D3.8 Final report on probabilistic analysis for mixed criticality on multicore
Version 1.0

Document Information

Contract Number	611085
Project Website	www.proxima-project.eu
Contractual Deadline	m36, 30-September-2016
Dissemination Level	PU
Nature	R
Authors	Liliana Cucu (leading author), Adriana Gogonel, Cristian Maxim, Iain Bate, Benjamin Lesage, David Griffin, Tullio Vardanega, Enrico Mezzetti, Jaume Abella, Francisco Cazorla
Contributors	All partners
Reviewer	BSC
Keywords	Measurement-Based, Probabilistic Timing Analysis, Multicore

Notices: *The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement n° 611085.*

Change Log

Version	Description of change
v1.0	Initial version released for project review

Executive Summary

This deliverable presents the final versions of the PROXIMA timing analyses. These timing analyses have been evolving since the beginning of the project up to advanced versions, all integrated today within Rapita tool RVS.
The PROXIMA timing analyses are the following:

- **EVT** method represents the utilization of the Extreme value theory within the PROXIMA project. This method is complementary to all methods within the project (EPC, pTC-VICI and FBI-VICI). The method has two main versions: one for independent execution times and a second one for dependent execution times. Each version provides a voting procedure between the two possible versions of the EVT to provide the pWCET estimate. Since D3.4 the main evolutions of the EVT method concerns its adaptability to the PROXIMA use cases. Indeed the statistical tests are requiring appropriate test values and during this period such values have been refined. Moreover the latest version captures the low variability in execution times that some programs may experience. While generally stable, its R implementation does not offer the robustness of an industrial utilization and moving to other implementations is considered as future work.

- **EPC** is the solution adopted in PROXIMA to cope with the problem of path representativeness in MBPTA. As it is generally not possible to guarantee that all paths have been triggered at analysis time, the Extended Path Coverage (EPC) aims at automatically generate synthetic observations to complement the set of measurements collected so that to obtain the same effect of full path coverage. Since D3.4, EPC have been ported from an early-stage prototype to a complete set of tools on top of the RVS tool chain. Extensions to the PROXIMA MBPTA tool chain and a set of specific tools have been realized in order to consolidate EPC into a well-defined end-to-end process. The current implementation still need some refinements and improvement especially on the infrastructure and exploitable structural information. Nonetheless, EPC has been tailored on the FPGA-HW Rand platform and methods to support EPC trace collection have been consolidated and integrated on top of the industrial-quality tool chain, either on bare-metal or PikeOS. On the FPGA-HWRand, EPC has been successfully applied to industrial case studies. Extension to SW Rand platforms and FPGA-SW Rand in particular seems to be near-at-hand.

- **pTC-VICI** is the method used to account for multicore contention on top of the FPGA-SWRand platform. This method builds upon two premises: (i) no assumption can be made on how requests from the different contenders will align in time and hence, whenever a request from another core can create contention, the worst contention for that request is assumed; (ii) pTC-VICI (Partially/Fully Time composable analysis-VICI) allows to budget for a limited number of requests of specific types (with the worst timing alignment) so that an appropriate timing bound is chosen at integration time once contenders are known, or simply the contention allowed by contenders is limited

(in terms of number and type of requests) and this is monitored with Performance Monitoring Counters (PMCs) during operation preventing contenders to exceed their allowed request budgets. The pTC model is fully built upon PMCs measurements so as to match industrial practice. The pTC model has been developed from scratch since m18 where it did not exist, and it was developed to cover the FPGA-SWRand platform, for which no solution was in place to use by the end users' case studies.

- **FBI-VICI** method is designed to model the effects on a task of a variable amount of interference generated by competing tasks running simultaneously on other cores. VICI analysis has been applied to synthetic benchmarks by academic partners to the AURIX and P4080 platforms. In addition, industrial partners have also applied the analysis to their codebases for these platforms, demonstrating the applicability of this approach. This approach is not restricted to the specific platforms for which it has been devised and integrated (P4080 and AURIX), and it can be potentially ported to other hardware platforms.

Contents

1 Introduction

The PROXIMA project has proposed modifications at both hardware and the software level with the goal of making all the sources of jitter (in the software's use of hardware resources) easier to observe and compliant with the assumptions of probabilistic and statistical analyses. These modifications are either done by randomizing the access to some resources (mainly memory), and / or by upper bounding the sources of jitter. This deliverable presents the PROXIMA timing analyses for determining the pWCET of a program on any of the PROXIMA architectures. The relationship between the architectures and the timing analyses is provided in Figure 1. A measurement protocol based on Bayesian reasoning has been also proposed (in green) but its utilization has been done only at research level. The figure illustrates the complementarity of the analysis, for example EVT we may note that all three timing analyses (EPC, pTC-VICI and FBI-VICI) are using EVT and all four platforms (FPGA with hardware and with software randomisation, AURIX, P4080 and Many Core simulator).

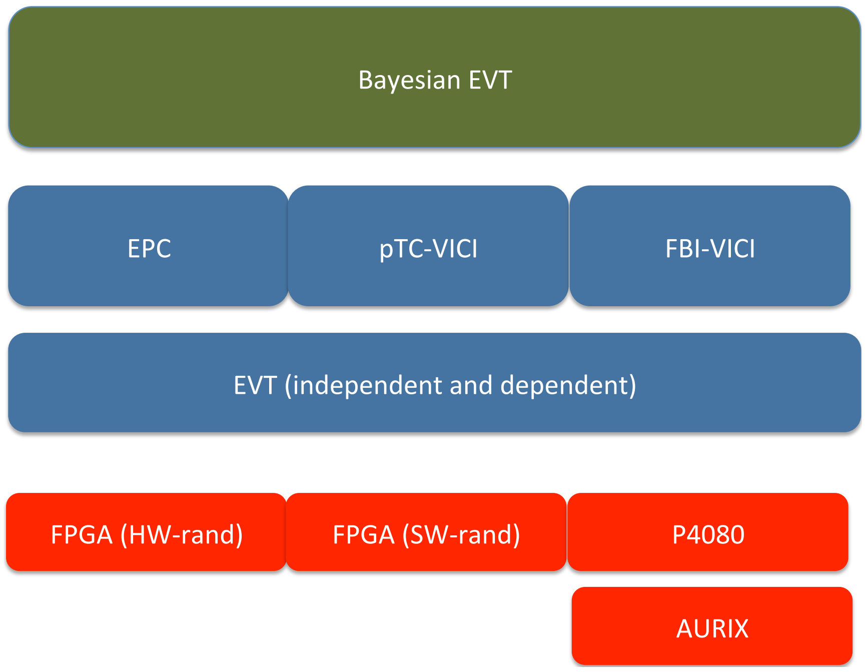


Figure 1: The complementarity of PROXIMA timing analyses (in blue) while covering all PROXIMA architectures (in red)

Organization of the deliverable The sections from Section 2 to Section 5 of this document provide a detailed description of the evolution together with the advantages and the limitations of each timing analysis provided previously in D3.4. Each analysis section starts with an executive summary, a description of the solution proposed by the technique and it ends with its evolution since D3.4 and the numerical evaluation of that technique on benchmarks. The RVS integration details of the tools are presented in D3.11 and the evaluation of the techniques on the PROXIMA use cases in D4.8 and D4.9.
The third section of this document provides a global view of the probabilistic ap-

proaches within the real-time community before and after the PROXIMA project. This view allows the reader to understand the scientific and industrial impact of our results and this measure may be completed by a description of the open problems that may be today appropriately formulated thanks to PROXIMA scientific and industrial achievements. The open problems are described in the fourth section of this deliverable that concludes the deliverable.

2 Extreme value theory - the PROXIMA basis

Participants: INRIA

2.1 Executive summary

In this section we present the utilization of the Extreme value theory within the PROXIMA project. This method is completementary to all methods within the project (EPC, pTC-VICI and FBI-VICI). The EVT method purpose is to provide a pWCET estimation from a set of execution times. Within its current form it guarantees a safe estimation with respect to the timing properties captured by the execution times. The method has two main vesions: one for independent execution times and a second one for dependent execution times. Each version provides a voting procedure between the two possible versions of the EVT to provide the pWCET estimate.
Since D3.4 the main evolutions of the EVT method concerns its adaptability to the PROXIMA use cases. Indeed the statistical tests are requiring appropriate test values and during this period such values have been refined. Moreover the latest version captures the low variability in execution times that some programs may experience.
Today the method is exploitable as tested on real industrial programs but its R implementation does not offer the robustness of an industrial utilization.

2.2 Description

Our measurement-based timing analysis is based on the utilization of the theory of extreme values (EVT) [20]. In this section we provide first an intuitive presentation of our results on pWCET estimation using EVT. We provide in Section 2.3 details of the main evolutions of our method since M18. These evolutions have been motivated by the specificity of the case studies execution times while our previous version of the method has been mainly tested on benchmarks. In Section 2.4 we enumerate the advantages and the current limitations of our method. We conclude in Section 2.5 with a numerical evaluation allowing to illustrate the independence related notions presented in Section 6.
According to EVT if the maximum of execution times of a program converges, then this maximum of the execution times $\mathcal{C}_i, \forall i \geq 1$ will converge to one of the three possible curves described in Figure 2: Fréchet, Weibull and Gumbel corresponding to a shape parameter $\xi < 0$, $\xi > 0$, and $\xi = 0$, respectively.
EVT has two different formulations: Generalized Extreme Value (GEV) and Gen-

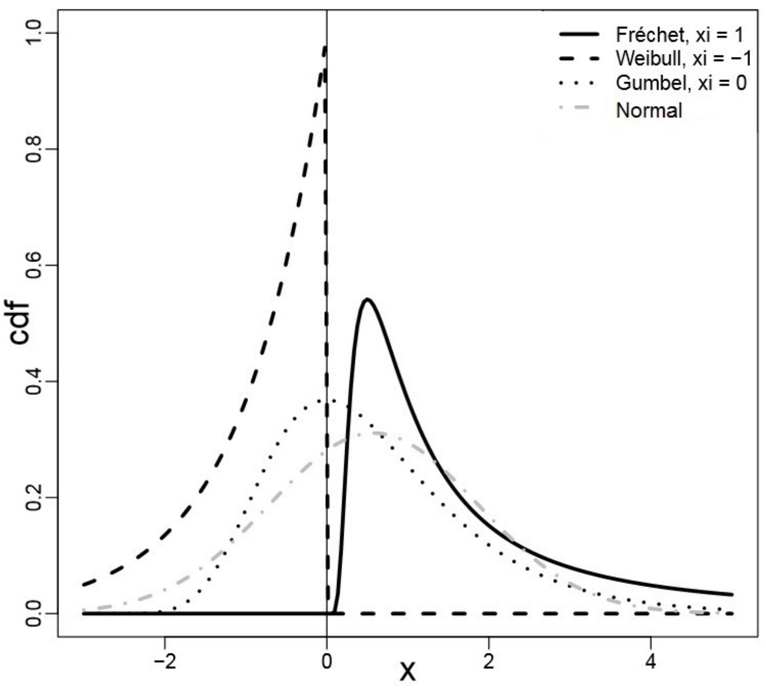


Figure 2: The three possible upper-bounds of a set of $\mathcal{C}_i, \forall i \geq 1$ of the same program

eralized Pareto Distribution (GPD). GEV is based on the block maxima reasoning that separates the execution times into smaller groups and only the largest value of each group is considered for the pWCET estimation (see Figure 3). GPD is a method based on the threshold approach that considers only the values larger than the chosen threshold for the pWCET estimation (see Figure 4).

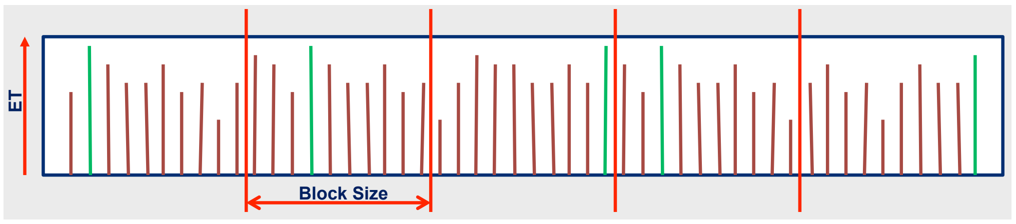


Figure 3: GEV keeps the largest value for each block

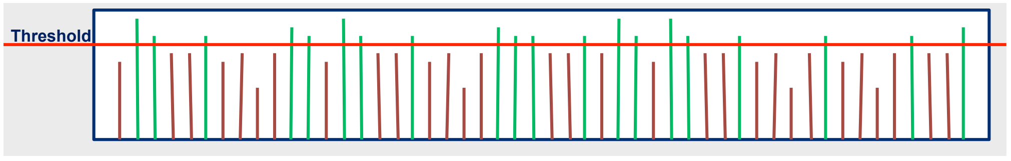


Figure 4: GPD keeps all values above the threshold

For similar reasons to previous works, e.g. [1,11,14,22], we consider that the upper-bound for the pWCET estimation is most often a Gumbel and the final pWCET estimate is calculated as a generalized EVT curve, a combination of Fréchet, Weibull and the identified Gumbel. With respect to previous results **our solution identifies the Gumbel closest to the function upper-bounding the execution times** and **our pWCET estimate is obtained twice by using separately**

GEV and GPD. The results of the two methods (GEV and GPD) are compared and the pWCET estimate is valid due to the fitting of two pWCET estimates (see Figure 5).

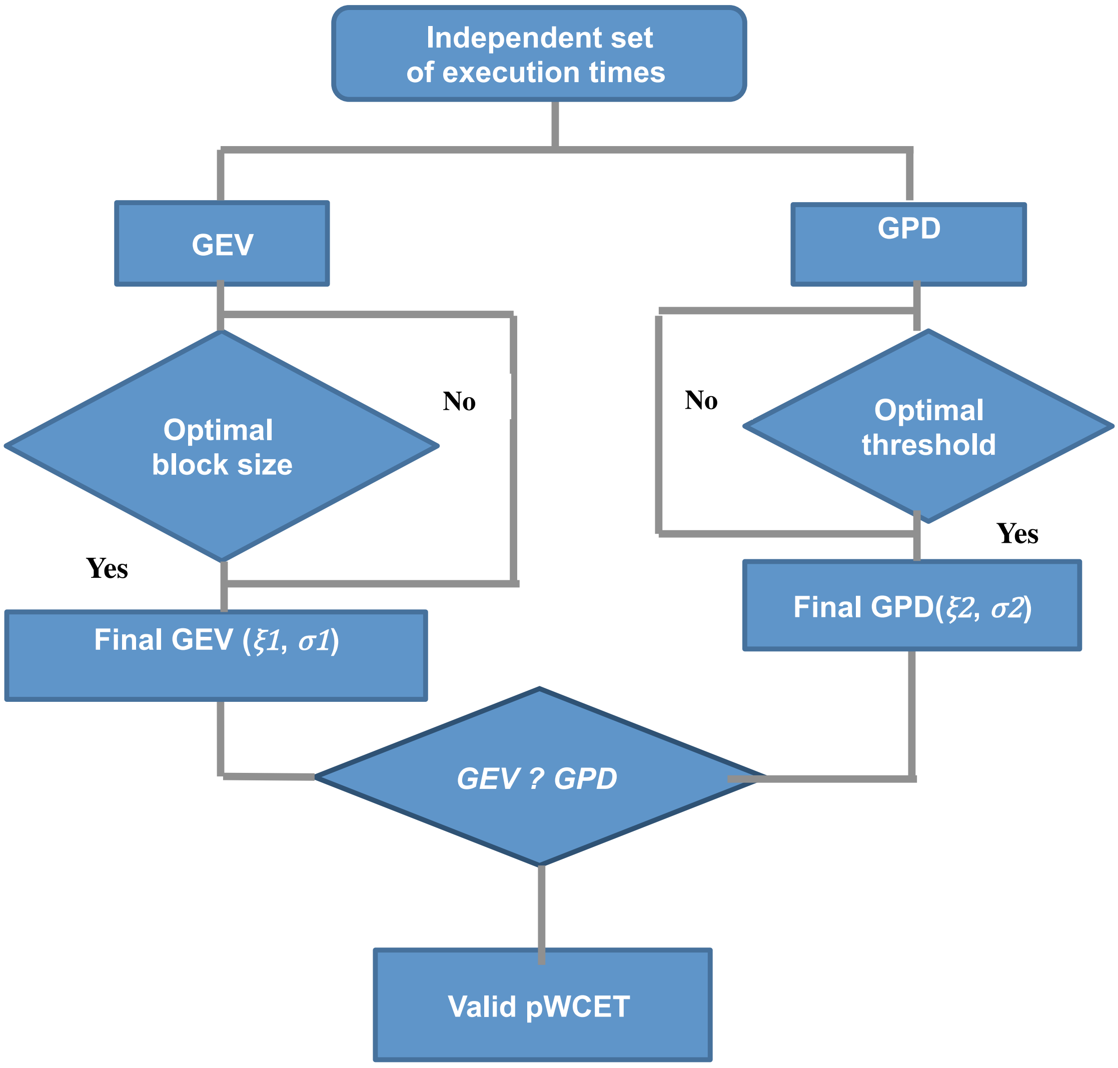


Figure 5: A global view of the pWCET estimation using GEV and GPD

2.3 Main evolutions since M18

1. **The case of execution times with small variability** In the case of execution times with small variability (see Figure 6) the pWCET estimation is modified with respect to the previous method. More precisely if the execution times are grouped in two or three sub-sets, then the pWCET is obtained as the joint pWCET estimate of GEV and GPD by choosing for each probability the smallest value between GEV and GPD. This choice is motivated by the fact that the previous version provides pessimistic pWCET estimation through its GPD branch. Indeed the existence of few sub-groups of possible values "force' GPD to keep in general the values from the largest value subset (sub-set 1 in Figure 6). GEV is not sensitive to this grouping effect and

the joint pWCET estimation would be mainly based on GPD if the choice of the largest value between GEV and GPD is kept.

For detecting this specific case we introduce a statistical test described below.

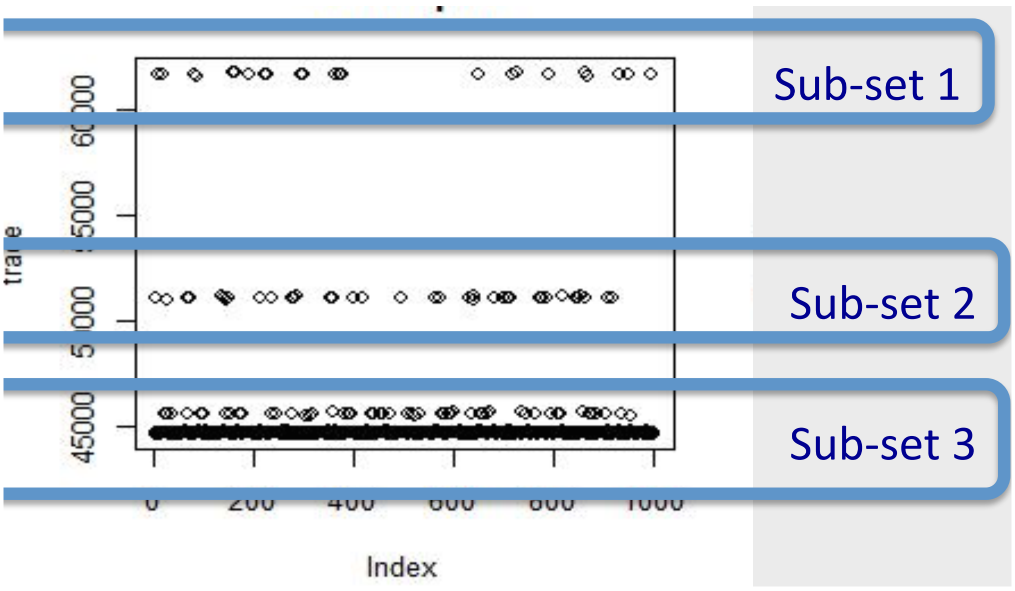


Figure 6: Set of execution times with small variability

We use the k-means algorithm to check the small variability of execution times. The k-means algorithm is an algorithm clustering n objects based on attributes into k partitions, where $k < n, k \in \mathcal{Z}, k > 0$. The aim of the k-means algorithm is to divide N data points into K disjoint subsets S_j containing data points such that the sum-of-squares criterion $J = \sum_{j=1}^{K} \sum_{n \in S_j} |x_n - \mu_j|^2$ is minimized where x_n is a vector representing the nth data point and u_j is the geometric centroid of the data points in S_j. The steps of the k-means algorithm are the following:

- **Step 1** We initialize k as an intuitive number of clusters
- **Step 2** We put any initial partition that classifies the data into k clusters as follows
  (a) We take the first k training sample as single-element clusters;
  (b) We assign each of the remaining $(N - k)$ sample to the cluster with the nearest centroid. After each assignment, we recompute the centroid of the gaining cluster;
- **Step 3** We take each sample in sequence and compute its distance from the centroid of each of the clusters. If a sample is not in the cluster with the closest centroid, we switch it to that cluster and update the centroid of the cluster gaining the new sample and the cluster losing the sample.
- **Step 4** We repeat Step 3 until the testing distances of the samples to centroid doesn't cause a new assignment.

2. **Pessimistic pWCET estimation**

We have tighten the pWCET estimation obtained previously (using the method presented in D3.4) for the case when the measurements may have large differences (for very large values we observe small associated probabilities) as shown in Figure 7. This difference has motivated the modification of the method by choosing for low variability sets of execution times the smallest value between GEV and GPD. For instance in Figure 8 we may note the decreased pessimism of the pWCET estimate with respect to the original pWCET given previously in Figure 9. The modification is based on the observation that the pessimistic pWCET estimation has been always obtained in presence of execution times with small variability.

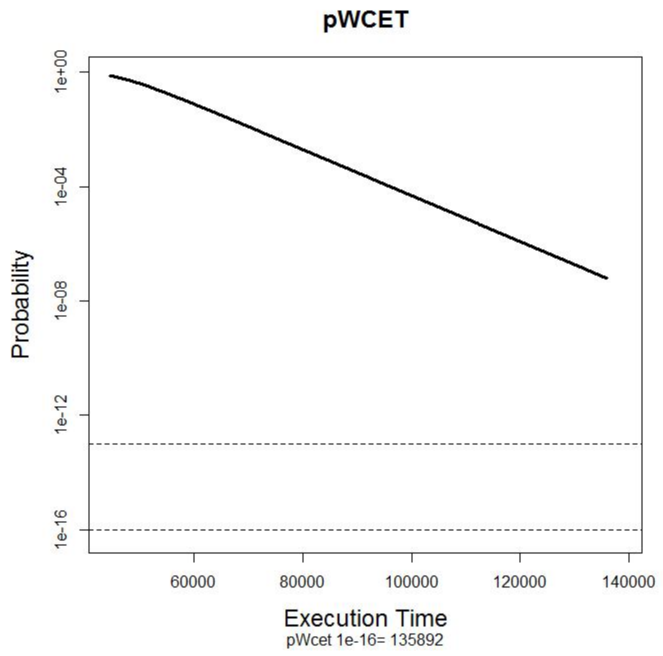


Figure 7: The pWCET estimation has a descreased pessimism

3. **Non-identically distributed sets of execution times**

   One mandatory requirement for the application of EVT is the identically distributed hypothesis. Testing this hypothesis should be done in a deterministic manner in the sense that for two applications of the test on the same set of execution times a unique answer should be obtained. This observation has been completed by the fact that the previous testing procedure of identically distributed data was providing different answers in case of high cardinal sets of execution times. The previous identical distributed test random sampling has been done on two randomly extracted sub-sets of samples that have their probability distributions compared. If picked perfectly randomly, the execution times within those two samples would be similar. In absence of a perfect random picking, we propose a new testing procedure (theoretically equivalent to the previous one) which is repeated 100 times (though 100 different sampling procedures). For all sets of execution times we have examined (both benchmarks and PROXIMA case studies) the sets failing the test were completely failing (all 100 times). All other tests have provided a success rate

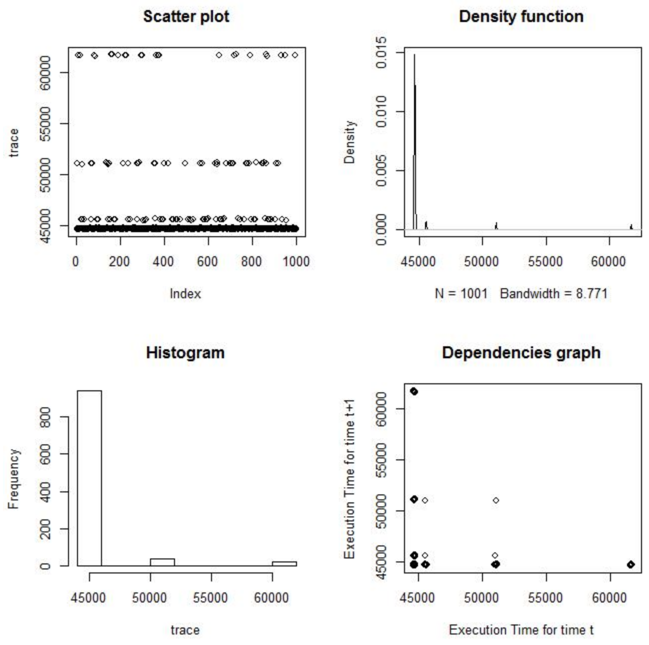


Figure 8: Low variability sets of execution times

larger than 90%. We may consider then that for a 90% rate of success the set is identically distributed. We use here the theoretical argument that for a population that is not obtained by the same procedure 0% of the test will succeed (100% failures).

2.4 Advantages and limitations

The main advantage of our method is the **theoretical proof of Gumbel as the most appropriate pWCET bound**. Within the PROXIMA project INRIA has provided means to check if for a given program and a given processor the closest EVT curve for a pWCET estimate is a Gumbel. We do not prove that Gumbel is the most appropriate pWCET bound for any program and any processor. Thus we show that for some programs and some processors this statement is correct.
The limitation of a direct utilization of the EVT-based method is the **path coverage of the pWCET estimate**. The EVT-based method provides a pWCET estimate that is a limited representativity of the feasible paths, i.e. only the paths that have been executed. This limitation has been answered within the PROXIMA project by EPC ([41] or Section 3). Moreover in D3.4 we provided a first version of a Bayesian reasoning allowing to prove the path coverage starting from benchmarks through a learning phase.
Execution time variability Our method provides a pWCET estimate from any set of execution times that present sufficient variability ($\theta > 0.5$). This variability may be obtained by randomization in the architectural components [8], randomization in the set of input data (as it has been done for Aurix board).
At the end of the PROXIMA project the minR procedure provides a number of execution times starting from which the pWCET seems stable in the sense that an increased number of execution times is not providing larger pWCET estimates.

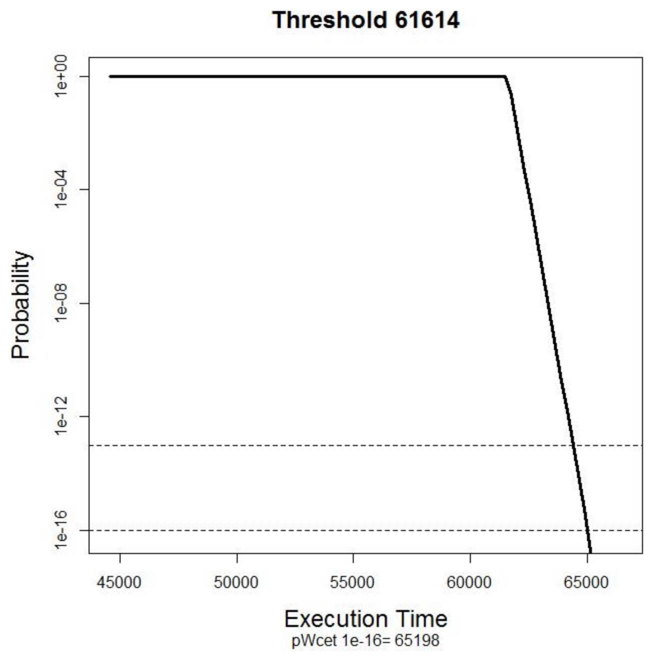


Figure 9: The pWCET estimation has a descreased pessimism

This fixed-point reasoning has no mathematical foundation and it is used today under this restriction, i.e., counter-examples may be found while using it. During the PROXIMA project, most of the time we have obtained a local convergence for 5 consecutive applications of EVT.
The inclusion of the Bayesian reasoning within EVT as a measurement protocol may solve this issue while using mathematically proved results. This new version of EVT provides a stop condition while adding new execution times to the input set of values for EVT application. We have applied it successfully for Aurix (bare metal) execution times (see the Annex B of D3.4).

2.5 Numerical evaluation

In this section we detail the pWCET estimation for the program *prime* (Mälardalen Benchmark [21]) executed on an Aurix board. The program *prime* calculates whether a number is prime. This program is used for a more detailed presentation as its structure allows injecting dependences directly through the values of the number checked (to be prime or not).
The Aurix board is composed of 3 cores: 2 TC1.6P and one TC1.6E. The major blocks of the Aurix processor are shown in Figure 10. During the experiments we use only one core and the programs are executed in isolation in bare metal mode (no other programs are on the other cores and no operating system is used).
Independent execution times We execute the *prime* program for 100 times with input values for the number (to be checked as prime) randomly picked from the set $\{13, \cdots, 5995\}$. We obtain independent execution times given in Figure 11. The execution times are varying from 0 to 500 cycles. A representation of the lag test for this set of execution times is given in Figure 12. The lack of pattern in this figure is also confirmed by the run test results with a p-value at $0.16 > 0.05$ indicating

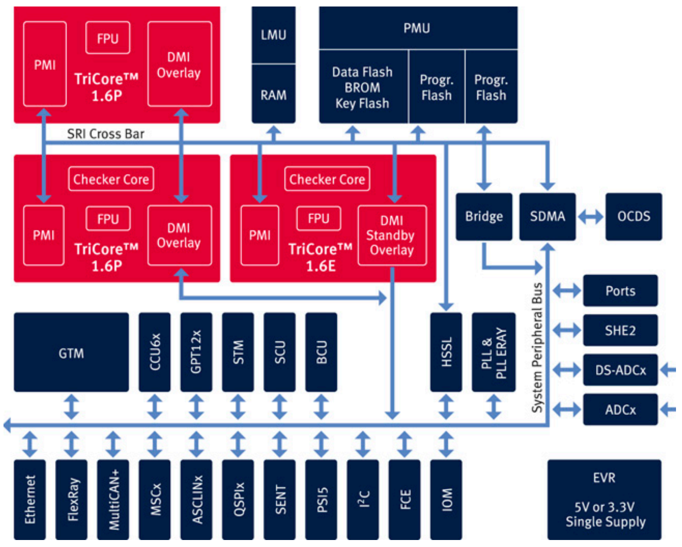


Figure 10: Tricore Aurix Block Diagram

the independence of the execution times. In this case the method presented in Section 2 is applied for the pWCET estimation and we obtain the value 526 cycles for a probability of 10^{-12}.

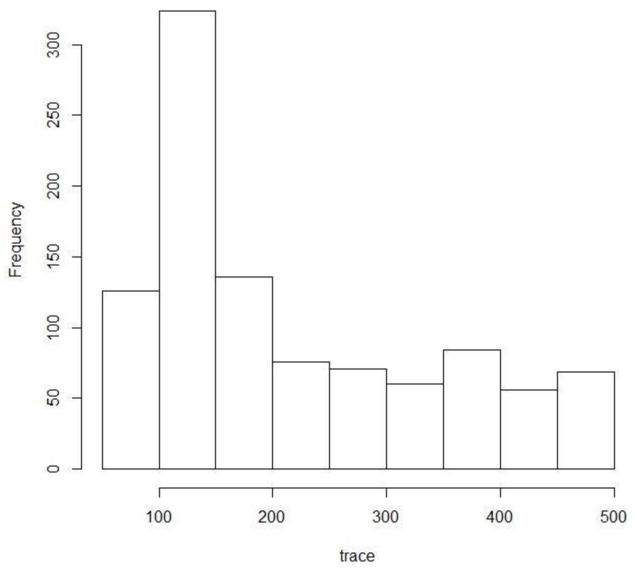


Figure 11: Distribution of independent execution times for the program *prime* on the Aurix architecture

For the second set of experiments with *prime* on Aurix board we consider dependent input values for the number (to be checked as prime) from the set $\{2010, \cdots, 5000\}$. We obtain a set of execution times described in Figure 14 which are subject to dependences. The lag plot in Figure 15 indicates patterns and the p-value for the run test ($8.77e-22 < 0.05$) confirms the weak dependences. We calculate the value $\theta = 0.77$ confirming also the utilization of EVT for dependent execution times (as described in D3.4). We obtain a pWCET estimate of 518 for a probability of 10^{-12},

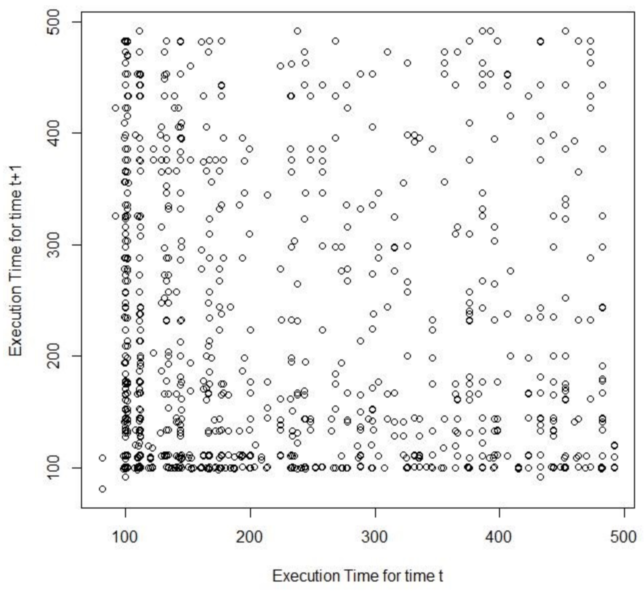


Figure 12: Lag plot for independent execution times for the program *prime* on the Aurix architecture

as seen in Figure 16.
In Table 1 we provide pWCET estimations on Aurix board for other programs of the Mälardalen Benchmark and we notice different branches of EVT participating to the pWCET estimation.

Trace	**ind**	**Method**	**pWCET**	**max**	**eval**
fir	*indep*	*GEV*	162022	135457	0.20
lcdnum	*dep*	*GEV*	2577	1171	1.20
minver	*indep*	*GEV*	61501	43380	0.42
qurt	*indep*	*GPD*	34478	28517	0.21
recursion_mutual	*indep*	*GEV*	762	346	1.20
select	*dep*	*GEV*	1689	1325	0.27
sqrt_input_inputnorm	*indep*	*GEV*	43702	38553	0.13
sqrt_input_random6	*indep*	*GPD*	43702	38553	0.13

Table 1: pWCET estimation on Aurix for some Mälardalen Benchmark programs

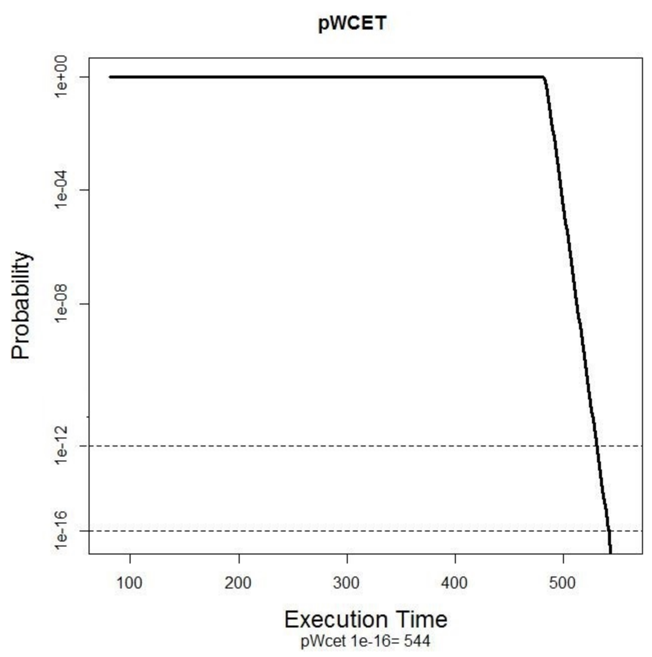


Figure 13: pWCET estimation of the program *prime* on the Aurix architecture from the measurements of Figure 12

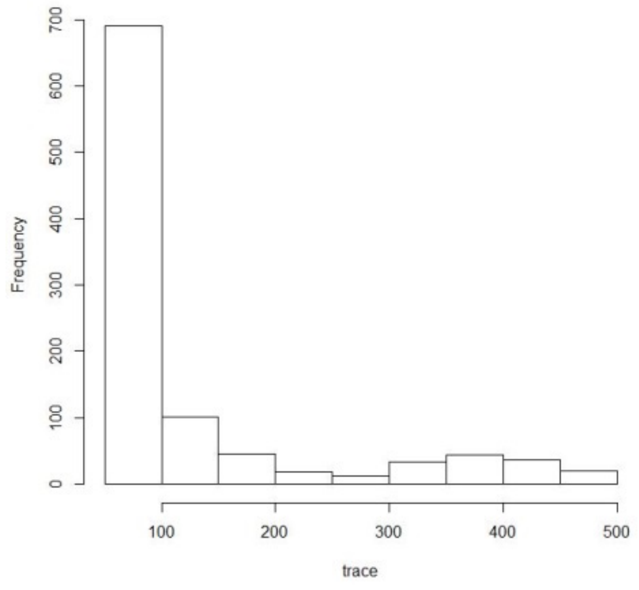


Figure 14: Dependent execution times for the program *prime* on the Aurix architecture with independent input values

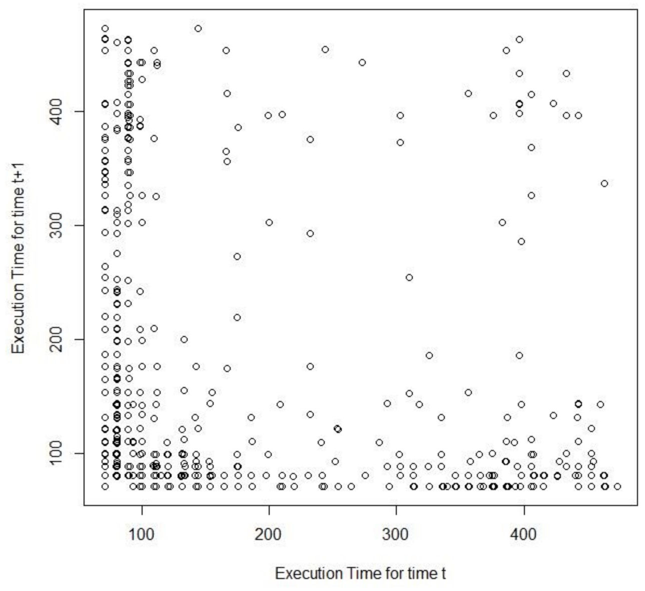


Figure 15: Lag test for dependent execution times for the program *prime* on the Aurix architecture while using independent input data

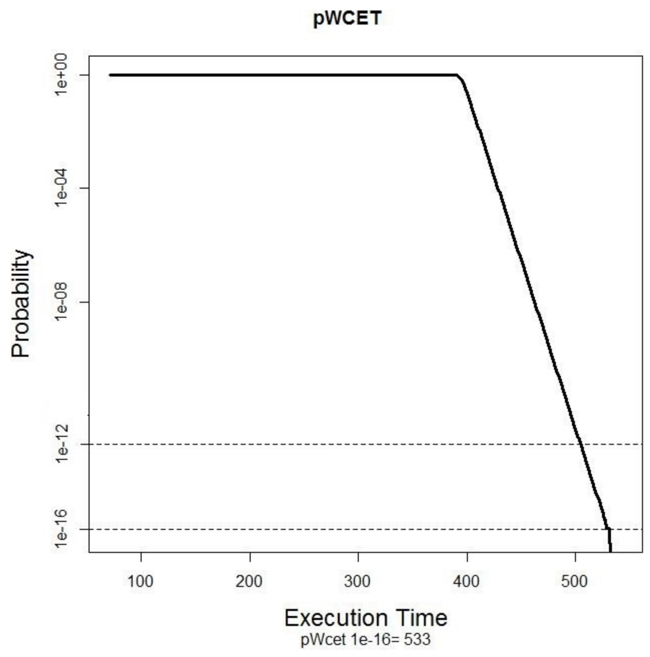


Figure 16: pWCET estimation from dependent execution times of the program *prime* on the Aurix architecture

3 EPC - the PROXIMA path coverage solution

Participants: UPD, BSC

3.1 Executive summary

In this section we introduce the solution adopted in PROXIMA to cope with the problem of path representativeness in MBPTA. In fact, MBPTA and measurement-based approaches in general the obtained (p)WCET estimates are valid only for the paths that have been exercised by the input vectors provided by the user. Unfortunately, it is generally not possible to guarantee that all paths have been triggered at analysis time. The Extended Path Coverage (EPC) aims at automatically generate synthetic observations to complement the set of measurements collected so that to obtain the same effect of full path coverage. EPC builds on the idea of probabilistic path-independence to enable the generation of synthetic measurements by simple aggregation of path-independent execution times for all basic blocks in a path.
In the last 18 months EPC have been ported from an early-stage prototype to a complete set of tools on top of the RVS tool chain. Extensions to the PROXIMA MBPTA tool chain and a set of specific tools have been realized in order to consolidate EPC into a well-defined end-to-end process.

EPC originated and have been developed exclusively within the scope of PROXIMA and therefore is a relatively young technique, whose level of industrial readiness cannot be compared to more consolidated MBPTA techniques. The current implementation still need some refinements and improvement especially on the infrastructure and exploitable structural information. Nonetheless, EPC has been tailored on the FPGA-HWRand platform and methods to support EPC trace collection have been consolidated and integrated on top of the industrial-quality tool chain, either on bare-metal or PikeOS. On the FPGA-HWRand, EPC has been successfully applied to industrial case studies. Extension to SWRand platforms and FPGA-SWRand in particular seems to be near-at-hand.

3.2 Description

The PROXIMA MBPTA approach builds on a mixture of randomization and upper-bounding (worst-case mode enforced at analysis time) to ensure that several of those SETV are transparently captured in the analysis results. Some SETV, however, remain under direct user control and need to be explicitly taken into account to guarantee that the timing behaviour observed at analysis time is representative of the system behaviour at deployment time. In MBPTA, and measurement-based approaches in general, the most evident and relevant SETV that falls into this category is the set of input vectors used to collect the timing observations: the obtained (p)WCET estimates are valid only for the paths that have been exercised by the input vectors provided by the user [31, 39].
The Extended Path Coverage (EPC) technique [41] provides a technical solution to increase the confidence on MBPTA results by guaranteeing the same degree of representativeness as that obtainable through full path coverage.

In this section we present first a technical description of EPC followed by the presentation of the main evolutions since D3.4. The advantages and limitations of EPC are detailed in Section 3.4. A numerical evaluation of EPC is provided in Section 3.5.
In a nutshell, EPC improves the representativeness of MBPTA results by synthetically extending the set of measurements that are fed to EVT and used to compute the pWCET distribution. The addition of synthetic time traces is done in a way to achieve the same effects of full path coverage as it would be guaranteed by exhaustively extending the set of input vectors. Figure 17 provides an overview on the interaction between EPC and MBPTA and how EPC operates to increase the representativeness of MPBTA results.

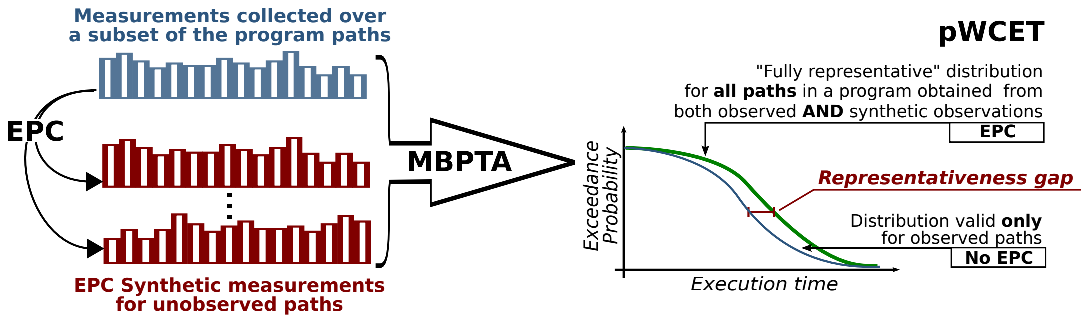

Measurements collected over a subset of the program paths
EPC
EPC Synthetic measurements for unobserved paths
MBPTA
pWCET
"Fully representative" distribution for **all paths** in a program obtained from both observed **AND** synthetic observations
EPC
Representativeness gap
Distribution valid **only** for observed paths
No EPC
Exceedance Probability
Execution time

Figure 17: EPC in relation to standard MBPTA process and effects on its results.

The main requirement of EPC consists in the availability of a set of measurements for each basic block (hence requiring basic block coverage[1]), irrespective of the path leading to it. In fact, to derive synthetic measurements, EPC focuses at the level of basic blocks and exploits the probabilistic nature of the PROXIMA hardware to derive *probabilistic path-independent* execution times for each basic blocks. Each basic block is likely to exhibit different timing behaviour depending on the execution history, that is the specific path leading to it. More specifically, in our reference PROXIMA architecture (i.e, FPGA), the execution time of a basic block may vary as an effect of time-randomized caches, which we consider the main sources of (intra-core) variability in time-randomized architectures. Consequently, we make execution times of basic blocks path independent simply by probabilistically summing up a penalty or padding to each observed execution time to compensate for any positive cache effect due to a specific path. To this extent we consider the contribution of each memory access to the execution time of a basic block and formalize the notion of Access-Time Profile (ATP) for each memory access as follows:

$$ATP(@_A, \phi) = \left\langle \begin{array}{cc} L_{hit} & L_{miss} \\ P_{hit}(@_A, \phi) & P_{miss}(@_A, \phi) \end{array} \right\rangle \quad (1)$$

Equation 1 simply models the fact that with time-randomized caches the latency (either L_{hit} or L_{miss}) incurred by an access to a memory location $@_A$ along a path ϕ follows the probability distribution of that access to be a cache hit or a cache miss.

[1]Basic block coverage represents a common coverage requirement in DO-178C already from DAL C [34].

Along each path, the ATP is determined by whether the intermediate accesses between the current and the previous access to $@_A$ hit or miss in cache. We make basic block execution times path-independent by adding a penalty ($L_{pad} = L_{miss-hit}$, which compensates for potential cache hits) to enforce the padded $ATP^+(@_A)$ to tightly over-approximate the theoretical worst-case $\overline{ATP}(@_A)$ along any path in the program, as graphically explained in Figure 18.

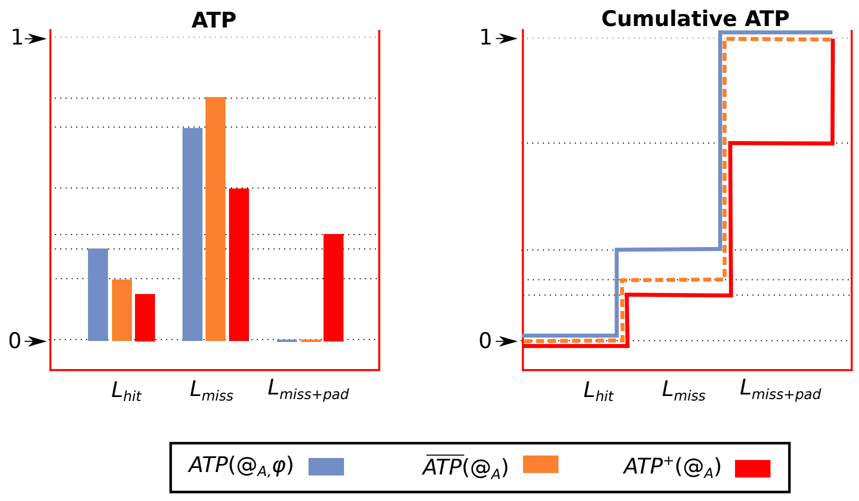


Figure 18: Relation among $ATP(@_A, \phi)$, $ATP^+(@_A)$ and $\overline{ATP}(@_A)$.

The $ATP^+(@_A)$ is obtained by adding a (scalar) padding with a given probability to each memory access in $ATP(@_A, \phi)$. More formally $ATP^+(@_A)$ can be formalized as follows:

$$\begin{aligned} ATP^+(@_A) &= ATP(@_A, \phi) \otimes \left\langle \begin{array}{cc} 0 & L_{pad} \\ 1 - P_{pad}(@_A, \phi) & P_{pad}(@_A, \phi) \end{array} \right\rangle \\ &= \left\langle \begin{array}{cc} L_{hit} & L_{miss} \\ P_{hit}(@_A, \phi) & P_{miss}(@_A, \phi) \end{array} \right\rangle \otimes \left\langle \begin{array}{cc} 0 & L_{pad} \\ 1 - P_{pad}(@_A, \phi) & P_{pad}(@_A, \phi) \end{array} \right\rangle \\ &= \left\langle \begin{array}{ccc} L_{hit} & L_{miss} & L_{miss} + L_{pad} \\ P^+_{hit}(@_A) & P^+_{miss}(@_A) & P^+_{miss+pad}(@_A) \end{array} \right\rangle \end{aligned} \quad (2)$$

To enforce this modified distribution we need to add a padding L_{pad} to each memory access $@_A$ along a path ϕ, according to the probability P_{pad}, as shown in Equation 2. As formally described in [41], the only constraint we need to meet is that:

$$P_{pad}(@_A, \phi) \geq 1 - \frac{\overline{P_{hit}}(@_A)}{P_{hit}(@_A, \phi)} \quad (3)$$

where $\overline{P_{hit}}(@_A)$ is a a lower bound to the hit probability of $@_A$ along any possible path and $P_{hit}(@_A, \phi)$ is the exact probability of $@_A$ to be a hit along path ϕ. To compute $\overline{P_{hit}}(@_A)$ and $P_{hit}(@_A, \phi)$ we exploit the concepts of *reuse distance* and *unique accesses.*

Definition (Reuse distance - *rd*)**.** *The reuse distance of $@_A$ on a path ϕ is defined as the number of memory blocks mapped to the same set of $@_A$ accessed between $@_A$ and the previous access to the memory block containing $@_A$. The worst-case reuse distance (along any path) for access $@_A$ is denoted by $\overline{rd}(@_A)$.*

Definition (Unique accesses - *un*)**.** *With unique accesses, instead, we refer to the number of* ***distinct*** *memory blocks mapped to the same set of $@_A$ accessed in between $@_A$ and the previous access to the memory block of $@_A$ on a path ϕ.*

Using reuse distance and unique accesses we compute $\overline{P_{hit}}(@_A)$ and $P_{hit}(@_A, \phi)$:

$$\overline{P_{hit}}(@_A) = 1 - \overline{P_{miss}}(@_A) \text{ where } \overline{P_{miss}}(@_A) = \begin{cases} 1 - \left(\frac{w-1}{w}\right)^{\overline{rd}(@_A)} & \text{if } \overline{rd}(@_A) < w \\ 1 & \text{otherwise} \end{cases} \tag{4}$$

$$P_{hit}(@_A, \phi) \leq uP_{hit}(@_A, \phi) = \begin{cases} 1 & \text{if } un(@_A, \phi) < w \\ \left(\frac{w-1}{w}\right)^{un(@_A,\phi)-w+1} & \text{otherwise} \end{cases} \tag{5}$$

where w is the number of ways in a set-associative cache. In fact, in our implementation, we over-approximate both $\overline{P_{hit}}(@_A)$ and $P_{hit}(@_A, \phi)$ by computing $\overline{rd}$ and un on a restricted scope (i.e., not considering all execution history backwards). Finally, we can substitute the values computed for $\overline{P_{hit}}(@_A)$ and $uP_{hit}(@_A, \phi)$ in Equation 3 to compute a sound value for $P_{pad}(@_A, \phi)$, as shown in Equation 6.

$$P_{pad}(@_A, \phi) = \begin{cases} 0 & \text{if } uP_{hit}(@_A, \phi) = 0 \\ 1 - \dfrac{\overline{P_{hit}}(@_A)}{uP_{hit}(@_A, \phi)} & \text{otherwise} \end{cases} \tag{6}$$

To obtain path-independent observations for a basic block *bb* we therefore augment each collected measurement *Obs* by adding a padding for each memory access and path, according to the computed $P_{pad}(@_A, \phi)$. Therefore for all accesses $@_A$ along path ϕ in *bb*:

$$Obs^{+}(bb) \mathrel{+}= \begin{cases} Obs(bb, \phi) + L_{pad} & \text{if } rand() \leq P_{pad}(@_A, \phi) \\ Obs(bb, \phi) & \text{otherwise} \end{cases}$$

These path-independent figures are then combined together to form synthetic end to end time traces. The use of path-independent basic block execution times as elements to build synthetic execution times for each non-observed path $\overline{\phi}$ in the program, ensures that the constructed collection of values for $\overline{\phi}$ is a valid upper-bound of any collection of real measurements. It is worth recalling that (probabilistic) path-independence is an absolute requirement as we cannot simply sum the execution time observations over basic blocks (even maxima values) to obtain the execution time of unobserved paths. The main reason is that observations are only relative to a particular observed path and its particular cache-level and core-level timing dependencies[2].

3.3 Main evolutions since M18

Although the EPC approach was already theoretically well defined and solid at MS2, its implementation was just prototypical: the whole EPC process was still far from being smoothly usable by PROXIMA industrial partners. In fact, the first prototype implementation of EPC aimed at a preliminary evaluation of the

[2]Timing anomalies, which we do not consider here as they do not appear in the PROXIMA platform, would make this naive approach even more unsound.

technique and as such it was mainly focused on the computation (and application) of the probabilistic padding while hiding the complexity of collecting all the information required by the analysis as well as of deriving synthetic end-to-end measurements. For the sake of completeness, we recall that the application of EPC only slightly deviates from the PROXIMA MBPTA approach and specifically in:

- *Instrumentation*: as EPC works on timing information at the granularity level of single basic blocks. Automatic support to instrument and build an instrumented version of the target executable is required.

- *Reconstruction of structural information*: EPC requires some degree of structural information in order to perform a precise probabilistic padding computation. In particular, EPC builds on the availability of information on the set of data and instruction addresses within each basic block.

- *Synthetic path generation*: EPC exploits probabilistic path-independent timing information to construct synthetic measurements for unobserved paths in a program. These synthetic observations are fed in input to the pWCET computation step as a complementary set of information to the set of the concrete timing observations. Support is therefore needed to find and synthesize those unobserved paths.

The subsequent steps, consisting in feeding (real and synthetic) timing traces to EVT to compute pWCET distributions, do not diverge from the PROXIMA MBPTA approach and therefore simply took advantage of the advancements produced in the EVT line of work. An equally critical and time-consuming line of work, aimed at improving the integration of EPC with the RVS suite. With respect to the integration within the tool-chain, the actual support available at MS2 was limited to the program representation and the time measurement collection phase in EPC, while data manipulation and most computations were performed by poorly efficient external python scripts. Details on the integration are provided in deliverable D3.11. Finally, we also refined the core technique itself and its implementation, based on some preliminary feedback from the familiarization phase of industrial partners with the prototype toolchain.

The RVS toolchain was already able to automatically support software instrumentation at the basic block granularity. The only modifications required to the RVS toolchain were related to management and storage of the relevant data. These modifications are discussed in deliverable D3.11 on Final industrial analysis tools, as part of the EPC-RVS integration status. In the following we detail on the work done to bridge the gaps in the EPC process and to bring it to a mature and industrially reproducible state on top of the PROXIMA FPGA platform, namely for reconstruction of structural information and synthetic path generation. All the implemented features have been made available to the industrial users as a set of small tools and scripts so that to fully automate the process.

Reconstruction of structural information. Structural information on the set of data and instruction addresses within each basic block is required by EPC in order to properly compute the probabilistic padding. In fact both reuse distance (rd) and number of unique accesses (un), as defined in Section 3, are determined by

combining information on memory accesses and how they map to the cache, which in turns depends on cache placement and replacement functions. Therefore we need to derive and store information on both cache mapping and memory accesses. With respect to cache mapping, in time-randomized cache it depends on the current seed used to determine random placement and replacement functions: while the effect of random replacement is transparently captured by the timing traces, the seed used to determine the random placement function is crucial in the computation of *rd* an *un*. We include the seed information to each time traces by dropping this information as a special instrumentation point in the trace file.
When it comes to memory accesses, EPC exploits information on all data and instructions in the program and how they map to basic block boundaries. Although this kind of information could be statically computed from, for example, information derived from the compiler, we derive it as a result of an additional dynamic analysis step consisting in the collection of address and instruction traces from concrete runs on the target platforms. In practice an additional run is performed for each observed path in the target program just to collect structural information and without considering the timing. In contrast with timing traces, where several runs of each path are required to characterize their timing behaviour, *one single run* is sufficient to collect an *address trace* with all the required structural information.
When moving to the PROXIMA target, the requirements of EPC translate into a methodology to extract those address traces for both data and instructions. Traces must comprise every and each memory address that has been accessed during program execution, including the target memory address for every read or write operation issued.
The methodology developed at M18 was based on the available hardware debug capabilities to extract the required information. The approach consisted in performing a step-by-step execution using the GRMON step command, and extracting the addresses stored in the instruction circular buffer, which saves the last 256 executed instructions. At the same time the AHB circular buffer, which stores every access to the main bus, was also dumped to extract the last 256 bus accesses. From these accesses, a trace of the latest accessed memory addresses was derived.
The old methodology required disabling L1 data and instruction caches so every access was accessing the main bus, where the access was logged in the AHB bus. It also required the step-by-step execution to execute in small groups of instructions (16 or 32 instructions at most) so the circular buffers were not overwritten; this caused the resulting traces to be quite big because of duplicate data. Finally, the extracted data had to be heavily post-processed to remove duplicates and match instruction and data traces, which are not easily linkable (timestamps do not exactly match). Further post-processing was also required to cope with the fact that an instruction trace stored virtual addresses while a data trace stored physical addresses: this was not a trivial task at all, as the translation tables could be altered by the RTOS.
The new methodology developed by M36 is no longer based on step-by-step execution, and uses the new tracing capabilities implemented by the PROXIMA target platform. The tracing mode for EPC has to be explicitly enabled because it will cause the processor to stall whenever the trace buffer is full (contrary to the usual behaviour, which requires real-time trace dumping to be able to analyze execution times with RVS). When this mode is enabled, the resulting traces will be suitable

for EPC. To this end, the tracing mechanism has been updated to extend each record to be 128-bit (instead of 64-bit, which is used to store regular traces which only save timestamp and ipoint). Each record has 4 fields: instruction address, ipoint, load/store address, and timestamp. The load/store address field is 0 when the instruction did not create a data access. With this information the address generation process is trivial, and only requires converting the trace from binary to text format, filtering instrumentation points so they are taken into account as special instructions, and separating instruction and data memory accesses.

Synthetic path generation. The practical concerns in the generation of synthetic measurements are an important aspect in EPC implementation that was not exhaustively analysed yet at M18. The idea of generating synthetic measurements from a path is a simple task or at least it is so in our setting where we have no timing anomalies and path-independent execution times for each basic blocks are available. In fact synthetic trace generation simply consists in summing up the (path-independent) execution times of each basic block along the path. What is left uncovered in the theoretical formulation of the technique [41] is how to derive the set of unobserved paths for which we want to collect synthetic measurements.

Reasoning on path coverage for WCET analysis is quite tricky as there is no such a metric or heuristic as in functional analysis. In the lack of any objective criteria to identify a priori the worst-case path (WCP) or the set of potential candidates among which the WCP lies, we are forced to consider the whole set of feasible paths. Clearly, the number of paths in a program is rapidly increasing with its increase in size and structural complexity. Reducing the number of paths to be considered in the analysis is a well-known research line in timing analysis, based on the observation that not all *structurally* feasible paths are actually *semantically* so. Typically only a small subset of the set of potentially feasible paths can happen in practice owing to data flow constraints. This same observation has led to the definition of the notion of *flow facts* [25], either automatically derived or manually defined by the user itself, to restrict the set of paths to be considered in the analysis.

The solution we elaborated in PROXIMA consists in an automated tool (`SynthPath`) to compute the set of synthetic paths that complement the set of collected observations. The tool basically relies on an extremely efficient program model that allows to explore and generate a large number of paths without compromising efficiency in time and space. At the same time the program model (and tool) also incorporates the maximum observed loop bounds and accommodates a set of simple control-flow constraints to refine the semantics of the program. The basic support to flow facts included in the current release of the `SynthPath` tool only includes a simple annotation language as we cannot take advantage of all the information usually available as part of static analyses (e.g., data flow analysis). However, the supported language still allows expressing a relevant set of flow constraints that can effectively be used to exclude semantically (as opposed to structurally) infeasible paths and thus reduce the number of synthetic paths generated by the tool. The cornerstone of the annotation language are control-flow constructs, namely if-then-else conditionals and switch/case statements, and loops. The language (and the tool) provides limited support for discriminating between different execution

contexts or scopes (e.g., specific calls to a sub program). This feature however is not exploited at the moment as control-flow constructs are simply addresses with globally unique IDs. The formal specification of a grammar for the annotation language as well as detailed description and examples of the supported flow-facts is reported in the `SynthPath` tool description and user guide (D3.11).
It is also worth noting that the number of semantically feasible paths is also reduced owing to the fact that some synthetic paths, though different, can be *equivalent* from the EPC standpoint. In fact, since basic block execution times in synthetic paths are path independent, two paths are equivalent whenever they exhibit the same basic blocks regardless of the order in which they appear. Consider, as an example, two paths represented as basic block ID sequences:

$$P_1 = 1, 2, 3, 4, 5, 6, 4, 5, 6, 2, 3, 4, 5, 6, 7$$
$$P_2 = 1, 2, 3, 4, 5, 6, 2, 3, 4, 5, 6, 4, 5, 6, 7$$

Clearly those sequences are different and also the corresponding paths are. However, if we just consider basic blocks and their frequencies then:

$$P_1 = 1(1), 2(2), 3(2), 4(3), 5(3), 6(3), 7(1)$$
$$P_2 = 1(1), 2(2), 3(2), 4(3), 5(3), 6(3), 7(1)$$

and P_1 and P_2 are identical. In EPC, when building synthetic measurements we do not consider (based on path independence) the order of basic blocks. Generating synthetic observation for both paths would be redundant: `SynthPath` discards equivalent paths by default.

3.4 Advantages and limitations

The clear advantage of EPC is that it provides a conclusive solution to the path-representativeness problem. The trustworthiness of MBPTA results is in fact conditional upon the representativeness of the input vectors used at analysis time and there is no generic scientific argument to tell whether a test campaign is triggering all the relevant paths in a program. When EPC is in place, instead, we are confident that the pWCET values computed by MBPTA are valid for all paths in the program. More generally, EPC allows to relax the strict dependency between the quality of the input vectors and the results of the analysis: no particular assumption is made on the set of inputs, as as long as basic block coverage is provided. On synthetic benchmarks in [41] and confirmed by the numerical evaluation in Section 3.5, the degree of over-approximation necessarily introduced by EPC is in general more than acceptable (always below the 20% margin traditionally adopted in industrial practice) and could be further improved by specializing the tool support. We consider this to be not only a relevant theoretical achievement but also a notable advancement from an industrial standpoint.

When it comes to EPC tracing requirements, it is worth noting that the new tracing approach developed after M18 enables a fast trace collection phase, compared to the old methodology. With the old approach from M18, a step-by-step execution was systematically stopping the execution by stalling the processor and returning control to the debug interface, where buffers were dumped to standard output using the debug link, before returning the control back to the processor just for a few cycles, only to be interrupted again. With the new methodology the tracing is done in the background and the processor is only stalled whenever it is required (i.e. when the trace buffer is full) and for a much shorter period of time, while saving the traces in binary format (which requires less storage) in the on-board debug RAM. Moreover, with the new mechanism, traces are dumped only once at the end of program execution.

On the downside, EPC comes with some limitations. Among the potential drawbacks we identified, some of them are inherent to the technique while other are more related to the current implementation.
A first limitation comes from the fine-grained instrumentation level it requires to collect timing traces at the level of basic blocks. This is more generally an issue for all fine-grained (hybrid) measurement-based timing analysis approaches. In fact, in software-level instrumentation approaches, as opposed to hardware-based ones, timing traces are collected through inserting instrumentation points directly in the code, which is inherently intrusive, on both the temporal and functional behaviour. The interference caused by instrumentation (known as probe effect) is exacerbated when the number of instrumentation points increases as in the case of EPC: deploying the instrumented code could be penalizing in term of performances or even difficult to justify against stringent industrial qualification and certification standards. We foresee two possible solutions to this problem. The easiest approach could be the exploitation of advanced hardware debug interface to collect timing data directly from hardware-level instrumentation. Whenever this is not allowed by the target platform, alternative methods to reduce the software-level instrumentation overhead can be adopted [12]. Or else, if full-path coverage can be guaranteed for some code regions, then end-to-end measurements of each different path can be captured without instrumenting those regions at finer granularity. However, this feature has not been developed yet and is left for future consideration.
A second drawback of EPC consists in the amount of pessimism it can potentially incur. In the empirical evaluations of EPC performed so far on top of synthetic benchmarks (e.g, Malardalen WCET benchmark and EEMBC automotive benchmarks in [41]) EPC effectively extended the representativeness of MBPTA results while incurring an increase in the pWCET always lower than 20% (typical reference value for industrial safety margins) with respect to the values computed with MBPTA alone. Looking at EPC results, however, it is not easy to tell apart the effects of overestimation (necessarily introduced by enforcing path independence) from those of unobserved paths. This is one of the objectives of the numerical evaluation presented in Section 3.5 below. In principle, we are aware that some particular code constructs and/or particular program structures may increase the pessimism introduced by EPC. And some preliminary feedback from the industrial users confirmed that. We identified two potential issues that may lead to more pessimistic results:

- In order to keep the computational complexity in deriving the padding probability P_{pad} relatively low we opted to compute reuse distance and unique accesses only on a restricted scope (immediate predecessors) rather than the whole execution history back to the program entry point. Experiments on the prototype, in fact, suggested this restriction could improve the computational efficiency with only marginal overestimation. This conclusion however may not be generally valid and it may be interesting to investigate larger scopes, which unfortunately is not supported by the current implementation on top of the RVS tool.

- According to the formula for computing P_{pad} (Equation 3) we do not apply any padding for a given access $@_A$ along a path ϕ in two different cases: either (i) when accessing $@_A$ always results in a hit independently of the path ($\overline{P_{hit}}(@_A) = uP_{hit}(@_A, \phi) = 1$), or (ii) when the access $@_A$ along ϕ cannot be other than a miss ($uP_{hit}(@_A, \phi) = 0$), as explicitly stated by the formula itself. Those cases correspond to the *always_hit* and *always_miss* categorization in static cache analysis approaches. The problem in the current implementation is that we cannot fall into the second case (*always_miss*) because we do not explore the whole execution history when computing the lower and upper bounds to the hit probabilities, as commented in the previous point. Therefore, the only safe choice is to consider that the address was accessed just before entering the scope of the computation, which may cause paddings to be unnecessarily applied. Basic blocks that include some code patterns such as long blocks of variable initializations may thus incur non-negligible pessimism. Some memory accesses are known to always incur a cache miss the first time they are accessed: they fall into the *first_miss* category. We were able to partially intercept *first_misses*, limitedly to instruction accesses outside loops. In general (and for data accesses) this issue cannot be solved by extending the scope of computation unless we consider the full history, which is not recommended from the computational standpoint. A possible solution would consist in dynamically extending the scope or defining a probabilistic threshold beyond which it would be allowed to consider that an access, even if actually happened, can be safely assumed to not being in the cache at some point.

- In the computation of P_{pad} for basic block within loops, it would be important to discriminate between the first and all the other iterations. This is because the real set of predecessors for the first iterations of the loop header are typically different (include the loop entry link and the back-link, which may negatively affect the computation of $\overline{P_{hit}}(@_A)$. This apsect was clear from the beginnning but it was hard to implement on top of the RVS framework. Although the RVS toolchain supports loop unrolling, the loop information is lost in the process of applying the padding. This issue can effectively be solved by modifying the way RVS holds EPC data. Unfortunately, the complexity of the toolchain did not allow to perform those changes within the scope of PROXIMA.

In all the above cases, the larger the basic blocks are (and the more accesses they include) the more pessimism we get. Automatically generated code, characterized by long initialization sequences and frequent updates of a large set of variables is likely to be largely penalized.

3.5 Numerical evaluation

The results obtained with EPC necessarily dominate those obtained with standard PROXIMA MBPTA as EPC is augmenting the MBPTA input space. However, when comparing pWCET figures with EPC against standard ones, it is not easy to distinguish between the contribution of unobserved paths and the effect of overestimation, which is inherently introduced by enforcing path independence. A first numerical evaluation of EPC was conducted on the prototype implementation (only loosely integrated with RVS) and presented in D3.4. That evaluation, however, was not looking into the factors that contribute to EPC results. Once the integration of EPC on RVS was accomplished, and before moving to the evaluation on industrial case studies, we wanted to better investigate two aspects of EPC: (i) to what extent EPC results are determined by improved representativeness or overestimation and (ii) how pessimism relates to original path coverage (i.e., how many synthetic path are considered). The results of this evaluation are reported in the following.

As a prerequisite to our investigation, we needed to have full control over the target program in terms of input vectors and coverage. We therefore concocted a synthetic demonstrative example, small enough to be easily managed but at the same time not trivial, to experiment the technique. The example we designed, which we call `MultiBranch`, is a single procedure exhibiting four `if-then-else` constructs in a sequence (thus generating a total of 16 paths), where each branch is performing some computation and executing some loops with a different number of iterations. An overview of the structure of `MultiBranch` is provided in Figure 19.

The outcome of each branch decision is explicitly controlled by a value in the input vector of `MultiBranch`. This way we were able to fully control the program behaviour, as well as to selectively trigger the minimum number of paths – that is just the two paths framed in red in Figure 19 – required to fulfil EPC coverage requirement. Knowledge on the program semantics also allowed us to identify a priori the worst-case path (i.e., leading to the worst-case behaviour), framed in black in Figure 19.
The main reason to run our investigation on a relatively simple program is that the low number of paths allows to perform a full path coverage test campaign and use its results as a term of reference for comparison with the results drawn from different degrees of path coverage. The sets of structurally and semantically feasible paths coincide in `MultiBranch`.

In our empirical evaluation we focused on five different scenarios of application of EPC with varying number of observed paths:

- `16paths` All paths triggered at analysis time so that measurements are already fully path-representative;
- `2paths` Only two paths actually measured (the two providing basic block coverage), excluding the worst-case path.
- `4paths` Four paths actually measured, excluding the worst-case path.
- `8paths` Half of the paths actually measured, excluding the worst-case path.
- `W-8paths` Half of the paths actually measured, in this case including the worst-case path.

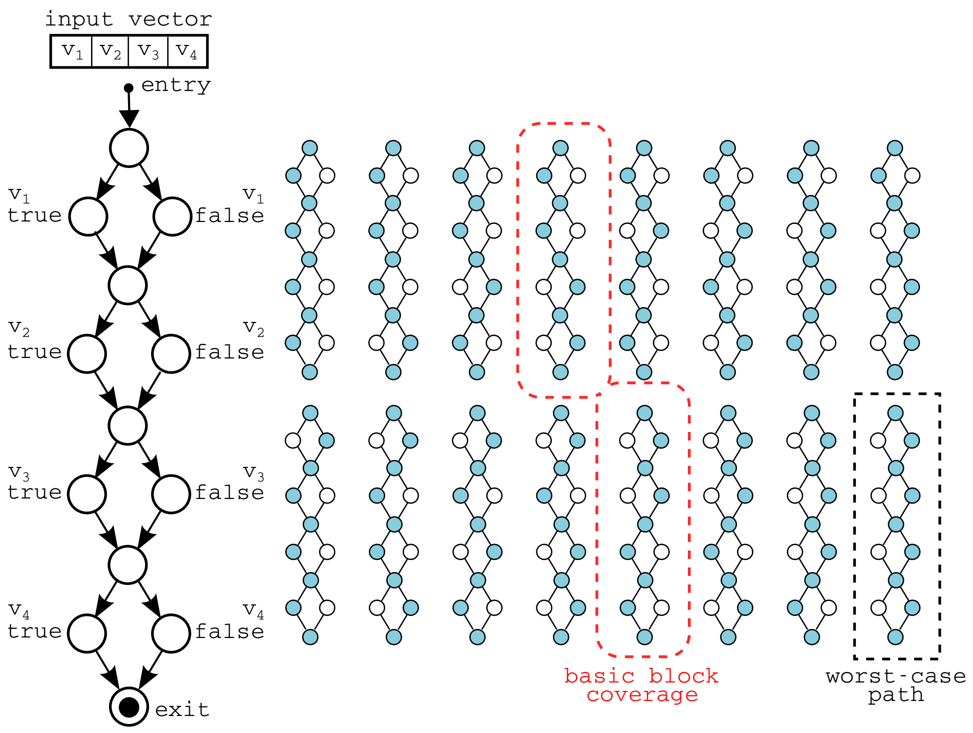


Figure 19: Overview of `MultiBranch` structure.

Under each configuration we applied EPC first[3] and then MBPTA on all program paths separately. Among all paths we then selected the maximum pWCET for the probability thresholds we were interested in. For each path we collected (or generated, for synthetic paths) 1200 runs. The so-obtained set of inputs successfully passed the i.i.d. test before applying EVT. Table 2 reports the Maximum Observed Execution Time (MOET) and pWCET at e^{-9} and e^{-10} thresholds computed by MBPTA over the set of measurements obtained by exhaustively observing all paths in `MultiBranch`. As expected path number 16 here corresponds to the worst-case path.

Path		MOET	pWCET 10^{-9}	pWCET 10^{-10}
16paths	P1	5420	5995	6065
	P2	5777	6398	6472
	P3	5649	6275	6354
	P4	5727	6322	6394
	P5	5621	5897	5940
	P6	5543	6150	6228
	P7	5424	6067	6143
	P8	5383	5580	5614
	P9	5528	6051	6115
	P10	5235	5862	5943
	P11	5844	6498	6580
	P12	5972	6511	6587
	P13	5714	5918	5947
	P14	5959	6654	6738
	P15	5193	5767	5841
	P16	6152	6786	6865

Table 2: MBPTA results on the full path-coverage scenario.

[3]As expected EPC had no effect in the full-coverage scenario.

We used the pWCET values obtained in the `16paths` scenario as the baseline for comparison of all the other scenarios. By having the MBPTA results on all observed paths, we can reason on both the un-representativeness (and un-safeness) of partial coverage results and the overestimation incurred by EPC. Table 3 reports largest MOET and pWCET at e^{-9} and e^{-10} and compare them against the respective values in the `16paths` scenario, by reporting the difference in percentage.

Scenario	MOET	%	pWCET 10^{-9}	%	pWCET 10^{-10}	%
`2paths`	5777	-6.1	6398	-5.71	6472	-5.72
`2paths+EPC`	6917	+12.43	7855	+15.75	7981	+16.25
`4paths`	5727	-6.91	6322	-6.83	6394	-6.86
`4paths+EPC`	7021	+14.12	8009	+18.02	8138	+18.54
`8paths`	5844	-5.00	6498	-4.24	6580	-4.16
`8paths+EPC`	6994	+13.68	8090	+19.20	8236	+19.97
`W-8paths`	6152	0	6786	0	6865	0
`W-8paths+EPC`	6987	+13.57	8056	+18.71	8194	+19.35

Table 3: MBPTA results on incomplete coverage plus EPC scenarios.

Several interesting observations can be drawn from the experiments. First, results confirms that EPC delivers fully representative pWCET figures that safely over-approximate the pWCET obtained by actually observing all paths in a program. MBPTA results obtained with partial coverage are necessarily flawed when the worst case scenarios are not captured at analysis time. Despite the limitations in the current implementation, the degree of overestimation incurred by EPC is always below the 20% safety margin, widely adopted in industry. In particular the pessimism introduced by path-independence and synthetic path generation, as observed in the increase on the MOET, is very limited (+14% in the worst-case). The residual overestimation can be observe in the pWCET distribution as an effect of "moving" the pWCET curve to the right. An extremely interesting evidence that can be drawn from these numbers is that the amount of overestimation does not seem to strongly depends on the degree of coverage before applying EPC. This is in part explained by the fact that synthetic measurements are not always identical as they sum up random element from basic block execution times and therefore their contribution to the pWCET is not constant nor easily predictable. Interestingly, this means that, basic block coverage is really the only requirement from EPC. The results on the `W-8paths` scenario show that the worst-case values after applying EPC are greater than real (measured) ones. This indeed may happen when the difference between the worst-case and other values is not so large that it can be flattened by EPC (in fact lays beyond the 14% average increase on MOET).

4 Handling COTS multicore contention via partial Time Composability (pTC-VICI): Applicability to FPGA-SWRand

Participants: BSC

4.1 Executive summary

In this section we introduce the method used to account for multicore contention on top of the FPGA-SWRand platform. This method builds upon two premises: (i) no assumption can be made on how requests from the different contenders will align in time and hence, whenever a request from another core can create contention, the worst contention for that request is assumed; (ii) pTC-VICI (or pTC for short) allows to budget for a limited number of requests of specific types (with the worst timing alignment) so that an appropriate timing bound is chosen at integration time once contenders are known, or simply the contention allowed by contenders is limited (in terms of number and type of requests) and this is monitored with Performance Monitoring Counters (PMCs) during operation preventing contenders to exceed their allowed request budgets. The pTC model is fully built upon PMCs measurements so as to match industrial practice.
The pTC model has been developed from scratch since m18 where it did not exist, and it was developed to cover the FPGA-SWRand platform, for which no solution was in place to use by the end users' case studies.
This approach is not restricted to the specific platform for which it has been devised and integrated (FPGA-SWRand), and it can be potentially ported to other hardware platforms. However, how to port it to significantly more complex platforms with out-of-order execution, parallelism in the memory access, etc. needs to be investigated for an appropriate adaptation of the pTC approach. Currently, this method has been successfully integrated into RVS and the case studies on top of the FPGA-SWRand platform.

4.2 Description

Current industrial practice for timing analysis is mainly driven by cost/benefit considerations compounded and industrial pragmatism making Measurement-Based (Deterministic) Timing Analysis, or MBDTA, the dominant state of practice [40]. MBDTA bases on capturing the "high-water mark" value in execution time across multiple runs of the program of interest and then add an engineering margin to it (e.g., 20%) margin, which compensates for the unknown.
In order to use MBDTA with sufficient confidence, the user must invest substantial effort to ensure that the worst-case execution conditions have been exercised or closely approximated [3]. This however is diluted in the overall testing campaign. Furthermore, software programs may have an inordinately large input space, which cannot be effectively exercised in a test campaign searching for the worst case. Usually, the input vectors used are those intended for functional testing, which may not be fit to incur the longest execution time, falling short on the side of the

worst-case path or in the coverage of the sources of jitter [4]. Further as platform complexity increases, the effort to test the longest path within a single test case exceeds practical and cost limitations. Furthermore, SoJ out of the control of the end user (such as cache state, jitter caused by floating point operations and the like) limit the reliability and confidence of this method.
For instance, the memory placement (and the resulting cache layouts) of software objects has been deemed a factor with high impact on execution time in the presence of cache memories as it determines how different addresses compete for cache space. Even if those addresses can be fixed so that inter-task side-effects can be avoided, it does not deal with services from the operating system that hold state whose execution time may then depend on execution history. Services using different memory locations as a result of past history, would cause different cache access patterns and different execution times to emerge depending on the type and cardinality of tasks included in the test. Another example is the floating-point unit, which for most architectures takes a variable latency depending on the particular values operated. This causes that depending on the particular values used in each experiment – which the user can hardly control – the program will suffer a variable impact due to floating point operations.
PROXIMA thesis is that the impact of the different SoJ should be managed by the timing analysis at the execution platform directly, without placing an untenable burden on the user. In the PROXIMA approach, this condition is either warranted without user intervention (hence transparently to the user) or the user is offered low-effort methods to achieve it. This particularly applies to low-level SoJ (e.g cache layout or conflicts in multicore shared resources), which arguably are the hardest to control in a multicore setting.
PROXIMA designs its hardware and system-software underneath the user application in such a way that the SoJ are controlled a priori. That is, their impact on execution time can assuredly be captured in the observation runs made at analysis time. This can be achieved in one of two ways.

- With deterministic upper bounding. By causing jittery resources to operate at their worst latency during analysis, effectively making them jitterless for the purposes of pWCET analysis.
- With probabilistic upper bounding. By randomizing the response time of the jittery resource such that i) there is a distinct discrete probability for every point in its latency range to arise in the observation runs made at analysis; and ii) the probability distribution of the jitter stays the same across analysis and operation, or the analysis distribution upper bounds that during operation.

The main PROXIMA timing analysis method, MBPTA, uses EVT to derive pWCET estimates of a program running on a computing platform. However, there is a fundamental challenge in applying EVT to solve the pWCET problem: EVT treats the system as a black box so that the projection it produces from the data it is

[4]As defined in D3.4 the sources of jitter (SoJ) – or sources of execution time variability (SETV) – are all those factors that affect the execution time of the program and that can vary across runs.

fed, solely holds for exactly that system. This requires the user to ensure that the data obtained from the system during analysis have an upper-bounding relation to those the system may produce during operation. Simple-minded application of EVT to analysis-time observations that do not warrant the above condition would fail to provide sound results for the operation-time behaviour of the system. Hence, EVT should be understood as a method that predicts the worst combined effects of phenomena individually observed during analysis but not to predict the occurrence of those never observed. Hence, as a precondition for use of EVT to solve the problem of measurement-based timing analysis it is required that that the sources of execution-time jitter phenomena observed during analysis have sufficient (upper-bounding) *representativeness* of their manifestation during operation [32] [7]. If this condition is warranted, then feeding these observations to EVT produces an approximation of the tail of the distribution of the worst-case timing behaviour that the program may exhibit in the operational life of the system.
This is better appreciated with an example taking memory placement as SoJ. Despite the user performing thousands of runs, there is no guarantee that the observed memory placements during the analysis phase (and the corresponding cache layouts) capture the worst memory placement that can appear during operation or at final integration stages when modifications of the system are overly expensive. Instead, with a random placement cache, the space of different cache layouts is naturally (and randomly) explored as the user performs more runs. Hence, with MBPTA the user focuses on providing a sufficiently large number of runs rather than to define test cases in which the cache layout changes across experiments. The latter is arguably hard to achieve and very costly.
As described in D3.4 Section 1.1 and summarized right above, the goal of MBPTA is to compute the pWCET function of a software program, using execution time measurements taken at *analysis time*, which are guaranteed to hold under the *operating time* conditions that may occur during the actual operational life of the system in which that program is embedded. In order to provide the evidence that the measurements taken at analysis occur under execution conditions worse than those that can arise during system operation, PROXIMA focuses on identifying the SoJ that impact program's execution time. Further PROXIMA proposes either to deterministically upper bound or time randomize the impact of those SoJ, so that an argument can be built on the fact that their impact is properly represented in the measurements captured during analysis.
Multicore processors introduce a new SoJ not present in single core processors: inter-core (multicore) contention. This contention arises when several tasks try to access a hardware shared resource at the same time, hence creating jitter in the execution time of program. While in the deliverables of WP1 it has been shown how hardware (i.e. the FPGA HW-randomized platform) can be changed so that this jitter is MBPTA compliant, in this section we focus on the FPGA SW-randomized platform (or FPGA-SWRand), in which the hardware is time-deterministic and randomization and deterministic upper bounding is performed at software level.
As presented in D1.1, the main SoJ in the FPGA-SWRand platform are as follows, see Figure 20.

- At single core level the floating-point unit takes a variable latency depending on the particular values operated. Also the first level instruction (IL1) and

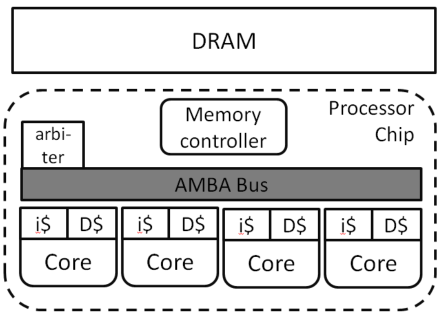


Figure 20: Reference FPGA-SWRandarchitecture

data (DL1) cache levels, as well as the instruction TLB (ITLB) and the data TLB (DTLB) are jittery.

- At the multicore level the contention in the access to the hardware shared resources, the bus and the memory controller, are the main *SoJ*. This is so because the L2 cache is partitioned so it generates no inter-core contention evictions between tasks simultaneously running. The DL1 is write-through, no write-allocate. The L2 is write-back, write allocate. In this platform the requests sent to the main AMBA AHB bus (or simply bus) are not split. Hence, the bus is locked all the time a request accesses the L2. If it misses in L2, the bus is locked until the request is solved in main memory and answered back.

Each of the SoJ has to be handled in a MBPTA-compliant way so that the entire architecture is MBPTA compliant.
Jittery FPU. There are resources whose execution time is jittery with no guarantees for their analysis-time jitter w.r.t. their deployment time jitter. Hence, in theory those resources are not MBPTA-compliant and hence, cannot be used in an architecture for which MBPTA pWCETs are to be obtained. However, we observe that some of those resources have an impact on execution time that is relatively low and can be easily upper bounded by padding (enlarging) the execution-time observations captured during the analysis phase. It is well known that some integer and floating-point (FP) operations may incur different latencies depending on the input values operated. For instance, integer/FP division takes a different latency depending on the particular values that are operated in several embedded architectures such as PowerPC P4080 [18] and Aeroflex-Gaisler LEON3 [5]. This can be handled with the execution time padding: the basic approach consists in counting, for every run j carried out during the analysis phase, the number of FP operations for each jitter type. In the FPGA-SWRand platform only the double precision division and square root are jittery. So for every run, it is required to count N^{ddiv} and N^{dsqrt}. The worst jitter suffered by each instruction is $jitter_{max} = 3$. The worst situation happens when all the measurements taken at analysis suffer their shortest latency (i.e. 15 and 23 cycles for the FDIVD and FSQRTD respectively) and during operation they suffer their worst latency (18 and 26 cycles respectively). This is captured by adding to each observation ri the value $jitter_{max}$ for every instruction of the mentioned types.

$$et_a^{i,pad} = et_a^{i,obs} + \left(N_a^{i,ddiv} + N_a^{i,dsqrt}\right) \times jitter^{max} \tag{7}$$

Single-core cache resources. Since different locations of the task's objects in memory (e.g., heap objects, libraries, system software services) lead to different cache placements, respectively, different execution times, then the modulo placement breaks MBPTA-compliant timing behaviour. In general, the location of those UoA objects cannot be controlled by the user, so there is no proof that the behaviour observed at analysis time upper bounds or corresponds to that at deployment time. This problem is tackled by means of software randomisation in its different incarnations [27, 28]. Software randomisation relies on placing memory objects in random locations so that their conflicts due to placement in cache are random (and with the same distribution) at analysis and deployment time, so that hits and misses occur with true probabilities and their impact in execution time is already captured by the measurements used by MBPTA. Thus, the user is released from having to deal with this *SoJ.*

L2 cache. The current solution for handling L2 cache consists in partitioning it across running tasks, so that inter-task evictions are prevented. This feature exists in the FPGA SW-rand and P4080 COTS platforms.

Bus and Memory controller. Other than the L2 cache, in the FPGA SW-rand multicore two other shared resources cause contention delay. These are the bus and the memory controller. Ensuring that the contention delay suffered by a task at analysis time upper bounds the contention delay that it can suffer during operation – as required by MBPTA – is hard to achieve.

4.3 Main evolutions since M18

The pTC model introduced in this section was not devised yet by M18. By that time solutions presented only focused on single-core effects leaving multicore contention for the second half of the project. Therefore, the pTC model is introduced next from scratch.

4.3.1 Goal and required properties

In all our experiments we consider one task under analysis (or *tua*), usually referred to as τ_a and several contender tasks, referred to as $c(\tau_a)$ or τ_b, τ_c and τ_d. τ_a is always a time-critical task for which a WCET estimate is to be derived.

The goal of our analysis section is then to determine how to trustworthily capture the impact of that multicore SoJ – and in particular the bus and the memory controller – on the pWCET estimates derived. The resulting multicore contention technique analysis has to have several properties as described below.

- MBPTA compliance. The proposed technique must be MBPTA compliant requiring minimum changes to the single-core MBPTA timing analysis approach. The technique must also work in conjunction with the solutions proposed to handle core sources of jitter (i.e. the FPU and the on-core cache memories).

- Time composability, if achieved, makes the WCET estimate derived for a task independent of the load the other tasks put on the multicore shared resources. Time composability is an essential property that allows incremental

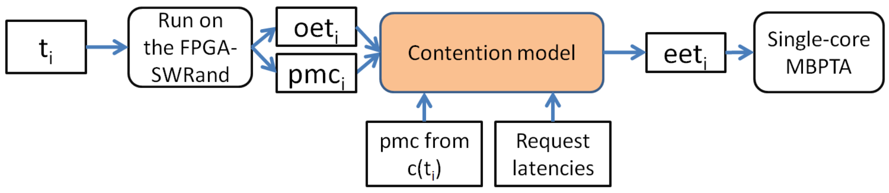


Figure 21: Schematical view of the proposed dpTC contention model

integration of applications, performing timing analysis of applications mostly in isolation, without the need of regression tests. Time composability also allows updating functionality during system operation without the need of analysing the entire task set but just those tasks that are updated.

- Reduced number of runs. In order to reduce experimentation cost, the technique should not need to perform runs (tests) of those tasks that can simultaneously run during operation. This may lead to high experimentation overhead requirements. Instead, runs should be performed of tasks in isolation collecting the required information so that by combining that information the model predict bounds to the impact of contention.

- PMC based. All the information required by the multicore contention technique from the running task should be obtainable without user intervention through performance monitoring counters (PMCs). This facilitates the applicability to the technique to real hardware.

- Integration into RVS. The developed multicore contention technique should be integrated into Rapita's RVS tool ecosystem.

We build on some information on the timing behaviour of those resources. In particular, both the bus and the memory controller have a time analysable arbitration policy, meaning that the maximum contention delay that a request from a task can suffer in the access to those resources is bounded. In particular, round-robin is used to arbitrate requests coming from different cores in each of those resources. For the memory controller a single request from each core can be queued and the arbitration policy is also time-analysable.
In our approach, during the analysis τ_a and each of the contenders are run in isolation. For each run r^i of task τ_a, we collect, through performance monitoring counters, τ_a's execution time oet^i (observed execution times). We further record i) access counts to the bus made by each task; and ii) we characterize the duration of each request type. With this information our analysis derives a bound (Δ_{cont}) for each run of a task a bound for τ_a that captures the impact of contention τ_a suffers in the bus and the memory controller from its contenders, $c(\tau_a)$. This results in an extended execution time $eet^i = oet^i + \Delta_{cont}$. The set of extended execution times is then given as input to the MBPTA analysis. It is worth noting that extended execution times capture (and hence expose to MBPTA) the variability caused by caches (via software randomization) and by the FPU and the multicore shared resources (via de padding). Hence MBPTA is fed with representative information to capture the impact of those SoJ.

Table 4: Request types and their latency

Type	**Lat**	**Description**
sh	$lat^{sh} = 1$	Store hit L2
lh	$lat^{lh} = 8$	Load miss in DL1-IL1 and hit in L2
lmn	$lat^{lmn} = 28$	Load miss L2 evicting a non-dirty line
smn	$lat^{smn} = 28$	Store miss L2 evicting a non-dirty line
lmd	$lat^{lmd} = 56 = 2x28$	Load miss L2 evicting a dirty line
smd	$lat^{smd} = 56 = 2x28$	Store miss L2 evicting a dirty line

4.3.2 Request types

We identify six types of request to the bus, each of them taking the bus different time. These are bus requests that are loads/stores that hit/miss in L2 cache. Further for the case of misses, since the L2 is write-back it can be the case that a dirty line needs to be evicted, in that case the request takes longer. Table 4 list the request types.
Ideally we would like to have a performance monitoring counters for the number of requests of each type made by the task. We refer to that ideal counter as n^{xxx} where xxx corresponds to one of the types in Table 4. However, in the target platform there is not a specific set of PMCs measuring those values. Instead with the existing PMC we approximate the number of requests of each type made by each task. In particular we use the PMCs listed in table 5. It is worth nothing that to approximate the number of requests of each type we have to take into account the request latency, that is upper bounding high latency requests first, which in our architecture are dirty misses (lmd and smd).

Table 5: PMCs in our reference architecture

Name	**Description**
pmc^{icm}	Bus reads caused by IL1 cache miss
pmc^{dcm}	Bus reads caused by DL1 cache miss
pmc^{st}	Writes to L2
pmc^{m}	Misses in the L2

Notice that $pmc^{icm} + pmc^{dcm} = n^{l}$, that is the number of loads performed to the L2 cache, with $n^{l} = n^{lh} + n^{lmn} + n^{lmd}$. Also note that the number of hits, n^{h}, could be obtained with said PMCs in our reference architecture with $n^{h} = (pmc^{icm} + pmc^{dcm} + pmc^{st}) - pmc^{m}$, that is the number of read/write accesses minus the number of L2 misses. Further, each store instruction accounted will cause a bus access because of the write-through policy of the DL1 cache. Finally, pmc^{m} does only count the number of direct misses, that is it does not count the number of memory accesses due to write backs.

4.3.3 Request latency

For our contention models, in addition to request count, we need request duration. The delays we derived are shown in the second column of Table 4. These cover the maximum latency (contention) that a request of each type can incur on other

requests. Note that contention delay is the time since a request is given the grant to the bus (cycle n), until the request is completed and the bus is relinquished (cycle $n + k$).
To derive the latency of each access type we have followed first the same process as described in [23]. We executed a benchmark performing a given type of bus operation as task under analysis (tua) against corunner tasks performing the same type of accesses. The corunners workloads are: $l2h$, $l2m$, $s2h$, $s2m$, that stand for L2 load hits ($l2h$), L2 load misses ($l2m$), L2 store hits ($s2h$) and L2 store misses ($s2m$) respectively. The tua used performs accesses of type $l2h$. To obtain the number of interference cycles per bus access type we apply equation 8 under the different workloads.

$$l_w = \left\lceil \frac{T_w - T_{isol}}{(N_c - 1) \times N_{req})} \right\rceil \tag{8}$$

We found this method to give an approximate value to the latencies we present in this document. In addition, to improve confidence we took advantage of the debugging system provided and prepared fine grained scenarios to measure the different access types.
The first step of this procedure is to setup the desired state in the cache hierarchy to measure the different access types. For example, to get the latency of a load that hits in L2, we warm the L2 cache with data we want to hit and perform accesses that clears the L1 set which would held the future access (miss in L1, hit in L2). Once we have the state ready, we set a breakpoint before the load that will trigger the desired latency. We tested the reliability of the measurements and took into account the delay experienced of the debugging system.

4.3.4 fTC model

The fully-time composable (fTC) model derives a WCET estimate that is an upper bound to the slowdown τ_a can suffer regardless of the load its contender tasks put on the shared resources. This requires the model making pessimistic assumptions in the following directions.

- Number of contenders. The model assumes that the number of contenders equals $Nc - 1$ where Nc is the number of cores – four in our platform.

- Request count and type: The model also assumes that for every request τ_a its corunners have a request of the worst type, i.e. causing the longest contention on it that in our architecture corresponds to xmd, i.e. a load or a store that misses in L2 and evicts a dirty line.

- Alignment: the model further assumes that the requests of the contender tasks align in the worst possible manner with each τ_a request. When round-robin the worst-case situation for a request from τ_a happens when it becomes ready right after τ_a's previous request is processed and by that time the tasks in the other cores already have a pending request. In this scenario, τ_a's request is stalled for $(Nc-1) \times lat^{bus}$ cycles, where Nc is the number of cores and lat^{bus} the time the contender request uses the bus.

Overall, with the fTC model we construct a set of enlarged execution time observations (EET) as follows:

$$\begin{aligned} eet^i &= oet^i + \Delta_{cont} \\ &= oet^i + t_a^{i_{bus}} \times (Nc - 1) \times l^{md} \end{aligned} \tag{9}$$

Where t_a^i is the total number of request of task τ_a performs in run r^i, that is $t_a = n_a^{lh} + n_a^{sh} + n_a^{lmn} + n_a^{smn} + n_a^{lmd} + n_a^{smd}$. Hence, the model that each request of τ_a is delayed $56 = lat^{smd} = lat^{lmd}$. The idea behind this approach is to account for the worst potential impact of contention a posteriori, after measurements are obtained, such in a way that can be properly factored in by the MBPTA process.

4.3.5 Deterministic pTC (dpTC) solution

While the fTC model derives time composable WCET bounds, with clear advantages on incremental integration, it may result in pessimistic WCET estimates. The solution presented in this part trades time composability to tight WCET estimates, which gives its name of (deterministic partially Time Composable (dpTC). With partial Time Composability [16] [17] the end user can yet enjoy benefits of incremental integration with small effort to assess time composability.

- Number of contenders. The fTC model can be easily adapted so that instead of $Nc-1$ the actual number of running tasks is used to predict the contention.
- Alignment. It is possible that a request from τ_a gets ready when a request from its contender τ_b is partially processed. As a result τ_a's request would not suffer a contention as long as τ_b's requests, but shorter. However, how the requests of τ_a and its contenders align during operation is hard, if at all possible, to predict. Hence, in our model we keep the worst-alignment assumption so that every request of a given contender τ_b is assumed to completely delay τ_a requests.
- Request count and type: Unlike previous, the number of requests of each type that a task performs is easy to track with PMCs. Further in our architecture, for a given task these values are not affected by integration, that is, do not vary regardless of the contender tasks. This offers a powerful solution to tight WCET estimates with a reasonable low impact on time composability.

In this first dpTC model we assume contender tasks can be derived an upper bound to the number of request of each type they can perform. Hence this model starts from a set of PMC readings for each contender task $pmc_b^{icm}, pmc_b^{dcm}, pmc_b^{st}, pmc_b^{m}$. From these values for each contender task we derive bounds to the requests of each type they can perform to the bus (i.e $n_b^{lh}, n_b^{sh}, n_b^{lmn}, n_b^{smn}, n_b^{lmd}, n_b^{smd}$). However, as we describe in this section we cannot derive exact values for the latter, but bounds to them. In doing so, we upper bound those request types with higher impact on contention (i.e. dirty misses) though this causes some underestimation of other types with less impact on contention (e.g. hits in DL1). Yet the overall results lead to an overestimation of the actual contention bounds.
We first derive an upper bound to the number of L2 misses that evict dirty lines. This is upper bounded by the minimum between the number of stores – which cause

lines to be dirty – and the number of L2 misses – which evict cache lines: $\hat{n}^{md} = min(pmc^{miss}, pmc^{store})$. This approximation may result in assuming some misses generate dirty evictions while in reality they do not, introducing some pessimism in the results. In particular, the lower bound to the number of misses that do not evict dirty lines is defined as $\check{n}^{mn} = pmc^{miss} - \hat{n}^{md}$. Hence, the model divides in misses and hits in such a way the result is not going to be optimistic.
The number of loads that hit in cache is upper bounded by the minimum among the number of total hits and the number of loads to the L2: $\hat{n}^{lh} = min(n^{hit}, pmc^{load})$. This results in lower bound of stores that hit in cache $\check{n}^{sh} = n^{hit} - \hat{n}^{lh}$
Once bounds to τ_b accesses have been computed, the model proceeds by "pairing" each request from τ_a which one request of τ_b from worst to best latency:

- The number of request from task τ_b of type miss dirty ($\hat{n}^{md}$), i.e the type with highest contention impact, that contend with requests of τ_a, t_a, is given by: $\hat{c}^{md} = min(t_a, \hat{n}^{md})$.
- The remaining τ_a's request $t'_a = max(0, t_a - \hat{c}^{md})$ contend with $\check{n}^{mn}$ (second most impacting type) requests of τ_b: $\check{c}^{mn} = min(t'_a, \check{n}^{mn})$.
- The remaining τ_a's request $t''_a = max(0, t'_a - \check{c}^{mn})$ contend with $\hat{n}^{lh}$ (third most impacting type) requests of τ_b:$\hat{c}^{lh} = min(t''_a, \hat{n}^{lh})$.
- The remaining τ_a's request $t'''_a = max(0, t''_a - \hat{c}^{lh})$ contend with $\check{n}^{sh}$ (fourth most impacting type) requests of τ_b: $\check{c}^{sh} = min(t'''_a, \check{n}^{sh})$.
- The remaining τ_a's request $t''''_a = max(0, t'''_a - \check{c}^{sh})$ do not contend with any request of τ_b.

The resulting contention dpTC is the result of assuming that each of these contentions among τ_a and its contender τ_b are aligned in the worst way, causing a contention delay as long as each τ_b request:

$$\begin{aligned}\Delta^{dpTC}_{muc} &= (\hat{c}^{md} \times l^{md}) + (\check{c}^{mn} \times l^{mn}) \\ &\quad (\hat{c}^{lh} \times l^{lh}) + (\check{c}^{sh} \times l^{sh}) \end{aligned} \tag{10}$$

This very same process is repeated for the other potential contender tasks τ_c and τ_d.

4.3.6 Probabilistic pTC (ppTC) solution

For the FPGA we have built two models to bound multicore contention: the fully-time composable (or fTC) and the deterministic partially time-composable (or dpTC). These models derive a deterministic bound (i.e. a single value) upper bounding the contention that in a run a task can suffer from its contender tasks. In both cases we have assumed that a bound can be derived to the number of accesses that contenders perform is fixed. This bound is passed as input to the fTC and dpTC models.
In this section we explore the case in which the contender tasks, similarly to the task under analysis (τ_a) are affected by their memory mapping. The latter affects

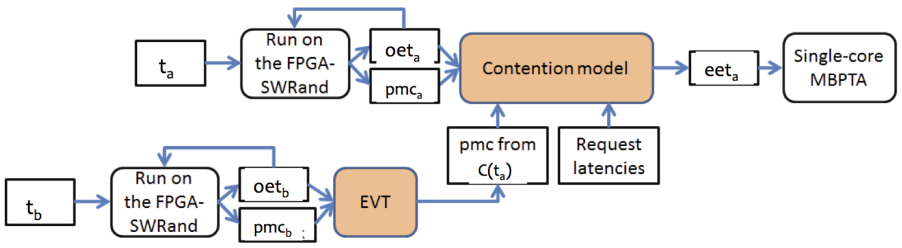


Figure 22: Skematical view of the first proposed ppTC contention model

contender tasks cache layout and hence the number of hits/misses in cache. This ultimately affects contender tasks' number of accesses to the bus.
In this scenario when we execute a contender task in isolation several times we do not see a single access count to the bus, but a distribution. From that distribution one has to derive an upper bound to the number of accesses the contender task can perform to the bus. This is provided as input to the dpTC method developed in the previous section. In this section we present early versions of two approaches to handle this. It is worth clarifying that these solutions have not been evaluated with the case studies, so they remain as early versions. Further the solutions do not cover the impact of path coverage. In that sense, we stick to the paths exercised during the analysis.
EVT to bound access counts. The basic idea of the approach in this section is to use EVT to derive bounds to access counts of each type for the contender tasks. To that end we perform runs of the contender task in isolation. In each run we collect access counts for each access type. On those observations we apply EVT to predict the probability that the task has a higher access count than that observed, see Figure 22. We then select a cut-off probability and take as access count for that task the predicted value for that probability.
Monte-carlo simulations. The problem of the previous approach is that EVT is applied twice, which can lead to pessimistic WCET estimates. First, EVT is applied to derive – for a given cutoff probability – a bound to contender access counts. Second, it is applied in the normal MBPTA process to derive bounds to execution time.
In this section, we do not derive a bound to contender access counts. Instead, we pair each run in isolation of τ_a ($\tau_{a,i}$) with a run in isolation of each of its contenders ($\tau_{b,j}, \tau_{c,k}, \tau_{d,l}$). That is, we randomly select one of the runs of τ_a and its contenders, see Figure 23. This process mimics the reality in which every run of τ_a (under a particular memory mapping) can share the multicore resources with any run of their contenders (which can be under any random memory mapping).
The next step is to apply the deterministic dpTC method taking as input the number of accesses performed by each task in the considered runs resulting in an extended execution time (eet_i). This process is repeated as many times as execution times need to be collected for τ_a. This process pairs the runs of the different tasks in a random manner, thus resulting in several *monte carlo simulations*.

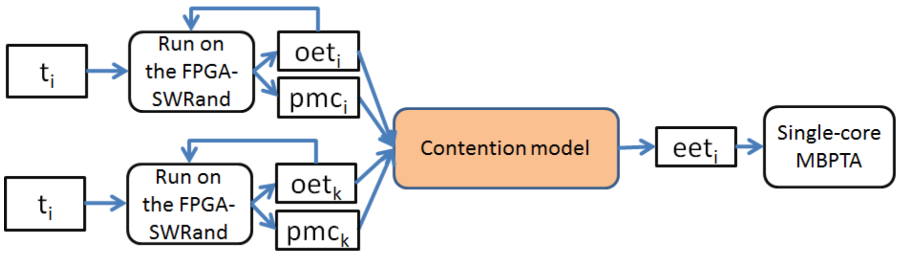


Figure 23: Schematically view of the second proposed ppTC contention model

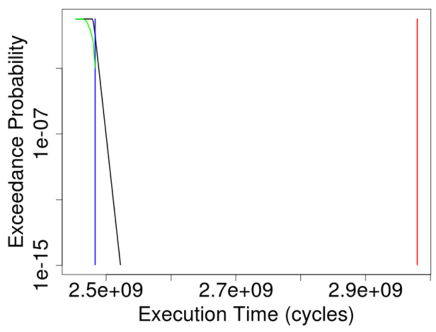


Figure 24: MBPTA projection when running one EEMBC Automotive in the the FPGA-SWRand single core

4.4 Advantages and limitations

The central tenet in our timing analysis techniques is to get accurate and cost-effective timing analysis. This requires representative testing so that the user can ensure that test inputs and test conditions exercise each component adequately. This contributes to provide confidence that all important sources of jitter have been observed without introducing additional conditions that are infeasible in practice.
Contention model. Our analysis model for the FPGA captures the variability caused by major SoJ (caches, FPU and multicore resources). It just requires the end user to perform runs in isolation collecting PMCs in each run. By operating on the different PMCs values we wisely pad the observed execution times resulting in extended ones that capture the impact of SoJ and hence exposes them to MBPTA that can upper bound them. It is noted that with the current implementation we have to infer and split PMCs in useful information for the models developed. By having more precise counters for the different access types we could tighten even more the contention calculated.

4.5 Numerical evaluation

The baseline for our multicore contention model are i) the software-randomization approach that makes the jitter of the cache MBPTA compliant; ii) the deterministic padding for the FPU that makes its jitter MBPTA compliant. Building on top of this single-core MBPTA-compliant processor we checked that for all EEMBC Automotive we pass the independence and identical distribution tests. As an example of obtained result, the curve (diagonal line) in Figure 24 shows the pWCET obtained for one EEMBC automotive benchmark and its comparison with current practice using a factor of 20% (red line) on top of the maximum observed value

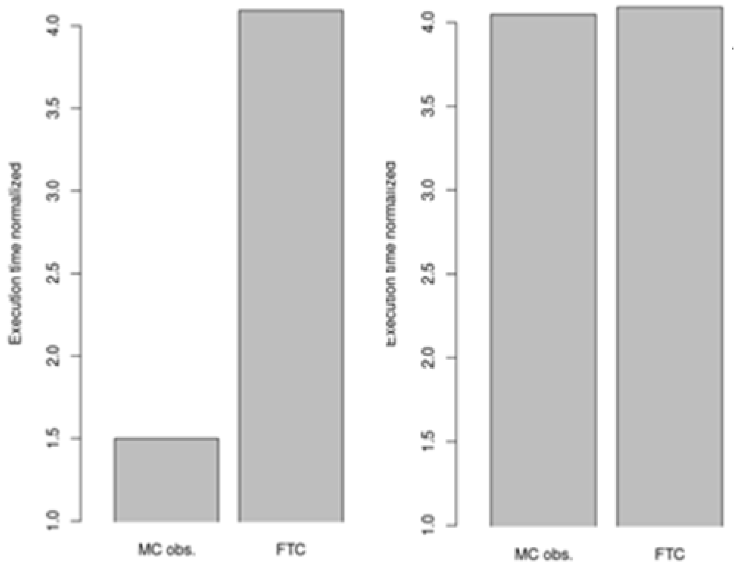


Figure 25: MBPTA projection when running one EEMBC Automotive in the FPGA-SWRand single core

(blue line).

4.5.1 fTC

We start the evaluation of the contention model by analysing the fTC model. For this experiment we run a reference synthetic application [5] in three different scenarios. First, in isolation; second, against three stressing programs that are constantly performing loads that miss in the L2; and third, against three stressing programs that constantly perform stores that miss in the L2 and cause dirty evictions.

The left two columns in Figure 25 show the execution time of the synthetic application when it runs against the load stressing kernel (MC obs) and the observed execution time plus the contention bound derived with the fTC model. We observe that the model provides a predicted time much higher than the observed execution time. However, this occurs because the fTC model assumes that the contender tasks perform accesses of the worst type, i.e. dirty misses, and in this case the misses generated by the contender task are not dirty.

In the second experiment, we observe that when the contenders actually perform dirty misses, which actually represents the worst-case scenario, the observed execution times for the synthetic application approach quite closely that predicted by the model.

We conclude that in our architecture the fTC model may easily lead to overly pessimistic estimates. A better balance between time predictability and tightness is required, as the one provided by the pTC models. On the advantages side, the fTC model perfectly integrates with the single-core MBPTA model and effectively controls the different SoJ identified.

[5]The synthetic application consists of several functions accessed sequentially in a loop one hundred times. Each function has a random size of instructions mixing arithmetic and read/write operations.

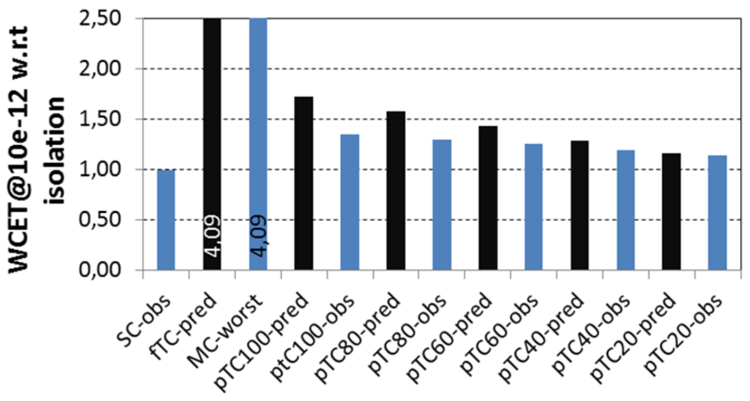


Figure 26: MBPTA projection when running one EEMBC Automotive in the the FPGA-SWRand single core

4.5.2 Deterministic pTC (dpTC)

For the evaluation of the dpTC model, we take as reference our synthetic application and run it against one contender in each experiment. The set of contenders used are as follows. First, a contender causing dirty misses to the L2 constantly; a contender with as many accesses to the bus as the synthetic application (100); a contender that performs 80% of the accesses of the synthetic example; and so on so forth down to 20%. With this example we want to show how the dpTC model adapts to the load that contenders put in the bus.

Results are shown in Figure 26, where all results are normalized to the execution time of the synthetic example in isolation (*SC-obs* in the first bar). When we run the synthetic example against the worst-contender, i.e. the one constantly causing dirty misses, the fTC model is tight. However, for all the other contenders (100, 80, 60, 40 and 20) the fTC model results in the same pessimistic pWCET estimate that do not adapts to the load of the contenders.

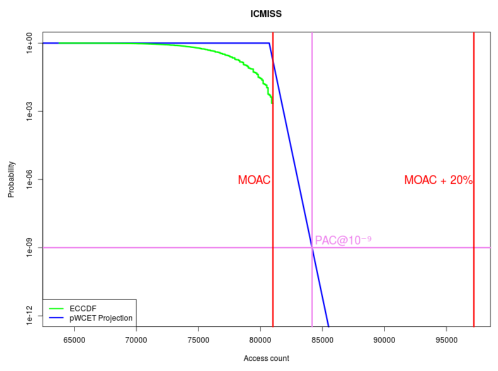
Instead, the dpTC model, provides pWCET estimates that adapt to the load of the corunner (see blue bars labelled as $pTC - XX\% - pred$). Further, for assessing the tightness of the model, we compare this prediction with the observed execution time of the synthetic application when it actually runs against the kernel that puts $XX\%$ load on the bus. We observe that for each pair $(pTC - XX\% - pred, ptC - XX\% - obs)$ the predicted value tightly upper bounds the observed one.

Similarly to the fTC, the dpTC model upper bounds the combined jitter of the caches, the FPU and the bus and memory with MBPTA. Its application requires small changes to the MBPTA canonical application – basically processing PMC readings and padding execution times. This simplicity has allowed its integration in the RVS framework as presented in D3.11. Further, the dpTC model reduces the pessimism of the fTC model.

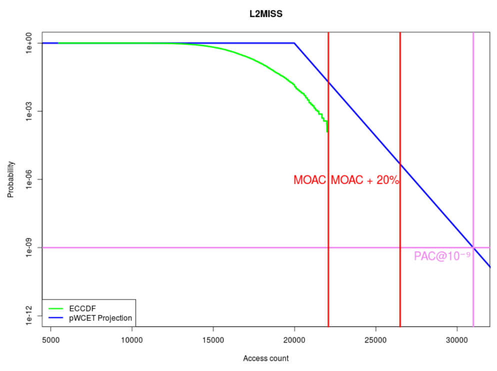
4.5.3 Probabilistic pTC (ppTC)

For ppTC, we take the synthetic as reference application under analysis and contender application.

EVT to bound access counts. For the contender applications we use EVT to derive a bound to the access count of each request type to the bus. Figure 27 shows the distribution obtained from executing the contender tasks 1,000 times in isolation plus the corresponding EVT projection. Note that for this experiment



(a) instruction cache misses



(c) L2 cache misses

Figure 27: Observed Access Counts for each PMC and corresponding EVT projections

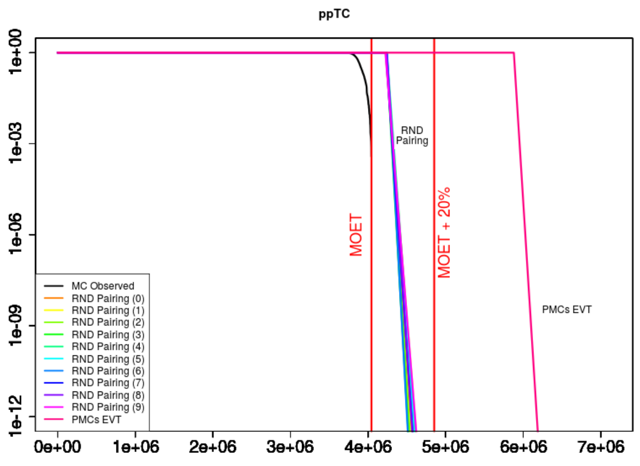


Figure 28: Comparison of ppTC approaches

we only randomized code, so the the data cache misses remain constant in all experiments.

We display the distribution of access counts. Following the same principle of *fudge factor* followed in current practice for timing, we show the result of simply taking as bound for the access count 20% over the maximum observed access count, called $(MOAC + 20\%)$. We observe that while for icmisses the EVT projection is much tighter that the $MOAC + 20\%$ bound, for L2 it is not. For instance, for a cutoff probability of 10^{-9} EVT results in a bound around 40% higher than MOAC. Hence, blindly taking as bound a value $MOAC + 20\%$ can result in undesired results.

Comparison. The comparison results in Figure 28 shows that the first ppTC model results in bounds much higher than the latter. This comes from the reason that it applies EVT at two levels – the access count and the execution time – whose pessimism accumulates.

For the latter model we derive several curves each of them each corresponding to a monte-carlo simulation. For all simulations we observe convergence in obtained pWCET curves.

Finally, the result taking as bound for the access count $MOAC + 20\%$ fits in the middle. However, this method lacks scientific reasoning.

5 FBI-VICI: a PROXIMA answer for multicore interferences

Participants: UoY

5.1 Executive Summary

An ongoing problem in Real Time Systems is the transition to multicore systems, and the effects that interference between cores can implact the Worst Case Execution Time of a task. Since D3.4, PROXIMA has developed a solution to this problem: Forecast Based Interference Variable Inter-Core Interference (FBI-VICI) analysis, which is designed to model the effects on a task of a variable amount of interference generated by competing tasks running simultaneously on other cores. VICI analysis has been applied to synthetic benchmarks by academic partners to the AURIX and P4080 platforms. In addition, industrial partners have also applied the analysis to their codebases for these platforms, demonstrating the applicability of this approach.
This approach is not restricted to the specific platforms for which it has been devised and integrated (P4080 and AURIX), and it can be potentially ported to other hardware platforms. However, how to port it to other platforms needs to be investigated for an appropriate adaptation. Currently this method has been successfully integrated into RVS and the case studies on top of the AURIX and P4080 platforms.

5.2 Description

Inter-core interferences stem as multiple tasks may share portions of the hardware or software states on a platform. Concurrent accesses to the same resources therefore need to be arbitrated, resulting in a delay for some contenders. Similarly, timing-relevant portion of a resource state, e.g. the contents of the cache, may be altered by a contender in an unexpected manner. The co-runners of a Unit of Analysis (UoA) contribute to the expression of interferences during analysis. Various techniques have been proposed to isolate the performance of tasks in a multicore system [9, 24]. Those may not be available on all platforms, or their implementation may still result in inter-core interferences [37]. Specific analysis methods are thus required to account for inter-core interferences. In the context of COTS architectures, the analysis should also accommodate for the limited information available regarding the inner-workings of shared resources, e.g. the CoreNet fabric on the P4080 [18].
This section presents an approach to the analysis of the impact of inter-core interferences and shared resources on a UoA. The Variable Inter-Core Interference (VICI) analysis is platform agnostic and only bears requirements on the nature of the observations collected for analysis. The results of the VICI analysis, presented as a interference margin, are factored into the timing estimates derived by or the observations fed to the PROXIMA baseline analysis (Section **??**) to account for the effects of inter-core interferences. Observations relevant to the behaviour of both the analysed task and its contenders are collected to build a parametric interference

model capturing the behaviour of the UoA under observed or predicted worst-case conditions.
Section 5.3 presents the overall analysis process, focusing on the evolutions of the analysis since its introduction in D3.4. We then discuss the strength and weaknesses of the approach as observed during our experiments (Section 5.4). A numerical evaluation of the method is finally provided in Section 5.5).

5.3 Main evolutions since M18

The contribution of inter-core interferences in the VICI analysis is evaluated through the construction of an interference model for the UoA. The resulting multi-variate model predicts the contribution of inter-core interferences on the execution time of the UoA from interference-relevant factors, beyond the sole UoA execution time. To build an understanding of the impact of interferences on the UoA, those additional factors should be collected during a careful exploration of possible interference scenarios. In the following, we briefly introduce the analysis process, its requirements, and its evolution since its introduction in D3.4:

1. Select interference-relevant factors which will constitute the inputs to the multi-variate interference model.
2. Build the interference model for the UoA through the collection of observations under variable interference configurations.
3. Extract a specific safety margin from the interference model, through predicted or observed interference scenarios.

Step 0. Analysis requirements Each observation in the context of the VICI analysis should hold more than just the execution time of the analysed task. The Performance Monitoring Counters (PMCs) available on each platform can provide information on the context under which each observation is obtained. In particular, PMCs can be used to understand how interferences cause variations in the behaviour of the UoA. In addition to a careful selection of the events tracked by the analysis, contenders are required to observe the UoA under different interference loads.
The different steps of the VICI analysis, as discussed in D3.4, have been originally prototyped for the Aurix COTS. It has since been ported to the different PROXIMA multicore platforms. This includes the implementation for each platform of specific instrumentation routines to capture the available PMCs, and the development of prototype contenders to exercise the specific shared resources identified in D3.1. Preliminary results have been collected on all platforms, leading the current refinements to the contenders and analysis tools. Due to the platform-agnostic nature of the analysis, refinements of the analysis process or results benefit to all platforms. The complete analysis has been successfully applied within the context of the PROXIMA tool chain to the Aurix COTS automotive case study (D4.8). Initial results have also been collected during the familiarization case study with DENSO on the Aurix COTS (D4.9), the internal railway case study on the P4080 COTS (D4.8), and the PROXIMA manycore hardware platform (D3.9).

Step 1. Select interference-relevant analysis factors The construction of an interference model relies first on the identification of factors best suited to characterise the execution time of a task subject to inter-task interferences, e.g. memory stalls or bus accesses. These are used to drive the main data collection phase to provide inputs to the multi-variate interference model. The interference model predicts, from selected or observed values of the interference-relevant factors, the impact of interferences on a task as a multiplier to its execution time. The selection of pertinent factors to train and query the model is thus crucial to the precision of the analysis.
An initial set of observations is collected capturing data across all available PMCs on the platform. A well-proven method, Principal Components Analysis (PCA), is then used to identify Principal Components in the observations, i.e. weighted groups of factors modelling the main sources of variability in the input dataset. A number of factors within the Principal Components are then selected to capture the effect of interferences.
The factor selection process from the PCA results in D3.4 has been refined. In particular, the metric used to evaluate the relevance of factors now takes into account the direct correlation between the observed factors and the execution time of the UoA. Multiple factors may thus be selected within a same Principal Component to both capture the main axes of variation in the data and minor variations along each axis. This further allows the analysis to capture factors related to both inter-core and intra-task variability.
Further evaluations have been conducted to understand the impact of the number of observations fed to the factor selection step. As a result, the analysis tools can prompt the user of additional data when no factor is identified as representative. This comes in addition to the minimum number of data required by the analysis, a function of the number of candidate factors and available entries in the UoA input vector [33],

Step 2. Build interference model Once the relevant factors have been identified, this steps builds the inter-core interference model. A set of observations, focused on observing variability alongside the selected factors, is used to train a forecast, interference model specific to the observed UoA. Neural networks are trained to learn how the inputs, rates of interferences along the selected factors, impact the output of the model, the variation in observed execution time. Using an Ensemble modelling approach, multiple models are constructed and the consensus amongst models provides the output of the interference models.
The construction of the ensemble model has been thoroughly revised since its original definition in D3.4 through application of the methods to different platforms. The training process includes a self-calibration phase to validate the accuracy of the output model. The analysis can then identify multi-modal datasets, e.g. resulting from distinct paths in the observations, and either trigger the creation of sub-models and an appropriate categorisation neural network, or prompt the user for more data of a specific nature.
Further evaluation has been conducted on the Aurix COTS platform to understand the impact of interferences on randomised application through Static Software Randomisation (SSR) on the results of the analysis. This led to the definition of refined approach to reduce the number of required builds and observations by the analy-

sis. More details are included in D4.8, where the results of the application of the PROXIMA Aurix toolchain are presented.

Step 3. Extract specific safety margin from interference model The interference model is a multi-variate model which can forecast the effect of interferences on the execution time of a UoA given rates of interference as captured by the relevant factors. Such interference rates can either be captured from a set of observations on the deployed system, or constrained by the underlying platform, e.g. a bounded miss rate for contenders. The effect of the interferences is predicted as a multiplier to be factored into observations fed into the baseline PROXIMA analysis or into the resulting timing estimates.
The extraction has further been extended to support different methods to extract the worst-case interferences suffered by a UoA according to its model. Those provide for an upper-bound of the interference multiplier in the absence of knowledge regarding the constraints or contenders exercised in the deployed system. The resulting estimates although composable, i.e. valid irrespective of the contenders exercised at runtime, may be more pessimistic.

5.4 Advantages and limitations

The VICI analysis offers a platform-agnostic solution to capture the impact of intercore interferences on an analysed task. The method can be applied to a platform without the need for a precise model of its shared resources or upper-bounds on their behaviour. Interferences and their effects are automatically captured by the approach
Shared resources need only to be identified as such and appropriate contenders synthesized to exercise them. While those have been in parts produced manually, the analysis does not require that they trigger absolute worst-case behaviours and they could be derived through automated benchmarks generators. The PCA analysis can assess whether or not the synthetic contenders trigger sufficient variability or not.
At every step, during the selection of the relevant factors or the construction of the interference model, the analysis can provide feedback to the end user. This includes a self-evaluation of the accuracy of the produced models, and additional requirements in terms of covered interference scenarios.
The method relies on low granularity instrumentation, requiring only end-to-end measurements. This offsets the need to collect additional data at each instrumentation point to capture information beyond the timing of the analysed task.
For interference-sensitive applications, the platform should also provide sufficient means to characterise the behaviour of contending tasks or the impact of interferences.
A lack of PMCs, assorted registers, or constraints on the combinations of observed events, may hinder the ability of the analysis to precisely model interferences. Those issues have been encountered in the different PROXIMA platforms. Furthermore, both the factor selection process and the interference model tend to favour PMCs capturing blocking delays instead of events.
With an increasing number of shared resources, as the analysed platforms become more complex, the number of inputs to the multi-variate interference may increase.

This would require the collection of more data to ensure sufficient coverage along the selected factors, and more complex queries both for the end-user and the proposed heuristics.

5.5 Numerical evaluation: AURIX

We focus our evaluation of the VICI analysis on the COTS Aurix multicore platform. The VICI analysis has been applied to all COTS multicore platforms in PROXIMA as a proof of concept. However the availability of the platform and the maturity of the tool chain led to a more refined analysis and contenders on the Aurix. We present our results for a subset of the TacleBench WCET benchmarks [15]. The following graphs focus on the results of the VICI analysis. They include only empirical, observed execution time distributions, without any tail extension or software-based randomisation. Results are presented in D4.8 for when the analysis was applied to the industrial use cases.
The code of analysed application and the Erika OS are mapped into the scratchpad local to each core. Inter-core interferences are generated by concurrent accesses to data stored in the shared memory RAM. The interference level and access patterns exercised by contenders are randomised after each execution of the analysed task. Contenders can be set to run on all cores while observations of the analysed task are collected. The VICI analysis has been configured to produced results with an acceptable error of $\pm 10\%$. 3 PMCs out of a maximum of 9 available on the Aurix have been used to build the interference models.
We present results for the *bsort100*, *matmult* and *missile* benchmarks respectively in Figures 29, 30 and 32. In each case, we present the observations obtained without and with contenders (respectively *No Contention* and *Contention*), with a maximised interference level (*MAX*) and the result of the VICI analysis applied to the contention-free observations. The latter would then be fed to the EVT-based analysis to derive a pWCET including the effects of interferences.

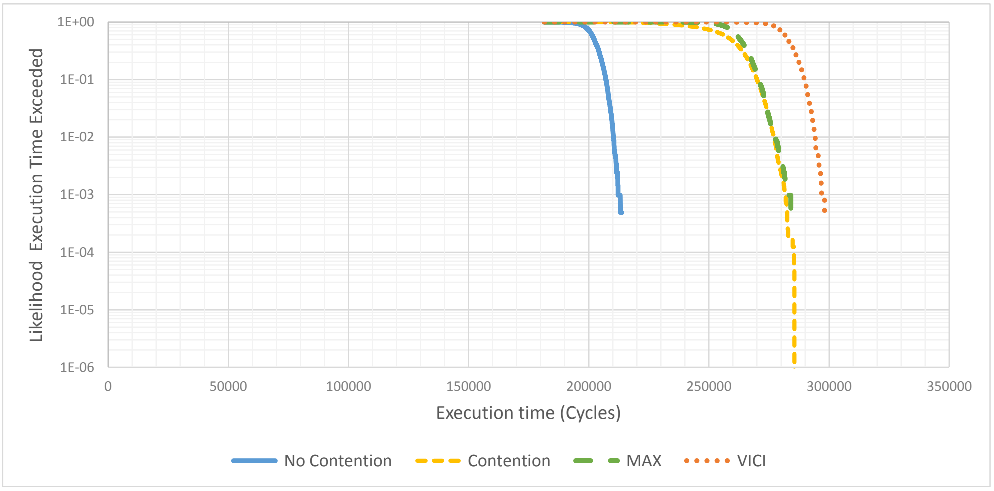


Figure 29: Overview of the VICI analysis results on the *bsort100* benchmark running on Core 0.

bsort100 illustrates the standard behaviour of most applications on the Aurix. The introduction of contenders for the shared resources increase the execution

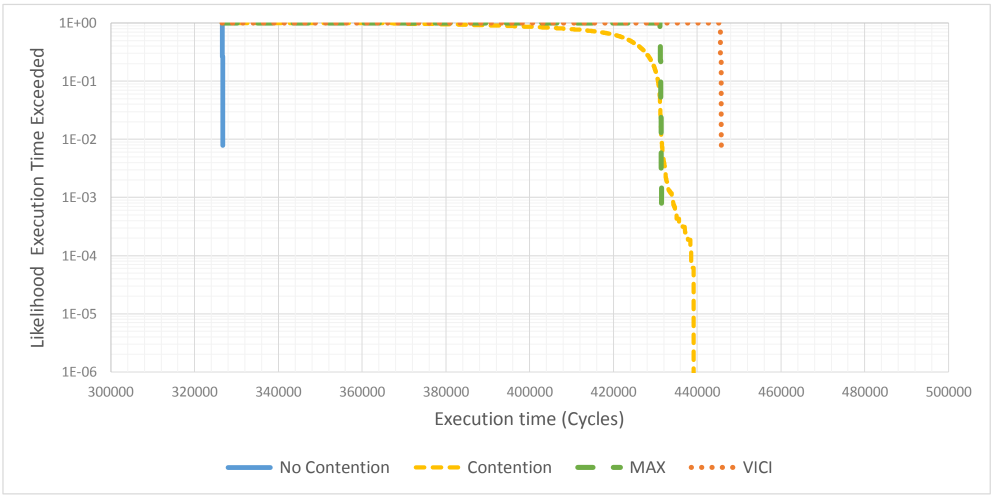


Figure 30: Overview of the VICI analysis results on the *matmult* benchmark running on Core 1.

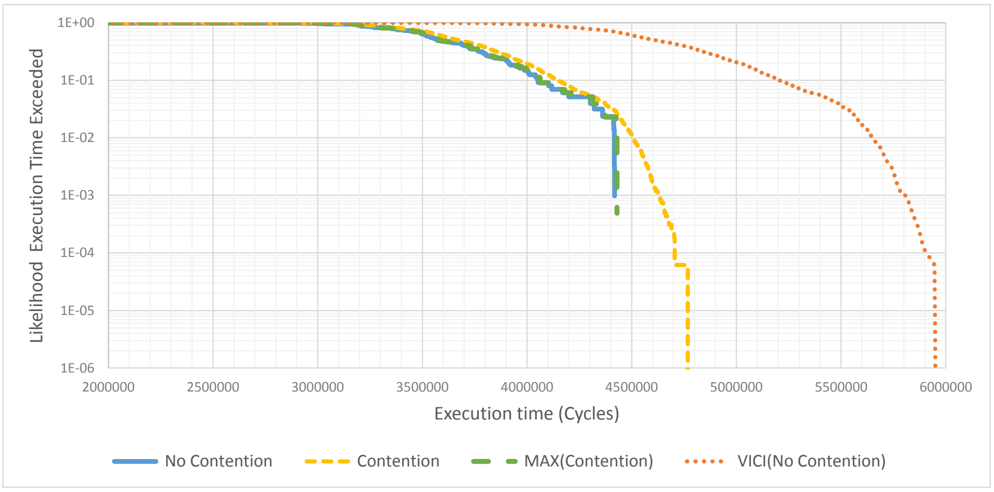


Figure 31: The VICI analysis results on the *dijkstra* benchmark running on Core 0

of the analysed task. Higher level of interferences result in a higher likelihood to observe higher execution times than a random selection of interference loads. This is shown by higher execution times at the same exceedance threshold, e.g. around 50%. The VICI analysis takes into account the effects of interferences to derive a interference factor. Said factor is applied to observations collected without interference to compute an upper-bound of the possible impact of inter-core interferences. *matmult* exhibits a similar behaviour. The matrix multiplication benchmark has a very regular access pattern. The effect of interferences is then not maximised with contenders' access rate to the shared resources; contenders conflict with each others for the shared resource. This illustrates the importance of the evaluation of a task under different scenarios as part of the VICI analysis process. *dijkstra* provides a worst-case behaviour for the VICI analysis, where there are multiple paths with significantly different behaviour; on one path no contention arises, and on another contension occurs in a similar manner to *matmult*. In this case, the analysis produces an upper bound, but with a decreased accuracy for the

amount of data provided. With additional measurements the accuracy could be increased.
The *missile* benchmark illustrates the behaviour of the analysis under multi-modal distributions. The benchmarks comprises multiple path tasks, each with its respective behaviour regarding timing and interferences. The VICI analysis derives a sound-upper bound of the impact of interferences across all paths. The shortest path in the missile benchmark, as shown in Figure 33, is relatively more sensitive to interferences than its longer counterpart. The safety margin extracted from the interference model is an absolute upper-bound across the whole model, including the smaller, interference-sensitive path. This results in pessimism when the safety margin is applied to the global distribution. While the results remain sound, the discrepancy is captured by the analysis tool; the analysis generates appropriate warnings that the two modes could be analysed separately.

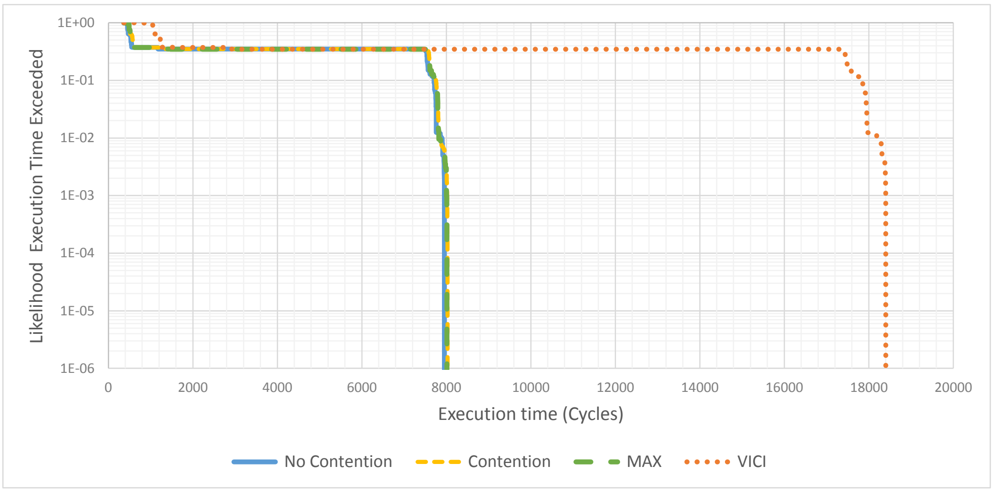


Figure 32: Overview of the VICI analysis results on the *missile* benchmark running on Core 1.

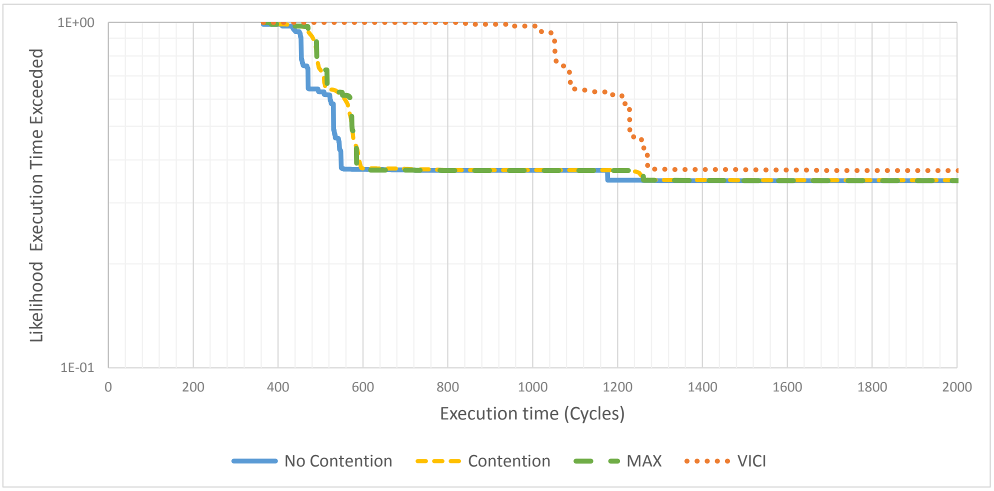


Figure 33: Impact of inter-core interferences on the shortest path of the *missile* benchmark running on Core 1.

5.6 Numerical Evaluation: P4080

In addition to the main results from the AURIX platform, we also provide results from the P4080 platform under PikeOS. Memory management is handled via PikeOS which provides extensive facilities for allocating memory segments. A major factor in the results for the P4080 is that the platform provides a large uncontested instruction/data cache. This feature reduces the number of memory accesses that require a contested resource. As memory accesses are a significant component of multicore interference, the overall variability due to VICI effects is decreased. However, the VICI analysis is still useful as it provides statistical guarantees on the magnitude of VICI effects.
Results are presented for a subset of the TacleBench suite, in the same form as for the Aurix. That is we present the observations obtained without and with contenders (respectively *No Contention* and *Contention*), with a maximised interference level (*MAX*) and the result of the VICI analysis applied to the contention-free observations. The result of the VICI analysis applied to the contention-free observations would then be used for applications requiring a WCET estimate.

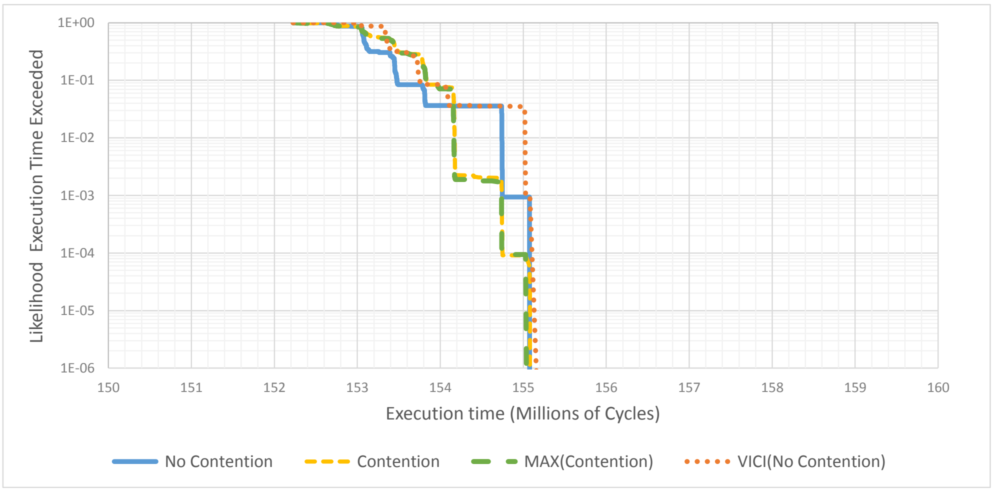


Figure 34: The VICI analysis results on the *matmult* benchmark running on the P4080.

Figure 34 gives an overview of the execution of the Matmult benchmark. As the benchmark is contained within the P4080's cache, we see that VICI effects account for a very low proportion of the tasks execution time. Counterintuitively, in certain, rare configurations, we observe that VICI effects are capable of slightly decreasing the execution time of the task. However, in all cases, the VICI model delivers a sound upper bound of the task, and apart from in the rare cases of interference decreasing the execution time, tracks the effect of contention very closely.
Figure 35 shows the same results for te Dijkstra benchmark. As with Matmult, there is very little interference from competitor tasks due to the benchmark being contained within the P4080's core local cache. Similarly, we observe that in all cases the VICI model delivers a sound upper bound of the task.
The bsort100 benchmark, shown in Figure 36, illustrates a rare example where contenders result in a slight, but measurable, decrease in execution time. While the reasons for this are unknown, it is likely due to the contenders causing a preloading

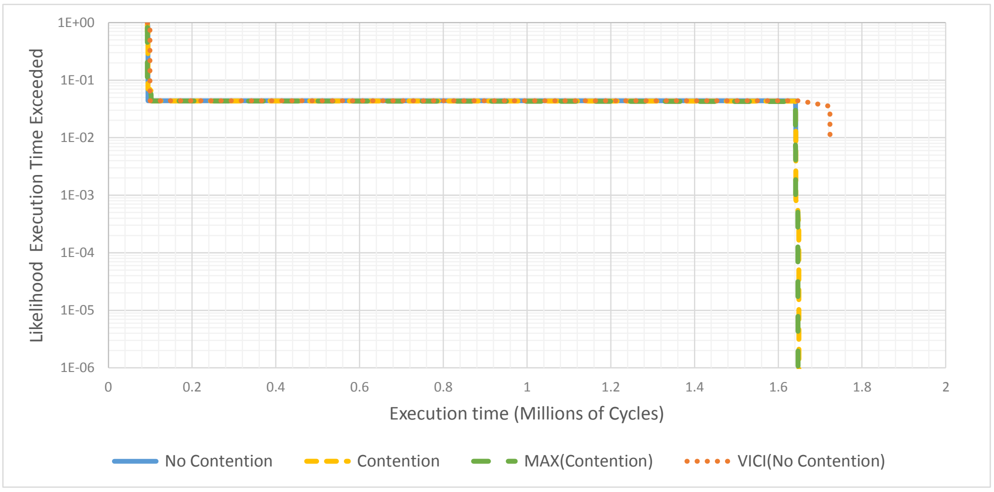


Figure 35: The VICI analysis results on the *dijkstra* benchmark running on the P4080.

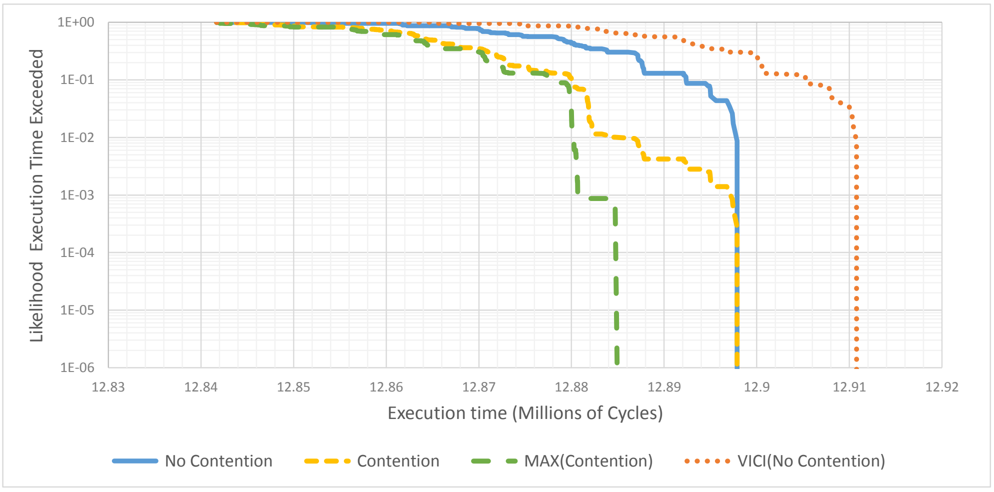


Figure 36: The VICI analysis results on the *bsort100* benchmark running on the P4080.

effect on the hardware, allowing the small amount of data that bsort100 fetches from memory to be fetched slightly faster, and this not being balanced out by negative contention effects due to bsort100 fitting within the cache. As with other examples, the VICI model again delivers a sound upper bound.

6 Probabilistic approaches from real-time to mixed-criticality systems: before and after PROXIMA

The first probabilistic approaches [4, 19, 36] for real-time systems have been proposed in the late 90s to support the *soft real-time constraints* satisfaction. Even if those approaches were using the commonly accepted hypothesis that larger values of the execution times of a program have lower probability of appearance (see Figure 6), these first results concern average timing behaviours. Indeed the real-time community considers at that moment that the probabilistic approaches are only able to ensure constraints that are defined using a ratio of non-satisfied timing constraints within a given time interval (mainly multimedia systems).

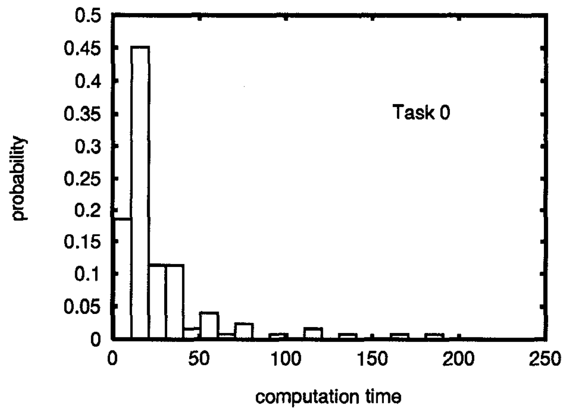


Figure 37: The probability execution time associated to a program (task) in [36]

Few years later two seminal papers [13, 14] introduce almost simultaneously two major concepts for the probabilistic approaches:

- **The worst execution times of a program are probably rare events** [14]: the first utilization of Extreme Value Theory (EVT) in the context of execution times estimation supports the hypothesis that the large values for the execution times of a program have extremely low probability of appearance.
- **The probabilistic analyses may cover worst case scenarios**: the first probabilistic worst case reasoning considers the program instances with the (proved) largest response time to provide probabilistic upper bounds on the response time of any instance.

The real-time community will wait for the first PROARTIS papers to measure the impact of these two concepts: probabilistic approaches may be used in the context of *hard real-time industries* like avionics. The appearance of mixed-criticality concept [38] increased the interest of the real-time community for probabilistic approaches as those approaches are until today the unique WCET estimation method for associating probabilities to different values for the execution times of a program. TODO: The link between MCC and pWCET is tenuous. For example in MCC you could use HWM and rapitime The PROXIMA project results complete the uniqueness of

the probabilistic timing analyses by providing arguments in favor of the certification of these approaches as well as on the integration within commercial tools, assessment against a wide variety of case studies, and integration with randomized HW/SW platforms.
Besides this positive (and natural) evolution, the PROXIMA project improves today the understanding of the probabilistic approaches by

1. clarifying the misunderstandings on the "supposed" limitations of probabilistic approaches with respect to the notion of independence;
2. providing a solid definition for the pWCET of a program;
3. identifying the main classes of open problems after PROXIMA. These open problems are detailed in Section 7 of this deliverable.

6.1 Dependent programs vs. statistical and/or probabilistic (in)dependence

The main criticism against probabilistic and statistical approaches concerns the independence hypothesis that such approaches may require. A common comment is that the dependent programs may not fulfil such requirement when in reality the probabilistic approaches and/or statistical approaches do not impose to the programs to be independent. In this section we define the three notions of independence: between two programs, probabilistic and statistical. These three notions do not exclude each other as they concern different levels of the execution of a program.
In order to ease the understanding, in Section 6.1 we present examples built using arbitrary values for the input data of the considered programs. The reader should be aware that such process of obtaining the execution times cannot be used today to provide a solution for the pWCET estimation of a program. Besides the reproducibility concerns that such approach may raise, it is not possible today to prove the pWCET dominance for these approaches. Moreover the considered programs are not representative of any PROXIMA case study, their utilization is limited to this section with the only pedagogical purpose of explaining the different concepts of independence.
Independent programs We consider two programs $Prog_1$ and $Prog_2$ to be independent if any execution of $Prog_1$ may be done before or after any execution of $Prog_2$ without any impact on their execution times.
Two programs, that are in any other situation that those covered by the definition of independent programs given previously, are dependent.
For instance let the program $Prog_{ex1}$ be described in Table 6 and $Prog_{ex2}$ described in Table 7. These two programs are kept simple in order to ease the understanding. The two programs $Prog_{ex1}$ and $Prog_{ex2}$ are dependent as $Prog_{ex1}$ produces a (positive integer) value for the global variable var_global_2 that is then used as an input by $Prog_{ex2}$. For instance each time $var\_global_1 = 1$, then $Prog_{ex1}$ has an execution time equal to 3 time units and $Prog_{ex2}$ has an execution time equal to 6 time units. For $var\_global_1 = 2$, then $Prog_{ex1}$ has an execution time equal to 4 time units and $Prog_{ex2}$ has an execution time equal to 10 time units.

Table 6: Body of program $Prog_{ex1}$

$Prog_{ex1}$ (var_global_1);	
$value = var\_global_1$;	// *execution time* = 1 *time unit*
for i = 1 to var_global_1	// *the loop cost is in the instruction*
$value = value + 1$;	// *execution time* = 1 *time unit*
endfor	
$var\_global_2 = 2 * value$;	// *execution time* = 1 *time unit*

Table 7: Body of program $Prog_{ex2}$

$Prog_{ex2}$ $(var\_global_1, var\_global_2)$;	
$value = var\_global_2$;	// *execution time* = 1 *time unit*
for i = 1 to var_global_2	// *the loop cost is in the instruction*
$value = value + 1$;	// *execution time* = 1 *time unit*
endfor	
$average\_global = \frac{value+var\_global_1}{2}$;	// *execution time* = 1 *time unit*

For these two programs we may obtain both statistical dependent execution times or statistical independent execution times.
Probabilistic independence Two probability distributions $\mathcal{C}_1$ and $\mathcal{C}_2$ respectively are independent if

$$P(\{\mathcal{C}_1 = c_1\} \cap \{\mathcal{C}_2 = c_2\}) = P(\mathcal{C}_1 = c_1) \cdot P(\mathcal{C}_2 = c_2)$$

For instance the probability distributions of the execution times of $pET(Prog_{ex1})$ and the probability distributions of the execution times of $pET(Prog_{ex2})$ are probabilistically dependent as there is a relation between the probability of appearance of an execution time for $Prog_{ex1}$ and the probability of appearance of an execution time for $Prog_{ex2}$. If one would like to estimate the probability distributions of the execution times of these two programs executed sequentially then, given their probabilistic dependence, a complex probabilistic operation is necessary to take into account the conditional probabilities. Sometimes these dependences are not strong and simple probabilistic operations are possible [35].
Nevertheless, if one is able to provide a pWCET estimation $pWCET(Prog_{ex1})$ for $Prog_{ex1}$ and a pWCET estimation $pWCET(Prog_{ex2})$ for $Prog_{ex2}$, then these two probability distributions are independent by the definition of a worst case bound [10], i.e., the bounds are obtained using worst case assumptions.

Statistical independence A set A is statistically independent if its elements are generated in a random manner. The value generated at one instant only depends on the generator and not on the values generated before.
For instance the (non-ordered) set $A_{ex} = \{8, 18, 21, 24, 28, 30\}$ is statistically independent. We have generated A_{ex} using an on line random generator[6]. Such independent set is used as an input to obtain independent execution times for our two programs, $Prog_{ex1}$ and $Prog_{ex2}$. If we consider var_global_1 to take the values from

[6]We have used the on line form from http://www.infowebmaster.fr/outils/generateur-nombre-aleatoire.php, but the reader may use any other such generator.

A_{ex} then the set of execution times of $Prog_{ex1}$ is $\mathcal{C}_{Prog_{ex1}} = \{10, 20, 23, 26, 30, 32\}$ which is statistically independent.
In order to obtain for $Prog_{ex2}$ a set of statistically independent execution times we consider the values of var_global_2 to take the values from (another) statistically independent set $A_{bis} = \{1, 45, 59, 75, 88, 90\}$. We obtain a set of statistically independent execution times for $Prog_{ex2}$ equal to $\mathcal{C}_{Prog_{ex2}} = \{3, 47, 61, 77, 90, 92\}$. These two sets of execution times are statistically independent, while the programs are dependent.
Moreover if we use a set of dependent elements like $B = \{1, 2, 3, \cdots, 8\}$, then we may obtain statistical dependent sets for the execution times of $Prog_{ex1}$ and $Prog_{ex2}$. In this case the execution times of $Prog_{ex1}$ are $\{3, 4, 5, 6, 7, 8, 9, 10\}$ and the execution times of $Prog_{ex2}$ are $\{6, 10, 14, 18, 22, 26, 30, 34\}$. These two sets are statistically dependent, while the programs are dependent.
We may underline that all sets of the execution times should be identically distributed [10], otherwise this indicates that the measurements are not obtained by observing the execution of the same program.

6.2 The definition of the probabilistic worst case execution time

The definition of the pWCET of a program is not intuitive in the sense that one may wonder how a program may have different WCETs. This lack of intuition comes from an incorect definition of the pWCET of a program that has been used by the community or at least commonly accepted: *the pWCET of a program is the probability distribution defining the probability that its WCET is equal to a given value.* This incorrect definition suggests that the WCET contains all possible values for the WCET thus that there are several possible WCETs.
A correct definition of the pWCET should indicate its relation to the possible execution times of a program by upper bounding such set. This may seem trivial today but the first correct definitions have been proposed for the first time by the PROARTIS/PROXIMA participants. We provide below the complete mathematical framework allowing a correct definition of the probabilistic worst case execution time.
Let $C_1^i, C_2^i, \cdots, C_n^i$ be n execution times of a program on a processor starting from a given scenario of execution S_i. A scenario of execution for a program on a processor is defined by a set of states indicating the values of the different execution time variability factors.
For a scenario S_i we may define a probabilistic execution time $\mathcal{C}_i$ as an empirical probability distribution of the execution time of that program for the given processor. For instance we have $\mathcal{C}_i$ defined as follows

$$\mathcal{C}_i = \begin{pmatrix} 2 & 3 & 5 & 6 & 105 \\ 0.7 & 0.2 & 0.05 & 0.04 & 0.01 \end{pmatrix} \tag{11}$$

The probabilistic worst-case execution time (pWCET) $\mathcal{C}$ of that program is then an upper bound on all possible probabilistic execution times $\mathcal{C}_i$ for all possible execution scenarios $S_i, \forall i \geq 1$. The relation $\succeq$ [13] describes the relation between the probabilistic execution times (pETs) of a program and its probabilistic worst case execution time (pWCET), $\mathcal{C}_i \succeq \mathcal{C}_i^j$, $\forall j$, defined as follows.

We say that $\mathcal{C} \succeq \mathcal{C}_i$ or $\mathcal{C}$ is worse than $\mathcal{C}_i$ if its complementary cumulative distribution function (1-CDF) has a higher or equal probability associated to each possible value, i.e., $P(C \geq c) \geq P(C_i \geq c)$, $\forall c$.

For instance, the probability distribution defined in Figure 38 has its 1-CDF provided in Figure 39 (blue). Moreover, the probability distribution defined in Figure 38 is upper bounded by the probability distribution described by Figure 39 (in red). There may exist two probability distributions that are not comparable with respect to the relation $\succeq$.

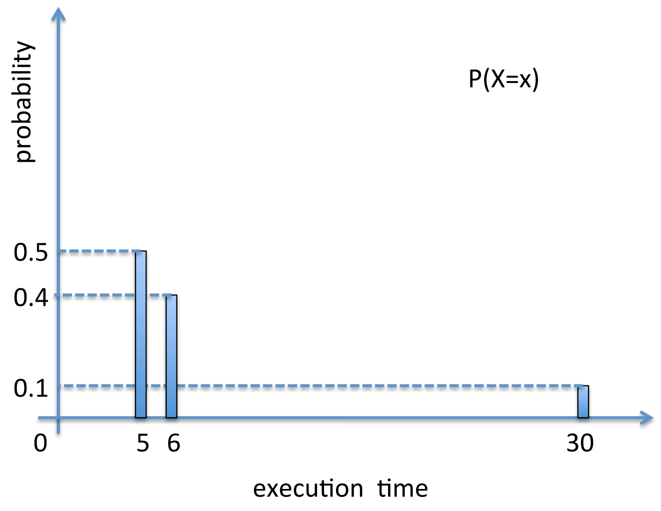


Figure 38: A probability distribution function (PDF)

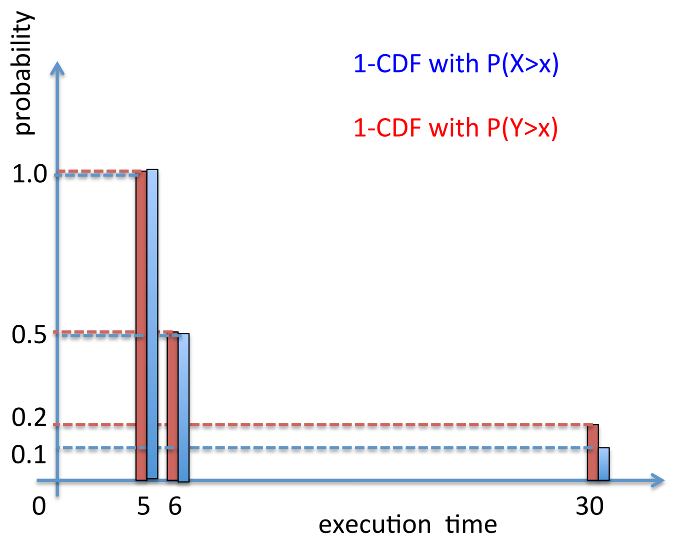


Figure 39: 1-CDF associated to the PDF defined in Figure 38 and 1-CDF that upper bounds the random variable described in Figure 38 (red)

7 PROXIMA influence: new open problems for the real-time community

We conclude this deliverable by providing a list of the main open problems that can be formulated today thanks to PROXIMA understanding on probabilistic timing analysis.

7.1 Classification of programs and processors accepting the Gumbel hypothesis

We have provided a method showing for a program and a processor that the Gumbel hypothesis is correct. This does not imply that pWCET estimate is defined by a Gumbel as it says that the generalized probability distribution (GEV) has the Gumbel as the closest upper bound.
We do not expect to be able to prove that Gumbel is the most appropriate pWCET bound for any program and any processor. A theoretical proof for those programs and processors accepting the Gumbel hypothesis should include a classification of the programs. We believe that such proof requires the introduction of probabilistic description at higher level of the conception of time critical embedded systems, e.g., controllers. Such probabilistic description should be then be used as interface between different design stages.

7.2 Representativity of a pWCET estimate

We have provided through its pWCET estimation method a measure of the representativity of two sets of execution times. This allows to propose reasoning like minR where the pWCET obtained for a set of execution times is compared to the pWCET obtained for the sub-set of the original set. The absence of a convergence principle for minR is connected to the absence of a representativity proof for the PROXIMA execution context. On a more positive note we believe that the PROXIMA project has allowed to mathematically formulated for the first time the relation between the convergence and the representativity.

7.3 Generalizing EPC towards other platforms

So far EPC has been proven doable on the FPGA HWrand platform within PROXIMA using first level data (DL1) and instruction (IL1) caches. However, the deployment of EPC in the context of (1) FPGA SWrand platform, (2) together with cache hierarchies (e.g., with a second level, L2 cache) or even (3) on top of different architectures has not been investigated yet buut seems a promising line of investigation.
We regard its extension towards the FPGA SWrand platform and towards systems with cache hierarchies – either HWrand or SWrand – as possible as EPC builds on upper/lower-bounding hit and miss probabilities and hit/miss probabilities have been already characterized in the past for SWrand and multilevel caches [26, 28]. Therefore, while the extension of EPC remains an open problem, we do not foresee any blocking issue.

Extending EPC towards other platforms, instead, is a challenge that needs to be considered platform-by-platform. For instance, as shown in D1.10, in the FPGA platform execution time mostly relates to cache misses and timing anomalies cannot occur. Hence, other platforms with these same characteristics are not expected to bring major challenges other than potential tooling difficulties. For platforms with different timing characteristics, we expect that adapting EPC is going to be much more difficult; however, we have not been considering this scenario so far and cannot attempt any reasoned prevision.

7.4 Proving whether a sample size contains all relevant information

As shown in D1.10, random placement in caches can produce relevant execution time variations with relatively low probability. Despite the low probability, those events need to be observed for a reliable application of MBPTA. We have developed a number of techniques for access-pattern insensitive programs for HWrand and SWrand platforms, and a technique with potentially high execution time costs for arbitrary programs also on top of HWrand platform using hash-based random placement.
Some open challenges remain: extending those techniques to arbitrary programs on top of random modulo placement (see D1.11) and SWrand platforms, extending them to multilevel caches and multipath programs, and keeping computational costs low. While these challenges still need an answer, the probabilistic nature of the events to study indicates that techniques in the same line as the ones already developed can be devised for the open challenges [2, 29].

7.5 Assessing completely hardware in complex COTS platforms

Platforms with low or moderate hardware complexity such as the FPGA COTS multicore and the AURIX platform have been shown to be analyzable with software-only techniques dealing with cache behavior, multicore interference and FPU latency variability. However, complex platforms such as the P4080 introduce a number of hardware features such as advanced branch prediction, out-of-order execution, complex pipelines, etc. not present in the other platforms.
Typically, execution time is much more sensitive to cache behavior and multicore contention than to other effects. However, determining whether this claim can be generalized to all programs in those platforms or it only holds under certain conditions is an open challenge that needs an answer. Some considerations related to this matter are provided in D1.10 in the preliminary analysis of timing anomalies for the FPGA HWrand platform. Still this analysis is preliminary and mostly fitting a specific platform.

7.6 Contender generation for inter-core interference analysis

The complexity of multi-core platforms thwarts in general the definition of upper-bounding scenarios in terms of inter-core interferences on shared resources. An intuitive definition of contenders as those which maximise their use of shared resources may indeed not results in the worst-case interference behaviour for an analysed task. Even if it exists, the successful definition of such a worst-case scenario may still lead to pessimistic timing estimates. To extrapolate the overall impact of interferences on a specific task including the worst, it is a requirement to observe it under different interferences scenarios. This is one part of the rationale behind the VICI analysis.
Contenders in the context of PROXIMA have been manually defined to randomly interfere with the analysed task on the shared resources, both in terms of access pattern and interference level. The definition of contenders is a arduous and error-prone process repeated for each analysed platform. Automated test-case generations techniques [30] are a demonstrably useful tool to improve the reliability of measurement-based timing analysis. In the context of multi-core systems, they could be extended to help measure and improve the confidence, i.e. coverage of the interference space, and certainty, i.e. uniformity of the coverage, of the interference analyses.

7.7 Interference-relevant factor selection with multi-modal distributions

Under different constraints a same task can exhibit correspondingly varying behaviours, e.g. through different paths or slightly different access patterns. Each of these modes may in turn react differently to interference from other tasks. Such variations must be taken into account by the interference analysis. Multi-modal datasets are captured by the VICI analysis as low accuracy data points in the training data. Appropriate sub-models and a classifier are then generated to identify the mode of an observation, and derive correct safety margins for each identified mode. All steps however rely on the same set of factors in the analysis.
The operating mode of a task is best observed through intra-core metrics, e.g. number of executed branches, whereas inter-core effects will be reflected in metrics relevant to the shared resources, e.g. memory stalls. Alternative interference-relevant factor selection strategies can be considered to distinguish between variability due to changes in interferences or modes. The inclusion of external data, such as user inputs, in the selection process can help tune the process without reducing the availability of observable factors (PMCs) for the interference analysis. Different modes similarly should be considered through different metrics such that an interference model can use a subset of the collected factors for specific sub-models.

7.8 Integrate inter-core interference analysis in schedulability analysis

The inter-core interference analysis provides a bound on the impact of interferences suffered by a task. A sound upper-bound is then valid irrespective of the actual

observed interferences, and encompasses the worst-case scenario. However, as observed in the context of Cache Related Preemption Delay (CRPD) analyses, such bounds can be overly pessimistic [6] assuming all cache blocks have been evicted or each access to a shared resource suffers its maximum latency. More refined techniques instead rely on the integration of both interference and scheduling analyses to identify potential preempting tasks.
Both the pTC and VICI analyses provide for a parametric definition of the impact of interferences on a task. A tight bound can be derived on the effect of interferences from observations pertaining to the contending tasks. This property could be conductive to the definition of an iterative inter-core interference-compliant scheduling algorithm. The impact of interferences can be accounted for similarly to the CRPD. The task is first assumed to suffer its worst-case interferences, and the estimate is then refined as the actual contenders for shared resources are identified. Each application of the interference analysis can result in lesser execution times, and in turn less potential contenders.

Acronyms and Abbreviations

- EPC: Extended Path Coverage.

- EVT: Extreme Value Theory.

- FBI-VICI: Forecast based interference - Variable Inter-Core Interference

- (pTC/fTC)-VICI: Partially/Fully Time composable analysis

- MBPTA: Measurement-Based Probabilistic Timing Analysis.

- PTA: Probabilistic Timing Analysis.

- RVS: Rapita Verification Suite.

- pWCET: Probabilistic Worst-Case Execution Time.

- WCET: Worst-Case Execution Time.

References

[1] L. Yue et al. A new way about using statistical analysis of worst-case execution times. September 2011.

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

[3] J. Abella et al. WCET analysis methods: Pitfalls and challenges on their trustworthiness. In *SIES*, 2015.

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

[5] Aeroflex Gaisler. *LEON3 GR-XC3S-1500 Template Design, V1.3.7*, 2014.

[6] Sebastian Altmeyer and Claire Maiza. Cache-related preemption delay via useful cache blocks: Survey and redefinition. *Journal of Systems Architecture*, 2010.

[7] Pedro Benedicte, Leonidas Kosmidis, Eduardo Quiñones, Jaume Abella, and Francisco J. Cazorla. Modelling the confidence of timing analysis for time randomised caches. In *11th IEEE Symposium on Industrial Embedded Systems, SIES 2016, Krakow, Poland, May 23-25, 2016*, pages 141–148, 2016.

[8] F.J. Cazorla et al. Proartis: Probabilistically analysable real-time systems. *ACM TECS*, 2012.

[9] D. Chiou, P. Jain, S. Devadas, and L. Rudolph. Dynamic cache partitioning via columnization. In *DAC*, 2000.

[10] L Cucu-Grosjean. Independence - a misunderstood property of and for (probabilistic) real-time systems. In *"Real-Time Systems: the past, the present, and the future", the 60th birthday of Professor Alan Burns*, 2013.

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

[12] E. Diaz, J. Abella, E. Mezzetti, I. Agirre, M. Azkarate-Askasua, T. Vardanega, and F.J. Cazorla. Mitigating software-instrumentation cache effects in measurement-based timing analysis. In *Proceedings of the 16th International Workshop on Worst-Case Execution Time (WCET) Analysis*, 2016.

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

[14] Edgar S and Burns A. Statistical analysis of WCET for scheduling. In *the 22nd IEEE Real-Time Systems Symposium (RTSS01)*, pages 215–225, 2001.

[15] Heiko Falk, Sebastian Altmeyer, Peter Hellinckx, Björn Lisper, Wolfgang Puffitsch, Christine Rochange, Martin Schoeberl, Rasmus Bo Srensen, Peter Wägemann, and Simon Wegener. TACLeBench: A benchmark collection to support worst-case execution time research. In *Proceedings of the 16th International Workshop on Worst-Case Execution Time Analysis (WCET '16)*, 2016.

[16] Gabriel Fernandez, Jaume Abella, Eduardo Quiñones, Tullio Vardanega, Luca Fossati, Marco Zulianello, and Francisco J. Cazorla. Introduction to partial time composability for COTS multicores. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing, Salamanca, Spain, April 13-17, 2015*, pages 1955–1956, 2015.

[17] Gabriel Fernandez, Javier Jalle, Jaume Abella, Eduardo Quiñones, Tullio Vardanega, and Francisco J. Cazorla. Resource usage templates and signatures for COTS multicore processors. In *Proceedings of the 52nd Annual Design Automation Conference, San Francisco, CA, USA, June 7-11, 2015*, pages 155:1–155:6, 2015.

[18] Freescale. *e500mc Core Reference Manual, Rev. 3*, 2013.

[19] M.K. Gardner and J.W. Lui. Analyzing stochastic fixed-priority real-time systems. In *the 5th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS99)*, pages 44–58, 1999.

[20] B.V. Gnedenko. Sur la distribution limite du terme maximum d'une seris aleatoire. *Annals of Mathematics*, 44:423–453, 1943.

[21] J. Gustafsson et al. The Mälardalen WCET benchmarks – past, present and future. In *the International Workshop on Worst-case Execution-time Analysis*, 2010.

[22] 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.

[23] Javier Jalle, Mikel Fernandez, Jaume Abella, Jan Andersson, Mathieu Patte, Luca Fossati, Marco Zulianello, and Francisco J. Cazorla. Bounding resource-contention interference in the next-generation multipurpose processor (NGMP). In *8th European Congress on Embedded Real-Time Software and Systems (ERTS2)*, 2016.

[24] Seongbeom Kim, Dhruba Chandra, and Yan Solihin. Fair cache sharing and partitioning in a chip multiprocessor architecture. In *PACT*, USA, 2004.

[25] R. Kirner and P. Puschner. Classification of Code Annotations and Discussion of Compiler Support for Worst-Case Execution Time Analysis. In *Proceedings of the 5th International Workshop on Worst-Case Execution Time Analysis*, 2005.

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

[27] L. Kosmidis, J. Abella, E. Quinones, F. Wartel, G. Farrall, and F.J. Cazorla. Containing timing-related certification cost in automotive systems deploying complex hardware. In *DAC*, 2014.

[28] L. Kosmidis, C. Curtsinger, E. Quinones, J. Abella, E. Berger, and F.J. Cazorla. Probabilistic timing analysis on conventional cache designs. In *DATE*, 2013.

[29] L. Kosmidis et al. Achieving timing composability with measurement-based probabilistic timing analysis. In *ISORC*, 2013.

[30] Stephen Law and Iain Bate. Achieving appropriate test coverage for reliable measurement-based timing analysis. In *Proceedings of the 28th IEEE Euromicro Conference on Real-Time Systems (ECRTS 2016)*, 2016.

[31] Enrico Mezzetti and Tullio Vardanega. On the Industrial Fitness of WCET Analysis. In *Proceedings of the 11th International Workshop on Worst-Case Execution Time Analysis (WCET)*, 2011.

[32] Suzana Milutinovic, Jaume Abella, and Francisco J. Cazorla. Modelling probabilistic cache representativeness in the presence of arbitrary access patterns. In *19th IEEE International Symposium on Real-Time Distributed Computing, ISORC 2016, York, United Kingdom, May 17-20, 2016*, pages 142–149, 2016.

[33] Jason W Osborne and Anna B Costello. Sample size and subject to item ratio in principal components analysis. *Practical assessment, research & evaluation*, 9(11):8, 2004.

[34] RTCA. DO-178C, software considerations in airborne systems and equipment certification, 2011.

[35] M. Santos et al. Sequential composition of execution time distributions by convolution. In *CRTS*, 2011.

[36] T.S. Tia, Z. Deng, M. Shankar, M. Storch, J. Sun, L.C. Wu, and J.S Liu. Probabilistic performance guarantee for real-time tasks with varying computation times. In *the 2nd IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS95)*, pages 164–174, 1995.

[37] P. K. Valsan, H. Yun, and F. Farshchi. Taming non-blocking caches to improve isolation in multicore real-time systems. In *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2016.

[38] S. Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *the 28th IEEE Real-Time Systems Symposium (RTSS*, pages 239–243, 2007.

[39] R. Wilhelm et al. The worst-case execution-time problem overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems*, 7:1–53, May 2008.

[40] Wilhelm R. et al. The worst-case execution-time problem overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems*, 7:1–53, May 2008.

[41] M. Ziccardi, E. Mezzetti, T. Vardanega, J. Abella, and F. J. Cazorla. Epc: Extended path coverage for measurement-based probabilistic timing analysis. In *Real-Time Systems Symposium, 2015 IEEE*, pages 338–349, 2015.