# On the Limits of Probabilistic Timing Analysis


Suzana Milutinović

Computer Architecture Department

Universitat Politècnica de Catalunya

A thesis submitted for the degree of

*Doctor of Philosophy in Computer Architecture*

September 2019

# On the Limits of Probabilistic Timing Analysis

Suzana Milutinović
September 2019

Universitat Politècnica de Catalunya
Computer Architecture Department

Thesis submitted for the degree of
Doctor of Philosophy in Computer Architecture

**Advisor:** Francisco J. Cazorla
*Barcelona Supercomputing Center and IIIA-CSIC*
**Co-advisor:** Jaume Abella
*Barcelona Supercomputing Center*

# Abstract

Over the last years, we are witnessing the steady and rapid growth of Critical Real-Time Embedded Systems (CRTES) industries, such as automotive and aerospace. Many of the increasingly-complex CRTES' functionalities that are currently implemented with mechanical means are moving towards to an electromechanical implementation controlled by critical software. This trend results in a two-fold consequence. First, the size and complexity of critical-software increases in every new embedded product. And second, high-performance hardware features like caches are more frequently used in real-time processors.

The increase in complexity of CRTES challenges the validation and verification process, a necessary step to certify that the system is safe for deployment. Timing validation and verification includes the computation of the Worst-Case Execution Time (WCET) estimates, which need to be trustworthy and tight. Traditional timing analysis is challenged by the use of complex hardware/software, resulting in low-quality WCET estimates, which tend to add significant pessimism to guarantee estimates' trustworthiness. This calls for new solutions that help tightening WCET estimates in a safe manner.

In this Thesis, we investigate the novel Measurement-Based Probabilistic Timing Analysis (MBPTA), which in its original version already shows potential to deliver trustworthy and tight WCETs for tasks running on complex systems. First, we propose a methodology to assess and ensure that all cache memory layouts, which can significantly impact WCET, have been adequately factored in the WCET estimation process. Second, we provide a solution to achieve simultaneously cache representativeness and full path coverage. This solution provides evidence proving that WCET estimates obtained are valid for all program execution paths regardless of how code and data are laid out in the cache. Lastly, we analyse and expose the main misconceptions and pitfalls that can prevent a sound application of WCET analysis based on Extreme Value Theory (EVT), which is used as part of MBPTA.

# Acknowledgements

I wish to give thanks to the many people who supported me during my PhD studies and contributed to the delivery of this Thesis.

First and foremost, I would like to give my heartfelt thanks to Dr Francisco J. Cazorla and Dr Jaume Abella for providing me with the opportunity to develop this Thesis under their direction and for being excellent supervisors. I am grateful for their time and effort to share their knowledge with us, teach us skills, organise productive and collaborative environment for work and encourage us when facing challenges. I would also like to express my gratitude to Dr Enrico Mezzetti and Dr Tullio Vardanega for their valuable contributions to this research work. Further, I wish to thank Rapita Systems for hosting me during the 5-months internship.

This Thesis would be much more limited in scope without the work carried out by prior and current students and engineers in the Computer Architecture / Operating System Interface team of the Barcelona Supercomputing Center. Many thanks for their effort, help, Friday meetings and development of simulation environment. I wish to extend my thanks to other students of the Barcelona Supercomputing Center, who gave me helpful pieces of advice related to PhD studies, life in Barcelona and in general: Milos, Mladen, Nikola, Ivan, Petar.

On the personal side, I would like to thank my parents and my sister for their unconditional love and support. To Vladimir and Gabriel for many meaningful discussions and fun during the break time from the lab. To Jovana and Marina who greatly contributed that my life in Barcelona is more enjoyable and social. To my friends in Serbia who, despite our distance, are still giving me a warm welcome. And to my current colleagues in Esperanto Technologies from whom I continued learning a lot following my student years.

This Thesis has been financially supported by the Barcelona Supercomputing Center and the Spanish Ministry of Economy and Competitiveness under the FPI grant (BES-2016-077561). Support was

also provided by the European Community's Seventh Framework Programme [FP7/2007-2013] under the PROXIMA project (grant agreement 611085), as well as the Spanish Ministry of Science and Innovation (grant TIN2015-65316-P). Collaboration with Rapita Systems was funded by FPI grant.

# Contents

# List of Figures

# List of Tables

# List of Acronyms

**CCP** Conflictive Cache Placement

**CCP-hRP** Conflictive Cache Placements for hash Random Placement method

**CCP-RM** Conflictive Cache Placements for Random Modulo Placement method

**CCPX** Conflictive Cache Placements method (for arbitrary Random Placement)

**CCDF** Complementary Cumulative Distribution Function

**CDF** Cumulative Distribution Function

**CRTES** Critical Real-Time Embedded Systems

**DL1** Data Cache Level 1

**ECCDF** Empirical Complementary Cumulative Distribution Function

**EVT** Extreme Value Theory

**FPGA** Field-Programmable Gate Array

**GTAB** Guilt Table

**HoG** Heart of Gold

**hRP** hash Random Placement

**IL1** Instruction Cache Level 1

**MBDTA** Measurement-Based Deterministic Timing Analysis

**MBTA** Measurement-Based Timing Analysis

**MBPTA** Measurement-Based Probabilistic Timing Analysis

**PDF** Probabilistic Distribution Function

**pETD** probabilistic Execution Time Distribution

**pWCET** probabilistic Worst-Case Execution Time

**pWCMC** probabilistic Worst-Case Miss Count

**PUB** Path Upper-Bounding

**ReVS** Representativeness Validation by Simulation

**RM** Random Modulo Placement

**SoJ** Sources of Jitter

**SDTA** Static Deterministic Timing Analysis

**SPTA** Static Probabilistic Timing Analysis

**STA** Static Timing Analysis

**TRC** Time Randomised Caches

**WCET** Worst-Case Execution Time

# List of Symbols

$< impact, probability >$ Conflictive Cache Placements (CCPs) are described with the average miss count (impact) they cause attached to their probability of occurrence.

$< combination; probability >$ The representative address combination attached to the probability of its occurrence or of any address combination with similiar impact.

$aC_i$ One combination of addresses from $@(\Omega_i)$.

$|aC_i|$ Address count in (i.e. cardinality of) $aC_i$.

$addr_{[j]}^{[X]}$ Memory address represented with a letter (e.g. A,B,C). j is an optional parameter meaning the number of times the memory address has been referenced in the sequence, or any access if omitted. X is an optional parameter which does not have any meaning for hash Random Placement (hRP) and represents the cache segment the address belongs to for Random Modulo Placement (RM).

$guilt$ Measure of the contribution of a given address to the $P_{guilty}$ of the address of interest.

$K$ Cardinality of (number of addresses in) a combination.

$P_{cth}$ Coverage threshold probability dictated by safety standard.

$P_{event}$ Probability of event of interest (e.g. Conflictive Cache Placement (CCP)).

$P_{guilty}$ Estimated probability of the address of interest to be evicted.

$P_{miss}$ Exact probability of miss of the address.

$\tilde{P}_{miss}$ Approximate probability of miss of the address.

$P_{nobs}$ Probability of missing event at analysis (depending on the standard, safety is broken if $P_{nobs}$ of relevant event is above $P_{cth}$).

$P_{obs}$ The lowest probability of the event that is expected to be captured in certain number of runs $R$.

$P_{rel}$ Threshold probability separating relevant from non-relevant events (events occurring with lower probabilities are deemed as not relevant).

$q$ Number of distinct addresses in a sequence $X_i$.

$\mathcal{Q}_i$ Sequence of accesses given in format $addr_{[j]}^{[X]}$.

$@(\mathcal{Q}_i)$ Set of unique addresses in $\mathcal{Q}_i$.

$|@(\mathcal{Q}_i)|$ Number of unique addresses in $\mathcal{Q}_i$.

$R$ Number of measurements to collect by MBPTA convergence criteria.

$R'$ Number of measurements to collect by representativeness criteria.

$s$ Number of distinct cache segments in a sequence $X_i$.

$S$ Number of cache sets.

$T$ Number of conflictive combinations returned by CCP-RM/CCP-hRP.

$U$ Number of unique (cache line) addresses in a program.

$U'$ Limited number of the most frequently accessed unique (cache line) addresses in a program, analysable by Representativeness Validation by Simulation (ReVS) method.

$W$ Number of cache ways.

$X_i$ Subsequence of accesses between two repeated accesses to the same address in a sequence $Q_i$.

# Chapter 1

# Introduction

## 1.1 Embedded Systems

In recent years, embedded systems – computers designed to perform a specific function within a larger system – have become ubiquitous in our professional and personal lives. The services they provide are numerous and increasingly involve critical aspects, such as controlling the engines and brakes of cars, flight control in planes, and monitoring health devices, to name a few. Currently, embedded systems are the fastest-growing portion of the computer market [62].

Embedded systems present key differences when compared to desktop and server systems. Due to their integrated nature, size, weight and power constraints for this kind of systems are more stringent. Embedded systems are designed to perform a certain domain-specific function, typically by the joint use of hardware and software, and are not intended for 'general-purpose' operation. This function often needs to be performed both correctly and *timely*, otherwise, the result might be useless or the system controlled by this function may fail. Other than meeting its functional and non-functional constraints, other key factors to consider when designing embedded systems are cost and time-to-market. In particular, embedded systems must meet specific performance levels within given power envelopes and cost constraints, and reaching the market timely. Sacrificing any of these metrics (e.g. cost) for the sake of further improving another beyond its requirements (e.g. increasing performance) is not an option for embedded systems [62].

The subclass of embedded systems in which functions are bounded by timing constraints are referred to as *real-time* embedded systems. They commonly comprise a set of concurrent *tasks*, where each task issues jobs to perform a computational activity. Unlike the common use of the term 'real-time' in other computing

domains, these systems are not designed to be fast in the average case or at peak performance. The objective is to guarantee that timing constraints will be *consistently* met, even when the system performs under worst-case conditions. Evidence has to be provided that each instance of a task will complete its execution before its assigned *deadline*. As predictability and stringent size, weight and power constraints are of prominent importance, processors deployed in real-time embedded systems have historically been simpler than general-purpose processors and programming guidance for software more restrictive (e.g. bounded loop iterations and limited usage of recursion).

In Critical Real-Time Embedded Systems (CRTES) a functional or timing failure may lead to catastrophic consequences. These systems can be characterised as safety-critical (malfunctioning of the system may cause loss of human lives or severe environmental damage), mission-critical (system failure will impede some goal-directed activity), or business-critical (failure will cause important economic losses to the organization). Before being deployed for the operation, CRTES need to pass a strict *certification process*, in which an independent certification body assesses whether a system complies with the *safety-standard* for its corresponding domain and is safe for its intended purpose (e.g. ISO 26262 for road vehicles [70]).

## 1.2 Specific Requirements of CRTES

### 1.2.1 High Performance

As any other embedded system, CRTES aim at providing additional and innovative features to improve products' competitive edge in the market [66]. To support these new services, the software frequently drives the decision-making process over vast amounts of data of diverse types, which significantly increases its complexity. This trend is illustrated in Figure 1.1, which shows the exponential increment in code size in space, avionic and automotive domains across products delivered in the last forty years. We see an exponential increase in all three analysed domains, with this trend expected to continue in coming years.

On the hardware side, CRTES have traditionally deployed rather simple processors with a single-core, short pipelines, in-order execution and no cache memories. However, the computational power provided by traditional hardware has already been shown insufficient to support the performance needs for today's critical software. The need for computation capacity is going to exacerbate in the next years, e.g. the performance required in the automotive domain is expected to increase by 100x [13]. The demand for performance is further increased as more functions are integrated together with the adoption of the integrated architecture paradigm (presented in Section 1.2.3).

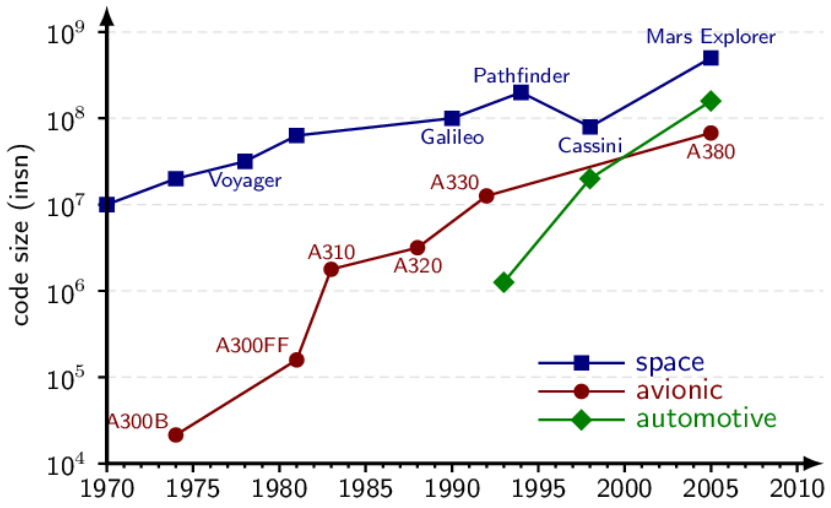



Figure 1.1: Code size in different CRTES products in space, avionic and automotive sectors. (Figure from J. Bin et al. [20])

To catch up with increasing computing performance demands, the CRTES industry is progressively assessing the use of performance-accelerating hardware features (e.g. caches), typically designed to increase average performance. The other side of the coin is that the presence of these features, together with the increased complexity of software, reduces the predictability of the system and challenges deriving the evidence of correct functional and timing behaviour needed for certification.

### 1.2.2 Timing Validation and Verification

The validation and verification process for CRTES requires collecting *sufficient evidence* that critical functions will execute correctly and timely. In this context, the term *sufficient evidence* relates to the corresponding functional safety standard and the *integrity level* of the task analysed. The safety integrity level is a quantitative measure of the amount of evidence for safety functions needed to achieve an acceptable risk for a process hazard [69]. A target safety integrity level is determined based on a process hazard analysis which identifies all potential hazards of a process, estimates their frequency of occurrence and the severity of their consequences. Examples of safety standards are ARP4761 in the avionics domain [105] and ISO26262 in the automotive domain [70].

Timing validation and verification of real-time tasks comprises estimating a bound to the Worst-Case Execution Time (WCET) of those tasks with appropriate methods and tools, and providing evidence that they can be scheduled into their allocated time budgets. In industrial environments, several factors deter-

mine the WCET analysis tool/technique to use. First, providing reliable WCET estimates – with the level of assurance defined by the relevant safety standards. Second, obtaining WCET estimates as *tight* as possible so that tasks can be successfully scheduled while minimising the number of hardware resources required. And third, keeping the overall cost of the timing analysis technique as low as possible to maintain the competitive edge. The growing complexity of the software and hardware used in CRTES affects all three factors and challenges state-of-the-art methods and practices for WCET estimation [6]. These methods are generally classified as static and measurement-based timing analysis techniques.

Static Timing Analysis (STA) [36, 63, 119, 120] computes WCET bounds by analysing a task code and potential control-flow paths through the task, and deriving an abstract model of the hardware on which the task runs. To calculate reliable estimates, for each event in the hardware influencing timing, if the outcome is unknown and in the absence of timing anomalies, STA makes the worst-case assumption (e.g. cache miss or branch misprediction). The tightness of the derived estimates depends on the ability of STA to predict which events in the platform that potentially increase tasks' execution time will not occur during the execution. The main challenge of STA is its dependence on detailed knowledge of the software and hardware, which may be absent (e.g. to protect the intellectual property) or simply incorrect (e.g. processor manuals are often revised with errata documents). Similarly, determining the outcome of hardware events influencing timing is immensely complicated, as it commonly depends on the history of tasks' execution and input values. The challenge exacerbates with the introduction of high-performance features (multi-level cache hierarchies, deep pipelines, branch prediction, etc.), decreasing the predictability of the system. As a result, the worst-case assumptions in a complex system lead to overly pessimistic WCETs.

Measurement-Based Timing Analysis (MBTA) [82, 118, 119] derives WCET bounds from measurements of the task running under stressful conditions on the target hardware platform. To account for the events that influence timing but are not observed in measurements, an engineering margin is added to the longest measured execution time, also referred to as *high-water mark*. The challenge of MBTA lies in ensuring that the measurements taken at analysis occur under conditions similar or worse to those that can arise during operation. The user (or timing analyst) is required to design stressful tests that will trigger events with detrimental effect on timing at analysis time, which builds on user's experience and control of those elements impacting application's execution time (e.g. timing of a task may be affected by the concrete memory layout, which users often do not have control over). The latter is challenged by the presence of complex hardware/software with massive interactions among components with non-obvious impact on timing, ultimately decreasing the level of assurance on MBTA's derived WCET estimates.

### 1.2.3 Time Composability

Traditionally, the design of CRTES has followed the *federated architecture* model in which different functions in a system are implemented in dedicated hardware units, physically separated from each other [94, 97]. This paradigm facilitates timing and functional isolation, improving system predictability. Additionally, the federated approach allows providers to implement system functions independently from other suppliers, creating a clear boundary of responsibility and error propagation, which simplifies the certification process. However, the increased complexity of modern CRTES challenges the scalability of this traditional paradigm as every new software functionality requires its specific hardware unit. For example, the number of electronic control units in 2017 already reached 200 for some cars [114].

The relentless need for more functionality in many CRTES has motivated the shift from the federated to the *integrated architecture* model, in which multiple functions execute on the same hardware unit. Following a divide-and-conquer strategy, the final product is the result of integrating together various pieces of hardware and software, which are typically made available by specialised providers in different steps in the industrial development chain. Several industrial domains have embraced this approach: integrated modular avionics [105] in avionics, automotive open system architecture [94] in automotive and integrated modular avionics for space [121] in the space domain. Integrated architectures bring significant technical and economic advantages, such as reducing hardware cost by decreasing the number of hardware dedicated units, wiring and connectors (which additionally increases reliability).

From the certification point of view, a key element of integrated architectures is *incrementality*, which allows an effective and economically efficient production and qualification of integrated systems. When it comes to software, incrementality means that different modules are progressively integrated to form more complex functionalities upon successive releases. Each software module follows its own development cycle so that different modules can be developed and qualified (at the test unit level) by their providers themselves. A *system integrator* is the person in charge of certifying a final product by bringing together pre-qualified parts. Unfortunately, in practice, integration may require modifications to already-integrated software to capture, for example, functional errors. As a result, it is not uncommon that a given application integrated in a given release suffers changes in later releases.

In this incremental integration and qualification approach, *composability* plays a central role [106]. At a functional level, composability guarantees that new software functionalities in a fresh release do not affect the functional behaviour of already integrated software. If composability does not hold, then costly regression tests are required to (re)assess the functional correctness of the whole system.

Likewise, in the timing domain, *time composability* [103] is also fundamental, guaranteeing that timing verification performed for a given software module in isolation remains valid upon integration.

In modern CRTES time composability is broken as tasks executing on the same processor system compete for shared resources like cache memories and branch predictors, whose access times are state dependent [103]. On the other hand, the state of these resources is dependent on the layout of code and data in the memory, on the history of accesses and the update strategy on access. As a consequence, the execution time of a task will vary depending on the state of shared resources left by other tasks or different instances of the same task. The challenge for MBTA is that the state seen at analysis time is very likely not the same state that will occur during the system operation.

## 1.3 A Probabilistic Approach to MBTA

The limitations of current timing analysis methods to cope with the increasing complexity of CRTES has motivated researchers to investigate alternative timing analysis paradigms. This includes probabilistic timing analysis [18, 19, 26, 28, 45], which gained attention due to its potential to derive estimates based on (black-box) observations, instead of relying on detailed knowledge on hardware and software internals. Probabilistic timing analysis techniques are divided into two categories Static Probabilistic Timing Analysis (SPTA) and Measurement-Based Probabilistic Timing Analysis (MBPTA), see Table 1.1. Over the last years MBPTA [7, 38] matured more rapidly owing to its cost-effectiveness which is attractive to industry.

Table 1.1: Broad classification of timing analysis techniques with acronyms.

| | Deterministic | Probabilistic |
|---|---|---|
| Static (STA) | SDTA | SPTA |
| Measurement-Based (MBTA) | MBDTA | MBPTA |

Instead of a single WCET estimate, probabilistic timing analysis delivers a probabilistic Worst-Case Execution Time (pWCET), a distribution that expresses the maximum probability with which one instance of the program can exceed a given execution time bound. For a distribution shown in Figure 1.2, the probability that the program will exceed the duration of 8 execution time units is lower than $10^{-13}$. Assuring that a single WCET bound covers all possible execution scenarios is normally done through *qualitative* assessments for traditional timing analysis techniques Static Deterministic Timing Analysis (SDTA) and Measurement-Based Deterministic Timing Analysis (MBDTA). In the case of SDTA the assurance depends on the level of detail known about the underlying model, while MBDTA

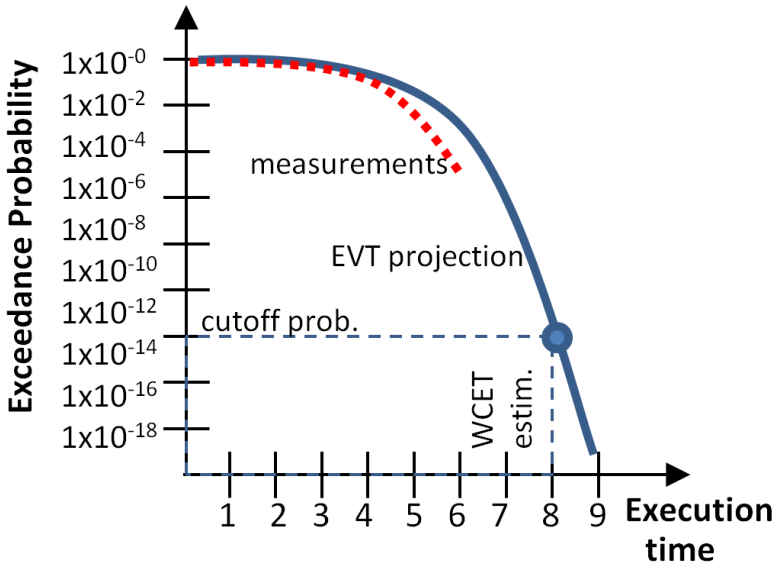



Figure 1.2: pWCET curve (blue, solid line) built based on a limited sample of measurements (red, dotted line). The selected WCET estimate is linked with a maximum (cut-off) probability of exceeding that execution time.

argues that all relevant test cases have been run and the WCET estimate provided is higher than any execution time observed. The increasingly complex processors affect both variants. In particular, they challenge delivering reliable models needed for SDTA and lead to higher variability of execution times, increasing the risk of missing relevant, high execution times for MBDTA. This lowers confidence on the WCET bounds provided based on qualitative reasoning. On the other hand, by assigning the probabilities of exceedance to the execution time bounds, probabilistic timing analysis provides *quantitative* means to upper-bound the residual risk of a timing fault. The WCET value assigned to a task is the one whose exceedance probability is deemed sufficiently low in relation to the safety integrity level of the task with respect to the corresponding safety standard (see Section 1.2.2). For instance, in the case of avionics, software with the highest design assurance level A must not exceed a failure rate of at most $10^{-9}$ per hour of operation [105].

### 1.3.1 Basic Concepts of MBPTA

As other MBTA techniques, MBPTA collects a sample of measurements (typically in the order of hundreds or thousands) on a target platform in the early stage of system development life cycle, called *analysis phase*. The stage after system deployment is known as *operation phase*. The conditions at which measurements are collected at the analysis phase need to be equal or more stressful than those during

the operation phase. This issue, common to all MBTA techniques, is known as *representativeness*. Unlike conventional MBDTA techniques that require explicit control over factors of influence on timing, called Sources of Jitter (SoJ), MBPTA attains representativeness through *statistical* control of SoJ.

To achieve their statistical control, MBPTA relies on the assumption that SoJ can be probabilistically modelled. This is done by upper-bounding jitter or making it have a random nature [77]. The latter is accomplished by injecting timing randomisation into the program's behaviour, e.g. by deploying random placement/replacement policies in cache memories [73]. In time randomised resources different executions of a program exercise different scenarios (e.g. different cache layouts). A probabilistic argument can be derived on how many executions of a program are needed so that the probability of missing a particular timing event is low enough such that residual risk of a timing fault is negligible. Alternatively, SoJ causing minor variation of execution times are upper-bounded by enforcing them to take their longest possible latency at the analysis phase only [27].

A key solution to manage those SoJ that cannot be controlled by end-users is time randomisation together with MBPTA. This reduces the dependency of timing events on the execution time history, facilitates time composability, preserves performance (i.e. we do not enforce the worst-case systematically), and relieves end users from having to exercise any control, which may be beyond their reach. One of the resources with substantial impact on timing, whose behaviour heavily depends on the execution time history and cannot be easily controlled by end-users is cache memory. L. Kosmidis et al. [75] show that the information needed to characterise the interference from other software units (e.g. other tasks) on shared cache memories and derive time composable WCET estimates is the number of unique data and instruction accesses of the code of those software units. On the other hand, characterising the interference on (time) deterministic cache memories requires detailed knowledge of all addresses of other software units.

Based on a sample of analysis phase measurements, MBPTA models a probabilistic Worst-Case Execution Time (pWCET) distribution to predict execution times occurring with rare probabilities. This is done by applying the well-known statistical method called Extreme Value Theory (EVT) [50, 80] on the collected sample. EVT intrinsically captures the impact and probability of multiple SoJ happening together, relieving the user of the need to trigger their simultaneous occurrence with a single test. This, along with timing randomisation and upper-bounding, significantly reduces the need for user's control and information that has to be provided. This is especially relevant in the presence of high-performance hardware, which vastly increases the need for a user's control.

While MBPTA has emerged only recently, the first industrial case studies show its promising potential [51, 116]. However, to reach the readiness level required for

transferring to industrial practice, MBPTA must respond to a set of challenges. Some of them have been addressed with recent works that we survey in the Section 1.3.2, while there are still some limitations that prevent the wide adoption of MBPTA (Section 1.3.3) and motivate the work of this Thesis (Section 1.4).

### 1.3.2 State of the Art on MBPTA

The selected timing analysis method must be compatible with the current certification procedures and standards in the corresponding domain. This requires the *qualification* of tools (or alternatively verification of their output) and *safety justification* for any additional manual work used in the analysis [6]. The tool needs to be qualified if its output directly impacts the software produced, or if faults undetected by this tool will reach the final product. The level of qualification needed spans from the most rigorous Level 1 to Level 5 and is on a par with the safety level of the system it is applied to. The compatibility of MBPTA with the actual certification practice (its *certifiability*) has been assessed in several studies [10, 113].

EVT has been used traditionally to model tail distributions in domains such as finance and hydrology. However, applying it in the execution time domain, as it is done by MBPTA, is not a straightforward process and requires the adaptation of the process for the particular problem being analysed. In the scope of this Thesis we build on the foundations set by J.Abella et al. [7] and use the *MBPTA-CV method* (more details in Section 3.2).

As discussed in Section 1.3.1, MBPTA assumes certain properties on the platform, in which the timing behaviour of jittery resources is either time upper-bounded or time randomised, resulting in the so-called *MBPTA-compliant platforms* [77]. A LEON3-based multicore Field-Programmable Gate Array (FPGA) board implementing these techniques has been developed in the project "Probabilistic real-time control of mixed-criticality multicore and manycore systems" (PROXIMA) [101] and it is now part of a commercial product for the space domain [33]. Platform-level randomisation uses a pseudo-random number generator developed by I. Agirre et al. [8], which has been shown to adhere to safety integrity level 3 under IEC 61508 [69] safety standard. IEC-61508 is a generic international safety standard for systems comprised of electrical and electronic elements that perform safety functions, upon which domain-specific safety standards (automotive, railway, etc.) are built.

MBPTA has also been applied on top of *time-deterministic* architectures, in which the program's execution time, in theory, does not vary under the same input values and initial platform state. In this case, randomisation is implemented with software techniques that work at the compiler/linker level [74] or source code level [79].

Assuring sufficient execution path coverage is a widely acknowledged requirement of all timing analysis approaches [119, 122]. In fact, the WCET estimates computed by MBTA methods hold only for the execution paths observed at analysis time. Measurement-based analyses typically assume the availability of path traversal conditions information to guide the identification of a subset of structurally feasible paths to be included in the analysis. In the context of MBPTA, two solutions to achieve full path coverage have been proposed: Path Upper-Bounding (PUB) [76] and extended path coverage [122]. PUB modifies the source code by injecting *functionally* neutral instructions with impact on timing, such that execution of any path in the modified program has higher latency of that of the original program. This way WCET can be computed with MBPTA using the single input, as it will trigger the execution of some path which is time upper-bound to the original program paths. Extended path coverage, instead, operates on fine-grain measurements at a basic block level to discount the benefits of specific path traversals and obtain path-independent measurements.

### 1.3.3 Problems and Limitations of MBPTA

The trustworthiness of pWCET estimates depends not only on the correctness of the EVT application but also the quality of input data. The latter is assessed through the evidence provided that analysis phase observations are representative of the execution conditions arising during the operation phase. This representativeness issue [54, 108, 122] is a universally recognised open problem for all measurement-based timing analysis techniques. The SoJ influencing representativeness can be broadly categorised as high-level, coming from a program's structure, and low-level, related to the execution platform.

The representativeness challenge is becoming critical as the number of SoJ increases in the novel processor designs. Many hardware features that aim at improving the average performance are dependent on either the history of execution or the data accessed by the program. This is the case of cache memories, which face the challenge of how to obtain representativeness as the number of their possible cache states explodes even for simple programs. In the near future, this issue is likely to exacerbate as the industry is transitioning to deploy more complex applications.

Another consequence of having advanced programs is that it is more difficult for the user to determine which is the required input vector set to obtain path coverage. MBPTA has already made first advancements over the current practices to simplify the process of exercising relevant paths. However, no previous work has studied how the selection of execution paths used during timing analysis influences the low-level representativeness. The aforementioned issue is relevant as the exercised paths directly impact the platform states seen during timing analysis.

## 1.4 Thesis Contributions

This Thesis proposes a methodology and guidance to the user for a trustworthy application of MBPTA techniques. The foreseen impact of achieving this goal is the adoption of MBPTA in the industrial practice, as a reliable and cost-effective timing analysis method for future CRTES deploying high-performance hardware. As other measurement-based approaches, to deliver good-quality WCET estimates, MBPTA needs to operate on the input data that represents the behaviour of the system under potentially unobserved conditions. The representativeness is required both on the software and platform level, as the subset of exercised paths in a program and explored platform states are hugely dependent on the provided input data. This Thesis develops a set of solutions to attain representativeness for input data. MBPTA benefits from EVT and its ability to predict the impact of unseen conditions on timing, reducing the level of information needed for timing analysis. However, EVT poses specific requirements on the platform and the input data which are elaborated in this Thesis. The contributions of this Thesis are based around three main themes which are described next:

I. Cache representativeness

II. Representativeness and path coverage

III. Trustworthiness of EVT application

### 1.4.1 Cache Representativeness

MBPTA-compliant platforms aid the timing analysis by reducing the number of hardware resources that need to be controlled by the user to collect the sample of measurements representative of the platform behaviour. In the processor architectures studied so far, the execution time observations used as input for MBPTA capture with high probability the impact of all timing events produced by MBPTA-compliant hardware resources except for Time Randomised Caches (TRC) [73].

In our first three contributions we propose a set of solutions to collect measurement samples representative of conflictive cache behaviours that can occur during the operation phase. Each of them, based on a trace of program addresses generated through inputs provided by the user, identifies the potential cache placements with high impact on timing that are unlikely to be observed in the number of measurements generally required by MBPTA. The probability of observing these placements increases as more measurements are collected, which also increases the application costs of MBPTA, drastically reducing its overall benefit/cost ratio. Our solutions provide means to quantify the probability (risk) of not capturing relevant cache placements and the increased cost of performing more runs. This

provides the system engineer with a mechanism to take a guided decision on the number of measurements to collect, properly balancing the time/effort available for the analysis, the criticality of the software being analysed, and the corresponding safety requirements in the reference application domain.

In particular:

- The first contribution provides an exact way to identify relevant cache placements, but with limited scalability, and proposes the overall process to follow in order to calculate the needed number of measurements. While this solution has prohibitive cost and therefore low applicability in practice, it provides a way to assess the accuracy of the next two scalable solutions.

- The second contribution proposes a technique to find critical sets of memory objects that lead to the highest overhead when mapped together under one of the state-of-the-art placement policies deployed in TRC: hRP [73]. Based on the probability to trigger these critical placements in measurements, the technique derives the minimum number of measurements to collect in order to derive a trustworthy WCET. This solution outperforms previous algorithms by orders of magnitude in terms of execution time cost, to the point of being practical to apply it on real applications.

- The third contribution provides an in-depth analysis of the behaviour of hRP and a more recent placement policy for TRC: RM [65], which shed light on how conflictive cache placements arise differently for each of them. Based on the analysis, this solution proposes a heuristic-inspired methodology for deriving a list of address combination that leads to high miss counts (together with their probability of occurrence) specifically targeting RM caches. From this, the needed number of measurements is computed. The second and third contribution perform similarly and complement each other to allow the analysis of the state-of-the-art TRC designs.

### 1.4.2 Representativeness and Path Coverage

Previous work proposed techniques to achieve path coverage in the context of MBPTA [76, 122], ensuring that measurement samples are representative of all possible program flows, but without addressing the cache-related representativeness. The first three contributions of this Thesis build measurement samples representative of the platform behaviour and in particular of cache memories, relying on the user to provide the input vectors to trigger the worst-case path. The fourth contribution provides an approach that produces trustworthy WCET estimates that hold for all program paths and achieve probabilistically measurable

cache representativeness, thus capturing the impact of both software and platform events on the execution time.

The solution proceeds hierarchically: it first makes modifications to the program under analysis to balance the impact of cache memories across any path and then derives the minimum number of runs required for the modified program factoring in cache layouts resulting in high execution times. The contribution provides a theoretical analysis of the effects of combining these two approaches and demonstrates its practicability through empirical evidence.

### 1.4.3 Trustworthiness of EVT application

Since EVT treats the system as a black box, it is the responsibility of the user to ensure that its parameters and provided input data are appropriate for its intended use. On the one hand, the sample of measurements given to EVT needs to represent the reality that has to be modelled. On the other hand, EVT parameters used for modelling should be adapted to its domain of usage, which is the execution time domain in this case.

The fifth contribution examines different approaches of applying EVT for MBPTA with respect to their viability for industrial applications and highlights the main misconceptions and pitfalls that risk to threaten the soundness of EVT-based WCET analysis. First, it shows the risk of relying solely on execution times randomisation or applying EVT on time-deterministic hardware platforms without ensuring that measurements are representative of all platform-level SoJ. Second, the proposal demonstrates the risk of using one EVT analysis for all paths of a multi-path program and proposes a multiple-bucket application of EVT. Third, the contribution explores the suitability of EVT and Gumbel distribution to analyse execution times originating from different classes of software programs.

## 1.5 Thesis Structure

The rest of this document is structured as follows:

- Chapter 2 describes the necessary background on MBTA and in particular its probabilistic variant MBPTA. It introduces the problem statement and related work on MBPTA representativeness.
- Chapter 3 discusses our experimental methodology and the case studies to assess the proposals.
- Chapters 4 to 6 present our proposals to attain cache representativeness, starting with an exact but expensive method (Chapter 4) and following with two timing efficient methods for hRP (Chapter 5) and RM (Chapter 6).

- Chapter 7 describes the process to achieve cache and execution path representativeness.
- Chapter 8 discusses relevant EVT aspects for its trustworthy application in the industrial practice.
- Chapter 9 concludes this Thesis and recognises the potential future work.

## 1.6 Publications

The publications stemming from this Thesis are the following:

1. **S.Milutinovic**, J.Abella, F.J.Cazorla, 'Modelling Probabilistic Cache Representativeness in the Presence of Arbitrary Access Patterns', ISORC 2016 19th IEEE Symposium On Real-Time Computing, York (UK), May 2016 **(Best Paper Award nominee)**
2. **S.Milutinovic**, J.Abella, F.J.Cazorla, 'On the Assessment of Probabilistic WCET Estimates Reliability for Arbitrary Programs', EURASIP 2017 EURASIP Journal on Embedded Systems 2017(1): 1-16, December 2017
3. **S.Milutinovic**, J.Abella, I.Agirre, M.Azkarate-Askasua, E.Mezzetti, T.Vardanega, F.J.Cazorla, 'Software Time Reliability in the Presence of Cache Memories', Ada Europe 2017 22nd International Conference on Reliable Software Technologies, Vienna (Austria), June 2017
4. **S.Milutinovic**, E.Mezzetti, J.Abella, F.J.Cazorla, 'On uses of Extreme Value Theory fit for industrial-quality WCET analysis', SIES 2017 12th IEEE International Symposium on Industrial Embedded Systems, Work in Progress, Toulouse (France), June 2017
5. **S.Milutinovic**, E.Mezzetti, J.Abella, F.J.Cazorla, 'Measurement-Based Cache Representativeness on Multipath Programs', DAC 2018 55th Design Automation Conference, San Francisco (California), June 2018
6. **S.Milutinovic**, E.Mezzetti, J.Abella, F.J.Cazorla, 'Increasing the Reliability of Software Timing Analysis for Cache-Based Processors', IEEE TC 2019, IEEE Transactions on Computers 68(6): 836-851, June 2019

Other publications:

1. I.Broster, G.Bernat, F.J.Cazorla, C.Evripidou, **S.Milutinovic**, 'Multicore Timing Analysis for Safety-Critical Software', Ada Europe 2018 23rd International Conference on Reliable Software Technologies, Lisbon (Portugal), June 2018

# Chapter 2

# Background

## 2.1 Measurement-Based Timing Analysis

Worst-Case Execution Time (WCET) has been an important concern for decades, with a plethora of methods proposed to estimate it [119], each of them with its benefits and limitations. While Static Timing Analysis (STA) is still (and will continue to be) the preferred technique to deal with relatively simple high-critical systems, Measurement-Based Timing Analysis (MBTA) approaches are the most effective means to analyse cutting-edge systems, exploiting high-performance hardware features. This has not only been directly acknowledged by automotive representatives [98], but also, in recent industrial works, since original equipment manufacturer/Tier1 teams and STA tool providers increasingly resort to MBTA to derive timing bounds for processor architectures like the NXP P4080 [95], Texas Instruments TMS320C6678 [81], and Arm-based SABRE Lite multicore systems [21].

The mentioned trend occurs because STA does not scale to handle the complexity of hardware and software. STA aims at delivering evidence for timing validation and verification models building on formal proofs that are meant to show the soundness of the application of the analysis steps. While this can be agreed to be scientifically sound, the confidence on STA results still builds on the fragile assumption of the correctness of the underlying timing model, which is expected to represent the behaviour of the real hardware being analysed faithfully. The extraordinary complexity of multicore platforms and the lack of enough technical details in manuals cause this assumption to be hardly sustainable in practice [6]. Thereby, the industrial practice is driven to build on empirical evidence (measurements) as the central element to show adherence to timing requirements. In this line, most recent industrial approaches to handle the timing of complex hardware [30] build on requirements that, for the same reasons above, can only be assessed empirically. This includes, for instance, the identification of interference

channels, which necessarily requires an extensive set of tests showing the processor resources in which tasks affect each other's timing behaviour.

Furthermore, empirical evidence and heuristics are widely used in hardware testing processes. For instance, ISO26262 [70] defines different failure rate thresholds for different automotive safety integrity levels according to the diagnostic coverage with respect to residual faults (see clause 9.4.3.6 in ISO26262 Part 5). Fault models used for assessing diagnostic coverage are in many cases restricted to models that can be tested at appropriate abstraction levels. For instance, stuck-at faults can be easily assessed at gate-level, where the model of the hardware is precise enough, and engineers have full controllability and observability of the circuit. However, circuit timing faults depend on physical parameters of the fabricated device, so they could only be reliably tested during post-silicon validation, when controllability, to set the inputs of individual components, and observability, of their outputs, is extremely limited. In the absence of *complete* fault models to assess diagnostic coverage, heuristics are used to generate a sufficiently low number of tests with high enough diagnostic coverage. For instance, tests are typically created with automatic test pattern generators [35, 109], which build upon heuristics (including guided search algorithms). Target test coverage can be as low as 90% even for the highest criticality levels (automotive safety integrity level D), thus leaving up to 10% of the design untested against relevant faults. Overall, *evidence for certification builds upon measurements obtained with heuristics due to the inability to afford exhaustive explorations.*

The WCET estimates obtained with MBTA are reliable to the degree that the user is capable of designing test scenarios whose conditions are close to those that can arise during operation. It is well known that the observable timing behaviour of a program reflects the specific execution conditions incurred in the measurement run, which are not guaranteed to stay the same from analysis to operation. In measurement-based approaches, this problem calls for some form of control by the user. Sufficient evidence for qualification or certification can be obtained, including for the highest criticality functions (e.g. automotive safety integrity level D), only as long as the user can prove the ability to exercise exhaustive control over all the execution conditions considered of consequence.

Looking at hardware only, the use of increasingly complex processors makes achieving the required level of control over all factors of influence on timing, which we refer to as Sources of Jitter (SoJ), much harder. For example, how program objects, such as code or stack, are assigned to memory defines their memory addresses, which in turn determines how they are mapped to cache sets and, ultimately, the program's pattern of hits and misses. Controlling the effect of memory layout to avoid incurring bad scenarios is not always feasible in practice. Existing techniques are typically exploitable only at the end of the development process

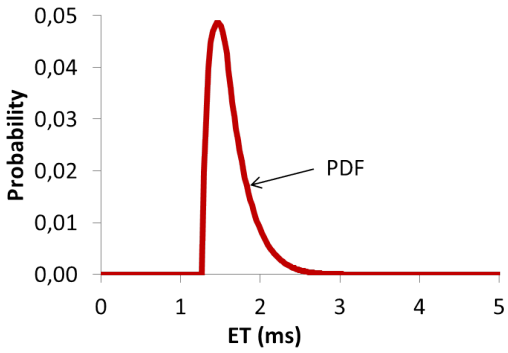
as any analysis result obtained on single software units gets inevitably disrupted after integration. This inherently clashes with the principle of incrementality (Section 1.2.3) in software development and analysis, which is a fundamental cross-domain industrial concern [91]. *This difficulty in analysing and controlling SoJ decreases the confidence that can be placed in the computed WCET estimates, and increases the effort intensiveness of the measurement collection, thereby reducing the benefit/cost ratio of MBTA dramatically.*

## 2.2 Measurement-Based Probabilistic Timing Analysis

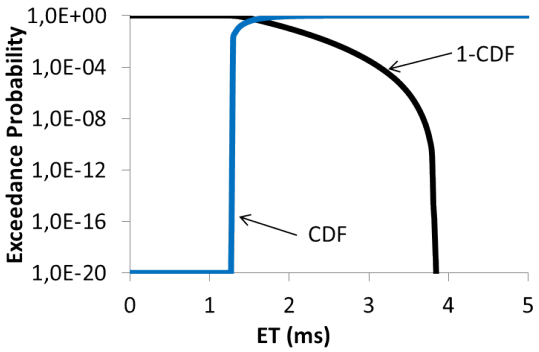
After a seminal paper [45], a lot of effort has been devoted to understanding the challenges of applying statistical approaches to WCET analysis [18, 55, 59]. Measurement-Based Probabilistic Timing Analysis (MBPTA) has been receiving increasing attention due to its promising potential to analyse complex systems while lessening the burden on the user control considerably and warranting soundness. MBPTA has been applied to avionics [116] and space [51] case studies and its impact on certification has also been addressed [113].

MBPTA delivers a probabilistic Worst-Case Execution Time (pWCET) distribution function that describes execution time bounds at different probability levels. This is better understood with the example in Figure 2.1. Figure 2.1a shows the Probability Distribution Function (PDF) of the execution times collected from $R$=1,000 runs of a synthetic program executing on an MBPTA-compliant platform [77]. The corresponding Cumulative Distribution Function (CDF) and the Complementary Cumulative Distribution Function (CCDF or 1-CDF) are depicted in Figure 2.1b in logarithmic scale. With $R$ observations (execution time measurements), one could accurately estimate the pWCET at an exceedance probability of $1/R$ at most. Since much smaller probabilities are needed in the context of critical real-time systems, Extreme Value Theory (EVT) is used to estimate the function that describes the rightmost tail of the execution time distribution. Figure 2.1c shows the result of applying EVT to estimate the pWCET distribution in our example. The dashed line corresponds to the 1-CDF for the 1,000 measurements collected, whereas the continuous line corresponds to the pWCET distribution.

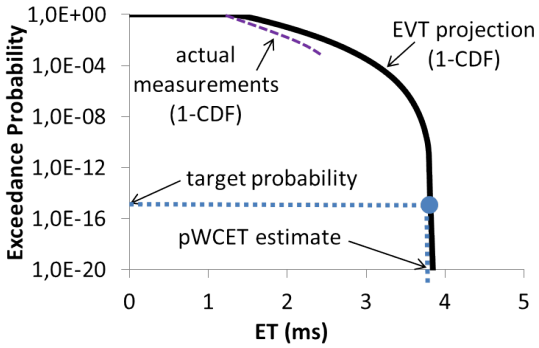
In its original fields of application, EVT is used to produce predictions on the extreme behaviour of the system from observations of it collected non-intrusively: the key (and highly realistic) postulate is that the observer cannot affect the events of interest. When EVT is applied to the execution time domain, the user determines the execution conditions under which the program is run during the timing analysis phase, which can introduce a bias in the measurements being collected. If




(a) Probabilistic Distribution Function (PDF)




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




(c) Example of pWCET curve

Figure 2.1: Synthetic program PDF, CDF, 1-CDF and pWCET curve.

different execution conditions may occur between analysis and operation, different (extreme) timing behaviours may emerge in the two situations: for EVT, they would describe two distinct systems, whose respective extreme behaviours may be far off one another. A simple-minded application of EVT will produce pWCET estimates that *solely* upper-bound the execution time of the program *under the execution conditions captured in the analysis time experiments*, but not necessarily those execution times occurring at operation phase.

In order to ensure that the computed pWCET estimates hold during operation, MBPTA creates a judicious framework of use around EVT. Firstly, it takes care of ensuring that the analysis time distribution of the observation measurements upper-bounds the operation time distribution. Subsequently, it feeds a sample of analysis time distribution, called analysis time sample, to EVT, which can then safely use it to model the tail of the distribution being sampled, which in turn is warranted to upper-bound the tail of the operation time distribution. This is better illustrated in Figure 2.2. The dotted line depicts the Empirical Complementary

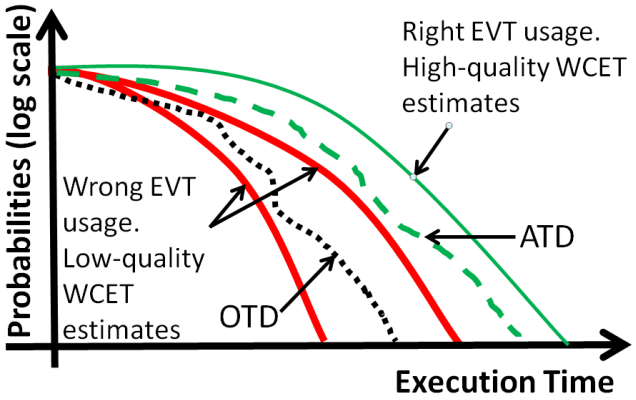



Figure 2.2: Example of EVT projections based on a sample of representative and non-representative analysis time distribution.

Cumulative Distribution Function (ECCDF) of the execution time behaviour of a program, derived from observations of it taken during operation. We plot this operation time distribution for illustrative purposes only since its retrospective nature (which observes operation time events) is too late to feed WCET analysis. The dashed line depicts instead the program's analysis time distribution, as resulting from the execution time conditions in effect during analysis. The solid light line depicts the curve computed by EVT when feeding it with the analysis time sample. MBPTA ensures that analysis time distribution upper-bounds operation time distribution, and makes sound use of EVT to compute high-quality pWCET estimates that upper-bound analysis time distribution, and hence operation time distribution. Incorrect use of EVT might instead result in low-quality pWCET estimates that may fail to upper-bound operation time distribution while still upper-bounding non-representative analysis time distribution.

The example in Figure 2.2 outlines three important steps of the MBPTA process to derive trustworthy pWCET estimates:

- *Ensuring that the analysis time distribution upper-bounds the operation time distribution.* This property holds by construction for MBPTA-compliant architectures (see Section 2.2.1), which makes them the favourable candidate for MBPTA. The relationship existing between the pWCET estimates obtained during analysis and the timing behaviour during operation is analysed by J. Abella et al. [7].

- *Ensuring that the analysis time sample is representative of analysis time distribution.* A representative sample of an execution time distribution captures all the relevant SoJ (see Section 2.3). We have no guarantees that the pWCET estimate built upon a non-representative sample is an upper-bound

of the analysis time and therefore of the operation time distribution. This has been recognised as an open problem [54] and we provide a solution in this Thesis.

- *Guaranteeing a sound use of EVT.* EVT treats the system as a black box, focusing just on its output. MBPTA process needs to ensure the correct application of EVT in our domain (see Section 2.2.2 and our contribution in Chapter 8).

### 2.2.1 MBPTA-Compliant Architectures

The notion of MBPTA compliance has been defined by L. Kosmidis et al. [77, 78], which reflects specific abilities of the observation process and particular features of the execution platform. The latter requires applying combinations of (hardware or software) randomisation or upper-bounding to the execution time behaviour of the hardware resources with the highest jitter.

Applying time upper-bounding at analysis time ensures that the observed program's timing behaviour upper-bounds its timing behaviour during operation. This technique is typically applied to the SoJ that cause a smaller variation on the execution time of a program, e.g. a floating-point unit. These SoJ are forced to work in their worst latency at analysis time, whereas during operation they preserve the original behaviour. By doing so, the user is relieved from triggering the worst-case condition of those resources.

On the other hand, applying time upper-bounding on SoJ that cause a high variation on execution time can easily lead to a large overestimation of the WCET (e.g. it is equivalent to assuming that each access to a memory address misses in cache). In MBPTA-compliant platforms, resources with high jitter are time randomised (both at analysis and operation time), ensuring that the distribution of execution times at analysis upper-bounds that of the program during operation. Note that, in all cases the functional behaviour of the program is unaltered.

With time randomisation all potential behaviours that a given SoJ can exhibit are naturally (and randomly) explored in every new test so that, if enough measurements are performed, the impact of their jitter in execution time is captured. This principle emanates from probabilistic and statistics theory, where a random variable can be modelled based on a sample of observations with increasing confidence and accuracy as the size of the sample grows. The user is not required to explicitly trigger the worst-case behaviour (which is the immensely difficult problem), but only to collect the *sufficient* number of measurements to reach the needed confidence. An example of a time randomised resource is a cache memory deploying both random placement and random replacement. Time randomisation can be realised directly by low-overhead modifications in the hardware or obtained

by software manipulations neutral to functional behaviour on top of commercial off-the-shelf processors.

### Time Randomised Caches

Time Randomised Caches (TRC) [73] are MBPTA's preferred cache designs and their value has been demonstrated on Field-Programmable Gate Array (FPGA) implementations [65]. The original design with random replacement and hash Random Placement (hRP) was proposed by L. Kosmidis et al. [73]. The drawback of hRP is that, due to random allocation of memory addresses across sets, conflicts may occur even when data would fit in the cache. To address this limitation, novel Random Modulo Placement (RM) has been proposed [65] that provides the randomisation required by MBPTA but ensures that addresses stored close in memory are mapped into different sets, thereby fulfilling benefits of modulo placement. To allow the analysis with MBPTA of existing legacy systems featuring deterministic caches, software randomisation techniques have been proposed. They perform by randomising the placement of memory objects (instructions and stack) through compiler/linker support [74] or by inserting random padding and reordering the memory objects at the source code level [79].

**Motivation for TRC.** TRC break the structural dependence between address location in memory and its cache set position. As a result, during the test campaign, users do not need to control the program's code/data memory placement, which is very sensitive to environmental execution conditions that may change across software integration steps. Instead, users just need to make sure that the impact of different cache placements on timing has been accounted for by performing enough execution time measurements at analysis time. This enables performing measurements *in isolation* factoring in the impact of any cache alignment independently of the memory placement produced by future integration. This has the potential of enabling *incremental software integration* – and its benefits – in the presence of caches.

Incremental software integration is the common practice in integrated architectures to handle software complexity and its desired property, *time composability*, is a key concept in the design of Critical Real-Time Embedded Systems (CRTES), see Section 1.2.3. In the timing domain, caches make the timing behaviour of the previously integrated modules change as their memory layout is altered across releases [91]. This has disruptive effects on time composability in systems with deterministic (modulo-placement based) caches. In this type of caches, the particular addresses in which application's code and data are allocated determine the cache sets where they are mapped. Hence the memory placement directly and significantly impacts cache behaviour, especially in the number of conflict misses.

Consequently, as applications are brought together across releases and the memory layout of existing modules changes, WCET figures previously derived for a software unit in isolation, during system early-design phase, are invalidated. This issue is exacerbated when different modules share code (functions) or data.

In the absence of time composability, consolidated timing estimates can only be obtained in late-design phase, in clear contrast with incremental integration and qualification. If timing violations (e.g. time budget overruns) are discovered in late-design phase, costly application re-factoring may be required, changing the system schedule or even requiring additional processing power if the current computation capabilities do not offer an adequate margin to accommodate all the required functionalities. This, of course, may significantly increase the overall product (system) cost and time-to-market.

On deterministic caches, the volatility of WCET figures can be handled by performing fine-grain control of code and data memory placement so that cache alignment is optimised and preserved. However, besides the associated complexity, these approaches are not completely robust to code changes and modification, may increase the burden on the user, may be challenging in practice, and may easily lead to memory fragmentation.

Random placement, instead, breaks the dependence between the memory alignment of program data/code and the cache sets used. Therefore, time composability is not threatened any more by the volatility of memory layout. Since in this case the cache behaviour does not depend on cache placement, *the WCET figures obtained for each software module in isolation hold valid across the layout changes occurred in subsequent software releases.*

**TRC Implementations.** TRC deploy random replacement and random placement policies. Random replacement chooses a cache line for eviction randomly on a miss. Random placement uses a random number, called random index identifier[1] and some address bits to derive the set to map an address. So far two mapping functions have been proposed: hRP and RM.

*hRP* hashes addresses with a random index identifier to derive the cache set to place the address. The random index identifier remains constant during program execution so that an address is always placed in the same set during the whole execution, but it is randomly changed across executions so that the particular set where an address is placed is also random and independent of the placement for the other addresses across executions. Thus, the probability of any two addresses[2]

[1]Random numbers are generated with a pseudo-random number generator that provides sequences with long periods to prevent any correlation.

[2]For the sake of simplicity we assume that the addressable unit matches the size of a cache line.

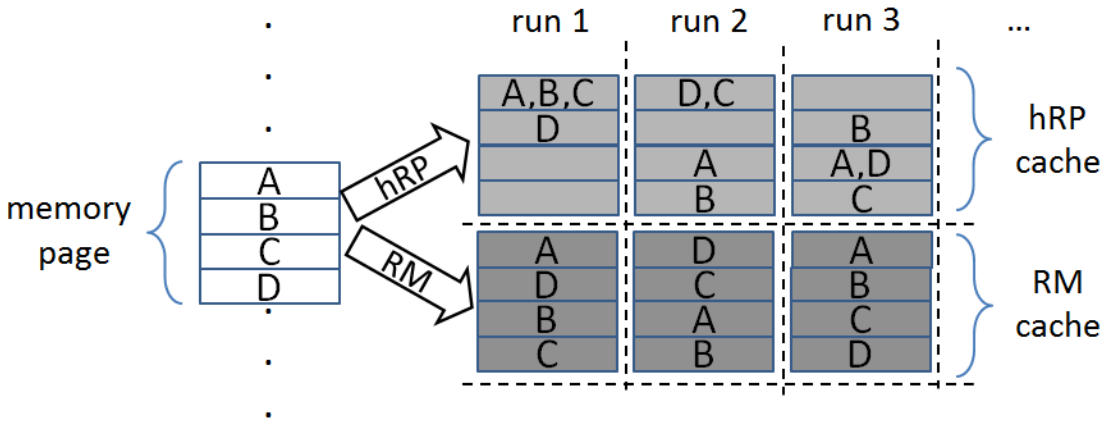



Figure 2.3: Small example for a 4-set hRP and RM cache.

to be placed in the same set is $1/S$ where $S$ is the number of sets. The main disadvantage of hRP is that it exhibits cache conflicts even in the scenarios where all data (or the subset of most-accessed data) is largely below cache capacity since all addresses can be potentially placed in the same set.

*RM* groups the addresses sharing the same memory page into *cache segments* and ensures that all addresses from the same segment are mapped to distinct sets. Randomisation is achieved by using the random number (changed across executions) and some address bits to drive the permutation of index bits to the bits denoting the set to map the address. By avoiding conflicts among consecutive memory addresses (those in the same page), RM better exploits spatial locality in a similar manner as modulo placement, and outperforms hRP in terms of both average and worst-case performance.

As an illustrative example, Figure 2.3 shows how 4 addresses could be mapped to a 4-set cache in different runs. We see that, since the addresses belong to the same memory page, they cannot be mapped to the same set under RM. Instead, with hRP every single address can be mapped to any set.

Note that RM requires that memory page alignment does not change upon integration. Otherwise, WCET estimates obtained under the assumption that some contents reside in the same page (and hence cannot conflict in cache), would be no longer valid if those contents move to separate pages. Thus, RM improves performance with respect to hRP, but also poses some additional constraints. RM preserves the properties needed by MBPTA (independence of actual memory addresses) as long as the page size is equal (or a multiple) of the way size since this guarantees that addresses in a page can be randomly mapped to any set and with even probability. Otherwise, if the page size is smaller than the way size, the location of the page in memory would determine the sets where it would be

mapped, thus making placement dependent on the memory location of the page. Hence, if the page is smaller, then hRP must be implemented instead of RM.

### 2.2.2 Extreme Value Theory

EVT is a branch of statistics that predicts the probability of events more extreme than those that can be usually observed in a sample (i.e. either maxima or minima). EVT has been traditionally applied in meteorological, hydrological, insurance and financial domains [1, 46], to predict extreme characteristics such as exceedance probability or return periods. The application of EVT for the timing analysis of CRTES needs to take into account the particular event EVT is expected to model: the WCET.

EVT is agnostic to the particular variable being measured and how it is measured, as long as some statistical properties hold for the sample. Therefore, when applied to execution time measurements, EVT makes no assumption on the internals of the system (a computing platform in our case) from which the measurements are collected. In our problem domain, EVT has to be understood as a technique to predict the probability distribution of the combined impact of the timing events observed in the provided sample of analysis time measurements. To contribute to the combined effect that EVT seeks to predict, the base events have to be observed, while their cumulative effect needs not. It follows that unobserved events (whose effect on execution time might be arbitrarily large) cannot contribute to the computation [3, 27]: if they can occur and are not captured in the observations, then the predictions are inaccurate and therefore fallacious.

EVT accurately approximates the tail (hence the extreme) of a given probability distribution, taking as input only those observations from the sample that belong to the tail. The block maxima and peak over threshold [50] methods are used to that end.

**Block maxima** defines a block size ($bs$) and splits the sample in smaller groups (e.g. a sample of $R = 2,500$ elements and a block size of $bs = 25$ elements yield $nb = \frac{R}{bs} = 100$ blocks of $bs$ observations each). EVT draws the highest value in each block and creates a sample of maxima to which it fits a probability distribution. Three families of continuous probability distributions are used for fitting in conjunction with the block maxima method: Gumbel, Fréchet and reversed Weibull. They are jointly described by the parametric *Generalised Extreme Value* distribution family. The CCDF of this distribution family is defined by the parameters $\mu$, $\sigma$ and $\xi$, known as the *location*, *scale* and *shape* respectively [34, 80]. Figure 2.4 shows illustrative tail shapes from those three families: a Weibull distribution (light tail) with $\xi = -0.1$ has a sharp slope that converges to a maximum value (1,000 in the figure); a Gumbel distribution (exponential tail) with $\xi = 0$ exhibits a relatively sharp slope, but does not converge to a maximum; a Fréchet

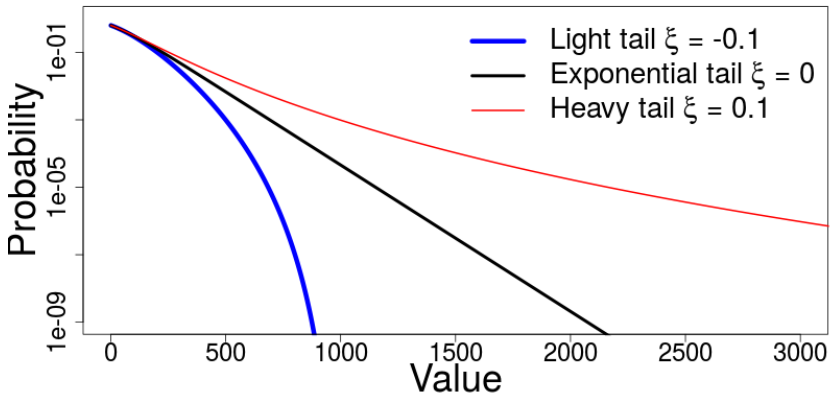



Figure 2.4: Example of CCDF of Generalised Extreme Value distributions with $\xi = -0.1$, $\xi = 0$ and $\xi = 0.1$ respectively. $\mu = 0$ and $\sigma = 100$.

distribution (heavy tail) with $\xi = 0.1$ has the gentlest (polynomial) descent slope, and does not converge to a maximum.

**Peak over threshold** defines a threshold ($th$) value and creates a maxima population drawing from the sample only the observations higher than $th$. The resulting probability distribution function can be described by the *Generalised Pareto distribution* family. This distribution family can be described with either 2 or 3 parameters. In the latter case, $\mu$, $\sigma$ and $\xi$, have the same interpretation as for Generalised Extreme Value distribution (and $\xi$ is identical): for the same $\xi$, and similar values for $\mu$ and $\sigma$, both Generalised Pareto distribution and Generalised Extreme Value yield the same distribution. S. Coles [34] explains how to determine $\mu$, $\sigma$ and $\xi$, and how $\mu$ and $\sigma$ vary for these two distribution families.

The tail shapes produced by the said distribution families are not equally apt to model WCET behaviour: the reversed Weibull family approaches asymptotically an exact (maximum) value, which helps when the WCET is known (but is not our problem domain); the Gumbel family decreases exponentially without converging to a finite upper-bound; the Fréchet family does the same as Gumbel but decreases polynomially, which makes it less tight for our problem domain. The same holds for the Generalised Pareto distribution model distributions. We restrict EVT to using Gumbel distribution [7] on the account that it has been argued to be the most stable (and always overapproximating) distribution to model worst-case execution of real-time software [7, 110]. We discuss the implications of fitting execution times with distributions potentially more pessimistic than exponential tails to the Gumbel in Chapter 8.

## 2.3 MBPTA Representativeness Arguments

The quality of the WCET estimates obtained with any MBTA technique depends on the *representativeness* of the measurements collected at analysis with respect to the timing behaviour of the system during operation [4]. The main concern in MBTA lies in the construction of test cases for the test campaign that exhaustively (and simultaneously) captures the worst-case behaviour of all SoJ. On the other hand, MBPTA controls the impact on the execution time of SoJ during the test campaign. SoJ that cause low execution time variation are enforced to work on their worst latency so that their impact on execution time is upper-bounded in analysis time runs. SoJ that cause high execution time variation are time randomised, such that worst-case timing behaviour is captured with a quantifiable increasing probability as more measurements are taken. Further, MBPTA simplifies the process of collecting observations during the test campaign by deploying EVT. EVT transparently derives the combined probability that the worst-case behaviour of different *SoJ* occur simultaneously in a single run, provided that the worst-case behaviour of each single *SoJ* has been captured in the collected runs. As a consequence, triggering the longest-latency timing of all *SoJ individually* suffices for a trustworthy WCET estimation with MBPTA. This is in contrast to MBDTA that requires all worst-case behaviour to be triggered in a single run. *Therefore, in the context of MBPTA, a representative analysis time sample of measurements needs to capture the longest-latency timing behaviour of each SoJ individually, while EVT is responsible for modelling the combined behaviour of several SoJ.*

For low-variability SoJ, which are enforced to work at their highest latency at analysis, a single measurement is sufficient to capture worst-case behaviour. For high-variance SoJ, which are instead meant to be time randomised, all *relevant events* need to be captured by carrying out *enough runs*. Events of interest, see Figure 2.5, are all those timing events occurring with a probability above a threshold that relates to the corresponding safety standard in the domain (e.g. $P_{rel} = 10^{-9}$). To understand relevant probabilities, we first clarify some concepts related to timing faults and next define probabilities $P_{rel}$ and $P_{obs}$.

**Timing Faults and Safety Standards**. A common misconception in real-time systems is that a program overrun (timing fault) necessarily causes a failure at a system level. In reality, a safety process factors in the impact that timing faults can have on the overall system failure rate. Taking as a reference the ISO26262 [70] standard in the automotive domain, the safety life cycle defines safety goals (and their associated automotive safety integrity level) for each system element. If those goals are reached, the *residual risk* of failure is deemed as sufficiently low. The safety process also defines the safety requirements on the hardware/software to reach the safety goal. Proper measures are put in place to reduce the probability that a fault in a hardware/software element can contribute to the violation of its

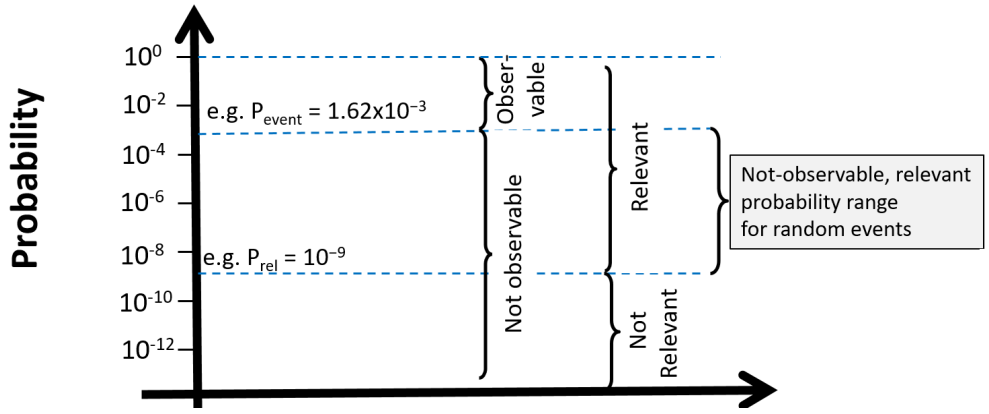



Figure 2.5: Probability range of interest derived from the relevant and observable probabilities.

safety requirements – and hence the safety goal – beyond an established threshold. For instance, random hardware residual faults are considered acceptable if their rate is below a given threshold and the diagnosis coverage – i.e. the mechanisms detecting whether this type of fault can occur for a given hardware block – is below a given target. For instance, for the highest automotive safety integrity level D the maximum allowed failure rate is $10^{-8}$ per hour of operation when the diagnosis coverage reaches 99%.

$P_{rel}$. Due to the representativeness requirement, all relevant events occurring with probabilities higher than a defined probability $P_{rel}$ (see Figure 2.5) need to be properly upper-bounded at analysis time. $P_{rel}$ probability relates to what the corresponding functional safety standard describes as reasonable or unreasonable risk, which is determined by the integrity level of the task and the probability of failure allowed under such integrity level as dictated by the corresponding functional safety standards in the domain. Based on the hazard analysis and risk assessment of the particular functionality implemented by the task, one can determine an appropriate probability threshold ($P_{rel}$). For instance, if $P_{rel} = 10^{-9}$ and a given event occurs with 0.9 probability, the probability of not observing it in 10 trials would be $(1.0 - 0.9)^{10} = 10^{-10}$, and hence irrelevant in this context. In other words, the risk of missing this event with 10 trials is not unreasonable.

$P_{obs}$. All relevant events, whose probability is above $P_{rel}$, need to be accounted for pWCET estimation, which requires that their effect is captured in the measurements collected at analysis time. However, given a number of runs $R$ carried out at analysis, only events with a relatively high probability can be observed in the measurement runs. The probability of not observing a relevant random event at analysis ($P_{nobs}$) is a function of the number of measurements taken ($R$) and the event probability ($P_{event}$): $P_{nobs} = (1 - P_{event})^R$, as presented in Figure 2.5. For

example, let us assume that $R$ = 10,000 measurements are taken during the test campaign at analysis. The events with a per-run probability of occurrence equal to $P_{event}$ may not be properly covered (i.e. missed) with negligible probability, $P_{cth}$, if $(1 - P_{event})^{10000} \leq P_{cth}$. For instance, for $P_{cth} = 10^{-7}$ events with a probability $P_{event} \geq 0.00162$ are captured with a probability higher than $1 - 10^{-7}$. To capture events with lower probabilities of occurrence with enough confidence, more runs are needed.

*Overall, the range of probabilities in which relevant events are unlikely to be observed (for $R$=10,000) is* $P_{event} \in [10^{-9}, 0.00162]$. If such events can occur in the system, the user is required to take an explicit action to ensure their representativeness in the analysis time sample.

### 2.3.1 The Cache-Related Representativeness Challenge

Time-randomised resources other than cache placement have been shown to have a sufficiently high $P_{obs}$ so that the minimum number of runs required by MBPTA already ensures that all their timing events are observed [4].

Set-associative (and direct-mapped) TRC deploy random placement, which makes each address to be mapped to a random and independent set across program runs. Therefore, each run results in random *cache (set) placement*. Previous work [4, 92, 104] identified that certain placements, called Conflictive Cache Placement (CCP), may produce a significant increase in the number of cache misses (and thus execution time), but occur with the probabilities falling into the non-observable and relevant probability range. This affects the trustworthiness of MBPTA pWCET estimates, as in these cases MBPTA is unlikely to capture the impact of this event of interest on the program's execution time. Thus, evidence that those CCPs are sufficiently represented in the measurements passed as input to EVT, known as the *cache-related representativeness challenge*, is needed to compute trustworthy MBPTA results.

So far only the Heart of Gold (HoG) [4] method and its extensions [14, 15] have addressed this challenge. In particular, the authors notice that the number of addresses competing for a set is the critical parameter affecting execution time noticeably: whenever up to $W$ addresses are mapped into the same set, those lines end up fitting in the cache set regardless of their access pattern. This occurs because, after some random evictions, each address can be stored in a different cache line in the set, thus not causing further misses. Conversely, if more than $W$ cache line addresses compete for the cache set space, then they do not fit and evictions will occur often, which is considered a CCP.

However, HoG relies on the assumption that *the impact of all addresses in execution time is homogeneous*, which happens, for instance, in access sequences in which addresses are accessed in a round-robin fashion. We observe that having

more than $W$ addresses mapped to the same set is a necessary condition to trigger a CCP, but it is not sufficient. Whether such cache placement causes an abrupt increase of the execution time *depends on the access pattern for those addresses*. This general case is addressed in this Thesis and solutions are proposed to identify CCPs. The benefit of TRC is that the user does not need to trigger the identified CCP explicitly, but as more measurements are collected, the coverage of possible placements increases. The proposed solutions derive how many measurements are needed to guarantee that relevant CCPs are captured with sufficiently high probability.

### 2.3.2 Representativeness of Execution Paths

Applying EVT on software programs brings the dependence of execution times (and traversed program paths) on input-data into the equation [86, 122]. Static and measurement-based approaches tackle input-data dependence by requiring program features like loop bounds or recursion level to be bounded to derive WCET estimates. Hence, input vectors mainly affect the paths traversed. Yet, path identification and generation of the respective input vectors remain a non-trivial problem for end-users, arising interest in methods to relieve the user from the burden to achieve sufficient path coverage.

With MBPTA achieving full path coverage is possible by building a synthetic upper-bounding path [76] or using the measurements from several paths to artificially derive measurements for all unobserved paths [122]. Next, we introduce Path Upper-Bounding (PUB) [76], since it is the basis for part of the work in this Thesis.

#### Path Upper-Bounding Method

The PUB method [76] builds a modified version of the program ($\mathrm{P}_{pub}$) such that any path of $\mathrm{P}_{pub}$ exhibits an execution time distribution that upper-bounds those of all paths in the original program $\mathrm{P}_{orig}$.

$$\forall i, j \in paths(\mathrm{P}_{orig}) : F(\mathrm{P}^{i}_{orig}(t)) \geq F(\mathrm{P}^{j}_{pub}(t)) \tag{2.1}$$

That is, the accumulated probability ($F(x)$) of any path $i$ of the original program for any execution time $t$ is equal or higher than for any path $j$ of the *pubbed* version of the program. Note that the extended program is used only to generate the pWCET at analysis, while at deployment time the original unmodified program is used.

PUB extends the source code of the original program with a compiler pass that inflates the code of each branch of conditional constructs (e.g. if-then-else, switch). It adds instructions and memory accesses so that each branch in the

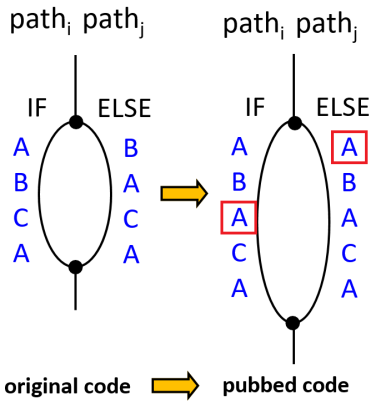



Figure 2.6: Example of pubbed code

*pubbed* code includes equivalent cache access patterns in all of the branches of the original conditional construct[1]. By applying this concept recursively starting from the innermost conditional constructs, operations and memory patterns can be upper-bounded. Interestingly, PUB inserts the needed memory accesses with functionally-innocuous operations (e.g. loading data into a read-only register). PUB builds on input vectors triggering the highest loop bounds and all basic blocks (but not all paths), which is a common requirement of safety standards for code coverage. PUB relieves end users from having to control memory mapping, values operated and paths traversed, which are much less obvious from code inspection.

PUB relies on a key property of TRC that does not hold for time-deterministic caches: given a memory access sequence, adding any memory access in any position necessarily causes a worse probabilistic execution time distribution [76]. For instance, given an if-then-else statement for which the 'if' branch contains the memory access sequence $M_{if} = \{ABCA\}$ and the 'else' branch $M_{else} = \{BACA\}$, the sequence $M_{pub} = \{ABACA\}$ is an upper-bound for both (see Figure 2.6). If we compare $M_{if}$ and $M_{pub}$, the first two accesses ($A$ and $B$) will have the same probabilistic behaviour. If the third access in $M_{pub}$ is a hit ($A$), cache contents are not altered and the rest of the sequence is identical to the remaining part of $M_{if}$. Hence, $M_{pub}$ increases execution time by 1 hit. Conversely, if the third access is a miss, the fifth access ($A$ again) has a higher probability to hit. Thus, $M_{pub}$ will experience 4 misses in the first 4 accesses and a likely-hit in the last access. Instead, $M_{if}$ will experience at most 4 misses, hence having a shorter execution time than $M_{pub}$. We refer the reader to the PUB technique for further proofs, details on instruction cache specifics and memory alignment issues [76]. Note that, this

[1]For clarity we refer to the sequence of addresses of one path, regardless of whether they are instructions or data since the reasoning provided applies to both.

method is not compatible with time-deterministic caches since, for instance, least recently used replacement policies may lead to lower execution time by inserting accesses. E.g. in a 2-way cache the sequence $\{ABCA\}$ experiences 4 misses (all accesses), whereas $\{ABACA\}$ experiences only 3 (second and third occurrence of $A$ are hits).

## 2.4 Related Work

WCET analysis has been broadly investigated both by conventional (deterministic) and probabilistic timing analysis. We refer readers interested in the conventional timing analysis techniques to the survey by Wilhelm et al. [119]. A recent study surveys the probabilistic timing analysis techniques [28]. Previous work performed an initial comparison between MBPTA and SDTA [3]. Results show that MBPTA, the central point of our study, provides competitive results with respect to SDTA techniques with lower information requirements for the application of the analysis.

The problem of timing analysis in the presence of single-level private caches deploying random replacement has also been approached in the context of SPTA [12, 31, 84]. The solutions offered in these works are complementary to the ones presented in this Thesis, and the user will choose the appropriate method according to the tradeoff of the advantages and disadvantages offered by STA and MBTA approaches. We note that the representativeness problem is only relevant for MBTA.

Cache placement has also been recognised in other related work as a key element in the amount of conflict misses experienced and consequently a great contributor to the execution time. To that end, several works propose average and worst-case cache optimization approaches [48, 61, 87, 99]. These techniques, whose focus is on modulo placement, rely on deriving the number of cache conflicts by either statically analysing or profiling a program under a heuristically determined subset of cache placements. This information is later used to select the cache placement (among those observed) that produces the minimum number of conflicts. These approaches, however, are built on the assumption that a cache layout is robust against program variations and incremental system integration, widely spread in automotive and avionics among other domains (see Sections 1.2.3 and 2.2.1). Our interest in CCPs is not in finding an optimal one (which is indeed a fragile concept), but rather in providing guarantees that relevant CCPs are covered in the analysis test campaign.

Several efforts have been directed to studying the statistical requirements and support needed to effectively use EVT. The initial works [55, 59] claim that observations provided as input to EVT have to be independent and identically distributed (i.i.d). MBPTA-compliant architectures guarantee that the i.i.d. requirement is satisfied by construction [78] and allow probabilistic reasoning on the

representativeness of input data and the quality of results. The i.i.d. requirement has later been softened [34, 56, 107] showing that EVT can be applied on data exhibiting stationary or weak dependence. This observation led to the definition of well-structured approaches to apply statistical tests on the sample data and to assess the goodness of fit of the model [56]. This has paved the way for the application of EVT tools to programs running on deterministic (i.e. non-randomised) hardware [17, 49, 107]. In this line, some authors consider together the use of EVT on deterministic hardware whenever different program runs may exhibit some form of dependence among them (e.g. applications with multiple execution modes that follow specific patterns) [57]. However, these usages lack explicit means to address low-level timing effects beyond the reach of the end-user. Those effects are the ones accounted for with randomisation to sustain representativeness against those sources of execution time jitter.

Our work is based on the assumption that randomisation is provided by the underlying MBPTA-compliant platform. Other works achieve randomisation by randomly sorting or selecting values from the measurements [55, 88] or by appending random padding to the observed measurements [85]. To the best of our knowledge, the representativeness challenge has not been studied on the specific, other than MBPTA-compliant platforms. An abstract procedure has been proposed to guide the measurement process without assuming the specific platform [90]. The process is considered representative if for some value $k$ of collected measurements a delivered pWCET is close to the ideal pWCET. However, the work leaves as an open issue how to attain the representativeness. Obtaining pWCET estimates in the presence of discrete and/or dependent data has been addressed by G. Lima and I. Bate [85]. Other studies analysed the impact of selecting different EVT parameters and confidence intervals [108]. Besides EVT methodology, several works investigated the computation of pWCET by using Copulas [18] and Markov models [42, 53].

An evaluation framework to assess the quality of results derived by MBTA techniques has been proposed [83] and the model was instantiated to assess the reliability of MBPTA. The results show that the quality of the derived WCET estimates is highly reliant on the achieved path coverage. Achieving full path coverage in MBTA has been studied from different angles. Some works attempt to reduce programs to a single execution path building on predicated instructions [102], but its requirements and cost are difficult to leverage with industrial practice. Other approaches, instead, use genetic algorithms and model checking to generate input vectors to achieve full path coverage [22, 117].

Regarding multicore architectures, pWCET estimates computed with MBPTA hold valid by construction for multicores if hardware shared resources are MBPTA-compliant [116]. Instead, if no hardware support is in place, contention in shared

resources can be accounted for by adding on top of the derived bound for isolation conditions, the possible contention suffered [43]. Then, the contention is monitored during operation to check that the margin added is not exceeded [95, 96]. EVT was also applied to estimate the worst-case timing behaviour of highly-parallel applications running on GPGPU [16].

Hardy et al. [60] propose two reliability mechanisms to mitigate the impact of faulty cache blocks in deterministic instruction caches on pWCETs. The impact of transient and permanent faults in time randomised caches has been covered both in the scope of SPTA [32] and MBPTA [111].

Apart from WCET estimation, probabilistic analysis has also been applied to scheduling [24, 68]. However, applying probabilistic timing analysis to the WCET computation does not require probabilistic scheduling. Instead, the user can derive the pWCET distribution for each task and select a WCET estimate at the probability defined by the safety standard. Then, standard scheduling approaches can be used.

# Chapter 3

# Experimental Methodology

## 3.1 Processor Modelling Framework

The overall goal of our experimental setup is to evaluate the accuracy and efficiency of Measurement-Based Probabilistic Timing Analysis (MBPTA) enhanced with the solutions proposed within this Thesis. On the one hand, the accuracy is measured by comparing the Worst-Case Execution Time (WCET) estimate delivered by MBPTA against empirical distributions for more than $10^6$ execution time measurements at different probability levels of interest. These measurements have been collected by running benchmarks on the modelled hardware, as it allowed the collection of bigger execution time samples. Let's call $ET^p_{mbpta}$ the probabilistic Worst-Case Execution Time (pWCET) delivered by MBPTA at probability $p$ and $ET^p_{emp}$ the execution time attached to the same probability level $p$ read from the Empirical Complementary Cumulative Distribution Function (ECCDF) built from collected measurements. Then, the accuracy metric can be computed as:

$$Accuracy = (ET^p_{mbpta} - ET^p_{emp})/ET^p_{emp}$$

On the other hand, the efficiency is measured in terms of the timing cost to perform the analysis needed to deliver the number of measurements to collect. We use as a reference point the Representativeness Validation by Simulation (ReVS) method, which provides the assuredly accurate results at the price of a generally unaffordable computation cost. Let's call $TC^p_{mbpta}$ the time cost of MBPTA enhanced with the proposed solution and $TC^p_{revs}$ the execution time of ReVS. Then, the efficiency metric can be computed as:

$$Efficiency = TC^p_{mbpta}/TC^p_{revs}$$

The reference architecture for this Thesis is an MBPTA-compliant architecture, with TRCs implemented in hardware, using random replacement, hash Random

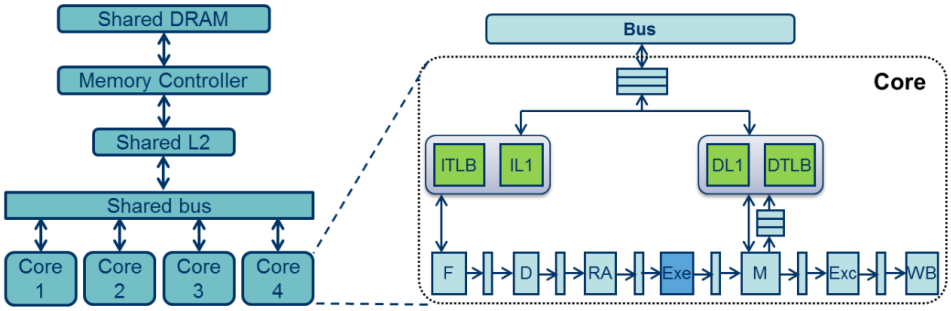



Figure 3.1: Schematic of the LEON3 4-core multicore architecture [64].

Placement (hRP) or Random Modulo Placement (RM) placement policies. In our experiments, we collect end-to-end measurements running a task in isolation and flushing caches across different executions. In particular, we use three different infrastructures:

- **Field-Programmable Gate Array (FPGA) prototype:** Within case studies performed in this Thesis we collect execution times measurements on top of a LEON3-based multicore FPGA prototype used in the space domain [64], whose schematic is shown in Figure 3.1. It comprises of four cores implementing SPARC V8 architecture interconnected by an AMBA AHB bus. The memory hierarchy comprises of private L1 caches in each core and a shared L2 cache and DDR2 memory controller. The FPGA prototype implements techniques in hardware to support MBPTA-compliance: the time upper-bounding technique is applied to the floating-point unit, and time randomisation to cache memories and the bus. The time randomised bus implements an enhanced random permutation arbitration [71], which defines time windows divided into as many slots as the number of cores in the system. Each core is assigned one random slot within each time window.

  Instruction and data addresses of program instructions are collected with the standard debug interface support present in our LEON3-based FPGA board. Notably, other processor architectures provide similar or more advanced tracing features, e.g. the Nexus Interface for NXP processors and the Coresight for Arm processors. Since address tracing can affect execution time, time measurements are collected only with the address tracing mechanism disabled. Address traces are collected in a single separate run, for which timing is not considered. The obtained data and instruction traces are used to perform analysis of our proposed methods, as well as to generate random-cache simulations. Note that this is possible because we are able to decouple the cache-set mapping from the address of the memory instruction.

Table 3.1: SoCLib-based simulator configuration.

| | |
|---|---|
| Architecture | PowerPC 750 |
| Pipeline | 4-stages, in-order, single issue |
| Branch Prediction | No, stalls until resolved |
| Floating Point Unit | Yes (implementing time upper-bounding) |
| Memory Management Unit | Yes |
| Multicore | No |
| Caches | Split L1 Instruction and Data caches |
| IL1 and DL1 Cache Size | 32B per line, 2 ways, 64 sets |
| DL1 Write policy | Write-Back |
| TLBs | Split Instruction and Data TLBs |
| ITLB and DTLB Size | 16 ways, 1 way, 1KB per page |
| Placement policies | hRP and RM |
| Replacement policies | Random eviction on miss |
| MSHR | 32 entries |
| Coherency | None |
| Memory request latency | 16 cycles |
| Memory read latency | 10 cycles |
| Memory write latency | 9 cycles |
| Write buffer | Single pending write |

- **SoCLib simulator**: In order to understand the program's timing behaviour at lower probabilities, and have a ground-truth reference, we resort to collect very large samples of execution times (in the order of millions of measurements). Although *a priori* an FPGA is faster than a cycle-accurate simulator, the limited access to that shared resource and the inability to parallelise the work make the timing cost of collecting that number of measurements on the real board prohibitive. To overcome this issue, we run some experiments in the execution-driven, cycle-accurate simulator based on the SystemC component library SoCLib [112]. Our simulator has been developed by the Computer Architecture/Operating System interface (CAOS) team at Barcelona Supercomputing Center (BSC) and it was already deployed in multiple research projects (PROARTIS, PROXIMA). Many simulations can then be run in parallel in appropriate computing clusters without having to serialise experiments, as it occurs in the FPGA prototype.

  Our SoCLib-based simulator models an architecture featuring a single-core four-stage (Fetch, Decode, Execute and Write-Back) pipelined in-order processor, with the features shown in Table 3.1. The memory hierarchy consists

of separated Instruction Cache Level 1 (IL1) and Data Cache Level 1 (DL1). Our cache module is fully custom and highly flexible, allowing us to evaluate multiple cache configurations, different cache policies and cache latencies.

We configure the IL1 as read-only and the DL1 cache as write-back/write-allocate. DL1/IL1 access latency is 1 cycle for hits with 3 extra cycles for misses, which are added to the main memory request latency (16 cycles unless otherwise stated). The selected configuration support virtualization, having both instruction and data Translation Lookaside Buffers (TLBs) to cache 1KB pages. The cache sizes and replacement policies are configurable, as described in Section 3.1.1.

- **Cache simulator:** The solutions developed in this Thesis propose measuring the cache miss count as part of the MBPTA process, in particular for achieving representativeness. However, a lot of simulation time may be required to get such level of representativeness using a cycle-accurate simulator like SoCLib. To accelerate the development cycle without losing precision we have implemented a light-weight *cache simulator* written in C++ that allows the quick measurement of the miss count metric under different random cache placements. The simulator allows enforcing a group of addresses to be mapped to the same set, such that we can assess the impact of specific cache placements and identify the ones causing the highest number of cache misses. The simulator is trace-driven, taking as input sequences of memory accesses to the data and instruction caches, e.g. such as those traced from the FPGA prototype. It supports random replacement policy and hRP/RM placement policies. Writing policies are write-back and write-allocate.

### 3.1.1 Cache Configurations

We use three different cache configurations across the Thesis: a baseline configuration sized according to the benchmarks' size, a small configuration that further stresses the proposed techniques, and the FPGA configuration for the assessment on a commercial product. The specific setups are as follows:

- **Default configuration**: 4KB 2-way set-associative 32B-line instruction and 32B-line data caches. This is our baseline configuration for experiments run in the simulators. Caches have been sized in accordance with the requirements of the benchmarks used in the evaluation, thus not being oversized in relative terms.

- **Small configuration**: 512B 2-way set-associative 32B-line instruction and 32B-line data caches. These uncommonly tiny configurations are intended

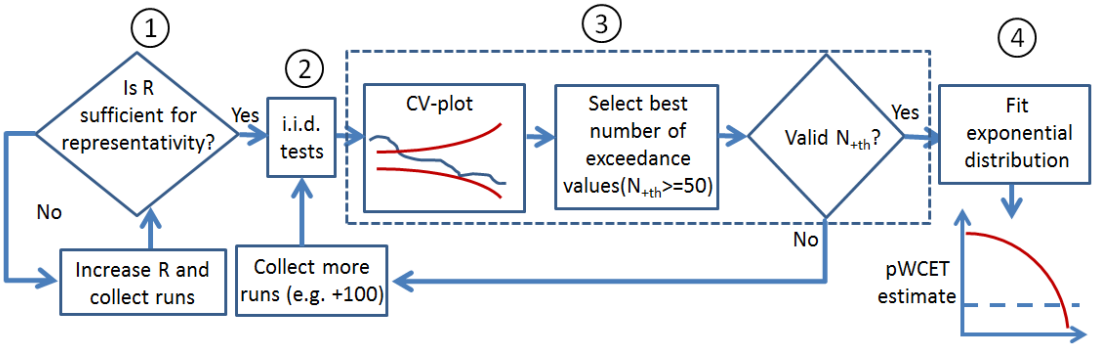



Figure 3.2: Steps in the application of the MBPTA-CV technique [7].

to exacerbate cache conflicts and allowed us to see the impact of some techniques in very stressful scenarios.

- **FPGA configuration**: 16KB 4-way set-associative 32B-line instruction and 16B-line data caches. This setup matches the one implemented in the LEON3-based FPGA and we use it for all case studies run on top of the board.

## 3.2 Timing Analysis Tools

To compute WCET estimates, we use MBPTA-CV performance tool [2, 7]. The overall process consists of four steps, as illustrated in Figure 3.2.

1. **Collecting observations.** First, execution time measurements are collected on the target platform and provided as input to the tool. The solutions developed in this Thesis are integrated into this step to compute how many measurements $R'$ are needed to ensure representativeness. For comparison and assessment of our solutions, we also apply MBPTA-CV on default $R$ numbers without the representativeness step. The lowest value of runs $R$ that we inspect is 300.

2. **Testing independence and identical distribution (i.i.d.) assumption.** The next step tests whether the collected measurement sample follows i.i.d. hypothesis. With MBPTA-compliant platforms, i.i.d. is attained by construction and, therefore, if a particular sample fails these statistical properties, it will eventually meet them as we increase the sample by collecting more measurements.

3. **Identifying tail values.** MBPTA-CV builds upon the coefficient of variation distribution to identify the sample size of tail values, for which exponentiality cannot be rejected with a confidence of 0.95. It passes to the next step the sample of tail values that best fit to the exponential tail, with at least 50 values, and whose exponentiality cannot be rejected for any smaller sample of the highest execution times (with the exception of samples with less than 10 values, which are not sufficient to derive reliable distributions).

4. **Fitting exponential distribution.** Based on the selected tail values, the method derives parameters to model pWCET as an exponential tail distribution, which has been proven to fit well with the WCET estimation problem. Finally, the obtained Extreme Value Theory (EVT) distribution is used to derive the WCET estimate at the chosen probability threshold (e.g. $10^{-12}$ per run).

## 3.3 Benchmarks

To evaluate our proposals, we use two benchmark suites widely adopted in the real-time community: EEMBC Automotive (Section 3.3.1) and Mälardalen (Section 3.3.2) benchmark suites, as well as a railway case study (Section 3.3.3). All benchmarks have been compiled targeting the specific architecture of the simulator/FPGA prototype used on each experiment using *static* and *-O2* optimization flags. Benchmarks have been executed in bare-metal, i.e., without any Real-Time Operating System (RTOS) controlling the program execution. This allowed us to run the benchmarks directly in hardware without preemption or additional overheads caused by task scheduling. We analyse the complexity of the benchmarks used in this Thesis in Section 3.3.4.

### 3.3.1 EEMBC Automotive Suite

We use the EEMBC AutoBench 1.1 Performance Benchmark Suite, which is representative of some safety-related real-time automotive applications [100]. The function of individual benchmarks from the suite is given in Table 3.2, which involves generic tests that include bit manipulation and cache-busting, basic automotive algorithms like controller area network or angle-to-time conversion, and signal processing algorithms commonly embraced by sensors in the automotive domain. The benchmarks comprise the main loop with few function calls in its body. Each iteration of the loop processes different input data embedded in the application.

Table 3.2: Description of EEMBC Autobench benchmark suite.

| | |
|---|---|
| a2time | Angle to Time Conversion |
| aifftr | Fast Fourier Transform (FFT) |
| aifirf | Finite Impulse Response (FIR) Filter |
| aiifft | Inverse Fast Fourier Transform (iFFT) |
| basefp | Basic Integer and Floating Point |
| bitmnp | Bit Manipulation |
| cacheb | Cache "Buster" |
| canrdr | CAN Remote Data Request |
| idctrn | Inverse Discrete Cosine Transform |
| iirflt | Infinite Impulse Response (IIR) Filter |

Table 3.3: Description of Mälardalen suite.

| | |
|---|---|
| bs | Binary search |
| cnt | Counts non-negative numbers in a matrix |
| crc | Cyclic redundancy check computation |
| edn | Finite Impulse Response (FIR) filter calculations |
| fdct | Fast Discrete Cosine Transformation |
| fir | Finite Impulse Response filter over a 700 items long sample |
| insertsort | Insertion sort on a reversed array |
| janne | Nested loop program |
| jfdc | Discrete-cosine transformation |
| matmult | Matrix multiplication |
| ns | Search in a multi-dimensional array |

### 3.3.2 Mälardalen Suite

Mälardalen WCET benchmarks [58] are a popular benchmark suite used in academia in the real-time domain. We use a subset of benchmarks with descriptions given in Table 3.3. Benchmarks follow simple structures, with a few small loops with upper-bounded number of iterations. The input data is embedded in the application (the program contains its own inputs) and for the majority of benchmarks, the input leading to the worst-case path is known.

### 3.3.3 Railway Case Study

We use a railway case study that is part of the European railway traffic management system [47] initiative that seeks to define a unique European train signalling

standard. The functionality of the case study is described by I. Agirre et al. [9]. Our focus is on the on-board unit of this system, called the European train control system. The European train control system is a safety-critical embedded system (safety integrity level 4 for EN-50126 [29]) that protects the train by supervising the travelled distance and speed and activates an emergency brake in case threshold values are exceeded. To that end, it relies on the distance and speed measurements from a set of diverse sensors such as wheel angular speed encoders, doppler radars and GPS positioning systems. As a fail-safe system, whenever an over-speed of the train is detected, the European train control system must switch to a safe-state where the emergency brake is active. This safety function shall be provided with the highest integrity level defined in the railway safety standards (safety integrity level 4) and has strict real-time requirements.

For evaluating the railway case study, we have generated 10 different traces that have been collected using different input sets (labelled `TEST0` to `TEST9`) derived by the end-user (IK4-IKERLAN). The path coverage is achieved by the user providing input vectors to exercise the relevant paths for the timing analysis, which is a common industrial practice. Note that, as each trace corresponds to a different execution path, traces should be treated independently when doing timing analysis. Each trace comprises 8,500 lines of code.

### 3.3.4 Benchmark Analysis

Table 3.4 shows important information about the benchmark suites in terms of the number of lines of code, unique instruction and data addresses.

The first thing worth to mention is that both the EEMBC Automotive and the Railway case study have around fifteen times more lines of code than Mälardalen benchmarks. The main reason is that the former benchmark suite belongs to the industrial world, and therefore is more complex than the latter suite, which is used in academia. Although the smaller size of Mälardalen benchmarks allowed us to test and evaluate some programming constructs, it makes difficult to see how those algorithms scale in larger programs.

Additionally, we can also observe that the ratio between unique instruction and data addresses changes on each benchmark suite. While EEMBC touches more data locations per instruction (and thus puts more pressure on the mechanisms to place and manage the data accessed during the program), the railway case study behaves the other way around, with the potential of experiencing great benefit if the data placement policies can manage the few data accessed in the program efficiently.

Table 3.4: Lines of code (LOC), unique instruction addresses (UIA) and unique data addresses (UDA) for benchmarks used in this Thesis.

| EEMBC Automotive | | | | Mälardalen | | | | Railway case study | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | LOC | UIA | UDA | | LOC | UIA | UDA | | LOC | UIA | UDA |
| a2time | 2380 | 1575 | 1191 | bs | 114 | 148 | 21 | Trace 0 | 8.5k | 2779 | 569 |
| aifftr | 9437 | 3026 | 15768 | cnt | 267 | 259 | 129 | Trace 1 | 8.5k | 2864 | 570 |
| aifirf | 3534 | 1948 | 2396 | crc | 128 | 324 | 231 | Trace 2 | 8.5k | 2696 | 558 |
| aiifft | 9343 | 2713 | 15769 | edn | 285 | 1187 | 533 | Trace 3 | 8.5k | 2722 | 556 |
| basefp | 5134 | 1301 | 4188 | fdct | 239 | 754 | 62 | Trace 4 | 8.5k | 2733 | 560 |
| bitmnp | 2458 | 4872 | 1222 | fir | 276 | 210 | 1461 | Trace 5 | 8.5k | 2717 | 558 |
| cacheb | 2430 | 1217 | 16830 | insertsort | 92 | 201 | 27 | Trace 6 | 8.5k | 2866 | 568 |
| canrdr | 4591 | 854 | 3105 | janne | 64 | 138 | 17 | Trace 7 | 8.5k | 2994 | 597 |
| idctrn | 17674 | 3560 | 4978 | jfdc | 375 | 619 | 91 | Trace 8 | 8.5k | 1843 | 449 |
| iirflt | 3774 | 3279 | 2194 | matmult | 163 | 271 | 1224 | Trace 9 | 8.5k | 1732 | 444 |
| | | | | ns | 535 | 150 | 642 | | | | |
| Average | 6076 | 2434 | 6764 | Average | 231 | 387 | 403 | Average | 8.5k | 2595 | 543 |

# Chapter 4

# An Exact Method to Reach Cache Representativeness

## 4.1 Introduction

Time Randomised Caches (TRC) are the preferred design for Measurement-Based Probabilistic Timing Analysis (MBPTA) as they reduce the level of control needed for timing analysis and favour incremental software integration (see Section 2.2.1). This makes them the focal point of our study. As deterministic (modulo) set-associative caches, TRC are also sensitive to the placement of the addresses across sets. Previous work [4] has shown that when the number of addresses mapped to a cache set exceeds its associativity ($W$), systematic cache conflicts may occur and potentially result in increased execution times. We call a placement with significant impact on execution time Conflictive Cache Placement (CCP).

With TRC, the user is not required to control the cache mapping to avoid or trigger some specific CCP, as the effect of cache placement is transparently exposed with an adequate number of measurements, which will lead to different random placements, thus exposing any relevant CCP. As CCPs occur with certain probabilities, and due to their random nature when using TRC, with more measurements different placements are explored, and hence, it is possible to derive probabilistic guarantees that relevant CCPs are captured for a given number of measurements $R'$.

The challenge with random placement is that it causes timing events with probabilities high enough to impact the timing budget of the system, but low enough to defy observation in the commonly required number of analysis runs $R$ [4, 92, 104]. For example, for an application that accesses 5 addresses in its execution, the probability that all of them are randomly mapped to the same set in a 32-set 4-way cache is $10^{-6} \approx (1/32)^4$, which can be of relevance for the

domain safety standards. If $R$=1,000 analysis runs are performed, a typical value for MBPTA, the probability of mapping the five addresses in at least one run to the same set is very low ($\approx 10^{-3}$).

However, the relevant events may occur with an as low probability as $10^{-9}$. The number of measurements needed to ensure that these events are captured (see Section 2.3) is, in the general case, too high. Therefore, it is necessary to assess whether CCPs can occur in the system and carefully size the needed number of measurements $R'$. So far, this issue has been solved in a limited scenario where the program accesses memory addresses uniformly [4]. However, access patterns of programs may be arbitrary, since addresses are accessed with different frequencies and with arbitrary interleaving. In this chapter, we propose a solution to identify CCPs *exactly* for the general case of arbitrary address interleaving.

**Contributions**. We present *ReVS*, a method *valid for arbitrary cache access patterns* to identify CCPs with both hash Random Placement (hRP) and Random Modulo Placement (RM) and assess whether probabilistic Worst-Case Execution Time (pWCET) estimates obtained with MBPTA – for a given number of runs – are reliable. Otherwise ReVS provides means to determine the number of extra runs needed.

In particular, we make the following contributions:

1. We present a method based on cache simulations to explore the space of cache random placements (with hRP and RM) and determine those leading to the highest execution times (CCPs) at different exceedance probability thresholds. In particular, we identify their probability of occurrence and their impact in terms of miss count for instruction and data caches. By applying MBPTA on the $R$ miss counts collected from the program by means of simulation, we derive a probabilistic Worst-Case Miss Count (pWCMC) distribution – an upper-bound of the miss count distribution of the program under analysis.

2. If the pWCMC distribution does not upper-bound the derived CCPs, ReVS increases the number of runs iteratively until the pWCMC distribution successfully upper-bounds those scenarios. At that point, the execution time observations with $R'$ runs can be used to derive a pWCET estimate that reliably upper-bounds the impact of the worst cache-placement scenarios.

3. ReVS determines $R'$ based on the analysis of the most accessed addresses in the program. Only a limited number of these addresses can be considered, due to the high computation cost of analysis. In order to understand the impact of dismissing the least frequently accessed addresses, we provide a qualitative analysis together with a quantitative assessment by comparing the results of ReVS for a different number of addresses considered.

4. We evaluate ReVS using the EEMBC automotive suite (see Section 3.3.1). Our results show that, differently to the default application of MBPTA, ReVS allows increasing confidence up to a given user-defined threshold (e.g. $10^{-9}$) by increasing the number of runs whenever needed.

5. So far, in the real-time domain, Extreme Value Theory (EVT) has been applied only to predict execution time [38, 59], whereas in other domains EVT has been applied to measure flow floods, stock min/max values, etc. In this respect, we contribute to extending the use of EVT to other metrics in the real-time domain, in particular to miss counts.

## 4.2 Motivating Example

Next, we illustrate the cache-related representativeness challenge and expose the limitations of the state-of-the-art method for attaining representativeness Heart of Gold (HoG) with two examples. For simplifying the discussion, in this section we focus on direct-mapped caches and hRP, though in the rest of the Thesis our focus is on set-associative caches with both hRP and RM. In the first example, the number of misses generated when a subset of addresses is mapped to a set is the same regardless of the particular addresses chosen, as assumed by HoG method. Hence, both HoG and ReVS are effective in this case. In the second example, different conflicting addresses (i.e. addresses mapped to the same set) produce different miss counts, so only ReVS is effective in this case.

Let $\mathcal{Q}_1 = (A_1 B_1 A_2 B_2 A_3 B_3 A_4 B_4 A_5 B_5)$ be a sequence of memory accesses, whose unique (cache line) addresses are $@(\mathcal{Q}_1) = \{A, B\}$ with $|@(\mathcal{Q}_1)| = 2$. Such a sequence may happen when $A$ and $B$ are accessed inside a loop body. For an $S$-set direct-mapped cache, the probability that $A$ and $B$ are randomly mapped to the same set is given by $P_{event} = S \times \left(\frac{1}{S}\right)^{|@(\mathcal{Q}_1)|}$, so $1/S$ in this case. The probability that in the $R$ measurement runs taken at analysis – in each of which a new random set is given to $A$ and $B$ – there is no run in which both are mapped to the same set, $P(s_A = s_B) = P_{event}$, is given by $\overline{P_{event}(R)} = (1 - P_{event})^R$.

For $R$=1,000, a typical value used for MBPTA, the two rows corresponding to $|@(\mathcal{Q}_1)| = 2$ in Table 4.1 show $P_{event}$ and $\overline{P_{event}(R)}$ for different values of $S$ representative of typical first and second level caches in real-time systems. CCPs are those where $P_{event} \in [10^{-9}, 0.021]$ (for $R = 1,000$), so that the event can occur with a non-negligible probability during operation, and there is a considerable probability of missing this event in the measurements taken at analysis time. All cells correspond to the events occurring with relevant probabilities, but only the cells in grey mark events that are unlikely to be observed in $R = 1,000$ runs. We notice that the larger the cache is, the lower the probability of $A$ and $B$ to conflict

Table 4.1: $P_{event}$ and $\overline{P_{event}(R)}$ as a function of $S$. ($P_{obs} = 0.021$)

| | | Number of sets (S) | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **8** | **16** | **32** | **64** | **128** | **256** | **512** | **1024** | **2048** | **4096** |
| $\lvert@(\mathcal{Q}_1)\rvert$=2 | $P_{event}$ | 0.125 | 0.063 | 0.031 | 0.016 | 8$E$-03 | 4$E$-03 | 2$E$-03 | 1$E$-03 | 5$E$-04 | 2$E$-04 |
| | $\overline{P_{event}(R)}$ | 1$E$-58 | 9$E$-29 | 2$E$-14 | 1$E$-07 | 4$E$-04 | 0.020 | 0.142 | 0.376 | 0.614 | 0.783 |
| $\lvert@(\mathcal{Q}_2)\rvert$=4 | $P_{event}$ | 0.590 | 0.333 | 0.177 | 0.091 | 0.046 | 0.023 | 0.012 | 6$E$-03 | 3$E$-03 | 1$E$-03 |
| | $\overline{P_{event}(R)}$ | 9$E$-388 | 6$E$-177 | 3$E$-85 | 3$E$-42 | 3$E$-21 | 6$E$-11 | 8$E$-06 | 3$E$-03 | 0.053 | 0.231 |

in the same set ($P_{event}$), with MBPTA likely missing the impact of this event when $S \geq 64$ (gray cells).

Let $\mathcal{Q}_2 = (A_1B_1A_2B_2A_3B_3A_4B_4A_5B_5C_1D_1)$ be another sequence with $@(\mathcal{Q}_2) = \{A, B, C, D\}$ and $|@(\mathcal{Q}_2)| = 4$. $\mathcal{Q}_2$ may occur when $A$ and $B$ are accessed in a loop and $C$ and $D$ after the loop. HoG assumes that all addresses have the same impact, so it will determine $P_{event}$ as the probability of *any two addresses* (i.e. $AB$, $AC$, $AD$, $BC$, $BD$ and $CD$) to be mapped in the same set, plus the probability of 3 addresses to be mapped in the same set (i.e. $ABC$, $ABD$, $BCD$), plus the probability of the 4 addresses to be mapped in the same set (i.e. $ABCD$). This will lead to the values in the two rows corresponding to $|@(\mathcal{Q}_2)| = 4$ in Table 4.1. However, the true CCP occurs only when $A$ and $B$ are mapped in the same set ($AB$, $ABC$, $ABD$, $ABCD$). In that case, all accesses are misses and otherwise, there will be exactly 4 misses (cold misses for the 4 different addresses accessed). Hence, in this case, HoG fails to determine $P_{event}$ for $\mathcal{Q}_2$. As a result, for instance, for $S = 256$ HoG determines that the probability of the CCP is 0.023, which is not in the range of interest since it is higher than $P_{obs}$ (0.021). In reality it is $4 \cdot 10^{-3}$, which falls in the range of interest: $[10^{-9}, 0.021]$. In this scenario, more runs are required to provide enough confidence in capturing CCPs in the measurements, but HoG fails to identify this situation.

Overall, *providing evidence* that those cache mappings where at least $W$+1 addresses compete for the same cache set have been observed with a sufficiently high probability increases evidence on whether cache jitter is captured with MBPTA. However, mapping more than $W$ addresses in the same set is necessary, but not sufficient condition for triggering a CCP. This requires being *aware of the actual access pattern of the program under analysis* so that only those cache placements producing a high impact on execution time are considered. If the probability of missing any such cache placement is too high, the number of runs needed at analysis needs to be increased so that confidence in observing those placements is high enough.

## 4.3 ReVS: a High-Level Description

The ReVS method identifies the *(conflictive) combinations of (cache line) addresses*, $aC_i$, with high impact on execution time when they are randomly mapped to the same cache set. ReVS also tightly upper-bounds the probability of occurrence of those scenarios and assesses whether the pWCET distribution derived with MBPTA upper-bounds their impact. The validation is performed in the miss count domain rather than in the execution time domain, and it is applied for each cache memory individually (i.e. instruction and data caches). ReVS relies on miss counts correlating with execution time. While this is usually the case as cache misses have been shown to be one of the major contributors to programs' execution time, we perform a quantitative assessment for our reference processor architecture (Section 4.6.1). Whenever this is not the case, then the impact of cache misses can be disregarded as jitter due to other resources is much larger, and therefore ReVS would not be needed. In that case, the default number of runs $R$ would suffice for a reliable application of MBPTA since so far only cache placement events have been shown to challenge MBPTA reliability [4]. Still, as long as cache misses are one of the main Sources of Jitter (SoJ), the use of ReVS is mandatory for a reliable application of MBPTA.

### 4.3.1 ReVS Main Steps

ReVS includes the following steps:

1. Due to the computational cost of ReVS, only a limited number of the most accessed cache line addresses are kept in the address trace (from which ReVS considers all potential combinations). Our proposals in Chapter 5 and Chapter 6 consider, in a first step, the most accessed $U'$ (cache line) addresses for comparison purposes against ReVS. Then, we evaluate complete address traces since our proposals quickly discard those combinations that cannot be the most conflictive ones, i.e. those that if mapped to the same set cause a low impact on execution time. This allows considering arbitrarily large and complex programs.

2. For each combination of addresses $aC_i$ regarded as conflictive – and also for each group of combinations – ReVS (i) determines its probability and (ii) performs cache simulations in which conflictive addresses are mapped to the same cache set. The probability (obtained analytically) and miss count information (obtained through simulation) allows ReVS identifying those conflictive $aC_i$ leading to CCPs that must be upper-bounded. ReVS uses a light-weight cache simulator for TRC (presented in Section 3.1) to estimate

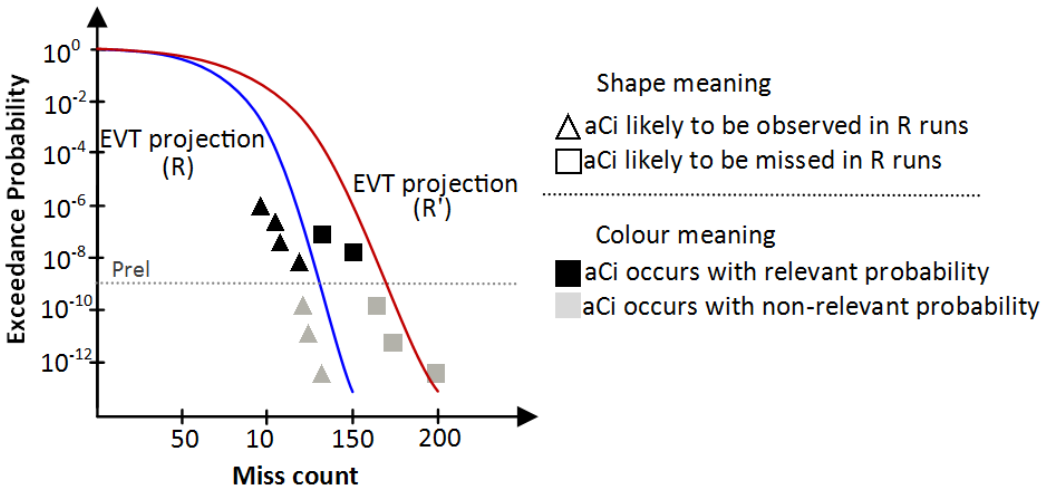



Figure 4.1: Illustrative application of *ReVS* in which address combinations are assessed against pWCMC.

the number of misses when a given $aC_i$ is mapped in the same cache set, where $|aC_i| > W$.

3. ReVS also performs cache simulations in which all addresses are randomly mapped and applies MBPTA with a default number of runs $R$ [7]. ReVS generates a pWCMC with these miss count measurements. By validating whether the pWCMC distribution obtained upper-bounds all CCPs (i.e. miss count and probability pairs), ReVS determines whether the number of runs $R$ used by MBPTA suffices. If this is not the case, more runs are performed until the validation step is passed with $R' \geq R$ runs. Whenever it is passed, the number of runs $R'$ is the minimum number of execution time measurements that MBPTA needs to use.

### 4.3.2 ReVS Process

ReVS process is illustrated in Figure 4.1. The blue curve represents the pWCMC estimate generated from the miss counts obtained from $R$ runs and the black triangles and black squares represent the miss counts obtained for all $aC_i$ – and their combinations – whose probability of occurrence is above $P_{rel}$. Their gray counterparts are those below $P_{rel}$, which are discarded by ReVS since their probability of occurrence is deemed as irrelevant. Triangles are those $aC_i$ (and their combinations) whose miss counts are upper-bounded by the pWCMC, while the miss counts of the $aC_i$ marked with squares are not. In this case, ReVS requires increasing

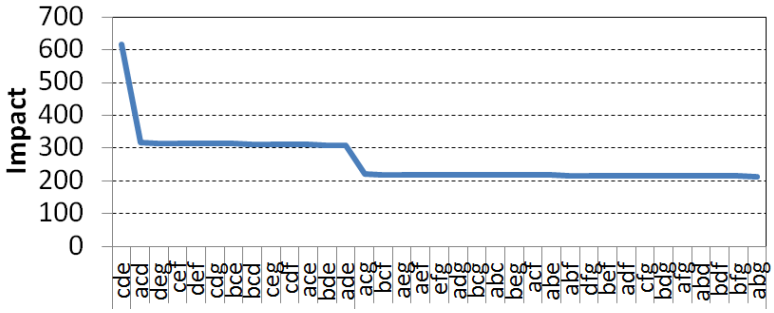



Figure 4.2: Impact (miss count) of different $aC_i$ in the example sequence.

the number of runs, from $R$ to $R'$, such that the impact of those $aC_i$ is properly upper-bounded. As shown, the resulting pWCMC curve with $R'$ runs, marked with red, upper-bounds the impact of all $aC_i$ occurring with relevant probabilities (black squares and triangles). Note that even one grey square may be missed, its probability is deemed as irrelevant, as it falls below $P_{rel}$. Therefore, the pWCET estimate obtained with $R'$ runs upper-bounds the timing impact of all CCPs with sufficient confidence.

### 4.3.3 An Illustrative Example

Let us assume a loop that contains the following sequence of accesses $\Omega_1 = (A_1B_1C_1D_1E_1C_2D_2E_2C_3D_3E_3C_4D_4E_4F_1G_1)$ which is being repeated. In this scenario there are 35 different $aC_i$ with cardinality $K$=3, $\{ABC, ABD, ABE, ...\}$. Figure 4.2 shows the impact when the addresses in each $aC_i$ (shown in the X axis) are forced to be mapped to the same set (in a direct-mapped cache) and the rest are mapped randomly. We observe that $aC_1 = \{C, D, E\}$ generates the highest impact. The second step occurs when two addresses of $<$ ($C$, $D$, $E$) $>$ and any other address are mapped to the same set (e.g. $aC_2 = \{C, D, F\}$). The lowest step in terms of impact occurs when only one or none of the three most repeated addresses is in the address combination (e.g. $aC_3 = \{C, F, G\}$). Intuitively, what ReVS needs to capture is the probability and impact of each step. ReVS considers incrementally only one $aC_i$, e.g. $\{C, D, E\}$, then combinations of $aC_i$, for instance the case where $\{C, D, E\}$ or $\{A, C, D\}$ occur, then $\{C, D, E\}$, $\{A, C, D\}$ or $\{D, E, G\}$, and so on and so forth, thus always considering the worst set of combinations and obtaining the corresponding $< impact, probability >$ pairs. A pair represents impact as the average miss count of one or several combinations, attached to their probability of occurrence. Each of these pairs will be compared against the pWCMC distribution as illustrated in Figure 4.1. This step will be repeated for all cardinalities $K$ (or $|aC_i|$) in the range $[W + 1, |@(\Omega_1)|]$.

## 4.4 ReVS Detailed Steps

In this section, we describe in detail ReVS' main application steps.

### 4.4.1 Generating Combinations of Conflictive Addresses

Let $\Omega_i$ be the sequence of accesses under analysis. In theory, all $aC_i$ such that $|aC_i|>W$ need to be properly upper-bounded. The number of those address combinations is computed as shown in Equation 4.1. Note that the generated $aC_i$ have cardinalities in the range $[W+1, |@(\Omega_i)|]$.

$$N_{aC_i}^{\Omega_i} = \sum_{K=W+1}^{|@(\Omega_i)|} \binom{|@(\Omega_i)|}{K} \tag{4.1}$$

Ideally, we would like to consider all addresses in the program under analysis $U$, such that $|@(\Omega_i)|=U$. However, computation costs to generate all combinations and simulate them in the cache simulator limits the actual number of addresses that can be considered to be up to $|@(\Omega_i)|=U'$, where $U' < U$.This may affect the minimum number of runs $R'$ provided by ReVS. How the address count $U'$ impacts $R'$, and so the confidence of the pWCET estimates, is discussed in detail later in Section 4.5.

### 4.4.2 $aC_i$ Impact and Probability

*Probability.* The probability of a given combination of addresses $aC_i$ to be mapped to the same set is shown in Equation 4.2. The probability of one address to be mapped in a specific set is $1/S$, and so the probability of mapping $|aC_i|$ addresses to a specific set is $(1/S)^{|aC_i|}$. Since there are $S$ sets in the cache, this probability needs to be multiplied by $S$.

$$prob_{same-set}(aC_i) = S \times (1/S)^{|aC_i|} \tag{4.2}$$

*Impact.* The impact is obtained by performing a Monte-Carlo experiment [39] where each observation is a cache simulation. In each simulation, all the addresses in $aC_i$ are forced to be mapped to exactly one random set. The other addresses in $\Omega_i$ are mapped randomly, with no restrictions. The number of observations ($M$) needs to be sufficiently high so that the impact of the random mapping of addresses not present in $aC_i$ is captured. The impact, i.e. miss count in our case, that is produced for the $aC_i$ is the average miss count under all $M$ mappings[1]. In our

[1]The expected value of a random variable is the average value obtained after infinite repetitions of the experiment. In the case of a finite sample, the expected value is approximated with the average of the observed values.

experiments we assume $M = 1,000$, which provides a confidence interval of $\pm 2\%$ with 99% confidence. The inputs for the cache conflict simulator include (i) the sequence of cache lines accessed; (ii) $aC_i$, whose addresses are mapped to the same (random) set, while the rest of the addresses in $\Omega_i$ are mapped randomly; and (iii) the cache configuration. While our work focuses on random placement as well as random replacement, the latter is not strictly needed. Instead, other replacement policies could be used (e.g. least recently used or pseudo least recently used). This could change the impact of the different address combinations. However, ReVS would be applied exactly in the same way. Studying the impact on $R'$ and the pWCET estimates of other replacement policies is beyond the scope of this Thesis.

### 4.4.3 Combined $aC_i$ Impact and Probability

If two combinations of addresses, $aC_j$ and $aC_k$, lead to the same miss count impact, the probability of that miss count is the union of the probabilities of both combinations of addresses. This is because mapping *any* of the two combinations to the same set leads to that miss count. Hence, in addition to considering each combination of addresses ($aC_i$) in isolation, *it is also needed to determine the joint probability of several* $aC_i$. For instance, let us consider an example where $aC_1$ and $aC_2$ have the same impact and $|aC_1| = |aC_2| = K$. Their individual probabilities are $P(aC_1) = P(aC_2) = S \times \left(\frac{1}{S}\right)^K$, but the probability of having *exactly* one of them is $P(aC_1 \cup aC_2) = P(aC_1) + P(aC_2) - P(aC_1 \cap aC_2)$, whereas the impact will be the same. In general, determining the impact and probability of joint scenarios exactly is challenging.

*Probability.* Determining the total probability for the union of any arbitrary number of $aC_i$ is overly complex in practice because we should be able to compute the intersections of each pair of $aC_i$, each group of three, four, and so on and so forth. Note that, $P(aC_j)$ and $P(aC_k)$ are not mutually exclusive in general because addresses may repeat across sets, thus leading to arbitrary intersections for each group. We address this issue by upper-bounding such union of probabilities as their addition. Note that this choice may lead to an increased risk of not passing the validation step because miss count and probability pairs will be more likely to be above the pWCMC. This, however, may imply collecting more runs than needed, but will not lead to false positives in validation assessment against pWCMC.

In extreme cases, some $< impact, probability >$ pairs could be set with such a high probability that the pWCMC never upper-bounds them, thus leading to a failure to pass ReVS. However, false positives cannot occur. Note also that pairs are never disregarded even if they reach $P_{obs}$ (0.021). However, in practice, we would need extremely tiny caches and large address counts to reach $P_{obs}$.

*Impact.* The impact of having any of two $aC_j$ or $aC_k$ with the same cardinal-

ities is obtained as the average of their impacts, since either of them can occur individually with the same probability. Note that individual probabilities for all of them have already been considered, and the case of having $aC_j$ and $aC_k$ simultaneously in the same set is captured when analysing those $aC_h$ with larger cardinality such that $|aC_h| = |aC_j \cup aC_k|$.[1]

For each cardinality $K$ in the range $[W+1, U']$ we analyse the combined impact of those $aC_i$ with higher individual impact. Conceptually, this can be implemented by sorting the different $aC_i$ from highest to lowest impact, and selecting those two combinations with the highest impact, then the three with the highest impact, and so on and so forth. For instance, if we have the following four combinations $aC_1 = \{A, B, C\}$, $aC_2 = \{C, D, E\}$, $aC_3 = \{F, G, H\}$ and $aC_4 = \{A, D, J\}$ for $K$=3, with their respective $< impact, probability >$ pairs $< 100, 0.0001 >$, $< 70, 0.0001 >$, $< 90, 0.0001 >$ and $< 80, 0.0001 >$, we would sort them and obtain: $aC_1$, $aC_3$, $aC_4$, $aC_2$. $aC_1$ in isolation has already been considered as an individual combination. We now consider groups of two, three and four combinations. The group of two combinations includes $aC_1$ and $aC_3$. Its $< impact, probability >$ pair would be $< 95, 0.0002 >$, thus reflecting the average impact and the added probabilities. The group of three combinations includes $aC_1$, $aC_3$ and $aC_4$, and would be represented with the pair $< 90, 0.0003 >$. The group with four combinations includes all of them and is represented with the pair $< 85, 0.0004 >$. Note that there is no way to select groups of 2, 3 or 4 combinations with the highest average impact than the ones chosen, and their probabilities would be exactly the same since all combinations with the same number of addresses have identical probabilities. This step delivers a list of pairs ($< impact, probability >$) that must be upper-bounded by the pWCMC curve.

### 4.4.4 Validation Against the pWCMC

The final step consists in collecting the miss counts for $R$ runs without enforcing any specific placement, so that all addresses are mapped randomly. Then, MBPTA is applied to obtain the pWCMC curve. Those $R$ runs can be performed, for instance, in the same simulator where the $< impact, probability >$ pairs have been obtained. Finally, those pairs are compared against the pWCMC curve.

#### Outcomes of the Validation

Different scenarios may arise for the set of $< impact, probability >$ pairs when assessing them against the pWCMC.

[1] $aC_h$ contains all addresses from $aC_j$ and $aC_k$.

- **Step passed**. If all pairs $< impact, probability >$ are upper-bounded by the pWCMC curve, or the curve falls within their confidence interval, then it can be argued that $R$ runs account for all relevant cache placements. Similarly to any statistical approach, there is some chance that the actual impact of a particular cache placement is larger than estimated simply because it is above the confidence interval estimated. In this case, we make the following considerations:

  1. Safety functional standards accept confidence levels of 99% even for the highest safety integrity levels. For instance, the verification of hardware design in terms of single-point fault metric in the context of ISO26262 in the automotive domain sets the coverage threshold at 99% for the highest automotive safety integrity level D [70] in Clause 8.4.5.
  2. The probability of an estimated impact of cache placement being above the confidence interval is very low ($< 1\%$ due to the 99% confidence). Due to the Gaussian distribution produced by Monte-Carlo experiments, the probability could only be above with decreasing probabilities. Therefore, the actual impact is very unlikely to be above the confidence interval and, if it was, it should be naturally very close to the confidence interval estimated. Therefore, the evidence obtained with this process is in line with the industrial practice since pWCMC reliability is proven to be probabilistically high.

- **Step failed**. If the pWCMC is below the confidence interval for at least one pair, ReVS asks for more runs. However, there is a risk of having a false positive despite the number of runs already suffices to upper-bound all pairs. For instance, the Monte-Carlo experiment may produce, by chance, a particular cache placement with very high impact but that occurs with very low probability. For instance, there is a 0.001 probability that 1,000 observations in the Monte-Carlo experiment trigger a placement occurring once every 1,000,000 times. Such placement, if observed, may shift the confidence interval towards higher impact values, thus making the pWCMC to be below the confidence interval for this pair. In this case, the cost of the false-positive relates to asking the end-user for more execution time measurements of his program, but it does not decrease the reliability of the method.

### Determining the Number of Runs

ReVS starts an iterative process by setting the value of $R'$ to the number of runs required by MBPTA ($R$) [38] . If more runs are required (i.e. pWCMC does not upper-bound all pairs), we increase the number of runs by $\Delta_R = 10$. As the

number of runs $R'$ increases, we also increase $\Delta_R$ accordingly for efficiency. That is, we make $\Delta_R = 100$ when $R'$>1,000 runs, $\Delta_R = 1,000$ for $R'$>10,000 runs, and so on and so forth. Whenever a value of $R'$ is found such that the pWCMC curve upper-bounds all pairs, then we explore the interval in steps of $\Delta_R = 10$ to provide a precise answer, although this last step is not strictly needed.

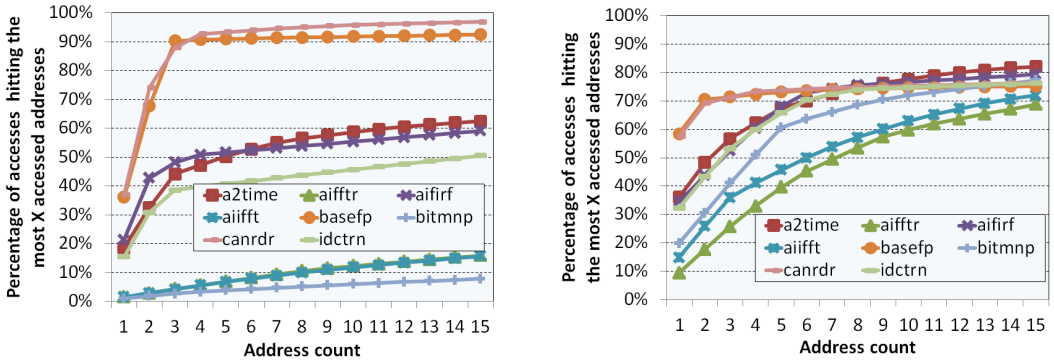
Whenever several caches are analysed, the number of runs to be performed is the maximum $R'$ across all caches obtained with ReVS. In our case we have instruction and data cache so, $R' = max(R'_{dcache}, R'_{icache})$. As miss counts in the instruction and data caches are independent events, it is sufficient to observe their CCPs separately. EVT is in charge of predicting the impact and probability of the different bad address placements for data and instructions to occur simultaneously (see Section 2.2.2). Note that using the maximum $R'$ across all caches may be pessimistic due to several reasons and a lower value for $R'$ could suffice:

- It could be the case that, for instance, $R'_{icache} < R'_{dcache}$, and IL1 placements observed with $R'_{icache}$ runs lead to higher pWCET estimates than those DL1 placements observed with $R'_{dcache}$ runs, thus meaning that only IL1 placements are relevant in practice for this program. In that case $R' = R'_{icache}$ would suffice. However, building such a proof is complex so we resort to using the maximum $R'$ across all caches for reliability of the method.

- The pWCET value will be chosen at a specific probability threshold (e.g. at $10^{-12}$ per run). Therefore, it may be completely irrelevant that the pWCET value at higher probabilities (e.g. at $10^{-6}$ per run) is not a true upper-bound as long as all relevant events are conveniently upper-bounded with the pWCET selected. For instance, the pWCET at $10^{-6}$ could be 100,000 cycles with $R < R'$ runs, whereas execution times of 101,000 cycles could occur at this probability. By using only $R$ measurements MBPTA would fail to observe the worst-case latency at probability $10^{-6}$. However, if the pWCET selected at $10^{-12}$ with $R$ runs is 150,000 cycles and the highest execution time that can occur at that probability is 145,000 cycles, then the pWCET estimate is reliable despite using only $R$ runs.

## 4.5 Reliability Considerations of ReVS

The computational cost of ReVS prevents us to apply it to all program addresses $U$. We limit our application of ReVS to the $U' = 15$ most accessed cache lines, as those are most likely to produce conflictive scenarios under different placements. The computational cost to analyse $U' = 15$ cache lines is around 1.5 hours per cache and per benchmark for EEMBC benchmark suite (Section 3.3.1) on a regular laptop. Assuming 32-byte cache lines, the selected lines represent $15 \times 8 = 120$




(a) Instruction accesses coverage.

(b) Data accesses coverage.

Figure 4.3: Coverage of all memory accesses by the $U' = 15$ cache line addresses for EEMBC benchmarks.

addresses. The number of accesses to these addresses account for 67% of all the memory accesses. In Figures 4.3a and 4.3b we show, for instructions and data respectively, the percentage of the total program's accesses (assuming 32-byte cache lines), covered by the most accessed cache-line addresses. We observe the variable behaviour across benchmarks, especially with respect to instruction access coverage. For some benchmarks the 15 most accessed addresses are sufficient to achieve very high coverage (e.g. `canrdr`, with $> 95\%$ instruction and $> 75\%$ data accesses covered), while for others we observe much lower coverage (e.g. `aifftr` and `aiifft`, with $< 20\%$ instruction accesses covered).

Using the higher number of addresses (e.g. $U'$=20) increases the computational cost of ReVS exponentially due to the exponential increase in the number of address combinations to explore (see Equation 4.1). Therefore, we had to set a limit arbitrarily. We regarded $U'$=15 as sufficient for illustrative purposes and did not go for $U'$=20 since it took around 2 days per cache and benchmark. Thus, increasing $U'$ would only be affordable by decreasing the number of cache simulations per combination (e.g. from $1,000$ down to 100), which would lead to much wider confidence intervals for the $< impact, probability >$ pairs. This would challenge the usefulness of ReVS as too wide intervals would make almost any pWCMC to fall within the interval, thus failing to identify those cases where the number of runs is too low to capture relevant placements.

In this section, we provide an analysis of the potential impact on the confidence of dismissing some addresses.

### 4.5.1 Impact of Address Choice

As a rule of thumb, the most accessed addresses are the ones able to create a higher miss variability with different cache placements, and hence higher execution time variability. This is typically the case since most accessed addresses are the ones with the highest potential to create miss count variations, thus affecting representativeness. This relates to instruction addresses accessed in loops and data accesses with reuse distances[1] long enough not to be mapped into registers. In both cases, this leads to reuse distances often above $W$, so that some random placements may make each of those addresses cause much higher miss counts than usual. However, there are some known, as well as some potential exceptions that we review next.

**Exception 1**. Due to the particular access patterns of the program, the most accessed addresses might have very high hit rates even if placed in the same set as other addresses simply because their reuse distance is pretty short. For instance, in a cache with $W$=4 a program whose access pattern is ($A_1B_1A_2C_1A_3D_1A_4E_1A_5F_1A_6$ $G_1A_7B_2A_8C_2A_9D_2A_10E_2A_{11}F_2A_{12}G_2$) would lead to very high hit probabilities for $A$ despite the particular cache placement. However, addresses $B$, $C$, $D$, $E$, $F$ and $G$ are much more sensitive to the particular cache placement since they may lead to high miss rates if 5 or more addresses are placed in the same set. Hence, although in general higher access counts relate to higher miss counts, and so higher sensitivity to the particular cache placement, this cannot be proven true in all cases.

**Exception 2**. In some cases the addresses considered may be as relevant as some of the addresses dismissed. This is, for instance, the case of instruction addresses in a loop. Let us assume a loop whose code spans to 20 cache lines. By using $U' = 15$, 5 of those 20 addresses will be ignored. In this case, all address combinations with a given address count (e.g. 5 addresses) have the same impact. However, the number of combinations produced with 15 addresses is lower than the one that would be obtained with 20 addresses. Hence, the corresponding pair $< impact, probability >$ with $U'$=15 will have a lower probability than the one obtained with $U'$=20. As a result, for a given $R$ the resulting pWCMC may be deemed as not reliable for $U$=20 (thus requesting more runs), whereas it may be deemed as reliable with $U$=15 (thus not requesting more runs).

### 4.5.2 Impact on $R'$

Using a limited number of unique addresses may affect the final number of runs $R'$ requested to the user. If $R'$ with $U'$=15, referred to as $R'_{15}$, is equal or higher than

[1]Reuse distance is the number of cache lines accessed between two consecutive accesses to the same cache line.

$R'_U$, where $U$ stands for all addresses in the program, then our method may be requesting some extra runs above those strictly needed. This increases the burden on the user side but delivers confidence levels equal to or higher than the desired ones.

Conversely, it can be the case that $R'_{15} < R'_U$. In this case, some risk exists that those $R'_{15}$ runs do not capture all relevant cache placements with sufficient confidence. Still, even in that case, the pWCET is not necessarily optimistic. Other cache placements or other sources of execution time variability may make MBPTA produce a reliable pWCET curve even if the particular event in the example has not been observed in the measurements collected during the analysis phase, which could occur with a probability higher than required (e.g. $10^{-6}$ instead of $10^{-9}$).

In order to understand the impact on $R'$ of different values of $U'$, in Section 4.6.4 we perform an analysis of ReVS comparing $U'$=15 and $U'$=10. This allows us to understand what we lose by discarding the 5 least accessed addresses out of the 15 most accessed ones.

## 4.6 Experimental Results

We run cache simulations and measure miss counts in our light-weight cache simulator (Section 3.1) with two different configurations: 4KB 2-way set-associative 32B/line DL1 and IL1 (default configuration from Section 3.1.1) for hRP and 512B 32B/line 2-way IL1/DL1 (small configuration from Section 3.1.1) for RM. We have chosen a different setup for RM, because by limiting the number of addresses to $U'$=15, the number of accessed pages reduces to 1 or 2 cache segments for most benchmarks, making that no CCP exists for RM in most of the cases. In our small cache configuration, even programs with small address footprints can exhibit a conflictive behaviour. Note that with hRP 15 addresses can easily exceed cache associativity. In the case of hRP reducing cache size would further increase the probability of CCP, which would easily make the default number of runs of MBPTA sufficient, so no insight would be shown if cache size was decreased.

We measure execution times in an analytical model that prototypes an in-order processor with a memory hierarchy comprising the aforementioned configurations for caches and main memory. The latency of an instruction depends on whether the access hits or misses in the instruction cache: a hit has 1-cycle latency and a miss has 100-cycle latency. The memory operations access the data cache so they can last 1 or 100 cycles depending on whether they miss or not. The remaining operations have a fixed execution latency (e.g. integer additions take 1 cycle).

We evaluate several EEMBC benchmarks described in Section 3.3.1. We consider the $U'$=15 most accessed addresses for instructions and data for each benchmark that covers on average 67% of the accesses across all benchmarks. We show

Table 4.2: Average and standard deviation for the reuse distances and instruction/data accesses coverage for EEMBC benchmarks.

| | **Reuse distance** | | | | | | | | **Coverage** | |
|---|---|---|---|---|---|---|---|---|---|---|
| | $U$ | | | | $U'$=15 | | | | $U'$ / $U$ | |
| | **IL1** | | **DL1** | | **IL1** | | **DL1** | | **IL1** | **DL1** |
| | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | | |
| a2time | 7.97 | 28.94 | 2.53 | 8.20 | 0.57 | 1.75 | 1.26 | 2.45 | 62% | 82% |
| aifftr | 3.16 | 9.05 | 7.92 | 71.45 | 0.34 | 0.99 | 0.52 | 0.77 | 16% | 69% |
| aifirf | 0.81 | 3.52 | 2.02 | 9.24 | 0.55 | 1.16 | 0.74 | 1.22 | 59% | 79% |
| aiifft | 3.27 | 9.29 | 7.93 | 73.55 | 0.34 | 0.93 | 0.51 | 0.75 | 16% | 72% |
| basefp | 0.37 | 2.21 | 2.81 | 38.22 | 0.30 | 0.71 | 0.35 | 0.61 | 92% | 75% |
| bitmnp | 9.55 | 27.49 | 1.92 | 4.51 | 0.56 | 1.52 | 1.24 | 0.85 | 8% | 77% |
| canrdr | 0.62 | 1.77 | 1.79 | 11.73 | 0.58 | 1.18 | 0.41 | 0.72 | 97% | 76% |
| idctrn | 0.96 | 3.78 | 2.07 | 14.38 | 0.29 | 0.77 | 0.46 | 0.72 | 51% | 76% |

the coverage for instructions and data of each benchmark individually in Table 4.2. In the same table we present the average reuse distances and their standard deviation for the full traces ($U$) and those with $U'$=15. As shown, there is a wide variety of behaviours across benchmarks, especially for the DL1, thus stressing the ability of ReVS to determine the number of runs $R'$ needed. In order to analyse the impact of dismissing some addresses, we also consider $U'$=10 and compare it against $U'$=15 (Section 4.6.4). For these experiments we use hRP. In all cases, we start by applying MBPTA with the number of runs $R$ regarded as sufficient by the MBPTA technique for each program [7]. Then we apply our approach, ReVS, for the instruction and data caches, and obtain the number of runs required to pass the validation step $R'$.

### 4.6.1 Correlating Execution Time and Miss Counts

ReVS and other cache-related solutions presented in Thesis rely on *the assumption that execution time and miss count are (highly) correlated.* While this is generally the case since cache misses lead to slow off-chip accesses, we perform a quantitative assessment of this fact. We first illustrate such correlation visually for some benchmarks and hRP. Then, we evaluate quantitatively such correlation for the whole set of EEMBC automotive benchmarks (Section 3.3.1) and both hRP and RM. For that purpose, we use our Field-Programmable Gate Array (FPGA) platform, whose specific setup is shown in Section 3.1. Executions on this FPGA take much longer than the ones on our analytical model, whose accuracy has been assessed against the FPGA implementation. However, as shown next, these results

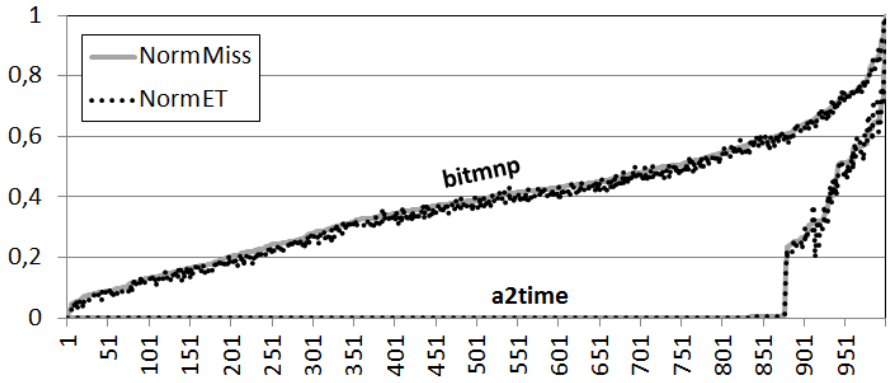



Figure 4.4: $NormMiss$ and $NormET$ with hRP for `a2time` and `bitmnp` sorted by $NormMiss$.

prove that modelling execution time mostly with cache behaviour is an extremely accurate proxy. Both, the cache simulator and the FPGA, implement write-back write-allocate policies in DL1. However, the FPGA includes a write buffer for dirty lines evicted, whereas the simulator does not.

**Qualitative assessment.** First, we perform $R$=1,000 runs for each benchmark collecting both their execution times and their total number of misses in first-level data and instruction caches (DL1 and IL1 misses) with hRP. In order to correlate the variation of both metrics, we normalise them to the values between 0 and 1: for each benchmark we subtract the minimum execution time (miss count) from the execution time (miss count) observed in each experiment. This differential is normalised to the differential between the minimum and maximum values observed. Formally, normalised misses for a given execution $i$, referred to as $NormMiss_i$, are obtained as follows, where $Miss_i$ stands for the number of misses measured in execution $i$:

$$NormMiss_i = \frac{Miss_i - \left(MIN_{j=0}^{R} Miss_j\right)}{\left(MAX_{j=0}^{R} Miss_j\right) - \left(MIN_{j=0}^{R} Miss_j\right)} \tag{4.3}$$

Likewise, we compute $NormET_i$:

$$NormET_i = \frac{ET_i - \left(MIN_{j=0}^{R} ET_j\right)}{\left(MAX_{j=0}^{R} ET_j\right) - \left(MIN_{j=0}^{R} ET_j\right)} \tag{4.4}$$

where $ET_i$ is the execution time measured in execution $i$.

$NormMiss$ and $NormET$ for `a2time` and `bitmnp` benchmarks are shown in Figure 4.4. As shown, both metrics overlap almost completely. Only some discrepancies are observed for `a2time` due to the effects of the store buffer. However,

the average deviation of one metric with respect to the other is 0.4% and 1.5% for `a2time` and `bitmnp`, respectively.

Table 4.3: Pearson and Spearman coefficients for $NormMiss$ and $NormET$.

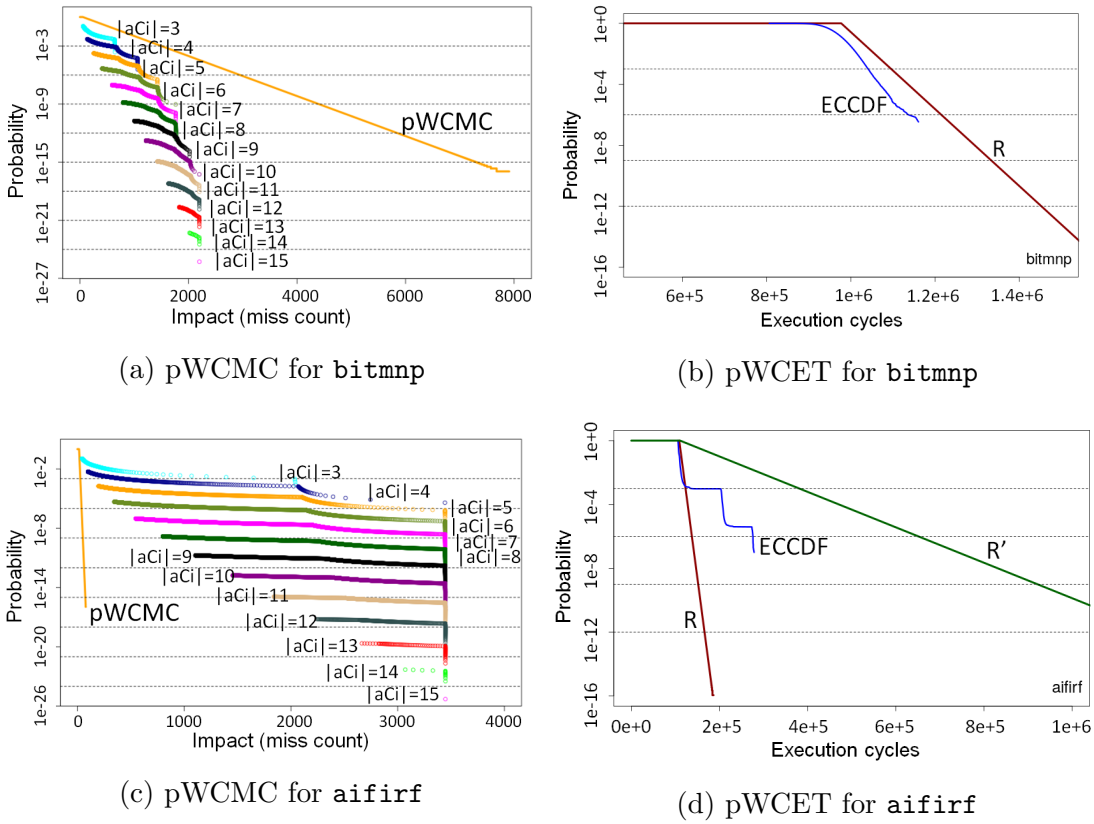
| | hRP | | RM | |
|---|---|---|---|---|
| | **Pearson** | **Spearman** | **Pearson** | **Spearman** |
| `a2time` | 0.997 | 0.992 | 0.976 | 0.970 |
| `aifftr` | 0.918 | 0.911 | 0.969 | 0.960 |
| `aifirf` | 0.960 | 0.956 | 0.988 | 0.986 |
| `aiifft` | 0.923 | 0.913 | 0.960 | 0.952 |
| `basefp` | 0.999 | 0.998 | 0.952 | 0.933 |
| `bitmnp` | 0.998 | 0.998 | 0.975 | 0.960 |
| `cacheb` | 1.000 | 0.970 | 0.992 | 0.988 |
| `idctrn` | 0.950 | 0.951 | 0.969 | 0.973 |
| `iirflt` | 0.997 | 0.979 | 0.977 | 0.997 |

**Quantitative correlation.** In order to assess the correlation between miss counts and execution times quantitatively, we have used two different correlation methods to obtain correlation coefficients [67]: Pearson product-moment correlation coefficient and Spearman's rank correlation coefficient. The Pearson product-moment correlation coefficient measures the linear dependence between two variables. Spearman's rank correlation coefficient measures the statistical dependence between two variables by assessing to what extent those variables can be modelled using a monotonic function. Both methods deliver as output value in the range [-1,1], where 1 indicates a total positive correlation, 0 no correlation and -1 total negative correlation. In our case we expect values close to 1, meaning that there is a linear positive correlation between execution times and miss counts. For both methods, we use a 5% significance level (a typical value for this type of tests [52]).

As shown in Table 4.3, all benchmarks obtain very high values for these tests, so miss counts and execution times are highly correlated and such correlation is highly linear (high values for Pearson's test) for both hRP and RM. We have further analysed benchmarks with the lowest values and have realised that they experience very low execution time and miss count variations. Thus, other SoJ, like those introduced by the store buffer, have a relatively higher impact than for other benchmarks.

### 4.6.2 ReVS Results: Illustrative Examples

To illustrate how ReVS works, we present results for one EEMBC Automotive benchmark passing the validation step with $R$ runs (`bitmnp`) and for one requir-




(a) pWCMC for `bitmnp`

(b) pWCET for `bitmnp`

(c) pWCMC for `aifirf`

(d) pWCET for `aifirf`

Figure 4.5: *ReVS* applied to the instruction accesses of `bitmnp` and `aifirf` (the analysis is performed for combinations of addresses with increasing cardinality, $|aC_i| \in [W+1, U']$) and pWCET estimates obtained with $R$ and $R'$ runs, with hRP.

ing extra runs (`aifirf`), both assuming hRP. For the purpose of this experiment, we perform ten million runs to compute the actual distribution of execution times, Empirical Complementary Cumulative Distribution Function (ECCDF). Note that performing that number of runs is not required for ReVS application; we just perform them for illustrative purposes in this section.

**ReVS passed**. Figure 4.5a shows the result of applying ReVS for the instruction accesses of `bitmnp`. The curves on the left show the $< impact, probability >$ pairs derived with ReVS for each cardinality $|aC_i| \in [W+1, U']$. It can be observed that all $< impact, probability >$ pairs are below the pWCMC curve, thus meaning that the number of runs $R$ suffices for a reliable application of MBPTA for this benchmark. This is corroborated in Figure 4.5b, where the ECCDF is reliably upper-bounded by the pWCET estimate derived with MBPTA with $R$ runs.

Table 4.4: Results for all EEMBC benchmarks with hRP.

| | **ReVS** ($U' = 15$) | | | | **MBPTA / HoG / HoG** ($U' = 15$) | |
|---|---|---|---|---|---|---|
| | $R'_{IL1}$ | $R'_{DL1}$ | $R'$ | likelihood($R'$) | $R$ | likelihood($R$) |
| `a2time` | 58,360 | 540 | 58,360 | $10^{-9}$ | 2,650 | 0.390 |
| `aifftr` | 6,840 | 5,500 | 6,840 | $10^{-9}$ | 2,200 | 0.001 |
| `aifirf` | 21,390 | 11,530 | 21,390 | $10^{-9}$ | 4,400 | 0.014 |
| `aiifft` | 8,920 | 8,770 | 8,920 | $10^{-9}$ | 1,900 | 0.011 |
| `basefp` | 82,080 | 20,010 | 82,080 | $10^{-9}$ | 300 | 0.927 |
| `bitmnp` | 4,640 | 3,510 | 4,640 | $10^{-9}$ | 850 | 0.007 |
| `canrdr` | 18,610 | 7,950 | 18,610 | $10^{-9}$ | 350 | 0.677 |
| `idctrn` | 65,770 | 47,700 | 65,770 | $10^{-9}$ | 3,650 | 0.317 |

**ReVS failed**. In the case of `aifirf`, our method detects that the number of runs obtained with MBPTA $R$=4,400 is not enough to provide a reliable pWCET estimate. In Figure 4.5c we observe that the pWCMC curve does not upper-bound the $< impact, probability >$ pairs generated by ReVS. As a result in the timing domain, the pWCET estimate derived with $R$ runs does not upper-bound the execution time of the program. ReVS requires the number of runs to be increased to $R'$=21,390. If MBPTA is applied in the timing domain with $R'$ runs, the resulting pWCET estimate is reliable as we can observe in Figure 4.5d.

In general, applying MBPTA with $R$ runs instead of $R'$ delivers reliable pWCET estimates. We have corroborated this fact empirically with all the benchmarks and through a number of experiments with different configurations in other works. However, if ReVS is not used there may be some probability of obtaining an unreliable pWCET upper-bound. Still, whether this occurs or not relates to the confidence level obtained with $R$ runs instead of $R'$ as explained later.

### 4.6.3 ReVS Results: EEMBC Automotive

Table 4.4 and Table 4.5 summarise the number of runs required by MBPTA ($R$) and ReVS ($R'$) in the miss domain for both DL1 and IL1 for all benchmarks and hRP and RM, respectively. For the sake of completeness, we also compare ReVS against different flavors of HoG: its original [4] and improved versions [15], considering the full address trace or only $U'$=15 cache line addresses. The number of runs required by ReVS is the maximum across DL1 and IL1. That is, $R' = max(R'_{IL1}, R'_{DL1})$. By comparing $R'$ and $R$ we can assess whether the number of runs required by MBPTA in the execution time domain could lead to lower confidence levels than desired, which occurs when $R < R'$ for MBPTA.

We observe that this is the case for all the benchmarks in Table 4.4 for MBPTA. For RM we observe that the number of runs needed is lower than that for hRP

Table 4.5: Results for all EEMBC benchmarks with RM.

| | **ReVS** ($U' = 15$) | | | | **MBPTA** ($U' = 15$) | |
|---|---|---|---|---|---|---|
| | $R'_{IL1}$ | $R'_{DL1}$ | $R'$ | likelihood($R'$) | $R$ | likelihood($R$) |
| a2time | 1,460 | 8,650 | 8,650 | $10^{-9}$ | 300 | 0.487 |
| aifftr | 480 | 670 | 670 | $10^{-9}$ | 670 | $10^{-9}$ |
| aifirf | 6,300 | 300 | 6,300 | $10^{-9}$ | 300 | 0.373 |
| aiifft | 410 | 8,500 | 8,500 | $10^{-9}$ | 6000 | $4.44 * 10^{-7}$ |
| basefp | 420 | 300 | 420 | $10^{-9}$ | 400 | $2.68 * 10^{-9}$ |
| bitmnp | 370 | 3,570 | 3,570 | $10^{-9}$ | 300 | 0.175 |
| cacheb | 300 | 690 | 690 | $10^{-9}$ | 800 | $< 10^{-9}$ |
| idctrn | 300 | 300 | 300 | $10^{-9}$ | 300 | $10^{-9}$ |

in all cases. Also, RM reduces the number of possible conflicts by construction, so for some examples ReVS does not ask more runs to those required by MBPTA convergence criteria. Regarding HoG (proposed only for hRP), we realise that, for the full address trace, the number of cache line addresses for either data or code is no less than 35 across benchmarks. Regardless of whether we use the original or improved version of HoG, the number of runs required as determined by those methods is upper-bounded by the number delivered by MBPTA. If we restrict the traces to $U'$=15, the same conclusion holds since, for instance, the improved version of HoG makes MBPTA start with 209 runs, which is always upper-bounded by the minimum number of runs required by MBPTA (300).

Note that these results are not independent of the actual cache setup. For instance, if we used a 4-way 32-set cache instead of a 2-way 64-set cache, then HoG improved would request at least 10,955 runs for $U'$=15, which is higher than the minimum number of runs required by MBPTA (300). Therefore the judgment on whether MBPTA number of runs is sufficient differs from the conclusion reached for 2-way 64-set caches. We can conclude that the number of runs needed to achieve the level of confidence desired is highly sensitive to the actual access patterns. Therefore, ReVS is needed in the general case.

On the other hand, this does not mean that results obtained with less than $R'$ runs are unreliable, but the confidence level placed on their sufficiency is lower than the target confidence level. ReVS keeps the likelihood of missing relevant CCPs below $10^{-9}$, as discussed in Section 4.2. Instead, if we only use the number of runs, $R$, determined by MBPTA (HoG), the likelihood of missing those scenarios becomes higher (see the corresponding *likelihood* columns). This decreases the confidence in the results below the levels defined in the corresponding safety standards. Still, it is often the case that relevant scenarios are observed and, whenever they are not, their effect may be superseded by other processor effects. Although this may result in pWCET estimates truly upper-bounding the program's execu-

Table 4.6: Results for all EEMBC benchmarks with hRP [$U'$=10].

| | ReVS | | | |
|---|---|---|---|---|
| | $R'_{IL1}$ | $R'_{DL1}$ | $R'$ | likelihood($R'$) |
| a2time | 78,930 | 1520 | 78,930 | $< 10^{-9}$ |
| aifftr | 14,230 | 19,600 | 19,600 | $< 10^{-9}$ |
| aifirf | 25,890 | 5,400 | 25,890 | $< 10^{-9}$ |
| aiifft | 18,030 | 46,230 | 46,230 | $< 10^{-9}$ |
| basefp | 72,900 | 1,700 | 72,900 | $1.0 \cdot 10^{-8}$ |
| bitmnp | 3,670 | 4,000 | 4,000 | $1.7 \cdot 10^{-8}$ |
| canrdr | 17,750 | 10,000,000 | 10,000,000 | $< 10^{-9}$ |
| idctrn | 65,460 | 58,860 | 65,460 | $1.1 \cdot 10^{-9}$ |

tion time, the lack of evidence on this challenges the development of arguments for certification. Regarding execution time cost, HoG method executes in around 100ms per cache and benchmark on average (in comparison to 27 hours running 100 cache simulations in parallel for ReVS method). This is expected as HoG neglects access patterns and can model the program as the number of unique addresses $U$. Instead, ReVS is a pattern-aware method that tradeoffs computational cost for accuracy.

Despite using different setups, we observe that ReVS asks fewer runs for RM than hRP. This occurs because CCP for RM (if any) can only occur with few addresses because there are few cache segments. Therefore, the probability to capture those CCPs is relatively high and few runs suffice in general. Conversely, with hRP it is often the case that CCPs involve more addresses than those for RM. Therefore, their probability of occurrence is lower and thus, a larger number of runs is required to guarantee that they are effectively captured.

### 4.6.4 Assessing ReVS Reliability

We assess the impact on reliability of analysing a limited number of addresses by comparing the results in terms of $R'$ of applying ReVS considering $U'$=10 ($R'_{10}$) and $U'$=15 ($R'_{15}$). Results for $R'_{15}$ are shown in Table 4.4, whereas results for $R'_{10}$ are shown in Table 4.6.

The first observation is that either $R'_{10}$ is higher than $R'_{15}$ or, if lower, close to it. In particular 5 out of 8 benchmarks meet the condition $R'_{10} \geq R'_{15}$. Therefore, the confidence level obtained is equal or higher than the desired one, but the number of runs requested to the user may increase. For instance, the most extreme cases are those of aifftr and aiifft, where the end-user is requested to increase the number of runs by 3x and 5x respectively with respect to the case where $U'$=15, as

well as the case of `canrdr`, where MBPTA required 10,000,000 runs[1]. The reason for a much higher number of measurements required with $R'_{10}$ is that a smaller number of addresses provide lower variability of execution times. Consequently, EVT needs bigger samples to converge. In those cases where $R'_{10} < R'_{15}$, the difference is between 0.5% and 13.8%. This makes that the confidence level of the pWCET estimates obtained with $R'_{10}$ is slightly lower than desired. In this case, the chance of missing relevant events grows to the range $[1.1 \cdot 10^{-9}, 1.7 \cdot 10^{-8}]$. While this is not desirable, still, the likelihood of these unwanted scenarios can be deemed as extremely low.

If we analyse the results in more detail, we realise that $R'_{10}$ is higher or only slightly lower than $R'_{15}$ for the IL1. This relates to the scenarios described in Section 4.5 for the IL1: the number of addresses accessed in a round-robin manner inside the main loop may be higher than $U'$. By using a low value for $U'$, the probability of producing CCPs is lower in the Monte-Carlo experiment and thus, more runs are needed to produce those placements so that the pWCMC curve upper-bounds all $< impact, probability >$ pairs. However, when increasing $U'$, the true probability of the relevant placements is higher than assumed by ReVS with low $U'$ values. A larger $U'$ value also increases the chances of randomly placing enough conflictive addresses in the same set and thus, triggering CCP, which leads to pWCMC curves upper-bounding all $< impact, probability >$ pairs. Hence, fewer runs are needed to guarantee that those placements are conveniently observed. Runs needed for the DL1 grow in all cases but for two notable exceptions: `aifirf` and `basefp`. This decrease in $R'_{DL1}$ with $U'$=10 with respect to $U'$=15 has no effect on the confidence level achieved since it is masked by the fact that $R'_{IL1}$ is typically higher than $R'_{DL1}$. However, this is not necessarily always the case and thus, discarding some DL1 addresses might potentially affect the confidence level achieved for the pWCET estimates. For instance, if we compute the probability of missing relevant placements only with $R'_{DL1}$, then likelihood for $R'_{10}$ would be $6.1 \cdot 10^{-5}$ and 0.17 for `aifirf` and `basefp` respectively.

In summary, using the most accessed addresses of programs typically allows achieving the desired confidence level for the pWCET estimates. However, ReVS reliability might be affected in some specific scenarios due to the effects of those addresses dismissed due to the computational cost of the method. This motivated our work on extending ReVS to be able to analyse all program addresses within acceptable computation time bounds presented in Chapters 5 and 6.

[1] In fact, due to the difficulties to search for the precise value in our toolchain for big samples, we could only determine that the pWCMC was not upper-bounded with 1,000,000 runs, but it was with 10,000,000 runs.

## 4.7 Summary

In this chapter, we introduce a representativeness challenge related to the cache placement. In particular, certain cache layouts with high impact on execution time may occur with a probability so low that they are likely to be missed during the analysis phase. This leads to some risk of not capturing all relevant events affecting execution time during analysis runs. Therefore, the confidence placed on the WCET estimates obtained is lower than desired. While this challenge has already been addressed for programs with homogeneously accessed addresses (with HoG method), access patterns are arbitrary in the general case.

We propose a validation step for MBPTA, needed to attain the desired confidence in pWCET estimates for arbitrary memory access patterns. Our method, ReVS, identifies the worst miss counts and their probabilities of occurrence and, by means of controlled cache simulations, tests whether the number of measurement runs used for pWCET estimation is high enough to capture all CCPs. Our results illustrate the effectiveness of our method to attain the desired confidence level in the pWCET estimates obtained.

We also show the main limitation of our method, which is low scalability and high computational cost, and assess the impact of analysing only a subset of addresses on reliability.

# Chapter 5

# Reducing Computational Cost to Attain Cache Representativeness for hRP

## 5.1 Introduction

In the previous chapter, we presented the Representativeness Validation by Simulation (ReVS) method, a solution for the cache representativeness problem for a limited scenario in which the program accesses a small number ($\leq 15$) of cache lines. ReVS considers all combinations of the most accessed cache line addresses with a cardinality bigger than cache associativity $W$, i.e. $\forall aC_i : |aC_i| > W$, and captures their impact in a cache simulator. However, the number of address combinations with a cardinality bigger than $W$ is huge: $\sum_{K=W+1}^{|@(\mathcal{Q}_i)|} \binom{|@(\mathcal{Q}_i)|}{K}$ for a sequence $\mathcal{Q}_i$, where in the general case $|@(\mathcal{Q}_i)|=U$, the number of unique cache line addresses in a program. In our reference benchmarks, $U$ is typically in the order of thousands (Table 3.4). Hence, evaluating in the cache simulator all potentially conflictive combinations of addresses is *not feasible in the general case* due to its exponential dependence on the number of addresses.

In this chapter, we propose a low-overhead solution to quickly identify Conflictive Cache Placements (CCPs) with hash Random Placement (hRP) and evaluate in the cache simulator only the most relevant subset of all possible address combinations, which allows analysing programs with an arbitrary number of addresses $U$. In the next chapter we address the same problem for Time Randomised Caches (TRC) deploying Random Modulo Placement (RM).

**Contributions.** In this chapter we present the Conflictive Cache Placements for hash Random Placement method (CCP-hRP), a general and computationally-tractable method that, from the program's sequence of accessed addresses, deter-

mines whether the number of runs performed by MBPTA, referred to as $R$, suffices to capture the CCPs with required probability. Else it derives a higher number of runs, referred to as $R'$, for which this can be asserted. Our contributions are the following:

1. CCP-hRP derives a list of address combinations that, when mapped to the same set, result in a high miss count. For each combination, CCP-hRP determines its probability and by means of a light-weight cache simulator, the number of misses that would be incurred when the addresses in each combination are mapped to the same set – while the rest of the addresses are randomly mapped. This results in an $< impact, probability >$ pair for each combination.

2. The user is advised to explore random cache placements with the cache simulator until the probabilistic Worst-Case Miss Count (pWCMC) curve derived with Extreme Value Theory (EVT) eventually upper-bounds the $< impact, probability >$ pairs determined by CCP-hRP. This occurs when enough address combinations ($R'$) singled out by CCP-hRP have been simulated and the number of observed miss counts becomes sufficient for EVT to converge to an exponential tail approximation (see Section 2.2.2). The user is then instructed to perform $R'$ runs on the actual system to assure a reliable application of MBPTA.

3. We compare CCP-hRP results against ReVS method presented in Chapter 4 for controlled scenarios where ReVS can be applied (for a subset of program addresses $U'$). The comparison shows that CCP-hRP covers all conflictive $aC_i$.

4. Results with EEBMC Autobench suite (Section 3.3.1) and a Railway Case Study (Section 3.3.3) running on an MBPTA-compliant Field-Programmable Gate Array (FPGA) show that CCP-hRP successfully identifies conflictive address combinations and determines the number of runs $R'$ required to bring the assurance level of the Worst-Case Execution Time (WCET) obtained with MBPTA to a desired threshold.

## 5.2 CCP-hRP Mechanism

For a sequence of addresses, CCP-hRP focuses on identifying combinations of $W$+1 or more addresses $aC_i$ that, when mapped to the same cache set, cause high execution times. The application of CCP-hRP comprises the following steps.

**Step 1. List creation**. Rather than considering all address combinations with a cardinality bigger than $W$ as ReVS does, CCP-hRP provides a list of potential conflictive $aC_i$ ranked according to their expected impact on execution time (the size of the list $T$ is defined as described later in this section). To that end, CCP-hRP builds a Guilt Table (GTAB) (Section 5.2.1) to quickly retrieve those combinations of addresses that, when mapped to the same set, can cause high miss counts.

**Step 2. Impact calculation**. Each combination in the list is evaluated with a cache simulator. Several Monte-Carlo simulations [39] are performed to derive the number of misses occurring when the addresses in the combination collide in the same set while the rest of the addresses are mapped randomly. The number of combinations in this list is fixed and, therefore, independent of the number of addresses in the program. ReVS, instead, simulates *all* combinations of addresses, which has a huge cost. The next steps are following the same flow as ReVS method. We briefly summarise them, while we refer the reader to Section 4.4 in Chapter 4 for more details.

**Step 3. Probability calculation**. CCP-hRP upper bounds the probability of occurrence of those $aC_i$ – and combinations of them. The probability of every $aC_i$ to occur is: $S \times (1/S)^{|aC_i|}$, where $|aC_i|$ is the number of addresses in $aC_i$. For the combined probability of several $aC_i$ we pessimistically use the addition of their individual probabilities. In reality, due to dependences among $aC_i$, their combined probability is smaller than that (see Section 4.4.3 in Chapter 4 for details).

*Step 2* and *Step 3* result in a pair $< impact, probability >$ for each combination, as previously shown in Figure 4.1 in Chapter 4. Black triangles and squares represent the miss counts obtained for all $aC_i$ – and their combinations – whose probability of occurrence is above $P_{rel}$. Meanwhile, their gray counterparts are those below $P_{rel}$, which are discarded by CCP-hRP since their probability is deemed as negligible.

**Step 4. pWCMC curve**. CCP-hRP uses MBPTA on the miss counts obtained from cache simulations in which all addresses are randomly mapped, as it would occur in reality, to obtain a pWCMC curve (see blue line in Figure 4.1). The number of simulations, $R$, is determined by MBPTA.

**Step 5. Assessment**. In Figure 4.1 triangles are those $aC_i$ (and their combinations) whose miss count is covered by the pWCMC, while the miss counts of the $aC_i$ marked with squares are not. Hence, by validating whether the pWCMC curve upper-bounds all conflictive mappings (i.e. $< impact, probability >$ pairs), we determine whether the number of runs $R$ used by MBPTA suffices. If this is not the case, more runs are performed until the validation step is passed with $R' \geq R$ runs. Whenever it is passed, the number of runs $R'$ is the minimum number of execution time measurements that MBPTA needs to use.

CCP-hRP builds on the correlation between miss counts and execution time that has been positively assessed for our target platform (Section 4.6.1). If such a correlation is weak, cache behaviour would have a low impact on execution time, which would have a higher dependence on other Sources of Jitter (SoJ). However, those other SoJ do not challenge MBPTA as probabilities of their events are higher than $P_{obs}$ [4].

### 5.2.1 The Guilt Table

CCP-hRP follows an iterative process in which, across iterations, an incremental number of addresses $K$ (starting from $K$=$W$+1) is considered to be mapped to the same set. This creates a cache conflict scenario exceeding cache space in one set. The process stops when $K$ is large enough so that the probability of occurrence of the event "$K$ addresses mapped to the same set for the most relevant combinations of $K$ addresses" is below a given cutoff probability $P_{cth}$[1]. In practice, we only need the most relevant combination for each value of $K$ since EVT already accounts for the probability of several of those events occurring simultaneously. Our results for controlled scenarios show that the worst combination is always among the few top-ranked ones by CCP-hRP. However, to increase the confidence of capturing the combination with the highest impact, our method returns the $T$=20 combinations predicted to cause the most conflicts for each value of $K$. Each combination is representative of several combinations that are predicted to cause a similar impact. Therefore the number of combinations considered by method for each value of $K$ is at least one order of magnitude higher than the number we empirically identified as necessary across all analysed benchmark suites.

CCP-hRP builds on the concept of *guilt*, which is intended to help identifying those $aC_i$ that, if mapped to the same set, result in high miss counts. For a given access $A_i$ with a non-null cache miss probability, *guilt* provides an approximation to the extent *each intermediate access* between $A_i$ and $A_{i-1}$ causes $A_i$ to miss in cache. Note that this concept, although related, differs from the probability of miss since we are not interested in how many misses each access experiences, but how much certain addresses can impact each other address if placed in the same cache set. For instance, given a direct-mapped (i.e. single way) cache and the sequence $\Omega_i = \{A_1 B_1 A_2\}$, if both addresses $A$ and $B$ are mapped to the same set, $A_2$ will miss in cache, and the cause of that is access $B_1$, so $B_1$ takes full *guilt* of $A$ eviction. Later in this section we present an efficient mechanism to approximate *guilt* for arbitrarily complex sequences.

**From probability of miss to *guilt*.** Several approaches [11] have been pro-

[1]Note that, while $P_{rel}$ stands for the threshold probability of relevant events at analysis (e.g. $10^{-9}$), $P_{cth}$ relates to the probability of events during operation (e.g. $10^{-15}$) [4].

posed to derive upper-bounds to the miss probability. However, in this work we are interested in the *actual* impact rather than on upper-bounds, and on *guilt* rather than on $P_{miss}$. Approaches exist to approximate $P_{miss}$ ($\tilde{P}_{miss}$) in the context of MBPTA [73]. These approaches are as shown in Equation 5.1, where $\sum \tilde{P}_{miss}(X_i)$ corresponds to the accumulated miss probability of the intermediate accesses.

$$\tilde{P}_{miss} = 1 - \left(\frac{W-1}{W}\right)^{\sum \tilde{P}_{miss}(X_i)} \tag{5.1}$$

While this approach provides good $\tilde{P}_{miss}$ approximations [93], it does not help to identify how much each intermediate access contributes to causing the miss.

CCP-hRP sorts address combinations based on their impact, which requires having means to estimate the relative impact that each address and group of addresses have on each other address (*guilt*) in terms of cache misses. To cover this gap we propose the $P_{guilty}$ estimator (see Equation 5.2) that targets providing a precise relative value for *guilt* as needed by CCP-hRP, rather than approximating $P_{miss}$.

$$P_{guilty} = 1 - \left(\frac{W-1}{W}\right)^{m_1} \qquad m_1 = \begin{cases} 0, & \text{if } q < W \\ q, & \text{if } W \leq q < K \\ K-1, & \text{otherwise} \end{cases} \tag{5.2}$$

When the number of intermediate addresses between $A_i$ and $A_{i-1}$ (let us call this $q$) is smaller than the number of cache ways $W$, they all would fit in a cache way, so misses may only be produced due to random replacement, whose impact is already captured with the default number of runs of MBPTA [4]. Hence, we assume that $A_i$ results in a hit, so the *guilt* of intermediate accesses is 0. Hence, we ignore $A_i$ and look for the next occurrence of $A$ until $q \geq W$ or we reach the end of the sequence. The rationale behind this is that hits do not change cache state in TRC, thus they can be ignored. On the other hand, ignoring intermediate accesses due to having extra hits in between $A_i$ and $A_{i-1}$ would be misleading. For instance, let us consider $W = 2$ and $\mathcal{Q}_1 = \{A_1B_1A_2C_1A_3\}$. We cannot assume that $A_3$ will always hit in $\mathcal{Q}_1$ since sooner or later $A$ will be evicted. Thus, $A_2$ is ignored and $A_3$ considers the *guilt* of $B_1$ and $C_1$. It can also be observed that we enforce $m_1$ to be smaller than $K$, the reason behind this is explained next.

**Guilt estimation**. When for an access $A_i$ $P_{guilty} \neq 0$, its value is 'distributed' among the intermediate accesses between $A_i$ and $A_{i-1}$. Each access is assigned a *guilt* value w.r.t. address $A$ computed as shown in Equation 5.3. For instance given a cache with $W = 2$ ways, the sequence $\mathcal{Q}_1 = (A_1B_1C_1D_1A_2)$ and $K = 3$, we obtain that $q = 3$ and $P_{guilty}(A_2) = 1-(1/2)^2 = 0.75$ according to Equation 5.2. In this scenario we assign a *guilt* of 0.375 to each of the $q = 3$ intermediate accesses. Note that the addition of *guilt* assigned to intermediate accesses is bigger than

$P_{guilty}$. The idea is that for $K = 3$, CCP-hRP constructs 3-address combinations that in this case can be any of $ABC$, $ABD$, $ACD$, $BCD$. In all those containing $A$, we want to assign one half of the *guilt* to each of the two intermediate accesses. That is, for $ABC$ one half of the *guilt* is assigned to $B$ and another half to $C$. At any moment only $K-1$ accesses will be simultaneously considered by CCP-hRP, so the *guilt* of a given access is not decreased because of having other intermediate accesses (more than $K$). As the value of $K$ increases – as part of CCP-hRP iterative process – those other intermediate accesses will be considered simultaneously.

$$guilt = \begin{cases} \frac{P_{guilty}}{m_1}, & \text{if } m_1 > 0 \\ 0, & \text{otherwise} \end{cases} \tag{5.3}$$

Based on the concept of *guilt*, which applies at access level, we build the Guilt Table. The GTAB comprises as many rows and columns as different (cache line) addresses are accessed in the program. Cell $GTAB[A].guilt[B]$ captures the *guilt* of $B$ on $A$, that is, a measure of to what extent misses of every access $A_i$ are caused by any access to $B_j$. It is computed as the addition of individual guilts assigned to any $B_j$ for potentially evicting $A$. The GTAB is built for every value of $K$. From the GTAB we infer information about the impact that each address has on the evictions of each other address. To that end we use the technique described in Section 5.2.2, which covers *Step 1* and *Step 2*.

The metric, obtained from the *guilt*, does not have a semantic meaning in the real world, yet it provides a way to rank address combinations so that if $aCi$ is ranked higher than $aCj$, the actual impact in miss count (and execution time) of $aCi$ is higher than that of $aCj$. This allows performing cache simulations for those highly ranked address combinations to measure their actual impact.

### 5.2.2 Smart Search of Address Combinations

**Exhaustive Search**. As a reference we use an algorithm that exhaustively searches the GTAB and later provide refinements to limit computational costs. For every value of $K$ we build all potential combinations of $K$ addresses out of $U$, so performing an Exhaustive Search. For each combination, we query the GTAB to obtain the expected impact if those addresses are mapped to the same set. The impact is obtained as follows: (1) for each address $i$ in the combination $aCi$ we compute a value $M_i$ obtained as the minimum impact that any of the $W$ most conflictive addresses in the combination may have on it. Hence, we take the minimum $M_i$ out of the highest $W$ values in the GTAB ($GTAB[i].guilt[X]$) where $X$ is any other address in the combination). Note that we care only about those $W$ addresses that can create the highest impact on the address of that row in the GTAB, since $W+1$ addresses suffice to exceed the cache set space. Then, we

select the minimum value out of those to reflect that, if an address produces few evictions, the others will not produce more evictions than that one because other accesses will become hits. (2) Finally, we obtain the impact as the harmonic mean of all $M_i$ values to, again, reflect that the number of evictions is limited by the address producing the lowest number of evictions. The harmonic mean has proved to be a better estimator than the average mean as it promotes the address combinations in which all addresses are conflictive among them. We exclude pairs for the same address (e.g. $GTAB[A].guilt[A]$) since an address cannot create evictions on itself. If one or some of the addresses have little impact on the other addresses, then its $M_i$ value is much lower and so the final impact, thus allowing to discard this combination. For instance, in the combination $aCi = \{A, B, C, D, E, F\}$, if $F$ has almost zero impact on the other addresses, this combination will be discarded for $K$=6. If the other 5 addresses have a high impact among them, they will be conveniently considered for $K$=5. Whenever all combinations are considered in the GTAB (without performing any cache simulation), we create a list of top-ranked combinations (*Step 1*) for which cache simulations are performed to measure miss counts (*Step 2*).

**Smart Search**. Since the computational cost of considering this Exhaustive Search in the GTAB is prohibitive, we propose a smart search algorithm that comprises the following steps. Steps 1-3 are done for each address (each row in the GTAB) unless the address is discarded as irrelevant.

*First*, we discard the rows in the GTAB whose $P_{guilty}$ is below 1% of the highest $P_{guilty}$ in the table since their combinations with relevant addresses ($P_{guilty}$ above the 1% threshold) will already be accounted by those other addresses, and their impact on irrelevant addresses is deemed irrelevant as well. Then, we create address *buckets* in each row of the GTAB with all the addresses with the same *guilt* value with respect to the address of that row. Empirically, we observed that EEMBC and the railway case study produce a low number of buckets. Otherwise, some difference is tolerated among addresses in the same bucket to reduce their count.

*Second*, the relevant buckets for a certain address are only those whose relative impact with respect to the total *guilt* in the row is significant for the address of that row. Such significance threshold $S_{th}$ (1% in our case) is used to explore combinations with meaningful impact. The remaining addresses (their *guilt* is below $S_{th}$) are simply regarded as irrelevant.

*Third*, we generate the combinations of $K$ elements for each row by making all possible combinations with the address corresponding to that row and $K-1$ elements from different buckets. For instance, assuming $K$= 4 and 2 buckets ($b1$ and $b2$), we make all combinations of 4 addresses using the one of the row and three addresses from the buckets: 3 from $b1$, 2 from $b1$ and 1 from $b2$, 1 from $b1$ and 2 from $b2$, and 3 from $b2$. We always choose those addresses with

the highest $P_{guilty}$ in each bucket. We take into account the size of the bucket by computing how many combinations are expected to have the same impact as the representative ones. For instance, if $b1$ and $b2$ contain 4 and 5 addresses respectively, when picking 2 addresses from $b1$ and 1 from $b2$, we determine that there are 30 different combinations meeting those constraints. This is used to set the probability of the pair $< impact, probability >$ if these combinations have a sufficiently high impact to be simulated.

*Fourth*, when all addresses have been analysed and the list with $T$= 20 combinations[1] for a particular value of $K$ is obtained (*Step 1*), we perform cache simulations to determine their miss counts (*Step 2*). In the case of addresses in a bucket, we simulate only those with the highest $P_{guilty}$ and assume the same impact for other combinations that could be generated with other addresses in the bucket. While this may lead to a little pessimism in terms of the impact of those addresses, such pessimism is very limited given that addresses belong to the same bucket. This may result in pairs $< impact, probability >$ further challenging the reliability of the pWCMC curve, thus potentially rejecting some very tight (yet reliable) pWCMC estimates.

## 5.3 Evaluation

We model a pipelined in-order processor with 4KB 2-way-associative 32B-line separated Instruction Cache Level 1 (IL1) and Data Cache Level 1 (DL1). Both caches deploy random placement and replacement policies [73], with DL1 implementing write-back (IL1 is read-only). DL1/IL1 access latency is 1 cycle for hits with 3 extra cycles for misses. The latter is added to the main memory latency (16 cycles). This corresponds to default configuration in Section 3.1.1.

We evaluate CCP-hRP on the EEMBC automotive benchmarks (Section 3.3.1). In particular we use these benchmarks: `a2time`, `aifftr`, `aifirf`, `aiifft`, `basefp`, `bitmnp`, `cacheb`, `idctrn` and `iirflt`. We consider all addresses accessed by each benchmark. Additionally, we analyse the same benchmarks in a *controlled scenario* in which we focus on a subset of the most accessed cache line addresses to allow for comparison against ReVS, which hardly scales for high values of $U$.

**CCP-hRP vs ReVS**. For this comparison, we focus only on the $U'$=15 most accessed addresses for which ReVS is capable of exploring all address combinations. While in this scenario we cover on average 58% of the accesses across all benchmarks – thus leaving some degree of uncertainty due to the remaining 42%

[1]One combination may be the representative of many others if addresses belong to buckets. Hence, simulating 20 combinations provides information of, at least, 20 actual address combinations, but generally many more than 20.

Table 5.1: Runs for CCP-hRP and ReVS for $P_{rel} = 10^{-9}$ and $U$'=15.

| | $R'_{IL1}$ | | $R'_{DL1}$ | | $R'$ | |
|---|---|---|---|---|---|---|
| | **ReVS** | **CCP-hRP** | **ReVS** | **CCP-hRP** | **ReVS** | **CCP-hRP** |
| a2time | 58,360 | 58,360 | 540 | 540 | 58,360 | 58,360 |
| aifftr | 6,840 | 6,840 | 5,500 | 5,500 | 6,840 | 6,840 |
| aifirf | 21,390 | 21,390 | 11,530 | 11,530 | 21,390 | 21,390 |
| aiifft | 8,920 | 8,920 | 8,770 | 8,770 | 8,920 | 8,920 |
| basefp | 82,080 | 82,080 | 20,010 | 20,010 | 82,080 | 82,080 |
| bitmnp | 4,640 | 4,640 | 3,510 | 3,510 | 4,640 | 4,640 |
| cacheb | 18,610 | 18,610 | 7,950 | 7,950 | 18,610 | 18,610 |
| idctrn | 65,770 | 65,770 | 47,700 | 47,700 | 65,770 | 65,770 |
| iirflt | 18,310 | 18,310 | 49,760 | 49,760 | 49,760 | 49,760 |

accesses that are neglected by ReVS it allows comparing CCP-hRP against ReVS, with the latter guaranteeing exact results.

Table 5.1 shows the number of runs that each of the methods regards as the minimum to use for a reliable MBPTA application. We show results for both IL1 and DL1. As shown, both approaches provide *exactly* the same number of runs ($R'$) for these limited address traces. In particular, CCP-hRP identifies the same address combinations most of the times or, alternatively, address combinations with roughly the same impact as those regarded by ReVS as the most conflictive ones for each value of $K$. The exceptions are the cases in which ReVS identifies for high values of $K$ combinations which, in fact, are the addition of two or more independent combinations. For instance, ReVS identifies combinations for $K$= 6 that, in reality, correspond to two combinations of $K$= 3 occurring at the same time. As explained before, EVT needs to observe high-impact events, but not their combination. Thus, this difference has no influence on $R'$.

Execution time cost. For $U'$= 15, ReVS requires on average 27 hours per benchmark with 1,000 cache simulations per address combination on a cluster running 100 jobs in parallel. CCP-hRP is 148 times faster requiring 2 seconds on average per program on a laptop computer to derive the address combinations and their cost, and around 11 minutes per benchmark to run cache simulations for the limited address combinations considered on the same cluster. For full benchmarks, i.e. unrestricted $U$, ReVS could not be applied, while CCP-hRP required 1 minute per program to generate the pairs $< impact, probability >$ and around 38 minutes per program to perform cache simulations in our cluster.

**CCP-hRP evaluation on full benchmarks**. In Table 5.2 we report the number of runs required by CCP-hRP to guarantee that relevant events can only be missed with a probability below a *parametrizable residual threshold*, e.g. $10^{-9}$.

Table 5.2: Results for complete EEMBC benchmarks.

| | CCP-hRP | | | | MBPTA | |
|---|---|---|---|---|---|---|
| | $R'_{IL1}$ | $R'_{DL1}$ | $R'$ | likelihood($R'$) | $R$ | likelihood($R$) |
| a2time | 67,150 | 300 | 67,150 | $10^{-9}$ | 300 | 0.911 |
| aifftr | 300 | 4,760 | 4,760 | $10^{-9}$ | 300 | 0.271 |
| aifirf | 20,080 | 8,090 | 20,080 | $10^{-9}$ | 14,260 | $10^{-7}$ |
| aiifft | 300 | 10,630 | 10,630 | $10^{-9}$ | 300 | 0.557 |
| basefp | 78,220 | 300 | 78,220 | $10^{-9}$ | 1,250 | 0.718 |
| bitmnp | 330 | 1,800 | 1,800 | $10^{-9}$ | 300 | 0.032 |
| cacheb | 19,840 | 1,500 | 19,840 | $10^{-9}$ | 9,360 | $10^{-5}$ |
| idctrn | 67,460 | 43,040 | 67,460 | $10^{-9}$ | 300 | 0.912 |
| iirflt | 29,920 | 2,430 | 29,920 | $10^{-9}$ | 300 | 0.812 |

We also show the runs requested by MBPTA together with the probability of missing those events with the default number of runs required. MBPTA takes as input the number of execution times belonging to the tail of the distribution that need to be observed in measurements, in our case 50 values [7] (see Step 3 in the Section 3.2). Then, starting from 300 runs, MBPTA inspects whether enough tail values are observed. If this is not the case, it asks for more runs until this condition is satisfied and EVT converges.

As shown, $R' \geq R$: in many cases we observe that the likelihood of missing critical address combinations in the default runs ($R$) determined by MBPTA only is high. This does not mean that probabilistic Worst-Case Execution Time (pWCET) estimates are necessarily wrong but indicates that there is a non-negligible risk of not observing some high-impact timing events in the analysis runs if CCP-hRP is not used. On average CCP-hRP requires around 11 times more runs compared to the default MBPTA.

When comparing the number of runs of CCP-hRP with full address traces with respect to only $U'$=15 addresses, we observe in most of the cases a limited variation in $R'$. However, in some cases $R'$ decreases noticeably (e.g. $R'_{IL1}$ for aifftr) because there are many combinations with a similar impact that cannot be observed with only 15 addresses. This makes the probability of observing one of those combinations much higher and thus, fewer runs are needed to observe one of them. In any case, differently to ReVS, which is limited to 15 addresses, CCP-hRP can deal with arbitrary access patterns without any explicit limit. Thus, CCP-hRP removes the uncertainty brought by ReVS due to non-analysed addresses.

Table 5.3: Runs needed by CCP-hRP and MBPTA to achieve a confidence of $10^{-9}$.

| | $IL1$ | | $DL1$ | |
|---|---|---|---|---|
| | $R$ | $R'$ | $R$ | $R'$ |
| TEST0 | 300(Y) | 300(Y) | 370(N) | 1,300(Y) |
| TEST1 | 300(N) | 600(Y) | 3,800(Y) | 3,800(Y) |
| TEST2 | 300(N) | 600(Y) | 300(N) | 1,000(Y) |
| TEST3 | 300(N) | 1,600(Y) | 300(N) | 850(Y) |
| TEST4 | 300(N) | 1,200(Y) | 750(N) | 1,100(Y) |
| TEST5 | 300(N) | 2,100(Y) | 480(N) | 900(Y) |
| TEST6 | 300(N) | 500(Y) | 890(Y) | 890(Y) |
| TEST7 | 300(N) | 500(Y) | 300 (N) | 4,400(Y) |
| TEST8 | 300(N) | 700(Y) | 300 (N) | 2,300(Y) |
| TEST9 | 300(N) | 4,800(Y) | 1,740(Y) | 1,740(Y) |

## 5.4 Railway Case Study

We evaluate our method for the railway case study presented in Section 3.3.3. Table 5.3 reports the results we obtained, in terms of the number of runs that MBPTA and CCP-hRP require in the miss domain. For each test, we show whether (Y) or not (N) MBPTA's default number of runs ($R$) and that reported by CCP-hRP ($R'$) suffice to upper-bound the pairs $< impact, probability >$. As it can be seen, the default application of MBPTA failed to upper-bound some address combinations for data and instructions for many input sets. Furthermore, in those cases where $R < R'$, confidence in having enough runs for a reliable application of MBPTA is not sufficiently high.

This is illustrated in Figure 5.1 for TEST7 and the DL1 where CCP-hRP $< impact, probability >$ pairs (points in the plot) are not upper-bounded by the pWCMC curve (lower straight line in the plot) when using $R$=300, the number or runs required by MBPTA. Instead, if we use $R'$=4,400, as determined by CCP-hRP, the pWCMC curve properly upper-bounds those pairs.

For this industrial application, CCP-hRP required, on average, $1,828$ runs per input set, which is affordable in a usual test campaign. CCP-hRP took 1.3 minutes to derive the conflictive combinations and 0.35 minutes per test for cache simulations.

## 5.5 Summary

In this chapter, we propose CCP-hRP, a low-overhead mechanism that determines whether the number of runs is enough to cover CCPs with hRP to a given quantifi-

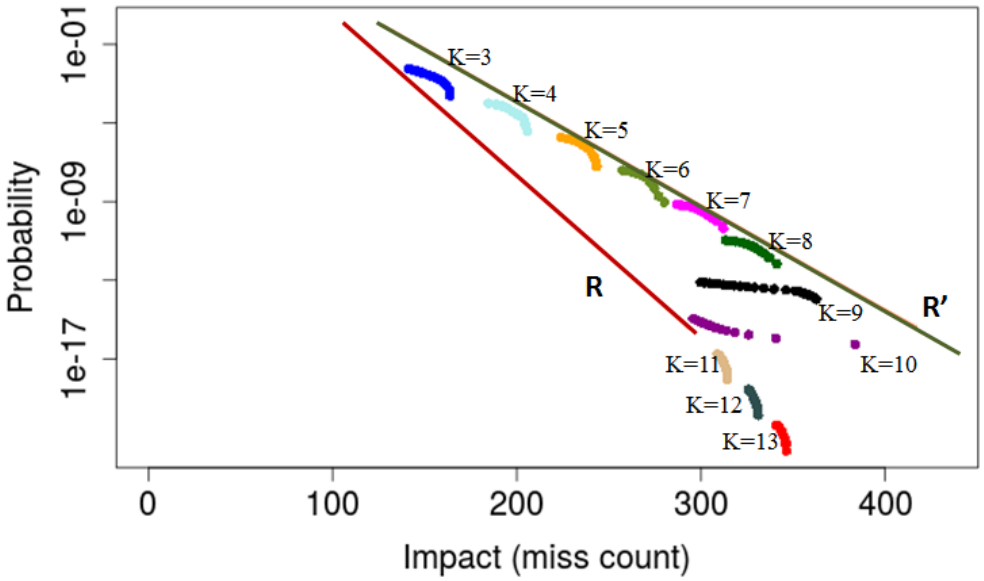



Figure 5.1: pWCMC for *TEST7* (DL1) by applying MBPTA ($R$) and CCP-hRP+MBPTA ($R'$).

able threshold. If this is not the case, CCP-hRP requests an increased number of runs to the user until the threshold is reached. Results with EEMBC Automotive benchmarks and a real railway case study show that CCP-hRP successfully identifies conflictive address combinations and increases the number of runs accordingly so that reliable WCET estimates can be obtained for programs with arbitrary access patterns. Compared to the previous solution ReVS, CCP-hRP is not limited to a (low) number of different addresses, so it conveniently captures the impact of all program addresses.

# Chapter 6

# Computationally Tractable Method to Attain Cache Representativeness for RM

## 6.1 Introduction

In Chapter 4 we have presented a theoretical evaluation framework originally proposed for hash Random Placement (hRP). This framework can be adapted for all Time Randomised Caches (TRC) which build on Monte-Carlo simulations [39] to explore the impact of a relatively large set of random cache placements. From these simulations, we evaluate the miss impact in a cache simulator of *all* potential Conflictive Cache Placements (CCPs). Therefore, Representativeness Validation by Simulation (ReVS) can be applied to analyse caches deploying Random Modulo Placement (RM), with the only difference of modifying accordingly the cache simulator. However, as in the case of caches with hRP, ReVS' execution time requirements are unaffordable in the general case as the number of cache placements is a function of the number of different (unique) addresses in the program: with address counts as small as 20 the number of simulations to perform reaches the order of billions, simply preventing ReVS as a valid general solution. Instead, ReVS can be used in controlled experiments, with a reduced number of unique addresses, to assess the accuracy of computationally-tractable techniques to capture CCPs.

In Chapter 5 we have presented Conflictive Cache Placements for hash Random Placement method (CCP-hRP), a technique to derive the minimum number of runs required for hRP with affordable computational cost, but such a solution does not work on RM. Yet, MBPTA-compliant hardware implements both of these random cache placement techniques, as they provide different and valuable tradeoffs. RM provides competitive results in comparison to standard (modulo

placement) caches [65]. On the other hand, hRP provides lower performance but imposes fewer constraints in hardware and software designs.

**Contributions.** In this chapter we present Conflictive Cache Placements for Random Modulo Placement method (CCP-RM), a timing efficient solution to the cache-related representativeness problem for caches deploying RM. In particular, the main contributions of this work are as follows:

1. We analyse the behaviour of RM and hRP to shed some light on how CCPs emerge in a different manner under those designs. We show that the particular set of $W+1$ (or more) addresses that, when mapped to the same set, cause a CCP are different under RM and hRP. Further, both the miss counts and the number of CCPs decrease under RM, which in turn reduces the number of runs required. This motivates the necessity of different design-specific techniques to intercept CCPs.

2. We propose the CCP-RM mechanism that identifies the CCPs for a given sequence of addresses, along with their probability and impact on execution time, for RM. For a given configurable coverage probability threshold $P_{cth}$, CCP-RM determines whether the impact of CCPs is captured in the default $R$ runs performed by MBPTA. Otherwise, more runs need to be carried out until the probability of not observing one of the random CCPs is below $P_{cth}$. While CCP-RM shares some steps with the previously proposed CCP-hRP method, we present the full methodology here for completeness.

3. We evaluate CCP-RM on a cycle-accurate timing simulator where we provide evidence of its benefits on reference EEMBC automotive benchmarks. The simulator experimental setup allows building controlled scenarios in which we can exhaustively derive *all* cache placements and show how CCP-RM captures the actual set of CCPs. This information can be used as evidence for certification.

4. We assess CCP-RM on a real setup with a case study, representative of the railway domain, running on a real implementation of RM on an Field-Programmable Gate Array (FPGA) board. Results show that CCP-RM identifies CCPs with a limited burden on the user side (in terms of effort and time) and derives the number of observations $R'$ that need to be collected to ensure that the probability of not capturing a relevant CCP is below acceptable levels.

## 6.2 Understanding Conflictive Cache Placementes Under RM and hRP

RM and hRP cause CCPs to be triggered differently, which motivates having two different CCPs detection techniques (for details of both mapping functions see Section 2.2.1). In order to better understand the differences between RM and hRP let us assume the address sequence $\mathcal{Q}_0$.

$$\mathcal{Q}_0 = \{A_1^0 B_1^0 C_1^1 D_1^1 C_2^1 D_2^1 C_3^1 D_3^1 A_2^0 B_2^0 A_3^0 B_3^0 C_4^1 D_4^1 C_5^1 D_5^1 C_6^1 D_6^1 A_4^0 B_4^0\}$$

Each element is a memory address with the superscript denoting the memory page (cache segment) it belongs to, and the subscript the number of times that address has been referenced. Further, let us assume a 2-set direct-mapped cache. With hRP, addresses have an equal probability to be mapped to a set. This also holds for RM except for addresses belonging to the same memory page that are prevented from colliding in the same set. As a consequence, the set of possible cache placements generated by RM is always a subset of the possible cache placements by hRP. For $\mathcal{Q}_0$, Table 6.1 lists all possible placements. As it can be seen, from all hRP placements, only two can arise with RM.

Table 6.1: Conflictive cache placements for $\mathcal{Q}_0$ under RM and hRP

| Placement | hRP | | RM | |
|---|---|---|---|---|
| {set0}{set1} or {set1}{set0} | Misses | CCPs? | Misses | CCPs? |
| {ABCD} {-} | 20 | yes | not possible | |
| {AB} {CD} | 20 | yes | not possible | |
| {BCD} {A} | 17 | yes | not possible | |
| {ACD} {B} | 17 | yes | not possible | |
| {ABC} {D} | 15 | yes | not possible | |
| {ABD} {C} | 15 | yes | not possible | |
| **{AC} {BD}** | 10 | yes | 10 | yes |
| **{AD} {BC}** | 10 | yes | 10 | yes |

hRP and RM produce different CCPs. hRP does not prevent any addresses to coexist in a set, resulting in potential conflicts among addresses in the same memory page. RM, instead, avoids such behaviour by design. Moreover, the CCPs with RM, in general, produce lower miss counts compared to those of hRP, as illustrated in Figure 6.1, reporting miss count and frequency for the `bitmnp` EEMBC benchmark executed in our reference setup (later presented in Section 6.4 and corresponding to default configuration from Section 3.1.1). RM incurs similar miss counts as hRP when addresses of the sequence causing conflict misses belong to different memory pages.

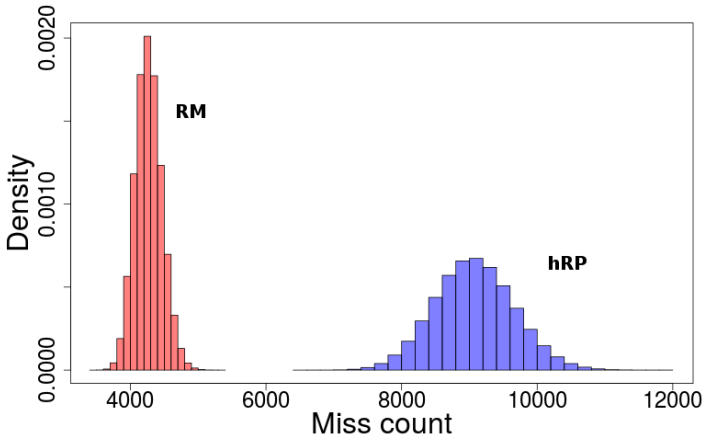



Figure 6.1: `bitmnp` behaviour in first level instruction hRP and RM caches.

## 6.3 The CCP-RM Mechanism

CCP-RM derives the minimum number of runs needed to ensure that all relevant cache events (i.e. CCP) have been observed with sufficiently high probability for an RM cache. For a given cache setup, CCP-RM analyses the sequence of memory accesses of the program (considering instruction and data accesses separately). From these inputs, CCP-RM:

1. First creates a list of those address combinations that, if placed in the same set, can result in high cache miss counts (Section 6.3.1).

2. Derives for each combination its impact in miss count and probability of occurrence (Section 6.3.2).

3. Assesses whether for a probability threshold $1 - P_{cth}$ the number of runs typically required by MBPTA ($R$) already captures these combinations and, otherwise, iteratively requests further runs ($R'$) until the probability of missing those combinations falls below a maximum allowed threshold $P_{cth}$ (Section 6.3.3).

CCP-RM is based on the heuristic to improve its computational efficiency.

### 6.3.1 Deriving Relevant Address Combinations

A necessary characteristic of CCP address combinations is that their cardinality (i.e. the number of addresses) exceeds the number of cache ways $W$ (e.g. for a 2-way cache, all address combinations of 3 or more addresses are potentially

conflictive). CCP must also have a probability of occurrence sufficiently high to be considered relevant by the corresponding safety standard.

For each address count $K{>}W$, CCP-RM derives a list of addresses (combinations) expected to generate many misses if they are randomly mapped to the same set. CCP-RM focuses on those combinations whose probability of occurrence is above a safety-standard defined probability (e.g. $10^{-9}$). It stops exploring $K$ values when the probability that $K$ addresses are mapped to the same set falls below that threshold. Such probability is analytically derived as $(1/S)^{K-1}$, (the multiplication of the number of available sets $S$ and individual probabilities to be mapped to the selected set $1/S$, for each of the $K$ addresses). CCP-RM sets a maximum size ($T$) for the list of most relevant combinations to be considered for a given value of $K$. As detailed later, those $T$ combinations, once evaluated, provide information of at least $T$ address combinations but, due to the way we select them, often represent many more than $T$ combinations.

Next, we describe how to produce the lists (for each value of $K$). We accompany the explanation with examples in which we assume a 2-way set-associative cache.

### Guilt Estimation

We present an estimator (loosely based on the probabilities of address misses) called *guilt* that, for each address $A^X$ in a program, classifies the other addresses with respect to how many evictions of $A^X$ they will cause if randomly mapped to the same set. The *guilt* attribute is later exploited to calculate the predicted impact of a group of addresses, which in turn is used to rank the address combinations based on the increase of the overall number of cache misses caused by these addresses placed together in the set. To estimate guilt, CCP-RM analyses the access sequence between consecutive accesses to the same address. The concept of *guilt* has been first introduced in Section 5.2.1. While its high-level meaning is the same as for CCP-hRP, the way how it is computed is different.

Notably, under RM only addresses belonging to different segments can evict each other. Hence, CCP-RM ignores addresses belonging to the same segment. For example, in the sequence $\mathcal{Q}_1 = (A_1^{100}, B_1^{100}, C_1^{100}, A_2^{100})$, $A_2^{100}$ is necessarily a hit, since all intermediate accesses, i.e. those between $A_1^{100}$ and $A_2^{100}$, belong to the same segment.

Also with RM, addresses mapped to a set necessarily belong to different segments. Therefore, an address can only conflict with *exactly one* address from every other segment, but all addresses from that segment are equally probable to conflict. For example, in $\mathcal{Q}_2 = (A_1^{100}, B_1^{102}, C_1^{102}, A_2^{100})$, addresses $B^{102}$ and $C^{102}$ both can be mapped with the same probability $1/S$ to the same set as $A^{100}$, but only one of them will be actually mapped (i.e. either $B^{102}$ or $C^{102}$). Hence, the number of addresses accessed in-between two accesses to the considered address ($A^{100}$) that

can be placed in the same set is at most the number of different cache segments accessed in between, denoted by $s$.

For each memory access $A_j^X$, we define $P_{guilty}$ as the likelihood of $A^X$ being evicted due to the conflicts with other addresses, see Equation 6.1. Since in RM caches, conflicts can only happen between addresses mapped to different cache segments, $s$ in Equation 6.1 represents the number of distinct cache segments accessed in between two accesses ($A_{j-1}^X$ and $A_j^X$) to the same address $A^X$.

$$P_{guilty}(A_j^X) = 1 - \left(\frac{W-1}{W}\right)^{m_1} \quad m_1 = \begin{cases} 0, & \text{if } s < W \\ s, & \text{if } W \leq s < K \\ K-1, & \text{otherwise} \end{cases} \tag{6.1}$$

The fraction $\frac{W-1}{W}$ represents the probability of a cache line to survive an eviction, whereas $m_1$ relates to the number of evictions occurring. When $s$ is smaller than $W$, the intermediate accesses would fit in a cache set, so misses may only be produced due to random replacement, whose impact is already captured with the default number of runs of MBPTA [4]. Hence, we assume that $A_j^X$ hits, so the guilt of intermediate accesses is 0. For values of $s$ larger or equal to $W$, we assume $s$ evictions, but bounding that number up to $K-1$, since we inspect different address group cardinalities ($K$) iteratively and for a given $K$ value at most $K-1$ addresses can conflict with the address in focus.

Equation 6.1 captures the case when the number of distinct segments accessed between $A_{j-1}^X$ and $A_j^X$ is higher or equal to $W$ (i.e., $s \geq W$). Conflicts can also occur under other address interleavings. For example, in the sequence $Q_3 = (A_1^{100}, B_1^{101}, A_2^{100}, C_1^{102}, A_3^{100})$, $A_3^{100}$ is likely to suffer misses, even though the estimator would predict hits because $s < W$. In this case, we may have misses because hits do not alter cache state so that they can be stripped out of the access sequence conceptually (e.g. $A_2^{100}$), thus making other accesses (e.g. $A_3^{100}$) likely miss due to evictions caused by $B_1^{101}$ and $C_1^{102}$. To account for this, when deriving $P_{guilty}$ of an access, CCP-RM searches for the previous access to the same address with derived $P_{guilty}$ value higher than 0. The access is predicted to hit only if $s$ is strictly lower than $W$ also in the subsequence between those two accesses. Otherwise, the access is considered a miss and the computed $P_{guilty}$ value is distributed over the subsequence. In the example, $A_1^{100}$ is a miss (the first access to an address is always a miss) and $A_2^{100}$ is a hit ($s = 1 < W$). When assessing $A_3^{100}$, CCP-RM searches for previous accesses until reaching $A_1^{100}$ ($A_2^{100}$ is ignored as it is predicted not to change cache state) and evaluate that $A_3^{100}$ may miss since $s = 2 > W$ due to accesses $B_1^{101}$ and $C_1^{102}$.

### The Guilt Table

Each eviction of an address in a program under a conflictive placement, which occurs with estimated probability $P_{guilty}$, is caused by a set of addresses. The share of responsibility of each of these addresses on causing the eviction is quantified with their *guilt* value. While *guilt* describes the pair-wise relation between addresses, the predicted impact of the group of addresses will depend on the connection between each pair of addresses in a group. Computed *guilt* values are stored in the Guilt Table (GTAB), which is later queried (either by inspecting exhaustively all possible address combinations or a subset of them as proposed in Section 6.3.1) to derive predicted impacts of address groups.

The GTAB is organised as a matrix. For each address $A^{seg(A)}$ in a program, the GTAB keeps track of:

- The overall likelihood of that address to miss due to conflicting with other addresses under CCP ($GTAB[A^{seg(A)}].\tilde{P}_{guilty}$). It is equal to the accumulated $P_{guilty}$ values of each access to address $A^{seg(A)}$.
- The overall guilt of each other address on potential evictions of address $A^{seg(A)}$. For instance, cell $GTAB[A^{seg(A)}].guilt[B^{seg(B)}]$ keeps *guilt* values of address $B^{seg(B)}$ with respect to misses of all accesses to address $A^{seg(A)}$ during the execution of the program.

To populate GTAB, CCP-RM iterates through the sequence of memory accesses and computes their $P_{guilty}$ value. If for a given access to address $A_i^{seg(A)}$ $P_{guilty} \neq 0$, then this value is added to $GTAB[A^{seg(A)}].\tilde{P}_{guilty}$ and distributed among the different segments in-between $A_i^{seg(A)}$ and the previous access to $A^{seg(A)}$, i.e. $A_{i-1}^{seg(A)}$, as shown in Equation 6.2, using $m_1$ value computed in Equation 6.1. Then, the *guilt* assigned to a segment is added on top of the $GTAB[A^{seg(A)}].guilt$ of each intermediate accessed address that belongs to that segment. This is done because each address in a given segment can be placed in the same set as the analysed address with identical probability. For example, let us consider the access $A_2^{100}$ in the sequence $\mathcal{Q}_4 = \{A_1^{100}, B_1^{102}, C_1^{100}, D_1^{103}, B_2^{102}, E_1^{102}, A_2^{100}\}$, with $W$=2 and the group cardinality $K$=3. The number of distinct cache segments accessed after $A_1^{100}$ is $s$=2 (segments 102 and 103). Those segments together with segment 100 exceed the cache associativity. We compute $P_{guilty}(A_2^{100}) = 1 - \left(\frac{1}{2}\right)^2 = 0.75$ and add it to $GTAB[A^{100}].\tilde{P}_{guilty}$. Next, we distribute 0.75 across $s$=2 segments, such that $P_{guilt-seg} = \frac{0.75}{2} = 0.375$. The *guilt* of all addresses belonging to segments 102 and 103 ($GTAB[A^{100}].guilt[B^{102}]$, $GTAB[A^{100}].guilt[D^{103}]$ and $GTAB[A^{100}].guilt[E^{102}]$) is then increased by 0.375.

$$P_{guilt-seg} = \begin{cases} \frac{P_{guilty}}{m_1}, & \text{if } m_1 > 0 \\ 0, & \text{otherwise} \end{cases} \tag{6.2}$$

Table 6.2: Relevant fields of GTAB to derive predicted impact of an address combination [$A^{100}$, $B^{102}$, $D^{103}$, $F^{104}$].

| | $P_{guilty}$ | guilt.B | guilt.D | guilt.F | guilt.A | guilt.xxx | $M_X$ |
|---|---|---|---|---|---|---|---|
| $B^{102}$ | 550.0 | 0.0 | 150.0 | 20.0 | 15.0 | 365.0 | 20.0 |
| $D^{103}$ | 400.0 | 145.0 | 0.0 | 30.0 | 10.0 | 215.0 | 30.0 |
| $F^{104}$ | 250.0 | 16.0 | 28.0 | 0.0 | 30.0 | 176.0 | 28.0 |
| $A^{100}$ | 235.0 | 15.0 | 5.0 | 30.0 | 0.0 | 185.0 | 15.0 |

A GTAB is derived for each cardinality $K$ and is later inspected to compute the predicted impact of address combinations. This is done in two steps:

First, for each address $A$ in the combination under analysis, we sort all the other addresses in the combination by their guilt on $A$, and take the value on the $W^{th}$ position and keep it as $M_A$. The reason is that address $A$ needs to conflict with at least $W$ addresses to exceed the cache set space and such scenario cannot occur more times than the number of conflicts between address $A$ and the least conflictive address with $A$ among $W$ of them. In the example in Table 6.2, we observe high guilt values between addresses $B^{102}$ and $D^{103}$, but not among the rest of addresses. This happens when those two addresses are interleaved together with other addresses that do not interleave systematically with these ones. In the table, *guilt.xxx* stands for the *guilt* values for other addresses omitted in the example for clarity.

And second, the predicted impact of an address combination is computed by applying the harmonic mean of all $M_A$ values of addresses in a combination so to give lower rank to combinations with low $M_A$ values, which reflects that $A$ cannot have many conflicts if one of the other addresses cannot create many conflicts. Instead, CCP-RM seeks address groups in which conflicts occur due to the interaction among *all* of them. If a conflictive behaviour occurs because of the interaction of a subset of these addresses, such combination is already accounted for by CCP-RM for lower $K$ values. For instance, in Table 6.2 the predicted impact of the combination [$A^{100}$, $B^{102}$, $D^{103}$, $F^{104}$] equals the *harmonic-mean*[1] of 20.0, 30.0, 28.0 and 15.0, which is 21.54.

### Generation of Address Combinations

To derive a list of the conflictive combinations, CCP-RM builds a GTAB for each $K$ value. Next, it generates possible combinations of $K$ addresses and derives

[1]The harmonic mean is a good estimator due to its sensitivity to lower values, giving lower ranking to combinations in which at least one address is not conflicting with others. Average mean, however, promotes combinations with only a subset of addresses conflicting.

**Algorithm 1** Generation of Address Combinations with CCP-RM

**Input:** K ▷ number of addresses in a combination;
GTAB ▷ address Guilt Table derived for K;
S ▷ number of cache sets;
**Output:** List of <combination; probability> pairs;
List of predicted impacts for each element in pairs listPI;

sort GTAB row-wise by $\tilde{P}_{guilty}$;
N ← nrows(GTAB);
**for** addr ← 1:N **do**
  **if** GTAB[addr].$\tilde{P}_{guilty}$ < 0.01∗GTAB[1].$\tilde{P}_{guilty}$ **then**
    break;
  **end if**
  totalGuilty ← $\sum_{j=1}^{N}$ GTAB[addr].guilt[j];
  addrSpace ← [ ];
  **for** m ← (addr+1):N **do**
    **if** (m ∉ getSeg(addr) **and** GTAB[addr].guilt[m]≥ $S_{th}$∗totalGuilty) **then**
      addrSpace.add(m);
    **end if**
  **end for**
  <combination; probability> pairs ← [ ];
  Segs ← list distinct segments in addrSpace;
  combsSeg ← list all combinations of (K-1) segments from Segs;
  **for all** cs: cs ∈ combsSeg **do**
    listA ← [ ];
    cntA ← [ ];
    **for** seg ← cs[1]:cs[K-1] **do**
      listA[seg] ← [ ];
      cntA[seg] ← [ ];
      **for all** g: distinct guilt values in seg **do**
        listTmp ← [ ];
        **for all** a: a ∈ seg **do**
          **if** (GTAB[addr].guilt[a] == g) **then**
            listTmp.add(a);
          **end if**
        **end for**
        listA[seg].add(address with the highest $\tilde{P}_{guilty}$ in listTmp);
        cntA[seg].add(number of addresses in listTmp);
      **end for**
    **end for**
    **for all** combinations< $a_1$,$a_2$,..,$a_{k-1}$,addr>: $a_j$ ∈ listA[j], where j=1,..,K-1 **do**
      prob ← $S * (\frac{1}{S})^K * \prod_{j=1}^{K-1}$ cntA[getSeg($a_j$)][$a_j$];
      pairs.add(<< $a_1$, $a_2$,...,$a_{k-1}$, addr>; prob>);
      listPI.add(predicted impact of < $a_1$, $a_2$,...,$a_{k-1}$, addr>);
    **end for**
  **end for**
**end for**

their impact as previously described. Ideally, CCP-RM would inspect all possible combinations (discarding the ones in which addresses from the same segment repeat). However, the number of combinations grows exponentially with the number of addresses. Therefore, CCP-RM adopts an algorithm (see Algorithm 1) to optimise this search by generating the subset of all possible combinations of addresses expected to be the most conflictive ones (similar to the smart search in Chapter 5).

Rows in GTAB are sorted by their overall $P_{guilty}$ value. For each row, e.g. that for address $A^{seg(A)}$, the algorithm generates combinations of $K$ addresses, containing $A^{seg(A)}$, belonging to addresses (rows) not yet inspected, i.e. with lower $P_{guilty}$. The search stops when $P_{guilty}$ of a row address is below 1% of the highest $P_{guilty}$ in the table (lines 6-8), since potential combinations could only consist of low impact addresses (each below the 1% threshold).

In each iteration, the algorithm considers the potential conflictive addresses with $A^{seg(A)}$ (lines 9-15). It excludes all addresses belonging to the same segment and those whose *guilt* value on the row address with respect to the total guilt on that address is below the defined significance threshold $S_{th}$ (1% in our case). Next, the remaining addresses are grouped by the segment they belong to into *Segs*, to account for the fact that only one address from each group can belong to the same combination. Our search derives all the possible ways to select $K-1$ addresses from *Segs* groups (lines 17-18).

Then we need to explore conflicts against all the segments in each combination (lines 22-35). In order to explore all addresses of any given segment in *Segs*, we take into account their individual guilt on the address *addr* with which we are generating combinations, and only consider one address representative (with the highest overall $P_{guilty}$ value) for those that have the same $GTAB[addr].guilt$ value, since their impact will be identical. When computing the probability of those combinations (lines 36-40), we account for the number of combinations that could be produced with addresses in that group (due to several of them having identical guilt value) to multiply the probability of having just one address. Considering a combination touching 3 segments: having 2 addresses with identical *guilt* value in a first segment, 3 in a second one, and 3 in the other, will lead to 18 potential combinations with exactly one address from each segment. Such probability is specifically computed at line 37. Still, multiplying probabilities without considering that combinations may overlap (the latter would diminish the overall combined probability) leads to some pessimism. As shown later, this could only lead to requesting more runs than strictly required, thus not diminishing the confidence of the method. On the other side, addresses from the same *Segs* group, but with different *guilt* values, are considered individually.

**Illustrative example**. Next, we show the generation of combinations of size $K = 3$ in one iteration. We inspect the row containing address $A^{100}$ and the set

of potentially conflictive addresses [$E^{102}$,$C^{103}$,$B^{102}$,$D^{103}$] with their corresponding values of *guilt* on $A^{100}$: [15, 15, 15, 20] and $P_{guilty}$: [72.5, 62.5, 58.75, 30.375], as illustrated in Table 6.3.

Table 6.3: The relevant fields of GTAB to generate combinations of addresses containing address $A^{100}$.

| | | Seg 100 | Seg 102 | | Seg 103 | |
|---|---|---|---|---|---|---|
| | $P_{guilty}$ | guilt.A | guilt.B | guilt.E | guilt.C | guilt.D |
| $A^{100}$ | | 0.0 | 15.0 | 15.0 | 15.0 | 20.0 |
| $E^{102}$ | 72.500 | | | | | |
| $C^{103}$ | 62.500 | | | | | |
| $B^{102}$ | 58.750 | | | | | |
| $D^{103}$ | 30.375 | | | | | |

Addresses are grouped in *Segs* 102 and 103. The only way to make a combination of $K-1$ addresses is to select one address from each segment. *Segs* 102 contains two addresses with identical *guilt* value, thus it returns the address with higher $P_{guilty}$, which is $E^{102}$, and marks that two addresses share this behaviour. *Segs* 103 returns addresses $C^{103}$ and $D^{103}$ since they have different *guilt* value. If $p$ is the probability that three addresses are mapped to the same set, then the step will result in the next $< combination; probability >$ pairs: [$A^{100}$,$E^{102}$,$C^{103}$; $2p$] and [$A^{100}$,$E^{102}$,$D^{103}$; $2p$]. Here $2p$ is the probability of combining the 2 addresses in segment 102 with the specific address in segment 103.

Finally, the address combination search returns $T < combination; probability >$ pairs, the highest predicted impacts in $listPI$, where the list is derived as shown in lines 36-40 of Algorithm 1 and predicted impact computed as described in Section 6.3.1 (see example in Table 6.2). While these combinations are expected to produce the highest impact in terms of the number of misses when placed in the same set, their actual impact needs to be determined since *guilt* and $P_{guilty}$ are estimators that are used just to rank address combinations.

While we simplified Algorithm 1 for the sake of readability, its implementation can be further optimised, which we did in our experiments. We recommend the recursive implementation to generate combinations (line 18). Finding distinct segments of addresses (line 17) can be done while iterating through addresses (lines 11 and 12) by storing segments into a structure with non-repeated values (e.g. a set). $listA$ and $cntA$ (lines 22-34) can be created at line 13, where the user does not need to keep all the addresses with the same *guilt*, but only $K$ with the highest $P_{guilty}$ value. For efficient searching of relevant data, we recommend the use of map structures (instead of lists/arrays).

### 6.3.2 Impact and Probability Calculation

The algorithm from the previous step derives a list of $< combination; probability >$ pairs, describing representative address combinations and the probability of observing them or any other with the same predicted impact. The impact in terms of miss count of each representative address combination is evaluated with a cache simulator, similarly to the CCP-hRP. The addresses in the combination are mapped to the same set, while others are randomly mapped. Several Monte-Carlo simulations are performed and the impact of the given combination is determined as the average impact across the different Monte-Carlo simulations.

Next we map $< combination; probability >$ pairs into the $< impact, probability >$ domain by ordering the pairs per derived impact, and computing the combined probability and impact for the first pair, first two pairs, first three pairs, and so on, until we cover all pairs in the list, in the same way as it is done by CCP-hRP. Given a number of first $N$ address combinations, the combined miss count (having one of them) is their average miss count, and the combined probability is the union of their probabilities. Since their individual probabilities do not have to be necessarily disjoint, to determine the exact joint probability one would need to determine the overlaps between all groups of 2, 3,...,$N$ combinations. However, to avoid the inherent computational complexity of such activity, similar to the previous step, we upper-bound such probability as the addition of their probabilities, which is generally a tight upper-bound. The individual probabilities of all combinations of $K$ addresses are identical, so determining the joint probability becomes trivial.

### 6.3.3 Assessment Against the pWCMC Curve

Previous steps result in a pair $< impact, probability >$ for each CCP, i.e. combination and group of combinations deemed as conflictive. As next step, we generate a probabilistic Worst-Case Miss Count (pWCMC) by applying Extreme Value Theory (EVT) to the miss counts in a sample of $R$ randomly generated RM mappings. We check whether the pWCMC distribution upper-bounds all CCPs (i.e their miss count). If this is the case, the default number of measurements $R$ used by MBPTA suffices to derive trustworthy Worst-Case Execution Time (WCET) estimates. Otherwise, more runs are performed, until pWCMC upper-bounds all pairs. The obtained number of runs $R'$ is finally returned as the number of runs to be collected by the end user.

## 6.4 Evidence for Certification

The CCP-RM mechanism that we propose to identify and capture CCPs, is meant to enhance the reliability of MBPTA results, to meet the reliability requirements of the validation and verification of embedded critical systems. Not surprisingly, CCP-RM features a heuristic search over the CCP space: a reasoned heuristic-based empirical evidence is *de-facto* the only means to analyse overly-complex hardware and software systems, where providing exhaustive evidence is generally untenable.

This section aims at showing that CCP-RM can concretely improve the reliability of MBPTA. We do so by conducting a three-fold assessment.

1. We show that the implemented heuristic is accurate by comparing its outcome with that of ReVS (see Chapter 4), which provides *exact* results by exhaustively exploring the impact of all address combinations with cardinalities higher than cache associativity. Since the cost of ReVS is prohibitive for real-size programs (hence the need for a heuristic), we stick to a controlled scenario as explained next.

2. We show the effectiveness of CCP-RM in detecting otherwise ignored CCPs, and the impact this has on the overall number of runs CCP-RM requires to use ($R'$) as compared to the default number of runs used by MBPTA ($R$).

3. We show that the benefits of CCP-RM come with extremely affordable computational requirements.

It is worth noting that, in our case, the implication of using a heuristic is that the CCP impact computed by CCP-RM may slightly differ from that computed by the exact method. However, our algorithm does not focus on a single CCP but on a list thereof, and the list of CCPs considered by the heuristic is always including the topmost (highest-impact) CCPs. For the benchmarks evaluated in this section and case study (next section) we consider the topmost 20 CCPs (i.e., $T = 20$), which in practice results in considering already more combinations than strictly required, as confirmed by the results.

### 6.4.1 Experimental Setup

We use a simulation environment based on the cycle-accurate SoCLib framework (see Section 3.1). We model an architecture featuring a pipelined in-order processor with separated Instruction Cache Level 1 (IL1) and write-back Data Cache Level 1 (DL1) caches, both deploying random replacement, and RM policy. Access latencies for IL1 and DL1 are 1 cycle for hits and 3 additional cycles for misses,

which sums up to the main memory latency, for a total of 20 cycles (3+1 cycles for misses and 16 cycles for memory latency). In the first set of experiments, we use the EEMBC benchmark suite ( 3.3.1).

### 6.4.2 CCP-RM Accuracy

Due to the prohibitive cost of ReVS, in this section, we focus on a controlled scenario with a small number of addresses, which was obtained by creating synthetic benchmarks accessing the 15 most-frequent cache line addresses from each EEMBC Automotive benchmark. We use the small cache configuration: 512B 32B/line 2-ways IL1/DL1 (Section 3.1.1), in order to observe conflicts with small address footprints.

Table 6.4: $R'$ for CCP-RM and ReVS in controlled scenario

| | $R'_{IL1}$ | | $R'_{DL1}$ | | $R'$ | |
|---|---|---|---|---|---|---|
| | **ReVS** | **CCP-RM** | **ReVS** | **CCP-RM** | **ReVS** | **CCP-RM** |
| `a2time` | 1,460 | 1,460 | 8,650 | 8,650 | 8,650 | 8,650 |
| `aifftr` | 480 | 480 | 670 | 670 | 670 | 670 |
| `aifirf` | 6,300 | 6,300 | 300 | 300 | 6,300 | 6,300 |
| `aiifft` | 410 | 410 | 8,500 | 8,500 | 8,500 | 8,500 |
| `basefp` | 420 | 420 | 300 | 300 | 420 | 420 |
| `bitmnp` | 370 | 370 | 3,570 | 3,570 | 3,570 | 3,570 |
| `cacheb` | 300 | 300 | 690 | 690 | 690 | 690 |
| `idctrn` | 300 | 300 | 300 | 300 | 300 | 300 |
| `iirflt` | 300 | 300 | 300 | 300 | 300 | 300 |

Table 6.4 reports the minimum number of measurements ($R'$) deemed sufficient for a reliable application of MBPTA according to CCP-RM and ReVS for IL1 and DL1. The final number of runs required for each technique is the maximum of both (IL1 and DL1). The accuracy of CCP-RM is confirmed by the fact that *in all cases it computes the same values as* ReVS. In particular, when CCP-RM identifies that conflictive placement can occur, it returns the same address combinations that are top-ranked by ReVS, or with a very close miss-impact. For the IL1, for six benchmarks (`ia2time, iaifftr, iaifirf, ibitmnp, icanrdr, iidctrn`) CCP-RM does not return any conflictive address combination: this is explained by the fact that in those cases all address combinations returned by ReVS are already upper-bounded with the default number of measurements by MBPTA. For other benchmarks, CCP-RM also returns no address combinations as potentially conflictive for high cardinalities of $K$, while ReVS does. This happens when the conflictive impact is actually caused by address groups smaller than $K$ that are instead considered by CCP-RM.

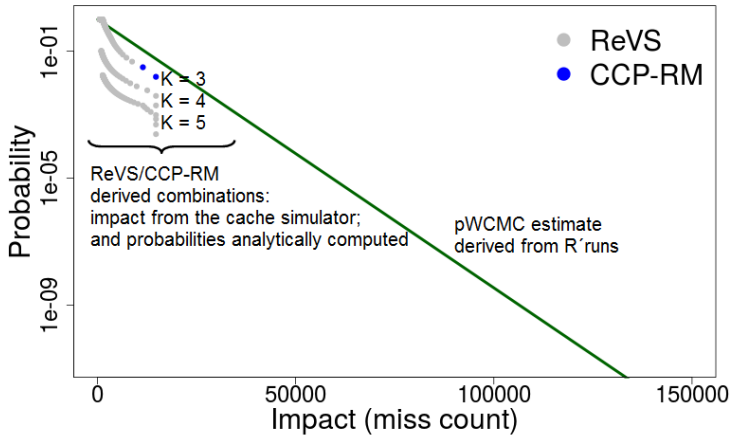



Figure 6.2: pWCMC for `cacheb` and a DL1 RM cache.

For instance, Figure 6.2 shows that, although conflictive placements – referred to as address combinations – can occur with cardinalities $K$=4 and $K$=5, as recognised by ReVS, those placements are upper bounded by the truly conflictive placement of combinations with cardinality $K$=3.

The exception to this comes from the case in which ReVS identifies for high values of $K$ combinations which, in fact, are the addition of two or more independent combinations. For instance, ReVS identifies combinations for $K$=6 that, in reality, correspond to two combinations of $K$=3 occurring at the same time. As explained before, EVT needs to observe high-impact events, but not their combination. Thus, this difference has no influence on $R'$.

### 6.4.3 Evaluation of CCP-RM Effectiveness

With the purpose of entailing an increase in the number of conflicts and hence, further stressing CCP-RM, in this section we evaluate CCP-RM for a relatively small cache 4KB 32B/line 2-ways IL1 and DL1 cache (default configuration in Section 3.1.1). We also focus on full-size EEMBC benchmarks to assess whether our method effectively improves MBPTA reliability by identifying potentially unobserved CCP. To that end, we compare the number of measurements required by a default application of MBPTA ($R$) against those required by CCP-RM ($R'$), for a set of EEMBC benchmarks. Note that for this experiment using all addresses, ReVS could not be used due to its exponential execution time requirements with the number of benchmarks' number of unique addresses.

For CCP-RM Table 6.5 reports different values of $R$ and $R'$. Only for one benchmark (`cacheb`) the default MBPTA application asks for more runs than

Table 6.5: CCP-RM on EEMBC ('lhood' stands for likelihood)

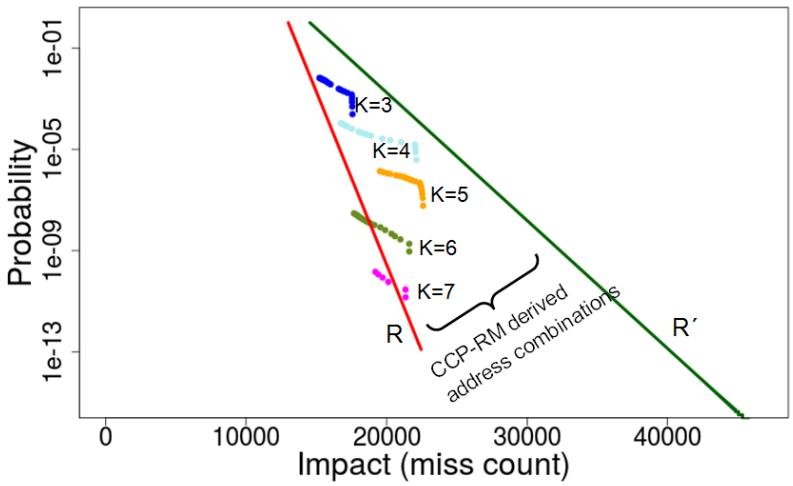
| | CCP-RM | | | | MBPTA | |
|---|---|---|---|---|---|---|
| | $R'_{IL1}$ | $R'_{DL1}$ | $R'$ | lhood($R'$) | $R$ | lhood($R$) |
| a2time | 300 | 730 | 730 | $10^{-9}$ | 300 | $2.00 * 10^{-4}$ |
| aifftr | 300 | 6,160 | 6,160 | $10^{-9}$ | 300 | $3.64 * 10^{-1}$ |
| aifirf | 470 | 22,490 | 22,490 | $10^{-9}$ | 370 | $7.11 * 10^{-1}$ |
| aiifft | 300 | 110,000 | 110,000 | $10^{-9}$ | 74,600 | $7.88 * 10^{-7}$ |
| basefp | 320 | 1,120 | 1,120 | $10^{-9}$ | 960 | $1.93 * 10^{-8}$ |
| bitmnp | 300 | 310 | 310 | $10^{-9}$ | 300 | $1.95 * 10^{-9}$ |
| cacheb | 460 | 390 | 460 | $10^{-9}$ | 500 | $1.65 * 10^{-10}$ |
| idctrn | 350 | 1,050 | 1,050 | $10^{-9}$ | 500 | $5.18 * 10^{-5}$ |
| iirflt | 300 | 930 | 930 | $10^{-9}$ | 320 | $8.00 * 10^{-4}$ |

those actually needed to observe conflictive placements. In this case, 460 runs suffice for MBPTA to converge in the cache miss domain, but few more runs are needed in the execution time domain to have enough high execution time values to converge due to variations across random samples. For the remaining eight benchmarks, collecting $R$ measurements results in a probability of not capturing CCP for RM higher than the established threshold, see $lhood(R)$ column. In fact, for two benchmarks (aifftr, iirflt) we observe that the EVT projection using $R$ does not upper-bound the Empirical Complementary Cumulative Distribution Function (ECCDF) applied on an arbitrarily large number of runs, which is actually upper-bounded when the probabilistic Worst-Case Execution Time (pWCET) is estimated using $R'$ runs as determined by CCP-RM. Figure 6.3 reports the results for one of those benchmarks, aifftr.

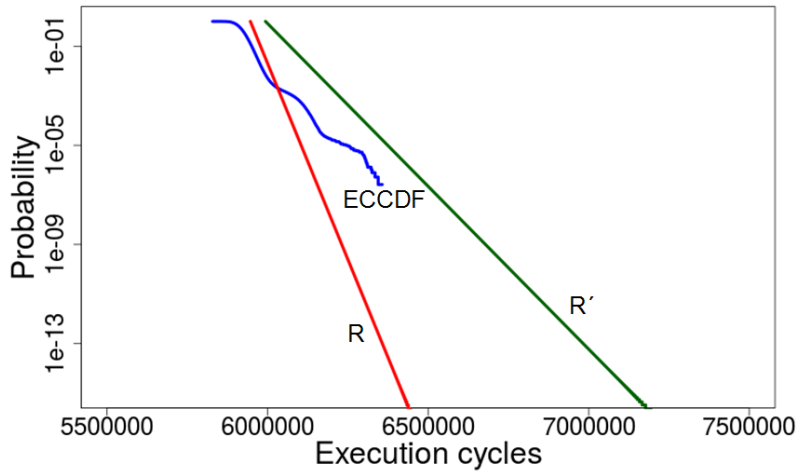
While in the comparison of our method with MBPTA we report the likelihood of missing relevant placements at a probability level of $10^{-9}$, the number of measurements computed is not dependent on this probability level. In fact, as we can see in Figure 6.3, the CCPs are assessed against pWCMC at all probability levels higher or equal than this one.

In Figure 6.3(a) the pWCMC estimate derived with $R$ runs and $R'$ is plotted against the conflictive sets of addresses (from 3 to 7 addresses) found by CCP-RM for DL1. As it can be observed, for the default $R$ the pWCMC does not cover all relevant address combinations, while with the CCP-RM provided $R'$ the resulting pWCMC does upper-bound all conflictive address combinations.

In the time domain we take as term of comparison the ECCDF derived from four million execution times we collected. Figure 6.3(b) shows that the pWCET curve obtained from $R$ runs does not upper-bound the ECCDF. The pWCET obtained with $R'$ runs returned by CCP-RM, instead, upper-bounds the ECCDF.




(a) pWCMC estimate with $R$ and $R'$ runs




(b) pWCET estimate with $R$ and $R'$ runs

Figure 6.3: EVT projections for benchmark `aifftr` with RM caches

### 6.4.4 CCP-RM Execution Time Requirements

The main execution time requirement of CCP-RM comes from the cache simulations (Section 6.3.2). In order to run the simulations, we used a cluster running 100 jobs in parallel.

- For the controlled scenario, CCP-RM requires 1 minute per program on average. ReVS, due to its complete exploration approach, required 2 hours.
- With full benchmarks, CCP-RM requires on-average 18 minutes per bench-

Table 6.6: CCP-RM results on the Railway case study.

| | CCP-RM | | | MBPTA |
|---|---|---|---|---|
| | $R'_{IL1}$ | $R'_{DL1}$ | $R'$ | $R$ |
| TEST0 | 1,560 | 300 | 1,560 | 300 |
| TEST1 | 300 | 410 | 410 | 300 |
| TEST2 | 300 | 340 | 340 | 300 |
| TEST3 | 350 | 300 | 350 | 300 |
| TEST4 | — | — | 300 | 300 |
| TEST5 | 10,300 | 300 | 10,300 | 300 |
| TEST6 | — | 3,200 | 3,200 | 300 |
| TEST7 | 1,240 | 300 | 1,240 | 300 |
| TEST8 | — | — | 300 | 300 |
| TEST9 | — | 300 | 300 | 300 |

mark. With ReVS, simulations take longer than affordable (years) to be executed.

Our results show that CCP-RM results in affordable execution time requirements.

## 6.5 Railway Case Study on RM Implementations on an FPGA

We also evaluate CCP-RM with a real industrial case study from the railway domain presented in Section 3.3.3. We run measurements on a LEON3 FPGA board modified to support RM cache (see Section 3.1). The case study is executed by the end-user on the FPGA (following the exact specifications of the FPGA setup). Both first-level caches are 16KB 4-way, with 32B cache line size for instructions and 16B for data (FPGA configuration in Section 3.1.1).

Table 6.6 reports the number of runs identified by CCP-RM ($R'$) for IL1, DL1 and globally, against the default number required by MBPTA for convergence ($R$). Both $R$ and $R'$ measurements suffice to upper bound all CCPs in the RM setup. Since the number of segments for each benchmark is generally low (between 4 and 13), conflictive behaviour can occur only for combinations of addresses of low cardinality, which are more likely to be observed already with a moderate number of runs. For data traces of two benchmarks (TEST8 and TEST9), the number of segments does not exceed cache associativity. For data trace of TEST2 and instruction traces of groups of benchmarks (TEST0, TEST1, TEST2, TEST3,

TEST5, TEST6, TEST7 and TEST9) the method does not identify any CCP: even though CCP can exist in theory, as the number of segments exceeds associativity, the addresses from distinct segments are barely interleaved, hence CCP-RM attaches small guilt values. For data traces of (TEST0, TEST1, TEST3, TEST5, TEST7), the method identifies relevant combinations at most for a single $K$ value (5 in all cases apart from TEST5 with 6). However, each of them is already upper bounded by the number of runs required by a default MBPTA application, 10,300 at most. Further, by comparing the EVT projection with the actual impact (ECCDF projection over 10 million observed miss counts), we observed that pWCMC estimates are very close to actual values.

Regarding computation time for the railway case study, CCP-RM took less than 0.5s on average per input vector to identify conflictive placements and less than 5s on average for cache simulations.

## 6.6 Summary

In this chapter, we propose CCP-RM, the conflictive-placement detection mechanism for high-performance TRC deploying RM placement. CCP-RM identifies the CCPs that result in high execution times. We exploit this information to derive the minimum number of measurements $R'$ to be performed so that the probability of missing the impact of those CCPs is below a configurable threshold (e.g. $10^{-9}$). The adoption of CCP-RM guarantees a cache-conflictive placement aware, reliable application of MBPTA. Our results using benchmarks and a real case study, respectively ran on a simulator and a real board deploying RM caches, show the effectiveness of CCP-RM in identifying CCPs and deriving an appropriate value for $R'$.

# Chapter 7

# Reaching Cache Representativeness and Path Coverage

## 7.1 Introduction

In Chapters 4, 5 and 6, we focus on Sources of Jitter (SoJ) stemming from the platform behaviour. We presented two computationally tractable methods that work on top of MBPTA to factor in cache layouts building on randomisation properties called Conflictive Cache Placements for hash Random Placement method (CCP-hRP) and Conflictive Cache Placements for Random Modulo Placement method (CCP-RM). Here we call them jointly Conflictive Cache Placements method (for arbitrary Random Placement) (CCPX). The first is appropriate to use for cache memories deploying hash Random Placement (hRP) and the second for Random Modulo Placement. CCPX guarantees that the number of execution time measurements fed to MBPTA is large enough to observe the effect of all relevant cache placements, whose probability of occurrence may be sufficiently low not to be observed in the default number of runs of MBPTA. If the number of runs satisfies this criteria, measurements are regarded as *cache-layout representative*. However, CCPX has only been demonstrated on single-path (single input) analysis (CCP-hRP in Chapter 5 and CCP-RM in Chapter 6).

The user also needs to take into account the execution time variability coming from software behaviour. Path Upper-Bounding (PUB) (see Section 2.3.2) builds an *upper-bounding program* to automatically extend path coverage beyond the subset of paths exercised with user's input vectors, effectively relieving the user from the enumeration of all paths. Nonetheless, the minimum number of runs required for factoring in cache representativeness for a reliable application of MBPTA has

not been explored in the scope of automatic full-path coverage, as provided by PUB.

**Contributions.** Overall, no existing MBPTA solution attains cache representativeness and full path coverage simultaneously. We cover this gap by proposing the first approach to determine the minimum number of runs required in MBPTA-compliant platforms for MBPTA techniques delivering full path coverage automatically. In particular, we make the following contributions:

1. We provide an in-depth analysis of the relation between the memory address sequences in the original program and its modified (*pubbed*) version, identifying key properties that guide the joint application of PUB and CCPX. We show that no relation can be made between the number of runs needed to account for conflictive cache scenarios for different versions of the program.

2. We devise a reliable combined application of PUB and CCPX mechanisms in which we create a *modified* program for analysis purposes only where cache impact is upper-bounded across any path, and derive the minimum number of runs required to capture in the test campaign cache layouts resulting in high execution times. As outcome PUB+CCPX returns a probabilistic Worst-Case Execution Time (pWCET) estimate that accounts for both path variability and cache conflictive scenarios at relevant probabilities.

3. We provide evidence of the reliability of the approach through a set of Corollaries and show its effectiveness in terms of computation cost and efficiency in achieving cache representativeness and full path coverage through evaluation on the Mälardalen benchmark suite. We further provide an in-depth analysis of the approach using the representative binary search benchmark from the Mälardalen suite.

## 7.2 Achieving Full Path Coverage and Cache Representativeness

### 7.2.1 Difficulties Integrating PUB and CCPX

In order to reach full path coverage, PUB modifies programs which result in address sequences that are very likely different from that of the original program. This complicates reasoning on CCPX, i.e. on the number of runs to ensure that random cache layouts causing high execution times are captured. Intuitively, since PUB adds cache accesses to the original program, one would expect the number of runs required for CCPX to reduce, since the more addresses in a program, the more likely they conflict in the same cache set, and hence the lower the number of runs

to randomly hit one of those cases. However, this is not the case. To illustrate so, let $M^j_{orig}$ be the address sequence of path $j$. Let $ins(M, x)$ be an operator to insert an address $x$ in a sequence $M$, which can be done in multiple ways, as long as the same ordering of address accesses in the original path is preserved. The sequence of addresses of a path $j$ in the *pubbed* program ($M^j_{pub}$) is the result of inserting several addresses, $n_j \geq 0$, in $M^j_{orig}$.

$$\forall j \in paths(P) : M^j_{pub} = ins(...(ins(ins(M^j_{orig}, x_1), x_2)...), x_{n_j}) \tag{7.1}$$

No relationship can be established between the number of runs required to achieve representativeness in $M^j_{orig}$ and $M^j_{pub}$. That is, $R_{CCPX}(M^j_{orig})$ can be higher/lower than $R_{CCPX}(M^j_{pub})$, where $R_{CCPX}(M)$ is the minimum number of runs required for address sequence $M$ as determined by CCPX. This is shown in two examples.

**Case 1:** $R_{CCPX}(M^j_{orig}) < R_{CCPX}(M^j_{pub})$

Let us assume a program with two paths, $\mathrm{P}^1_{orig}$ and $\mathrm{P}^2_{orig}$ with address sequences $M^1_{orig} = \{ABCA\}^{1000}$ and $M^2_{orig} = \{ADEA\}^{1000}$, and a cache with $S$=8 cache sets and $W$=4 ways. The exponent of the address sequences represents the number of times that those address sequences repeat. PUB application tries to minimise the number of addresses inserted to reduce the impact in the pWCET estimate. Hence, a potential result of applying PUB could be as follows:

$$\begin{aligned} M^1_{pub} = M^2_{pub} = (ins(ins(M^1_{orig}, D), E))^{1000} = \\ (ins(ins(M^2_{orig}, B), C))^{1000} = \{ABCDEA\}^{1000} \end{aligned} \tag{7.2}$$

Since both $M^1_{orig}$ and $M^2_{orig}$ have 3 different addresses each, none of them can exceed the space in a cache set (4 ways), i.e. they cannot generate a cache layout resulting in high execution time. As a result, CCPX does not impose a minimum number of runs higher than that required by the standard application of MBPTA. Note that, for instance, in the case of $M^1_{orig}$ addresses $A$, $B$ and $C$ will end up fitting in a cache set after, potentially, few random replacements.

Instead, $M^1_{pub}$ (and $M^2_{pub}$) has 5 different addresses, so it exceeds the space in a cache set. For instance, if addresses $A$, $B$, $C$, $D$ and $E$ are randomly placed in the same set, they will experience at least 1000 misses due to the address not present in the cache set in each of the 1000 traversals of the address sequence. Hence, CCPX determines that a sufficient number of runs is needed to guarantee that this sequence is not observed with a maximum configurable probability. The

probability of observing such a sequence is exactly $(1/S)^4 = 0.000244$, so the probability of not observing it in $R$ runs is $(1 - 0.000244)^R$. If such probability needs to be lower than $10^{-9}$ to be regarded as negligible w.r.t. the corresponding safety standard, then $R > 84875$. Hence, PUB address sequences may require a higher minimum number of runs than the original ones.

**Case 2:** $R_{CCPX}(M^j_{orig}) > R_{CCPX}(M^j_{pub})$

Let us now consider that $M^1_{orig} = \{ABCDEA\}^{1000}$ and $M^2_{orig} = \{ABCDFA\}^{1000}$. In this case, and building upon the results obtained in the example before, each original path would require $R > 84875$ runs. $M^1_{pub}$ (and $M^2_{pub}$) would be $\{ABCDEFA\}^{1000}$, thus including 6 different addresses. Abrupt cache miss counts would require 5 out of the 6 addresses to be placed in the same set, whose probability is $(1/S)^4 \cdot 6 = 0.00146$. Therefore, $R > 14138$. Hence, PUB address sequences may require a lower minimum number of runs than the original sequences.

### 7.2.2 Sensible Application of PUB and CCPX

Since no relation can be established between the minimum number of runs required by the original program and its *pubbed* version, our method relates both using only the pWCET and their execution time distributions. To that end, we build on the following reasoning.

**Observation 1**: *The probabilistic Execution Time Distribution (pETD) obtained for any given path of the* pubbed *program is an upper-bound to the pETD of* any *path of the original program [76].*

This observation is formalised in Equation 2.1 in Chapter 2. Note that the pETD refers to those distributions that would be obtained by running the corresponding program paths an infinite number of times. Thus, while this observation would generally hold also for the statistical samples obtained from the pETD, it can only be guaranteed to hold for the reference distributions due to the statistical nature of samples.

**Observation 2**: *The pWCET curve obtained from the sample of a pETD is (probabilistically) an upper-bound for the execution time distribution that has been sampled [7].*

Under the appropriate setup for execution time collection, e.g. ensuring independence across experiments, and sources of randomisation injected at hardware or software level, it has been shown that pWCET estimates upper-bound pETD [7, 110].

**Corollary 1**: *From Observations 1 and 2, it follows that the pWCET curve obtained from a sample of any given path of a* pubbed *program upper-bounds the pETD of any path of the original program.*

From Corollary 1, we make the following key observation:

**Observation 3**: *Any path choice from the* pubbed *program is equally valid to derive a minimum number of runs and a reliable pWCET distribution that upper-bounds the pETD of all paths in the original program.*

For instance, by applying CCPX on an arbitrary path $\mathrm{P}^j_{pub}$ we obtain that the minimum number of runs needed for cache representativeness is $R^j_{pub}$. Since $pWCET(\mathrm{P}^j_{pub})$, when it is derived from a sample of at least $R^j_{pub}$ runs, is an upper-bound for the pETD of any path of the original program, $pWCET(\mathrm{P}^j_{pub})$ achieves both *full path upper-bounding and cache representativeness.*

Some observations must be made to complete the view on the approach taken to use PUB and CCPX appropriately.

**Observation 4**: *For two different paths $j$ and $k$ of a pubbed program, CCPX results in $R^j_{pub}$ and $R^k_{pub}$ minimum number of runs, respectively. In general, no relation can be established between $R^j_{pub}$ and $R^k_{pub}$ since the insertion of the missing addresses in each path with $ins(M, x)$ may not lead to identical address sequences.*

For instance, if $M^1_{orig} = \{ABA\}$ and $M^2_{orig} = \{ACA\}$, by applying PUB we could obtain $M^1_{pub} = \{ABCA\}$ and $M^2_{pub} = \{ACBA\}$. In the general case, address sequences can be arbitrarily different, thus leading to different results when applying CCPX. This is analogous to the scenario when relating the number of runs of the original and *pubbed* paths, described in Section 7.2.1.

Nevertheless, building on previous observation, even if a different *pubbed* path is used, the pWCET remains reliable and representative with respect to all paths in the original program. Building on the reasoning above, we also note that:

**Observation 5**: *The pWCET estimates obtained for $\mathrm{P}^j_{pub}$ and $\mathrm{P}^k_{pub}$ cannot be related to each other. In other words, potentially, the pWCET estimate for path $\mathrm{P}^j_{pub}$ can be above or below that of $\mathrm{P}^k_{pub}$ for the full range of probabilities or only for a subset of them. In any case, this is irrelevant given that both pWCET estimates are reliable and representative upper-bounds of all paths of the original program.*

**Corollary 2**: *Since every single pWCET estimate for each* pubbed *path is a reliable and representative upper-bound for the pETD of all paths of the original program, we can take as pWCET for any given exceedance threshold the lowest value across all* pubbed *paths.*

Corollary 2 builds upon the fact that the pWCET estimates of all paths are equally reliable and representative. Differences only relate to pessimism, so we can trade analysis cost for tightness by analysing an increased number of paths of the *pubbed* program. While only one is strictly needed, the larger the number of paths analysed, the higher the chances of obtaining tighter pWCET estimates. Note, however, that no guarantee exists on whether tighter pWCET estimates will be obtained since it could turn to be that the lowest one across the available paths is the first one we obtain.

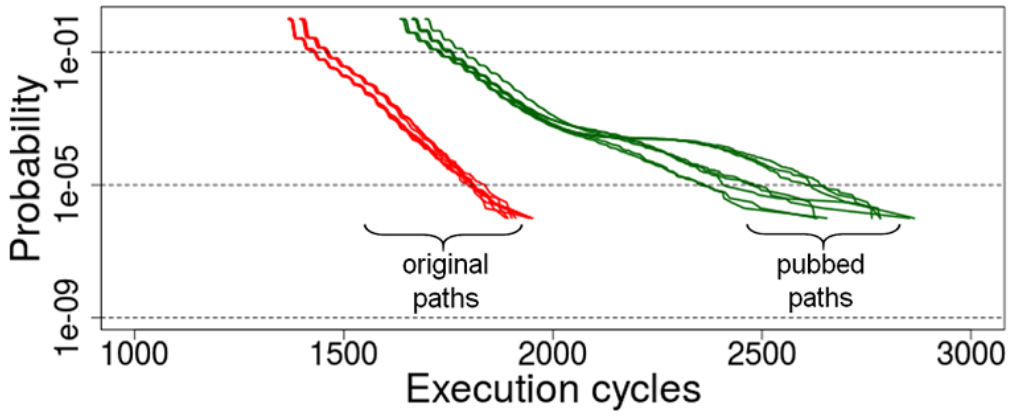



Figure 7.1: ECCDF for `bs`'s original paths and pubbed paths.

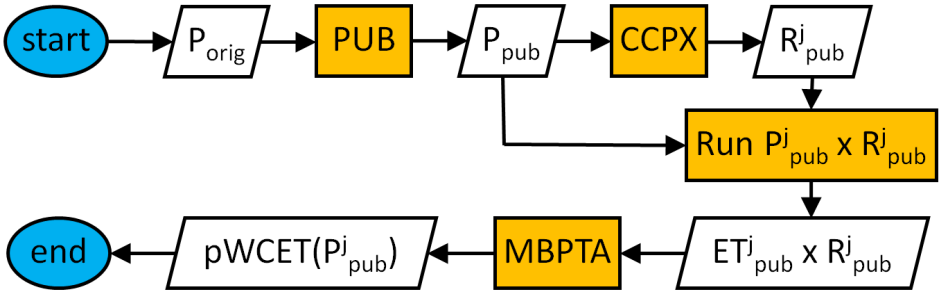



Figure 7.2: Overall application process of PUB and CCPX.

Overall, the application process of PUB and CCPX comprises the steps shown in Figure 7.2. Starting from the original program $P_{orig}$, we apply PUB to generate the pubbed program $P_{pub}$. By default, we choose an arbitrary input vector provided by the user, which executes path $j$, to collect its address sequence. Optionally, we may explore more paths to search for tighter pWCETs, as explained before. We analyse the address sequence with CCPX to derive the number of measurements required, $R^j_{pub}$. We execute the pubbed program with the same input vector as many times as dictated by CCPX, thus producing a sample of execution times, $ET^j_{pub}$. Finally, we apply MBPTA on the resulting sample to obtain a pWCET estimate that reliably upper-bounds any path of the original program, under any layout of objects in memory/cache.

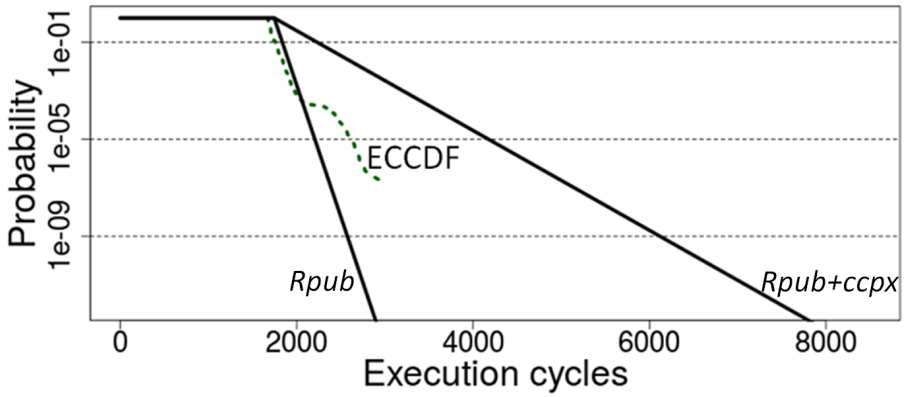



Figure 7.3: pWCET for `bs` (path $v9$) with $R_{pub}$ and $R_{pub+ccpx}$ runs.

### 7.2.3 A Practical Example

We illustrate how our method works with an example based on the binary search (`bs`) benchmark from the Mälardalen suite (Section 3.3.2). Details on the hardware setup are provided later in Section 7.3. `Bs` is a multipath program whose input determines the number of loop iterations and the actual path traversed. For the sake of illustration, we stick to its default input to set the loop bounds (15 integer elements) and explore the different type of input vectors (related to the degree of ordering of the elements to sort) that lead to the maximum number of iterations. In particular, 8 different cases lead to different paths triggering the maximum number of iterations.

The first experiment provides evidence for Corollary 1: we collect 1,000,000 execution times for each of the 8 original paths characterised by the maximum number of iterations, before and after applying PUB. Their ECCDF is shown in Figure 7.1. As it can be seen, each *pubbed* path upper-bounds *all* original paths. Hence, when applying MBPTA to any pubbed program, resulting pWCET curves upper-bound all original paths. For instance, the highest observed execution time across all paths is below 2,000 cycles, whereas the lowest pWCET estimate across all *pubbed* paths at an exceedance probability of $10^{-6}$ per run (so at the same probability as the highest execution time, $1/R = 10^{-6}$) is 2,297 cycles (for path $v9$).

In a second experiment, we compute the pWCET curve for input $v9$, with $R_{pub} = 1,000$ and $R_{pub+ccpx} = 70,000$ (as computed by CCPX), see Figure 7.3 that also shows the ECCDF with 6,000,000 runs of the path triggered with $v9$ (dashed green line). A sample of $R_{pub} = 1,000$ runs does not capture the 'knee' (abrupt variation) in the ECCDF, so when used with MBPTA it fails to capture the abrupt change. This change is caused by a cache placement with high impact

on execution time that occurs with low probability. With $R_{pub+ccpx} = 70,000$ runs the impact of that cache placement is observed and the resulting pWCET upper-bounds it. In general, for all inputs ($v_i$), see Table 7.2, CCPX requires more runs than PUB to account for CCPs.

Table 7.1: BS. Execution Time Domain.

| | Runs | | pWCET@$10^{-12}$ | |
|---|---|---|---|---|
| | $R_{pub}$ | $R_{pub+ccpx}$ | PUB | PUB+CCPX |
| v1 | 1,000 | 40,000 | 3,212 | 4,125 |
| v3 | 2,000 | 20,000 | 3,149 | 4,432 |
| v5 | 50,000 | 50,000 | 6,712 | 6,712 |
| v7 | 20,000 | 20,000 | 4,317 | 4,317 |
| v9 | 1,000 | 70,000 | 2,850 | 7,571 |
| v11 | 1,000 | 8,000 | 3,455 | 4,003 |
| v13 | 1,000 | 80,000 | 3,026 | 7,377 |
| v15 | 6,000 | 40,000 | 2,995 | 3,694 |

Table 7.2: Runs for PUB, PUB+CCPX and MBPTA.

| | Runs | | |
|---|---|---|---|
| | $R_{orig}$ | $R_{pub}$ | $R_{pub+ccpx}$ |
| bs | 1,000 | 1,000 | 40,000 |
| cnt | 10,000 | 2,000 | 70,000 |
| fir | 6,000 | 9,000 | 600,000 |
| janne | 3,000 | 1,000 | 200,000 |
| crc | 3,000 | 5,000 | 10,000 |
| edn | 1,000 | 1,000 | 70,000 |
| i.sort | 40,000 | 40,000 | 80,000 |
| jfdc | 2,000 | 2,000 | 50,000 |
| m.mult | 200,000 | 200,000 | 200,000 |
| fdct | 8,000 | 8,000 | 8,000 |
| ns | 3,000 | 3,000 | 500,000 |

## 7.3 Evaluation

We model a pipelined in-order processor with Instruction Cache Level 1 (IL1) and Data Cache Level 1 (DL1) in the simulator based on the cycle-accurate SoCLib framework (see Section 3.1). IL1/DL1 caches are 4KB 2-way 32B/line, and implement hRP and random replacement policy (SoCLib configuration in Section 3.1.1). Therefore in our experiments we apply CCP-hRP variant of CCPX. The content of cache memories is flushed before each run of a program. We conduct the evaluation using the Mälardalen benchmark suite (Section 3.3.2) with default input sets, considering them representative of the worst-case for loop bounds. Some of the analysed benchmarks are single-path, so the execution time variability is due to hardware effects, while for others execution time varies depending on input values. Applying PUB+CCPX took approximately 15 minutes per benchmark on average.

### 7.3.1 Representative Number of Runs

First, we have collected the number of runs required for MBPTA convergence on the *pubbed* version of the programs, $R_{pub}$, and the number of runs on those same *pubbed* versions as reported by CCPX (in particular its CCP-hRP variant), $R_{pub+ccpx}$. Results are shown in Table 7.2. As shown, despite in some cases the number of runs does not increase, CCPX often requires $R_{pub+ccpx} > R_{pub}$ to attain representativeness. In practice, $R_{pub}$ runs often provide enough measurements from the tail of the execution time distribution, but not with enough confidence. Moreover, in some cases, as shown later in the discussion of Figure 7.4, using $R_{pub}$ runs instead of $R_{pub+ccpx}$ may lead to pWCET estimates that do not account for some events that occur with significant probability. For completeness, we also report the number of runs required on the original version of the programs, applying neither CCPX nor PUB, so only determined by MBPTA ($R_{orig}$). As shown, $R_{pub+ccpx} > R_{orig}$, although there is no specific relation between $R_{pub+ccpx}$ (or $R_{pub}$) and $R_{orig}$, since they correspond to different programs in practice.

### 7.3.2 pWCET Estimates

For some benchmarks, the particular input data available leads to the worst-case path (or they are simply single-path). Hence, for those benchmarks, we can determine that any difference between the pWCET obtained with and without PUB is fully accountable to overestimation introduced to attain full path coverage. On the remaining benchmarks, instead, it is not possible to clearly tell apart how much of the pWCET increase corresponds to unobserved (worst) paths and how much is a plain overestimation.

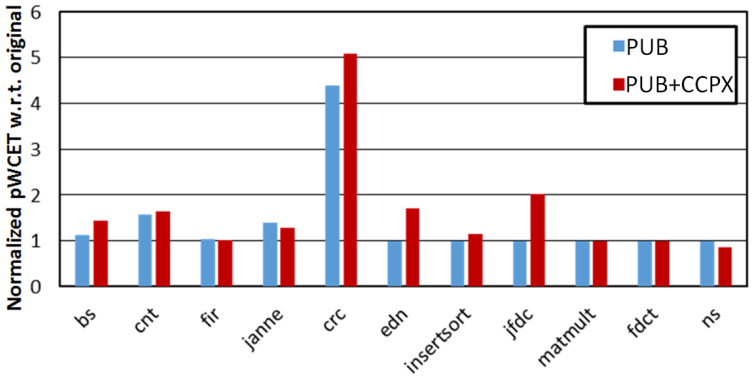



Figure 7.4: pWCET estimates of PUB and PUB+CCPX with respect to the original pWCET with user-provided input sets at the probability level $10^{-12}$.

Some of the benchmarks evaluated are single path (the 6 rightmost ones in Figure 7.4): `edn`, `insertsort`, `jfdc`, `matmult`, `fdct` and `ns`. Others (the 4 leftmost ones in Figure 7.4) have multiple paths, but the default input vector used already triggers the worst-case path: `bs`, `cnt`, `fir` and `janne`. Finally, for `crc` we are unable to identify the worst-case path and, based on code inspection, we are highly confident that the default input does not trigger it.

Figure 7.4 shows the pWCET increase caused by the application of PUB and PUB+CCPX with respect to the direct application of MBPTA with neither PUB nor CCPX at the probability level $10^{-12}$. We have chosen the probability level representative for tasks of the highest criticalities in Critical Real-Time Embedded Systems (CRTES). First, we observe that the application of PUB increases pWCET estimates between 4% and 59% for the 4 leftmost benchmarks. In this case, since we know that we already exercised the worst-case path, we can tell this increase is pessimism due to PUB. For the 6 rightmost benchmarks PUB has no impact as they are single-path and hence, PUB application is innocuous. Finally, in the case of `crc` PUB leads to a 4.4x higher pWCET estimate (340% above that without PUB). While code inspection reveals that this increase captures unobserved paths, it is hard to tell apart how much of this increase is needed to attain path coverage and how much is pessimism incurred by PUB. However, this is the general situation we can expect in real applications: the worst-case path cannot be determined a priori and PUB accounts for it automatically.

When applying CCPX on top of PUB, we observe a variety of different behaviours:

1. In most cases (e.g. `bs` and `fir`) pWCET variation is relatively small. This occurs because $R_{pub}$ runs already included execution time measurements for all relevant cache placements. Hence, variations are mostly caused by random

variations in the execution time sample used. Note that this may lead to either higher or lower pWCET estimates.

2. In some cases, CCPX accounts for cache placements likely not observed otherwise, such as for `edn` and `jfdc`.

3. Finally, in the case of `ns`, the pWCET estimate decreases by 15% with PUB+CCPX w.r.t. PUB only. In this case, we realise that execution times observed are already close to the maximum execution time possible in practice. Moreover, the application of CCPX increases the number of runs from 3,000 to 500,000 to guarantee that high execution times are observed sufficiently. As a consequence, the set of highest execution times obtained with $R_{pub+ccpx}$ is relatively more homogeneous, since an increasing number of runs makes execution times approach the maximum possible value asymptotically. As a result, MBPTA delivers a tighter bound and hence, the pWCET estimate decreases by 15%.

## 7.4 Summary

In this chapter, we tackle the issues of achieving full path coverage and representativeness against potential cache layout scenarios that may appear with low-probability. We demonstrate that both objectives can be achieved simultaneously, supporting our proposed method with empirical evidence. We show that the execution time distribution of any path of a *pubbed* program upper-bounds the execution time distributions of all paths in the original program. Then, we also show that by collecting a sufficient number of execution time measurements of any *pubbed* path, the pWCET estimate upper-bounds all paths in the original program under all cache layouts occurring with relevant probabilities. As a result, we simultaneously deliver both, full path coverage and cache representativeness.

# Chapter 8

# Applicability of EVT Fit for Industrial Quality WCET Analysis

## 8.1 Introduction

The quest for novel Worst-Case Execution Time (WCET) analysis methods capable of keeping pace with the trend towards more complex hardware has caused considerable interest to arise in statistical tools based on Extreme Value Theory (EVT), see Section 2.2.2. Since EVT builds on observations, it is particularly attractive to real-time systems industry. In particular, it fits very naturally with Measurement-Based Timing Analysis (MBTA), which in spite of its theoretical limitations dominates industrial practice for WCET analysis. Measurement-Based Probabilistic Timing Analysis (MBPTA) approach (Section 2.2) combines EVT's potential with the cost/benefit attractiveness of MBTA. MBPTA much improves on standard Measurement-Based Deterministic Timing Analysis (MBDTA) by enabling the user to attach *quantitative confidence* to the probabilistic Worst-Case Execution Time (pWCET) bounds computed from timing measurements of the software program of interest, collected on the target platform.

Several works have consolidated various flavours of MBPTA [3, 27, 92, 107], showcasing industrial-strength experiments run on processor simulators [115] or real hardware boards [37, 51]. However, before being deemed ready for transfer to industrial practice, MBPTA needs to be proven to:

- Deliver *trustworthy* results.
- Be *adaptable* enough to embrace different kinds of software programs.

- Cause *lightweight* impact on an already effort-intensive validation and verification process (Section 1.2.2).

Arguably, the latter trait is a generally ascertained fact, with a score of industrial case studies [37, 51, 115] confirming that MBPTA requires an affordable number of measurement observations in addition to what normal practice already provides. The same cannot be said with respect to the *trustworthiness* and *adaptability* concerns, which have been questioned [86, 104] in the past. At the time of writing different views exist on how to guarantee the trustworthiness of results delivered by probabilistic methods and which class of programs is suitable to analyse with these methods (see Section 2.4). These concerns are not inherent to EVT (or MBPTA) per se, but rather pertain to the way EVT is applied to the WCET domain, which may be seriously fallacious.

**Contributions.** The work delivered in this Thesis relies on the assumptions surrounding the usage of the MBPTA techniques as developed in the PROARTIS and PROXIMA projects. In this Chapter, we reassess the assumptions underlying the way EVT is applied in this Thesis, with respect to the two concerns raised around the viability of the application of EVT in industrial practice:

1. **The *trustworthiness* of EVT** has been challenged noting that the quality of its results critically reflects the quality of its inputs (for data, coverage, state control) [86, 104], with poor inputs yielding wholly unsound probability models. This should not surprise in fact, as the application of EVT to observations over user-controlled program runs instead of uncontrolled (e.g. natural) phenomena needs very careful attention and adaptation. We show that this misunderstanding originates from solely minding that the maxima of the input data (i.e. execution-time observations) pass EVT's all-famous pre-requisite tests of identical distribution and (sufficient) independence [34, 107]. However, it stands to reason that the measurement observations that one collects reflect the execution conditions that the program incurs (for input data, coverage of execution space and state control) during analysis runs. In particular, those conditions are intended to bear a sound relation with the worst execution conditions that can arise at operation. Only if that relation can be asserted, the analysis-time measurements can be said to be **representative**, thus useful for MBPTA. Ensuring representativeness is very complex indeed, for it requires controlling all sources of significant variation in the execution conditions explored in the analysis process.

2. **The *adaptability* trait** has been criticised too, on the argument that EVT-based analysis would not apply to all classes of software programs [86]. The critique showed that EVT was unable to produce good-fit distributions even for programs that exhibited sufficient execution-time variability, and was not

applicable at all for those that had near-constant execution-time behaviour. This is where adaptation kicks in, which requires a profound understanding of what differentiates the application of EVT to execution time observations of the program runs from its use in other domains, where the observed events and their sources have an altogether different nature.

We show that the MBPTA techniques applied under the assumptions adopted in this Thesis, and established in the PROARTIS/PROXIMA projects, solve the adaptation problem and can be used to produce trustworthy pWCET results. In particular, we show how platform time randomisation and upper-bounding (see Section 2.2.1), combined with appropriate treatment of the observations coming from different program paths, allow the sound application of EVT, making it fit for transfer to industrial practice. We also show how system settings that lack those features (for example, time-deterministic execution platforms taken as-is) challenge representativeness. We back our claims with several illustrative examples.

## 8.2 EVT Specifics for WCET Domain

EVT cannot be applied sensibly without understanding the specifics of the domain of interest, and how their nature can be captured by the underlying theory and its mathematical devices. It is evident that (real-time) software programs differ substantially from the phenomena that EVT is classically used to study: this difference warrants thorough scrutiny, which must precede and enable adaptation. This appreciation, however, has only recently started to emerge: numerous efforts to apply EVT to the WCET problem did not really contemplate that need. In continuation we discuss the assumptions that characterise the problem domain of WCET analysis and review what they imply for the application of EVT, particularly the way it models the tail of the probability distribution.

### 8.2.1 WCET Existence and Finiteness

The base axioms to consider when applying EVT to the timing domain is that in order to produce useful results, WCET analysis rests on two main assumptions: (i) target programs execute a finite number of instructions; and (ii) each instruction executes in a finite number of cycles. Those assumptions guarantee that a WCET does actually exist and that an upper-bound to it is computable. Parametric static timing analyses have been proposed to soften the finiteness assumptions [25]: while they are useful to reuse results from static analysis, they only produce preliminary formulas that need instantiation before use. A real-time software program is therefore assumed to have a theoretical maximum execution time, the WCET, which needs to be estimated and upper-bounded for safety. For

this condition to hold true, the input space of the program itself may require to be known and/or controlled. For example, it would be impossible to derive an upper bound to the execution time of the Factorial function if we did not know or constrain its execution context, in this case, the range of admissible input values. We identify two main classes of relevant information on the input ranges that affect the search space, also known as *feasible region*, of the WCET:

**Finite execution conditions:** these are determined by input values that affect loop bounds and/or the depth of recursion when they are not hard-coded in the program. Excluding parametric methods, which we do not consider in this work, all timing analysis methods, whether static or measurement-based, deterministic or probabilistic alike, need the user to provide some information on finite execution conditions. The quality of that information may affect the tightness and trustworthiness of the analysis results. For instance, Static Timing Analysis (STA) may require the user to provide *flow facts* to fill in bounding information that could not be automatically derived by, e.g. data flow or range analysis.

**Path traversal conditions:** when the software program features multiple structural execution paths, identifying the set of feasible and relevant paths is crucial to precision or trustworthiness. This is especially true for MBTA, which critically depends on the availability of input vectors capable of providing *sufficient* path traversal conditions coverage, to warrant cognizant confidence that the paths exercised in the measurement observations include the path leading to the WCET. STA techniques do not need path traversal conditions information to achieve full path coverage, but excluding non-relevant or semantically infeasible paths, may improve the quality of their output.

### 8.2.2 Tail Distributions and pWCET Estimation

The existence and finiteness assumptions in the WCET problem domain impact the expressiveness of the Generalised Extreme Value and Generalised Pareto distribution models, especially for the characterisation of the shape parameter $\xi$ (see Section 2.2.2). As we noted in Section 2.2.2, the reversed Weibull family of distributions should be used when a maximum value exists and it is known. The former hypothesis generally holds for the WCET of real-time software programs; the latter does not. When the WCET is unknown, which is the case in the vast majority of situations in our problem domain, it can be upper-bounded by either a Gumbel or a Fréchet distribution, as the slope of their right tail decreases much more gently than the reversed Weibull one. The descent of the Fréchet distribution is gentler than that of its Gumbel correspondent, and therefore it upper-bounds the latter's right tail. The Fréchet distribution is most appropriate when a maximum value does not exist, which places it outside of the core WCET problem domain. Gumbel, instead, is most appropriate when a maximum value exists but

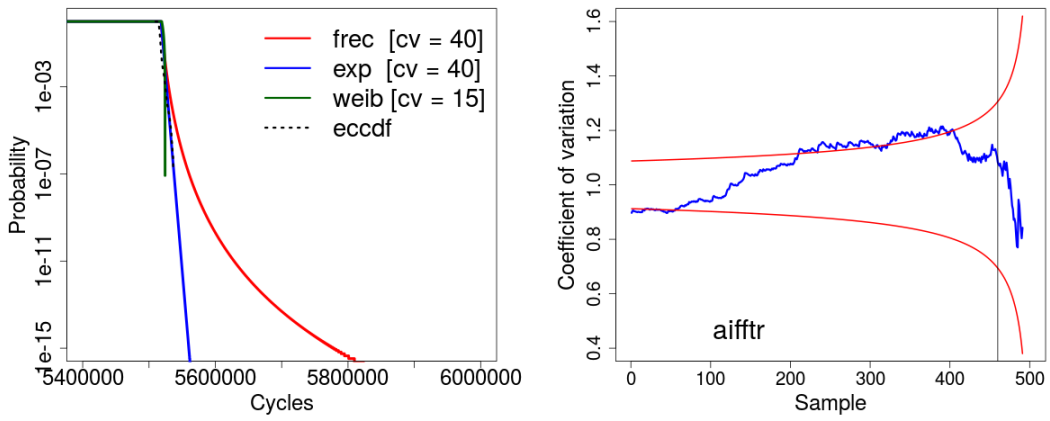



Figure 8.1: Effect of misclassified tails. The left part shows pWCET of `aifftr` benchmark modelled as Fréchet, Gumbel and Weibull distribution. The right part shows the coefficient of variation over the sample of 500 execution time measurements of `aifftr`.

it is unknown.

Methods as the Exponential Test [44] and the coefficient of variation [41] help to determine whether the sample fits a Fréchet, Weibull or Gumbel distribution (or their Generalised Pareto distribution counterparts). Yet, for the argument we have given above (further developed by J. Abella et al. [7]), the extreme behaviour of the execution-time distribution of a real-time software program can *always* be modelled with a Gumbel distribution: it may not be the tightest one, but it is always – at least – a safe over-approximation of it. Unless we assume that our program of interest has an unbounded WCET, using the Fréchet distribution should be considered inappropriate. If a Fréchet distribution were to best fit a given sample of execution-time measurements, this should happen because either the sample does not contain enough tail values or some of them do not really belong to the tail of the distribution. The remedy would be to increase the sample to capture more tail values or use different block sizes with block maxima (see Section 2.2.2) to discard less tail values.

To demonstrate our reasoning, we apply MBPTA to `aifftr`, a Fast Fourier Transform procedure included in the EEMBC automotive benchmark suite (Section 3.3.1), which we executed 1,000 times on an MBPTA-compliant platform. We first tested exponentiality with the coefficient of variation test (with 95% confidence): the right part of Figure 8.1 shows that the exponentiality hypothesis, considered for the 95 highest observations, cannot be rejected, since the blue line (that projects those 95 values) falls within the red lines, meaning that exponentiality cannot be rejected with 95% confidence. This means that less than

96 maxima observations are sufficient to approximate the Gumbel (exponential) pWCET distribution. Note, however, that passing such test means that data are compatible with the exponentiality hypothesis ($\xi = 0$), but the best fit will likely be non-exponential (although $\xi \approx 0$). As explained before, the decision of using an exponential tail as upper-bound resorts to knowledge about the domain (pWCET estimation), not about the best fit for the data. We obtained fitting pWCET distributions for the three cases plotted in the left part of Figure 8.1 (using point estimation, although confidence intervals for the rate of the Exponential distribution could also be considered): (1) Using the 40 highest values yielded a curve that was more a Fréchet than a Gumbel distribution. This is no surprise, in fact, for the probability of obtaining exactly $\xi = 0$ for a sample, as required for a Gumbel distribution, is close to nil; actual sampling typically yields $\xi > 0$ or $\xi < 0$. With 40 values from the sample, we had $\xi = 0.086$, which causes the tail to decrease polynomially (the red line in the plot). (2) Using any other number of values (always less than 96), e.g. 15, yielded $\xi = -0.55$. The resulting pWCET curve was in the domain of the reversed Weibull family (the green line in the plot). As noted earlier, going this way could lead to an unsound, optimistic bound. (3) Picking 40 maxima values and fitting the best Gumbel distribution to them, yielded the tightest and most reliable pWCET curve (the blue line in the plot), which tightly upper-bounds the actual sample (the dashed line) increased to 5,000,000 measurements to prove our point.

Using the best fit distribution from any family in the Generalised Extreme Value distribution (or Generalised Pareto distribution) can lead to arbitrarily pessimistic pWCET estimates. For instance, choosing an exceedance probability of $10^{-15}$ per run, the pWCET estimate for the test program would be around 5,560,000 cycles for the exponential fit with 40 excesses. For the best fit (leaning toward Fréchet), it would go to 5,780,000 cycles. However, for 95 values, whose exponentiality cannot be rejected, we obtain a Fréchet distribution and a pWCET of more than 23 million cycles.

Whenever the best fit is a reversed Weibull distribution, we can use a Gumbel equivalent in its stead since, for tail (extreme) behaviour, the Gumbel curve always over-approximates its Weibull correspondent. In other words, the reversed Weibull Complementary Cumulative Distribution Function (CCDF) may be above the Gumbel one in the first part of the curve, which corresponds to high probabilities, but it is going to be always below it for low probabilities. And only the latter matter in the WCET problem domain.

If the method used to test exponentiality rejects the hypothesis that the tail can be modelled (or upper-bounded) with an exponential tail (e.g. Gumbel or, at least, Weibull), then the sample size needs to be increased until the test passes.

EVT requires the random variable being observed to exhibit a “variable” tim-

ing behaviour. Lack of variability prevents EVT from producing good quality models and likely from converging to an exponential distribution. In theory, this can happen regardless of time randomisation when the timing behaviour of the program under analysis has a *degenerate distribution*. While this is extremely rare in practice, at least for real-world programs, it is not necessarily a bad scenario: if representativeness is guaranteed, lack of variability would suggest that the maximum observed execution time could be reasonably regarded as a precise estimator for the WCET.

**Conclusions**: in general, using EVT without passing this step cannot be considered a valid approach to estimate the WCET of real-time programs: it might be used for other classes of programs characterised, for example, by unbounded execution times (e.g. modelling the execution time distribution of the Factorial function for *any* input value). However, as already discussed, these scenarios are typically not considered as they do not produce directly "usable" WCET estimates.

We therefore conclude that EVT is *adaptable* (can be applied) to all programs that are analyzable by other timing analysis techniques. Moreover, we observed that the tail distributions of this class of programs are properly described by a Gumbel. This contrasts with the conclusions by G. Lima et al. [86] that the Gumbel family does not always provide a reliable model, resulting in low-quality WCET estimates. In fact, those conclusions were drawn outside of the WCET finiteness and existence assumptions, considering a class of programs that can only be analysed parametrically.

## 8.3 MBPTA and EVT Trustworthiness

The trustworthiness of the EVT results, when applied to the WCET analysis problem domain, is not a given. The accuracy of the model used to represent the timing behaviour of the program of interest is a major factor of influence, not only for the representativeness of observations but also for how they are collected and fed to EVT. In fact, even when representativeness is assured (which requires collecting a minimum number of runs, as we have seen in Chapters 4, 5 and 6), the way the observations are collected and submitted to analysis critically affects the quality of the EVT results. This is especially evident for the contribution of individual paths to the pWCET distribution of a multi-path program.

In the following, we show how MBPTA deals with those concerns, and discuss the role that platform-level time randomisation and time upper-bounding play in achieving representativeness. In describing an ideal MBPTA approach, we focus first on those programs whose input vectors (or all those regarded as relevant for the WCET) restrict the paths of interest to one. We then consider how the MBPTA guarantees extend to multi-path programs.

### 8.3.1 Representativeness on Single-Path (Single-Input) Programs

EVT, as a black-box technique, can produce a high-quality WCET estimate under the conditions that the analysis input (i.e. measurements from the analysis-time distribution) do exhaustively capture all factors with bearing on execution time variability, with EVT being only responsible for combining their effect. MBPTA relies on time randomisation and upper-bounding, applied to Sources of Jitter (SoJ), to guarantee that measurements capture execution conditions that are no better than those that may occur at system operation [27].

Two main steps are needed for assuring representativeness with MBPTA: (1) singling out the resources with jittery timing behaviour; (2) either randomising or upper-bounding their response time, to make the analysis-time observations representative of worst-case operation conditions.

#### (Step 1) Singling Out SoJ

SoJ can affect even the simplest of single-path programs. The nature and number of such factors are platform-dependent and have to be determined by an expert. Examples of SoJ include variable-latency floating-point unit operations, cache behaviour, or simple contention effects in a multicore. Each SoJ should be controlled in a manner that guarantees that the timing behaviour observed at analysis time captures the full extent of possible variability and, hence, is representative of system behaviour at operation.

Mastering the impact of all SoJ with MBDTA techniques is not generally viable. Understanding and explicitly triggering scenarios in which diverse hardware components may have to interact with one another is untenable in practice. Moreover, the increasing complexity of commercial off-the-shelf processors causes the cost of any classic timing analysis approach to explode and the quality of their results to decrease. A qualitative analysis of the contribution of hard-to-predict resources is thus the prerequisite to the application of *any* timing analysis technique. For MBPTA, this effort allows singling out the SoJ, to determine how randomisation and upper-bounding can capture their jitter.

#### (Step 2.a) A Case for Randomisation

The seemingly insurmountable complexity of guaranteeing representativeness of the diverse SoJ can be attained by injecting randomisation in the timing behaviour of selected hardware or software components (see Section 2.2.1). Randomisation ensures that several of the sources of execution-time variability are *transparently* captured in the analysis results without needed direct user intervention (whether

by providing specific input vectors or setting the internal state). Time-randomised resources may include placement and replacement policies in the different cache memories, as well as arbitration policies in shared resources such as interconnection networks (e.g. buses, trees) and memory controllers.

#### (Step 2.b) A Case for Upper-Bounding

Some processor operations incur different latencies depending on their operands. This is the case for floating-point operations (or even integer ones) in such embedded processor architectures as PowerPC P4080 and Cobham-Gaisler LEON3. To complement randomisation and attain representativeness, a possible solution to this problem consists in enforcing the unit to execute on a worst-latency mode at analysis time (Section 2.2.1). Specific hardware support is required to this end. Alternatively, by thoroughly logging and examining the observed latencies, the user should determine whether the jitter occurred at analysis does indeed capture the impact that it can cause during operation.

An MBPTA-compliant platform – which addresses the SoJ concerns as described above – meets the EVT statistical requirements for single-path programs. Each, conveniently collected (see [27, 78]), execution-time measurement corresponds to an independent and identically distributed (i.i.d.) observation of a random variable. Hence, if all requirements are met by construction, EVT can be applied to that problem space safely. A note of caution is in order in this regard: with that procedure, EVT is not applied to the real distribution (the full universe of values), but on a sample of it. Hence, although with MBPTA the required properties hold on the full population, they need to be empirically confirmed to hold for the specific sample, with appropriate i.i.d. tests [5, 38].

Once the exponential test and i.i.d. tests are passed, and the representativeness constraint holds, the distribution yielded by the sample can be regarded as a valid pWCET distribution.

### 8.3.2 Limits of Randomisation

It has been suggested that randomisation alone does not suffice to enable the use of MBPTA [86]. As already observed, while randomisation might not cure degenerate distributions, this is not a problem as long as the collected observations are truly representative. We insist, however, that an inattentive use of randomisation does not guarantee representativeness.

Applying time randomisation to individual processor components (e.g. the cache) makes their jitter MBPTA-compliant. However, doing that on a single resource does not cover the jitter of other resources: for example, G. Lima at al. [86] note that randomising the cache does not cover the jitter caused by the

floating-point unit or by multiple program paths. Instead, all SoJ have to be studied individually and a solution has to be proposed to address each of them. Hence, time randomising the cache does certainly not ensure per se that *all* SoJ in the platform can be captured in the measurement campaign.

Moreover, the probability of capturing random events for each SoJ is strongly related to the minimum number of runs required by MBPTA to capture execution-time variability. A *convergence criterion* has been proposed by L. Cucu-Grosjean et al. [38] for the determination of the number of observations needed to compute trustworthy bounds by characterizing the effects of time randomised caches on the execution time of a given (single-path) program. While this criterion has been shown to guarantee MBPTA convergence, it does not assure that critical but rare events (e.g. bad cache placements) have been properly captured. Solutions to detect those situations and compute the minimum number of runs required to capture all hardware events that may happen with a probability above a given threshold have been proposed in the previous Chapters of this Thesis.

### 8.3.3 Probability Distribution of Multi-Path Programs

MBPTA, as described, promises that measurements taken at analysis do capture (or upper-bound) all the possible execution conditions that may occur at operation. MBPTA only requires that every single factor with bearing on timing behaviour has been intercepted in the collected measurements, while it is up to EVT to model their combined effect. When the program of interest has multiple execution paths, the concept of representativeness extends to *whether* and *how frequently* each path is traversed by the input data provided and, thus, captured in the analysis.

#### Representativeness of Execution Paths

Assuring sufficient path coverage is a widely acknowledged requirement of all timing analysis approaches based on measurements [119, 122]. In contrast with STA, the results computed by MBTA methods hold only for the execution paths observed at analysis. Measurement-based analyses typically assume the availability of path traversal conditions information to guide the identification of a subset of structurally feasible paths to be included in the analysis.

MBPTA-compliant architectures simplify the identification of worst-case scenarios and the production of input vectors that trigger them. They do so by providing true probabilistic coverage for a number of SoJ, through randomisation and upper-bounding (e.g. cache placement, bus arbitration, input values of variable-latency units). However, path identification and generation of the respective input vectors remain a non-trivial problem for the end-users. When path traversal conditions information is not available (or is considered unreliable), it is still possible to

build a synthetic upper-bounding path [76] or use the measurements from several paths to artificially derive measurements for all unobserved paths [122].

### Modelling the Distributions of Several Paths

The critical question is whether the timing behaviour of a multi-path program can still be modelled as the outcome of a single random variable, or else each path in the program needs to be considered separately. In the former case, indistinctly feeding all measurements to EVT on account that all paths concur to a unique distribution postulates that the distribution occurred at analysis does match what will happen at operation. While there have been some works on timing analysis based on *a priori* knowledge of path frequency [40], assuming that the analysis-time observations always match the eventual distribution at operation is very hazardous. The average user is most likely unable to predict how often each input that leads to a distinct execution path will happen at operation for all programs of interest. Therefore, even if we assume that users can provide input vectors capable of triggering all execution paths of interest for WCET estimation, we cannot expect them to reason about their probability of occurrence at operation.

Instead, we observe that each single path potentially leads to a different execution time distribution. This is especially evident in programs with significant input-data dependence (and operation modes), where observations may fit significantly different distributions depending on the provided input vectors. Putting all measurements into a single bucket and applying EVT simple-mindedly on them is not advisable, as the collected measurements, while possibly independent, do not necessarily have identical distribution. Even seeking fair path frequency at analysis, for example by enforcing *uniformly distributed inputs* [89], does not solve the problem that the path distribution at operation is generally unknown and thus cannot be modelled.

Hence, one option is to study each execution path in isolation and derive an envelope distribution that safely accounts for all considered paths. We refer to this solution as *multiple-bucket* application of EVT, as opposed to its single-bucket alternative. We do not consider how these paths have been selected, whether by the user or by other means [76, 122]. In this scenario, the whole MBPTA process is applied on a per-path basis to obtain individual Complementary Cumulative Distribution Function (CCDF)s. Then a *Max-Envelope* can be computed taking the maximum value for each exceedance threshold across the pWCET distributions of all the considered paths. The resulting pWCET envelope is necessarily a discrete function as it is built from point-to-point over-approximations of all pWCET distributions. If the cost of considering each path separately is high, the user can apply path upper-bounding [76] and then analyse any of the modified paths, which is an upper-bound of all possible traversed paths.

Considering measurements from different paths in the same bucket can potentially lead to unreliable results that can equally be unsafe or more pessimistic than those obtained with the Max-Envelope method, which is safe by construction. We illustrate this on a real case study from the railway domain (Section 3.3.3), running on a LEON3-based Field-Programmable Gate Array (FPGA) implementation, with 16KB randomised Instruction and Data caches (Section 3.1). We collect observations for 10 input vectors, which correspond to 10 distinct paths in the programs. To demonstrate the inappropriateness of the single-bucket approach, we compared the results of applying MBPTA on each path in isolation against the results from the pairwise combination of different paths. The left side of Figure 8.2 shows that combining samples from paths 0 (yellow) and 3 (blue) in a single bucket causes the computed distribution (red) to fall even behind the distribution of each individual path considered in isolation. The right side of Figure 8.2 shows the combined distribution (red) obtained by merging the samples from paths 7 (yellow) and 9 (blue). In this case, the selection of the subset of data that should belong to the tail of the target distribution causes the final combined distribution to fall largely beyond the Max-Envelope values (in this case corresponding to the yellow line).

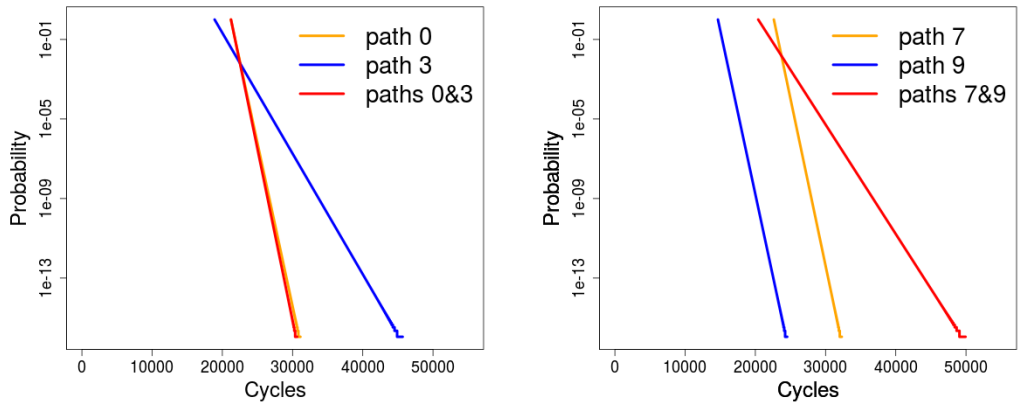



Figure 8.2: Unsafe and pessimistic results when combining paths.

## 8.4 Applying EVT on Deterministic Platforms

A discussion on the industrial viability of MBPTA must necessarily consider how EVT alone or embedded within MBPTA works on purely deterministic platforms, where no support for hardware or software randomisation is available. This matter has been considered in a number of recent studies [17, 56]. On purely deterministic platforms, the execution of a program under the same inputs and the same

initial state will take roughly the same amount of time, yielding a fully degenerate distribution. With all measurement data collected indistinctly across the paths traversed in the analysis, EVT, which is a black-box approach, unaware of the internals of the system being observed, is bound to produce results extremely sensitive to the frequency distribution of the observed path traversal. The relation that this has with the system behaviour (for path traversal) at operation can hardly be determined, as the latter is indefinable in practice.

Therefore, while applying EVT to deterministic platforms is possible in theory, we deem it very fragile for results and very difficult to control in the general case. To preserve trustworthiness and representativeness, the user is required to (i) enumerate all execution conditions with bearing on timing, (ii) generate all input vectors required to control and trigger them, and (iii) sample from these vectors in a way that matches or upper-bounds the distribution that may occur at operation.

Enumerating all combinations of execution conditions with bearing on the program execution time is an intractable problem in general. The sheer amount of possible combinations of memory placements, operands for jittery units (e.g. floating-point unit), contention for shared resources in multicores, and execution paths is overwhelming, other than in extremely limited scenarios that do not apply to most real-time industrial programs.

A second stumbling block stems from the cost and complexity of generating input vectors that trigger each and every set of identified conditions. This is further complicated by the fact that some SoJ may be very far outside of user control. This is a single point of failure for EVT, as not producing input vectors for scenarios that can occur at operation an arbitrary number of times defeats the whole point of attempting to relate pWCET estimates with actual operation conditions (i.e., representativeness) and threaten to undermine their reliability.

Finally, even in the hypothetical case we were able to enumerate and produce input vectors for all individual execution conditions, we would still have the problem of determining how often each combination of them will occur at operation with effect on each individual unit of the system. Again, this is clearly untenable.

The typical workaround, as some works have proposed (e.g. Y. Lu et al. [89]), consists in performing random sampling across a subset of input vectors (and so potentially across a subset of execution conditions). In fact, no evidence is had of the representativeness of those subsets of input vectors with respect to operation conditions. And yet, this creates an execution-time sample that will very likely pass all i.i.d. tests (except in case of very few program paths with low variability), which will be taken to allow using EVT to compute a pWCET distribution. In reality, however, each input vector would lead to a different and mostly-degenerate execution-time distribution, which means that the identical distribution require-

ment would not hold by construction. In this case, the user would be fooling EVT (or its use) and the results of it could not even be called "a distribution". Instead, before applying EVT, the user should accurately understand the population of operation-time events. This would require knowledge of the frequency of execution of each individual path, which some researchers have been able to explore [40]. However, expecting users to enumerate all combinations of execution conditions, *also considering how often each one of them would occur at operation*, and to produce the input vectors that trigger them is utterly unrealistic.

## 8.5 Summary

MBPTA builds on EVT to provide a novel approach to analyse the timing behaviour of real-time programs. Together they offer an extremely cost-effective and scalable solution when compared to standard timing analysis approaches, especially when applied to complex commercial off-the-shelf platforms. As such, MBPTA appears particularly interesting from an industrial standpoint and its industrial viability has been already demonstrated on real case studies [115, 116]. Nonetheless, an inattentive application of EVT can lead to low-quality results, eventually preventing the industry to consider MBPTA industrially viable.

In this chapter, we argued that the main misconceptions on the application of EVT to the WCET problem can be defeated by considering the peculiarities of execution time as the event EVT is expected to model. With a view to the application of EVT to industrial-quality programs, we demarcate MBPTA, a well-defined, rigorous analysis process that guarantees a WCET-centric application of EVT with guarantees on the trustworthiness of its results and wide adaptability to software programs.

# Chapter 9

# Conclusions and Future Work

## 9.1 Summary of Contributions

The need for new functionalities motivated Critical Real-Time Embedded Systems (CRTES) industry to adopt advanced hardware features used in the high-performance domain to increase average performance. These functionalities are implemented in software, which makes it grow in size and complexity. The increased complexity prompted CRTES industry to search for novel timing analysis techniques to provide evidence of timing correctness for certification. The traditional timing analysis has shown to inflate derived Worst-Case Execution Time (WCET) to the point of being unusable, or the confidence on the trustworthiness of derived estimates is not sufficient for certification.

Measurement-Based Probabilistic Timing Analysis (MBPTA) is a promising candidate to estimate WCET in modern CRTES. While pessimism is of less concern for Measurement-Based Timing Analysis (MBTA) approaches, as measurements are collected on the real platform rather than using a model, evidence needs to be delivered relating measurements collected at analysis time with the platform behaviour during operation. MBPTA simplifies building that evidence by reducing the control of resources influencing timing needed by the user and moving it to the platform itself, through features like time upper-bounding and time randomisation. In this Thesis, we have improved the applicability of MBPTA in complex CRTES by providing means to derive evidence of the trustworthiness of WCET estimates in the presence of cache memories and software programs traversing potentially different paths during operation than what is seen at analysis.

In particular, we have reached the following achievements:

- We proposed the Representativeness Validation by Simulation method (ReVS) that, assuming a known memory access pattern, computes and instructs the user on how many measurements to collect on the target platform to ac-

count for cache-related representativeness. ReVS advances the state-of-the-art Heart of Gold (HoG) method by extending the solution from programs with homogeneously accessed addresses to the general case of programs with arbitrary patterns. Our method performs a validation step in the miss count domain, applying for the first time Extreme Value Theory (EVT) to a metric other than execution time. Being based on cache simulations, the method works independently of the underlying random cache placement/replacement policy. However, this is also its weak point, as the cost of applying ReVS is prohibitive for real-size programs due to the exponential increase of potential placements to simulate.

- We proposed Conflictive Cache Placements for hash Random Placement method (CCP-hRP) and Conflictive Cache Placements for Random Modulo Placement method (CCP-RM) methods to identify CCPs for hash Random Placement (hRP) and Random Modulo Placement (RM), respectively. They are placed together under the term Conflictive Cache Placements method (for arbitrary Random Placement) (CCPX). Both methods are based on heuristics and increase the generality of ReVS by trading off its accuracy (ReVS is able to derive exact results) with the execution time cost. Our experiments with benchmarks representative of the automotive domain and a Railway case study have shown that: 1) CCPX can analyse full programs with affordable execution time; 2) CCPX successfully identifies CCPs with high enough confidence.

- All proposed methods for cache-related representativeness allow the user to derive WCET estimates at the desired confidence level. By complimenting MBPTA with our solutions, the user can obtain the trustworthiness evidence needed for the estimated WCET, and supplement it with the quantitative assessment of the coverage of relevant platform events impacting WCET. This is an important advancement with respect to traditional Measurement-Based Deterministic Timing Analysis (MBDTA) techniques, which are only able to provide a qualitative assessment of the coverage of relevant platform events, which is highly dependent on the user's expertise and ability to control those events.

- Our proposed methods for cache-related representativeness assume that relevant program paths are known and we have shown that it is unsafe to pursue a single-bucket approach for multi-paths. We proposed a PUB+CCPX method that extends the applicability of MBPTA to multi-path programs running on systems deploying cache memories. Our experiments demonstrated that the method achieves both, full path coverage and cache representativeness, with sufficient trustworthiness and minimal engagement of the user.

- We discussed misconceptions on the application of EVT to the domain of worst-case executions times and argued how its unmindful use might compromise otherwise trustworthy results. In particular, we have shown that users should not mix measurements belonging to different execution paths. We have also demonstrated that the appropriate distribution to fit data to is closely related to the nature of the event it represents. In the case of execution time, the Gumbel distribution has been shown to deem stable – yet tight – estimates.

## 9.2 Future Work

We foresee several directions in which the results of this Thesis may evolve.

1. As a short-term objective, the work of this Thesis can be extended to analyse multi-level cache hierarchies. While we analysed separately hRP (in general used for L2 caches) and RM (generally deployed in L1 caches), the interaction between different cache levels and the impact of various inclusion properties on CCPs have not been studied so far.

2. In our work we have assumed partitioned caches, which is the state-of-the-art design practice for time-critical tasks adopted by CRTES industry, as partitioning allows better control over cache interference and tighter WCET estimates [72]. However, cache sharing can alleviate the drawback of fragmentation due to partitioning, which makes it a suitable choice for soft and non-real-time tasks in emerging mixed-criticality systems [23]. The influence of sharing on CCPs should be studied in the future.

3. In this Thesis, we considered hardware solutions for time randomisation, which, although providing promising potential, cannot be applied to commercial off-the-shelf processors and typically require a longer time to be adopted to the market compared to software solutions. Therefore, we see the adaptation of methods proposed in our work to some of the existing software randomisation tenchiques [74, 79] an interesting line of research.

4. Our work considered Path Upper-Bounding (PUB) method and demonstrated how to apply it together with our cache-related representativeness solutions efficiently. Future work may investigate means to analyse multi-path programs by using the extended path coverage method and combine it with cache-related representativeness solutions. Previous work [122] has shown that extended path coverage obtains tighter WCET estimates simplifying the implementation of PUB, at the expense of providing full block rather than full path coverage.

5. The long-term goal for MBPTA is allowing timing analysis of arbitrary complex architectures. Some of the challenges toward reaching this goal are covered by other PhD students at the Universitat Politècnica de Catalunya (UPC). In addition, the work of this Thesis can be extended to consider the reliability of MBPTA on heterogeneous platforms, NoC-based platforms and platforms with accelerators (GPUs, FPGAs, etc.).

# Bibliography


[1] H.D.I. Abarbanel and Mitre Corporation. *Statistics of Extreme Events with Application to Climate*. Jason, the MITRE Corporation, 1992. 24

[2] J. Abella. *MBPTA-CV*. https://www.bsc.es/research-and-development/software-and-apps/software-list/mbpta-cv. 39

[3] J. Abella, D. Hardy, I. Puaut, E. Quiñones, and F. J. Cazorla. On the Comparison of Deterministic and Probabilistic WCET Estimation Techniques. In *26th Euromicro Conference on Real-Time Systems (ECRTS)*, July 2014. 24, 31, 113

[4] J. Abella, E. Quiñones, F. Wartel, T. Vardanega, and F. J. Cazorla. Heart of Gold: Making the Improbable Happen to Increase Confidence in MBPTA. In *26th Euromicro Conference on Real-Time Systems (ECRTS)*, July 2014. 26, 28, 45, 46, 49, 64, 72, 73, 86

[5] J. Abella, J. Del Castillo, F. J. Cazorla, and M. Padilla. Extreme Value Theory in Computer Sciences: The Case of Embedded Safety-Critical Systems. In *6th International Conference on Risk Analysis (ICRA)*, May 2015. 121

[6] J. Abella, C. Hernández, E. Quiñones, F. J. Cazorla, P. R. Conmy, M. Azkarate-askasua, J. Perez, E. Mezzetti, and T. Vardanega. WCET Analysis Methods: Pitfalls and Challenges on their Trustworthiness. In *10th IEEE International Symposium on Industrial Embedded Systems (SIES)*, June 2015. 4, 9, 15

[7] J. Abella, M. Padilla, J. Del Castillo, and F. J. Cazorla. Measurement-Based Worst-Case Execution Time Estimation Using the Coefficient of Variation. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 22(4):72:1–72:29, June 2017. xiii, 6, 9, 19, 25, 39, 50, 60, 78, 104, 117

[8] I. Agirre, M. Azkarate-askasua, C. Hernández, J. Abella, J. Perez, T. Vardanega, and F. J. Cazorla. IEC-61508 SIL 3 Compliant Pseudo-Random

Number Generators for Probabilistic Timing Analysis. In *2015 Euromicro Conference on Digital System Design (DSD)*, Aug 2015. 9

[9] I. Agirre, M. Azkarate-Askasua, A. Larrucea, J. Perez, T. Vardanega, and F.J. Cazorla. A Safety Concept for a Railway Mixed-Criticality Embedded System Based on Multicore Partitioning. In *13th IEEE International Conference on Dependable, Autonomic and Secure Computing (DASC)*, October 2015. 42

[10] I. Agirre, F. J. Cazorla, J. Abella, C. Hernández, E. Mezzetti, M. Azkarate-askasua, and T. Vardanega. Fitting Software Execution-Time Exceedance into a Residual Random Fault in ISO-26262. *IEEE Transactions on Reliability*, 67(3):1314–1327, September 2018. 9

[11] S. Altmeyer and R. I. Davis. On the Correctness, Optimality and Precision of Static Probabilistic Timing Analysis. In *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2014. 72

[12] S. Altmeyer, L. Cucu-Grosjean, and R. Davis. Static Probabilistic Timing Analysis for Real-Time Systems Using Random Replacement Caches. *Real-Time Systems*, 51(1):77–123, January 2015. 31

[13] Autotech Council. ARM Expects Vehicle Compute Performance to Increase 100x in Next Decade, 2015. https://www.autotechcouncil.com/autotech-blog/archive/arm-expects-vehicle-compute-performance-to-increase-100x-in-next-decade/. 2

[14] P. Benedicte, L. Kosmidis, E. Quiñones, J. Abella, and F. J. Cazorla. A Confidence Assessment of WCET Estimates for Software Time Randomized Caches. In *14th IEEE International Conference on Industrial Informatics (INDIN)*, July 2016. 28

[15] P. Benedicte, L. Kosmidis, E. Quiñones, J. Abella, and F. J. Cazorla. Modelling the Confidence of Timing Analysis for Time Randomised Caches. In *11th IEEE Symposium on Industrial Embedded Systems (SIES)*, May 2016. 28, 64

[16] K. Berezovskyi, L. Santinelli, K. Bletsas, and E. Tovar. WCET Measurement-Based and Extreme Value Theory Characterisation of CUDA Kernels. In *22nd International Conference on Real-Time Networks and Systems (RTNS)*, October 2014. 33

[17] K. Berezovskyi, F. Guet, L. Santinelli, K. Bletsas, and E. Tovar. Measurement-Based Probabilistic Timing Analysis for Graphics Processor

Units. In *29th International Conference on Architecture of Computing Systems (ARCS)*, April 2016. 32, 124

[18] G. Bernat, A. Colin, and S. M. Petters. WCET Analysis of Probabilistic Hard Real-Time System. In *23rd IEEE Real-Time Systems Symposium (RTSS)*, pages 279–288, December 2002. 6, 17, 32

[19] G. Bernat, A. Burns, and M. Newby. Probabilistic Timing Analysis: an Approach Using Copulas. *Journal of Embedded Computing*, 1(2):179–194, April 2005. 6

[20] J. Bin, S. Girbal, D. Gracia Pérez, A. Grasset, and A. Mérigot. Studying Co-running Avionic Real-Time Applications on Multi-core COTS Architectures. In *Embedded Real Time Software and Systems (ERTS)*, February 2014. xiii, 3

[21] A. Blin, C. Courtaud, J. Sopena, J. Lawall, and G. Muller. Maximizing Parallelism without Exploding Deadlines in a Mixed Criticality Embedded System. In *28th Euromicro Conference on Real-Time Systems (ECRTS)*, July 2016. 15

[22] S. Bünte, M. Zolda, M. Tautschnig, and R. Kirner. Improving the Confidence in Measurement-Based Timing Analysis. In *14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC)*, March 2011. 32

[23] A. Burns and R. I. Davis. A Survey of Research into Mixed Criticality Systems. *ACM Computing Surveys (CSUR)*, 50(6):82:1–82:37, November 2017. 129

[24] A. Burns, G. Bernat, and I. Broster. A Probabilistic Framework for Schedulability Analysis. In *Embedded Software (EMSOFT)*, volume 2855, pages 1–15, April 2003. 33

[25] S. Bygde, A. Ermedahl, and B. Lisper. An Efficient Algorithm for Parametric WCET Calculation. In *15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, August 2009. 115

[26] F. J. Cazorla, E. Quiñones, T. Vardanega, L. Cucu, B. Triquet, G. Bernat, E. Berger, J. Abella, F. Wartel, M. Houston, L. Santinelli, L. Kosmidis, C. Lo, and D. Maxim. PROARTIS: Probabilistically Analyzable Real-Time Systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 12 (2s):94:1–94:26, May 2013. 6

[27] F. J. Cazorla, T. Vardanega, E. Quiñones, and J. Abella. Upper-Bounding Program Execution Time with Extreme Value Theory. In *13th International Workshop on Worst-Case Execution Time Analysis (WCET)*, July 2013. 8, 24, 113, 120, 121

[28] F. J. Cazorla, L. Kosmidis, E. Mezzetti, C. Hernandez, J. Abella, and T. Vardanega. Probabilistic Worst-Case Timing Analysis: Taxonomy and Comprehensive Survey. *ACM Computing Surveys (CSUR)*, 52(1):14:1–14:35, February 2019. 6, 31

[29] CENELEC. *EN-50126:2012 - Railway Applications: the Specification and Demonstration of Dependability, Reliability, Availability, Maintainability and Safety (RAMS)*, 2012. 42

[30] Certification Authorities Software Team. Multi-core Processors - Position Paper. Technical report, CAST-32A, November 2016. 15

[31] C. Chen and G. Beltrame. An Adaptive Markov Model for the Timing Analysis of Probabilistic Caches. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 23(1):12:1–12:24, August 2017. 31

[32] C. Chen, L. Santinelli, J. Hugues, and G. Beltrame. Static Probabilistic Timing Analysis in Presence of Faults. In *11th IEEE Symposium on Industrial Embedded Systems (SIES)*, May 2016. 33

[33] Cobham Gaisler. LEON3 Processor (Probabilistic platform), 2016. URL `http://www.gaisler.com/index.php/products/processors/leon3`. 9

[34] S. Coles. *An Introduction to Statistical Modeling of Extreme Values*. Springer Series in Statistics. Springer-Verlag, 2001. 24, 25, 32, 114

[35] F. Corno, P. Prinetto, M. Rebaudengo, and M. Sonza Reorda. GATTO: a Genetic Algorithm for Automatic Test Pattern Generation for Large Synchronous Sequential Circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(8):991–1000, August 1996. 16

[36] P. Cousot and R. Cousot. Abstract Interpretation: a Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, January 1977. 4

[37] F. Cros, L. Kosmidis, F. Wartel, D. Morales, J. Abella, I. Broster, and F. J. Cazorla. Dynamic Software Randomisation: Lessons Learned from an Aerospace Case Study. In *Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2017. 113, 114

[38] L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzetti, E. Quiñones, and F. J. Cazorla. Measurement-Based Probabilistic Timing Analysis for Multi-Path Programs. In *24th Euromicro Conference on Real-Time Systems (ECRTS)*, July 2012. 6, 47, 55, 121, 122

[39] Z. I. Botev D. P. Kroese, T. Taimre. *Handbook of Monte Carlo Methods*. Wiley Series in Probability and Statistics. Wiley, 2011. 52, 71, 81

[40] L. David and I. Puaut. Static Determination of Probabilistic Execution Times. In *16th Euromicro Conference on Real-Time Systems (ECRTS)*, July 2004. 123, 126

[41] J. Del Castillo, J. Daoudi, and R. Lockhart. Methods to Distinguish Between Polynomial and Exponential Tails. *Scandinavian Journal of Statistics*, 41 (2):382–393, January 2014. 117

[42] F. Despaux, Y. Song, and A. Lahmadi. Extracting Markov Chain Models from Protocol Execution Traces for End-to-End Delay Evaluation in Wireless Sensor Networks. In *IEEE World Conference on Factory Communication Systems (WFCS)*, May 2015. 32

[43] E. Díaz, M. Fernández, L. Kosmidis, E. Mezzetti, C. Hernández, J. Abella, and F. J. Cazorla. MC2: Multicore and Cache Analysis via Deterministic and Probabilistic Jitter Bounding. In *22nd Ada-Europe International Conference on Reliable Software Technologies*, June 2017. 33

[44] J. Diebolt, M. Garrido, and S. Girard. The ET Test, a Goodness-of-Fit Test for the Distribution Tail. In *2nd International Symposium on Extreme Value Analysis*, January 2001. 117

[45] S. Edgar and A. Burns. Statistical Analysis of WCET for Scheduling. In *22nd IEEE Real-Time Systems Symposium (RTSS)*, December 2001. 6, 17

[46] P. Embrechts, T. Mikosch, and C. Klüppelberg. Modelling Extremal Events: for Insurance and Finance. *Stochastic Modelling and Applied Probability*, 33, December 1997. 24

[47] European Railway Agency (ERA). ERTMS - Set of specifications - 2 (ETCS baseline 3 and GSM-R baseline 0), 2014. URL `http://www.era.europa.eu/Core-Activities/ERTMS/Pages/Set-of-specifications-2.aspx`. 41

[48] H. Falk and H. Kotthaus. WCET-Driven Cache-Aware Code Positioning. In *14th International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES)*, October 2011. 31

[49] I. Fedotova, B. Krause, and E. Siemens. Applicability of Extreme Value Theory to the Execution Time Prediction of Programs on SoCs. In *5th International Conference on Applied Innovations in IT (ICAIIT)*, March 2017. 32

[50] W. Feller. *An Introduction to Probability Theory and its Applications*. Wiley Series in Probability and Statistics. John Willer and Sons, 1996. 8, 24

[51] M. Fernandez, D. Morales, L. Kosmidis, A. Bardizbanyan, I. Broster, C. Hernandez, E. Quinones, J. Abella, F. Cazorla, P. Machado, and L. Fossati. Probabilistic Timing Analysis on Time-Randomized Platforms for the Space Domain. In *Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2017. 8, 17, 113, 114

[52] R. A. Fisher. *The Arrangement of Field Experiments*. Breakthroughs in Statistics: Methodology and Distribution. Springer New York, 1992. 62

[53] B. V. Frías, L. Palopoli, L. Abeni, and D. Fontanelli. Probabilistic Real-Time Guarantees: There is Life Beyond the i.i.d. Assumption. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2017. 32

[54] S. Jiménez Gil, I. Bate, G. Lima, L. Santinelli, A. Gogonel, and L. Cucu-Grosjean. Open Challenges for Probabilistic Measurement-Based Worst-Case Execution Time. *IEEE Embedded Systems Letters (ESL)*, 9(3):69–72, September 2017. 10, 20

[55] D. Griffin and A. Burns. Realism in Statistical Analysis of Worst-Case Execution Times. In *10th International Workshop on Worst-Case Execution Time Analysis (WCET)*, July 2010. 17, 31, 32

[56] F. Guet, L. Santinelli, and J. Morio. On the Reliability of Probabilistic Worst-Case Execution Time Estimates. In *8th European Congress on Embedded Real Time Software and Systems (ERTS2)*, January 2016. 32, 124

[57] F. Guet, L. Santinelli, and J. Morio. On the Representativity of Execution Time Measurements: Studying Dependence and Multi-Mode Tasks. In *17th International Workshop on Worst-Case Execution Time Analysis (WCET)*, June 2017. 32

[58] J. Gustafsson, A. Betts, A. Ermedahl, and B. Lisper. The Mälardalen WCET Benchmarks - Past, Present and Future. In *10th International Workshop on Worst-Case Execution Time Analysis (WCET)*, July 2010. 41

[59] J. Hansen, S. A. Hissam, and G. Moreno. Statistical-Based WCET Estimation and Validation. In *9th International Workshop on Worst-Case Execution Time Analysis (WCET)*, July 2009. 17, 31, 47

[60] D. Hardy, I. Puaut, and Y. Sazeides. Probabilistic WCET Estimation in Presence of Hardware for Mitigating the Impact of Permanent Faults. In *Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2016. 33

[61] A. H. Hashemi, D. R. Kaeli, and B. Calder. Efficient Procedure Mapping Using Cache Line Coloring. *ACM SIGPLAN Notices*, 32(5):171–182, May 1997. 31

[62] J. L. Hennessy and D. A. Patterson. *Computer Architecture, Sixth Edition: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., 2017. 1

[63] A. Hergenhan and W. Rosenstiel. Static Timing Analysis of Embedded Software on Advanced Processor Architectures. In *Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2000. 4

[64] C. Hernández, J. Abella, F. J. Cazorla, J. Andersson, and A. Gianarro. Towards Making a LEON3 Multicore Compatible with Probabilistic Timing Analysis. In *20th Data Systems In Aerospace Conference (DASIA)*, May 2015. xiii, 36

[65] C. Hernández, J. Abella, A. Gianarro, J. Andersson, and F. J. Cazorla. Random Modulo: a New Processor Cache Design for Real-Time Critical Systems. In *53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2016. 12, 21, 82

[66] E. Heymann. The Digital Car. More Revenue, More Competition, More Cooperation. Technical report, Deutsche Bank Research, July 2017. 2

[67] M. Hollander and D.A. Wolfe. *Nonparametric Statistical Methods*. Wiley Series in Probability and Mathematical Statistics. Wiley, 1973. 62

[68] R. I. Davis, , and L. Cucu-Grosjean. A Survey of Probabilistic Schedulability Analysis Techniques for Real-Time Systems. *Leibniz Transactions on Embedded Systems (LITES)*, 6(1):04:1–04:53, May 2019. 33

[69] IEC 61508. *Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related System*, 2010. 3, 9

[70] International Organization for Standardization. *ISO/DIS 26262. Road Vehicles – Functional Safety*, 2009. 2, 3, 16, 26, 55

[71] J. Jalle, L. Kosmidis, J. Abella, E. Quiñones, and F. J. Cazorla. Bus Designs for Time-Probabilistic Multicore Processors. In *Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2014. 36

[72] T. King. Cache Partitioning Increases CPU Utilization for Safety-Critical Multicore Applications, 2013. URL http://mil-embedded.com/articles/cache-utilization-safety-critical-multicore-applications/. 129

[73] L. Kosmidis, J. Abella, E. Quiñones, and F. J. Cazorla. A Cache Design for Probabilistically Analysable Real-Time Systems. In *Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2013. 8, 11, 12, 21, 73, 76

[74] L. Kosmidis, C. Curtsinger, E. Quiñones, J. Abella, E. Berger, and F. J. Cazorla. Probabilistic Timing Analysis on Conventional Cache Designs. In *Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2013. 9, 21, 129

[75] L. Kosmidis, E. Quiñones, J. Abella, T. Vardanega, and F. J. Cazorla. Achieving Timing Composability with Measurement-Based Probabilistic Timing Analysis. In *IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC)*, June 2013. 8

[76] L. Kosmidis, J. Abella, F. Wartel, E. Quiñones, A. Colin, and F. J. Cazorla. PUB: Path Upper-Bounding for Measurement-Based Probabilistic Timing Analysis. In *26th Euromicro Conference on Real-Time Systems (ECRTS)*, July 2014. 10, 12, 29, 30, 104, 123

[77] L. Kosmidis, E. Quiñones, J. Abella, T. Vardanega, I. Broster, and F. J. Cazorla. Measurement-Based Probabilistic Timing Analysis and its Impact on Processor Architecture. In *17th Euromicro Conference on Digital System Design (DSD)*, August 2014. 8, 9, 17, 20

[78] L. Kosmidis, E. Quiñones, J. Abella, T. Vardanega, C. Hernández, A. Gianarro, I. Broster, and F. J. Cazorla. Fitting Processor Architectures for Measurement-Based Probabilistic Timing Analysis. *Microprocessors & Microsystems*, 47(PB):287–302, November 2016. 20, 31, 121

[79] L. Kosmidis, R. Vargas, D. Morales, E. Quiñones, J. Abella, and F. J. Cazorla. TASA: Toolchain-Agnostic Static Software Randomisation for Critical Real-Time Systems. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, November 2016. 9, 21, 129

[80] S. Kotz and S. Nadarajah. *Extreme Value Distributions. Theory and Applications*. World Scientific, 2000. 8, 24

[81] A. Kritikakou, C. Rochange, M. Faugère, C. Pagetti, M. Roy, S. Girbal, and D. G. Pérez. Distributed Run-Time WCET Controller for Concurrent Critical Tasks in Mixed-Critical Systems. In *22nd International Conference on Real-Time Networks and Systems (RTNS)*, October 2014. 15

[82] S. Law and I. Bate. Achieving Appropriate Test Coverage for Reliable Measurement-Based Timing Analysis. In *28th Euromicro Conference on Real-Time Systems (ECRTS)*, July 2016. 4

[83] B. Lesage, D. Griffin, F. Soboczenski, I. Bate, and R. I. Davis. A Framework for the Evaluation of Measurement-Based Timing Analyses. In *23rd International Conference on Real Time and Networks Systems (RTNS)*, November 2015. 32

[84] B. Lesage, D. Griffin, S. Altmeyer, L. Cucu-Grosjean, and R. I. Davis. On the Analysis of Random Replacement Caches Using Static Probabilistic Timing Methods for Multi-Path Programs. *Real-Time Systems*, 54(2):307–388, April 2018. 31

[85] G. Lima and I. Bate. Valid Application of EVT in Timing Analysis by Randomising Execution Time Measurements. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2017. 32

[86] G. Lima, D. Dias, and E. Barros. Extreme Value Theory for Estimating Task Execution Time Bounds: a Careful Look. In *28th Euromicro Conference on Real-Time Systems (ECRTS)*, July 2016. 29, 114, 119, 121

[87] P. Lokuciejewski, H. Falk, and P. Marwedel. WCET-Driven Cache-Based Procedure Positioning Optimizations. In *20th Euromicro Conference on Real-Time Systems (ECRTS)*, July 2008. 31

[88] Y. Lu, T. Nolte, I. Bate, and L. Cucu-Grosjean. A New Way About Using Statistical Analysis of Worst-Case Execution Times. *SIGBED Review,*, 8 (3):11–14, September 2011. 32

[89] Y. Lu, T. Nolte, I. Bate, and L. Cucu-Grosjean. A Statistical Response-Time Analysis of Real-Time Embedded Systems. In *IEEE 33rd Real-Time Systems Symposium (RTSS)*, December 2012. 123, 125

[90] C. Maxim, A. Gogonel, I. Asavoae, M. Asavoae, and L. Cucu-Grosjean. Reproducibility and Representativity: Mandatory Properties for the Compositionality of Measurement-Based WCET Estimation Approaches. *SIGBED Review*, 14(3):24–31, November 2017. 32

[91] E. Mezzetti and T. Vardanega. A Rapid Cache-Aware Procedure Positioning Optimization to Favor Incremental Development. In *IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2013. 17, 21

[92] E. Mezzetti, M. Ziccardi, T. Vardanega, J. Abella, E. Quiñones, and F. J. Cazorla. Randomized Caches can be Pretty Useful to Hard Real-Time Systems. *Leibniz Transactions on Embedded Systems*, 2(1):01–1–01:10, 2015. 28, 45, 113

[93] S. Milutinovic, E. Quiñones, J. Abella, and F. J. Cazorla. PACO: Fast Average-Performance Estimation for Time-Randomized Caches. In *52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2015. 73

[94] M. Di Natale and A. L. Sangiovanni-Vincentelli. Moving from Federated to Integrated Architectures in Automotive: the Role of Standards, Methods and Tools. *Proceedings of the IEEE*, 98(4):603–620, April 2010. 5

[95] J. Nowotsch, M. Paulitsch, D. Bühler, H. Theiling, S. Wegener, and M. Schmidt. Multi-Core Interference-Sensitive WCET Analysis Leveraging Runtime Resource Capacity Enforcement. In *26th Euromicro Conference on Real-Time Systems (ECRTS)*, July 2014. 15, 33

[96] J. Nowotsch, M. Paulitsch, A. Henrichsen, W. Pongratz, and A. Schacht. Monitoring and WCET Analysis in COTS Multi-Core-SoC-Based Mixed-Criticality Systems. In *Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2014. 33

[97] R. Obermaisser, C. El Salloum, B. Huber, and H. Kopetz. From a Federated to an Integrated Automotive Architecture. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(7):956–965, July 2009. 5

[98] S. Quinton (Organizer). Industrial Panel: Multicore Architectures in the Automotive Industry: Existing Solutions, Current Problems and Future Challenges, 2017. URL `http://2017.rtss.org/industrial-panel/`. 15

[99] K. Pettis and R. C. Hansen. Profile Guided Code Positioning. *ACM SIGPLAN Notices*, 25(6):16–27, 1990. 31

[100] J. Poovey. *Characterization of the EEMBC Benchmark Suite*. North Carolina State University, 2007. 40

[101] PROXIMA. Probabilistic Real-Time Control of Mixed-Criticality Multicore and Manycore Systems, 2013-2016. URL http://www.proxima-project.eu/. 9

[102] P. Puschner. The Single-Path Approach Towards WCET-Analysable Software. In *IEEE International Conference on Industrial Technology (ICIT)*, December 2003. 32

[103] P. Puschner, R. Kirner, and R. G. Pettit. Towards Composable Timing for Real-Time Programs. In *Software Technologies for Future Dependable Distributed Systems (STFSSD)*, March 2009. 6

[104] J. Reineke. Randomized Caches Considered Harmful in Hard Real-Time Systems. *Leibniz Transactions on Embedded Systems (LITES)*, 1(1):03–1–03:13, June 2014. 28, 45, 114

[105] SAE International. *ARP4761: Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment*, 1996. URL http://standards.sae.org/arp4761/. 3, 5, 7

[106] A. Sangiovanni-Vincentelli and M. Di Natale. Embedded System Design for Automotive Applications. *Computer*, 40(10):42 – 51, November 2007. 5

[107] L. Santinelli, J. Morio, G. Dufour, and D. Jacquemart. On the Sustainability of the Extreme Value Theory for WCET Estimation. In *14th International Workshop on Worst-Case Execution Time Analysis (WCET)*, July 2014. 32, 113, 114

[108] L. Santinelli, F. Guet, and J. Morio. Revising Measurement-Based Probabilistic Timing Analysis. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2017. 10, 32

[109] M. H. Schulz, E. Trischler, and T. M. Sarfert. SOCRATES: a Highly Efficient Automatic Test Pattern Generation System. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 7(1):126–137, January 1988. 16

[110] K. P. Silva, L. F. Arcaro, and R. S. d. Oliveira. On Using GEV or Gumbel Models When Applying EVT for Probabilistic WCET Estimation. In *IEEE Real-Time Systems Symposium (RTSS)*, December 2017. 25, 104

[111] M. Slijepcevic, L. Kosmidis, J. Abella, E. Quiñones, and F. J. Cazorla. DTM: Degraded Test Mode for Fault-Aware Probabilistic Timing Analysis. In *25th Euromicro Conference on Real-Time Systems (ECRTS)*, July 2013. 33

[112] SoCLib, 2003-2012. URL http://www.soclib.fr/trac/dev. 37

[113] Z. Stephenson, J. Abella, and T. Vardanega. Supporting Industrial Use of Probabilistic Timing Analysis with Explicit Argumentation. In *11th IEEE International Conference on Industrial Informatics (INDIN)*, July 2013. 9, 17

[114] C. Wagner. The Future of Mobility. In *Technology Day Paris*, June 2017. 5

[115] F. Wartel, L. Kosmidis, C. Lo, B. Triquet, E. Quiñones, J. Abella, A. Gogonel, A. Baldovin, E. Mezzetti, L. Cucu, T. Vardanega, and F. J. Cazorla. Measurement-Based Probabilistic Timing Analysis: Lessons from an Integrated-Modular Avionics Case Study. In *8th IEEE International Symposium on Industrial Embedded Systems (SIES)*, June 2013. 113, 114, 126

[116] F. Wartel, L. Kosmidis, A. Gogonel, A. Baldovino, Z. Stephenson, B. Triquet, E. Quiñones, C. Lo, E. Mezzetta, I. Broster, J. Abella, L. Cucu-Grosjean, T. Vardanega, and F. J. Cazorla. Timing Analysis of an Avionics Case Study on Complex Hardware/Software Platforms. In *Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2015. 8, 17, 32, 126

[117] I. Wenzel, B. Rieder, R. Kirner, and P. Puschner. Automatic Timing Model Generation by CFG Partitioning and Model Checking. In *Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2005. 32

[118] I. Wenzel, R. Kirner, B. Rieder, and P. Puschner. Measurement-Based Timing Analysis. In *3rd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISOLA)*, October 2008. 4

[119] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström. The Worst-Case Execution-Time Problem - Overview of Methods and Survey of Tools. *ACM Transactions on Embedded Computing Systems (TECS)*, 7(3):36:1–36:53, May 2008. 4, 10, 15, 31, 122

[120] R. Wilhelm, S. Altmeyer, C. Burguière, D. Grund, J. Herter, J. Reineke, B. Wachter, and S. Wilhelm. Static Timing Analysis for Hard Real-Time

Systems. In *11th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI)*, January 2010. 4

[121] J. Windsor, M. H. Deredempt, and R. De-Ferluc. Integrated Modular Avionics for Spacecraft - User Requirements, Architecture and Role Definition. In *IEEE/AIAA 30th Digital Avionics Systems Conference (DASC)*, October 2011. 5

[122] M. Ziccardi, E. Mezzetti, T. Vardanega, J. Abella, and F. J. Cazorla. EPC: Extended Path Coverage for Measurement-Based Probabilistic Timing Analysis. In *IEEE Real-Time Systems Symposium (RTSS)*, December 2015. 10, 12, 29, 122, 123, 129