Microprocessors and Microsystems 47 (2016) 287–302

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

Fitting processor architectures for measurement-based probabilistic timing analysis

Leonidas Kosmidis[a,b], Eduardo Quiñones[b], Jaume Abella[b,*], Tullio Vardanega[c], Carles Hernandez[b], Andrea Gianarro[d], Ian Broster[e], Francisco J. Cazorla[b,f]

[a] Universitat Politecnica de Catalunya, Barcelona, Spain
[b] Barcelona Supercomputing Center, Barcelona, Spain
[c] University of Padova, Padova, Italy
[d] Cobham Gaisler, Gotemburg, Sweden
[e] Rapita Systems Ltd, York, England
[f] Spanish National Research Council (IIIA-CSIC), Barcelona, Spain

ARTICLE INFO

Article history:
Received 3 January 2016
Revised 21 June 2016
Accepted 18 July 2016
Available online 20 July 2016

Keywords:
Worst-case execution time
Processor architecture
Cache memories
Probabilistic analysis
Time randomization

ABSTRACT

The pressing market demand for competitive performance/cost ratios compels Critical Real-Time Embedded Systems industry to employ feature-rich hardware. The ensuing rise in hardware complexity however makes worst-case execution time (WCET) analysis of software programs – which is often required, especially for programs at the highest levels of integrity – an even harder challenge. State-of-the-art WCET analysis techniques are hampered by the soaring cost and complexity of obtaining accurate knowledge of the internal operation of advanced processors and the difficulty of relating data obtained from measurement observations with reliable worst-case behaviour. This frustrating conundrum calls for novel solutions, with low intrusiveness on development practice. Measurement-Based Probabilistic Timing Analysis (MBPTA) techniques offer the opportunity to simultaneously reduce the cost of acquiring the knowledge needed for computing reliable WCET bounds and gain increased confidence in the representativeness of measurement observations. This paper describes the changes required in the design of several high-performance features – massively used in modern processors – to meet MBPTA requirements.

1. Introduction

The market for Critical Real-Time Embedded Systems (CRTES), which includes the automotive and avionics sectors, is experiencing an unprecedented growth [1]. While crucial to keeping competitive advantage, the inclusion of increasingly sophisticated value-added functions, such as for example Advanced Driver Assistance Systems, causes CRTES makers to continually seek higher guaranteed computational performance while striving to contain cost and power budget. This goal can only realistically be achieved by adding complex and powerful hardware accelerator features such as caches or multicore designs.[1]

However, the use of aggressive performance-enhancing hardware features may highly complicate the computation of reliable and tight timing bounds.[2] Worst-Case Execution Time (WCET) analysis is an integral step of verification for real-time systems in general, and for CRTES in particular. One common use of WCET bounds is for schedulability analysis to ascertain whether application tasks can complete within their assigned deadlines under all conditions.

Numerous techniques exist for performing WCET analysis, ranging from measurement-based to static analysis, via hybrid variants that use elements of both [2]. Measurement-based techniques rely on user's ability to design stressful tests in which the application under test is run in conditions similar to the worst ones that can arise during operation. Static timing analysis is challenged by the difficulty to model accurately the timing of complex hardware designs, and also by the increasing amount of information needed to feed the models to estimate the WCET. Finally, hybrid approaches alleviate some of the problems of those techniques to handle com-

* Corresponding author. Fax: +34 934137721.
E-mail address: jaume.abella@bsc.es (J. Abella).

[1] This trend deflects from prior practice in CRTES, where processors used to be in-order and cacheless, to simplify verification of timing behaviour.

[2] In the context of timing analysis, a reliable bound is a bound that can be supported by strong arguments and proofs.

http://dx.doi.org/10.1016/j.micpro.2016.07.014

plex hardware, but hybrid approaches are subject to similar limitations.

The availability of more powerful hardware and the quest for more functional value per unit of product also prompt CRTES industry to consider adopting mixed-criticality design solutions for their systems. From the timing perspective, which is the focus of this paper, the challenge with mixed-criticality systems lays in the need for solutions to ensure strict temporal isolation between programs assigned to different criticality levels, so that their behaviour can be deemed composable in the time dimension.[3] In the absence of effective means to abate the pessimism of WCET analysis, however, mixed-criticality solutions that achieve time isolation by fencing budget allowances, risks incurring massive over-provisioning, which defeats the purpose of combining systems together.

Probabilistic techniques may greatly aid on all of those fronts. In particular, with Measurement-Based Probabilistic Timing Analysis (MBPTA) methods [3–6], the execution time of the application can be accurately modelled – at some level of execution granularity – by a probability distribution. MBPTA seeks to determine WCET estimates for arbitrarily low probabilities of exceedance, termed probabilistic WCET or pWCET. As a consequence, there is some residual risk (in the form of an exceedance probability) beyond which it cannot be proven that a pWCET bound cannot be exceeded. However, this residual risk is upper bounded with a given probability, which can be determined at a level low enough to suit the needs of system design in the application domain. For example, the residual risk can stay in the region of 10^{-9} per hour of operation, largely below the acceptable probability of failure in certified systems.

Under MBPTA, at a given granularity of execution, the response time of every individual execution component at that level (e.g., an instruction) is assigned a distinct probability of occurrence. This trait – which shall not be confused with the probability of that component *being* executed in a run of the program – is described by a probabilistic Execution Time Profile (ETP), expressed by the pair: <timing vector; probability vector>. The timing vector in the ETP enumerates all its possible response times. For each response time in the timing vector, the probability vector lists the probability of occurrence of that response time in an instance of execution. Hence, for execution component $\mathcal{C}_i$ we have $ETP(\mathcal{C}_i) = \langle \vec{t_i}, \vec{p_i} \rangle$ where $\vec{t_i} = (t_i^1, t_i^2, \ldots, t_i^{N_i})$ and $\vec{p_i} = (p_i^1, p_i^2, \ldots, p_i^{N_i})$, with $\sum_{j=1}^{N_i} p_i^{N_j} = 1$. At the program level, MBPTA requires that the ETP for the program exercised during analysis matches or upper-bounds program's ETP during operation.

The processor architecture is instrumental in ensuring that individual instructions have an associated ETP. As this guarantee in turn is a crucial enabler to a sound and effective application of MBPTA, the processor architecture is the level of execution granularity on which we focus in this work.

Contribution. Within the context of the FP7 PROXIMA project [7] we describe the architecture features that a processor should possess to be amenable by construction to the use of MBPTA. We term this quality *MBPTA compliance*. In presenting our case, we offer insight on the costs that may be incurred in actual implementation of a MBPTA-compliant processor. To that end, we categorise processor resources according to their timing behaviour and detail how they should be designed for use in a MBPTA-compliant processor. Without loss of generality, we consider the inner operation of the processor to employ a number of passive resources (e.g., caches, buffers, buses, etc.). We assume each processor instruction to use some of those resources in a given order, whether in sequence or in parallel. We design processor resources so that each of them can be assigned a given ETP. To achieve this for all resources, we use *time randomisation* in *some*, actually very few, of them. Resources that are not time randomised must be assigned a local upper bound to their response time that can be safely composed. We assume a time anomaly free baseline architecture.

The remainder of this paper is organised as follows. Section 2 introduces PROXIMA and contextualises this work. Section 3 presents the requirements that MBPTA places on processor hardware. Section 4 classifies hardware resources in a taxonomy specifically related with MBPTA. Section 5 presents software-only solutions that could be applied to make commercial-off-the-shelf processor hardware fit for MBPTA. Section 6 presents a demonstrative implementation of a processor architecture, purposely designed for compliance with MBPTA. Section 7 surveys related work. Section 8 draws some conclusions and outlines the future of this line of work.

2. Context within PROXIMA

This work has been performed within the scope of PROXIMA [7], an Integrated Project (IP) of the Seventh framework programme for research and technological development (FP7). PROXIMA objectives include providing a complete toolchain enabling low-cost timing verification for systems based on multicore and manycore processors implementing critical real-time functionalities. In particular, PROXIMA toolchain includes the following main elements:

- *Hardware and software platforms amenable for MBPTA*. One of the key elements of the toolchain is a hardware platform providing the timing properties required by MBPTA to facilitate obtaining reliable and tight pWCET estimates. This hardware platform has been implemented in a FPGA prototype used in the Space domain. Alternative software-only solutions have been developed to enable MBPTA on top of commercial off-the-shelf (COTS) processors that include a non-MBPTA-compliant version of the Space prototype, an Infineon AURIX T277 and a Freescale P4080 processors. MBPTA compliance in future manycore processors has also been investigated by means of architectural simulators.
- *MBPTA-compliant real-time operating systems (RTOS)*. The RTOS needs to be enhanced with features so that its contribution to the execution time of the analysed tasks is made constant, and hence, time-composable, and its impact on the hardware and software state is neutral w.r.t. the properties needed to attain MBPTA compliance, thus being transparent for the timing analysis process. RTOS features have been implemented as part of PikeOS, RTEMS-SMP, ERIKA and some research-oriented RTOS.
- *Timing analysis tools*. Appropriate methods for the estimation of pWCET are required to account for the timing behaviour of the underlying hardware/software platform. They must be compatible with the tracing methods in place, and capable of providing pWCET estimates that hold valid in front of the different sources of execution time variation that can be exercised during operation such as hardware/software initial state, input values, execution path traversals, etc. Some of these methods have been implemented as part of RapiTime commercial toolchain whereas others will remain as standalone tools.

These elements have been implemented by a set of industrial and academic partners including hardware, RTOS and timing analysis tool vendors and related research institutions. Evaluation is performed on a number of case studies from the avionics, space,

[3] Time composability is had when the timing behaviour of an individual software component does not change in the face of composition when the system is integrated, and so, the timing analysis performed in isolation remains valid at system integration.

railway and automotive domains. The project finishes in September 2016, so most technologies have reached a high degree of maturity.

This paper reviews MBPTA-compliant hardware behaviour to deliver the timing properties needed to estimate reliable and tight pWCET. We also show how some of these goals can be achieved in the absence of MBPTA-compliant hardware.

3. MBPTA requirements on hardware design

3.1. Taxonomy of timing analysis techniques

We differentiate three main timing analysis types, each of which has a deterministic and probabilistic variant.

- Measurement-based deterministic timing analysis (MBDTA) techniques take advantage of the observation data obtained from executing the programs of interest on the real processor hardware. Simple high-watermark techniques have been used in industry for many years. They are usually coupled with detailed analysis of the software structure that provides confidence in exercising those worst-case paths or scenarios at the application level that can arise during system operation. To make safety allowances for the unknown (which the cognizant associates with the difficulty of determining the hardware worst case), an engineering margin is often added to the computed bound. The intent of the margin is striking some sound balance between pessimistic overkill and risk of underestimation. Determining a reliable and tight engineering margin is extremely difficult – if at all possible – especially when the system may exhibit discontinuous changes in timing due to unanticipated timing behaviour. The confidence had on the WCET estimate determined with MBDTA is, therefore, fully-dependent on the ability of the end user to identify what behaviour needs to be triggered in the hardware and software to observe the WCET that can occur during operation (or execution times close to it) and to produce program inputs that trigger that behaviour. The increasing complexity of the hardware (i.e. the use of cache hierarchies and multicores) is also a threat for the scalability of this approach [8].
- Static deterministic timing analysis (SDTA) techniques rely on the construction of a cycle-accurate model of the processor and an abstract representation of the application code. SDTA searches the resulting state space for the worst case, with constraint-based integer linear programming. Obviously, such an analysis cannot carry forward *all* the possible states of execution. Hence, conservative choices are made during the process, thus trading a reduction in the state space for increased pessimism [9–11]. SDTA has abundant need for information about the timing specification of the processor hardware and flow facts for the application. As the prediction must necessarily err on the side of pessimism, any lack of information about the timing behaviour of the object of analysis (e.g., the address of a memory access needed to determine if execution hits or misses in cache) or about processor timing behaviour degrades the tightness of the WCET estimate. Further, the result of the analysis is as reliable as the input provided to it [8]. The rise in complexity of next-generation CRTES greatly exacerbates this problem: the volume of detailed knowledge needed to construct a sufficiently accurate execution model as well as the time, effort, cost and complexity entailed in acquiring that information, challenge the adoption of SDTA for CRTES applications.
- Hybrid techniques build upon MBDTA, but collect execution time measurements at finer granularities such as, for instance, per function, per basic block, etc. Then, they operate on those measurements to account for unobserved behaviour. For instance, RapiTime [12] creates a representation of the control flow of the program and operates on the measurements obtained for each of the elements in the program, to generate measurements for unobserved execution paths. This approach can lead to higher confidence than that for traditional MBDTA, by inflating WCET estimates, and a lower effort/cost of use. However its confidence still depends on the ability of the user to make sound assertions on flow facts and to understand and control numerous hardware-related aspects such as cache interactions among programs and inter-task interference in the use of hardware shared resources in multicores [8].

At the present state of the art, probabilistic timing analysis (PTA) can be applied in either a static (SPTA) [5] or measurement-based (MBPTA) [4] fashion: we refer the interested reader to those works for details on PTA fundamentals. In this work we focus on MBPTA only since it is more mature for industrial use than SPTA [8].

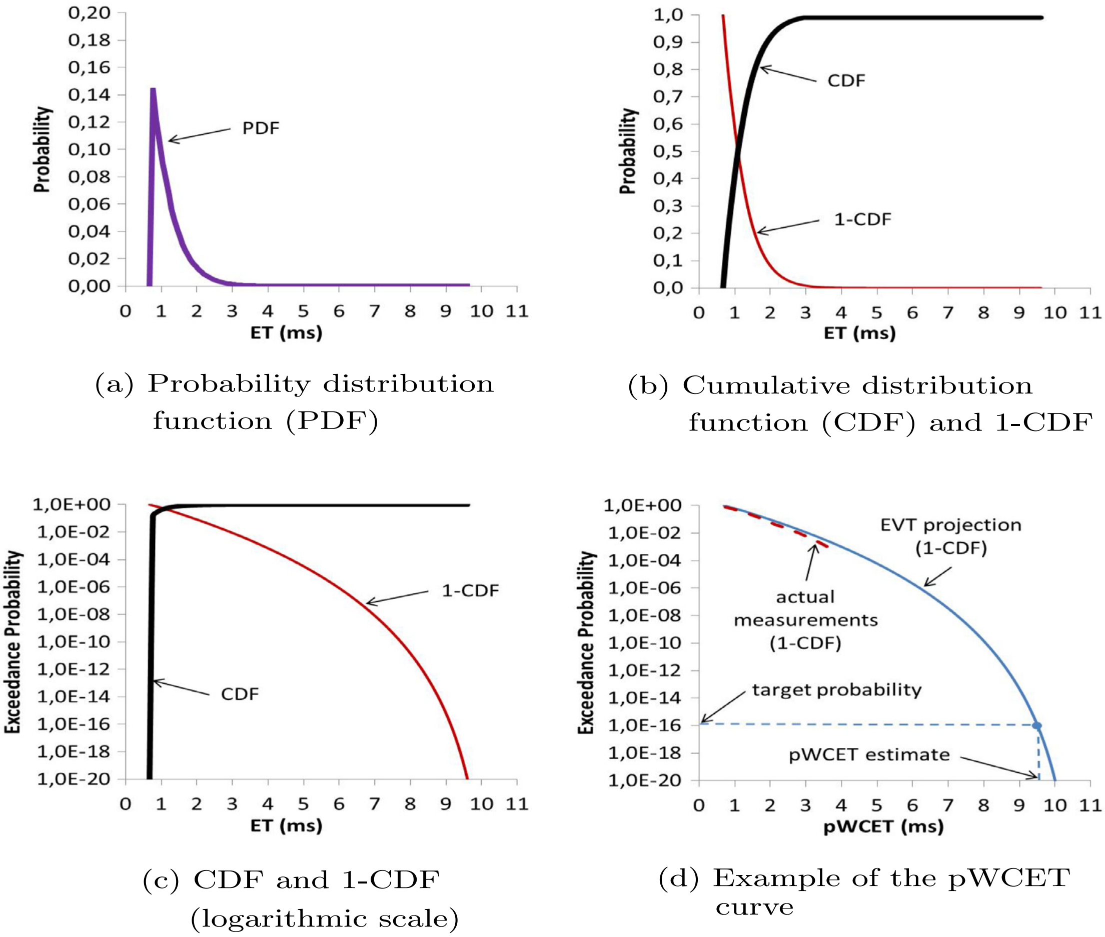
MBPTA generates a probability distribution that describes the maximum probability with which an instance of the program can exceed its assigned budget. As illustrative example, Fig. 1(a) shows the probability distribution function (PDF) of the execution times of a (single-path) synthetic program on a MBPTA-compliant processor architecture. From the PDF, one can build the cumulative distribution function (CDF) and its complementary (1-CDF) exceedance function or pWCET, which tells the probability that the execution time of one run of that program may exceed a given threshold (see Fig. 1(b) and 1(c)). Using conventional means, for a set of R runs, one could only derive an exceedance probability of $1/R$ at most. For smaller probabilities, techniques such as Extreme Value Theory (EVT) [13] are needed: Fig. 1(d) illustrates the hypothetical result of applying EVT to a collection of 1000 measurement runs taken on a MBPTA-compliant processor. The dotted line represents the 1-CDF derived from the observed execution times. The continuous line represents the projection obtained with EVT.

3.2. Requirements

MBPTA considers events resulting from the observation of end-to-end measurement runs of the program, thus at coarser granularity than processor instructions. MBPTA builds upon EVT [4,13] to estimate pWCET. Yet MBPTA and EVT are not the same thing. We clarify this by differentiating the requirements that MBPTA imposes due its use of EVT and other MBPTA requirements to satisfy *representativeness* requirements.

- Extreme Value Theory: The use of EVT requires that its input, i.e. the observed execution times in our case, to be described with independent and identically distributed (i.i.d.) random variables. Two random variables are said to be independent if they describe two events such that the occurrence of one event does not have any impact on the occurrence of the other event. Two random variables are said to be identically distributed if they have the same probability distribution. Specific statistical tests can be used to check these properties on a set of execution times, see Section 6.

  It is worth noting that some authors have shown that independence across observations is not strictly needed as long as maxima are independent or the dependence across maxima is weak [14,15]. However, in the rest of this paper we build upon independent data since it is a by-product of MBPTA-compliant platforms presented in this work.
- Representativeness: The goal of MBPTA is to derive – from execution times obtained during analysis – WCET estimates that hold valid during operation. However, the pWCET estimates obtained with EVT stay valid under the execution conditions considered at analysis. Those execution conditions include all



(a) Probability distribution function (PDF)

(b) Cumulative distribution function (CDF) and 1-CDF

(c) CDF and 1-CDF (logarithmic scale)

(d) Example of the pWCET curve

Fig. 1. Synthetic program PDF, CDF, 1-CDF and pWCET curve.

events that may impact the execution time of the program under analysis (e.g., memory layout, arbitration in shared resources). Analysis-time conditions experienced can differ from those during operation simply because the latter may be unknown. In order to cover this gap, MBPTA imposes several representativeness-related requirements beyond those of EVT (a data sample of a random variable so that each execution time observation is i.i.d.). MBPTA defines representativeness as the requirement in which the impact of any relevant event affecting execution time is properly upper-bounded at analysis time, where a relevant event corresponds to any event occurring with a probability above a cutoff threshold (e.g., 10^{-9} per hour of operation). Hence, MBPTA requires providing evidence on the fact that analysis time observations capture the impact of those events that can arise during operation and significantly impact execution time and so, pWCET [16,17].

3.3. Execution time profiles

The axiomatic existence of an ETP per *dynamic* instruction (i.e. an individual instance of that program instruction in a given run of the program) ensures that, under MBPTA, each potential execution time of the program has a distinct probability of occurrence. It therefore follows that every program run has an associated ETP, which enables to achieve the prerequisite i.i.d. execution time behaviour [18] (EVT requirement). To obtain reliable results, it is also necessary that the ETPs, which characterise the program runs during WCET analysis, can be shown to upper bound the probabilistic distribution of the program's execution time that may occur during operation (representativeness requirement). The wisdom and consequence of this particular requirement are discussed in Section 4.

Unfortunately, regardless of whether ETPs are sought for program instructions or full programs, they *cannot* be determined in most current processor architectures since the events that affect instructions' execution time, e.g. cache hits/misses, cannot be soundly attached a probability of occurrence. So we need to understand what features a processor architecture should possess to allow ETPs to exist.

For the sake of keeping the discussion simple, the rest of the paper focuses on single-path programs. However, we note that MBPTA has been proven effective on arbitrary multi-path programs. At least three techniques can be employed to that effect: (1) applying MBPTA to each program path – if feasible – and choosing the highest pWCET estimate obtained across them. (2) Collecting measurements on an extended version of the target program, where all conditional constructs are modified to exhibit a probabilistic timing behaviour that upper-bounds all possible alternative branches [6]. This solution requires the availability of the program sources which is difficult to meet in practice. (3) Using more elaborate methods that require basic block[4] coverage of measurement observations, augment the resulting data by negatively padding the cost of each basic block for positive (acceleration) effects that could occur across unobserved program paths, and syn-

[4] A basic block is a fragment of the program's code, which has a *single* entry point and a *single* exit point.

thetically construct the worst-case path from them. The details of the latter method are presented in [19].

4. Probabilistically modelling the timing behaviour of processor resources

When the latencies with which each resource responds should have an attached probability of occurrence, the execution time of the instructions using those resources can then also be captured probabilistically. In this respect, the probabilistic execution time of an instruction is a function of the ETP of the resources it uses and how they are arranged, in series or in parallel. Ultimately, this enables capturing the execution of the whole program, which is comprised of instructions, in a probabilistic manner.

For a processor to be MBPTA compliant, the pWCET estimates obtained for the programs that run on it must hold valid for the whole operational life of the system. Hence, they must hold valid for every run of the programs of interest under all (or a desired subset of those that can arise during operation) execution conditions. To understand how the timing behaviour of processor resources needs to be modelled for those guarantees to be obtained, we first show how the MBPTA process works.

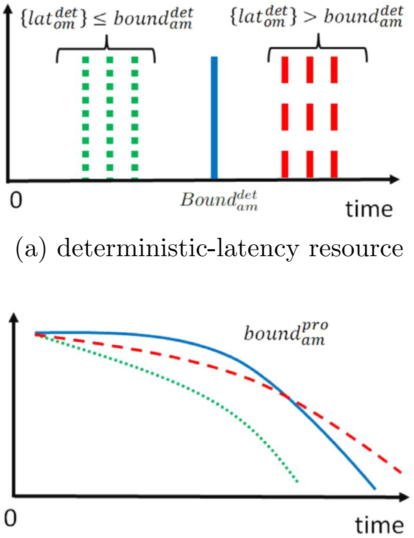
4.1. Probabilistic timing analysis process

Systems amenable to MBPTA have two distinct modes of use: one for analysis, and another for operation.

- The analysis mode is used to obtain pWCET estimates that hold valid during system operation. To this end, the timing behaviour of the system in that mode must upper bound that of the system after deployment, as used in real scenarios. This guarantees that circumstances that can occur during the lifetime of the system cannot alter its timing behaviour in a way that has not already been upper bounded at analysis time.
- The operation mode is used during actual operation. In this mode, timing conditions are unrestricted (or restricted to a specific subset) and can thus lead to lower execution times than those experienced in the analysis mode.

By intent, the analysis mode requires that the timing behaviour of the system as a whole and of its individual components in isolation (seen at the granularity of execution of interest) either upper bounds or matches that which will occur in operation mode. For MBPTA-compliant processor architectures, this condition can be achieved in either a deterministic or a probabilistic manner. Accordingly, any pWCET estimate obtained by analysis is a reliable upper bound of the execution times that may occur after deployment in operation. Next we discuss what needs to be done for different hardware resources.

Fig. 2 provides a schematic view of the meaning of (a) deterministic upper-bounding and (b) probabilistic upper-bounding. In both figures, the x-axis represents execution time, and the y-axis the probability for any particular latency to occur (this is obviously 1 in the case of deterministic resources). In Fig. 2(a), the solid vertical line represents the analysis-mode bound (am), $Bound_{am}^{det}$ for the latency of a component. If in the operation-mode (om), the actual latencies, $\{lat_{om}^{det}\}$, are below $Bound_{am}^{det}$, which is shown with the dotted lines, then the obtained bound is reliable. If it cannot be ensured that this is the case, the operation-time actual latencies (dashed lines) can be bigger than the analysis-mode bound $\{lat_{om}^{det}\} > Bound_{am}^{det}$, hence the bound is not reliable and cannot be used. In Fig. 2(b) the solid curve represents the analysis-mode upper-bound ETP of the latency of the resource, $Bound_{am}^{pro}$. We say that $ETP_i \geq ETP_j$, that is, ETP_i probabilistically upper-bounds ETP_j, if for any cutoff probability the execution time of ETP_i is higher or equal than the execution time of ETP_j. Hence, if actual latencies for the resource are like the dotted curve, then they are probabilistically upper-bounded by $Bound_{am}^{pro}$ (solid line). If latencies match those described by the dashed curve, they are not probabilistically upper-bounded by $Bound_{am}^{pro}$.



(a) deterministic-latency resource

(b) probabilistic-latency resource

Fig. 2. Deterministic and probabilistic upper-bounding latencies.

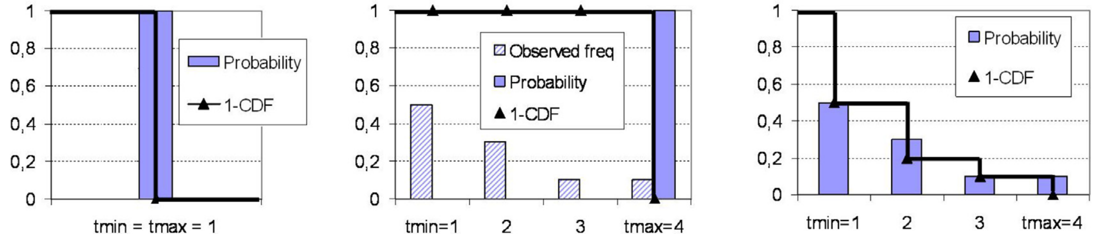
4.2. Taxonomy of hardware resources for canonical MBPTA compliance

We term *jitterless resources* the processor resources that have a fixed latency, independent of the input request and of the past history of service. Several hardware resources in current processor architectures are jitterless such as, for instance, integer additions or read operations in a register file. Jitterless resources are easy to model for all types of static timing analysis. For MBPTA techniques, the ETP of a jitterless resource jl is given by: $ETP_{jl} = \langle(l), (1.0)\rangle$, where l is the latency of the resource. Its PDF is shown in Fig. 3(a).

Other resources, for instance cache memories, have a variable latency: we call them *jittery resources*; their latency depends on their history of service, i.e. the execution history of the program, the input request, or a combination of them. Let us discuss each such case in turn:

- Dependence on execution history. Some resources are stateful and their state is affected by the processing of requests. If latency depends on the internal state of the resource and this state is in turn affected by previous requests, then we say that the resource latency depends on the execution history of the program. With caches, the latency of an access request depends on whether it is a hit or a miss, which in turn depends on the sequence of previous accesses to memory.
- Dependence on input request. The latency is determined by the data carried by the request: data are usually encoded in the instruction that issues the request, or stored in its input registers. This is the case for some floating-point operations whose latency depends on the actual values operated. For instance, typically dividing by a power-of-2 takes shorter than dividing by any other value.

Jittery resources have an intrinsically variable impact on the WCET estimate for a given program. The significance of this impact depends on the magnitude of the jitter, the program under study, and the analysis method. For any given jittery resource, either all requests to it are assumed to incur the worst-case latency



(a) Jitterless resource (jl)

(b) Jittery upper-bounded resource (wl)

(c) Jittery time-randomised resource (rj)

Fig. 3. Probabilistic timing behaviour of a single instruction for each type of resource.

– as long as timing anomalies can be excluded [20] – or the resource is time-randomised. The design choice for a given resource needs to trade the design cost for time randomising against the degradation of WCET tightness for always assuming worst-case latency.

The ETP for a resource r_{wl}, assumed or configured to worst-case latency, can be expressed as $ETP_{r_{wl}} = \langle (l_{max}), (1.0) \rangle$, where l_{max} is the worst-case latency of the resource. An example of the impact of such upper bounding is shown in Fig. 3(b). In the example, the actual probabilities for each latency are unknown; only frequencies can be obtained; upper bounding therefore is needed. This would correspond, for instance, to the case of a floating-point divider whose latency depends on the input values operated since, typically, we cannot determine what their distribution will be during operation.

Conversely, the ETP of a time-randomised jittery resource r_j is: $ETP_{r_j} = \langle (l_j^1, l_j^2 \ldots, l_j^k), (p_j^1, p_j^2, \ldots p_j^k) \rangle$ where l_j^i and p_j^i represent the different latencies of the resource r_j and their associated probabilities of occurrence. This is shown in Fig. 3(c). This could be the case of a cache access to a time-randomised cache, whose hit and miss probability depend on the (probabilistic) state left by previous cache accesses. Note that the probability of a given latency is different from the frequency with which it may occur. For instance, consider a resource R_1 with $\vec{t_1} = (t_1^1, t_1^2)$. Latency t_1^1 in the timing vector would have a true probability of occurrence $p_1^1 = 0.5$ if – in the implementation of that resource – on every request to it we tossed a coin and the request had latency t_1^1 if we saw heads and t_1^2 otherwise. In contrast, we could have a deterministic stateful resource R_2 with latency $\vec{t_2} = (t_2^1, t_2^2)$. If for R_2 we *observed* that, for a given program, 50% of the requests take t_2^1 and 50% t_2^2, we would have a 50% observed frequency for each possible latency of that resource, but not necessarily a true 50% probability.

For the purposes of MBPTA, the timing behaviour of jitterless and jittery (either upper-bounded or time-randomised) resources can all be described probabilistically by ETP.

4.3. MBPTA compliance via padding

The ultimate goal of a MBPTA compliant architecture is to ensure that measurements taken during analysis at program granularity are subject to a probabilistic behaviour defined by an ETP that upper-bounds that of the program during operation. The previous three cases (jitterless, upper-bounded and time-randomised resources) – together with the proper control on input-dependent sources of jitter (e.g. execution paths) – define the canonical approach to reach MBPTA compliance.

Notably, there are other ways to achieve MBPTA compliance such as *execution time padding*. With padding, a fixed value or a distribution is composed (added) to the program ETP at analysis such that the result of the composition is another ETP that upper-bounds that of the program during operation. This is better illustrated with an example. Let us assume we have a single path program comprising floating point operations. Further assume that all floating point operations can take a variable latency from l_{min} to l_{max} depending on the values operated. Controlling values operated is, in general, beyond the reach of the user. In this scenario padding can be used by adding to each of the measured execution times of the program $n_{fpops} \times (l_{max} - l_{min})$. This approach makes the pessimistic assumption that during analysis measurements, each of the n_{fpops} floating point operations of the program experienced a delay of l_{min}, while during operation each of them may take l_{max}. Hence, for each operation we increase the execution time observation by the maximum impact this can have $l_{max} - l_{min}$. Note that this is a form of enforcing the worst-case latency by software-only means.

More sophisticated forms of execution time padding are possible. For instance, let us assume that the ETP of an instruction i at analysis does not upper-bound its ETP during operation. Further assume another instruction j for which its analysis-time ETP upper-bounds the operation one. If an argument can be built on the fact that the reduction in execution time caused by i is smaller than the increase caused by j, and both i and j always execute, then the net impact is an analysis-time ETP upper-bounding the one during operation, which suffices for the application of MBPTA.

4.4. ETP of several execution components

A composite ETP can easily be determined for every individual program component (ETP_{pc}), e.g. a dynamic instruction, that uses processor resources, which has an associated ETP describing their latency. That is $ETP_{pc} = f(ETP_1, ETP_2, \ldots, ETP_n)$, where ETP_i is the probabilistic execution time of resource r_i.

- *Sequential composition*: the ETP, $f_s(ETP_1, ETP_2, \ldots, ETP_n)$, resulting from sequential composition is one where latencies and probabilities are determined by the type of dependence across the input ETP (whether systematic or probabilistic, as shown later in Section 4.5). The reader should note that sequential composition as intended here is architectural, hence referring to execution, and not mathematical, hence related to abstract interpretation. The latter is employed in SPTA and uses the convolution operator for combining the ETPs of static instructions (e.g., instructions in the object code of the program).

  Let us assume two ETPs, $ETP_1 = \langle (1, 2), (0.5, 0.5) \rangle$ and $ETP_2 = \langle (5, 10), (0.5, 0.5) \rangle$. Further assume that whenever ETP_1 takes latency 1, then $ETP_2 = \langle (5, 10), (0.8, 0.2) \rangle$ and whenever ETP_1 takes latency 2, then the second ETP is $ETP_2 = \langle (5, 10), (0.2, 0.8) \rangle$. In this case, $ETP_{1+2} = f_s(ETP_1, ETP_2)$, leading to $ETP_{1+2} = \langle (6, 7, 11, 12), (0.4, 0.1, 0.1, 0.4) \rangle$. Still, ETP_2 takes, for instance, latency 5 with probability 0.5 be-

cause $P(ETP_1 = 1) \times P(ETP_2 = 5) + P(ETP_1 = 2) \times P(ETP_2 = 5)$ is $0.5 \times 0.8 + 0.5 \times 0.2 = 0.5$.
The key trait here is that the dependence that ETP_2 has on ETP_1 can be modelled probabilistically. As a result, the executions carried out during analysis, capture the behaviour of this dependence and hence, cause it to be covered by the pWCET estimate derived to bound the execution time during operation.
This is the typical case for the ETP of cache accesses since the ETP of a given cache access depends on what the previous accesses did. For instance, if a first access hits, it does not evict any data and the second access may have a given hit probability. However, if the first access misses, it will evict some data likely decreasing the hit probability of the second access. Still, the second access has an ETP since the dependence between the first and the second access is probabilistic given that the first access will hit or miss with a true probability when using time-randomised caches.
- *Parallel composition*: processor resources may also be arranged in parallel. Examples of parallel resources are some particular designs of cache memories and translation lookaside buffers (TLB), where cache access and address translation can occur in parallel. With parallel arrangements, no dependence across ETP can exist, since for that to exist some sequential relation across ETP should occur, which should be addressed by sequential composition. The probabilities of the parallel composition ($f_p(ETP_1, ETP_2, \ldots, ETP_n)$) correspond to the multiplication of probabilities across ETP. However, the latencies correspond to the maximum latency of the probabilities multiplied. This is illustrated with the following example. Let the ETP for two program components be $ETP_1 = \langle(1, 4), (0.4, 0.6)\rangle$ and $ETP_2 = \langle(2, 3), (0.3, 0.7)\rangle$ respectively. The ETP from their parallel composition, $ETP_{1+2} = f_p(ETP_1, ETP_2)$, is $ETP_{1+2} = \langle(2, 3, 4), (0.12, 0.28, 0.6)\rangle$.

4.5. Dependence across ETP

The property of independence and identical distribution can be erroneously construed as needing instructions, and their associated ETP, to be independent of one another. This is incorrect: the i.i.d. property applies – in certain conditions – to the observation of the execution time of individual dynamic instructions across multiple executions. Notably however, the i.i.d. properties may not apply across distinct dynamic instructions (that is to say, to fragments of program execution that contain more than one instruction). Instructions may in fact have dependences among them when the outcome of one random event that represents the execution of one dynamic instruction has an impact on the ETP of following instructions.

We call *causal dependence* any dependence among two instructions in a given precedence order such that the execution of the earlier one affects the timing behaviour of the later one. Obviously, the execution time of the earlier one determines when the later one can start executing, but our notion of causal dependence actually means that the latency a given instruction not only affects the time at which the later one starts but also its duration.

We differentiate two types of causal dependences among a source (preceding) instruction and a target (subsequent) instruction that do not prevent the latter instruction from exhibiting a MBPTA-compliant timing behaviour across program runs.

- *Systematic dependence*: The ETP of the target instruction is affected by the execution of the source instruction. This effect may alter the ETP of the target instruction in any way like, for instance, shifting some latencies in its ETP or making new latencies appear in the ETP of that instruction. None of this however causes the target instruction to lose its MBPTA-compliant behaviour.
This can be better understood with an example. Recall the goal of MBPTA is to control sources of execution time variability in such a way that the observations taken during the analysis stage can be used to upper bound probabilistically the timing behaviour of the program during operation. Let us consider two instructions, one source and one target, on a given basic block, *bb*1. Let us also assume that the ETP of the target instruction is $ETP_{target}^{isol} = \langle(t_1, t_2, t_3), (p_1, p_2, p_3)\rangle$ if it runs in isolation. Further assume that the execution of the target instruction as part of the basic block *bb*1, hence in the presence of the source instruction, is $ETP_{target}^{bb1} = \langle(t_1, t_2, t_3', t_4), (p_1, p_2, p_3', p_4)\rangle$. In this new ETP the probability of t_3 changes and a new latency t_4 can be experienced. In this example the target instruction, which in fact represents a dynamic instruction, is executed as part of the basic block. Such target (dynamic) instruction is attached to a single ETP, ETP_{target}^{bb1}, irrespective of this being different from the ETP holding when the instruction was executed in isolation. Further, this ETP stays constant during analysis and operation. Therefore, all observations of the execution time of the target instruction as part of this basic block are observations of this ETP_{target}^{bb1}. The key trait here is that the ETP must hold for every dynamic instruction over successive executions of the program. In the previous example, if the initial conditions are fixed, the target dynamic instruction in *bb*1 will have a single ETP.
- *Probabilistic dependence*: The execution of the source instruction has a probabilistic effect on the ETP of the target instruction. This is the case of memory accesses to a time randomised cache. A probabilistic causal dependence causes that dynamic instruction to suffer a transformation in its ETP. However, given that the causal effect in the target instruction is probabilistic, this is equivalent to applying a transfer function *transf*() that takes as an input an ETP and provides as an outcome another ETP $tranf(ETP_{target}^{isol}) = ETP_{target}^{bb}$. Again, the key trait is that the target (dynamic) instruction is always subject to the same ETP_{target}^{bb} thus enabling MBPTA to properly capture its timing effects at analysis time analogously as they will occur during operation.

Overall, on a PTA-compliant platform, any hardware and software state with bearing on the execution time after of any dynamic instruction of the program is reached with a given probability. Therefore, one can build the ETP *of every single program path* that can be traversed by an observable execution by collecting the execution time of each final state of that system and its corresponding probability of occurrence. Therefore, the execution time of the program as a whole (seen as the traversal of a given path) has an ETP and is, hence, a random variable with i.i.d. properties.

4.6. More complex single-core processor architectures

We have shown that jittery deterministic resources need to be redesigned to make their timing behaviour amenable to MBPTA by construction. This can be done by either randomising their timing behaviour or enforcing them to their worst-case latency. Resources with probabilistic latency perfectly fit the MBPTA principles. However, jittery processor resources exist that do not easily fit in the taxonomy we used in Section 4.2. This is the case of resource buffers, also known as first-in first-out (FIFO) queues or simply buffers.

A buffer resource may stall if it gets full, which increases the latency of the requests that use it. Stalls across pipeline stages may for example occur owing to contention for buffer space; those stalls would be real enough to fear, but difficult to predict causally.

The main characteristic of buffer resources, however, is that they are not *sources of jitter* but rather *jitter propagators* [21]. The intuition here is that if all jitter that occurs in a processor is probabilistic, that is, it is solely due to time-randomised resources, any combination of random events has a given probability of occurrence. Now, as every single combination of events causes the program to incur a distinct execution time, each execution time has a distinct probability of occurrence. For each combination of random events, resource buffers may get full and consequently increase the execution time of the program. However, buffers themselves do *not* introduce any change in the probability distribution of random events. The presence of buffers may well cause the execution time of the program to vary, but each execution time continues to have a true probability of occurrence, which is what MBPTA requires.

In general, all hardware resources can be made MBPTA-compliant as long as they either do not introduce jitter on their own (hence they are fixed-latency or else just jitter propagators), their jitter can be upper-bounded or else it can be randomised.

4.7. Multicore processor architectures

In single-core architectures, the execution time of a software program is influenced by (1) the initial processor state when the program starts executing – which in turn is affected by previous execution, (2) the RTOS interferences that it may suffer during execution, (3) the input data that influence control flow or data-dependent jitter in jittery processor resources, and (4) the randomisation occurring in processor resources.

The effect of initial conditions, (1) above, can be taken into account by flushing the state of all stateful resources (e.g., caches) prior to the execution of the program. For the RTOS, state-of-the-art solutions exist to make its interference amenable to probabilistic analysis [11].

The effect of input data on the control flow of the program is controlled by state-of-the-art techniques that work in unison with MBPTA [19]. For instance, authors in [19] show how to pad execution time measurements at basic block granularity to discount the benefit obtained by executing specific paths when that benefit would not be obtained through other paths. The effect of input data on the latency of processor instructions using resources with data-dependent jitter as well as the jitter introduced by the randomised hardware resources are controlled with standard PTA techniques [5].

In multicore architectures, in addition to all the sources of execution time variability that appear in a single-core architecture, a further one arises: inter-task interference.[5]

In general in single-core architectures, given two instructions i_x and i_y of the same program, where the subscripts determine the order in which each instruction is executed into the processor, i_y may have a potential impact on the execution time of i_x only if $y < x$, meaning that i_y executes prior to i_x. In a multicore, when several programs run in parallel, the execution time of one instruction $i_x^{T_1}$ in program T_1 may be affected by any other instruction $i_y^{T_j}$ from any program T_j that may run at the same on any other available core. If precedence or exclusion constraints are set in the system such that T_j can be asserted to not execute in parallel T_1, then the inter-task interference generated by $i_y^{T_j}$ does not affect $i_x^{T_1}$. If no such assertion can be made instead, T_1 and T_j can execute in any order. Hence they may execute in parallel on different cores, so that $i_y^{T_j}$ may cause inter-task interference on $i_x^{T_1}$. It is evident that we cannot conceivably capture the effect that any single instruction of any task $i_j^{T_k}$ may have on any other instruction of any other task $i_l^{T_m}$ in the system. Should this be required, MBPTA would become intractable. To prevent this, the design of MBPTA-compliant multicores must ensure that the worst effect that one program can have on the execution of any other program owing to inter-task interference can be probabilistically bounded.

Interestingly, the MBPTA-compliant design principles already outlined for single-core processors extend quite well to the design of multicore architectures. The resources for which this approach is most advantageous are those that are shared upward the processor hardware architecture off the core, where they may cause massive inter-task interference. Next we review them in detail.

Shared bus. The authors of [23] show that the arbitration latency of a shared bus can either be upper bounded at analysis time or randomised so that the timing behaviour observed at analysis matches or upper-bounds that which may emerge during operation. In fact, upper bounding the bus arbitration latency has been shown to be viable also for time-deterministic systems [24]. This approach ensures that the latencies and probabilities of the ETP derived for this resource already account for worst-case interaction in this shared resource. For instance, if latency is upper-bounded, the ETP accounting for arbitration delay will have the form $ETP_{bus} = \langle(latbus_{max}), (1.0)\rangle$, where $latbus_{max}$ stands for the maximum bus arbitration latency. Alternatively, if random (lottery) or random permutations arbitration is used, ETP can also be derived as already proven in [23].

Shared memory controller. The same approach used for buses can be applied to the arbitration in the memory controller. Thus, the latency of a shared memory controller can be upper bounded, which is fine for MBPTA compliance. Again, that measure is in line with findings for time-deterministic systems [25]. Thus, if latency is upper-bounded, the ETP for the memory controller will have the form $ETP_{memctrl} = \langle(latmemctrl_{max}), (1.0)\rangle$, where $latmemctrl_{max}$ stands for the maximum memory controller arbitration latency. Note that random (lottery) or random permutations arbitration can also be alternatively used since ETPs exist for both policies [23]. However, memory latency can also vary based on the last operation performed due to the fact that the latency of a read (or write) operation varies depending on whether the last operation was a read or write operation. Authors in [25] describe how to upper-bound memory access latency, so an ETP can also be derived for this component with the form $ETP_{DRAM} = \langle(latDRAM_{max}), (1.0)\rangle$, where $latDRAM_{max}$ stands for the maximum memory access latency. Note that in this case, latency cannot be randomised since it depends on non-probabilistic events such as the particular memory accesses performed by tasks running in other cores, which are unlikely to be known at analysis time.

Shared cache. Cache partitioning has been proved to be a practical way to attenuate the interference effects from cache sharing. This solution was first shown for time-deterministic systems [24]. However, since it eliminates all cache conflicts among tasks running on different cores, it cancels out the multicore side of the cache problem, and allows using, for each multicore, the solutions devised for single-core processors.

An alternative approach has been put forward in [26], where a hardware feature is proposed to limit the eviction frequency caused by individual tasks on a shared time-randomised cache. That mechanism allows controlling inter-task interference without resorting to cache partitioning, which reduces the pWCET against the partitioned case, as long as inter-task interference distributes randomly across sets. The rationale behind that mechanism is as follows: during the analysis phase the program under analysis is exposed to a given eviction rate in the shared cache. Then, during operation such eviction rate is not allowed to be exceeded by

[5] This term does not include the interference that in single core processor occurs in caches and TLBs owing to context switches. This is intentional as this overhead can be quantified probabilistically in the context of MBPTA [22].

tasks in other cores. Hence, the ETP experienced at analysis time upper-bounds operation conditions. In other words, the miss rate during operation in the shared cache can only be lower than the one during the analysis phase. Therefore, the multicore case does not differ from the single-core case for the purposes of MBPTA.

5. Software-only alternatives

Recently, for some (COTS) time-deterministic hardware resources (e.g., caches) software-only solutions have been shown to achieve the effects of the hardware design proposals presented above. So far, the design of those solutions has focused on cache memories [27,28], seeking the same type of MBPTA-related benefits as warranted by hardware-implemented random placement. The essence of those solutions is to place the data and the code of the application at random locations in memory so that their placement in time-deterministic caches that implement modulo placement becomes also random and thus, MBPTA requirements for caches are met. Obviously, this random placement is entirely transparent to the application and has no functional effect on it. Next we review those solutions and compare their properties against their hardware-only correspondents.

5.1. Software-only random placement

Software-only random placement aims at causing cache conflicts in sets to occur randomly by placing objects at random memory locations. For instance, if an object is placed in a random memory location *Loc*, given a cache with *S* cache sets, the particular set where the object will be placed in cache, $Loc \mod S$, is also random.

At the present state of the art, software-only random placement operates on individual software functions (i.e., syntactically defined program fragments), static variables, and stack frames. As some padding is required for those entities to be moved in isolation, the memory footprint of the program grows as a result of the application of this technique. Current experience shows [27,28] that the resulting bloat may be contained within acceptable limits.

5.2. Software vs hardware solutions

Hardware solutions place each cache line in an independent and random location in cache. Therefore, one can build an ETP for cache accesses of the form $ETP_{HWcache} = \langle (l_j^1, l_j^2 \ldots, l_j^k), (p_j^1, p_j^2, \ldots p_j^k) \rangle$ where latencies correspond to the different outcomes of the cache access (e.g., cache hit and cache miss) and probabilities depend on the previous (random) events in cache.

Conversely, software-only solutions do not randomise the placement of cache lines independently. Instead, cache lines in different objects have a true probability of conflicting in cache, whereas cache lines inside a given object have a fully deterministic behaviour among them. Still, this does not break MBPTA requirements since those deterministic behaviours observed at analysis time stay exactly the same during operation as the memory location of a given object is randomised but the lines that form the object retain their position relative to one another. Hence, there is a probability [29] that two lines from different objects are placed in the same cache set and thus, are able to evict each other. However, if those two lines belong to the same object, the probability of being in the same set is either 0 or 1 depending on whether their relative alignment is different or matches the size of one cache way respectively.

Still, probabilities can be attached to all events and thus, one can also build an ETP of the form $ETP_{SWcache} = \langle (l_j^1, l_j^2 \ldots, l_j^k), (p_j^1, p_j^2, \ldots p_j^k) \rangle$ for cache accesses under software-only random placement. While latencies will be the same for $ETP_{HWcache}$ and $ETP_{SWcache}$, probabilities will not, given that the probabilities of the different latency outcomes differ across hardware and software-only solutions.

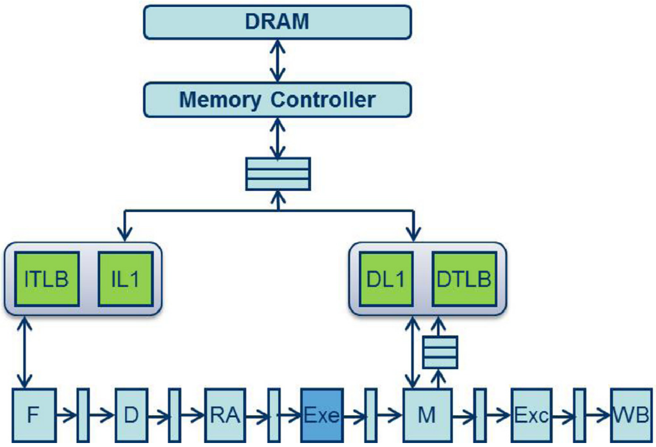


Fig. 4. Reference processor architecture.

It is important to appreciate however, that the actual values of probabilities need *not* be known in order for MBPTA to be applied. What is needed is that MBPTA requirements are satisfied, which is indeed the case for both hardware and software-only solutions. We can therefore contend that software-only solutions for cache placement can also be regarded as MBPTA compliant.

6. Case study

6.1. Designing a MBPTA-compliant processor architecture

The core architecture shown in Fig. 4 is an enhanced version of LEON3 processor used by the European Space Agency and its industrial suppliers in a number of missions [30].

The said processor consists of a pipeline with the following stages: fetch (F), decode (D), register access (RA), execution of non-memory operations (Exe), DL1 access (M), Exceptions (Exc) and write back (WB). The operations occurring in each stage are as follows:

- Fetch stage. The IL1 is accessed (and the instruction TLB, ITLB, on a IL1 miss) to obtain the next instruction to be executed. Branches are predicted to be taken always.
- Decode stage. Instructions are decoded. This stage is, in essence, an extra delay in the pipeline.
- Register access. Instructions read their input registers with fixed latency.
- Execute stage. Non-memory instructions are executed with a fixed latency that depends solely on the type of operation. Although originally floating-point division (FDIV) and floating-point square root (FSQRT) instructions had input data dependent latencies, they have been modified as described later. Memory operations compute their addresses.
- Memory stage. Load instructions access the DL1 (and data TLB, DTLB, on a DL1 miss). Indeed, they also access the write buffer. Store operations are placed in the write buffer for their offline processing. If the write buffer is full the pipeline will be blocked.
- Exception stage. Exceptions are managed here.
- Write-back stage. Results (if any) are sent to the register file.

The IL1 and DL1 are 16KB in size, 4-way set-associative, with 16B/line IL1 and 32B/line DL1. All caches implement random place-

Table 1
Description of the EEMBC benchmarks.

Name	Description
a2time	Angle to Time Conversion
basefp	Basic Integer and Floating Point
bitmnp	Bit Manipulation
cacheb	Cache “Buster”
canrdr	CAN Remote Data Request
aifft	Fast Fourier Transform (FFT)
aifirf	Finite Impulse Response (FIR) Filter
aiifft	Inverse Fast Fourier Transform (iFFT)
aiirflt	Infinite Impulse Response (IIR) Filter
matrix	Matrix Arithmetic
pntrch	Pointer Chasing
puwmod	Pulse Width Modulation (PWM)
rspeed	Road Speed Calculation
tblook	Table Lookup and Interpolation
ttsprk	Tooth to Spark

ment and replacement policies presented in [31]. The DL1 is write-through and no-write-allocate, so all store operations are propagated to memory. ITLB and DTLB are 8-entry fully-associative random-replacement, with 4KB page size, and their misses are handled by a hardware page-walker.

A demonstration prototype of the above processor design has been implemented in an Altera Stratix IV GX EP4SGX230 FPGA device operating at 80 MHz.

For its evaluation we use the EEMBC Automotive Benchmarks [32], which is a well-known benchmark suite representative of some existing real-time automotive functionalities. The description of each benchmark is conveniently provided in Table 1 for the sake of completeness.

6.2. Hardware modifications

In the quest for MBPTA-compliance, we have modified cache placement and replacement policies, as well as selected floating-point (FP) operations with a comparatively high jitter dependent on the input parameters. In the original processor design, all caches (DL1, IL1, DTLB, ITLB) implemented modulo placement and least recently used (LRU) replacement, whose sensitivity to history of execution makes them unable to meet the MBPTA prerequisites [31] unless appropriate software support is provided to the application [27].

Random placement and replacement have been implemented as described in [31]. In particular, random placement implements the latest design as described in [33]. Random replacement relies on the use of a pseudo-random number generator. While the one described in [33] has been shown to be convenient, the one described in [34] has appeared to generate random numbers with similar quality while being amenable to a much easier implementation on a FPGA.

For the FP unit we concentrated on the FDIV and FSQRT operations, whose latency jitter is highly dependent on the input parameters. The FDIV latency varies between 15 and 18 cycles, whereas the FSQRT latency varies between 23 and 26 cycles. Table 2 provides examples of input values leading to different latency outcomes.

Since, from the processor design perspective, the actual latency of those operations does not occur with a given probability, and all that one can infer from the application program is the frequency of their execution, which is of no use for MBPTA, the solution described in Fig. 3(b) needs to be applied. The implementation of FDIV and FSQRT has therefore been modified so that they always operate in 18 and 26 cycles respectively in the analysis mode. As we noted earlier, modifications of this kind cause the pWCET estimates to incur some (though limited) pessimism, but they make the corresponding hardware resources MBPTA compliant, which is what we are after here.

6.3. Deriving ETP

In view of the hardware modifications discussed above, the processor architecture of interest includes two main sources of randomised jitter, TLB and caches, each of which makes random contributions to the cumulative execution time of a program running on it.

We differentiate between two types of instructions: those that operate on the core (e.g. add, div, mult); and those that operate on memory (e.g. load, store). Core operations take a variable latency depending on whether they hit in the instruction cache and instruction TLB, whose ETP (ETP_{IL1} and ETP_{ITLB} respectively) are composed in parallel, and memory latency, which is accessed in case of a miss and whose ETP (ETP_{DRAM}) is composed sequentially with the composition of the instruction cache and the instruction TLB. This leads to what we term the ETP of the front-end (`fend`): $ETP_{fend} = f_s\big(f_p(ETP_{IL1}, ETP_{ITLB}), ETP_{DRAM}\big)$. Then, the resulting ETP, ETP_{fend} needs to be composed with the ETP of the buffer between the front-end and the back-end (ETP_{buf1}), the ETP of the decode stage (ETP_{dec}), the buffer after decode (ETP_{buf2}), the register access stage (ETP_{ra}), the buffer after register access (ETP_{buf3}), the core operations (ETP_{exec}), the buffer after execution (ETP_{buf4}), the memory operations stage (ETP_{mem}), the buffer after memory operations (ETP_{buf5}), the exceptions stage (ETP_{excep}, the buffer after exceptions (ETP_{buf6}) and the write-back stage (ETP_{wb}).

While ETP_{dec}, ETP_{ra}, ETP_{exec}, ETP_{mem}, ETP_{excep} and ETP_{wb} have the form <(l), (1.0)> for core operations, ETP for buffers have as many latencies as potential stalls they may produce, and their probability vector is 0.0 for all latencies but one, whose probability is 1.0. Which latency has probability 1.0 is determined by the state left by previous instructions. More details about how buffers increase execution time without expanding the number of probabilistic states can be found in [21]. If all actions occurred sequentially (thus omitting interactions in the buffer to memory), the ETP for core operations would be as follows:

$$\begin{aligned} ETP_{core} = f_s\big(&ETP_{fend}, ETP_{buf1}, ETP_{dec}, ETP_{buf2},\\ &ETP_{ra}, ETP_{buf3}, ETP_{exec}, ETP_{buf4}, ETP_{mem},\\ &ETP_{buf5}, ETP_{excep}, ETP_{buf6}, ETP_{wb}\big) \end{aligned} \tag{1}$$

Memory operations have the same ETP as core operations for the different stages and buffers except for the memory stage

Table 2
Input value examples triggering different latencies for FDIVD and FSQRTD.

Op.	Lat	Input 1		Input 2	
		hexa	decimal	hexa	decimal
FDIVD	15	0 × BFF0000000000000	-1.0	0 × 4000000000000000	2.0
FDIVD	18	0 × 001ABC0000000010	$3.717(\ldots) \cdot 10^{-308}$	0 × 3FF000400A07610C	1.00006107(...)
FSQRTD	23	0 × 4030000000000000	16.0		
FSQRTD	26	0 × 4008000000000000	3.0		

(ETP_{mem}). The memory latency, instead of depending on ETP_{mem}, depends on the time of the data memory path (`dmpath`) composed by the data cache and the data TLB, which are accessed in parallel, and memory latency, which is accessed sequentially: $ETP_{dmpath} = f_s\big(f_p(ETP_{DL1}, ETP_{DTLB}), ETP_{DRAM}\big)$. Therefore, the ETP for memory operations (still omitting interactions in the buffer to memory) is as follows:

$$\begin{aligned} ETP_{mem} = f_s\big(&ETP_{fend}, ETP_{buf1}, ETP_{dec}, ETP_{buf2}, ETP_{ra},\\ &ETP_{buf3}, ETP_{exec}, ETP_{buf4}, ETP_{dmpath},\\ &ETP_{buf5}, ETP_{excep}, ETP_{buf6}, ETP_{wb}\big) \end{aligned} \tag{2}$$

Finally, we must consider that the misses occurring in the DL1/DTLB and in the IL1/ITLB are serialised in the buffer that connects the core to the memory controller. Again, this buffer has an ETP of the same form as any other buffer ($ETP_{bufDRAM}$). Unlike previous buffers, where an instruction could only be delayed due to activities of older instructions, here data requests from some instructions may get delayed by instruction requests of younger instructions. Still, the buffer can only have a finite number of states and each state will have a probability that, hypothetically could be derived by expanding the probability tree from the beginning of the execution of the program. Thus, $ETP_{bufDRAM}$ should be composed serially with the ETP of the memory accesses, so ETP_{fend} and ETP_{dmpath} should be $ETP_{fend} = f_s\big(f_p(ETP_{IL1}, ETP_{ITLB}), ETP_{bufDRAM}, ETP_{DRAM}\big)$ and $ETP_{dmpath} = f_s(f_p(ETP_{DL1},, ETP_{DTLB}), . .ETP_{bufDRAM}, ETP_{DRAM})$ for a correct calculation of the ETP of core (Eq. 1) and memory operations (Eq. 2).

6.4. Checking the i.i.d. hypothesis

The existence of an ETP for individual instructions ensures that the program execution times exhibit the prerequisite i.i.d. property of MBPTA. With MBPTA, we empirically ascertain whether this claim holds, by using proper i.i.d. tests applied on the execution times of running EEMBC benchmarks [32] on the processor architecture.

To assert independence we use the Ljung–Box test [35] (LB). The Ljung-Box test is a powerful method that tests autocorrelation for different lags simultaneously, so for each datum with the next one (lag 1), the one after (lag 2), and so on and so forth. In particular we test all lags up to 20 as shown appropriate by authors in [36].

To check that the identical distribution hypothesis stands, we use the Kolmogorov–Smirnov (KS) goodness-of-fit test [37]. We use a 5% significance level (a typical value for this type of tests), whereby absolute values obtained with both the LB and KS tests should be above the threshold (0.05) to assert independence and identical distribution respectively. In particular, both tests, LB and KS, deliver values in the range [0,1]. Any value below the significance level (0.05) rejects the hypothesis, and cannot reject it otherwise.

For each benchmark, less than 1000 runs were needed for each program, in line with previous experience [4,31]. Running 1,000 times a program whose typical execution time is in the order of few milliseconds (as typical of CRTES) implies that pWCET estimates for that program can be obtained in a few seconds altogether, which is a rather affordable overhead for an industrial development timescale. Under the heading ‘Statistical tests’ Table 3 reports the results of both tests for all benchmarks. Since values for both tests, LB and KS, are always above the significance level, 0.05, both tests are passed in all cases, which proves that the example architecture meets the i.i.d. requirement of our MBPTA approach.

Table 3
Independence and identical distribution test results (2nd and 3rd columns), and average execution time and pWCET bounds of the complex MBPTA-compliant processor vs. an equivalent conventional processor (4th, 5th and 6th columns).

Benchmarks	Statistical tests		Timing analysis results		
	Independence	Identical distribution	Average Exec. Time	Max Exec. Time	pWCET 10^{-15}
a2time	0.31	0.34	0.10%	1.79%	8.44%
basefp	0.85	0.91	0.00%	0.06%	0.36%
bitmnp	0.97	0.77	−0.01%	0.12%	0.26%
cacheb	0.87	0.06	0.02%	0.41%	3.27%
canrdr	0.18	0.82	0.00%	0.00%	0.12%
matrix	0.25	0.70	0.00%	0.01%	0.11%
pntrch	0.79	0.93	0.00%	0.00%	0.12%
puwmod	0.99	0.85	0.00%	0.00%	0.12%
rspeed	0.16	0.50	0.00%	0.00%	0.12%
tblook	0.75	0.86	0.07%	0.86%	2.80%
ttsprk	0.36	0.80	0.00%	0.03%	0.24%

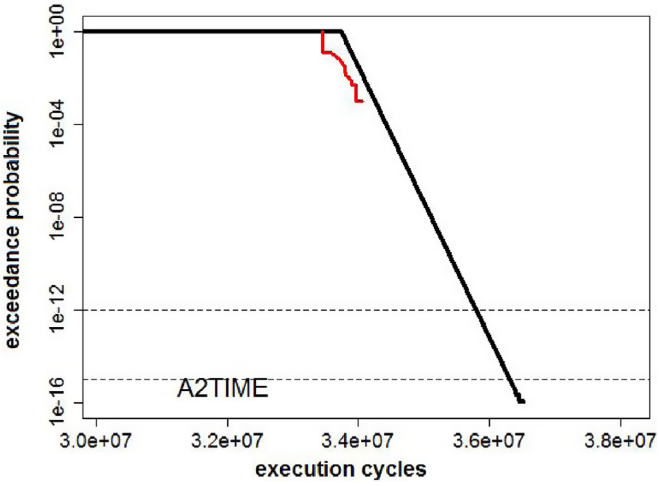


Fig. 5. pWCET estimates for the *a2time* benchmark program.

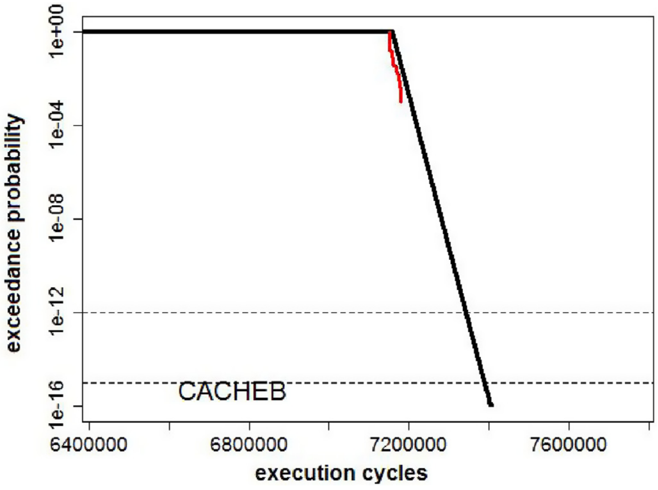


Fig. 6. pWCET estimates for the *cacheb* benchmark program.

6.5. pWCET

In this section we show the type of probabilistic WCET estimates that can be obtained for the example architecture, with the method presented in [4]. The black line reaching arbitrarily low exceedance probabilities in Figs. 5 and 6 plots the pWCET distribution obtained for the *a2time* and *cacheb* benchmark programs of the EEMBC suite, run on the example architecture. The red line (reaching only down to 10^{-3} exceedance probability) plots the empirical 1-CDF of the 1000 runs performed for these benchmarks. The *x*-axis shows the execution time and the *y*-axis the probability of exceeding it. Assuming for the sake of argument that an exceedance event represents a timing failure in the system, we may concentrate on exceedance probabilities of 10^{-12} and 10^{-15} per run, well within acceptable probabilities of failure in the safety regulation of automotive and avionics systems.

Increasing the exceedance probability (moving up in the *y*-axis) does not translate into large increases in the WCET estimates (moving right in the *x*-axis); quite the contrary in fact, as the pWCET curves appear to have a very steep slope. In general, the larger the number of random events entailed in a run (e.g., the number of cache accesses), the less likely that abrupt performance variations occur other than (if at all) at extreme exceedance thresholds. Thus, execution time variation is moderate and the pWCET curve is steep.

As the example processor architecture demonstrably meets the requirements needed for MBPTA, it can be argued that MBPTA can be applied to performance-aggressive hardware features. Interestingly, the MBPTA process stays unchanged in procedure and effort, while the pWCET estimates become considerably smaller (up to 9% in the specific experiment) than the engineering margin often applied in measurement-based deterministic timing analysis by industry (20%) in the case of [38].

To the best of our knowledge, complex architectures including caches, TLB, and staged pipelines with buffers, have not been unrestrictedly used with static timing analysis, unless with cautionary restrictions that mitigate the rapid degradation in the tightness of the WCET estimates that arise from resources being used whose state cannot be determined exactly. MBDTA also is at a loss with those processor architectures, because no suite of observation runs can possibly cover the whole state space of all resources exhaustively. Those techniques are also known to be fragile even to the way the program is built, because small changes in the way program objects are allocated in memory, which are hard to capture in test suites, may lead to abrupt changes in execution time.

6.6. MBPTA-compliant architectures performance

The above results show that the proposed MBPTA techniques, united with the proposed modification to the design of processor resources, enable CRTES designers to aim at considerably higher levels of guaranteed performance.

Notably, there is a further angle of interest to quantify the benefit of the MBPTA approach discussed in this paper. Under the heading ‘Timing analysis results’, Table 3 reports average execution times and pWCET estimates for an exceedance threshold of 10^{-15} per run, obtained for the EEMBC benchmark programs on the example processor architecture. The values are normalised against those obtained running the same programs on an analogous architecture that implements modulo placement LRU replacement caches and TLB instead of random placement and replacement, and where the latency behaviour of the FDIV and FSQRT operations had been set to operation mode, hence not set to the worst-case outcome as in the analysis mode.

For instance, *cacheb* executes in 7,150,507 cycles in the non-time-randomised architecture. The 1000 runs of *cacheb* on our time-randomised architecture take 7,152,211 cycles on average (0.02% more than without time randomisation) and 7,179,573 cycles at most (0.41% more than the actual execution time without time randomisation). The pWCET curve at an exceedance threshold of 10^{-15} per run is 7,384,084 cycles, as shown in Fig. 6 (3.27% higher than the actual execution time without time randomisation).

The average execution time of the MBPTA-compliant architecture is roughly the same as for the time-deterministic alternative, thus showing that time randomisation fares well even in the average case. If we compare the maximum observed execution time in the MBPTA-compliant architecture, it is only slightly above that of the time-deterministic alternative. Even more interesting, pWCET estimates are, on average, only up to 9% higher than the average performance obtained for a processor architecture implementing modulo and LRU as the placement and replacement policies for all caches, and without upper bounding FDIV and FSQRT latencies.

Whereas the WCET values for those programs on the time-deterministic architecture are not available (computing them would require the porting of static timing analysis tools, which was outside of the scope of this work), relevant literature shows that WCET values are intrinsically very conservative and can be many times greater than the average case [39]. Our study shows that, for our MBPTA-compliant design, the observed inflation was up to 8.44% for `a2time`, which allows arguing that MBPTA-compliant processors are viable for CRTES industry, thus below the usual 20% engineering margin applied on top of the maximum observed execution time [38]. The pWCET estimates are only up to 8.44% higher than the actual execution time on a time-deterministic architecture because the platform is properly designed so that the instruction and data sets of the programs fit in IL1 and DL1 caches respectively, thus experiencing very few misses, and FDIV and FSQRT operations are extremely infrequent in general, and in the EEMBC benchmarks in particular.

It is worth noting that the ETPs for individual dynamic instructions in our processor are non-independent across them (see Section 4.5). This occurs because random-placement caches (as an instance of a state-sensitive time-randomised resource) create dependence across instructions in the same run since any (random) cache set conflict occurring during a particular run holds systematically across the whole run. Such dependence across ETPs for different instructions is captured in the observations taken at analysis time, and are accounted for in the pWCET estimate derived with MBPTA.

7. Related work

There is an increasingly rich literature on the problem of WCET analysis. One substantial part of the state of the art, with more history and tradition, addresses Deterministic Timing Analysis (DTA) techniques, which include its static and measurement-based variants, SDTA and MBDTA respectively. The state of the art in DTA is comprehensively surveyed in [2], so we omit discussing it here. The other, more recent, but rather vibrant, considers the various flavours of PTA.

The static variant of MBPTA, known as SPTA, has been studied for relatively simple processor architectures that use time-randomised caches [5,40,41]. Authors in [42] present the first comprehensive comparison among SDTA, SPTA and MBPTA techniques showing that if enough information is had for the timing analysis, whether one technique is superior to the others depends on the particular characteristics of the program under analysis. Therefore, there is not a dominant technique. However, as detailed in [8], techniques such as SDTA and SPTA are more sensitive to the amount of information had for the timing analysis process, with effects on either tightness or reliability. This relates to the fact that SDTA and SPTA need a detailed timing model of the hardware and additional flow-facts describing the operation of the software such as value ranges and memory addresses.

Conversely, MBPTA, the focus of this paper, requires amounts of information comparable to those obtained by end users in the context of MBDTA, but it scales to arbitrarily complex software running on top of high-performance hardware easing the collection of evidence usable for certification purposes [43]. MBPTA has been used in the context of time-randomised architectures for single-path programs [4,44] and multi-path programs [6,19].

At hardware level, random placement was proposed in [31] to enable the use of set-associative caches for MBPTA. [45] and [16] discuss the reliability of pWCET estimates obtained with MBPTA on top of random placement caches. In particular authors discuss *representativeness* related to the fact that some random events may have a low probability to be captured in the measurement runs, yet have a high impact on execution time. The latter work [16] and other recent works [17] conduct thorough analysis of those scenarios in the context of MBPTA and propose ways to address them.

EVT has been applied to time deterministic architectures to derive execution time bounds [46]. While randomisation – and creating deterministic bounds to jitter resources – is not needed for the application of EVT, deterministic architectures seriously difficult deriving a representativeness argument. That is, with EVT-only approaches, building a representativeness argument that analysis-time execution conditions capture those that can arise during operation is completely left to the user. Instead, with MBPTA-compliance – through randomisation and deterministic upper bounding – the space of potential execution conditions is automatically, transparently and randomly sampled as the user makes more runs. Hence, representativeness just requires the user to perform enough runs to probabilistically capture the impact of the different sources of jitter, rather than the user designing specific experiments to reach that goal [47].

8. Conclusions and future work

In this paper we have shown that in order for MBPTA to be usable economically and assuredly, the target processors should be designed such that every program instruction have a distinct probabilistic ETP. We have shown that this ETP can be built incrementally from the timing behaviour of the processor resources used by that instruction.

Using MBPTA on MBPTA-friendly processor architectures, the timing interference between competing applications, which is one of the key problems in mixed-criticality systems, can be studied from the angle of exceedance probability: the probability that the execution time of a program exceeds a given threshold. We have shown that this threshold is tight, owing to the natural attenuation of multiple worst-case events generated as i.i.d. random variables. We have shown that the probabilistic worst-case execution time bounds obtained with the proposed technique are only marginally greater (around 12% in our case study) than the average-case performance of time-deterministic processor architectures. This allows achieving higher guaranteed (feasible) utilisation for mixed-criticality systems, because little would be lost, if at all, in raw processor performance, and a great reduction would be had in the pessimistic over-provisioning incurred with traditional techniques. The use of Extreme Value Theory allows setting bounds for execution-time budgets at levels of exceedance probability that satisfy the system assurance requirements. Normal mitigation measures (i.e. adding some form of redundancy, setting up a safe state, etc.) can be taken if protection guarantees had to be provided for higher-criticality applications at conditions past the given exceedance threshold.

Acknowledgment

This work has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement 611085 (PROXIMA, www.proxima-project.eu). Support was also provided by the Ministry of Science and Technology of Spain under contract TIN2015-65316-P and the HiPEAC Network of Excellence. Leonidas Kosmidis is funded by the Spanish Ministry of Education under FPU grant AP2010-4208. Jaume Abella has been partially supported by the MINECO under Ramon y Cajal postdoctoral fellowship number RYC-2013-14717. The authors wish to acknowledge Michael Houston, Liliana Cucu–Grosjean and Luca Santinelli for contributing to the genesis of this work.

References

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

[2] 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, G. Staschulat, P. Stenströem, The worst-case execution time problem: overview of methods and survey of tools, Trans. Embed. Comput. Syst. 7 (3) (2008) 1–53.

[3] J. Hansen, S. Hissam, G.A. Moreno, Statistical-based WCET estimation and validation, in: the 9th International Workshop on Worst-Case Execution Time (WCET) Analysis, 2009.

[4] L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzetti, E. Quiñones, F.J. Cazorla, Measurement-based probabilistic timing analysis for multi-path programs, in: Euromicro Conference on Real-Time System (ECRTS-12), 2012.

[5] F.J. Cazorla, E. Quiñones, T. Vardanega, L. Cucu, B. Triquet, G. Bernat, E. Berger, J. Abella, F. Wartel, M. Houston, L. Santinelli, L. Kosmidis, C. Lo, D. Maxim, PROARTIS: Probabilistically analysable real-time systems, ACM TECS (2013).

[6] L. Kosmidis, J. Abella, F. Wartel, E. Quiñones, A. Colin, F. Cazorla, PUB path upper-bounding for measurement-based probabilistic timing analysis, in: Euromicro Conference on Real-Time Systems (ECRTS-14), 2014.

[7] PROXIMA, Probabilistic real-time control of mixed-criticality multicore and manycore systems, http://www.proxima-project.eu.

[8] J. Abella, C. Hernandez, E. Quinones, F. Cazorla, P. Conmy, M. Azkarate-askasua, J. Perez, E. Mezzetti, T. Vardanega, WCET analysis methods: Pitfalls and challenges on their trustworthiness, in: International Symposium on Industrial Embedded Systems (SIES), 2015, pp. 1–10, doi:10.1109/SIES.2015.7185039.

[9] R. Kirner, I. Wenzel, B. Rieder, P. Puschner, Using measurements as a complement to static worst-case execution time analysis, Intel. Syst. Serv. Mankind (2005).

[10] R. Kirner, P. Puschner, Obstacles in worst-case execution time analysis., in: 11th IEEE International Symposium on Object-oriented Real-time distributed Computing, 2008, pp. 333–339.

[11] E. Mezzetti, T. Vardanega, On the industrial fitness of WCET analysis, International Workshop On Worst-Case Execution Time Analysis (WCET 2011), 2011.

[12] Rapitime, http://www.RapitaSystems.com/RapiTime.

[13] S. Kotz, S. Nadarajah, Extreme Value Distributions: Theory and Applications, World Scientific, 2000.

[14] S. Coles, An Introduction to Statistical Modeling of Extreme Values, Springer, 2001.

[15] L. Santinelli, J. Morio, G. Dufour, D. Jacquemart, On the sustainability of the extreme value theory for WCET estimation, International Workshop on Worst–Case Execution Time Analysis (WCET), 2014.

[16] J. Abella, E. Quiñones, F. Wartel, T. Vardanega, F. Cazorla, Heart of gold: Making the improbable happen to extend coverage in probabilistic timing analysis, in: Euromicro Conference on Real-Time System (ECRTS-14), 2014.

[17] S. Milutinovic, J. Abella, F. Cazorla, Modelling probabilistic cache representativeness in the presence of arbitrary access patterns, in: Symposium on Object/Component/Service-oriented Real-time Distributed Computing (ISORC), 2016.

[18] J. Abella, F.J. Cazorla, E. Quiñones, T. Vardanega, Measurement-based probabilistic timing analysis and i.i.d property. White Paper., 2013. http://www.proartis-project.eu/publications/MBPTA-white-paper.

[19] M. Ziccardi, E. Mezzetti, T. Vardanega, J. Abella, F.J. Cazorla, EPC: extended path coverage for measurement-based probabilistic timing analysis, 36th IEEE Real–Time Systems Symposium (RTSS), 2015.

[20] J. Reineke, B. Wachter, S. Thesing, R. Wilhelm, I. Polian, J. Eisinger, B. Becker, A definition and classification of timing anomalies, International Workshop On Worst-Case Execution Time Analysis (WCET 2006), 2006.

[21] L. Kosmidis, T. Vardanega, J. Abella, E. Quiñones, F.J. Cazorla, Applying measurement-based probabilistic timing analysis to buffer resources, International Workshop On Worst-Case Execution Time Analysis (WCET), 2013.

[22] L. Kosmidis, E. Quiñones, J. Abella, T. Vardanega, F. Cazorla, Achieving timing composability with probabilistic timing analysis, Symposium on Object/Component/Service-oriented Real-time Distributed Computing (ISORC), 2013.

[23] J. Jalle, L. Kosmidis, J. Abella, E. Quiñones, F. Cazorla, Bus designs for time-probabilistic multicore processors, in: Design Automation and Test in Europe (DATE), 2014.

[24] M. Paolieri, E. Quiñones, F. Cazorla, G. Bernat, M. Valero, Hardware support for WCET analysis of hard real-time multicore systems, International Symposium on Computer Architecture (ISCA), 2009.

[25] M. Paolieri, E. Quiñones, F. Cazorla, M. Valero, An Analyzable Memory Controller for Hard Real-Time CMPs., Embedded System Letters (ESL), 2009.

[26] M. Slijepcevic, L. Kosmidis, J. Abella, E.Q. nones, F. Cazorla, Time-analysable non-partitioned shared caches for real-time multicore systems, in: Design Automation Conference (DAC), 2014.

[27] L. Kosmidis, C. Curtsinger, E. Quiñones, J. Abella, E. Berger, F. Cazorla, Probabilistic timing analysis on conventional cache designs, Design Automation and Test in Europe (DATE), 2013.

[28] L. Kosmidis, et al., Containing timing-related certification cost in automotive systems deploying complex hardware, in: Design Automation Conference (DAC). (Best Paper Award), 2014.

[29] P. Benedicte, L. Kosmidis, E. Quinones, J. Abella, F. Cazorla, A confidence assessment of WCET estimates for software time randomized caches, in: International Conference on Industrial Informatics (INDIN), 2016.

[30] http://www.gaisler.com/cms/index.php?option=com_content&task=view&id=13&Itemid=53, Leon3 Processor, Cobham Gaisler,

[31] L. Kosmidis, J. Abella, E. Quiñones, F. Cazorla, A cache design for probabilistically analysable real-time systems, Design Automation and Test in Europe (DATE), 2013.

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

[33] L. Kosmidis, J. Abella, E. Quiñones, F. Cazorla, Efficient cache designs for probabilistically analysable real-time systems, IEEE Trans. Comput. 63 (12) (2014).

[34] I. Agirre, M. Azkarate-Askasua, C. Hernandez, J. Abella, J. Perez, T. Vardanega, F. Cazorla, Iec-61508 sil 3 compliant pseudo-random number generators for probabilistic timing analysis, in: Digital System Design (DSD), 2015 Euromicro Conference on, 2015, pp. 677–684, doi:10.1109/DSD.2015.26.

[35] G. Box, D. Pierce, Distribution of residual autocorrelations in autoregressive-integrated moving average time series models, J. Am. Stat. Assoc. 65 (332) (1970) 1509–1526.

[36] J. Abella, J. del Castillo, M. Padilla, F. Cazorla, Extreme value theory in computer sciences: The case of embedded safety-critical systems, in: International Conference on Risk Analysis (ICRA), 2015.

[37] M. DeGroot, M. Schervish, Probability and Statistics, Addison-Wesley, Reading MA., 2002.

[38] F. Wartel, L. Kosmidis, C. Lo, B. Triquet, E. Quiñones, J. Abella, A. Gogonel, A. Baldovin, E. Mezzetti, L. Cucu, T. Vardanega, F. Cazorla, Measurement-based probabilistic timing analysis: Lessons from an integrated-modular avionics case study, International Symposium on Industrial Embedded Systems (SIES), 2013.

[39] J. Gustafsson, A. Ermedahl, Experiences from applying WCET analysis in industrial settings, Symposium on Object/Component/Service-Oriented Realtime Distributed Computing (ISORC), 2007.

[40] S. Altmeyer, R. Davis, On the correctness, optimality and precision of static probabilistic timing analysis, in: Design Automation and Test in Europe (DATE), 2014.

[41] R. Davis, L. Santinelli, S. Altmeyer, C. Maiza, L. Cucu-Grosjean, Analysis of probabilistic cache related pre-emption delays, in: Euromicro Conference on Real–Time System (ECRTS), 2013.

[42] J. Abella, D. Hardy, I. Puaut, E. Quiñones, F. Cazorla, On the comparison of deterministic and probabilistic WCET estimation techniques, in: Euromicro Conference on Real-Time System (ECRTS-14), 2014.

[43] Z. Stephenson, J. Abella, T. Vardanega, Supporting industrial use of probabilistic timing analysis with explicit argumentation, in: 11th IEEE International Conference on Industrial Informatics (INDIN), 2013.

[44] L. Kosmidis, J. Abella, E. Quiñones, F. Cazorla, Multi-level unified caches for probabilistically time analysable real-time systems, Real-Time Systems Symposium (RTSS), 2013.

[45] J. Reineke, Randomized caches considered harmful in hard real-time systems, Leibniz Trans. Embed. Syst. 1 (1) (2014) 03:1–03:13.

[46] L. Yue, I. Bate, T. Nolte, L. Cucu-Grosjean, A new way about using statistical analysis of worst-case execution times, in: ACM SIGBED Review, 2011.

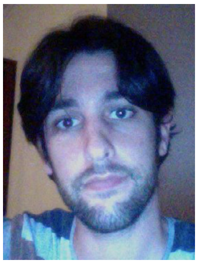
[47] F.J. Cazorla, T. Vardanega, E. Quiñones, J. Abella, Upper-bounding program execution time with extreme value theory, International Workshop On Worst-Case Execution Time Analysis (WCET), 2013.

Leonidas Kosmidis is a PhD. student in the CAOS group at Barcelona Supercomputing Center (BSC), Spain. Leonidas joined BSC in 2009 and received his MSc in Computer Architecture, Networks and Systems in 2011 from Universitat Politècnica de Catalunya (UPC), Spain. He holds a BSc in Computer Science from University of Crete, Greece and worked as an intern at ARM Cambridge and École Centrale Paris. Leonidas participates in FP7 PROARTIS and PROXIMA projects. His main focus is hardware and software design for real-time embedded systems.

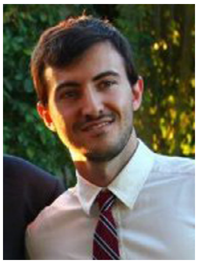
Eduardo Quiñones is a senior PhD. Researcher at BSC and member of HiPEAC. He received his MS degree in 2003 and his PhD. in 2008 at the UPC. His area of expertise is in safety-critical systems and high performance compiler techniques. He is involved in several FP7 European projects (parMERASA, PROARTIS, PROXIMA and P-SOCRATES), as well as some bilateral ESA-BSC projects. He spent one year as an intern at Intel Research Labs (2002 – 2003).

Jaume Abella is a senior PhD. Researcher in the CAOS group at BSC and member of HIPEAC. He received his MS (2002) and PhD. (2005) degrees from the UPC. He worked at the Intel Barcelona Research Center (2005–2009) in the design and modeling of circuits and microarchitectures for fault-tolerance and low power, and memory hierarchies. He joined the BSC in 2009 where he is in charge of hardware designs for FP7 PROARTIS and PROXIMA, and BSC tasks in ARTEMIS VeTeSS. Jaume is also involved in two ESA-BSC bilateral projects and FP7 parMERASA. He has authored more than 15 patents and 60 papers in top conferences and journals in the area. He is (has been) co-advisor of ten MS and PhD students.

Tullio Vardanega currently is at the University of Padua, Italy. He holds a MSc in Computer Science from the University of Pisa, Italy, and a PhD in Computer Science from the Technical University of Delft, Netherlands. He worked at European Space Agency (ESA) from July 1991 to December 2001. At the University of Padua, he teaches and leads research in the areas of high-integrity distributed real-time systems and advanced software engineering methods. He has a vast network of national and international research collaborations. He has co-authored 90+ refereed papers and held organizational roles in several international events and bodies, for ESA, the European Commission, ISO, IEEE and Ada-Europe.



Carles Hernandez received the M.S. degree in telecommunications and PhD in computer sciences from Universitat Politcnica de Valncia, in 2006 and 2012, respectively. He is currently senior PhD. Researcher at the Barcelona Supercomputing Center. His area of expertise includes network-on chip and reliability-aware processor design. He participates (has participated) in NaNoC, parMERASA, PROXIMA IP7 and VeTeSS ARTEMIS projects. In the context of PROXIMA he is in charge of probabilistic technology developments in the Leon3 processor. In 2012 he was intern at Intel Mobile Communications Munich.



Andrea Gianarro is a hardware engineer at Cobham Gaisler experienced in digital development on FPGA (Xilinx, Altera), Computer Science graduate with honors. Proficiency in VHDL programming with a focus on system-level integration and hardware architectures. Experienced in on-chip and off-chip interconnects: AMBA AHB, APB, CAN, RapidIO, Ethernet, SGMII, DDR2.

Ian Broster is a founder and director of Rapita Systems Ltd. He earned his PhD in 2003 at the Real-time Systems group of University of York for work on the timing analysis of realtime communication. He has been involved in significant real-time research projects involving scheduling analysis, predictable multi-core, real-time communication and fault tolerance. He has a valuable mix of academic and industrial experience of real-time and embedded systems. He is actively involved in the transfer of research technologies to practical industrial uses in the domain of reliable embedded systems.

Francisco J. Cazorla is the leader of the CAOS group at BSC and member of HIPEAC Network of Excellence. He has led projects funded by industry (IBM and Sun Microsystems), by the European Space Agency (ESA) and public-funded projects (FP7 PROARTIS project and FP7 PROXIMA project). He has participated in FP6 (SARC) and FP7 Projects (MERASA, VeTeSS, parMERASA). His research area focuses on multithreaded for both high-performance and real-time systems on which he is co-advising several PhD theses. He has co-authored 3 patents and over 100 papers in international refereed conferences and journals.