Measurement-Based Probabilistic Timing Analysis and Its Impact on Processor Architecture

Leonidas Kosmidis*†, Eduardo Quiñones,†, Jaume Abella†, Tullio Vardanega‡, Ian Broster§, Francisco J. Cazorla†¶
*Universitat Politecnica de Catalunya
†Barcelona Supercomputing Center
‡University of Padova
§Rapita Systems LTD
¶ Spanish National Research Council (IIIA-CSIC)

***Abstract*—Critical Real-Time Embedded Systems (CRTES) industry needs increasingly complex hardware to attain the performance/cost ratio required to keep competitive edge in the market. Worst-case execution time (WCET) analysis is central to CRTES development. Whereas current timing analysis techniques are sound, their viability is hampered by the soaring cost of acquiring detailed knowledge of the internal operation and state of the system, at both software and hardware level. This is a major hurdle to using them for increasingly complex hardware platforms. Measurement-Based Probabilistic Timing Analysis (PTA) reduces the cost of acquiring the knowledge needed for computing trustworthy WCET bounds. This paper presents the changes required to hardware design to facilitate the use of the PTA techniques.**

***Keywords*-worst-case execution time; processor architecture; cache memories; probabilistic analysis; time randomization**

I. Introduction

The market for Critical Real-Time Embedded Systems (CRTES), which the automotive and avionics sectors dominate in production volume, 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 more guaranteed computation 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 is known to complicate the computation of trustworthy and tight timing bounds[2]. Worst-Case Execution Time (WCET) analysis is an essential stage in the design and verification of real-time systems in general, especially so for CRTES. WCET bounds are needed for schedulability analysis to ascertain whether tasks can complete within their assigned deadlines under all conditions. Static timing analysis techniques [2] rely on the construction of a cycle-accurate model of the processor and on the code of the application. As the analysis cannot carry forward *all* the possible states of execution, conservative choices are made during the process, thus trading a reduction in the state space for increased pessimism [3], [4], [5]. The rise in complexity of next-generation CRTES greatly exacerbates this problem: the mass 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 this type of analysis for all critical applications in industrial systems. Measurement-based analysis techniques attempt to reduce costs by resorting to extensive testing on the real hardware, and on the recording of the so-called high watermark, i.e. the longest observed execution time. As there is some risk of failing to exercise the worst-case timing during testing, users add an engineering margin to make safety allowances for the unknown, trying to strike some balance between pessimistic overkill and risk of underestimation. This is especially risky when the system may exhibit discontinuous changes in timing due to pathological access patterns to hardware resources or other anomalous timing behaviour [6].

The same motivations outlined above pushes CRTES industry to venture considering mixed-criticality solutions to their system design, so that even more functional value can be obtained per unit of product. From the timing perspective, which is the focus of this paper, the risk of approaching mixed-criticality systems with conventional techniques is that strict mechanisms need to be put in place to ensure temporal segregation as a way to achieve composability in the time dimension. Evidently, this tenet contradicts the very intent of integrating mixed-criticality programs in one and the same system. Temporal segregation realised by static solutions, in the absence of effective means to remove or significantly attenuate the pessimism intrinsic in the very notion of static worst-case timing analysis, entails over-provisioning, which defeats the purpose.

Probabilistic techniques may greatly aid in this regard. In particular, with Measurement-Based Probabilistic Timing Analysis (MBPTA) methods [7], [8], [9], [10], 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

[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 trustworthy bound is a bound that can be supported by strong arguments and proofs.

or pWCET. As a consequence, even if any pWCET may in principle be exceeded, this can only happen with a given probability, which can be determined at a level low enough for the application domain; for example, in the region of 10^{-15} per hour of operation, largely below the acceptable probability of failure in certified systems. Under MBPTA, fixed a given level of granularity of execution, the response time of each execution component (e.g an instruction) at that level is assigned a distinct probability of occurrence. This trait is described by a probabilistic Execution Time Profile (ETP), which is expressed by the pair: <timing vector; probability vector>. The timing vector in the ETP of an execution component enumerates all its possible response times. For each response time in the timing vector, the probability vector lists the associated probability of occurrence. Hence, for execution component $\mathcal{C}_i$ we have $ETP(\mathcal{C}_i) = <\vec{t_i}, \vec{p_i}>$ where $\vec{t_i} = (t_i^1, t_i^2, ..., t_i^{N_i})$ and $\vec{p_i} = (p_i^1, p_i^2, ..., p_i^{N_i})$, with $\sum_{j=1}^{N_i} p_i^{N_j} = 1$.

The processor architecture may facilitate ensuring that individual instructions have an associated ETP. As this feature in turn enables a sound and effective application of MBPTA, this is the level of execution granularity at which we concentrate in this work.

Contribution. In this paper, we describe the architecture features that a processor should possess to guarantee MBPTA fitness by construction (also termed *PTA-friendliness* in this paper) and we also offer insight on the costs that may be incurred in actual implementation. To that end we categorise processor resources according to their timing behaviour and detail how they should be designed so that they can be used in a MBPTA-friendly processor. Without loss of generality we consider the inner operation of the processor to employ a number of passive resources (e.g. cache, 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 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.

The rest of the paper is organised as follows. Section II describes MBPTA requirements on hardware design. Section III classifies and analyses hardware resources in relation with MBPTA. Section IV presents a case study of a MBPTA-friendly processor architecture. Section V discusses the static variant of PTA, called SPTA for Static Probabilistic Timing Analysis. Section VI reviews related work. Section VII draws some conclusions. e

II. MBPTA Requirements on Hardware Design

PTA as understood in this paper, can be applied in either a static (SPTA) [9] or measurement-based (MBPTA) [8] fashion. In this work we focus on the measurement-based variant of PTA, and discuss SPTA in Section V. We refer the reader to those citations for the basics of PTA.

MBPTA considers events resulting from the observation of end-to-end measurement runs of the program, thus at coarser granularity than processor instructions. MBPTA uses a statistical method called Extreme Value Theory (EVT) [11][8] to compute pWCET estimates and therefore needs the observed execution times to describe *probabilistically independent* events. EVT requires that the observed execution times of program runs (of the same path) have a distinct probability of occurrence and can be modelled with independent random variables. The MBPTA method described in this paper also causes those variables to be identically distributed, so that MBPTA treats i.i.d. random variables[3]. MBPTA places strong requirements on how the observations are made (which of the many possible runs are considered) and what they can observe of all possible outcomes (which execution paths they cover).

The axiomatic existence of an ETP per *dynamic* instruction (by which we mean 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, that is every program run has an associated ETP, which is a sufficient and necessary condition to achieve the prerequisite i.i.d. execution time behaviour [12].

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

III. Probabilistically Modelling the Timing Behaviour of Processor Resources

In order to enable the use of MBPTA, 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 architecture to be MBPTA-friendly, the pWCET estimates obtained for the programs that run on

[3]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. Unfortunately, these properties are not met by current processors due to their state-sensitive nature.

it must hold valid for the whole operational life of the system, hence for every run of the programs of interest under all execution conditions. To understand how the timing behaviour of processor resources needs to be modelled for those guarantees to be obtained, we first need to appreciate how the MBPTA process works.

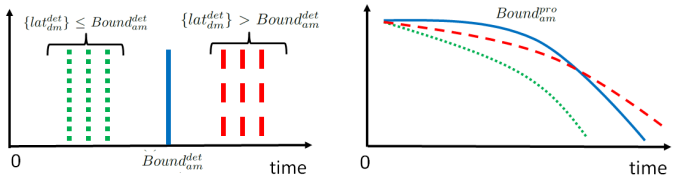
A. Probabilistic Timing analysis process

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 any circumstance 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 deployment mode is used during actual operation. In this mode, timing conditions are unrestricted 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 deployment mode. For MBPTA-friendly processor architectures, this condition can be achieved in either a deterministic or a probabilistic manner. Accordingly, any pWCET estimate obtained by analysis is a trustworthy 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.

Figure 1 provides a schematic view of the meaning of (a) deterministic upper-bounding and (b) probabilistic upper-bounding. In both sides of the figure, 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 Figure 1(a), the solid vertical line represents the analysis-mode bound (*am*), $Bound^{det}_{am}$ for the latency of a component. If in the deployment-mode (*dm*), the actual latencies, $\{lat^{det}_{dm}\}$, are below $Bound^{det}_{am}$, which is shown with the dotted lines, then the obtained bound is trustworthy. If it cannot be ensured that this is the case, the deployment-time actual latencies (dashed lines) can be bigger than the analysis-mode bound $\{lat^{det}_{dm}\} > Bound^{det}_{am}$, hence the bound is not trustworthy and cannot be used. In Figure 1(b) the solid curve represents the analysis-mode upper-bound ETP of the latency of the resource, $Bound^{pro}_{am}$. 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



(a) deterministic-latency resource (b) probabilistic-latency resource

Figure 1. Deterministic and probabilistic upper-bounding latencies

upper-bounded by $Bound^{pro}_{am}$ (solid line). However, if actual latencies match those described by the dashed curve, they are not probabilistically upper-bounded by $Bound^{pro}_{am}$.

B. Taxonomy of hardware resources

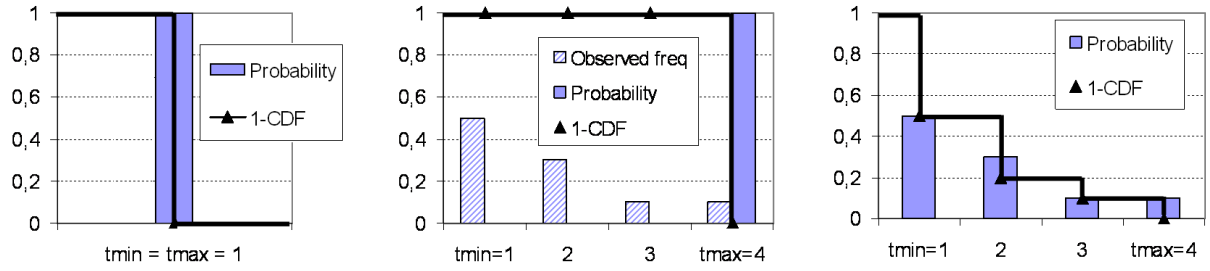
We term *jitterless resources* the processor resources that have a fixed latency, independent of the input request and of the past history of service. several hardware resources in current processor architectures are jitterless. Jitterless resources are easy to model for all types of static timing analysis. For MBPTA techniques, the ETP of a jitterless resource jl is given by: $ETP_{jl} =< (l), (1.0) >$, where l is the latency of the resource. Its PDF is shown in Figure 2(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 the access 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: for a processor, data are usually encoded in the instruction that issues the request, or stored in its input registers.

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 – as long as timing anomalies can be excluded [13] – 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 $ETP_{r_{wl}} =< (l_{max}), (1.0) >$, where l_{max} is the worst-case latency of the resource. An example of the impact of such upper bounding is shown in Figure 2(b). In the example, the



(a) Jitterless resource (b) Jittery upper-bounded resource (c) Jittery time-randomised resource

Figure 2. Probabilistic timing behaviour of a single instruction for each type of resource

actual probabilities for each latency are unknown; only frequencies can be obtained; upper bounding therefore is needed. Conversely, the ETP of a time-randomised jittery resource r_j is: $ETP_{r_j} =< (l_j^1, l_j^2 ..., l_j^k), (p_j^1, p_j^2, ...p_j^k) >$ 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 Figure 2(c).

C. Assigning ETP to individual processor resources

The probability for a given latency is different from its frequency. This is best shown by an example. 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, if for a deterministic stateful resource R_2 with latency $\vec{t_2}= (t_2^1, t_2^2)$ 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 a true 50% probability. This is so because for events that are strictly dependent on the history of execution, information on past events *cannot* be used to provide guarantees about the appearance of future events.

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.

D. 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 $ETP_{pc} = f\left(ETP_1, ETP_2, ..., ETP_n\right)$, where ETP_i is the probabilistic execution time of resource r_i.

Sequential composition: sequential composition of ETP, $f_s\left(ETP_1, ETP_2, ..., ETP_n\right)$, leads to an ETP where latencies and probabilities are determined by the type of dependence across the input ETP (whether systematic or probabilistic, as shown in [12]). Sequential composition as intended here is architectural and not mathematical, hence different from the convolution used in the context of SPTA for combining ETPs of static instructions (e.g. instructions in the object code of the program).

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

The key appreciation 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.

Parallel composition: processor resources may also be arranged in parallel. Examples of parallel resources are some particular designs of cache memories and 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 ($f_p\left(ETP_1, ETP_2, ..., ETP_n\right)$) 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 $ETP_1 =< (1, 4), (0.4, 0.6) >$ and $ETP_2 =< (2, 3), (0.3, 0.7) >$ respectively. The ETP from their parallel composition, $ETP_{1+2} = f_p\left(ETP_1, ETP_2\right)$, is $ETP_{1+2} =< (2, 3, 4), (0.12, 0.28, 0.6) >$.

E. 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 III-B. This is the case of resource buffers, also known as 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* [14]. 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-friendly: it suffices that they either do not introduce jitter on their own (hence they are fixed-latency or else just jitter propagators) or their jitter can be upper-bounded or else it can be randomised. In that manner, initial conditions no longer need to be accounted for as sources of execution time variation [15], and trustworthy pWCET estimates can be obtained. Those pWCET bounds can also be tight because MBPTA-friendly architectures cause the build-up effects of abrupt pathologically large variations in execution time (for the same random variable) to naturally smooth out so that pessimism can be much reduced.

F. First steps towards MBPTA-friendly multicores

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[4]. 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 fetched 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 tasks run in parallel, the execution time of one instruction i_x^{T1} belonging to task T_1 may be affected by any other instruction i_y^{T2} from task T_2. If there is precedence ordering in task execution such that T_2 executes after T_1, then the inter-task interference generated by i_y^{T2} does not affect i_x^{T1}. If no precedence ordering can be asserted instead, T_1 and T_2 can execute in any order. Hence they may execute in parallel on different cores, so that i_y^{T2} may cause inter-task interference on i_x^{T1}. It is evident that we cannot conceivably capture the effect that any single instruction of any task i_j^{Tk} may have on any other instruction of any other task i_l^{Tm} in the system. Should this be required, MBPTA would become intractable. To prevent this, the design of MBPTA-friendly multicores must ensure that the worst effect that one task can have on the execution of any other task due to inter-task interference can be probabilistically bounded.

Interestingly, the MBPTA-friendly 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.

- **Shared bus**. The authors of [17] 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 [18].
- **Shared memory controller**. The same cited work [17] shows that the latency of a shared memory controller can be upper bounded, which is fine for MBPTA-friendliness. Again, that measure is in line with findings for time-deterministic systems [19].
- **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 [18]. 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 CPU, the solutions devised for single-core processors.
  An alternative approach has been put forward in [20], 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.

IV. Case Study

A. Designing a MBPTA-friendly processor architecture

We use a 4-stage pipelined core architecture as depicted in Figure 3. The four stages operate as follows: Instructions are fetched in-order from the instruction cache (IC) into the processor. The ITLB is accessed in parallel to translate pages. Instructions are decoded in one cycle. Decoded instructions

[4]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 [16].

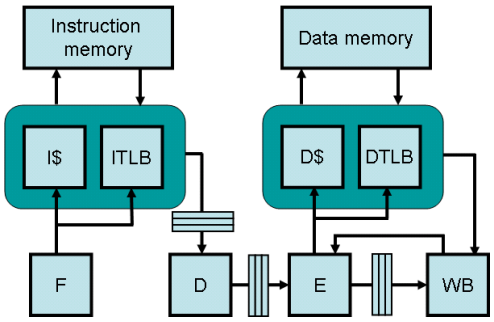


Figure 3. Reference core architecture.

Table I
INDEPENDENCE AND IDENTICAL DISTRIBUTION TEST RESULTS (2ND AND 3RD COLUMNS), AND AVERAGE EXECUTION TIME AND PWCET BOUNDS OF THE COMPLEX MBPTA-FRIENDLY PROCESSOR VS. AN EQUIVALENT CONVENTIONAL PROCESSOR (4TH AND 5TH COLUMNS)

Benchmarks	Statistical tests		Timing analysis results	
	Inde-pence	Identical distribution	Average Exec. Time	pWCET 10^{-16}
a2time	0.949	0.387	1.04	1.13
aifirf	0.380	0.791	1.09	1.12
cacheb	0.063	0.668	1.14	1.45
canrdr	0.126	0.529	1.01	1.04
puwmod	0.253	0.993	1.00	1.01
rspeed	0.635	0.628	1.00	1.04
tblook	0.127	0.252	0.87	0.94
ttsprk	0.696	0.187	1.13	1.25

are issued in-order to the execute stage. Instructions are executed in one cycle, except for memory operations. Read operations access the data cache (DC) during this cycle, blocking the execution of further instructions until data are fetched. The DTLB is accessed in parallel with the DC. The instructions commit in one cycle. Write operations update the data cache during this cycle.

IC and DC are 16 KB in size, 8-way set-associative, with 16 bytes per line. The DC and IC use the random placement and replacement policies presented in [21]. The DC is write-through, so that all store instructions are propagated to memory. The DC is write-allocate, hence data are fetched on a store miss. ITLB and DTLB are 8-entry fully-associative random-replacement, with 1 KB page size, and their misses are handled by a hardware page-walker. The hit and miss latencies are 1 and 80 cycles respectively. The TLB are accessed in parallel with the caches so that instructions are stalled in the corresponding stage until both the cache *and* the TLB can serve the request.

B. Deriving ETP

In the example architecture, there are two main sources of execution time variability: TLB and caches, which inject random events in the execution time of a program.

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_{IC} and ETP_{ITLB} respectively) are composed in parallel, and memory latency, which is accessed in case of a miss and whose ETP (ETP_{IM}) 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`): $ETP_{fend} = f_s\left(f_p\left(ETP_{IC}, ETP_{ITLB}\right), ETP_{IM}\right)$. 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 for core operations, ETP_{exec}, the ETP of the buffer after execution, ETP_{buf2}, and the ETP of the write-back stage, ETP_{wb}. While ETP_{exec} and ETP_{wb} have the form $< (l), (1.0) >$, 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 [14]. Since all those actions occur sequentially, the ETP for core operations is as follows:

$$ETP_{core} = f_s\left(ETP_{fend}, ETP_{buf1}, ETP_{exec}, ETP_{buf2}, ETP_{wb}\right) \quad (1)$$

Memory operations have the same front-end ETP as core operations. The back-end latency, instead of depending on ETP_{exec}, 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: $ETP_{dmpath} = f_s\left(f_p\left(ETP_{DC}, , ETP_{DTLB}\right), ETP_{DM}\right)$. Therefore, the ETP for memory operations is as follows:

$$ETP_{mem} = f_s\left(ETP_{fend}, ETP_{buf1}, ETP_{dmpath}, ETP_{buf2}, ETP_{wb}\right) \quad (2)$$

C. 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 [22] on the processor architecture.

For independence we use the Wald-Wolfowitz (WW) test [23]. For identical distribution hypothesis we use the Kolmogorov-Smirnov (KS) goodness-of-fit test [24]. We use a 5% significance level (a typical value for this type of tests), whereby absolute values obtained with the WW test must be below 1.96 to prove independence, and the outcome of the KS tests should be above the threshold (0.05) to assert identical distribution.

For each benchmark, less than $1,000$ runs were needed for each program, in line with previous experience [8], [21].

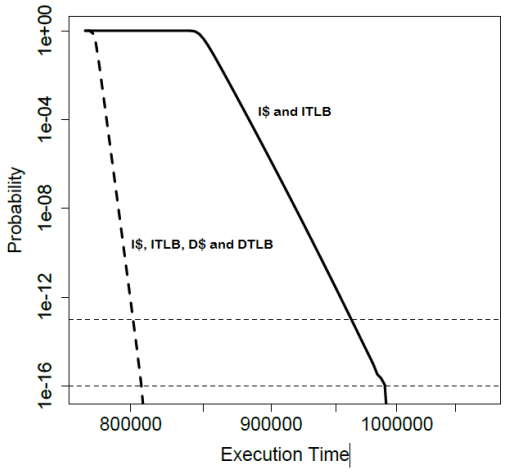


Figure 4. pWCET estimates for the *puwmod* benchmark program on different architectural setups.

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 I reports the results of both tests for all benchmarks. Both tests are passed in all cases, which proves that the example architecture meets the i.i.d. requirement of our MBPTA approach.

D. 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 [8]. The dashed line in Figure 4 plots the pWCET estimates obtained for the *puwmod* benchmark program of the EEMBC suite, run on the example architecture. The solid line plots the result from running the same program on the example architecture *without* the DC and the DTLB. 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^{-13} and 10^{-16} 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. To sustain this claim, we observe that the dotted line in Figure 4 plots the pWCET estimates obtained after reinstating the DC and the DTLB in the processor architecture. Interestingly, the MBPTA process stays unchanged in procedure and effort, while the pWCET estimates become considerably smaller (around 20% in the specific experiment) owing to the performance boost of using the DC and the DTLB.

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. Measurement-based timing analysis 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.

E. MBPTA-friendly 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 I reports average execution times and pWCET estimates for an exceedance threshold of 10^{-16} 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. The average execution time of the MBPTA-friendly architecture is only 4% worse than the time-deterministic alternative, thus showing that time randomisation fares well even in the average case. Even more interesting, pWCET estimates are, on average, only 12% higher than the average performance obtained for a processor architecture implementing LRU as the replacement policy for all caches. Whereas the WCET values for those programs on the time-deterministic architecture are not available (computing them would require the port 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 [25]. Our study shows that, for our MBPTA-friendly processor architecture, the observed inflation was up to 45% for `cacheb`, which

allows arguing that MBPTA-friendly processors are viable for CRTES industry.

In concluding our outline of the application of MBPTA to our example processor it is worth noting that the ETPs for individual dynamic instructions in our processor are non-independent across them [12]. This is so 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 are captured in the observations taken at analysis time and hence are accounted for in the pWCET estimate derived with MBPTA.

V. SPTA REQUIREMENTS

The static variant of PTA, called SPTA, uses a model of the processor to derive a-priori probabilities for the latency of each processor instruction in the execution of the program, i.e. one ETP per (static) instruction in the assumed control-flow path.

It it worth noting that the ETP of an instruction in the program, i.e. static instruction, encapsulates the timing behaviour of all the instances of that static instructions executed at run time, called dynamic instructions. In that respect, the ETP has to trustworthily upper-bound the behaviour of all such instances. The level of abstraction is, hence, different for the ETP described for MBPTA that stays at dynamic instruction level.

Further, SPTA requires that the timing probability distribution captured by the ETP of the instruction is fully independent of execution history: the ETP of that instruction should therefore hold constant (i) for all executions of it across all control-flow paths that lead to it, and (ii) regardless of the outcome of previous instructions [9].

This is not often the case due to the dependence on execution history intrinsic in the cache state, and thus in the hit/miss probability of the ETP of memory operations. However, since SPTA is intended to produce an upper bound distribution for the execution time of the program, the sought ETP can be replaced by another ETP that upper bounds[5] the ETP of that instruction for all execution paths that execute it and all combinations of outcomes for the random events triggered by the preceding instructions in those paths. The upper-bound ETP provides the desired independence for the purposes of WCET estimation.

Classic static timing analysis techniques (STA) [2] operate on deterministic processor architectures, which require knowledge of all the factors that influence *execution history*, where that affects timing behaviour. At cache level, for instance, STA accuracy in predicting the hit/miss outcome of a given memory access on a cache model with Least Recently Used (LRU) replacement policy, depends on the knowledge of the addresses, or their alignment in cache, of all previous memory accesses made by the program up to the point of interest. This knowledge allows building an accurate abstract representation of the cache state. Any reduction in that knowledge (when e.g. the addresses of some memory accesses are unknown), leads to a distinct degradation of the tightness of the WCET estimate [26].

In the specific case, if provided with all the required information about the hardware and software state, arguably STA computes the tightest estimates that can be obtained of a program's WCET. PTA techniques, including SPTA, therefore provide a less tight WCET estimate if all state information about the system can be tracked. PTA techniques focus instead on reducing the information needed for computing tight-enough WCET estimates, where 'enough' can be graded in accord with domain requirements. This is advantageous for the industrial user because for the ever increasingly complex processor architectures, the cost of acquiring and modelling detailed knowledge about the hardware and software internals is exceedingly high.

Recent work [27][26] has shown that (i) SPTA does not require less information than STA for fully associative caches with random replacement; (ii) SPTA has not been shown to work with random placement yet; and (iii) with modulo placement, SPTA does not require less state information than STA. Oppositely, MBPTA works for random placement caches and requires no information about the addresses or the alignment of cache accesses. Overall, MBPTA significantly reduces information required about execution history and is able to analyse architectures much more complex than current SPTA methods.

VI. 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 static timing analysis (STA). The other, more recent, but rather vibrant, considers the various flavours of probabilistic timing analysis (PTA). The state of the art in STA is comprehensively surveyed in [28], so we omit discussing it here.

At hardware level, random placement was proposed in [21] to enable the use of set-associative caches for MBPTA. [27], [29] discuss the trustworthiness of pWCET estimates obtained with MBPTA on top of random placement caches. The latter work [29] conducts a thorough analysis of especially thorny cases with the use of MBPTA and proposes ways to address them.

MBPTA has been used in the context of time-randomised architectures for single-path programs [8][30] and multi-path programs [10]. In [31], the authors use EVT for for time-deterministic architectures to derive execution time bounds.

[5] An ETP_A upper bounds another ETP_B if and only if the latency of ETP_A is equal or higher than the latency of ETP_B for any cumulative probability. E.g., if $ETP_A = \{1, 2\}\{0.5, 0.5\}$ and $ETP_B = \{1, 2\}\{0.6, 0.4\}$, ETP_A upper bounds ETP_B.

[15] discusses the advantages of using EVT in the context of time-randomised architectures, that is MBPTA, in comparison to applying EVT to time-deterministic architectures.

SPTA techniques have been studied for relatively simple processor architectures that use time-randomised caches [9], [32], [16]. [26] presents the first comprehensive comparison among STA, SPTA and MBPTA techniques.

VII. Conclusions and Future Work

In this paper we have shown that in order for the measurement-based variant of probabilistic timing analysis (MBPTA) to be usable economically and assuredly, the target processors should be designed such that every program instruction have a distinct probabilistic Execution Time Profile (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 overprovisioning 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 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 TIN2012-34557 and the HiPEAC Network of Excellence. Leonidas Kosmidis is funded by the Spanish Ministry of Education under FPU grant AP2010-4208. The authors acknowledge Michael Houston, Liliana Cucu-Grosjean and Luca Santinelli for their take in the initial ideas behind 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, and P. Stenström, "The worst-case execution-time problem overview of methods and survey of tools," *ACM Transactions on Embedded Computing Systems*, vol. 7, pp. 1–53, May 2008.

[3] R. Kirner, I. Wenzel, B. Rieder, and P. Puschner, "Using measurements as a complement to static worst-case execution time analysis," *Intelligent Systems at the Service of Mankind*, 2005.

[4] R. Kirner and P. Puschner, "Obstacles in Worst-Case execution time analysis." *11th IEEE International Symposium on Object-oriented Real-time distributed Computing*, pp. 333–339, 2008.

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

[6] G. Bernat, A. Colin, and J. Esteves, "Considerations on the LEON cache effects on the timing analysis of on-board applications," in *Proceedings of the Data Systems in Aerospace Conference (DASIA)*, 2007.

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

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

[9] 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, and D. Maxim, "PROARTIS: Probabilistically analysable real-time systems," *ACM TECS*, 2013.

[10] L. Kosmidis, J. Abella, F. Wartel, E. Quiñones, A. Colin, and F. J. Cazorla, "PUB Path Upper-Bounding for Measurement-Based Probabilistic Timing Analysis," in *Euromicro Conference on Real-Time Systems (ECRTS-14)*, July 2014.

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

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

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

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

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

[16] R. I. Davis, L. Santinelli, S. Altmeyer, C. Maiza, and L. Cucu-Grosjean, "Analysis of probabilistic cache related pre-emption delays." in *Euromicro Conference on Real-Time System (ECRTS-13)*, 2013.

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

[18] M. Paolieri, E. Quiñones, F. J. Cazorla, G. Bernat, and M. Valero, "Hardware support for WCET analysis of hard real-time multicore systems," in *36th Annual International Symposium on Computer Architecture (ISCA-09)*, 2009.

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

[20] M. Slijepcevic, L. Kosmidis, J. Abella, E. Quiñones, and F. J. Cazorla, "Time-analysable non-partitioned shared caches for real-time multicore systems," in *51st Annual Design Automation Conference on Design Automation Conference (DAC'14)*, 2014.

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

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

[23] J. Bradley, *Distribution-Free Statistical Tests*. Prentice-Hall, 1968.

[24] M. DeGroot and M. Schervish, *Probability and statistics*. Addison-Wesley, Reading MA., 2002.

[25] J. Gustafsson and A. Ermedahl, "Experiences from applying WCET analysis in industrial settings," in *10th IEEE Computer Society Symposium on Object/Component/Service-Oriented Realtime Distributed Computing*, 2007.

[26] J. Abella, D. Hardy, I. Puaut, E. Quiñones, and F. J. Cazorla, "On the Comparison of Deterministic and Probabilistic WCET Estimation Techniques," in *Euromicro Conference on Real-Time System (ECRTS-14)*, July 2014.

[27] J. Reineke, "Randomized caches considered harmful in hard real-time systems," *Leibniz Transactions on Embedded Systems*, vol. 1, no. 1, pp. 03:1–03:13, 2014.

[28] 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, and P. Stenström, "The worst-case execution time problem: overview of methods and survey of tools," *Transactions on Embedded Computing Systems*, vol. 7, no. 3, pp. 1–53, 2008.

[29] J. Abella, E. Quiñones, F. Wartel, T. Vardanega, and F. J. Cazorla, "Heart of Gold: Making the Improbable Happen to Extend Coverage in Probabilistic Timing Analysis," in *Euromicro Conference on Real-Time System (ECRTS-14)*, July 2014.

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

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

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