET.
— There are some resources whose latency varies significantly. Hence, simply assuming the worst-case latency is not convenient because the WCET may become extremely pessimistic. However, if accurately predicting the latency of those instructions can be afforded in terms of the effort/cost required for getting the information to have good predictions (i.e., highly predictable events), we can make pessimistic assumptions in those cases where full guarantee on their latency cannot be had. The impact on the WCET determination from that pessimism will be rather low.
— Finally, some resources may have significantly variable latency, and there is no affordable way to predict it (i.e., hard-to-predict events). Those are the main candidates to be redesigned so that their latency can be characterized probabilistically.

PROARTIS focuses on the latter type of resources, whose impact in timing is high and unpredictable or, if predictable, too costly to reason about. The most prominent

examples of this type of resources are cache-like resources, which we analyze in more detail next. The first three types of resources in the preceding taxonomy can be analyzed with static analysis techniques as well as with PROARTIS techniques.

Core Design. No core resources have highly variable latency. Two factors influence possible probabilistically analyzable designs: timing anomalies and performance. We disallow the use of any hardware component that may introduce timing anomalies [Albrecht et al. 2010; Lundqvist and Stenstrom 1999]. From a hardware-only perspective and according to Albrecht et al. [2010], no timing anomalies can occur in processors that do not allow for allocation of resources to instructions in a way that could delay the completion of older instructions. The downside of this argument is that we cannot use some performance acceleration features because they break this restriction. However, as shown in the MERASA project [MerasaD2.2 2008], the memory and not the core is the performance bottleneck in current CRTES. An in-order core processor design with in-order execution of the instructions and allocation of resources at issue time makes the execution time of an instruction not dependent at all on subsequent instructions. Those processors avoid by design the occurrence of timing anomalies [Albrecht et al. 2010]. By the same reasoning, more complex processor pipelines can in principle be used without any problem in our context. Branch prediction, in particular the instructions executed on a misprediction, may affect the state of the processor and hence WCET estimations. As for the case of static analysis, we can simply prevent speculative instructions to change the processor state so that the effect on WCET is bound. Overall, if a pipeline can be analyzed with pipeline analysis techniques for static timing analysis, nothing prevents its use with a probabilistic time analysis tool.

Cache Design. Caches are among the most difficult resources to analyze for timing, as their timing behavior, which is inherently jittery, greatly depends on previous execution history. Two main sources of dependence can be singled out: the placement policy, which selects the set to which a memory address maps, and the replacement policy, which selects the cache line to be evicted from a cache set.

— Fully-associative caches with a random replacement policy effectively reduce such dependence by making each cache line eviction-independent of previous cache accesses. By doing so, we can probabilistically characterize the outcome (hit or miss) of memory accesses, simply based on the reuse distance (see Section 3.1).
— Fully-associative caches are complex to implement and energy-consuming. However, when using a set-associative (with random replacement) or a direct-mapped cache, the placement policy is deterministic. Deterministic placement policies create dependencies and lead to cache risk patterns and hence to time pathological cases. We devise designs based on set-associative or direct-mapped caches using random replacement algorithms, which exhibit a similar behavior at a lower cost. At hardware level, we devise a solution where we can randomize the placement function at program start-up. To that end, we XOR the address accessing the cache with a randomly generated number and use the result as the new address to access the cache. Under every placement setup, the addresses colliding in the different sets are different. Hence, different runs will lead to different (random) colliding addresses in each set and hence different (random) conflicts and execution times. Alternatively, the compiler can provide software-level support to randomize cache placement, as we explain in the next section.

Communication Elements Design. The communication elements between hardware components, such as buses or interconnection networks, can also be a source of dependence, especially those which use access policies based on execution history, such as

```
:LOOP_START     load @1
                load @2
                ...
                load @100
                iter = iter + 1
                compare iter, 100
                jump LOOP_START if smaller
```

Fig. 5. Synthetic benchmark used in this section. The loop performs 100 iterations.

round-robin or least-recently accessed. We devise using random access-granting policies to remove such dependencies (e.g., lottery bus [Lahiri et al. 2001]).

3.2.2. Software Level. Random-replacement caches with deterministic placement policies can be complemented with compiler support, as the timing behavior of the placement policy depends directly on how objects are mapped in memory, that is, on the memory layout. Thus, compiler techniques for randomizing the allocation of objects in memory, such as the code, the stack, or the heap, can be applied [Berger and Zorn 2006]. By doing so, the effective memory addresses of objects, and so, too, the cache set to which they are mapped, will periodically change, thereby randomizing cache placement and consequently enabling probabilistic analysis for cache memories with deterministic placement.

At the runtime level, it is known that not all operating system calls have constant timing behavior; some system calls may in fact exhibit timing behavior that depends on the invocation time of the call in relation to the internal operations of the operating system and on which applications have executed and when and for how long since the previous system call. An operating system with constant-time services also capable of not causing execution history dependence in the subsequent runtime of the application would warrant time composability between the operating system itself and the application. Working towards this goal is also part of our future work.

4. AN ILLUSTRATIVE EXAMPLE

In this section, we discuss results obtained from the application of static probabilistic timing analysis. The development of techniques based on measurement-based probabilistic timing analysis is left for future work.

4.1. Experimental Setup

For the experiments shown in this section, we focused on the data cache and assumed a perfect instruction cache (so that all its accesses were hits). The analysis we conducted can equally be applied to the instruction cache removing the all-hits hypothesis. All core operations have a fixed latency of one cycle. Latencies for the data cache are one cycle in case of hit and 100 cycles in case of miss.

To study the potential of probabilistic timing analysis, we used a synthetic benchmark consisting of a loop with 100 distinct data memory accesses (see Figure 5). Apart from memory operations, we included three core operations with a fixed latency of one cycle for the loop control. The only relevant event in this experiment is the latency of data accesses, as the fixed latency control operations simply add three cycles per iteration.

Results are shown for 100 consecutive iterations after executing the loop at least once. This way we avoid compulsory misses that would simply add 10,000 cycles (100 misses with a 100-cycle latency each) to the WCET estimation regardless of the cache and analysis used.

For the conventional static timing analysis, we consider two-way, four-way, and eight-way LRU replacement caches. For the probabilistic timing analysis, we consider a fully-associative cache in which elements can be allocated randomly in any position. In our particular example, $K = 100$ for all accesses.

While time-deterministic caches (e.g., those based on LRU) produce a single WCET, time-probabilistic caches produce a distribution of execution times. In all experiments, we use the execution time whose exceedance probability is at most 10^{-9} per invocation for the WCET, unless otherwise stated.

We assume that addresses do not conflict in the conventional LRU-based caches. So all accesses are hits in each LRU cache configuration if data fit in the cache.

4.2. Static Probabilistic Timing Analysis Results

With SPTA, the distribution function is computed by convolving the discrete probability distribution calculated for each instruction. These distributions describe the probability that each instruction will take a given execution time. In our example, the distributions for memory operations contain only two values: the execution time when the instruction hits an existing cache entry (one cycle), with probability $p(hit)$, and the execution time when the instruction causes a cache miss (100 cycles), with probability $1 - p(hit)$. The distribution for core operations will contain a single value of one cycle with a probability of 1.

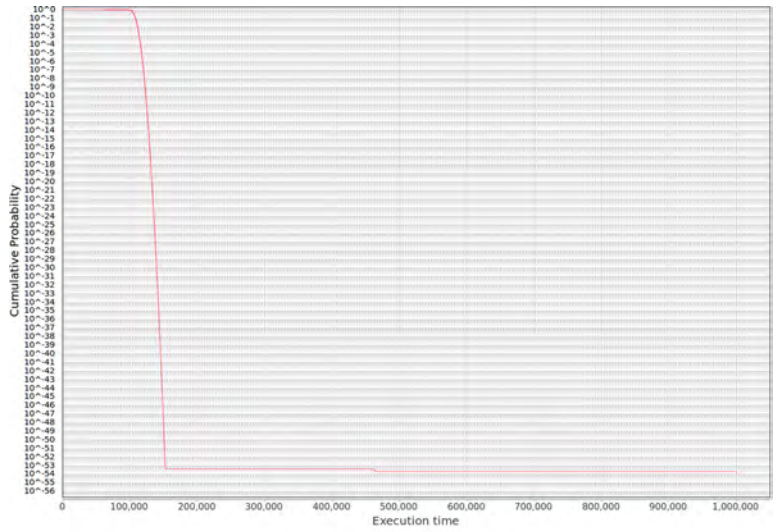
Figure 4 shows an exceedance plot of the distribution function for a very short, ten-access sequence. With longer sequences of instructions, we can obtain WCET estimations for arbitrarily low probabilities (see Figure 6). Probabilities can be in the region of, for example, 10^{-50} or lower.

The horizontal 'shelf' which can be seen in Figure 6 corresponds to the level of precision used in the calculation of the probabilities: for probabilities lower than those representable by the level of precision selected, the value is not representable and results in the shown artefact . The 'real' distribution continues to follow the shape of a normal distribution, which puts the probability of the maximum value (the far right of Figure 6(a), 1,000,000 cycles) at an extremely small probability: 100 iterations of a 100-memory access loop in which every memory access misses in a cache of 1,000 entries (N = 1,000) has a probability of $1 \times 10^{-30,000}$.

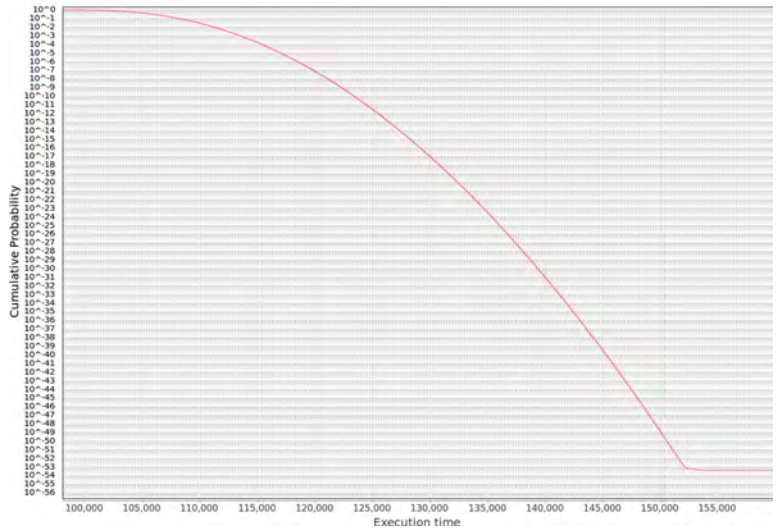
4.3. Comparing Conventional Static Timing Analysis and Static Probabilistic Timing Analysis

We have shown that static probabilistic timing analysis requires less information than conventional static timing analysis. Moreover, the dependence between different parts of an application can be reduced when using probabilistic techniques. In this section, we provide examples of how the probabilistic and conventional analyses react as the amount of information available about execution history decreases.

Making a fair comparison between two timing analysis techniques is difficult. There are many factors that affect the accuracy of the provided WCET estimations in each case. As an illustrative example, in this section we focus on the timing analysis of the cache only. To that end, we propose an evaluation framework in which the cache is the only source of WCET estimation inaccuracy. In order to prevent the processor pipeline affecting timing variability, which can be significant if the processor suffers from timing anomalies or domino effects, we assume a simple in-order pipeline in which each instruction that is not a memory operation takes a fixed latency which is known a-priori. Analogously, in order to discount the effect of other phases of the analysis, such as path analysis or loop-bound analysis, we take programs with only one possible path, that is, the input data is fixed so that all loop bounds are fixed and any branches are chosen deterministically. In order to extend this approach to software with multiple



(a) Overall execution time distribution.



(b) Zoomed region of the function focusing on the area of the large drop in probability.

Fig. 6. WCET exceedance function plotted on a logarithmic scale. Results are for the synthetic benchmark described in this article with N = 1,000 and no unknown addresses.

paths, we intend initially to compute the SPTA distribution for each path under analysis, and then to combine them using the maximum envelope of the inverse cumulative distribution functions. This can be shown to be a pessimistic upper bound on the execution time of the system, regardless of any relative weighting which could be given to different paths in the system.

Addressing the feasibility of analyzing every path in a program, the sequences of instructions which make up a basic block or a single path through a function could be considered a smaller-scale version of our single path problem. By using a tree-based approach to combine the blocks using the envelope method at branch locations and

(a) Initial state. (b) One unknown address processed. (c) Two unknown addresses processed.

Fig. 7. Cache state kept by the conventional static analysis after several unknown addresses are processed.

convolutions for sequential segments, it should be possible to construct an SPTA profile without the need to evaluate all paths in the software separately. This is part of our future work.

— Conventional static timing analysis of the cache distinguishes between certain cache hits/misses and unclassified accesses. An exact static data cache analysis requires knowledge of the target addresses of all memory access operations, which is often not possible to calculate statically. As a reference configuration in our experimental environment, we assume that the cache is the only source of inaccuracy for conventional WCET estimations. We further assume that all addresses are known, so in this configuration we have the tightest WCET estimation possible, which is, in fact, the actual WCET of the program.
— For the static probabilistic timing analysis of the cache, we assume that the reuse distance is known. Note that knowing the reuse distance requires much less information about the memory operations than knowing the exact sequence of memory operations, which is the information required for tight conventional WCET estimations. For example, we may not know until runtime the address of a variable on the stack or the target of a function pointer, but we can compute the reuse distance for accesses to such a variable, and similarly for pointers if the pointer itself is not modified between the accesses.

Reducing the Amount of Required Information. In addition to the idealized case in which all the information required by the analysis is available, we show how different WCET techniques react when part of this information is missing. In particular, for the conventional analysis, we assume that a given percentage of the accesses to memory have an unknown address. For the conventional WCET analysis and in our experimental setup, this lack of information translates into assuming that unknown accesses are misses and that in the cache state kept by the conventional analysis, every unknown address that accesses the cache moves all the data one position towards the LRU position.

For example, Figure 7(a) shows a two-way, eight-set cache with a given initial state. Each cell represents a line in the cache, and in each cell we have a piece of data, represented by a letter. The subscript is the LRU-stack position. In this case, in each set a line can be either the most-recently used line (MRU) or the least-recently used (LRU) line. With exact cache analysis and full information on the accesses, this would be the state of the cache analysis. Figure 7(b) shows the effect on the cache state of one unknown address. Even if the address will affect only one set, the particular set that is affected cannot be determined, meaning that the unknown access can hit any set. Hence, all lines in all sets are moved to the LRU position: the data in the LRU position is assumed to be evicted, and the data in the MRU position in our

Address	Reuse Distance (K)	Hit Prob. (N=32)	Address	Reuse Distance (K)	Hit Prob. (N=32)
(1) A	∞	0.00	(6) B	4	0.87
(2) B	∞	0.00	(7) C	4	0.87
(3) C	∞	0.00	(8) A	7	0.76
(4) D	∞	0.00	(9) B	3	0.90
(5) **?**	∞	0.00	(10) C	3	0.90

Fig. 8. Example of sequence of memory operations and their reuse distance and probability of hit for a cache of 32 entries, when the reuse distance of one access is missing.

two-way cache moves to the LRU position. Figure 7(c) shows the cache state after two unknown addresses are processed. All data have been virtually evicted from the cache because we cannot guarantee any particular piece of data to be in cache after those two accesses with unknown addresses.

In the case of static probabilistic timing analysis, we assume that the reuse distance of some of the addresses is not known. To track reuse distance, we only need to know the number of different accesses between two accesses to the same cache entry. This can be calculated without knowing the actual addresses.

One of the properties of this probabilistic model is that every unknown address only effects previous instances of the same address. Let's illustrate this concept with the same simple example used previously. With our random cache, the probability of evicting each line is $((N-K)/(N-K+1))^K$, where K is the reuse distance. Let us again assume the sequence of accesses *A B C D A B C A B C* discussed in Section 3.1. If the second instance of *A* is unknown, the K value of the third instance of *A* increases from 3 to 7 (see Figure 8). Obviously, the K value for the access with an unknown address is ∞.

We observe how the lack of information for a particular access affects two accesses at most. So the effect of lack of information at analysis time about some addresses has a small effect on WCET.

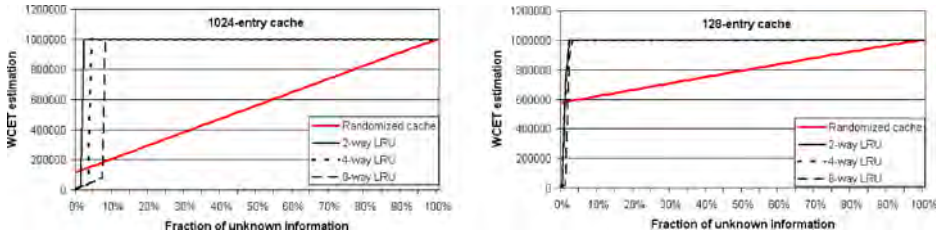
4.4. Results

Effect of Lack of Information on WCET Estimations. The objective of this experiment is to show the dependence of each type of analysis on the amount of available knowledge. To that end, in this experiment we show the WCET bounds obtained for different cache configurations with varying extents of unknown information.

In terms of WCET estimation, those accesses where the address is unknown are assumed to miss in cache and to produce the worst potential impact, as considered by the timing analysis methods. In particular, any element in the LRU position of each set cannot be assumed to be in cache after an unknown access. Conversely, an unknown access does not vary the K for all remaining accesses in the loop for the randomized caches.

Figures 9(a) and 9(b) show WCET bounds for the benchmark example for two different cache sizes: 1,024 and 128 entries for different percentages of unknown information, that is, the number of accesses for which the address is unknown in the conventional analysis and the number of accesses where the reuse distance is unknown in the case of probabilistic analysis. When the number of entries is significantly larger than the working set (N = 1,024), LRU-based caches provide the best WCET if all or almost all addresses are known. However, as the amount of uncertainty grows (which is the common case), the WCET for LRU-based caches becomes rapidly pessimistic. Conversely, the randomized cache still provides low WCET estimates and degrades smoothly as the amount of uncertainty grows. This is a desirable property, since small changes in the deployment conditions with respect to the analysis conditions do not cause abrupt changes in the WCET.

Figure 9(b) shows results for a cache where data occupies most of the space (N = 128). LRU-based caches show an abrupt increase in WCET when only a few addresses



(a) 1,024-entry caches. (b) 128-entry caches.

Fig. 9. WCET estimations for caches with different sizes, expressed in number of cache entries, when varying the fraction of unknown addresses.

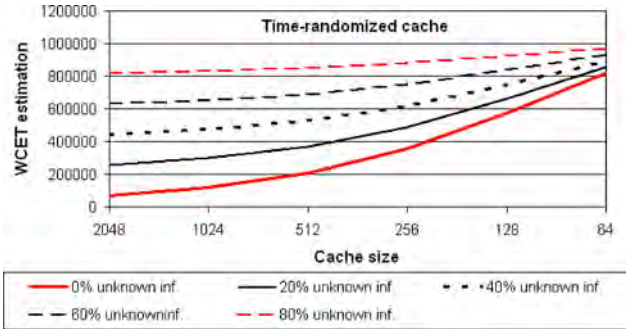
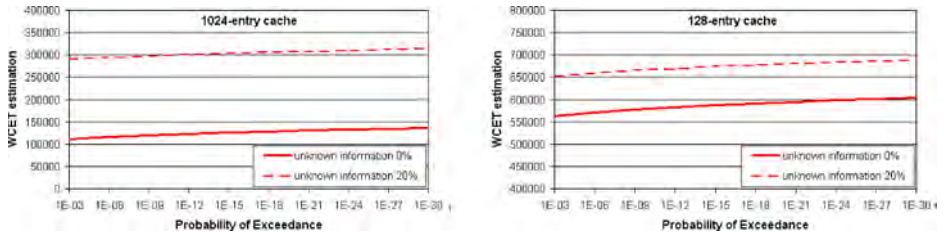


Fig. 10. WCET estimations for a time-randomized cache and different fractions of unknown addresses when varying the cache size expressed in number of cache entries.

are unknown. With SPTA, caches experience higher replacement probabilities because of the lower N (see Equation (3)), but the WCET increases smoothly and is significantly lower than that of LRU-based caches when at least 3% of the addresses are unknown. Although not shown, when the cache size is lower than the working set, LRU-based caches are unable to provide any cache benefit, even if all addresses are known. Time-randomized caches still provide some hits because any piece of data has some probability of remaining in cache after an arbitrary number of evictions.

Sensitivity Analysis. The second set of experiments studies the sensitivity of the different cache designs to the cache size for different levels of unknown information (0%, 20%, 40%, 60%, and 80%). In Figure 9, we can observe that WCET estimations for LRU-based caches are good only if the working set fits in cache and all (or almost all, i.e., 10%) addresses are known. However, if some addresses are unknown or the working set does not fit in cache, the WCET becomes very pessimistic. Conversely, the time-randomized cache provides low WCET estimates that degrade smoothly as we reduce the cache size or increase the fraction of unknown addresses, as shown in Figure 10.

Note that in our experimental setup, we made the cache the only source of WCET overestimation. Under this scenario, when the amount of unknown information is 0%, the WCET estimation matches the actual execution time. What we change in each experiment is the amount of information the analysis method knows, which is the source of pessimism of the WCET with respect to the actual execution time. In the particular case of no unknown information, the WCET estimation of static timing analysis with LRU caches is 5–7x better than static probabilistic timing analysis with random caches. However, this scenario is unrealistic, as in most cases a significant fraction of the information needed for the timing analysis is unavailable. This is what we show



(a) 1,024-entry cache. (b) 128-entry cache.

Fig. 11. WCET estimations for a time-randomized cache of varying size, expressed in number of cache entries, when varying the exceedance probability.

in Figure 9 as we move right on the x-axis. Some examples of lack of information are (i) matrices indexed with input-dependent variables, (ii) pointers, (iii) stack size and location for functions with input-dependent parameters, etc.

Sensitivity to Exceedance Probability. The third set of experiments studies the sensitivity of the time-randomised cache to the exceedance probability used as threshold to determine the WCET. Results are shown for four particular combinations of cache sizes (1,024 and 128 entries) and unknown information fractions (0% and 20%), although trends are very similar for other scenarios. The exceedance probabilities we studied range between 10^{-3} and 10^{-30}. Figures 11(a) and 11(b) show results for 1,024 and 128 entries caches, respectively. Decreasing the exceedance probability to increase the timing margin in the system does not have a significant impact in the WCET obtained. Decreasing the exceedance probability by three orders of magnitude requires increasing the WCET by 0.9% on average across configurations. Hence, the WCET estimations can be used in systems with the highest timing constraints.

5. CONTEXTUALIZING PROARTIS

Cost pressure, flexibility, extensibility, and the demand for increased functional complexity are changing the fundamental paradigms for the definition of automotive and aeronautics architectures. In this section, we motivate the applicability of the PROARTIS approach in the avionics domain. Similar reasoning can be applied to the automotive domain.

In the past, conventional avionics systems were based on the *federated architecture* paradigm, in which each computer is a fully dedicated unit, which may have local optimizations. However, since most of the federated computers perform essentially the same functions (input acquisitions, processing and computation, and output generation), a natural optimization of resources is to share the development effort by identifying common subsystems, standardizing interfaces, and encapsulating services—in other words, adopting a modular approach. This is the purpose of the Integrated Modular Avionics (IMA) concept, where 'integrated' refers to the sharing of these resources by several systems. To that end, federated architectures have been replaced by *integrated architectures* in which the same computer can execute one or more applications, or partitions, potentially of different criticality levels.

One fundamental requirement of IMA is to ensure the possibility of *incremental qualification*, whereby each partition can be subject to verification and validation—including timing analysis—in isolation, independent of the other partitions, with obvious benefits for cost, time, and effort. From the perspective of timing analysis,

incremental qualification relies on each hardware and software component that is part of the system, such as the cache at hardware level or the linker at software level, exhibiting the property of *time composability*. In the most general definition, composability ensures that, given a certain property of each item of a collection, that property can be determined for each item taken in isolation and does not change when that item is brought together with other items. Time composability refers to the fact that the execution time of a software partition, observed in isolation by the timing analysis, is not affected by the presence of other partitions on the same system.

The PROARTIS approach attacks the root of this problem: first, by reducing, or even completely eliminating, the information required to calculate WCET estimations, which reduces the cost of acquiring the required knowledge to model the timing behaviour of the system; second, by reducing the dependence of the WCET estimations on execution history. In this way, software execution times are less dependent on previous executed software components, which can belong to other partitions. As a result, the system integration can be easily achieved. In the extreme case, if all the execution history dependence is completely eliminated, each individual resource will be timing composable, allowing partitions to be replaced without requiring the timing of other components to be re-analyzed.

6. BACKGROUND

The first papers in the real-time community related to the contribution of this article used the words 'stochastic analysis' [Gardner and Lui 1999], 'probabilistic analysis' [Tia et al. 1995], 'statistical analysis' [Abeni and Buttazzo 1998] and 'real-time queuing theory' [Lehoczky 1996]. Since Díaz et al. [2002], the 'stochastic analysis' of real-time systems has been used regularly by the community regardless of the approach (probabilistic or statistical). We note that the terminology used in the past is somewhat inconsistent; one of the objectives of this article is to define precisely those terms so that they can be widely used.

The literature on probabilistic methods is vast. It is difficult to draw a cohesive view of the field. We rather describe here the precise application of these techniques to the timing analysis of real-time systems. Probabilistic approaches for real-time systems are promising because they answer questions that cannot be addressed in a deterministic manner (e.g., distribution of different task parameters) and consider more realistic models of task execution time or the expression of soft real-time constraints. In general, a probabilistic approach provides the distribution function of a parameter.

Probabilistic approaches for real-time systems can be divided into two main classes.

— One class consists of extracting quantitative information for one or more parameters (e.g., distribution of execution times) from samples of observations collected by monitoring the system. The first paper considering such statistical estimation of worst-case execution times is Edgar and Burns [2001] in which extreme value statistics are used to model the behavior of caches. More recently, Hansen et al. [2009] improved the analysis, addressed flaws in the previous work, and produced a refined method also based on extreme value statistics. This line of work is promising if one can show that hypotheses like independence (for instance) are met, which unfortunately is not the general case [Griffin and Burns 2010]. Such hypotheses can be met for instance by designing platform (hardware and software) support as we suggest in Section 3.2.

  The RapiTime tool [2008] from RAPITA is the only timing analysis tool that captures execution time distributions, also called execution time profiles (ETP), and implements a method based on the theory of copulas [Bernat and Newby 2006] to compute the distribution of execution times of the worst-case path. The results in

Navet et al. [2007] and Binns [2004] are the only response time analyses that belong to this class of probabilistic approaches.

— The second class of approaches concerns the temporal analysis of systems with at least one parameter described by a random variable. We present here the papers that consider the temporal analysis of systems with WCET described by random variables. In this case, the probabilistic analysis specifically decides the schedulability of the system, that is, answers the question of whether the deadlines are met, and with what probability, when the elementary execution times and thus the WCET are random variables?

Therefore, such analysis takes WCET estimations for each program in input (as provided as random variables) and returns the probability of missing the deadline for the set of programs. The first results on probabilistic schedulability analysis were proposed in Lehoczky [1996], where the author extends queuing theory under real-time hypotheses. Even if this work was subsequently improved by results such as those of Zhu et al. [2002], the main limitation concerns the use of the same probability law for the execution times of all programs. This latter hypothesis is not always realistic, for example, the traces of execution of two different programs could fit (after a WCET estimation) to different probability laws.

Another relevant work [Abeni and Buttazzo 1998] proposes an interesting definition of probabilistic deadline, but the authors consider a special scheduling model providing isolation between tasks. The results presented in Gardner and Lui [1999] and Tia et al. [1995] consider also execution times as random variables and special assumptions are made on the critical instant.

The main result of this class is proposed in López et al. [2008], but the analysis is difficult to use in practice for computational reasons. An improvement based on resampling was proposed in Refaat and Hladik [2010], but this work fits better in the first class of approaches for the statistical analysis of real-time systems.

The main difficulty when using both approaches is the consideration of worst-case behaviors that are known to be rare events. In this case, Rare Events Theory needs to be used in order to ensure that the worst-case behavior is considered. These approaches have the advantage of predicting unexpected behaviors, but they also have the disadvantage of offering incomplete information (the distribution function is not entirely defined). Except for Bernat et al. [2002; Bernat and Newby 2006], all existing approaches consider that the parameters given by random variables are independent. This hypothesis is due to theoretical limitations, but also to the difficulty of describing and capturing the relations between different parameters, like the relation between the execution times of two programs for instance.

Besides some known probabilistic results (such as the stability of Markov chains) the results used for the schedulability of real-time systems are mainly based on operations with random variables which extend deterministic calculation. Queuing theory is a notable exception, but it is usually classified as operations research.

As regards the three distributions of Extreme Values Theory (i.e., Gumbel, Frechet, and Weibull) [Gnedenko 1943], we are interested in using Gumbel's which provides information on the right-tail part of a distribution (thus the interesting part for a WCET estimation). A first step toward fitting the data to this particular distribution is needed and errors could be introduced during this step (thus different levels of confidence might be proposed). One solution to decrease the errors is proposed in Hansen et al. [2009] and is based on block maxima (grouping different values within a distribution into the maximal one). Another rare events theory that interests us is Large Deviation Theory [Lewis and Russel 1997]. This theory also deals with tail events, and extends Central Limit Theory. Thus it has the feature that it can only be applied to a

sum of independent and identically distributed random variables. This property could be useful for instance if we wanted to reason on the combination of execution traces taken from different parts (e.g., paths, units) of a program.

7. ONGOING AND FUTURE WORK

In this article, we have shown the feasibility and the requirements to properly implement a probabilistic timing analysis approach. Given the seminal nature of this work, we consider several lines of future work.

First, providing more measurement-based and static probabilistic analysis techniques. A key aspect of this line is that these techniques have to be designed hand-in-hand with the platform, which has to ensure that the hypotheses used by the technique (e.g., i.i.d) are demonstrated by the hardware and low-level software.

Second, at the hardware level, our aim is to find designs which facilitate analysis in general and improve the behavior of both static and measurement-based probabilistic timing analysis. We want to extend our solutions to hardware resources that are hard to analyze with current analysis techniques, for example, the Translation Lookaside Buffer (TLB).

Third, at the software level, our objective is to determine how software support for randomisation can be used in conjunction with the hardware techniques to further facilitate probabilistic timing analysis.

And finally, there is a shift towards multicore processors in the real-time market. The main impediment of the use of multicores in real-time systems is the difficulty of predicting how tasks will interact in shared hardware resources. We believe that probabilistic timing analysis is a promising solution to bound (in a probabilistic manner) intertask interactions so that we know the effect that one task may introduce in the WCET estimations for another.

8. CONCLUSIONS

The industrial demand for new functionality and higher levels of performance in addition to reduced cost, weight, and power dissipation calls for the adoption of advanced hardware acceleration features, of which cache memories are the epitome. However, the timing behavior of systems using these advanced hardware features is very hard to analyze with current timing analysis techniques: the level of knowledge needed to compute tight WCET estimations, as well as the time, effort, cost, and complexity required to acquire and comprehend, all down to the finest the system details, become unaffordable. This is because the large amount of state information that is carried by a modern processor incurs a combinatorial explosion when one tries to enumerate all possible histories of execution even for simple programs. Thus, as long as current analysis techniques are unable to scale up to the challenge, increased hardware complexity will lead to a significant degradation of the quality of the resulting products.

We call these problems the time analysis walls. Probabilistic timing analysis attacks these walls and has potential for tearing them down. It is our contention that the cost of acquiring the required knowledge to perform trustworthy analysis can be significantly reduced by adopting a hardware/software architecture whose execution timing behavior eradicates dependence on execution history. One way to achieve this independence is via introducing randomization in the timing behavior of the hardware and software (while the functional behavior is left unchanged), together with new probabilistic analysis techniques.

We have devised two different flavors of probabilistic timing analysis. The first, SPTA, is static in the sense that it derives a-priori probabilities from a model of the processor or software. The second, MBPTA, derives probabilities from complete runs of the program under study that are made on the target hardware. From these runs we

collect data about the timing behavior of the program when run on low-level software and hardware elements of the PROARTIS platform with randomized timing behavior.

In each case we have shown the main challenges and opportunities in implementing that approach. Our initial results for the cache using the SPTA approach show that (1) SPTA requires knowing the reuse distances of every access, which in fact are much easier to obtain than the addresses of memory operations required for static timing analysis. (2) SPTA has minimal dependence on the amount of knowledge required to provide tight WCET estimations. Moreover, with SPTA, any lack of information in the analysis has modest negative effect on the WCET determination and the tightness of the resulting WCET estimations degrades smoothly with increasing lack of knowledge.

Finally, we have outlined the direction of our current and future work in the regard of the subject matter.

ACKNOWLEDGMENT

The authors would like to thank Robert I. Davis from the University of York for his help in deriving Equation (3).

REFERENCES

Abeni, L. and Buttazzo, G. 1998. Integrating multimedia applications in hard real-time systems. In *Proceedings of the 19th IEEE Real-Time Systems Symposium (RTSS98)*. 4–13.

Albrecht, K., Raimund, K., and Puschner, P. P. 2010. Avoiding timing anomalies using code transformations. In *Proceedings of ISORC*.

Berger, E. D. and Zorn, B. G. 2006. DieHard: Probabilistic memory safety for unsafe languages. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM Press, New York, 158–168.

Bernat, G. and Newby, M. 2006. Probabilistic WCET analysis, an approach using copulas. *J. Embed. Comput*.

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

Binns, P. 2004. Statistical estimation of aperiodic response times when scheduled on top of static timelines. In *Proceedings of the 1st International Workshop on Probabilistic Analysis Techniques for Real-Time and Embedded Systems (PARTES'04)*.

Charette, R. 2009. This car runs on code. *IEEE Spectrum Online*. http://spectrum.ieee.org/green-tech/advanced-cars/this-car-runs-on-code.

Clarke, P. 2011. Automotive chip content growing fast, says gartner. http://www.eetimes.com/electronics-news/4207377/Automotive-chip-content-growing-fast.

Díaz, J., Garcia, D., Kim, K., Lee, C., Bello, L., López, J. M., and Mirabella, O. 2002. Stochastic analysis of periodic real-time systems. In *Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS02)*. 289–300.

Edgar, S. and Burns, A. 2001. Statistical analysis of WCET for scheduling. In *Proceedings of the 22nd IEEE Real-Time Systems Symposium (RTSS'01)*. 215–225.

Gardner, M. and Lui, J. 1999. Analyzing stochastic fixed-priority real-time systems. In *Proceedings of the 5th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99)*. 44–58.

Gnedenko, B. 1943. Sur la distribution limite du terme maximum d'une seris aleatoire. *Ann. Math. 44*, 423–453.

Griffin, D. and Burns, A. 2010. Realism in statistical analysis of worst case execution times. In *Proceedings of the 10th International Workshop on Worst-Case Execution Time Analysis (WCET'10)*. 44–53.

Hansen, J., Hissam, S., and Moreno, G. A. 2009. Statistical-based wcet estimation and validation. In *Proceedings of the 9th International Workshop on Worst-Case Execution Time (WCET) Analysis*.

Lahiri, K., Raghunathan, A., and Lakshminarayana, G. 2001. LOTTERYBUS: A new high-performance communication architecture for system-on-chip designs. In *Proceedings of the 38th annual Design Automation Conference (DAC'01)*. 15–20.

Lehoczky, J. 1996. Real-time queueing theory. In *Proceedings of the 10th IEEE Real-Time Systems Symposium (RTSS'96)*. 186–195.

Lewis, J. T. and Russel, R. 1997. An introduction to large deviations for teletraffic engineers. http://www.statslab.cam.ac.uk/%7Errw1/ld/LD-tutorial.ps.

López, J., Díaz, J. L., Entrialgo, J., and Garcia, D. 2008. Stochastic analysis of real-time systems under preemptive priority-driven scheduling. *Real-time Syst. 40,* 2, 180–207.

Lundqvist, T. and Stenstrom, P. 1999. Timing anomalies in dynamically scheduled microprocessors. In *Proceedings of RTSS*.

MerasaD2.2. 2008. Merasa project. `http://www.merasa.org/downloads/Deliverable_2_2.pdf`.

Navet, N., Cucu, L., and Schott, R. 2007. Probabilistic estimation of response times through large deviations. In *Proceedings of the WIP Session of the 28th IEEE Real-Time Systems Symposium (RTSS'07)*.

PROARTIS. 2010. Probabilistically analyzable real-time systems. http://www.proartis-project.eu/.

Quinones, E., Berger, E., Bernat, G., and Cazorla, F. 2009. Using randomized caches in probabilistic real-time systems. In *Proceedings of the 22nd Euromicro Conference on Real-Time Systems (ECRTS)*. IEEE, 129–138.

RapiTime. 2008. RapiTime tool. www.rapitasystems.com.

Refaat, K. S. and Hladik, P.-E. 2010. Efficient stochastic analysis of real-time systems via random sampling. In *Proceedings of ECRTS*. 175–183.

Reineke, J., Wachter, B., Thesing, S., Wilheim, R., Polian, I., Eisinger, J., and Becker, B. 2006. A definition and classification of timing anomalies. In *Proceedings of WCET*.

Tia, T., Deng, Z., Shankar, M., Storch, M., Sun, J., Wu, L., and Liu, J. 1995. Probabilistic performance guarantee for real-time tasks with varying computation times. In *Proceedings of the 2nd IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'95)*. 164–174.

Wilhelm, R., Engblom, J., Ermedahl, A., Holsti, N., Thesing, S., Whalley, D., Bernat, G., Ferdinand, C., Heckman, R., Mitra, T., Mueller, F., Puant, I., Duschner, P., Staschulat, J., and Stenström, P. 2008. The worst-case execution-time problem—overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst. 7*, 1–53.

Yue, L., Bate, I., Nolte, T., and Cucu-Grosjean, L. 2011. A new way about using statistical analysis of worst-case execution times. In *Proceedings of ECRTS*.

Zhou, S. 2010. An efficient simulation algorithm for cache of random replacement policy. In *Proceedings of the IFIP International Conference on Network and Parallel Computing (NPC'10)*.

Zhu, H., Hansen, J., Lehoczky, J., and Rajkumar, R. 2002. Optimal partitioning for quantized EDF scheduling. *Proceedings of the 23rd IEEE Real-time Systems Symposium (RTSS'02)*. 202–213.

Received June 2011; revised October 2011; accepted November 2011