

This is a repository copy of *A Survey of Probabilistic Timing Analysis Techniques for Real-Time Systems*.

White Rose Research Online URL for this paper:

<https://eprints.whiterose.ac.uk/id/eprint/162338/>

Version: Published Version

---

**Article:**

Davis, Robert Ian [orcid.org/0000-0002-5772-0928](https://orcid.org/0000-0002-5772-0928) and Cucu-Grosjean, Liliana (2019) A Survey of Probabilistic Timing Analysis Techniques for Real-Time Systems. LITES: Leibniz Transactions on Embedded Systems. pp. 1-60. ISSN: 2199-2002

<https://doi.org/10.4230/LITES-v006-i001-a003>

---


**Reuse**

Other licence.

**Takedown**

If you consider content in White Rose Research Online to be in breach of UK law, please notify us by emailing [eprints@whiterose.ac.uk](mailto:eprints@whiterose.ac.uk) including the URL of the record and the reason for the withdrawal request.

# A Survey of Probabilistic Timing Analysis Techniques for Real-Time Systems

Robert I. Davis 

University of York, UK and Inria, France  
rob.davis@york.ac.uk

Liliana Cucu-Grosjean

Inria, France  
liliana.cucu@inria.fr

---

## Abstract

This survey covers probabilistic timing analysis techniques for real-time systems. It reviews and critiques the key results in the field from its origins in 2000 to the latest research published up to the end of August 2018. The survey provides a taxonomy of the different methods used, and a classification of existing research. A detailed review is provided covering the main subject areas: static

probabilistic timing analysis, measurement-based probabilistic timing analysis, and hybrid methods. In addition, research on supporting mechanisms and techniques, case studies, and evaluations is also reviewed. The survey concludes by identifying open issues, key challenges and possible directions for future research.

**2012 ACM Subject Classification** Software and its engineering → Software organization and properties, Software and its engineering → Software functional properties, Software and its engineering → Real-time schedulability, Computer systems organization → Real-time systems

**Keywords and Phrases** Probabilistic, real-time, timing analysis

**Digital Object Identifier** 10.4230/LITES-v006-i001-a003

**Received** 2018-01-04 **Accepted** 2019-02-26 **Published** 2019-05-14

## 1 Introduction

Systems are characterised as *real-time* if, as well as meeting functional requirements, they are required to meet timing requirements. Real-time systems may be further classified as *hard* real-time, where failure to meet their timing requirements constitutes a failure of the system; or *soft* real-time, where such failure leads only to a degraded quality of service. Today, both hard and soft real-time systems are found in many diverse application areas including; automotive, aerospace, medical systems, robotics, and consumer electronics.

Real-time systems are typically implemented via a set of programs, also referred to as tasks, which are executed on a recurring basis. The programs used in a real-time system have a *functional* behaviour, and also a *timing* behaviour. The functional behaviour of a given run of a program depends on the *input state*, which comprises a set of values for the input variables, and a set of values for the software state variables (which are related to the values of the input variables used on previous runs). The input state affects the path taken through the code, and the values of the outputs produced. Typically programs have a functional behaviour which is *deterministic*, in other words, given the exact same inputs they will produce the exact same outputs. Functional behaviour may also be *non-deterministic*, for example in a randomised search the same input state may lead to different outputs depending on the behaviour of a random number generator. In this survey we are mainly concerned with the timing behaviour of programs that have deterministic functionality.



© Robert I. Davis and Liliana Cucu-Grosjean;  
licensed under Creative Commons Attribution 3.0 Germany (CC BY 3.0 DE)  
Leibniz Transactions on Embedded Systems, Vol. 6, Issue 1, Article No. 3, pp. 03:1–03:60



Leibniz Transactions on Embedded Systems  
LITES Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In keeping with the majority of the work on program timing behaviour, in the following we consider programs that are run without interruption or preemption and without any interference from other programs that could be running on the same or different processor cores (i.e. no multi-threading and no cross-core interference). We return to this point in the conclusions.

The timing behaviour of a program with deterministic functionality depends on both the input state, and the initial values of hardware state variables, referred to as the *hardware state*. Examples of hardware state variables include the contents of internal buffers, pipelines, caches, scratchpads, and certain register values. While the hardware state may affect the timing behaviour of the program, it has no effect on the functional behaviour. A hardware platform is referred to as *time-predictable* if it always takes the same amount of time to execute a deterministic program when starting from the same input state and the same hardware state. By contrast, a *time-randomised* hardware platform may take a variable amount of time to execute such a program when starting from the same input state and hardware state, due to the behaviour of underlying random elements in the hardware<sup>1</sup>.

## 1.1 Conventional Timing Analysis Techniques

Understanding the timing behaviour of each program is fundamental to verifying the timing requirements of a real-time system. Key to this is *timing analysis*, which seeks to characterise the amount of time that each program can take to execute on the given hardware platform. Typically, this is done by upper bounding or estimating the *Worst-Case Execution Time*.

► **Definition 1.** The *Worst-Case Execution Time (WCET)* of a program is an upper bound on the execution time of that program for any valid input state and initial hardware state (i.e. the WCET is an upper bound on the execution time for any single run of the program, and there is at least one run of the program that can realise the WCET).

The methods traditionally used for timing analysis can be classified into three main categories:

- *Static Analysis:* These methods do not execute the program on the actual hardware or on a simulator. Rather they analyse the code for the program and some annotations (providing information about input values), along with an abstract model of the hardware. Typically static analysis proceeds in three steps. First, control flow analysis is used to derive constraints on feasible paths, including loop bounds. Second, micro-architectural analysis is used to provide an over-approximation of the program execution on the feasible paths, accounting for the behaviour of hardware features such as pipelines and caches. Third, path analysis uses integer linear programming (ILP) to combine the results of control flow analysis and micro-architectural analysis, and so derive an upper bound on the WCET of the program.

To derive an upper bound on the WCET static analysis has to determine properties relating to the dynamic behaviour of the program without actually executing it. In practice, it may not be possible to precisely determine all of these properties due to issues of tractability and decidability. For example determining the precise cache contents at a given program point may not be possible when there is a dependency on the input values. Properties which cannot be precisely determined must be conservatively approximated to ensure that the computed WCET remains a valid upper bound; however, such approximations may lead to significant pessimism. For advanced hardware platforms, there are two main challenges for static analysis methods. Firstly, obtaining and validating all of the information necessary to build an accurate

---

<sup>1</sup> Note here the basic random elements in the hardware (e.g. a random number generator) are not considered to be part of the hardware state.

model of the hardware components that impact program execution times. Secondly, modelling those components and their interactions without substantial loss of precision in the derived WCET upper bound.

- *Dynamic or Measurement-Based Analysis:* These methods derive an estimate of the WCET by running the program on the actual hardware or on a cycle-accurate timing simulator. A measurement protocol provides test vectors (sets of input values) and initial hardware configurations that are used to exercise a subset of the possible paths through the code, as well as the possible hardware states that may affect the timing behaviour<sup>2</sup>. The execution times for multiple runs of the program are collected and the maximum observed execution time recorded. This value may be used as a (lower bound) estimate of the WCET, or alternatively, an engineering margin (e.g. 20%) may be added to give an estimate of the WCET. This margin comes from industrial practice and engineering judgement [129]; however, there is no guarantee that it results in an upper bound on the actual WCET. For complex programs and advanced hardware platforms, the two main challenges for measurement-based analysis methods both involve designing an appropriate measurement protocol. Firstly, if it were known which values for the input variables and software state variables would lead to the WCET, then the measurement protocol could ensure that those values were present in the test vectors used; however, typically these values are not known and cannot easily be derived. Secondly, it may not be known, or easy to derive, which initial hardware states will lead to the WCET; it may also be difficult to force the hardware into a particular initial state. Nevertheless, measurement-based analysis is commonly used in industry and may give engineers a useful perspective on the timing behaviour of a program.
- *Hybrid Analysis:* These methods combine elements of both static analysis and measurement-based analysis. For example, a hybrid approach may record the maximum observed execution time for short sub-paths through the code, and then combine these values using information obtained via static analysis of the program's structure (e.g. the control flow graph) to estimate the WCET. The aim of hybrid analysis is to overcome the disadvantages of both static and dynamic methods. By measuring execution times for short sub-paths through the code, hybrid methods avoid the need for a model of the hardware, which may be difficult to obtain and to validate. By using static analysis techniques to determine the control flow graph, the problem of having to find input values that exercise the worst-case path is ameliorated. Instead, the measurement protocol can focus on ensuring that measurements are obtained for all sub-paths, i.e. structural coverage, which is far simpler to achieve than full path coverage. Nevertheless, hybrid methods still inherit many of the challenges of measurement-based methods. On advanced hardware (e.g. with pipelines and caches) the execution time of a sub-path may be dependent on the execution history, and hence on the previous sub-paths that were executed. This exacerbates the problem of composing the overall WCET estimate from observations for individual sub-paths, and may degrade the precision of that estimate. In general, today's hybrid methods cannot guarantee to upper bound the WCET; however, the estimates they produce may be more accurate than those based on measurements alone.

During the past two decades, the hardware platforms used, or proposed for use, in real-time systems have become increasingly more complex. Architectures include advanced hardware acceleration features such as pipelines, branch prediction, out-of-order execution, caches, write-buffers, scratchpads, and multiple levels of memory hierarchy. These advances, along with increasing software complexity, greatly exacerbate the timing analysis problem. Most acceleration

---

<sup>2</sup> Exercising all possible paths and initial hardware states is often impractical.

features are designed to optimise average-case rather than worst-case behaviour and can result in significant variability in execution times. This is making it increasingly difficult, if not impossible, to obtain tight WCET estimates<sup>3</sup> from conventional static timing analysis methods that seek to provide an upper bound on the WCET. Further, increases in software and hardware complexity make it difficult to design measurement protocols capable of ensuring that the worst-case path(s) through the code are exercised, and that the worst-case hardware states are encountered when using measurement-based and hybrid analyses. (Appendix A provides further discussion of measurement protocols).

## 1.2 Probabilistic Timing Analysis Techniques

*Probabilistic timing analysis*<sup>4</sup> differs from traditional approaches in that it lifts the characterisation of the timing behaviour of a program from the consideration of a single run, to the consideration of a repeating sequence of many runs, referred to as a *scenario*, and hence lifts the results from a scalar value (the WCET) to a probability distribution (the pWCET distribution, defined in Section 2.1). Traditional timing analysis methods aim to tightly upper bound the execution time that could occur for a single run of a program out of all possible runs. Similarly, probabilistic timing analysis methods aim to tightly upper bound the distribution of execution times that could potentially occur for some scenario of operation, out of all possible scenarios of operation.

Research into probabilistic timing analysis can be classified into five main categories. This classification forms the basis for the main sections of this survey. Note, for ease of reference we have numbered these categories below, starting at 3, to match the section of survey.

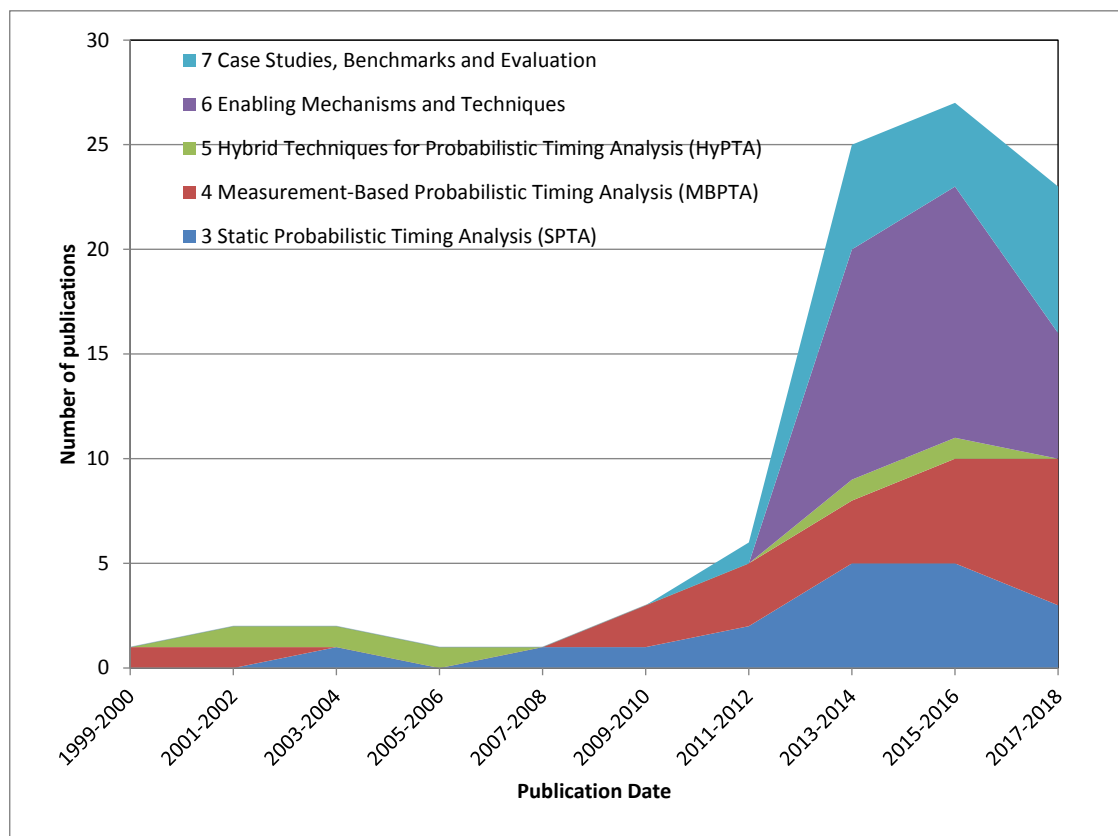
3. *Static Probabilistic Timing Analysis (SPTA)*: Similar to traditional static analyses, SPTA methods do not execute the program on the actual hardware or on a simulator. Instead they analyse the code for the program and information about input values, along with an abstract model of the hardware behaviour. The difference is that SPTA methods account for some form of random behaviour in either the hardware, the software, or the environment (i.e. the inputs) by using probability distributions, and therefore construct an upper bound on the pWCET distribution rather than a upper bound on the WCET. In common with traditional static analyses, SPTA methods have to determine properties relating to the dynamic behaviour of the program without actually executing it. Here, the conservative approximation of properties which cannot be precisely determined (e.g. cache states in a random replacement cache) may lead to significant pessimism in the estimated pWCET distribution. The two main challenges for SPTA methods are obtaining and validating the information necessary to build an accurate model of the hardware components; and modelling those components and their interactions without substantial loss of precision in the upper bound pWCET distribution derived.
4. *Measurement-Based Probabilistic Timing Analysis (MBPTA)*: Today, most of the current MBPTA methods use Extreme Value Theory (EVT) to make a *statistical estimate* of the pWCET distribution of a program. This estimate is based on a sample of execution time observations obtained by executing the program on the hardware or a cycle accurate simulator according to a *measurement protocol*. The measurement protocol samples some scenario(s) of operation, i.e. it executes the program multiple times according to a set of feasible input states and initial hardware states. As with traditional measurement-based analysis methods, the

<sup>3</sup> By a tight WCET estimate we mean one that is relatively close to the actual WCET, for example perhaps no more than 10-20% larger.

<sup>4</sup> In this survey, we adopt the widely used term “probabilistic timing analysis” noting that it can easily be misinterpreted. To clarify, while the results produced are expressed in terms of probability distributions, the analysis methods themselves are deterministic in the sense of always producing the same results from the same inputs, unlike for example randomised search techniques.

main challenge for MBPTA methods involves designing an appropriate measurement protocol. In particular, in order for the estimated pWCET distribution derived by EVT to be valid for a future scenario of operation, then the sample of input states and hardware states used for analysis must be *representative* of those that will occur during that future scenario of operation. An important issue here is that there may not be a single sample of input states and hardware states that is representative of all possible future scenarios of operation.

5. *Hybrid Probabilistic Timing Analysis (HyPTA)*: These methods combine in some way both statistical and analytical approaches. For example by taking measurements at the level of basic blocks or sub-paths, and then composing the results (i.e. the estimated pWCET distributions for the sub-paths) using structural information obtained from static analysis of the code.
6. *Enabling mechanisms*: These mechanisms aim to facilitate the use of one or other of the above analysis methods.
7. *Evaluation*: Case studies, benchmarks, and metrics, which aim to evaluate the efficiency, effectiveness, and applicability of probabilistic timing analysis methods.



**Figure 1** Intensity of research in the different categories corresponding to Sections 3 to 7 of this survey.

The research in these categories is summarised by authors and citations in Table 1. Note the sub-categories correspond to the subsections of this survey.

It is interesting to note how research in the different categories has progressed over time. Figure 1 illustrates the number of papers reviewed in each of the main categories covered by this survey that have been published during 2-year time intervals from 1999 to 2018. (This figure is best viewed online in colour). A number of observations can be drawn from Figure 1. Firstly, the volume of research into probabilistic timing analysis was relatively flat until around 2008

■ **Table 1** Summary of publications from different authors in the categories described in the main sections and subsections of this survey.

<b>3 Static Probabilistic Timing Analysis (SPTA)</b>
3.1 SPTA based on Probabilities from Inputs David and Puaut [35], Liang and Mitra [86]
3.2 SPTA based on Probabilities from Faults Hofig [62], Hardy and Puaut [58, 59], Chen et al. [31, 29]
3.3 SPTA based on Probabilities from Random Replacement Caches Quinones et al. [106], Burns and Griffin [23], Cazorla et al. [26], Davis et al. [39], Altmeyer and Davis [7], Altmeyer et al. [6], Griffin et al. [52], Lesage et al. [83, 82], Chen and Beltrame [28], Chen et al. [30]
<b>4 Measurement-Based Probabilistic Timing Analysis (MBPTA)</b>
4.1 EVT and i.i.d. observations Burns and Edgar [22], Edgar and Burns [45], Hansen et al. [57], Griffin and Burns [51], Lu et al. [89, 90], Cucu-Grosjean et al. [34]
4.2 EVT and observations with dependences Melani et al. [93], Santinelli et al. [112], Berezovskyi et al. [16, 15], Guet et al. [53, 54], Fedotova et al. [47], Lima and Bate [87]
4.3 EVT and representativity Lima et al. [88], Maxim et al. [92], Santinelli et al. [110], Santinelli and Guo [111], Guet et al. [55], Abella et al. [3], Milutinovic et al. [101]
<b>5 Hybrid Techniques for Probabilistic Timing Analysis (HyPTA)</b>
5.1 HyPTA and the Path Coverage Problem Bernat et al. [19, 18, 17], Kosmidis et al. [70], Ziccardi et al. [136]
<b>6 Enabling Mechanisms and Techniques</b>
6.1 Caches and Hardware Random Placement Kosmidis et al. [68, 69, 67], Slijepcevic et al. [120], Anwar et al. [9], Hernandez et al. [61], Trillia et al. [127], Benedicte et al. [11]
6.2 Caches and Software Random Placement Kosmidis et al. [72, 73, 71, 78]
6.3 Cache Risk Patterns with Random Placement Abella et al. [4], Benedicte et al. [13, 14], Milutinovic et al. [98, 99, 97, 100]
6.4 Buffers, Buses and other Resources Slijepcevic et al. [119], Cazorla et al. [27], Kosmidis et al. [74], Jalle et al. [65], Panic et al. [102], Agirre et al. [5], Hernandez et al. [60], Slijepcevic et al. [117, 118], Benedicte et al. [12]
<b>7 Case Studies, Benchmarks and Evaluation</b>
7.1 Critiques Reineke [108], Mezzetti et al. [96], Stephenson et al. [123], Gil et al. [49]
7.2 Case Studies and Evaluation Santos et al. [113], Kosmidis et al. [75, 77], Abella et al. [2], Wartel et al. [129, 128], Lesage et al. [85], Mezzetti et al. [94], Fernandez et al. [48], Cros et al. [33] Diaz et al. [43], Silva et al. [116], Reghenzani et al. [107]

and then increased rapidly in the decade from 2010. Inline with this the number of publications on the main theme of measurement-based probabilistic timing analysis (Section 4) has steadily increased since 2009/2010. Work on static probabilistic timing analysis (Section 3) peaked during the period 2013–2016, coinciding with research effort on the EU Proxima project<sup>5</sup>. Also, as a result of that project, there was a significant peak in research on enabling techniques aimed at facilitating the use of MBPTA (Section 6). More recently, in 2017/2018, the focus has been on extending MBPTA to more complex systems and exploring the effectiveness of the approach via case studies and benchmarks (Section 7).

Before moving to the sections of this survey which review the literature, we first discuss (in Section 2) fundamental concepts and methods relating to probabilistic timing analysis.

Note that conventional timing analysis techniques aimed at providing an upper bound on the WCET value as the solution to the timing analysis problem are outside of the scope of this survey, they are reviewed in detail by Wilhelm et al. [133].

## 2 Fundamental Concepts and Methods

The term *probabilistic real-time systems* is used to refer to real-time systems where one or more of the parameters, such as program execution times, are modelled by random variables. Although a parameter is described (i.e. *modelled*) by a random variable, this does not necessarily mean that the actual parameter itself exhibits random behaviour or that there is necessarily any underlying random element to the system that determines its behaviour. The actual behaviour of the parameter may depend on complex and unknown or uncertain behaviours of the overall system. As an example, the outcome of a coin toss can be modelled as a random variable with heads and tails each having a probability of 0.5 of occurring, assuming that the coin is fair. However, the actually process of tossing a coin does not actually have a random element to it. The outcome could in theory be predicted to a high degree of accuracy if there were sufficiently precise information available about the initial state and the complex behaviour and evolution of the overall physical processes involved. There are however many useful results that can be obtained by modelling the outcome of a coin toss as a random variable. The same is true of the analysis of probabilistic real-time systems.

In this section, we first discuss a fundamental and often misunderstood concept in probabilistic timing analysis, *probabilistic Worst-Case Execution Time (pWCET)* distributions. The remainder of the section then provides an outline of the two main approaches to obtaining pWCET distributions, *Static Probabilistic Timing Analysis (SPTA)* and *Measurement-Based Probabilistic Timing Analysis (MBPTA)*.

### 2.1 probabilistic Worst-Case Execution Time (pWCET)

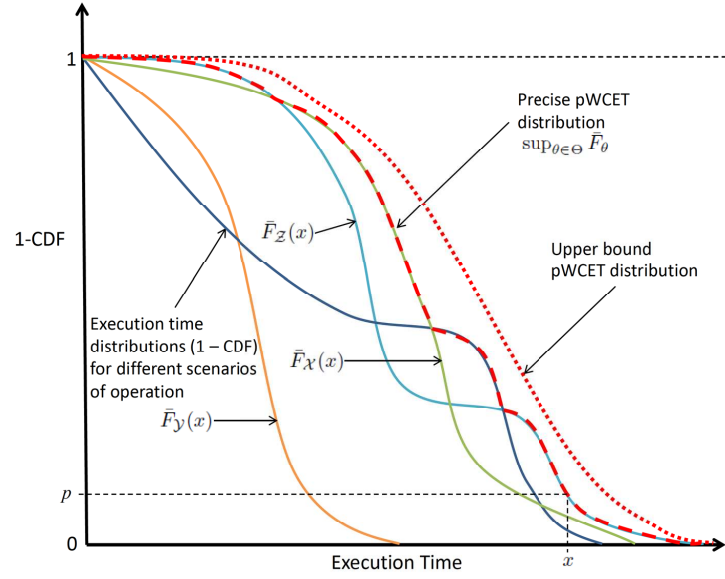
The term *probabilistic Worst-Case Execution Time (pWCET)* distribution has been used widely in the literature, with a number of different definitions given. Below, we provide an overarching definition of the term. (We use calligraphic characters, such as  $\mathcal{X}$ , to denote random variables).

► **Definition 2.** The *probabilistic Worst-Case Execution Time (pWCET)* distribution for a program is the least upper bound, in the sense of the greater than or equal to operator  $\succeq$  (defined below), on the execution time distribution of the program for every valid scenario of operation, where a *scenario* of operation is defined as an infinitely repeating sequence of input states and initial hardware states that characterise a feasible way in which recurrent execution of the program may occur.

<sup>5</sup> <http://www.proxima-project.eu/>

► **Definition 3.** (From Diaz et al. [44]) The probability distribution of a random variable  $\mathcal{X}$  is *greater than or equal to* (i.e. upper bounds) that of another random variable  $\mathcal{Y}$  (denoted by  $\mathcal{X} \succeq \mathcal{Y}$ ) if the Cumulative Distribution Function (CDF) of  $\mathcal{X}$  is never above that of  $\mathcal{Y}$ , or alternatively, the 1-CDF of  $\mathcal{X}$  is never below that of  $\mathcal{Y}$ .

Graphically, Definition 2 means that the 1 - CDF of the pWCET distribution is never below that of the execution time distribution for any scenario of operation. Hence the 1 - CDF or *exceedance function* of the pWCET distribution may be used to determine an upper bound on the probability  $p$  that the execution time of a randomly selected run of the program will exceed an execution time budget  $x$ , for any chosen value of  $x$ . This upper bound is valid for any feasible scenario of operation.



■ **Figure 2** Exceedance function or 1-CDF for the pWCET distribution of a program, and also execution time distributions for specific scenarios of operation.

Figure 2 illustrates the execution time distributions of a number of different scenarios of operation (solid lines), the precise pWCET distribution (red dashed line) which is the least upper bound (i.e. the point-wise maxima of the 1 - CDF) for all of these distributions, and also some arbitrary upper bound pWCET distribution (red dotted line) which is a pessimistic estimate of the precise pWCET. Also shown (on the y-axis) is an upper bound  $p$  on the probability that any randomly selected run of the program will have an execution time that exceeds  $x$  (on the x-axis). The value  $x$  is referred to as the pWCET estimate at a probability of exceedance of  $p$ . (More formally, the least upper bound pWCET distribution is given by  $\sup_{\theta \in \Theta} \bar{F}_{\theta}$ , where  $\bar{F}_{\theta}$  is the 1 - CDF for scenario of operation  $\theta$ , and  $\Theta$  is the space of all valid scenarios of operation).

Note that the greater than or equal to relation  $\succeq$  between two random variables does not provide a total order, i.e. for two random variables  $\mathcal{X}$  and  $\mathcal{Z}$  it is possible that  $\mathcal{X} \not\succeq \mathcal{Z}$  and  $\mathcal{Z} \not\succeq \mathcal{X}$ . Hence the precise pWCET distribution may not correspond to the execution time distribution for any specific scenario. This can be seen in Figure 2, considering the execution time distributions  $\mathcal{X}$ ,  $\mathcal{Y}$  and  $\mathcal{Z}$ . It is the case that  $\mathcal{X} \succeq \mathcal{Y}$ , but  $\mathcal{X} \not\succeq \mathcal{Z}$  and  $\mathcal{Z} \not\succeq \mathcal{X}$ . By contrast, as the greater than or equal to relation for scalars ( $\geq$ ) does provide a total order, the precise WCET does correspond to the execution time for some specific run of the program.

The WCET upper bounds all possible execution times for a program, independent of any particular run of the program. Similarly, the pWCET distribution upper bounds all possible execution time distributions for a program, independent of any particular scenario of operation.

We note that the term pWCET is open to misinterpretation and is often misunderstood. To clarify, it does *not* refer to the probability distribution of the worst-case execution time, since the WCET is a single value. Rather informally, following Definition 2, the pWCET may be thought of as the “worst-case” (in the sense of upper bound) probability distribution of the execution time for any scenario of operation.

Below, we provide some simple hypothetical examples that illustrate the meaning of the pWCET distribution.

Consider a program  $A$  running on time-randomised hardware. Further, assume that the program has two paths which may be selected based on the value of an input variable. The discrete probability distributions  $\mathcal{X}$  and  $\mathcal{Y}$  of the execution time of each path may be described by *probability mass functions* as follows:

$$\mathcal{X} = \begin{pmatrix} 10 & 20 & 30 & 40 \\ 0.4 & 0.3 & 0.2 & 0.1 \end{pmatrix} \quad \mathcal{Y} = \begin{pmatrix} 20 & 30 & 40 & 50 \\ 0.8 & 0.15 & 0.04 & 0.01 \end{pmatrix}$$

Indicating, among other things, that there is a probability of 0.1 that the execution time of the first path is 40 and a probability of 0.01 that the execution time of the second path is 50.

The *Complementary Cumulative Distribution Function* (1-CDF) or *Exceedance Function* defined as  $\bar{F}_{\mathcal{X}}(x) = P(\mathcal{X} > x)$  is given by:

$$\bar{F}_{\mathcal{X}} = \begin{pmatrix} 0 & 10 & 20 & 30 & 40 \\ 1 & 0.6 & 0.3 & 0.1 & 0 \end{pmatrix} \quad \bar{F}_{\mathcal{Y}} = \begin{pmatrix} 0 & 10 & 20 & 30 & 40 & 50 \\ 1 & 1 & 0.2 & 0.05 & 0.01 & 0 \end{pmatrix}$$

The precise pWCET distribution for the two paths can be found by taking the point-wise maxima of their exceedance functions. Hence:

$$pWCET = \begin{pmatrix} 0 & 10 & 20 & 30 & 40 & 50 \\ 1 & 1 & 0.3 & 0.1 & 0.01 & 0 \end{pmatrix}$$

Note that the above pWCET distribution is precise on the assumption that repeatedly executing one path or the other or any combination of them is a valid scenario of operation, otherwise it may be that it is an upper bound rather than the precise pWCET.

Observe that the program has a WCET of 50, which equates to the last value in the pWCET distribution (where the 1-CDF becomes zero). The pWCET distribution gives more nuanced information than this single value, as it upper bounds the probability of occurrence of the extreme execution time values (e.g. 30, 40, and 50) that occur rarely and form the tail of the distribution.

Execution on time-randomised hardware is not the only way in which a non-degenerate<sup>6</sup> pWCET distribution can arise. As an alternative, consider another program  $B$  that implements a software state machine with four states and hence four paths that runs on time-predictable hardware. Here the main factor which affects the execution time is the path taken, which is determined by the value of the software state variable. For this program, all valid scenarios of operation involve the software state variable cycling through its four possible values in order, and hence the four possible paths executing in order on any four consecutive runs of the program. Further, assume that there is a small variability in the execution time of each path depending on the value of some input variable, which may take any value on any run. Hence the execution

<sup>6</sup> A degenerate distribution has only a single value.

times of the different paths are  $10 \pm 2$ ,  $20 \pm 2$ ,  $30 \pm 2$ , and  $40 \pm 2$ , each with a probability of 0.25. For this program, the pWCET distribution valid for any scenario of operation is:

$$pWCET = \begin{pmatrix} 0 & 12 & 22 & 32 & 42 \\ 1 & 0.75 & 0.5 & 0.25 & 0 \end{pmatrix}$$

Finally, consider a program  $C$  which again runs on time-predictable hardware. This program has an input variable  $v$  which may take one of four values selecting one of four different paths. The execution times of the paths are 10, 20, 30, and 40. Further, we are given some additional information about all the valid scenarios of operation, over a large number of runs of the program, the first 3 input values occur with the same probability, while the 4th is a fault condition that occurs at most 1% of the time. The pWCET distribution is as follows:

$$pWCET = \begin{pmatrix} 0 & 10 & 20 & 30 & 40 \\ 1 & 0.67 & 0.34 & 0.01 & 0 \end{pmatrix}$$

We note that in all three examples, the pWCET distribution upper bounds the execution time distribution for a randomly selected run of the program in any scenario of operation. However, it is not always the case that the pWCET distribution is *probabilistically independent* of the value realised for the execution time of previous runs of the program. For example in the case of program  $B$ , the pWCET distribution does not provide a valid upper bound on the execution time distribution of the program *conditional* on specific execution times having occurred for previous runs. (If the previous execution time of the program was 30, then the execution time of the current run has a probability of 1 of exceeding 32, since the state variable will increment by 1 and the longest path will be selected with an execution time of  $40 \pm 2$ ). Further, in the case of program  $C$ , although the fault condition may occur only 1% of the time, there may well be a cluster of faults. Hence for programs  $B$  and  $C$ , it is not valid to compose the pWCET distributions using basic convolution to obtain a bound on the interference (total execution time) of two or more runs of the program. This has implications for how the pWCET distribution may be used in probabilistic schedulability analysis, which are discussed in the companion survey on that topic [38].

► **Definition 4.** Two random variables  $\mathcal{X}$  and  $\mathcal{Y}$  are *probabilistically independent* if they describe two events such that knowledge of whether one event did or did not occur does not change the probability that the other event occurs. Stated otherwise, the joint probability is equal to the product of their probabilities  $P(\{\mathcal{X} = x\} \cap \{\mathcal{Y} = y\}) = P(\mathcal{X} = x) \cdot P(\mathcal{Y} = y)$ . (In this context, the events are the execution times of runs of the program taking certain values).

We note that while the above simple examples are useful to illustrate the concept of a pWCET distribution, in practice the exceedance probabilities of interest are very small, typically in the range  $10^{-4}$  to  $10^{-15}$ . These probabilities derive from the acceptable failure rate per hour of operation for the application considered. (Note, the relationship between failure rates per hour of operation and probabilities of timing failure depend on various factors considered in fault tree analysis, including any mitigations and recovery mechanisms that may be applied in the event of a timing failure [50], see the companion survey [38] for further discussion).

It is interesting to consider the use and interpretation of the pWCET distribution for a program. Let us assume that the program will be run repeatedly a potentially unbounded number of times, and that a fixed execution time budget of  $x$  applies to each run. Further, we assume that this budget is enforced by the operating system, and therefore that any run of the program which has not completed within an execution time of  $x$  is terminated and assumed to have failed. The pWCET distribution provides the following information (by reading off the probability of exceedance  $p$  associated with the execution time budget  $x$ , see Figure 2):

- (i) An upper bound  $p$  on the probability (with a long-run frequency interpretation) equating to the number of runs expected to exceed the execution time budget  $x$  divided by the total number of runs in a long (tending to infinite) time interval.
- (ii) An upper bound  $p$  on the probability that the execution time budget  $x$  will be exceeded on a randomly selected run. (This is broadly equivalent to the above long-run frequency interpretation).

Contrast this with the binary information provided by the WCET. If  $x$  is greater than or equal to the WCET, then we can expect the budget to never be exceeded. However, if  $x$  is less than the WCET, then we expect the budget to be exceeded on some runs, but we have no information on how frequently this may occur. Hard real-time systems in many application domains can in practice tolerate a small number of consecutive failures of a program to meet its execution time budget, but cannot tolerate long black-out periods when every run fails to complete within its budget. The problem of reconciling requirements on the length of potential black-out periods and a probabilistic treatment of execution times has, as far as we are aware, received little attention in the literature. We note that calculation of the probability of such black-out periods occurring requires information about the dependences (or independence) of the pWCET distributions for consecutive runs of the program. This topic is discussed further in the companion survey on probabilistic schedulability analysis [38].

We note that some researchers have interpreted the pWCET distribution as giving the probability or confidence  $(1 - p)$  that the WCET does not exceed some value  $x$ . This interpretation can be confusing, since the meaning of the WCET is normally taken to be the largest possible execution time that could be realised on any single run of the program, as in Definition 1. Instead, in line with Definition 2 and (i) above, we view the 1 - CDF of the pWCET distribution as providing, for any chosen value  $x$  for the execution time budget, an associated upper bound probability  $p$  (with a long-run frequency interpretation) equating to the number of runs of the program expected to exceed the execution time budget  $x$  divided by the total number of runs of the program in a long time interval.

## 2.2 Overview of Static Probabilistic Timing Analysis (SPTA)

The aim of Static Probabilistic Timing Analysis (SPTA) methods is to *construct* an upper bound on the pWCET distribution of a program by applying static analysis techniques to the code of the program (supplemented by information about input values) along with an abstract model of the hardware behaviour. Typically, a precise analysis is not possible due to issues of tractability, and thus over-approximations are made which may lead to pessimism in the upper bound pWCET distribution computed. For static analysis to produce a non-degenerate pWCET distribution there has to be some part of the system or its environment that contributes random or probabilistic timing behaviour.

SPTA methods for programs running on time-randomised hardware (e.g. with a random replacement cache) effectively consider each path through the code. For each path, these methods construct a pWCET distribution that upper bounds the probability distribution of the execution time for that path, considering all possible initial hardware states and all possible input states that cause execution of the path. For simple hardware, it is sufficient to consider an empty cache (the worst-case hardware state) and a single input state that drives the path, since there is no difference in execution times for different inputs that select the same path. Nevertheless, significant approximations need to be made in the analysis, since the problem of determining the possible cache states and their probabilities at each program point is intractable. The upper bound pWCET distributions for every path are then combined using an envelope function (taking the point-wise maxima over the 1-CDFs) to determine an upper bound on the pWCET distribution

for the program that is valid *independent* of the path taken. (More sophisticated SPTA methods analyse sub-paths and use appropriate join operations at path convergence to compute tighter upper bounds on the pWCET distribution of the program). While the results provide a valid upper bound on the pWCET distribution, they are not necessarily tight even for simple architectures.

Note that SPTA methods typically do not explicitly consider different valid scenarios of operation, but rather they effectively assume that any scenario (sequence of input states and hardware states) could occur, and thus over-approximate.

### 2.3 Overview of Measurement-Based Probabilistic Timing Analysis (MBPTA)

The aim of Measurement-Based Probabilistic Timing Analysis methods is to make a *statistical estimate* of the pWCET distribution of a program. This estimate is derived from a sample of execution time observations obtained by executing the program on the hardware or on a cycle accurate simulator according to an appropriate *measurement protocol*. The measurement protocol executes the program multiple times according to some sequence(s) of feasible input states and initial hardware states, thus sampling one or more possible scenarios of operation.

Provided that the sample of execution time observations passes appropriate statistical tests (see below), then *Extreme Value Theory* (EVT) can be used to derive a statistical estimate of the probability distribution of the *extreme values*<sup>7</sup> of the execution time distribution for the program, i.e. to estimate the pWCET distribution. By modelling the shape of the distribution of the extreme execution times EVT is able to predict the probability of occurrence of execution time values that exceed any that have been observed.

Early results in Extreme Value Theory required the sample of observations used to be *independent and identically distributed (i.i.d.)*; however, later work by Leadbetter [81] showed that EVT can also be applied in the more general case of a series of observations which are *stationary*, but are not necessarily independent. Both i.i.d. and stationary properties can be checked using appropriate statistical tests.

► **Definition 5.** A sequence of random variables (i.e. a series of observations) are *independent and identically distributed (i.i.d.)* if they are mutually independent (see Definition 4) and each random variable has the same probability distribution as the others.

► **Definition 6.** A sequence of random variables (i.e. a series of observations) is *stationary* if the joint probability distribution does not change when shifted in time, and hence the mean and variance do not change over time.

We note that simple software state machines that produce a cyclically repeating behaviour typically result in a stationary series of execution time observations.

In order for the estimated pWCET distribution derived by EVT to be valid for a future scenario of operation, then the sample of input states and initial hardware states used for analysis must be *representative* of those that will occur during that scenario.

► **Definition 7.** A sample of input states and initial hardware states used for analysis is *representative* of the population of states that may occur during a future scenario of operation if the property of interest (i.e. the pWCET distribution) derived from the sample of states used for analysis matches or upper bounds the property that would be obtained from the population of states that occur during the scenario of operation.

---

<sup>7</sup> By extreme values we mean large values towards the tail of the distribution, which have a small probability of occurrence.

► **Definition 8.** As determined by MBPTA, the estimated pWCET distribution for an entire program (or path through a program) is a statistical estimate of the probability distribution of the *extreme values* of the execution time of that program (or path), valid for any future scenarios of operation that are properly represented by the sample of input states and initial hardware states used in the analysis.

Ideally, MBPTA would provide a pWCET distribution which is valid for any of the many possible scenarios of operation; however, an important issue here is that there may not be one single distribution of input states and hardware states that is representative of all possible future scenarios of operation. This issue of *representativity* is a key open problem in research on the practical use of MBPTA.

The difficulty in ensuring representativity can be ameliorated by taking steps to make regions of the input state space equivalent in terms of execution time behaviour, similarly regions of the hardware state space. As a simple example, a floating point operation may have a variable latency that depends on the values of its operands; however, if a hardware test mode is used whereby the floating point operation always takes the same worst-case latency regardless of these values, then any arbitrary values can be chosen, and they will be representative (as far as execution times are concerned) of any other possible values in the input state space. Similarly, a fully associative random replacement cache can make many, but not necessarily all, patterns of accesses to data at different memory locations have equivalent execution time behaviour. Unfortunately, these steps typically require modifications to the hardware used. The problem of representativity is most acute for COTS (Common Off The Shelf) hardware. In particular, it is challenging to ensure representativity with hardware that has complex deterministic behaviour based on substantial state and history dependency. Examples include LRU/PLRU caches, and caches with a write-back behaviour. Here, the latency of instructions that access memory can vary significantly based on the specific history of prior execution, i.e. the specific sequence of addresses accessed. Further, it is hard to determine which regions of the input state space are equivalent in terms of execution time behaviour. This makes it very difficult to ensure that the input data used is representative.

Two EVT theorems and associated methods have been employed in the literature on MBPTA: the *Block Maxima* method based on the Fisher-Tippett-Gnedenko theorem, and the *Peaks-over-Threshold* (PoT) method based on Pickands-Balkema-de Haan theorem (see the books by Coles [32] and Embrechts et al. [46] for details of these theorems).

The Block Maxima method can be summarised as follows:

- Obtain a sample of execution time observations using an appropriate measurement protocol.
- Check, via appropriate statistical tests, that the execution time observations collected are analysable using EVT.
- Divide the sample into blocks of a fixed size, and take the maximum value for each block. (Note, in practice only the maxima of the blocks need be independent for the theory to apply, not necessarily all of the sample data).
- Fit a *Generalised Extreme Value* (GEV) distribution to the maxima. This will be a reversed Weibull, Gumbel, or Frechet distribution, depending on the shape parameter. (Alternatively fit a GEV with the shape parameter fixed to zero i.e. a Gumbel distribution).
- Check the goodness of fit between the maxima and the fitted GEV (e.g. using quantile plots).
- The GEV distribution obtained for the extreme values estimates the pWCET distribution.

The Peaks-over-Threshold method can be similarly summarised as follows:

- Obtain execution time observations using an appropriate measurement protocol.
- Check, via appropriate statistical tests, that the execution time observations collected are analysable using EVT.

- Choose a suitable threshold, and select the values that exceed the threshold. Note that *de-clustering*<sup>8</sup> may be required for data that is not independent.
- Fit a *Generalised Pareto Distribution* (GPD) to the excesses over the threshold.
- Check the goodness of fit between the peak values selected and the fitted GPD (e.g. using quantile plots).
- The GPD distribution obtained for the extreme values estimates the pWCET distribution.

There are two main ways of applying MBPTA, referred to as *per-path* and *per-program*:

1. *Per-path*: MBPTA is applied at the level of paths. A measurement protocol is used to exercise all feasible paths, then the execution time observations are divided into separate samples according to the path that was executed. EVT is then used to estimate the pWCET distribution for each path. The pWCET distribution for the program as a whole is then estimated by taking an upper bound (an envelope or point wise maxima on the 1 - CDFs) over the set of pWCET estimates for all paths.
2. *Per-program*: MBPTA is applied at the level of the program. A measurement protocol is again used to exercise all paths. In this case, all of the execution time observations are grouped together into a single sample. EVT is then applied to that sample, thus estimating directly the pWCET distribution for the program.

There are a number of advantages and disadvantages to the per-path and per-program approaches for MBPTA. Recall that the execution time observations analysed using EVT must be identically distributed, which can be checked using appropriate statistical tests. In the per-program case, these tests can fail if the execution times from different paths come from quite different distributions. The independent distribution (i.d.) tests could also fail in the case of observations for an individual path; however, for this to happen there must be significant differences in the execution time distributions obtained for the same path with different input states. This is less likely in practice, although it may still happen.

With the per-path approach, issues of representativity arise only at the level of paths, whereas the per-program approach raises issues of representativity at the program level. With the per-path approach, it is sufficient that the sample of input states and initial hardware states used to generate execution time observations for a given path are representative of the input states and hardware states for that path that could occur during operation. For example, for relatively simple time-randomised hardware it may be possible to obtain a single representative input state for each path by resetting the hardware on each run to obtain worst-case initial conditions (e.g. an empty cache). For more complex hardware, it becomes more difficult to ensure that the input states and initial hardware states used in the analysis of an individual path are representative; nevertheless, the problem is somewhat simpler than in the per-program case.

With the per-program approach, the frequency at which different paths are exercised in the observations used for analysis can impact the pWCET distribution obtained. For example if a path with large execution times is exercised much more frequently during operation than it was during analysis, then the pWCET distribution obtained during analysis may not be valid for that behaviour. Since the sample of input states used during analysis was not representative, the pWCET distribution obtained may be optimistic.

Despite the above disadvantages, the per-program approach may be preferable in practice, since it does not require the user to separate out the execution time observations on a per-path basis.

---

<sup>8</sup> De-clustering typically involves using only the single maximum value in any group of continuous observations that exceeds the threshold.

Further, the per-path approach loses information about the ordering and dependences between execution time observations, which may be relevant to obtaining a tight pWCET distribution.

In seeking to employ EVT, there are two questions to consider: First, are the samples of observations obtained for input into EVT *representative* of the future scenarios of operation that can occur once the system is deployed? If the answer to this question is “no”, then any results produced (e.g. estimated pWCET distributions) cannot be trusted to describe the behaviour of the deployed system. Secondly, can the particular EVT method chosen be applied to the observations? This question can be answered by applying appropriate statistical tests (e.g. checking for i.i.d., goodness-of-fit etc.). It is important to note that an answer “yes” to this second question does not necessarily imply that the results produced provide a sound<sup>9</sup> description of the behaviour of the deployed system. They may only provide a sound description of its behaviour in precisely those scenarios used for analysis, i.e. used to produce the observations. *Representativity* is essential to extend the results obtained via EVT to the actual operation of the system.

### 3 Static Probabilistic Timing Analysis (SPTA)

In this section, we review analytical approaches to determining pWCETs distributions, commonly referred to as Static Probabilistic Timing Analysis (SPTA). These methods *construct* an upper bound on the pWCET distribution of a program by applying static analysis techniques to the code of the program along with an abstract model of the hardware behaviour. In order for static analysis to produce a (non-degenerate) pWCET distribution there has to be some part of the system or its environment that contributes variability in terms of random or probabilistic timing behaviour. Examples include: (i) probabilities of certain inputs occurring, (ii) probabilities of faults occurring, (iii) time-randomised hardware behaviour, such as the use of random replacement caches. In the following subsections, we review the research on SPTA relating to each of these factors.

#### 3.1 SPTA based on Probabilities from Inputs

Initial work on SPTA considered the probability of different input values occurring, or different conditional branches being taken in the code.

The first work in this area by David and Puaut [35] in 2004 assumed that the input variables are independent and have a known probability distribution in terms of the values that they can take. In this work, a tree-based static analysis is used to compute the execution time of each path and the probability that it will be taken. This generates a probability distribution for the execution time of the program, not considering hardware effects. The method has exponential complexity, which depends strongly on the external variables. The main drawback is the requirement for the input variables to be independent, which is unlikely to be the case in practice. Further the probability distribution for each input variable must be known.

In 2008, Liang and Mitra [86] analysed the effects of cache on the probability distribution of the execution times of a program, assuming that probabilities of conditional branches and statistical information about loop bounds are provided. They introduce the concept of *probabilistic cache states* to capture the probability distribution of different cache contents at each program point, and an appropriate join function for combining these states. The analysis computes the cache miss probability at each program point enabling the effects of the cache to be included in

<sup>9</sup> In this survey, we use the adjective *sound* to indicate a description, an analysis, or a probability distribution that provides information about the system that is not optimistic with respect to its timing behaviour. Thus the information provided may be precise, or it may be pessimistic.

an estimation of the execution time distribution of the program. Aside from issues of tractability, the main difficulty with this approach is the assumption that values for the probability of taking different conditional branches can be provided, and the implicit assumption that these values are path independent, which they may not be.

### 3.2 SPTA based on Probabilities from Faults

Systems may exhibit probabilistic behaviour due to the occurrence of faults, either external to the microprocessor, for example due to sensor failures, or internal to it, for example due to faulty cache lines.

The initial work in this area by Hofig [62] in 2012 introduced a methodology for *Failure-Dependent Timing Analysis*. This combines static WCET analysis of the code with probabilities propagated from a safety analysis model. Thus a set of WCET bounds are obtained that are associated with particular situations, for example a combination of sensor failures, and probabilities that those situations will occur. These values can then be used to determine a valid WCET at any given acceptable deadline miss probability. The work assumes that sensor failures are independent. The authors argue that when failures are dependent then this dependence can be captured in the safety analysis resulting in revised probabilities for the different situations considered.

In 2013, Hardy and Puaut [58] presented a SPTA for systems with deterministic instruction caches in the presence of randomly occurring permanent faults affecting the RAM cells that implement the cache lines. This method first computes the WCET assuming no faults, and then determines an upper bound on the probabilistic timing penalty due to the additional fault induced cache misses. The timing penalty is derived independently for each cache set, and the results combined to obtain the overall penalty. The motivation for this work is that as microprocessor technology scales decrease so the probability of permanent RAM cell failures will increase. A journal extension by the same authors [59] examines the scalability of the method to larger cache sizes, and also covers the effect of different values for the failure probability of memory cells, reflecting different technology scales.

More recently in 2016, Chen et al. [31] presented a SPTA for systems with a random replacement cache subject to both transient and permanent faults. This approach uses a Markov Chain model to capture the evolution of cache states and their probabilities. The cache states are modified taking account of the impact of faults, and hence the resulting pWCET distributions obtained reflect the fault rates specified as well as the random replacement policy of the cache. The evaluation shows that the method produces tight bounds with respect to simulation results. The cache assumed is however only 2-way, hence it is not clear whether the method scales to larger cache associativities. In a subsequent paper, also in 2016, Chen et al. [29] addressed an issue with their previous work whereby increasing permanent fault rates substantially degrades the pWCET. In this work they compare the previous rule-based detection mechanism with a more complex method that uses Dynamic Hidden Markov Model detection. The former is simple to implement, but the latter is significantly more effective, providing better pWCET estimates for high fault rates.

### 3.3 SPTA based on Probabilities from Random Replacement Caches

Caches are small fast memories used to bridge the speed difference between the processor and main memory. Access times to cache can be in the region of 10 to 100 times faster than accesses to main memory, thus cache often has a significant impact on the execution time of programs. Much of the work on SPTA (and indeed MBPTA) has focussed on caches that use a random replacement policy. Before reviewing this work, we outline some fundamentals about caches and their operation.

Main memory, used to store both instructions (the program) and data, is logically divided up into memory blocks, which may be cached in cache lines of the same size. When the processor requests a memory block, the cache logic has to determine whether the block already resides in the cache, a *cache hit*, or not, a *cache miss*. To facilitate efficient look-up each memory block can typically only be stored in a small number of cache lines referred to as a *cache set*. The number of cache lines or *ways* in a cache set gives the *associativity* of the cache. A cache with  $N$ -ways in a single set is referred to as *fully-associative*, meaning a memory block can map to any cache line. At the other extreme, a cache with  $N$  sets each of 1-way is referred to as *direct-mapped*; here each memory block maps to exactly one cache line. With set-associative and direct mapped caches, a *placement policy* is used to determine the cache set that each memory block maps to. The most common policy is modulo placement which uses the least significant bits of the block number to determine the cache set. Since caches are usually much smaller than main memory, a *replacement policy* is used to decide which memory block (i.e. cache line in the set) to replace in the event of a cache miss. Replacement policies include Least Recently Used (LRU), Pseudo LRU (PLRU), FIFO, and Random Replacement. Early work by Smith and Goodman [121, 122] in 1983 considered FIFO, LRU, and random replacement caches, concluding that the performance of random replacement is superior to FIFO and LRU, when the number of memory blocks accessed in a loop exceeds the size of the cache. LRU is superior when it does not.

The origins of SPTA for systems with random replacement caches can be traced to initial work by Quinones et al. [106] and Cazorla et al. [26] which provided a simple analysis restricted to single-path programs. Subsequently, Davis and Altmeyer and their co-authors Griffin and Lesage developed more sophisticated and precise analysis that also covers multi-path programs [39, 7, 6, 52, 83, 82].

The early work of Quinones et al. [106] in 2009 explored, via simulation, the performance of caches with a random replacement policy compared to LRU. The evaluation involves programs with a number of functions (greater than the cache associativity) called from within a loop. The different cases explored correspond to different memory layouts for the functions. Here, a small number of layouts result in pathological access patterns (referred to as *cache risk patterns* [95]) where, assuming LRU replacement, each function evicts the instructions for the next function from the cache. The simulation results show that random replacement has better performance than LRU in these cases, and lower variability in performance over the range of memory layouts explored. Further, when the code is too large to fit in the cache, then LRU has relatively poor performance, with random replacement providing better cache hit rates and less variability across different layouts. The authors suggest a simple method of statically computing the probability of pathological behaviour in the case of a random replacement cache; however, this formulation was later shown to be optimistic (i.e. unsound) by Davis et al. [39], since random replacement does not eliminate all of the dependences on access history.

In 2011, Burns and Griffin [23] explored the idea of predictability as an emergent behaviour. They show that if components are designed to exhibit independent random behaviour, then an execution time budget with a low probability of failure can be estimated that is not much greater than the average execution time. Their experiments examine hypothetical programs made up of instructions with execution times that are independent and identically distributed (i.i.d). The execution of each instruction is assumed to take either 1 cycle (representing a cache hit) or 10 cycles (representing a cache miss), with a 10% probability of the latter. Thus the average execution time of an instruction is 1.9 cycles. The authors found that for programs of more than  $10^5$  instructions, the execution time budget at an exceedance probability of  $10^{-9}$  was only a few percent larger than the average execution time. They note, however, that current hardware does not result in instructions with random execution times.

A simple SPTA for caches with an *evict-on-access*<sup>10</sup> random replacement policy based on reuse distances was presented by Cazorla et al. [26] in 2013. This analysis upper bounds the probability of a miss on each access in a way that is independent of the execution history, thus enabling the distributions for each access to be combined using basic convolution to produce a pWCET distribution for the execution of a trace of instructions, i.e. a single path through the program. Comparisons are made with analysis for a LRU cache, showing that random replacement is less sensitive to missing information. Reineke [108] later observed that the formula given for a random replacement cache reduces to zero whenever the re-use distance is equal to or exceeds the cache associativity. Hence, in a like-for-like comparison, analysis for LRU strictly dominates that for random replacement presented in this paper. We conclude that the results given by Cazorla et al. are somewhat misleading; they are an artefact of the difference in associativity rather than a difference in replacement policy, since 2-, 4-, and 8-way LRU caches are compared to a fully-associative random replacement cache. We note; however, that when more precise analysis of a random replacement cache is used there are some circumstances when it can outperform state-of-the-art analysis for LRU, as shown by Altmeyer et al. [6] in 2015.

In 2013, Davis et al. [39] introduced a SPTA technique based on re-use distances that is applicable to random replacement caches that use an *evict-on-miss policy*<sup>11</sup>. This dominates the earlier analysis of Cazorla et al. [26] which assumes *evict-on-access*. The authors show that it is essential that any analysis providing probability distributions per instruction that are then convolved to form a pWCET distribution for the program must provide distributions for each instruction that are *independent* of the pattern of previous hits or misses, otherwise the analysis risks being unsound (i.e. optimistic). The authors extend their analysis to cover multi-path programs and the effect of preemptions. By taking a simplified view of preemption, effectively considering that it flushes the cache, they derive analysis that bounds the maximum impact of multiple preemptions on a program and show how computation of probabilistic cache related preemption delays (pCRPD) can be integrated into SPTA. We note that although the analysis is presented for fully associative caches, it is also applicable to set-associative caches, since each cache set operates independently.

Subsequently in 2014, Altmeyer and Davis [7] introduced effective and scalable techniques for the SPTA of single path programs assuming random replacement caches that use the *evict-on-miss* policy. They show that formulae previously published by Quinones et al. [106] and Kosmidis et al. [68] for the probability of a cache hit can produce results that are optimistic and hence unsound when used to compute pWCET distributions. In contrast, the formula given by Davis et al. [39] is shown to be optimal with respect to the limited information it uses (reuse distances and cache associativity), in the sense that no improvements can be made without requiring additional information. The authors also introduce an improved technique based on the concept of *cache contention* which relates to the number of cache accesses within the reuse distance of a memory block that have already been considered as potentially being a cache hit. An exhaustive approach is also derived that computes exact probabilities for cache hits and cache misses, but is intractable. They then combine the exhaustive approach focussing on a small number of the most *relevant* memory blocks with the cache contention approach for the remaining memory blocks.

Altmeyer et al. [6] extended their earlier work [7] in a journal publication in 2015. In this paper, they introduce a new formula for the probability of a cache hit based on stack distance, correct an error in the formulation of basic cache contention, and provide an alternative cache

<sup>10</sup>The *evict-on-access* random replacement policy evicts a randomly chosen cache line whenever an access is made to the cache, irrespective of whether that access is a cache hit or a cache miss.

<sup>11</sup>The *evict-on-miss* random replacement policy evicts a randomly chosen cache line whenever an access is made to the cache and that access is a cache miss.

contention approach based on the simulation of a feasible evolution of the cache contents. Both cache contention approaches are formally proven correct, and are shown to be incomparable with the new stack distance approach. They also present an alternative more powerful heuristic for selecting relevant memory blocks, with blocks no longer considered as relevant once they are no longer used. This improves accuracy, while ensuring that the analysis remains tractable. The evaluation shows that the cache contention techniques improve upon simple methods that rely on reuse distance or stack distance, and that the combined approach with 4 to 12 relevant memory blocks makes further substantial improvements in precision. Specific comparisons with LRU show that the simple reuse distance and stack distance approaches are always outperformed, in fact dominated, by analysis for LRU. In contrast, the sophisticated combined analyses for random replacement caches are incomparable with those for LRU. LRU is more effective when the number of memory blocks accessed in a loop does not exceed the associativity of the cache, whereas random replacement is more effective when they do.

In 2014, Griffin et al. [52] applied lossy compression techniques to the problem of SPTA for random replacement caches. They built upon the exhaustive collecting semantics approach developed by Altmeyer and Davis [7], exploiting three opportunities to improve runtime while trading off some precision: (i) memory block compression, (ii) cache state compression, and (iii) history compression. Two methods of memory block compression were considered. The first excludes memory blocks with a hit probability below some threshold, while the second excludes those with a forward reuse distance that exceeds a given threshold. Both cache state and history compression are applied via the use of fixed precision fractions. The lossy compression techniques are highly parameterisable enabling a trade-off between precision and runtime. Further, the runtime is linear in the length of the address trace, compared to quadratic complexity for the combined technique derived by Altmeyer and Davis [7].

An effective SPTA for multi-path programs assuming a random replacement cache was developed by Lesage et al. [83] in 2015. This work adapts the cache contention and collecting semantics approach derived by Altmeyer and Davis [7] to the multi-path case. It also introduces a conservative join function which provides a sound over-approximation of the possible cache contents and pWCET distribution on path convergence. The analysis makes use of the control-flow graph. It first unrolls loops, since this allows both the cache contention and collecting semantics to be performed as simple forward traversals of the control-flow graph. Approximation of the incoming cache states on path convergence, via the conservative join operator, keeps the analysis tractable. The distributions obtained via the cache contention and collecting semantics are then combined using convolution. The main analysis technique is supplemented by *worst-case execution path expansion*. This method uses the concept of *path inclusion* to determine when one path includes another, necessarily resulting in a larger pWCET distribution. This enables the included path to be removed from the analysis. This concept simplifies the analysis of loops, since only the maximum number of loop iterations need be considered. The evaluation shows that this multi-path SPTA reduces the number of misses predicted for complex control flows by a factor of 3 compared to the simple path merging approach proposed earlier by Davis et al. [39]. Incomparability between analysis for LRU and sophisticated analysis for random replacement caches is also demonstrated on multi-path programs. The authors also investigate the runtime of their methods, showing that it is tractable with 4-12 relevant memory blocks. To reduce the runtime for long programs they also consider splitting such programs into Single Entry Single Exit regions<sup>12</sup> which can be analysed separately, with the pessimistic assumption of an empty cache at the start of each region. This approach is shown to be effective, permitting the use of more relevant memory blocks and

<sup>12</sup> An idea first suggested by Pasdeloup [104].

hence in some cases improving precision. In a journal extension, Lesage et al. [82] incorporate advances in SPTA techniques for single path programs into the analysis framework for multi-path programs.

In 2017, Chen and Beltrame [28] derived a SPTA for single path programs running on a system with an evict-on-miss random replacement set-associative cache. They use a non-homogeneous Markov chain to model the states of the system. For each step (i.e. memory access in the trace of the program), the method computes a state occurrence probability vector for the cache and a transition matrix which determines how the state vector is transformed. The computational complexity of the basic method increases as a polynomial with a large exponent given by the number of memory addresses. To contain the complexity and make the approach tractable, the authors adapt the method as follows. For the first  $n$  addresses, the state space is constructed using the Markov chain method as described above. Then when a new memory address is accessed that is not in the current state, another memory address that is part of the state and is either not used in the future, or has the longest time until it is used in the future, is discarded. This is a form of lossy compression that ensures the number of states does not increase further. The authors show how the method can be adapted to write-back data caches, and compare its performance with the SPTA method of Altmeyer et al. [6]. The evaluation shows that the adaptive Markov chain method provides improved performance, with the geometric mean of the execution times at an appropriate probability of exceedance reduced by 11% for the set of Mälardalen benchmarks studied. The runtime of the two methods is shown to be similar.

Also in 2017, Chen et al. [30] developed a SPTA for random replacement caches with a detection mechanism for permanent faults. The detection mechanism periodically checks the cache for faulty blocks and disables them. The analysis builds upon the combined approach of Altmeyer et al. [6]. The authors derive formula for two operating modes: with and without fault detection turned on. By combining these two modes, the method provides timing analysis accounting for the integrated fault detection. The experimental evaluation provides a comparison with simulation for the Mälardalen benchmarks, assuming a 2-way, 1 KByte instruction cache, with 16 byte cache lines, and a permanent fault rate that equates to a mean time between failures of 5 years. Other experiments consider higher fault rates, larger cache lines and a 4-way cache. The results show that when sufficient relevant blocks are used, SPTA provides results that are close to those obtained via simulation.

### 3.4 Summary and Perspectives

Static Probabilistic Timing Analysis (SPTA) for systems using random replacement caches has matured considerably since its origins in the work of Quinones et al. [106]. State-of-the-art techniques by Altmeyer et al. [6], Chen and Beltrame [28] and Lesage et al. [83, 82] provide effective analysis for single and multi-path programs respectively on systems using set-associative or fully-associative random replacement caches. This analysis is however limited to systems with single-level private caches. As far as we are aware SPTA techniques have not yet been developed for multi-level caches or for multi-core systems where the cache is shared between cores. A preliminary discussion of these open problems can be found in the work of Lesage et al. [84] and Davis et al. [40] respectively.

## 4 Measurement-Based Probabilistic Timing Analysis (MBPTA)

In this section, we review Measurement-Based Probabilistic Timing Analysis (MBPTA) methods. These methods use statistical techniques based on Extreme Value Theory (EVT) to derive an estimate of the pWCET distribution of a program from a sample of execution time observations.

These observations are obtained by executing the program on the hardware or a cycle accurate simulator under a measurement protocol. The measurement protocol executes the program multiple times according to a set of test vectors and initial hardware states, thus sampling one or more possible scenarios of operation.

Whether or not useful results can be determined using MBPTA methods depends crucially on the sample of execution time observations obtained and their properties. Early work in this area required execution time observations to be independent and identically distributed (i.i.d.). Later work relaxed this assumption, but still required that the sequence of execution time observations exhibit stationarity and weak dependences. Further, the results are only valid for future scenarios of operation for which the sample of observations is representative (see Section 2.3).

In the following subsections, we review: (i) early research into MBPTA that requires execution time observations to be i.i.d., (ii) subsequent research that relaxes the i.i.d. assumption, (iii) research which focusses on issues of representativity.

## 4.1 EVT and i.i.d. observations

In this section, we review early work on MBPTA based on the application of Extreme Value Theory that assumes the execution time observations are i.i.d. This work began with a number of papers by Burns and co-authors Edgar and Griffin [22, 45, 51], and culminated with the work of Cucu-Grosjean et al. [34] which inspired a substantial body of subsequent research into MBPTA.

In 2000, Burns and Edgar [22] introduced the use of EVT (the *Fisher-Tippett-Gnedenko* theorem) in modelling the maxima of a distribution of program execution times. They motivate the work by noting that advanced processor architectures make it prohibitively complex or pessimistic to estimate WCETs using traditional static analysis techniques. The following year, Edgar and Burns [45] introduced the statistical estimation of the pWCET distribution for a task. They use measurements of task execution times to build a statistical model. These observations are assumed to be *independent*, obtained over a range of environmental conditions, and of sufficient number for generalisations to be applied. The method they propose involves fitting a Gumbel distribution directly to the observations, using a  $\chi$ -squared test to determine the scale and location parameters. Such direct fitting is however not strictly correct, since the distribution is used to model the *maxima* of subsets of values, not all of the values themselves, as noted by Hansen et al. [57]. Further, there are also issues in using a  $\chi$ -squared test, which is not well suited to this purpose, as noted by Cucu-Grosjean et al. [34]. The authors show that if all tasks are executed with execution time budgets that must sum to some fixed total, then the optimal setting for the budgets achieves that total with an equal probability of each task exceeding its budget.

In 2009, Hansen et al. [57] considered the use of EVT to estimate pWCET distributions with an appropriate level of confidence. They correct a key issue in the work of Edgar and Burns [45] by using the *Block Maxima* method. Here the observations of execution times are first grouped into blocks, then the maximum value is taken for each block. The Gumbel distribution is then fitted to the distribution of the block maxima, with a  $\chi$ -squared test used to determine goodness of fit. The experimental evaluation presented involves over 100 tasks running on a commercial PowerPC-based device using the VxWorks operating system. The block size is initially set to 100, and doubled repeatedly if the  $\chi$ -squared test does not indicate a sufficiently good fit. (We note that this doubling may have been sufficient to capture stationarity in the observations, see the later work of Santinelli et al. [112]). The Gumbel parameters were obtained by using linear regression on a QQ-plot. The results produced using EVT were validated against additional measurements (over 2 million in the case of one task reported upon in detail). These results show that the EVT method provides a much better estimate of the extreme values with respect to subsequent observations, than simply taking the maximum observed value and assuming that it is

the WCET. Nevertheless, there were some tasks where the rate at which execution times (captured during the validation stage) exceeded some large value was greater than the exceedance probability predicted by the Gumbel distribution, i.e. the pWCET distribution was optimistic.

The use of EVT to model the extreme execution times of programs was considered by Griffin and Burns [51] in 2010. They provide a critique, discussing a number of issues that need to be addressed in order for the results produced to be sound. These include issues relating to the use of upper bounds (e.g. Gumbel distribution) that are continuous when the possible execution times of a program may have large discrete steps, and the need for the observations used to be *independent and identically distributed* (i.i.d.). They describe various ways in which observations may be dependent (e.g. via internal state such as cache contents, and also via external factors such as input variables that are affected by the previous outputs of the system leading to a history dependence). They also note that different paths through the program lead to different execution time distributions, and hence issues arise with the assumption that all the observations are identically distributed. Some possible solutions are suggested, such as shifting the 1 - CDF of the continuous distribution so that it upper bounds its discrete counterpart at all execution time values, including those that cannot actually be obtained. Possible solutions to issues with the i.i.d. assumption include resetting the system state between observations, and determining the set of potential worst-case paths by some other means, and then analysing each of those paths separately, i.e. the per-path approach (discussed in Section 2.3 ).

In 2011, Lu et al. [89, 90] examined issues with the application of EVT highlighted by Griffin and Burns [51]. They address the requirement that the observations must be i.i.d. They note that a Gumbel distribution should not be directly fitted to the observations as done by Edgar and Burns [45], since the Gumbel (and other EV distributions) are intended to model distributions of the maxima (or minima) of a large number of random variables. Instead, the authors propose the use of Simple Random Sampling in terms of randomising program inputs, and dividing observations into groups of  $N$  sets, each of  $m$  observations, and only using the largest of the  $m$  observations in each set, i.e. the *Block Maxima* method. We note that Simple Random Sampling may break dependences in the observations with the potentially for pWCET estimates to then be optimistic. (The authors give no proof that re-sampling the observations in this way ensures a sound, i.e. pessimistic result).

In a seminal paper published in 2012, Cucu-Grosjean et al. [34] introduced a statistically sound method of applying EVT to probabilistic timing analysis, referred to in the paper as Measurement-Based Probabilistic Timing Analysis (MBPTA). This method uses end-to-end execution time measurements obtained from the system during testing. For the method to be applicable, the observations must be i.i.d. Statistical tests (two sample Kolmogorov-Smirnov test, and Runs test) are used to check that this is the case. The Block Maxima method is used to group the observations, and the Exponential Tail test is used to check if the distribution of the maxima fits the Generalised Extreme Value (GEV) distribution, in particular, a Gumbel distribution. The method also proposes an approach for determining the number of observations required for each program path. In applying MBPTA to multi-path programs, the authors note that *path coverage* is required, otherwise the requirements for i.i.d. observations would be broken, and give an example of how the results could be unsound in this case. With full path coverage; however, the method appears to be relatively insensitive to the number of observations on each path, provided there are sufficient of them. The authors further note that for the i.i.d. property to be preserved for multi-path programs, either the inputs can be selected randomly when making measurements and the observations grouped sequentially, or testing can be done with all possible inputs and then observations selected via random sampling without replacement when performing the grouping into blocks. We note that this may bias the block maxima towards the worst-case,

since if high execution times are grouped, random sampling may increase the likelihood that a block contains a high value. This can potentially result in both pessimism and optimism in the pWCET distribution, since it changes the shape of the distribution of the block maxima. The authors evaluate the method for both single-path and multi-path programs running on a simulated microprocessor with a random replacement cache. Comparisons against the simple SPTA method of Cazorla et al. [26] show that the pWCET distribution obtained via MBPTA typically overestimates that derived via SPTA by 3-15% at small probabilities. We note that since later work by Altmeyer et al. [6] shows that the SPTA method described by Cazorla et al. [26] typically produces results that have substantial pessimism, substantive conclusions cannot be drawn about the absolute accuracy of the MBPTA method from these comparisons.

## 4.2 EVT and observations with dependences

The initial work on MBPTA by Cucu-Grosjean et al. [34] in 2012 resulted in increased interest in this area of research. Subsequent developments have focussed on relaxing the i.i.d. assumption and thus facilitating analysis of systems where execution time observations exhibit dependences. Much of the work in this area emanates from Santinelli and his co-authors Melani, Berezovskyi, and Guet [93, 112, 16, 53, 54]. They leverage early work by Leadbetter et al. [81] in 1978 and Hsing [63] in 1991 showing that EVT can be applied to stationary and weakly dependent time series.

In 2013, Melani et al. [93] investigated the use of statistical methods based on EVT when execution time observations show dependences on some other event in the system. For example preemptive rather than non-preemptive scheduling, or different preempting tasks that either heavily load the processor or the cache. They suggest a characterisation of dependences by effectively shifting, by an amount  $\Delta$ , the pWCET distribution for a program as obtained in isolation. Evaluation shows that this method is effective for code from the Maladarlen benchmark suite [56] running on an Intel Xeon processor with 3 levels of cache. A small shift, compared to the overall execution time, is sufficient to account for the effect of the events which the execution times are dependent on. We note that shifting the pWCET distribution in this way equates to adding a fixed overhead to account for the extra interference due to preemption and scheduling.

The case for applying EVT when there are dependences between observations was examined by Santinelli et al. [112] in 2014. They note the prior work by Leadbetter et al. [81] and investigate less constraining hypotheses than independence such as stationarity and extremal dependence. To verify stationarity, autocorrelation is computed using lag plots. Experimental evaluation on an Intel Xeon processor with four cores (per socket) and 3 levels of cache shows that there is sufficient execution time variability for EVT to be applied. Observations from multi-path code exhibit stationarity, but pass the tests required in order for EVT to be applied. This indicates that the use of time-randomised hardware (e.g. random replacement caches) is not necessary in order to apply EVT. They also discuss dependence between extreme samples. This can be depicted using an *extremogram* (described by Davis and Mikosch [36]), which shows whether extreme samples are correlated in terms of their separation, i.e. whether they are clustered, or distributed throughout the trace of observations. Further, the authors examine the effect on the pWCET distributions of choosing different values for the size of the blocks in the Block Maxima method and different thresholds in the Peaks-Over-Threshold (PoT) method. Significant sensitivity is demonstrated to these values, with a degradation in performance at larger block sizes. It is not clear whether this is because the total number of samples is kept constant and thus the larger block sizes equate to too few blocks. Similarly, using too high a threshold (equating to the 98 percentile) results in a very large pWCET estimation. Again it may be that too few samples remain. We note that while this work provides useful insight into the importance of appropriate parameter selection, no guidance is given on how such parameters should be selected to achieve accurate results.

In 2014, Berezovskyi et al. [16] investigated the use of MBPTA techniques based on EVT to estimate the pWCET of CUDA kernels running on an NVIDIA GPU. They discuss the use of the Generalised Extreme Value (GEV) distribution, and developments that show that independence of observations is not necessary for the application of EVT. They note the prior work by Leadbetter et al. [81] and Hsing [63] and recount theorems relating to EVT for sequences with *long range independence* and *extremal independence* [81], and note that randomness is not sufficient in itself to show independence in these cases. Further, the Runs test previously used by Cucu-Grosjean et al. [34] does not suffice; rather time series<sup>13</sup> tests based on auto-regression and autocorrelation are needed. The authors suggest the use of lag plots to determine autocorrelation. They use autocorrelation tests together with the notion of stationarity to quantify statistical independence. They also use an autoregressive model to determine the stationarity of a trace of observations. The Ljung-Box test is used to look for correlations between lags. Finally, extremeograms are used to estimate dependence at the extreme values. The evaluation uses a Kepler GK104 GPU. Tests on the trace of observations for the GPU computations show them to be independent. By contrast, the observations of the execution times including data transfer to and from the host processor are not independent, but they are stationary. Here, the Runs test would have concluded independence; however, in fact there is a strong stationary relationship, but since this does not reflect as dependence in the extreme observations, EVT can still be applied.

In 2016, Guet et al. [53] proposed a logical work-flow (embedded in a tool called DIAGXTRM) that checks the applicability of EVT for the pWCET estimation problem. This work discusses Pickands-Balkema-de Haan Theorem [105], the Generalised Pareto Distribution (GPD), and the Peaks-Over-Threshold (PoT) method. The authors again note the prior work by Leadbetter et al. [81] showing that EVT is applicable in the more general stationary case without independence. Here, the hypotheses to check on the trace of observations are (i) stationarity, (ii) short range dependence, (iii) local independence of the peaks, and (iv) that the empirical peaks over the threshold follow a GPD. The efficiency of the proposed logical work-flow is likely affected by the relation between these hypotheses, which are not independent. Moreover, the authors motivate the introduction of the hypothesis of stationarity based on the impossibility of checking the identically distributed hypothesis when the bounds on the execution time of a program have unknown probability distributions. Here, we note that the arguments are not correctly used as stationarity is an alternative to the hypothesis of statistical independence, rather than an alternative to the identically distributed hypothesis, further EVT is itself applied only to those cases where the measurements follow unknown probability distributions. To evaluate stationarity, the authors use the Kwiatkowski–Philips–Schmit–Shin (KPSS) test. Further, the Brock–Dechert–Scheinkman test is used to evaluate short range dependence, by examining the correlation between different sub-sequences of the same length. The reliability of EVT also depends on the independence of extreme observations. Here, the *Extreme Index* (see Embrechts et al. [46]) is used to give the probability that peaks are far enough apart to be considered independent, as opposed to constituting a burst relating to a single rare event. The threshold selection in the PoT method is a source of uncertainty, since different thresholds may lead to different pWCET estimates. Reviews of threshold selection methods by Scarrott and MacDonald [114] show that there is no recognised systematic process for selection. The authors therefore propose an approach that aims to provide an appropriate tail sample. They evaluate their approach using execution time measurements obtained from an Intel Xeon processor with 4 cores and 3 levels of cache. Code from the Maladarlen benchmark suite is run in various configurations including isolation, alternately with a program that makes heavy use of the cache, and with that program running on the other

---

<sup>13</sup>The time series here is the observations of execution times in the order in which they were made.

cores. All the traces of observations were found to be stationary with high confidence. Short range and extreme independence was also verified. Note, Berezovskyi et al. [15] later applied the PoT approach proposed by in this paper to the NVIDIA Kepler GK104 GPU system that they had previously investigated using the Block Maxima method [16].

Later in 2016, Guet et al. [54] investigated using measurement based probabilistic analysis to estimate the number of cache misses for programs running on a COTS multi-core processor with two Intel Xeon E5620 sockets, using 3 levels of deterministic PLRU caches, with the first two levels private to each core and the last level shared by cores on the same socket. Their aim was to estimate the worst-case number of cache misses for programs under different configurations: (i) isolation, (ii) on a single core alternating with another program on the same core, (iii) on one core with another program executing simultaneously on a different core, and (iv) combining (ii) and (iii). (Note, the other program makes a large number of memory accesses). The authors applied EVT to the problem using the Block Maxima method. They note that the observations of cache misses are unlikely to be independent, nevertheless, they show that the observations for most of the programs examined from the Mälardalen benchmark suite exhibited stationarity and/or extremal independence and so can be analysed using EVT.

In 2017, Fedotova et al. [47] applied the PoT method of EVT to estimate the upper bound values for a timer acquisition task. This involves setting two consecutive timers using the *HighPerTimer* library and calculating the time difference between them. The platform used runs the standard Linux kernel on ARM Cortex-A5 hardware. The authors discuss how the choice of threshold to use in the PoT method is a compromise between precision and bias, and make use of two graphical approaches for threshold selection: (i) mean residual life, which plots the mean excess against the threshold, and (ii) parameter stability, which plots the shape and modified scale parameters of the GPD against the threshold. Using these methods, they select an appropriate threshold which obtains 24 extreme values from the trace. They show that both Gumbell and Frechet distributions fit the data and compute pWCET estimates with various probabilities of exceedance for both cases. Due to the different shape parameters for the two distributions, these pWCET estimates exhibit large differences e.g. 7.7ms v. 5900ms at a probability of exceedance of  $10^{-9}$ . These large differences raise questions about the variability of results that can be obtained from the same data, and hence the confidence in them. We note that  $10^{-9}$  represents a large amount of extrapolation from the sampled data, with the pWCET estimates much closer together at a probability of exceedance of  $10^{-7}$  (2.5ms v. 11.3ms).

Also in 2017, Lima and Bate [87] proposed a technique called IESTA (Indirect Estimation in Statistical Time Analysis) to support the application of EVT without the need for hardware or software randomisation. The basic idea is to add a randomised component (e.g. from a normal distribution with values in the range  $[a, b]$ ) to the set of execution time observations. This additional component removes some of the discreteness from the original distribution, and makes the new values more likely to pass the statistical tests required for the application of EVT. Once the random component has been added to the observations, then the maxima are sampled, using either PoT or Block Maxima methods, and the pWCET estimated by fitting to a GEV distribution. Goodness-of-fit tests indicate if the estimate is a close match to the empirical distribution of the maxima or peaks over the threshold. If it is not, then the dispersal of the random component can be increased, which makes it more likely that EVT can be applied and the goodness-of-fit tests passed. The authors explore two benchmarks, one from the Mälardalen benchmark suite where EVT could not be applied even with time-randomised hardware, and a second using data from an aircraft control application from Rolls-Royce running on entirely time-predictable hardware. The latter has significant dependences between execution times, as shown in autocorrelation plots with lags of up to 40. For both benchmarks, the IESTA method is able to add a random component

enabling EVT to be applied. Both the random padding and a high threshold selection help make it unlikely that dependences appear in the sample of the maxima. The evaluation shows that even when the dispersal of the random component is large, then the increased pessimism in the analysis remains small (i.e. only a few percent). It remains an open question whether this method may result in optimistic pWCET estimations.

### 4.3 EVT and representativity

Representativity is a fundamental issue in applying statistical methods (i.e. MBPTA) to estimate the pWCET distribution of a program. The problem is that the results obtained are only valid for those scenarios of operation for which the sample of observations used in the analysis is representative (see Section 2.3). Often it may not be possible to devise a measurement protocol that provides a single sample of observations that is representative of all possible future scenarios of operation. Research considering the problems of representativity are reviewed below.

In 2016, Lima et al. [88] took a careful look at the use of EVT in determining pWCET bounds for programs running on both time-predictable (deterministic) and time-randomised architectures (i.e. with random replacement caches). They point out that execution time observations depend on the probability distribution describing how often different input values occur. Thus there may not be a single distribution that describes the execution time behaviour. The authors therefore introduced the concept of a *weak pWCET* which is valid for a particular distribution  $D(\tau_i)$  describing the frequency of occurrence of input values. This raises the problem of *representativity* of input data. A simple experimental example shows how the estimated pWCET distribution produced depends on the distribution of input values, and hence that if the input data distribution used for analysis does not match that occurring during operation of the system, then the results obtained may be unsound. Applying uniformly distributed input values during analysis may also result in poor or unsound estimates. The authors note that in some cases it is not possible to estimate a reliable Generalised Extreme Value (GEV) model and they cite appropriate extreme value condition tests that can be used to determine if this is the case. They show that the GEV distribution should be used rather than assuming a specific distribution (e.g. Gumbel). The models determined for various experiments belong to all three classes of EV distribution (reversed Weibull, Gumbel, and Frechet) not just Gumbel. There were also cases where the models belonged to none of them and EVT could not be used to provide sound results. The experimental results reported in this work also show that EVT can be successfully applied on time-predictable platforms, thus indicating that hardware randomisation is not necessary for the application of EVT. They also found that hardware randomisation is also not in general sufficient to ensure that EVT can be applied. Random replacement caches that were either too large or too small were observed to make estimation of the pWCET distribution either harder or not possible at all.

Issues of *reproducibility* and *representativity* with respect to MBPTA methods were discussed by Maxim et al. [92] in 2016. They consider two separate steps: (i) the measurement protocol, i.e. obtaining execution time observations and (ii) the method used to estimate the pWCET distribution. They note that to be useful, the estimation method must be reproducible in the sense that it must provide, within a close approximation, the same pWCET distribution, given the same set of observations. Further, the measurement protocol must also be reproducible, in the sense that the two sets of observations that it provides for two different uses of the method, starting from the same conditions (processor state, external inputs etc.) must result in the same or sufficiently close pWCET distributions. The authors also consider representativity. They note that a measurement protocol is representative if there is some value of  $k$  (the number of observations) for which taking any larger number of observations results in a pWCET distribution that closely approximates the distribution that would be obtained if the whole domain of possible observations

were considered. In other words after  $k$  observations the method *converges* on a sound pWCET distribution. The properties of reproducibility and representativity are shown to be mandatory for convergence and hence required for any MBPTA method intended for use in practice. How to obtain these properties is; however, left as an open issue.

MBPTA techniques were revisited by Santinelli et al. [110] in 2017. In this work, they apply EVT to a case study comprising traces of observations from various example systems using both time-randomised and time-predictable hardware. They discuss the validity of EVT with respect to different execution conditions (similar to the issues of representativity raised by Lima et al. [88]) and suggest two ways of integrating all possible execution conditions into EVT: (i) trace merging and (ii) the use of an envelope (i.e. upper bounding the results for each separate execution condition). We note that it is not clear whether trace merging always produces valid results. The authors further propose a reliability measure based on the confidence levels of each of the hypotheses. They show that confidence is lower if the distribution is artificially forced to fit a Gumbel distribution rather than obtaining the best fit for the shape parameter. They also discuss how to choose the threshold in the PoT method, suggesting that a threshold should be selected that maximises confidence in the matching hypothesis with the largest amount of independent peaks.

In 2017, Santinelli and Guo [111] proposed a probabilistic representation framework, modelling tasks via multiple pWCET distributions. They noted that the pWCET distributions obtained via EVT strongly depend on the execution conditions used for analysis. These execution conditions describe aspects such as the environment, inputs, task mapping, presence of faults etc. The authors therefore propose describing each task via a *Worst-Case Set* which is a collection of pWCET distributions obtained via EVT for different execution conditions. They note that the number of different execution conditions is finite, and a partial ordering may be possible between some of them, indicating dominance relationships (in the sense of the greater than or equal to operator  $\succeq$  defined by Diaz et al. [44]). They note that enumerating all possible execution conditions is a complex problem (since there may be very many of them); however, tasks could be represented by pWCET distributions which bound those for collections of execution conditions with incomparable pWCET distributions. The authors propose the use of their model for mixed criticality systems, with different execution conditions for different criticality levels, and for systems with and without faults.

The problems of applying EVT when execution time observations have a *mixture distribution*, resulting in a step-like curve on an exceedance (1 - CDF) graph, were discussed by Abella et al. [3] in 2017. (Note, mixture distributions can occur as a result of caches that employ random placement, discussed in Sections 6.1 and 6.2). In this case, it is important that sufficient observations are obtained and that the threshold (PoT method) or block size (Block Maxima method) is set appropriately, such that sufficient values from the actual tail of the distribution are passed to EVT, as opposed to including too many values from other parts of the distribution. The authors present a method for selecting the observations required and suitable EVT parameters. First, they argue that in real-time systems, programs have finite execution times and so heavy tailed GEV/GPD distributions may be discarded. Further, since somewhat pessimistic pWCET estimates are acceptable, but optimistic estimates are not, they argue that it is sufficient to consider distributions with exponential tails (i.e. Gumbel distributions) as a bound for all potential light-tailed distributions. They propose a method of determining appropriate parameters based on considering the Coefficient of Variation (CV) of the residual distribution, i.e. the distribution of the excess over the threshold in the PoT method. (Note, the CV is given by the standard deviation of a distribution divided by its mean). The method employs a plot of the CV versus the number of samples over the threshold to find a suitable number of samples in the tail distribution, for which the assumption of an exponential tail is not rejected by statistical tests and the CV

is closest to 1, implying an exponential tail. The approach is evaluated using a number of the EEMBC benchmarks, but considering only single paths. Execution time observations are obtained by making runs in isolation on a cycle accurate simulator for the Cobham Gaisler Next Generation Multicore Processor (NGMP) with modifications such that all caches use random placement and random replacement (see Section 6). The CV-plots show that in some cases 2000 or 3000 observations are required, compared to the default of 1000. Comparison with empirical distributions for  $10^7$  runs (generated on a large compute cluster) show that the pWCET estimates are 1% to 25.8% larger than the maximum observed values at a probability of exceedance of  $10^{-7}$ .

In 2017, Milutinovic et al. [101] discussed issues relating to representativity with respect to the use of MBPTA in an industrial context. They argue, citing the above work of Abella et al. [3], that for real-time programs (with a finite but unknown WCET and non-degenerate behaviour) it is sound to fit a Gumbel distribution, since for this class of program using a Gumbel distribution may result in an over-estimate at probabilities of exceedance that are of interest, but not an under-estimate. Further, the authors point out, as was done by Griffin and Burns [51] in 2010, that taking observations from multiple paths as input into MBPTA (i.e. the per-program approach, see Section 2.3) may produce untrustworthy results. The problem being that the observations may no longer be identically distributed (particularly if the different paths are quite distinct modes of operation), and also the frequency of occurrence of the different paths may not match that which occurs during operation. The authors give a concrete example of this problem which shows that combining observations for two paths from a simplified European Train Control System can result in a pWCET distribution which is below the pWCET distribution for either path considered alone. (We note that no comparison is made to the empirical distribution, so it is unclear if any of the three pWCET distributions are actually under-estimates). The authors suggest that the per-path approach (see Section 2.3) can be used instead to provide a solution.

Also in 2017, Guet et al. [55] considered two issues relating to representativity. First, how traces of observations are handled when there are dependencies between execution times. In this case, the authors show that re-sampling the traces of observations can reduce the degree of dependency which is apparent and this may in turn result in lower pWCET estimates which can be unsound (i.e. optimistic). Secondly, they consider the use of EVT with multi-mode tasks, where different modes of operation have distinct execution time distributions. Here, they show via examples that combining all of the observations into a single sample (i.e. covering all modes) can result in cases where either EVT cannot be applied, or it can result in unsound pWCET estimates. They infer that it is necessary to apply EVT to each mode separately and then form an envelope upper bounding the pWCET distributions for each mode to provide an overall pWCET distribution that is sound.

#### 4.4 Summary and Perspectives

MBPTA methods have matured considerably since the early work of Burns and Edgar [22, 45]. In particular, we note the work of Cucu-Grosjean et al. [34] which spawned a large number of subsequent publications on supporting mechanisms and techniques (see section 6), and prompted further research into the application of EVT to the probabilistic timing analysis problem. The recent state-of-the-art has shown that time-randomised architectures (e.g. with random replacement caches) are neither sufficient nor necessary for the application of MBPTA methods [112, 88, 87]. Further, EVT can be applied when there are dependences between the observed execution times, provided that the series of observations exhibits stationarity and/or extremal independence [112, 16, 53, 54]. These are useful advances, since industry has a strong preference for methods that can be applied to Common-Off-The-Shelf (COTS) processors, rather than requiring specific (custom) hardware architectures. We note however, that there are significant hazards in applying MBPTA

methods. If potential sources of significant execution time variability do not exhibit this variability during analysis, i.e. such variation does not appear in the observations, then they will not be accounted for in the estimated pWCET distribution produced, which may as a result be optimistic (i.e. unsound).

A major open issue that remains is the problem of *representativity*. It is essential that the measurement protocol is representative of the future scenarios of operation that could occur in practice. In general this requires suitable coverage of the different input states, hardware states, and the worst-case path(s) through the program. While worst-case path coverage may be possible for relatively simple and well structured programs in domains such as aerospace, in general, the problem of identifying and exercising the worst-case path(s) cannot be left to the user. Research aiming at a solution to this problem is reviewed in Section 5.

The main rationale for using time-randomised hardware (e.g. random replacement caches, random permutation buses etc.) as well as software time-randomisation techniques (e.g. random placement) is to make it easier to provide an argument that the measurement protocol used during analysis is representative of the scenarios of operation that could occur in future. These enabling mechanisms and techniques are reviewed in Section 6.

We note that there remains significant scope for work on the application of EVT to the problem of estimating pWCET distributions. Currently, there are differing views on whether it is sufficient to assume that any program in a real-time system will have an execution time distribution which can be modelled as having a light or exponential tail, and hence a Gumbel distribution can be used as suggested by Abella et al. [3], or whether it is necessary to fit to a GEV distribution, since the execution time of some programs may exhibit heavy tails as shown by Lima et al. [88]. There has also been little work on the reproducibility of the method: whether small changes in the set of observations may propagate to large differences in the resulting pWCET distributions, and thus on how far the pWCET distributions can realistically be extrapolated to very low probabilities e.g.  $10^{-9}$ ,  $10^{-12}$ ,  $10^{-15}$  and so on, while retaining confidence in the results.

## 5 Hybrid Techniques for Probabilistic Timing Analysis (HyPTA)

A number of researchers have sought to combine the advantages of static analysis and measurement based methods. The main advantage of measurement-based methods is that measurements record the precise timing behaviour of the real system, whereas static analysis relies on a model of that behaviour. For advanced processors, obtaining a precise model or even one that does not introduce significant pessimism may be very difficult. The main disadvantage of measurement-based methods is the difficulty in covering the worst-case behaviour, including covering the worst-case path(s) through the code. The MBPTA methods based on EVT provide a pWCET distribution which estimates the extreme execution time behaviour of future scenarios of operation, sample(s) of which were exercised during an analysis phase. It is however up to the user to provide the necessary input vectors to test all relevant paths, including the worst-case path. Typically, neither manually determining the worst-case path, nor obtaining full path coverage are feasible in practice for complex programs. Further, as shown by Lesage et al. [85], the analysis rapidly becomes optimistic when some paths are omitted. In this section, we review hybrid methods which seek to address this problem of *path coverage*.

### 5.1 HyPTA and the Path Coverage Problem

In a series of papers from 2002–2005, Bernat et al. [19, 18, 17] addressed the problem of path coverage in measurement-based execution time analysis by reducing it to the much simpler problem of basic block (i.e. statement) coverage. In these works, the authors introduce the concept of

*Execution Time Profiles* (ETPs) used to represent the *relative frequencies* of execution times for basic blocks of a program. They also provide a timing scheme for constructs such as conditionals and loops, and an algebra for combining *independent* ETPs to provide an ETP for the whole program. For sequential combination, this involves using the basic convolution operator. Further, they point out the importance of accounting for dependences between basic blocks when combining ETPs otherwise unsound results could be obtained. Potential sources of dependence include: low-level hardware features such as caches and pipelines, and high-level path dependences. They suggest using biased convolution<sup>14</sup> to account for unknown dependences. (We note that Ivers and Ernst [64] later showed that biased convolution is insufficient to produce a distribution that results in the worst-case exceedance probability for every possible value of the execution time budget). A tool set is described by Bernat et al. [18] that implements the mechanisms described above. The main components are: instrumentation, trace generation, trace analysis, structural analysis, and timing program generation. This tool set was the precursor and prototype for the RapiTime tool from Rapita Systems<sup>15</sup>. Later in 2005, Bernat et al. [17] examined the problem of combining the ETPs for two basic blocks of code A and B when there are dependences between them. Here, the authors propose applying *copulas* (representing the dependences) to the marginal (i.e. separate) distributions of A and B to obtain the joint distribution.

In 2014, Kosmidis et al. [70] introduced the idea of Path Upper Bounding (PUB) for time-randomised hardware with a random replacement cache. With PUB, the program is modified so that it retains the same functionality, but has the same timing behaviour for every path, and that timing behaviour upper bounds that of the original program. PUB inspects each conditional in the program recursively and adds accesses such that all sub-paths of a particular conditional contain the same set of accesses. Once PUB has been applied, then the modified program is subject to MBPTA, with no particular attention needed to the input vectors used or the paths exercised, since all will result in a timing behaviour that upper bounds that of the original program. Evaluation, using the EEMBC benchmarks, shows that PUB increases code size by 20-100% depending on the amount of nesting and conditionals present, with the pWCET estimate at an appropriate probability of exceedance increased by up to 50%. We note that the application of PUB requires a trusted or certified implementation, since it modifies the code of the original program to create the version that is tested for timing correctness. Further, the method only works for systems that use a single-level random replacement cache.

The Extended Path Coverage (EPC) approach proposed in 2015 by Ziccardi et al. [136] also addresses the issue of path coverage via a hybrid approach. EPC requires that execution time measurements are made at the basic block level, while also recording the path taken, and that sufficient coverage is obtained to provide an Execution Time Profile (ETP) for each basic block. The ETPs are then modified to discount any benefit that may have accrued due to the path taken when the measurements were made. This is done by computing a probabilistic padding via the SPTA techniques developed by Altmeyer and Davis [7], considering the path actually taken and the memory accesses in the immediately preceding basic blocks. Synthetic end-to-end observations for all feasible paths can then be generated by randomly sampling the ETPs for the relevant basic blocks along each path. These synthetic observations are then fed into MBPTA. The authors evaluated EPC compared to directly applying MBPTA. For single path code there was no difference. For multi-path code with a known worst-case path, the EPC method resulted in pessimism of up to 30%. Comparisons with PUB [70] showed that EPC achieved on average

<sup>14</sup> Biased convolution combines values from two distributions assuming perfect correlation with respect to the percentile of the execution time, i.e. largest correlated with largest, smallest with smallest etc.

<sup>15</sup> <https://www.rapitasystems.com/products/rapitime>

similar performance with a  $\pm 30\%$  variation apparent between the two methods. EPC has the advantage with respect to directly applying MBPTA that it reduces the burden of path coverage, and the advantage over PUB that it does not require changes to be made to the executable code in order for measurements to be taken. EPC does however inherit some of the disadvantages of SPTA in that it needs to know the addresses of accesses to determine which cache sets they map to, and so compute the padding. Further, fine grained observations are needed at the basic block level, and accompanying structural analysis to re-construct the possible paths. EPC was developed for systems with separate, single level data and instruction caches, given the current lack of SPTA for multiple levels of cache (discussed in Section 3.4), it is not clear if the method could be extended to more complex memory hierarchies. Like PUB, EPC only applies to systems with random replacement caches.

## 5.2 Summary and Perspectives

A limited amount of research has aimed at addressing the path coverage problem via the use of hybrid methods [19, 70, 136]. However, these methods have some drawbacks, initial work by Bernat et al. [19] recognises the key issue of dealing with dependences between the measurements obtained for different basic blocks, but is unable to provide a practical approach for dealing with them. Later work on Path Upper Bounding (PUB) [70] requires substantial changes to the executable code, inflating code sizes by up to 100% and execution times by up to 50%, and requiring a trusted or certified implementation. Finally, the Extended Path Coverage (EPC) method [136] makes use of SPTA to compute a probabilistic padding to account for the dependences between the execution times of basic blocks due to the random replacement cache. EPC thus inherits some of the disadvantages of SPTA in that it needs to know the addresses of accesses to determine which cache sets they map to, and so compute the padding. Both PUB and EPC methods only work with random replacement caches.

In general, the issue of path coverage in relation to MBPTA remains unsolved. One approach for simple systems with few paths is to apply EVT on a per-path basis, i.e. to traces of observations taken for each specific path separately, thus producing a pWCET distribution for each path. The overall pWCET for the program can then be obtained as a tight upper bound on the pWCET distributions for all of the constituent paths. This *envelope* approach is well-suited to systems where there are a limited number of paths and the user can provide input vectors which exercise all of them. For many systems this is however unrealistic. The alternative is to apply EVT per-program i.e. to a sample of observations that cover all of the different paths. (Note, all paths need to be covered or the pWCET estimates can quickly become unsound as shown by Lesage et al. [85]). This approach raises difficulties in applying MBPTA, since the resulting observations will most likely form a mixture distribution (with the different distributions from the different paths contributing to the mixture), and thus care needs to be taken to ensure that sufficient observations are obtained from the tail of the distribution, i.e the worst-case path(s). This can be heavily impacted by the frequency of occurrence of different paths during analysis, and thus gives rise to issues regarding representativity. Further research is needed in this area.

## 6 Enabling Mechanisms and Techniques

MBPTA methods [34] based on EVT work well when the observations are able to characterise the different sources of execution time variability and their probability of occurrence. As an example, consider a program running on hypothetical time-randomised hardware where each instruction takes an random amount of time to execute, from 1 to 6 cycles, independent of all of the other instructions. Assume the program has 10 instructions. The overall execution time will behave

like the sum of the values of 10 fair dice. While the worst-case (i.e. 10 sixes) is unlikely to be observed (probability  $\approx 10^{-8}$ ), EVT is able to estimate the tail of the distribution from a relatively small number of observations (e.g. 1000). Contrast, this with a hypothetical time-predictable system, here the execution time might depend on say 10 different input variables (each with values from 1-6). Let us assume that there is some small variability in overall execution time which depends on the input values themselves e.g. the overall execution time is 10, 20, 30, 40, 50, or 60; however, if all 10 inputs take a specific combination of values then there is some conflict in the hardware which slows processing down and the execution time is 600. The probability of observing this via random testing of the inputs is  $\approx 10^{-7}$ . In this case, applying EVT on 1000 observations would most likely result in a prediction based on the small variability that has been observed, but would obviously not account for the much larger variability that has gone unobserved. As a result, the pWCET estimate would be unsound. This is an example of a *needle-in-a-haystack* problem in testing. The intent of using time-randomised hardware is to avoid hazards similar to the one described above. These can potentially occur with time-predictable hardware such as LRU caches, when the unlikely, but not impossible, scenario of multiple input-data dependent accesses mapping to the same cache set occurs. We note that such hazards, referred to as *cache risk patterns*, can also occur with time-randomisation techniques such as random placement (see Section 6.3).

Since the work of Cucu-Grosjean et al. [34], considerable efforts have been expended to support MBPTA methods via the use of additional hardware and software time-randomisation mechanisms. These include hardware and software random placement for set-associative random replacement caches, random permutation buses, degraded test modes, and better random number generators. In this section, we review the supporting work in these areas.

## 6.1 Caches and Hardware Random Placement

Fully-associative caches are slower and use more energy than set-associative caches of the same overall size but a much smaller number of ways. This is due to the extra logic needed to check if a memory block is in any, as opposed to a small number of, cache lines. Because of this, in practice caches tend to either be direct mapped or to have 2, 4, 8 or 16 ways. The idea of using a random placement policy is to provide more randomisation with a set-associative random replacement cache than is possible with modulo placement, while improving access times and energy consumption compared to a fully-associative random replacement cache. Random placement can be achieved in hardware (via a randomised hash function used to control the mapping of memory blocks to the cache) or via software (through the use of compiler techniques and runtime support that randomise the positioning of code and data in memory). In each case, a random placement is selected at the start of each run of the program and remains in place for the duration of that run. The cache is then flushed between runs ready for a different random placement to be used on the next run.

The idea of random placement has its origins in the 1990s. In 1993, Schlansker et al. [115] first proposed random placement as a means of improving the cache miss ratio of matrix operations. In 1999, Topham and Gonzalez [124] proposed the use of polynomial modulus functions (operating on memory addresses) as a means of pseudo-randomising cache placement, in order to reduce the number of conflicts. We note that while these methods can avoid systematic conflicts due to particular code structures, they provide a deterministic mapping that depends on the memory addresses, and thus always produce the same placement for a given memory layout. (The placement does not change on each run of the program).

The majority of the work on hardware random placement for random replacement caches was developed in a series of papers from 2013–2016 by Kosmidis and co-authors including Abella, Cazorla, Quinones, and later Hernandez [68, 69, 67, 61].

In 2013, Kosmidis et al. [68] proposed the use of a hardware random placement policy as part of the cache design. The parametric hash function used aims to avoid systematic collision of two memory blocks in the same cache set. The evaluation considers random placement applied on top of set-associative random replacement caches, thus it is difficult to quantify the contribution of random placement versus random replacement. Where these factors can be separated e.g. for a 1 way – 256 set (i.e. direct mapped) cache with random placement, and a 256 way – 1 set (i.e. fully associative) random replacement cache, the results show that random placement results in a factor of 2.6 degradation in execution time performance (measured in terms of the average number of instructions per cycle) compared to modulo placement. By comparison, in the fully associative case, random replacement shows approximately 10% degradation in performance compared to LRU. In a journal extension in 2014, Kosmidis et al. [67] provided a detailed analysis of the quality of the hash function used for random placement, as well as more detailed evaluation results, including an investigation into energy consumption and cache access times.

As part of an analysis for caches with random placement and random replacement, Kosmidis et al. [68] gave formulae for the cache hit probability of a given access in a set-associative random replacement cache; however, as shown by Davis et al. [41], this formula cannot be used in SPTA since it may result in a pWCET distribution that is optimistic (i.e. unsound). Similarly, the formulae given for the cache hit probability assuming caches using random placement, or both random placement and random replacement also cannot be used in SPTA.

In a number of works, Kosmidis et al. [68, 74, 67]) describe the concept of an Execution Time Profile (ETP) as defining the different execution times of a program (or instruction) and their associated probabilities. They state that *“the existence of an ETP ensures that each potential execution time of the program (for MBPTA) or instruction (for SPTA) have an actual probability of occurrence, which is a sufficient and necessary condition to achieve the desired probabilistic i.i.d. execution time behaviour”* so that MBPTA can be applied. However, this is not entirely correct. The argument given in more detail in a white paper by Abella et al. [1] shows that the time-randomised architecture proposed ensures the existence of an ETP for a single path through the program starting from a fixed initial software and hardware state. Thus the observations *for that path and initial state* will be *independent and identically distributed* (i.i.d.). However, for a complete program, with multiple paths, this does not necessarily mean that the execution time observations collected over multiple paths for use in MBPTA will be i.i.d. As a simple example, consider a program that implements a state machine with states 1-5 that it cycles through in order. Each state may have an execution time that is quite different from the others e.g.  $100 \pm 20$ ,  $200 \pm 20$ ,  $\dots$   $500 \pm 20$ , where  $\pm 20$  represents the random variation and the first value is determined by the state variable. Now when multiple runs of the program are performed, it cycles through the states, and hence the execution times observed are *not independent*, rather there is a strong correlation between them with a lag of 5. In cases such as this MBPTA may or may not be applicable; appropriate statistical tests are needed to determine if the execution times observed are actually i.i.d., as there is no relation between the statistical properties of the execution times of an instruction and the statistical properties of the execution times of an entire program containing that instruction. The architecture alone cannot ensure that the execution times will be i.i.d. for all real-time programs.

Later in 2013, Kosmidis et al. [69] applied MBPTA [34] to a system with multiple levels of cache that use random replacement along with a random placement policy implemented in hardware [68]. Here, separate L1 instruction and data caches are assumed, with a unified L2 cache. Assuming latencies for the L2 cache and memory of 10 and 100 cycles respectively, the evaluation shows that using a second level of cache improves the pWCET estimate at an appropriate probability of exceedance for the EEMBC benchmark code by 10-90% depending on the size of the working set.

The authors also give a series of formulae that approximate the probabilities of cache misses for different cache arrangements. These formulae relate to those from earlier papers [68, 67], which can be optimistic by orders of magnitude as shown by Davis et al. [41].

Building on the work of Kosmidis et al. [68], in 2014 Slijepcevic et al. [120] considered a time-randomised (i.e. both random placement and random replacement) last-level cache that is shared between cores in a multi-core system. The basic idea proposed is that a shared time-randomised cache makes the cache interference suffered by a task due to co-runners depend only on the frequency of co-runner cache misses (i.e. evictions) and not on the precise cache lines that are accessed. The authors present a hardware mechanism that limits the eviction frequency by enforcing a minimum inter-eviction delay for each core. The pWCET distribution of each task is then estimated at analysis time with this mechanism limiting its rate of cache misses (i.e. stalling execution if two misses would otherwise occur too close together), and synthetic co-runners causing the maximum possible rate of evictions (i.e. separated by exactly the minimum inter-eviction delay). To avoid problems due to systematic interleaving, the minimum inter-eviction delay is itself set to a uniform random value with the desired mean. Cache sharing with the proposed mechanism limiting contention is compared to cache partitioning into 4 partitions of 2 ways each. The latter results in estimated pWCETs at an appropriate probability of exceedance that are on average 56% higher.

In 2015, Anwar et al. [9] developed a VHDL implementation of random replacement and the hardware random placement policy proposed by Kosmidis et al. [68] for instruction and data caches, and integrated it with an Ion MIPS32 processor core on an FPGA. They used the Mersenne Twister algorithm, which is considered a good hardware solution for random number generation, replacing the Multiply With Carry random number generator previously proposed [68]. For a set of benchmarks operating on arrays and matrices, with nested loops, the authors claim that the time-randomised hardware gives an improvement of 6% and 19% over LRU for 8-way and direct mapped caches respectively, when comparing observed execution times. We note; however, that these figures use the worst-case execution times observed in 30 runs with the time-randomised hardware; whereas the predicted pWCET at an exceedance probability of  $10^{-3}$  is consistently larger than the measurements shown for LRU.

An improved hardware random placement policy called *random modulo* was described by Hernandez et al. [61] in 2016. With the parametric hash function for random placement introduced by Kosmidis et al. [68] there is a finite probability that adjacent memory blocks will be mapped to the same cache set in some placements and thus conflict with each other. This degrades performance, particularly for an instruction cache with respect to loop constructs that span a number of adjacent memory blocks. The random modulo approach seeks to avoid this problem. It effectively creates a random permutation which maps from addresses in a memory segment (the same size as a cache way) to cache sets. This mapping retains the property of modulo placement, whereby memory blocks separated by less than the size of a cache way will not conflict in the cache. The hardware implementation also has a number of advantages over the parametric hash function method of Kosmidis et al. [68]; it uses less than 10% of the silicon area and can operate at a higher frequency. The evaluation of random modulo for instruction and data caches shows that it provides a significant reduction in the pWCET estimate at an appropriate exceedance probability, compared to the parametric hash function, with an advantage of 25-62% across the different EEMBC benchmarks studied.

Later in 2016, Trillia et al. [127] improved the resilience of caches implementing the random modulo random placement policy described by Hernandez et al. [61]. They note that the original form of this policy does not result in an even distribution in terms of how often the cache lines are used. This is due to the fact that the index bits that are used are entirely dependent on the access

pattern of the program. By the simple expedient of XORing part of the selected random seed with these index bits, homogeneity of cache line use can be achieved. This has the desirable effect of making the cache more reliable, since one of the main sources of transistor degradation (called Hot Carrier Injection) is proportional to the amount of use. The additional XOR gate has no effect on the maximum operating frequency of the cache, since it is not on the critical path.

In 2018, Benedicte et al. [11] considered the use of random replacement policies in multi-level caches. They show that performance, in terms of pWCET estimates at an appropriate probability of exceedance, can be improved by using different policies in the L1 and L2 caches. They explore the use of random modulo placement [61] in the L1 cache and the use of a parametric hash function [68] across L2 cache segments combined with either modulo or random modulo placement within L2 cache segments. These approaches provide an improvement in performance of approx. 30% compared to using parametric hash functions at both levels.

## 6.2 Caches and Software Random Placement

The majority of the work on software random placement for random replacement caches was also developed in a series of papers from 2013–2016 by Kosmidis and co-authors including Abella, Cazorla, and Quinones [72, 73, 71, 78].

In 2013, Kosmidis et al. [72] proposed the use of compiler techniques and runtime support to randomise the layout of both code and data in memory; effectively providing a software means of random placement applicable to hardware with direct mapped caches or set-associative caches using LRU replacement. The code and data layout in memory is changed before the start of each run of the program. Due to the deterministic mapping to cache, this has the effect of randomising the cache lines at which code and data objects start in the cache, thus randomising conflicts between objects, but not within them. Evaluation on examples from the EEMBC benchmark suite produce observations of execution times that pass the i.i.d. tests, thus allowing MBPTA to be applied. Software random placement does however have a number of drawbacks. The results show that the overheads increase the pWCET estimate at an appropriate exceedance probability by a factor of 10 for code that contains a loop that is repeated 100 times, and by a factor of 2 if the loop repeats 1000 times. We note that re-arranging code and data objects at runtime may be unacceptable in many industry sectors, such as automotive and aerospace. Such a re-arrangement means that deployed systems could run code and data layouts that have *never* been tested. This may reveal some subtle bug in functionality which was dormant in tested configurations, and could be very difficult to reproduce.

Subsequently, in 2014 Kosmidis et al. [73] acknowledged issues with their software random placement scheme [72]. This scheme conflicts with the design principles of the ISO26262 standard for the development of automotive software: “(i) *limited use of pointers*, (ii) *recommendations against the use of dynamic objects*, and (iii) *no hidden data flow or control flow*”. The authors therefore proposed an alternative: *Static Software Randomisation*. Here, the idea is to create a series of binaries at compile time that have locations for functions and data such that their mapping to cache follows a random selection. The method chooses random offsets for each function and data object from the start of the cache. It then finds a suitable ordering of the functions in memory, with some additional spacing to achieve the desired mapping to cache via modulo placement without wasting too much memory. The locations of stack frames and global variables are similarly randomised at compile time. The authors propose creating  $N$  binaries, running each one once, measuring its end-to-end execution time, and using this data as input into MBPTA. They assume that the estimated pWCET distribution so obtained will apply to all of the binaries. Then at deployment they propose either (i) a different binary is deployed in each production unit, or (ii) a single binary is chosen and deployed in all production units. They note that (i) may

not be acceptable if each individual unit is not fully tested. The authors claim that the pWCET distributions obtained from their scheme will be valid for case (ii), however, this is *not* correct. For the MBPTA method based on EVT to give sound results, it is necessary that the observations made in the analysis phase are either directly representative of those that could occur during deployment of the system, or they upper bound a set of values that are representative of those that could occur during deployment. Neither is the case when observations are taken from different binaries, which are effectively different systems to the one deployed. In this case, the observations are not identically distributed, but rather they come from the different distributions associated with each of the different binaries.

In 2016, Kosmidis et al. [71] investigated the use of static software randomisation applied to software from an automotive cruise control system running on a single core of an AURIX multi-core system. Dynamic software randomisation is not possible for this system, since the AURIX platform, in common with many used in automotive systems, does not permit self-modifying code. Instead, the authors use static software randomisation where program code, stack and global data are allocated random locations in memory across different binaries. A pool of different binaries are used during the analysis phase to obtain observations, with a single binary selected for deployment. The authors claim this allows EVT-based methods to be applied; however, as discussed above, this approach is not valid. In particular, the “overall system” used during analysis, which operates by selecting and then running a binary to obtain a single observation, is *not* the same as the system used during operation, which runs the same binary every time. Thus the execution time observations made at the analysis stage are not identically distributed with respect to those obtained during deployment, and hence the results of applying EVT-based methods are invalid in this case. Unfortunately, the authors misunderstand the point of applying tests on the observations to determine if they are independent and identically distributed (i.i.d.). They apply these tests on observations from 1000 different binaries used during the analysis phase and then claim that a positive result enables the use of EVT. This may be the case if the deployed system were the same as the one used during analysis (i.e. it switched binary every run), but it is not.

A different approach to software random placement called TASA *Tool-Chain Agnostic Static Software randomisation Approach* was proposed by Kosmidis et al. [78] in 2016. This approach randomises the location in memory at which different objects defined in the source code are placed. To do so, TASA relies on the fact that the compiler generates code and data in the order in which they appear in the source files. Thus it adds functionally neutral padding code and data and re-orders the declarations of variables and functions to achieve a degree of randomisation. Stack frames are also randomised by adding a randomly sized array to the list of local variables and also by re-ordering those variables. Similarly, the members of compound structures are also shuffled. TASA has the advantage over previous efforts at software randomisation in that it operates at the source code level and is thus portable, depending only on the programming language. The TASA approach relies on creating  $N$  binaries to use in the analysis stage (collecting end-to-end measurements for input into MBPTA) and then deploying a single binary for which it is assumed the pWCET distribution derived via EVT will apply. Unfortunately, as discussed above this logic is faulty. The  $N$  binaries represent different systems from that which is deployed, and hence the results from applying EVT are not valid for the deployed system.

### 6.3 Cache Risk Patterns with Random Placement

A significant issue with both hardware and software random placement is that some randomly chosen placements may result in a *cache risk pattern* [106] whereby a number of accesses within a loop conflict in the cache, thus resulting in a large increase in the execution time for that particular configuration. Further, the probability of such a pattern occurring can be below that which might

reasonably be observed in the limited number of execution time observations used in MBPTA, but above the probability threshold (e.g.  $10^{-9}$ ) at which such long execution times can safely be ignored. Thus random placement can create a needle-in-a-haystack problem, making the results provided by MBPTA unreliable.

Research aimed at addressing the above problems was developed predominantly by the same group of researchers who investigated software and hardware random replacement methods (see Sections 6.1 and 6.2, including Abella, Benedicte, Cazorla, Kosmidis, and later Milutinovic [4, 13, 14, 98, 99, 97].

In 2014, Abella et al. [4] identified the above issue with the random placement scheme of Kosmidis et al. [68] and suggested taking observations using a smaller cache as a way of revealing cache risk patterns by making them more likely to occur in the limited number of runs employed by MBPTA. Analysis is given which aims to compute the reduction in the size of the cache needed to achieve this. This analysis makes an assumption that a cache risk pattern for a given placement will always be observed in a run that uses that placement; however, unless the code is single path, or the loop is executed on every path, then this assumption may not hold. Different runs may exercise different paths, only a few of which may cause the loop with the cache risk pattern to execute. Hence the reduction in the size of the cache necessary to achieve sound results may be substantially greater than that computed using the formulation given in the paper. We note that the analysis provided by Abella et al. [4] assumes that each unique address is re-mapped independently by the random placement mechanism. While that may be the case using a hardware approach to random placement [68], with software random placement schemes [72] code and data are subject to random placement only at the level of functions and objects. It is not clear if the method proposed by Abella et al. [4] can be applied in that case; however, it is apparent that the issue of unobserved large variations in execution time also exists with software random placement schemes.

In 2016, Benedicte et al. [13] studied the problem of cache risk patterns with hardware random placement. In particular, they review the analysis given by Abella et al. [4] which determines the probability  $P_{eoi}$  of a particular event of interest (i.e. a cache risk pattern) occurring based on an approximation using *weak compositions theory*. The authors present a precise calculation of  $P_{eoi}$  using an approach based on multinomial coefficients. This calculation shows that the value of  $P_{eoi}$  computed by Abella et al. [4] using weak compositions theory may over-estimate the precise value. Such an over-estimate leads to an *under-estimate* of the number of runs required to have confidence that the event will occur during the runs used to collect execution time observations for input into EVT, thus undermining the soundness of the estimated pWCET distribution. Calculation of the precise value for  $P_{eoi}$  takes significantly longer, and may become intractable due to the need to enumerate all possible cache allocations of interest. The method is only valid for programs with “*homogeneously accessed objects*” in other words, programs where every object (e.g. instruction) is accessed the same number of times, i.e. in one single outer control loop with no conditional branches.

The issue of cache risk patterns due to software random placement schemes was also addressed by Benedicte et al. [14] in 2016. The authors note that unlike hardware random placement, software random placement works at the level of functions and objects and thus the probability that a particular object is allocated to a given cache set is dependent on the allocation of previous objects. They also note that determining a precise model of the probability  $P_{eoi}$  of a particular event of interest (a cache risk pattern) is intractable. Given a number of objects and their sizes, Monte-Carlo simulation is therefore used to estimate  $P_{eoi}$ , and hence whether it is greater than the required threshold e.g.  $10^{-9}$ , but nevertheless too low to have confidence that the event will occur during the runs used to collect execution time observations for input into EVT. If so, the

number of runs  $R$  is computed such that the probability that the event will not be seen in  $R$  runs is less than  $P_{conf} = 10^{-9}$ . This is given by the formula  $R \geq \log(P_{conf}) / \log(1 - P_{eoi})$ . The evaluation shows that when the number of objects is in the range  $[5 - 60]$  (depending on cache and object sizes), then  $P_{eoi}$  can fall into the problematic range requiring an increase in the number of runs. Some examples are given showing that with  $P_{eoi} = 0.085$  then the number of runs needs to be increased to approx 2500. However, we note that for smaller values of  $P_{eoi}$  e.g.  $10^{-6}$  which also appears in the evaluation figures for a smaller number of objects, then the number of runs required such that the probability of not observing the event is less than  $10^{-9}$  becomes very large e.g. approx.  $2.10^7$ , which is unlikely to be attainable in practice. Aside from the potential for requiring very large numbers of runs, the main drawback of this approach is that, as the authors note, it only works for programs with homogeneously accessed objects. This severely restricts the use of the method when realistic programs, e.g. with functions within conditionals and nested loops, are considered.

In 2016, Milutinovic et al. [98] addressed an issue with the approach of Abella et al. [4] to identifying cache risk patterns caused by random placement that have a very low probability of being captured in the observations used in MBPTA. They showed that the approach of Abella et al. [4] *"only works when the impact on execution time of mapping any subset, bigger than  $W$  (the number of cache ways) is the same"*. This is only the case if all of the accesses are made in a round-robin fashion, for example single path code within a single large loop; however, this is not the case for programs in general which contain conditionals and other constructs. The method proposed by the authors aims to solve this problem. It considers all  $\binom{U}{x}$  combinations of  $x$  out of the  $U$  different memory block accesses in the program, where  $x$  is varied in the range  $[W + 1, U]$ . The probability of a given combination of  $x$  accesses mapping to the same cache set is computed and if this is found to be in the range of interest, then cache simulations are performed with the given combination of  $x$  accesses mapped to the same cache set and other accesses mapped at random. The simulation runs are used to determine an average miss count for the combination. This information is used to obtain a set of  $(miss\_count, probability)$  pairs indicating the number of misses and their probability, derived from all of the combinations simulated. These points are plotted on a graph and compared with the probabilistic Worst-Case Miss Count (pWCMC) distribution obtained using MBPTA. If the pWCMC distribution does not upper bound all of the  $(miss\_count, probability)$  points, then the number of observations used in MBPTA is increased until it does. The complexity of the method makes it intractable for practical programs with large numbers of memory block accesses due to the number of different combinations involved. For example 100 distinct memory block accesses and a 4-way cache would require cases where 5 address mapped to the same cache set are considered, of which there are  $\binom{100}{5} \approx 7.6 \times 10^7$  combinations. The authors attempt to deal with this problem by limiting  $U$  to the 15 most heavily accessed memory blocks. In the EEMBC benchmarks used for evaluation, this restriction means that approximately two thirds of the accesses are covered. No argument is given explaining why this is sufficient to ensure that the results obtained are sound.

In an extension [99] published in 2017, to their previous work [98] from 2016, Milutinovic et al. gave an example showing how the original [4] and improved [13] analysis of cache risk patterns, given by Abella et al. and Benedicte et al. respectively, fails to provide trustworthy results in cases when accesses are made in a non-homogeneous way. Evaluation using the EEMBC benchmark suite shows that for all of the benchmarks considered, these works under-estimate the number of observations required by MBPTA leading to results (i.e. pWCET distributions) that are untrustworthy below a probability of exceedance that is in the range 0.9 to 0.001 depending on the benchmark. (By contrast, sound results would be trustworthy down to at least a probability of exceedance of  $10^{-9}$ ). Further, they also evaluate issues of trustworthiness with the earlier

approach of Milutinovic et al. [98]. They show that issues can occur for simple cases where there are more addresses accessed in a loop than are considered by the analysis. Comparing the results for  $U = 10$  to those for  $U = 15$  on the EEBMC benchmarks shows that limiting the number of addresses considered can lead to either an over- or an under-estimation of the number of observations required by MBPTA, with under-estimation (occurring in 3 out of the 8 benchmarks) leading to untrustworthy results at the required probability of exceedance ( $10^{-9}$ ).

Prior to the above works, many papers were published assuming random placement, but as far as we can tell they did not check that the number of observations made was sufficient to avoid the hazards of cache risk patterns described above. The validity of the evaluation results for systems using random placement in the following papers is therefore questionable: [68, 72, 69, 73, 67, 71, 119, 129, 128, 60, 61, 136].

Subsequently in 2017, Milutinovic et al. [97] recognised some of the problems with their previous work [98, 99] on identifying cache risk patterns caused by random placement that have a very low probability of being captured in the observations used in MBPTA. In particular, they note that “*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*”. For  $U = 15$  different addresses, exhaustive evaluation via cache simulation is shown to require 27 hours per benchmark on a compute cluster running 100 jobs in parallel. To address this problem the authors propose a Time-aware Address Conflict (TAC) approach which aims to identify a list of address combinations that if mapped to the same cache set can result in a high miss count. The basic method is the same as that described by Milutinovic et al. [98, 99], but rather than examining an exhaustive list of conflicting address combinations, only a limited number (i.e. 20) for each number  $K$  of conflicting addresses are considered. Values of  $K$  are examined from  $W + 1$  upwards, where  $W$  is the number of cache ways, stopping when the probability of  $K$  addresses mapping to the same cache set is below the cutoff probability. The address combinations on the TAC list are ranked according to the concept of *guilt*, which aims to identify those combinations of addresses which are likely to result in high cache miss counts. Guilt is a heuristic estimate formulated via an expression which reflects the re-use distance of each address access. This is used to populate an *address guilt matrix* which aims to capture the extent to which misses on accesses to some address  $A$  are caused by accesses to some other address  $B$ . The TAC approach uses the information in the address guilt matrix to determine a ranking for different combinations of address accesses. A small number of the top ranked combinations are then evaluated via cache simulation. This reduction in the number of combinations considered improves the runtime by orders of magnitude compared to the exhaustive approach. The approach is shown to be effective, giving the same results as the exhaustive approach when the number of different addresses considered is limited to 15. Further evaluation on a railway case study considers 10 different test cases defined by specific input vectors, with the address traces used as input into the method. It appears that the approach is limited in its applicability to traces of addresses and thus to single paths through a program. MBPTA can be applied on a per-path basis giving a result that is valid for a single path, and can also be applied on a per-program basis. It is not clear how, or even if TAC would work on a per-program basis. We note that the *guilt* metric is a heuristic, and there is no proof given that it is guaranteed to always ensure that the address combinations with the highest likelihood of resulting in cache misses will be examined. Thus the improvement in runtime efficiency appears to come at a cost in terms of reduced confidence that the resulting pWCET distribution is valid.

In 2018, Milutinovic et al. [100] considered the problem of cache risk patterns due to random placement policies when applying the Path Upper Bounding (PUB) technique of Kosmidis et al. [70] (see Section 5.1). Recall that PUB pads the code with extra accesses in such a way that the pWCET distribution for any path through the modified code upper bounds the pWCET

distributions for all paths through the original code. Thus only one arbitrary path through the modified code needs to be exercised to obtain a sound pWCET estimate. The authors apply the TAC approach [97] (discussed above) to the modified code produced by PUB. They show that applying PUB can make cache risk patterns due to random placement either more or less likely to occur, and may in some cases significantly increase the number of observations required by MBPTA to obtain sound estimates of the pWCET distribution (e.g. from 3000 to 500,000). They give examples where cache risk patterns lead to an under-estimate of the number of observations required when using PUB alone, resulting in an estimated pWCET distribution which is unsound compared to the empirical distribution obtained by taking a large number of observations. The use of TAC is intended to address this problem.

## 6.4 Buffers, Buses and other Resources

As well as random replacement caches with/without random placement, other resources have been investigated and adapted to provide randomised behaviour, with the aim of making systems more amenable to being analysed using MBPTA techniques. The majority of the work in this area was published by a group of researchers including Slijepcevic, Cazorla, Kosmidis, Jalle, Abella, Hernandez, and Quinones from 2013 – 2016 [119, 27, 74, 65, 25, 60].

In 2013, Slijepcevic et al. [119] proposed a *Degraded Test Mode* (DTM) which when combined with fault tolerant random replacement caches enables MBPTA to be used to obtain pWCET distributions which estimate the execution time behaviour of the system in the presence of a given number of hardware faults that cause some cache lines to become unavailable. The motivation for this work is that although test methods can verify that processors do not contain faults when first deployed, degradation can cause latent defects to manifest into permanent faults. Hardware mechanisms can address such failures to some extent, e.g. by disabling cache lines when a faulty bit is detected. The rate at which this occurs depends on the technology scale used. With random placement and a set-associative random replacement cache or a fully-associative random replacement cache, the authors argue that there is little dependence on the actual cache lines which are faulty. Thus the degraded test mode configures the cache to have a number of lines disabled, commensurate with the fault probability per bit over the required lifetime of the system. MBPTA can then be applied to the system in degraded mode. Due to a lack of dependence on the location of any failed lines, the resulting pWCET distribution is valid for any set of bit failures in the cache lines that could be expected during the lifetime of the system.

Also in 2013, Cazorla et al. [27] looked at the requirements placed on the *analysis phase* runs of a program on a hardware platform, with the aim of ensuring that the observations of execution times are representative of or upper bound those for any population of such values that might be obtained during operation. As discussed in Section 4 *representativity* is necessary for the estimated pWCET distribution derived via MBPTA to be valid for future scenarios of operation. The authors reinforce the point made by Cucu-Grosjean et al. [34] in 2012 that adequate path coverage is required to ensure that observations from the worst-case path are included. The authors classify a number of different sources of execution time variation that need to be controlled for. For example, division operations may take a variable time dependent on the operands. Such variation could potentially be controlled for by using worst-case values, or in hardware via a worst-case mode, where such operations always take their worst-case time to execute. Code and data placement in memory typically affect the mapping to cache, which impacts execution times. Here, the authors suggest using random placement techniques to ensure representativity; however, as discussed in Section 6.3, random replacement also has significant problems relating to representativity due to cache risk patterns.

In 2014, Kosmidis et al. [74] aimed to set out the properties or architectural features that a

processor needs to have to *guarantee* that the MBPTA method [34] can be applied. They classify hardware resources into either *jitterless*, having no execution time variability, or *jittery* resources. Jittery resources may result in latencies that depend on the execution history or the input values or a combination of the two. They propose that jittery resources should either be assumed to always take their worst-case latency or be time-randomised. In the evaluation, the authors apply MBPTA to programs running on a time-randomised architecture with both instruction and data caches using random placement and random replacement. They use the statistical tests described by Cucu-Grosjean et al. [34] to check if the execution time observations are i.i.d., and this was shown to be the case for the EEMBC benchmarks used. The authors claim that “*Both tests are passed in all cases, which proves that the example architecture meets the i.i.d. requirement.*”. This claim of *proof* is in our view extended too far, rather the experiments show that the observations are i.i.d., but *only* for the particular instances of those benchmarks on the given architecture. There is no evidence that this would necessarily be the case for any program on the given architecture. Later work by Lima et al. [88] (discussed in Section 4.3) shows that using a random replacement cache is not sufficient to guarantee that observations are i.i.d., neither does an LRU cache necessarily preclude it. We note that in general any hardware resources or software variables that preserve state between runs of a program could potentially lead to dependences between execution time observations, breaking the independence property.

A *random permutation* bus was proposed by Jalle et al. [65] in 2014 as a means of connecting cores and memory in a multi-core system. With a random permutation bus, the bus arbiter produces a new random permutation (order for contenders to access the bus) every  $N$  rounds, where  $N$  is the number of contenders. The random permutation bus is compared to a lottery bus (introduced by Lahiri et al. [79] in 2001) that makes a random selection of which contender gains access to the bus on each round, and a conventional Round-Robin bus. Applying MBPTA, the authors show that the pWCET estimate at an appropriate exceedance probability is reduced by between 1.5% and 9.6% for a random permutation bus as compared to a Round-Robin bus. In the latter case, an assumption is made that every access incurs the worst-case delay.

In 2015, Panic et al. [102] showed how systems that incorporate buses and other resources with TDMA arbitration may be analysed using MBPTA. The key idea is that the largest difference between execution times that can be caused by a misalignment of any number of synchronous (blocking) requests with the TDMA cycle is  $w - 1$  where  $w$  is the length of the overall cycle as shown by Kelter et al. [66]. The authors extend this result to multiple TDMA resources showing that it generalises to  $LCM(w_1, w_2, \dots) - 1$ . Given these results, the simple expedient of analysing the system using MBPTA and then padding the results by adding the largest difference that could be caused by misalignment with the TDMA cycle, results in an upper bound. Since the padding is typically small compared to overall execution times, this method is effective and provides superior performance to the random permutation bus [65] or forcing all accesses to take the worst-case time. An extended version of this work was published in 2017 by Panic et al. [103], adding a discussion of the impact of timing anomalies.

The behaviour of random replacement caches, random permutation bus arbitration, and other hardware components with time-randomised behaviour depends on the quality of the underlying random number generator. In 2015, Agirre et al. [5] described a Pseudo-Random Number Generator (PRNG) designed to provide the random numbers needed to implement these hardware components. The proposed PRNG uses a Linear-Feedback-Shift-Register (LFSR) design, which provides a long period ( $> 2^{60}$ ) for the random number sequence. The LFSR design is modified to produce a 32-bit pseudo random number on each cycle, groups of bits from which are then used as the random numbers required for different components (e.g. instruction and data cache etc.). The authors discuss a number of ways in which the basic design can be hardened to meet

the safety requirements of IEC-61508 SIL 3. These include duplicating the PRNG and checking via a voter that the outputs match, and also using a watchdog timer to check that new values have been produced. They report that the PRNG passes 187 out of the 188 tests specified by the US National Institute of Standards and Technology for assessing randomness properties. Failing only the Linear Complexity test which is used to determine if the sequence produced is complex enough to be considered random [109]. The evaluation shows that the observed execution times for the EEMBC matrix benchmark failed the i.i.d. tests used as part of the MBPTA method when obtained from a LEON 3 processor prototyped on an FPGA using the default random policy, but passed those tests when the LFSR PRNG implementation was used.

In 2015, Hernandez et al. [60] took a preliminary look at the changes potentially needed to the LEON3 multi-core processor in order to support the use of MBPTA. They identify a number of points: The existing random replacement policy, which can be configured for the caches, uses a randomisation method that is not of sufficient quality. It is replaced by the PRNG described by Agirre et al. [5]. Further, hardware enabled random placement is implemented using the method based on a parametric hash function described by Kosmidis et al. [67]. The core-to-L2 and L2 cache are modified so that requests from one core cannot cause stalls for another core. Finally, the bus and memory controller arbitration policies are round-robin and FIFO respectively. The authors suggest that these need to be modified to be either random permutation [65] or lottery bus [79]; however, this change is not made, instead benchmarks are run on only one core thus avoiding issues of contention. The brief evaluation shows that the 1000 execution time observations used for each of the two programs vary by less than 0.001% for *rspeed*, and by 0.04% for the matrix benchmark. This tiny variation is due to the fact that the benchmarks fit comfortably in the cache. The authors draw no conclusions, but suggest that they will in future study further benchmarks with different cache requirements.

In 2016, Cazorla et al. [25] and Kosmidis et al. [76] summarised the research and development work on support mechanisms for the MBPTA method [34] developed during the EU PROXIMA project and described in previous papers.

Also in 2016, Slijepcevic et al. [117] proposed a tree-based Network-on-Chip (NoC) for a many-core system, adapted with the aim of permitting MBPTA of tasks running on the cores. The tree-based NoC uses either Round-Robin (with a worst-case mode enabled for analysis), Lottery, or Random Permutation methods for arbitration at each level in the tree. The evaluation shows that the tree-based NoC provides higher performance than a bus for clusters of 8 or 16 cores. Slijepcevic et al. followed this work with a further paper [118] in 2017 on the use of Random Permutation methods in the routers of a wormhole NoC. The aim being to avoid the systematic worst-case behaviour which has to be accounted for in the analysis of worst-case traversal times when deterministic arbitration policies are used. Applying MBPTA, and using a probability of exceedance of  $10^{-13}$ , the authors show that the pWCET estimates improve on the WCETs for an equivalent NoC with deterministic routing by on average 22% to 75% for 3x3 and 6x6 NoCs respectively. (This assumes that the number of in-flight requests are limited, a technique that improves the analysed performance in the randomised case, but not in the deterministic case).

An alternative approach to implementing random replacement caches was proposed by Benedicte et al. [12] in 2018. Recall that on a cache miss, the conventional random replacement policy selects at random any one of the cache lines in the cache set for replacement. With  $W$  ways this means that on a cache miss the probability of eviction is  $1/W$  for each cache line in the set. Although this form of randomisation is effective in supporting MBPTA, it is also inefficient. For example, alternate accesses to two addresses that map to the same cache set may cause mutual evictions even in the case where there are 4-ways available. The authors propose a form of random permutation to avoid this problem and thus improve performance. With this Random

Permutations Replacement (RPR) method a random permutation is generated for each cache set. This permutation determines the order in which all of the ways in the set will be subject to eviction. Once all of the ways have been evicted then another random permutation is chosen and so on. This has the advantage that any repeating sequence of  $K < W$  distinct address accesses can only result in a maximum of  $K - 1$  evictions. (This worst-case can happen across the boundary of two permutations). The authors describe an efficient hardware implementation of the RPR mechanism which trades off the number of distinct permutations used against the number of bits required for implementation. The evaluation shows that the approach results in pWCET estimates at an appropriate probability of exceedance that are on average 24% better than those for conventional random replacement for the Mälardalen benchmarks studied, and 16% better on average for a railway case study.

## 6.5 Summary and Perspectives

The concept of an “*MBPTA-compliant*” platform has been pursued in many publications with the intent that any program running on the platform will be amenable to analysis using MBPTA methods. While a commendable goal, recent research indicates that there is no such panacea. Time-randomised architectures are neither necessary nor sufficient for the application of MBPTA methods based on EVT. Rather the combination of input values and hardware states, the program, and the hardware platform all influence whether MBPTA methods can be applied. Appropriate statistical tests are needed on the sample of execution time observations to determine if the method can be applied, with goodness-of-fit tests used to determine if the estimated pWCET distribution is a close match to the empirical distribution of the maxima or peaks over threshold. While time-randomised architectures may make it more likely that MBPTA methods can be applied, they cannot guarantee it. Neither do time-predictable (deterministic) architectures preclude the use of MBPTA methods [112, 16, 53, 54, 88, 87]. Time-randomised architectures may help in avoiding hazards due to pathological cases where large variations in execution time can occur very rarely (i.e. below the level at which they can reasonably be observed in testing) without being made up of a combination of smaller variations that can be observed (see the hypothetical example at the start of Section 6). However, time-randomisation does not always achieve this and can sometimes make the situation worse, as has been shown in the case of random placement.

The work on random placement policies (reviewed in Sections 6.1 and 6.2) has a number of issues. Randomising the mapping of memory blocks to cache sets across different runs of a program has been shown to lead to cache risk patterns which may not be detected during testing and analysis, but which can seriously impact execution times at much higher probabilities than are acceptable in terms of the resulting timing overruns. Despite work by Abella et al. [4], Benedicte et al. [13, 14], and Milutinovic et al. [98, 99, 97] there are as yet no viable practical solutions to this problem that can guarantee to produce trustworthy results for realistic programs. Static Software Randomisation [73, 71, 78] where a single randomly chosen placement is used in the deployed system with various different random placements used in testing and timing verification, appears to misinterpret the requirements of MBPTA methods that build upon EVT. For these methods to give sound results, it is necessary that the observations made in the analysis phase are either directly representative of those that could occur during operation, or upper bound them as noted by Cazorla et al. [27]. This is not the case when observations are taken from different binaries, which effectively constitute different systems. Such observations are not identically distributed with respect to those from the system during operation, and hence it is invalid to use the estimated pWCET distribution obtained via EVT in this case. Finally, while hardware random placement requires custom hardware [9], dynamic software random placement [72] goes counter to engineering practice, conflicting with the design principles set out in standards such

as ISO26262. Such re-arrangement of code and data objects at runtime means that deployed systems could run code and data layouts that have *never* been tested. This is unacceptable in many industries.

## 7 Case Studies, Benchmarks and Evaluation

Many papers surveyed in the previous sections include some form of evaluation. In this section, we review work that specifically focuses on performance evaluation, through the use of case studies, benchmarks, or evaluation frameworks. In addition, we review papers that provide a critique of SPTA and MBPTA methods and the open challenges that remain.

### 7.1 Critiques

In 2014, Reineke [108] made a critical technical comparison between deterministic analysis of LRU caches and probabilistic analysis of random replacement caches. He derives a number of logical conclusions, considering first random replacement and then random placement versus deterministic alternatives. Conclusion 1: LRU is preferable to random replacement in that it *always* has better guaranteed performance compared to the simple re-use distance formula for SPTA given by Davis et al. [39]. (We note that this is no longer the case with more effective SPTA techniques [7, 6]). Regarding MBPTA, Reineke [108] notes that the measurement protocol introduced by Cucu-Grosjean et al. [34] flushes the cache on program start and also prevents all input-dependent memory accesses from accessing the cache. Thus for a given path the sequence of accesses going to the cache is the same on every run, independent of the input data. Under that assumption, the behaviour of an LRU cache is fixed for a given path and so the program has only one execution time for each path. Conclusion 2 follows: MBPTA can be more efficiently employed on top of an LRU cache. With random placement, Reineke [108] shows that no independent non-zero probabilities of a cache hit can be assigned to accesses with a stack distance greater than zero. Conclusion 3 follows: random placement would require complex conditional static probabilistic analysis and is not amenable to current analysis techniques. Regarding MBPTA, Reineke [108] shows that with random placement, MBPTA cannot necessarily detect that there may be rare layouts (e.g. with a probability of  $10^{-6}$ ) that result in a large number of cache misses. Such layouts may not be observed in the runs used in the analysis phase of MBPTA. Conclusion 4 follows: random placement is not suitable for use with MBPTA. (We note that the same conclusion has also been reached by Maxim et al. [92] by considering the reproducibility property of a measurement protocol required to ensure the convergence of any set of measurements towards a representative, and thus sufficient, number of execution time observations for EVT-based estimation of the pWCET distribution).

The technical critique given by Reineke [108] was discussed by Mezzetti et al. [96] later in 2015. Regarding the use of SPTA techniques with random replacement caches (Conclusion 1), they point to the more effective SPTA methods developed by Altmeyer and Davis [7], published after Reineke's paper, which show that the guaranteed performance of random replacement caches is incomparable with that for LRU caches. With respect to SPTA and random placement (Conclusion 3), they agree that while it is true that current SPTA approaches cannot be used with random placement, future advances may be possible along this line. To the best of our knowledge, as yet no such advances have been made. Regarding the use of MBPTA with random replacement caches (Conclusion 2), the authors note that random replacement has better performance than LRU when the stack distance of accesses exceeds the number of cache ways, since in that case LRU treats all accesses as misses. With respect to MBPTA and random placement (Conclusion 4) they note the work on identifying the potential for cache risk patterns [4] as a possible means of addressing this issue.

In 2015, Stephenson et al. [123] outlined a certification argument structure aimed at providing an appropriate argument for the use of MBPTA methods [34] in an industrial context. This makes use of Goal Structuring Notation (GSN) to provide a modular structure showing the relationships between different parts of the required argument that have different concerns. The required argument is broken down into a number of elements relating to execution time measurement, soundness of the MBPTA method, soundness of the timing data extraction, and uncertainty mitigation. Further details of what is required for the “sound method” argument are then discussed. These include the need for testing to adequately sample the path or paths that represent the worst-case, and a number of factors related to the statistical methods used. Such factors include the use of appropriate parameter values in hypothesis testing, e.g. in the i.i.d. test, and testing the hypothesis that the tail of the distribution matches a Gumbel distribution. The authors point out that a lack of confidence in the values of the parameters used could lead to a lack of confidence in the overall method. Similarly, confidence is needed that the size of the sample of observations used is sufficient. They also note that *“The application of the MBPTA method requires providing evidence that the hypothesis on which it stands hold in the context of use.”* and that currently the parameters used are based on general practice for statistical approaches.

Subsequently in 2017, Gil et al. [49] discussed the open challenges in MBPTA. They identified three main areas:

1. How to ensure that a representative set of observations is obtained? This involves determining the requirements for representativity, generating appropriate test vectors that will result in representative observations, checking that coverage of the program and hardware states are sufficient for representativity, and determining how many observations are needed.
2. How to ensure a trustable application of EVT? This involves demonstrating that the methods used to obtain EVT configuration parameters are reliable, and that the application of those methods is reproducible.
3. How to interpret the results of EVT? This involves understanding the uncertainties in the overall measurement and analysis process, and determining an appropriate exceedance probability to use.

## 7.2 Case Studies and Evaluation

In 2011, Santos et al. [113] investigated the composition of statistical models of execution time for components, and how this is affected by different architectural features. They considered pairs of components  $c1$  and  $c2$ , and examined whether the simple convolution of the execution time distributions obtained for the components in isolation (which is valid assuming independence) gives a good approximation of the joint distribution obtained when  $c2$  is executed immediately following  $c1$ . They used the Kolmogorov-Smirnov goodness-of-fit test to test the hypothesis that the two distributions are the same. The evaluation used code from the MiBench suite, with the SimpleScalar tool chain used to simulate modern processors with features such as out-of-order execution. Out of 100 compositions (pairs and triples) only 3 resulted in rejection of the null hypothesis. (The null hypothesis is that the distribution formed from convolution of the distributions for single components obtained independently is the same as the distribution obtained by running the components consecutively). They also investigated which of 37 different hardware configuration parameters had the strongest influence on the difference between the distribution obtained via convolution and that directly measured. The re-order buffer had the most influence overall; however, for the cases where the null hypothesis was rejected the branch predictor had the most influence.

Time composability was also investigated by Kosmidis et al. [75] in 2013 in relation to software running on a system with fully-associative random replacement caches. They showed that the cache interference due to some other software component B running between invocations of a

particular component A, can be characterised in terms of the maximum number of cache evictions that B can cause. This is limited to at most the number of distinct addresses  $u$  that the code in B accesses. They propose using a micro-benchmark to characterise the maximum impact that any such code B with  $u$  distinct accesses can have on the pWCET distribution of A. Equation (2) in the paper aims to compute the number  $e$  of evictions required to evict *at least*  $u$  distinct entries from the cache. (We note that with random replacement no such guarantee can be made; instead, the formula approximates the number of evictions required such that the *expected* number of distinct entries evicted is  $u$ ). This formula is used to determine how many evictions a micro-benchmark should make to upper bound the impact that the interfering code in B can have on the subsequent execution time of the code in A by evicting at most  $u$  useful lines from the cache. The improvements found in the pWCET estimates at an appropriate probability of exceedance compared to flushing the cache were 5-10% for a 4Kbyte cache and 10-25% for a 32Kbyte cache for programs from the Mälardalen benchmark suite.

Later in 2013, Kosmidis et al. [77] applied the MBPTA method developed by Cucu-Grosjean et al. [34] (see Section 4.1) to systems that include *buffer resources*. They showed that resources with deterministic behaviour such as FIFO buffers do not create different probabilities of execution time outcomes for any given sequence of events, but rather propagate execution time variability or jitter. (We note that this result is somewhat unsurprising, since such buffers have deterministic behaviour any single fixed pattern of inputs yields a single fixed pattern of outputs).

In 2014, Abella et al. [2] compared deterministic and probabilistic methods of estimating the WCET / pWCET distribution using code from the Maladarlen benchmark suite [56]. This work compares classical static deterministic timing analysis using the Heptane tool for an LRU cache to SPTA and MBPTA for a random replacement cache. The comparisons investigate the sensitivity of the different approaches to cache line size and associativity. The SPTA method used is the initial approach derived by Davis et al. [39] (see Section 3.3) that uses only reuse distances, and provides a simple multi-path analysis. The MBPTA method used is the one developed by Cucu-Grosjean et al. [34] (see Section 4.1). The authors only consider single-path benchmarks. They found that in this case the results for MBPTA were less pessimistic compared to simulation than those from SPTA. We note that it would be interesting to see these comparisons repeated using the more sophisticated SPTA methods subsequently developed by Altmeyer et al. [7, 6], as well as for multi-path programs using the SPTA derived by Lesage et al. [83, 82] (see Section 3.3).

In 2013, Wartel et al. [129] applied the MBPTA method developed by Cucu-Grosjean et al. [34] (see Section 4.1) to an Integrated Modular Avionics case study. The software for the case study comprises five functions from an application that performs data concentration and maintenance of the flight control computers. This software was run on a simulator composed of a PowerPC MPC755 instruction set and pipeline emulator combined with a time accurate cache simulation. The memory hierarchy comprised separate 32KB, 8-way set-associative write-through L1 caches, and a 64KB, 8-way set-associative unified L2 copy-back cache. The caches used both random replacement and random placement. MBPTA was successfully applied, with only a few hours needed to extract the measurements and analyse the resulting observations to produce estimated pWCET distributions. The pWCET estimates at appropriate exceedance probabilities resulted in values between 0.1% and 3.8% greater than the highest observed execution time for an equivalent configuration with caches using LRU replacement and modulo placement. We note that it is not clear if these figures include the overheads of random placement, which were previously shown to be considerably higher by Kosmidis et al. [68] (see Section 6.1).

In a further avionics case study in 2015, Wartel et al. [128] applied the MBPTA method of Cucu-Grosjean et al. [34] to two different programs. The first performs data concentration and maintenance of the flight control computers, while the second computes an estimation of the centre

of gravity of the aircraft. The programs were run on a cycle accurate timing simulator which models the MPC755 architecture. The case study investigates two different hardware configurations. The first is a single core that uses software random placement of functions and stack frames with LRU caches. The second is a multi-core that uses hardware randomisation (random replacement caches, random placement, and a random permutation bus arbiter). In both cases the observations of execution times pass the i.i.d. tests enabling the MBPTA method to be applied. The results show that software randomisation increases the pWCET estimate at an appropriate exceedance probability by 12% for the first application compared to the maximum observed value for an equivalent system with modulo placement. In the multi-core case, the increases were 15% and 28% for the two applications. The authors note that the observed execution times in the multi-core case were relatively independent of the load running on the other processors, and that this was not the case with an equivalent system using a Round-Robin bus arbiter where the execution times varied by more than a factor of 3. We note that this is a consequence of the fact that Round-Robin is work-conserving whereas random permutation is effectively a variation on TDMA, which re-orders the slots allocated to each core on each cycle. A more interesting comparison would have been with traditional TDMA.

In 2015, Lesage et al. [85] introduced a framework that can be used to evaluate the precision of MBPTA. This is difficult to do with complex programs and hardware due to the difficulty in obtaining a ground truth in terms of the precise pWCET distribution. Instead of providing real measurements as input to the MBPTA method developed by Cucu-Grosjean et al. [34] (see Section 4.1), the proposed framework instead provides realistic data from synthetic tasks. Restrictions on the abstract model used enable the precise pWCET distribution to be computed and used as a reference. The technique operates by first creating an Abstract Syntax Tree (AST) representing the program. Execution Time Profiles (ETPs) are then obtained for basic blocks, via measurements of the real program. These are attached to the AST, which is used to represent a synthetic task. To evaluate the MBPTA method, the synthetic task is “executed”, i.e. a random path is chosen through the task and values from the ETPs for the basic blocks visited by that path are chosen at random. The overall synthetic execution time is then passed to MBPTA as an observation. With this model, the precise pWCET distribution can be computed from the AST via a tree-based analysis. For the simple synthetic tasks considered, the path randomisation used is sufficient to ensure path coverage. With full path coverage, the MBPTA method provides a tight and sound bound on the execution time obtained from the computed (precise) pWCET distribution at an exceedance probability of  $10^{-9}$ . Further evaluation was performed removing nodes (and hence complete paths) from the AST, these experiments show that a lack of path coverage quickly degrades the soundness of the results with optimistic execution time predictions appearing and becoming more prevalent as the number of omitted nodes is increased. This illustrates the critical importance of achieving path coverage when analysing real systems. Some aspects of the approach reported are favourable to current MBPTA techniques, for example the random choice of values from the ETPs, and the use of completely independent ETPs for basic blocks avoids path and state dependences which may be present with real programs running on a real system. The authors suggest that the framework could be adapted to model different forms of dependences in future, for example dependences between successive runs, and dependences between the ETPs for successive blocks.

In 2017, Mezzetti et al. [94] applied the EPC method [136] (see Section 5.1) to a railway application; a simplified European Train Control System. They describe how EPC was integrated into an industrial tool chain (Rapita RVS<sup>16</sup>). Recall, that EPC collects observations at the basic

<sup>16</sup>See <https://www.rapitasystems.com/>

block level. It then uses probabilistic padding computed using SPTA techniques derived by Altmeyer and Davis [7] to increase the execution times observed to account for any advantage that may have accrued due to the path taken to that block. Observations can then be synthesised for all paths. The authors note that although EPC does not require that all paths are exercised when obtaining the execution times for basic blocks, as shown by Lesage et al. [85], all paths *do* need to be covered by the synthesised observations. In practice the number of paths may be very large, the authors therefore make use of semantic information in the form of flow facts to reduce the number of paths considered as feasible, accounting for maximum loop iterations and correlations between conditionals. This is highly effective in the case of the railway application studied, reducing the number of paths considered from 12996 to just 26. The hardware platform used is an FPGA prototype implementing a modified LEON3/4 architecture, with random modulo placement [61] and random replacement caches. We note that there is no mention of whether the issue of cache risk patterns previously identified with random placement was addressed (see Section 6.3). The results of a simple control experiment indicate that the EPC method leads to approx. 10-20% over-approximation of the pWCET at an appropriate exceedance probability, compared to exhaustively exercising all paths and using MBPTA. With the railway case study, unfortunately the worst-case path could not be exercised during testing; however, from analysis of a similar path that was executed, the authors estimate the over-approximation at around 30%. They note that only 23% of basic blocks needed padding, and that padding increases the original observations by on average 10%, and up to 400% in the worst-case.

In 2017, Fernandez et al. [48] reported on a case study using software random placement [72] (see Section 6.2) in support of MBPTA. These techniques were applied to an application that controls the active optics of a space telescope. This application runs on a LEON3 processor with separate L1 data and instruction caches and a unified write-back L2 cache. The evaluation shows that the maximum observed execution time is changed little by the overheads of dynamic software randomisation. Further, the pWCET estimate at a probability of exceedance of  $10^{-15}$  is only approx. 2% larger than the maximum observed execution time for the original version, and hence substantially less than the 20% engineering margin that the authors claim is typically applied in this system. We note that in this work there is no mention of any mitigation of the issues of cache risk patterns (see Section 6.3) that can occur with software random placement.

In a 2-page paper in 2017, Cros et al. [33] reported on a case study applying MBPTA to a space application (Thrust Vector Control). They used a LEON3 processor with random modulo placement [61] and an FPU modified to have fixed latency operations in analysis mode. The results show that the pWCET at a probability of exceedance of  $10^{-6}$  is 1.5 times larger than the maximum observed execution time for the equivalent deterministic system. This equates to the 50% engineering margin that the authors claim is usually applied in this system. The values for an exceedance probability of  $10^{-9}$ ,  $10^{-12}$ , and  $10^{-15}$  were approx. 1.75, 2.0, and 2.2 times larger respectively.

The use of the MBPTA method developed by Cucu-Grosjean et al. [34] (see Section 4.1) was extended to multi-core hardware (a 4-core LEON 3 platform implemented on an FPGA) by Diaz et al. [43] in 2017. The key challenge here is to address the additional contention delay on memory requests due to tasks running on the other cores. With the proposed method, the task under analysis is run on a single core in isolation, i.e. with no contenders, and execution time observations recorded. To account for possible contention, Performance Monitoring Counters (PMCs) are used to record the number of requests of different types. Two approaches are then used to add on appropriate upper bounds on the additional delays that could be caused by contention. A fully time composable (fTC) approach includes the maximum possible delay that any contending tasks on other cores could cause. A partially time composable (pTC) approach matches up

each memory request of the task under analysis with the worst-case additional delays that could be caused by specific contending tasks, taking into account the maximum number of delays of different types that they may cause. pTC provides a tighter bound than fTC, but requires specific information about the contenders. The experimental evaluation shows that the results from MBPTA for tasks running in isolation are a few percent above the maximum observed value, thus the remaining evaluation which factors in contention reveals mainly the effectiveness of the fTC and pTC approaches. The pTC approach provides good results with a bound of less than 1.5 times the observed multi-core value for the benchmarks considered. The fTC approach is much more pessimistic with bounds 12 times larger.

Also in 2017, Silva et al. [116] evaluated the reliability and tightness of the estimated pWCET distributions derived by MBPTA via fitting (i) GEV and (ii) Gumbel distributions using the Block Maxima approach. They used the L-moments method to fit a GEV distribution, and the Maximum Likelihood Estimator to fit a Gumbel distribution. The evaluation assesses *reliability* in terms of whether the pWCET estimate at a probability of exceedance of  $10^{-15}$  and its confidence intervals are above the High Water Mark (HWM), i.e. the maximum, obtained from  $10^8$  observations used for validation. The *tightness* is assessed by comparing the pWCET estimate at a probability of exceedance of  $10^{-7}$  with the maximum values observed from  $10^6$  and  $10^8$  samples. Experiments were performed on the *bssort*, *insertsort*, and *bs* sorting algorithms from the Mälardalen benchmark suite using input data that selected the worst-case path (effectively single path examples). Variability in run times was therefore only due to hardware randomisation in the execution platform. This was an FPGA implementation of a dual-core processor with a randomised bus and a random replacement cache. The experiments show that using a GEV distribution, there is significant variability in the pWCET estimates at a probability of exceedance of  $10^{-15}$  for different analysis sample sizes (from 3 to 100 blocks of size 50). Further, since the HWM was often within the confidence interval, and so could be above the pWCET estimate, the results were not reliable. Fitting to a Gumbel distribution, however, provided reliable results which were also tight, i.e. only a few percent higher than the HWM. Further experiments were also performed using synthetic input data from GEV distributions with specific shape parameters in the range  $-1/2$  to  $+1/2$ . In all cases, fitting to a GEV proved unreliable. For shape parameters that were negative or zero, using a Gumbel distribution was reliable and provided reasonably tight results. For positive shape parameters, fitting to a Gumbel distribution is not appropriate and indeed it produced results that were optimistic. The evaluation also showed that the Continuous Ranked Probability Score (CRPS) metric used in the method derived by Cucu-Grosjean et al. [34] to determine when sufficient samples have been obtained for analysis resulted in reliable results, but could be improved upon. Improvements could be achieved either by using a smaller threshold, or via assessing convergence using plots of the pWCET estimate and confidence intervals versus sample size, as shown in the paper. We note that fitting a distribution minimises the absolute error rather than attempting to minimise the over-approximation while avoiding any under-approximation. Fitting a Gumbel distribution to data from a distribution with a finite maximum (which would be better represented by a reversed Weibull distribution) typically results in an over-approximation. The authors consider this over-approximation to be indicative of the reliability of the method when using a Gumbel distribution; however, we would caution against drawing such a conclusion. In these cases, the distribution is an over-approximation because there is a mismatch with the shape parameter. This results in a fitted distribution which is pessimistic at small probabilities of exceedance. This pessimism is not necessarily an indicator that the method is reliable per se (i.e. always produces sound results). Indeed, for cases where the shape parameter is positive, using a Gumbell distribution produces results which are optimistic.

In 2018, Reghenzani et al. [107] described *chronovise*, an open source software framework for MBPTA. *chronovise* is a static C++ library that supports the Block Maxima, Peak-over-Threshold and MBPTA-CV [3] approaches.

### 7.3 Summary and Perspectives

Evaluation of the MBPTA method introduced by Cucu-Grosjean et al. [34] on avionics case studies at Airbus [129, 128] has shown both the applicability of the method to real systems, and also that the use of random replacement caches comes at a relatively small performance penalty compared to using deterministic replacement policies such as LRU. Recent work (discussed in section 4.2) shows that MBPTA methods can also be successfully applied to systems running on time-predictable architectures. For example Berezovskyi et al. [16, 15] successfully apply MBPTA methods to programs running on an NVIDIA Kepler GK104 GPU, and Guet et al. [53, 54] to benchmarks running on an Intel Xeon with 4 cores and 3 levels of cache. Initial work by Diaz et al. [43] in 2017, shows how the MBPTA approach can be extended to a multi-core system.

The important issue of proving the validity and correctness of the results from MBPTA has been investigated by Lesage et al. [85]. They showed that a lack of path coverage quickly degrades the soundness of the results, with optimistic execution time predictions appearing and becoming more prevalent as the number of omitted paths increases. Mezzetti et al. [94] addressed the path coverage problem using the EPC method [136] and demonstrated its effectiveness on a railway application.

Silva et al. [116] evaluated the reliability and tightness of the pWCET estimates derived using MBPTA by fitting GEV and Gumbel distributions, concluding that using GEV is unreliable (with large confidence intervals), while Gumbel can provide results that are reliable and tight in those cases where it is applicable. To show that it is applicable; however, requires checking against the maxima of a very large number of observations obtained in a validation phase. It is important to note here that fitting a distribution minimises the absolute error rather than attempting to minimise the over-approximation while avoiding any under-approximation. Fitting a Gumbel distribution to data from a distribution with a finite maximum, which would be better represented by a reversed Weibull distribution, typically results in a fitted distribution which is pessimistic at small probabilities of exceedance. This pessimism is not necessarily an indicator that the method is reliable per se (i.e. always produces sound results). Indeed, for cases where the shape parameter is positive, using a Gumbell distribution is likely to produce results which are optimistic.

## 8 Conclusions

In this survey, we reviewed research into probabilistic timing analysis techniques for hard real-time systems. We covered the main subject areas: static probabilistic timing analysis (SPTA), measurement-based probabilistic timing analysis (MBPTA), and hybrid methods (HyPTA), as well as reviewing supporting mechanisms and techniques, case studies, and evaluations.

### 8.1 Open Issues

We conclude by identifying open issues, key challenges and possible directions for future research. We present these open issues and challenges as a series of questions. While there has been progress in some of these areas, as highlighted in this survey, comprehensive solutions are currently tantalizingly out-of-reach. Further research is needed to secure a sound and comprehensive basis for the potential subsequent development and deployment of industry strength tools.

1. How much hardware time-randomisation is needed to ease the use of MBPTA methods in practice? Is custom time-randomised hardware with random replacement caches, random permutation buses etc. really necessary?
2. What are the hazards involved in applying MBPTA methods to systems comprising time-predictable COTS hardware that use entirely deterministic policies, and how can these hazards be overcome?
3. How can we solve the *path coverage problem* such that the user of MBPTA methods is not required to provide a measurement protocol that exercises all possible paths through the code? What level of coverage is needed for the MBPTA methods to provide sound results?
4. How can we apply MBPTA to multi-path programs? Do we have to apply MBPTA to each path individually and then combine the results to obtain a valid upper bound? Or is it possible to use measurements from different paths as input into EVT and still obtain sound results? (There is an argument that doing so makes it much harder to ensure representativity).
5. How can we solve the *representativity problem* such that the sample of observations used as input to MBPTA result in a pWCET distribution that correctly characterises the behaviour of the system during any future scenario of operation?
6. Given that testing can continue to produce execution time observations almost indefinitely, how do we know when sufficient observations have been obtained for an accurate estimate of the pWCET distribution to be derived?
7. How can we validate that a MBPTA implementation actually produces correct results?
8. How can we provide MBPTA for systems where execution time observations exhibit dependencies?
9. How to apply MBPTA methods to systems using multi-core processors where there is substantial contention for shared hardware resources (e.g. interconnect, memory hierarchy, caches, DRAM etc.) between programs running on different cores?
10. How to apply MBPTA methods to systems that make use of multi-threading on a single core, and thus interleave the execution of a number of programs resulting in interference on shared hardware resources?
11. How can we mechanise MBPTA so that, for example block sizes and thresholds can be selected automatically?
12. How can we convince certification authorities that estimated pWCET distributions derived via MBPTA methods are safe?

## 8.2 Directions for Future Research

We end this survey with a discussion of an important direction for future real-time systems research which probabilistic analysis techniques may be able contribute to.

There is a continuing trend in industry sectors including avionics, automotive electronics, consumer electronics, and robotics away from development and deployment on single-core processors towards using significantly more powerful and complex Common-Off-The-Shelf (COTS) multi-core and many-core hardware platforms. This trend is driven by requirements on size, weight and power consumption, increasing cost pressures and the demand for more complex and capable functionality delivered through software. The use of COTS multi-core hardware poses significant challenges in terms of verifying timing behaviour and ensuring that real-time constraints are met. These challenges stem from the complexity of the architecture and the way in which hardware resources such as the interconnect and the memory hierarchy are shared between different processing cores. Some researchers are seeking to address these problems through approaches based on partitioning and separation (e.g. single-core equivalence [91]), while others aim for solutions based on considering the explicit interference on each hardware resource from co-running programs and

how this demand can be served by the available resource supply [8, 37]. There is the potential for probabilistic timing analysis and probabilistic schedulability analysis techniques (reviewed in a companion survey [38]) to play a role in the timing verification of such complex real-time systems.

There are a number of ways in which the interference effects of cross-core contention can be considered within a framework of probabilistic timing verification:

- By running synthetic “worst-case” contenders on other cores during the analysis phase. The idea being to account for the worst-case cross-core interference that could possibly occur irrespective of the actual co-runners, within the estimated pWCET distributions. The difficulty with this context-independent approach lies in determining which contenders, or combination of contenders, actually produce the worst-case interference for a given program. Further, the cross-core interference assumed may be very pessimistic, i.e. much larger than can actually occur in the operational system with the real co-runners.
- By running the real co-runners during the analysis phase. This approach has the advantage that it can potentially avoid some of the pessimism in the above context-independent approach. The estimated pWCET distributions obtained would only apply for the specific set of co-runners. Thus they would be context-dependent. This approach has the disadvantage that it makes the problem of representativity even more acute, since different combinations and timings of co-runners need to be considered.
- By using isolation or partial isolation techniques (implemented in either software or hardware) to limit or bound the interference that can occur due to contention for shared resources in a way that is independent of the actual behaviour of the co-runners. Synthetic contenders can then be used to generate this maximum interference at analysis time enabling sound upper bounding of the pWCET distributions that can occur during normal operation of the system.
- By running the program under analysis in isolation with all the other cores idle. In this case, the estimated pWCET distributions obtained would not include the impact of any co-runners. Instead, these effects would need to be integrated at a later stage, for example within some form of probabilistic schedulability analysis. The advantage of this approach is that it reduces the issues of representativity to the single core case; however, difficulties remain in soundly including the effects of cross-core contention without the results becoming either unsound or highly pessimistic.
- By running the complete system and applying statistical analysis (e.g. based on EVT) to response times, rather than execution times (see Section 7 of the companion survey [38] for a review of the initial work in this area). This approach again raises difficulties with representativity; however, it has the advantage that it treats the entire system as a grey box requiring less detailed information about the hardware and software behaviour.

Since the initial work of Burns and Edgar [22] in 2000 on probabilistic timing analysis, significant progress has been made in the development of both measurement-based and static probabilistic timing analysis techniques. However, there are still many important unanswered questions and open issues that need to be addressed. In particular, work on probabilistic timing analysis and probabilistic schedulability analysis for multi-core and many-core systems is in its infancy with opportunities for significant advances addressing an important research challenge.

**Acknowledgements.** The research that went into writing this survey was funded, in part, by the Inria International Chair program and the EPSRC grant MCCps (EP/P003664/1). EPSRC Research Data Management: No new primary data was created during this study.

The authors would like to thank David Griffin and Alan Burns for their comments on an earlier draft of this survey.

## References

- 1 J. Abella, F. J. Cazorla, E. Quinones, and T. Vardanega. Measurement-based probabilistic timing analysis and i.i.d property. White Paper Version 2. Technical report <http://www.proartis-project.eu/publications/MBPTA-white-paper>, BSC, July 2014.
- 2 J. Abella, D. Hardy, I. Puaut, E. Quiñones, and F. J. Cazorla. On the Comparison of Deterministic and Probabilistic WCET Estimation Techniques. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pages 266–275, July 2014. doi:10.1109/ECRTS.2014.16.
- 3 J. Abella, M. Padilla, J. Del Castillo, and F. J. Cazorla. Measurement-Based Worst-Case Execution Time Estimation Using the Coefficient of Variation. *ACM Trans. Des. Autom. Electron. Syst.*, 22(4):72:1–72:29, June 2017. doi:10.1145/3065924.
- 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 *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pages 255–265, 2014. doi:10.1109/ECRTS.2014.33.
- 5 I. Agirre, M. Azkarate-askasua, C. Hernandez, J. Abella, J. Perez, T. Vardanega, and F. J. Cazorla. IEC-61508 SIL 3 Compliant Pseudo-Random Number Generators for Probabilistic Timing Analysis. In *Proceedings of the Euromicro Conference on Digital System Design (DSD)*, pages 677–684, August 2015. doi:10.1109/DSD.2015.26.
- 6 S. Altmeyer, L. Cucu-Grosjean, and R. I. Davis. Static probabilistic timing analysis for real-time systems using random replacement caches. *Springer Real-Time Systems*, 51(1):77–123, 2015. doi:10.1007/s11241-014-9218-4.
- 7 S. Altmeyer and R. I. Davis. On the Correctness, Optimality and Precision of Static Probabilistic Timing Analysis. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 26:1–26:6, 2014. URL: <http://dl.acm.org/citation.cfm?id=2616606.2616638>.
- 8 S. Altmeyer, R. I. Davis, L. Indrusiak, C. Maiza, V. Nelis, and J. Reineke. A Generic and Compositional Framework for Multicore Response Time Analysis. In *Proceedings of the International Conference on Real-Time Networks and Systems (RTNS)*, pages 129–138, 2015. doi:10.1145/2834848.2834862.
- 9 H. Anwar, C. Chen, and G. Beltrame. A probabilistically analysable cache implementation on FPGA. In *IEEE International New Circuits and Systems Conference (NEWCAS)*, pages 1–4, June 2015. doi:10.1109/NEWCAS.2015.7181984.
- 10 I. Bate and U. Khan. WCET Analysis of Modern Processors Using Multi-criteria Optimisation. *Empirical Softw. Engg.*, 16(1):5–28, February 2011.
- 11 P. Benedicte, C. Hernandez, J. Abella, and F. J. Cazorla. Design and integration of hierarchical placement multi-level caches for real-time systems. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 455–460, March 2018. doi:10.23919/DATE.2018.8342052.
- 12 P. Benedicte, C. Hernandez, J. Abella, and F. J. Cazorla. RPR: A Random Replacement Policy with Limited Pathological Replacements. In *Proceedings of ACM Symposium on Applied Computing (SAC)*, pages 593–600, 2018. doi:10.1145/3167132.3167197.
- 13 P. Benedicte, L. Kosmidis, E. Quinones, J. Abella, and F. J. Cazorla. Modelling the confidence of timing analysis for time randomised caches. In *Proceedings of the IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 1–8, May 2016. doi:10.1109/SIES.2016.7509421.
- 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 *Proceedings of the IEEE International Conference on Industrial Informatics (INDIN)*, pages 90–97, July 2016. doi:10.1109/INDIN.2016.7819140.
- 15 K. Berezovskyi, F. Guet, L. Santinelli, K. Bletsas, and E. Tovar. Measurement-Based Probabilistic Timing Analysis for Graphics Processor Units. In *Proceedings of the International Conference on the Architecture of Computing Systems (ARCS)*, pages 223–236, April 2016. doi:10.1007/978-3-319-30695-7\_17.
- 16 K. Berezovskyi, L. Santinelli, K. Bletsas, and E. Tovar. WCET Measurement-based and Extreme Value Theory Characterisation of CUDA Kernels. In *Proceedings of the International Conference on Real-Time Networks and Systems (RTNS)*, pages 279–288, 2014. doi:10.1145/2659787.2659827.
- 17 G. Bernat, A. Burns, and M. Newby. Probabilistic Timing Analysis: An Approach Using Copulas. *J. Embedded Comput.*, 1(2):179–194, April 2005. URL: <http://dl.acm.org/citation.cfm?id=1233760.1233763>.
- 18 G. Bernat, A. Colin, and S. Petters. pwcet: A tool for probabilistic worst-case execution time analysis of real-time systems. Technical report, Department of Computer Science, University of York, 2003.
- 19 G. Bernat, A. Colin, and S. M. Petters. WCET analysis of probabilistic hard real-time systems. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, pages 279–288, 2002. doi:10.1109/REAL.2002.1181582.
- 20 B. Braams, S. Altmeyer, and A. D. Pimentel. EDiFy: An execution time distribution finder. In *Proceedings of the Design Automation Conference (DAC)*, pages 1–6, June 2017. doi:10.1145/3061639.3062233.
- 21 S. Bunte, M. Zolda, M. Tautschnig, and R. Kirner. Improving the Confidence in Measurement-Based Timing Analysis. In *Proceedings of the IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing (ISORC)*, pages 144–151, March 2011. doi:10.1109/ISORC.2011.27.

- 22 A. Burns and S. Edgar. Predicting computation time for advanced processor architectures. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pages 89–96, 2000. doi:10.1109/EMRTS.2000.853996.
- 23 A. Burns and D. Griffin. Predictability as an emergent behaviour. In *Proceedings of the Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS)*, pages 27–29, 2011.
- 24 S. Bunte, M. Zolda, and R. Kirner. Let’s get less optimistic in measurement-based timing analysis. In *Proceedings of the IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 204–212, June 2011. doi:10.1109/SIES.2011.5953663.
- 25 F. J. Cazorla, J. Abella, J. Andersson, T. Vardanega, F. Vatrinet, I. Bate, I. Broster, M. Azkarate-Askasua, F. Wartel, L. Cucu, F. Cros, G. Farrall, A. Gogonel, A. Gianarro, B. Triquet, C. Hernandez, C. Lo, C. Maxim, D. Morales, E. Quinones, E. Mezzetti, L. Kosmidis, I. Aguirre, M. Fernandez, M. Slijepcevic, P. Conmy, and W. Talaboulma. PROXIMA: Improving Measurement-Based Timing Analysis through Randomisation and Probabilistic Analysis. In *Proceedings of the Euromicro Conference on Digital System Design (DSD)*, pages 276–285, August 2016. doi:10.1109/DSD.2016.22.
- 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*, 12(2s):94:1–94:26, May 2013. doi:10.1145/2465787.2465796.
- 27 F. J. Cazorla, T. Vardanega, E. Quiñones, and J. Abella. Upper-bounding Program Execution Time with Extreme Value Theory. In *Proceedings of the Workshop on Worst-Case Execution Time Analysis (WCET)*, pages 64–76, 2013. doi:10.4230/OASICS.WCET.2013.64.
- 28 C. Chen and G. Beltrame. An Adaptive Markov Model for the Timing Analysis of Probabilistic Caches. *ACM Trans. Des. Autom. Electron. Syst.*, 23(1):12:1–12:24, August 2017. doi:10.1145/3123877.
- 29 C. Chen, J. Panerati, and G. Beltrame. Effects of online fault detection mechanisms on Probabilistic Timing Analysis. In *Proceedings of IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 41–46, September 2016. doi:10.1109/DFT.2016.7684067.
- 30 C. Chen, J. Panerati, I. Hafnaoui, and G. Beltrame. Static probabilistic timing analysis with a permanent fault detection mechanism. In *Proceedings of the IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 1–10, June 2017. doi:10.1109/SIES.2017.7993373.
- 31 C. Chen, L. Santinelli, J. Hugues, and G. Beltrame. Static probabilistic timing analysis in presence of faults. In *Proceedings of the IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 1–10, May 2016. doi:10.1109/SIES.2016.7509422.
- 32 S. Coles. *An Introduction to Statistical Modeling of Extreme Values*. Springer, 2001. doi:10.1007/978-1-4471-3675-0.
- 33 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 *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 103–108, March 2017. doi:10.23919/DATE.2017.7926966.
- 34 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 *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pages 91–101, July 2012. doi:10.1109/ECRTS.2012.31.
- 35 L. David and I. Puaud. Static determination of probabilistic execution times. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pages 223–230, June 2004. doi:10.1109/EMRTS.2004.1311024.
- 36 R. A. Davis and T. Mikosch. The extremogram: A correlogram for extreme events. *Bernoulli*, 15(4):977–1009, November 2009. doi:10.3150/09-BEJ213.
- 37 R. I. Davis, S. Altmeyer, L. S. Indrusiak, C. Maiza, V. Nelis, and J. Reineke. An extensible framework for multicore response time analysis. *Springer Real-Time Systems*, 54(3):607–661, July 2018. doi:10.1007/s11241-017-9285-4.
- 38 R. I. Davis and L. Cucu-Grosjean. A Survey of Probabilistic Schedulability Analysis Techniques for Hard Real-Time Systems. *Leibniz Transactions on Embedded Systems (LITES)*, 6(1):04:1–04:53, May 2019. doi:10.4230/LITES-v006-i001-a004.
- 39 R. I. Davis, L. Santinelli, S. Altmeyer, C. Maiza, and L. Cucu-Grosjean. Analysis of Probabilistic Cache Related Pre-emption Delays. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pages 168–179, July 2013. doi:10.1109/ECRTS.2013.27.
- 40 R. I. Davis, J. Whitham, and D. Maxim. Static Probabilistic Timing Analysis for Multicore Processors with Shared Cache. In *Proceedings of the Real-Time Scheduling Open Problems Seminar (RTSOPS)*, pages 3–5, 2013.
- 41 R. I. Davis. Improvements to Static Probabilistic Timing Analysis for Systems with Random Cache Replacement Policies. In *Proceedings of the Real-Time Scheduling Open Problems Seminar (RTSOPS)*, pages 22–24, July 2013.
- 42 J-F. Deverge and I. Puaud. Safe measurement-based WCET estimation. In *Proceedings of the Workshop on Worst-Case Execution Time Analysis (WCET)*, 2005.
- 43 E. Díaz, M. Fernández, L. Kosmidis, E. Mezzetti, C. Hernandez, J. Abella, and F. J. Cazorla. MC2: Multicore and Cache Analysis via Deterministic and Probabilistic Jitter Bounding, pages 102–118.

- Springer International Publishing, Cham, 2017. doi:10.1007/978-3-319-60588-3\_7.
- 44 J. L. Diaz, J. M. Lopez, M. Garcia, A. M. Campos, Kanghee Kim, and L. L. Bello. Pessimism in the stochastic analysis of real-time systems: concept and applications. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, pages 197–207, December 2004. doi:10.1109/REAL.2004.41.
  - 45 S. Edgar and A. Burns. Statistical analysis of WCET for scheduling. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, pages 215–224, December 2001. doi:10.1109/REAL.2001.990614.
  - 46 P. Embrechts, C. Kluppelberg, and T. Mikosch. *Modelling extremal events for insurance and Finance*. Springer, 1997. doi:10.1007/978-3-642-33483-2.
  - 47 I. Fedotova, B. Krause, and E. Siemens. Applicability of Extreme Value Theory to the Execution Time Prediction of Programs on SoCs. In *Proceedings of the International Conference on Applied Innovations in IT (ICAIIT)*, March 2017.
  - 48 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 *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 738–739, March 2017. doi:10.23919/DATE.2017.7927087.
  - 49 S. Jimenez 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*, PP(99):1–1, 2017. doi:10.1109/LES.2017.2712858.
  - 50 D. Griffin, I. Bate, B. Lesage, and F. Soboczenski. Evaluating Mixed Criticality Scheduling Algorithms with Realistic Workloads. In *Proceedings of Workshop on Mixed Criticality (WMC)*, 2015.
  - 51 D. Griffin and A. Burns. Realism in Statistical Analysis of Worst Case Execution Times. In *Proceedings of the Workshop on Worst-Case Execution Time Analysis (WCET)*, pages 44–53, 2010. doi:10.4230/OASICS.WCET.2010.44.
  - 52 D. Griffin, B. Lesage, A. Burns, and R. I. Davis. Static Probabilistic Timing Analysis of Random Replacement Caches Using Lossy Compression. In *Proceedings of the International Conference on Real-Time Networks and Systems (RTNS)*, pages 289–298, 2014. doi:10.1145/2659787.2659809.
  - 53 F. Guet, L. Santinelli, and J. Morio. On the Reliability of the Probabilistic Worst-Case Execution Time Estimates. In *Proceedings of the European Congress on Embedded Real Time Software and Systems (ERTS)*, January 2016. URL: <https://hal.archives-ouvertes.fr/hal-01289477>.
  - 54 F. Guet, L. Santinelli, and J. Morio. Probabilistic analysis of cache memories and cache memories impacts on multi-core embedded systems. In *Proceedings of the IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 1–10, May 2016. doi:10.1109/SIES.2016.7509420.
  - 55 F. Guet, L. Santinelli, and J. Morio. On the Representativity of Execution Time Measurements: Studying Dependence and Multi-Mode Tasks. In Jan Reineke, editor, *Proceedings of the Workshop on Worst-Case Execution Time Analysis (WCET)*, volume 57 of *OASICS*, pages 3:1–3:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/OASICS.WCET.2017.3.
  - 56 J. Gustafsson, A. Betts, A. Ermedahl, and B. Lisper. The Mälardalen WCET Benchmarks – Past, Present and Future. In *Proceedings of the Workshop on Worst-Case Execution Time Analysis (WCET)*, pages 137–147, July 2010.
  - 57 J. Hansen, S. A. Hissam, and G. A. Moreno. Statistical-based WCET estimation and validation. In *Proceedings of the Workshop on Worst-Case Execution Time Analysis (WCET)*, volume 252, 2009.
  - 58 D. Hardy and I. Puaut. Static Probabilistic Worst Case Execution Time Estimation for Architectures with Faulty Instruction Caches. In *Proceedings of the International Conference on Real-Time Networks and Systems (RTNS)*, pages 35–44, 2013. doi:10.1145/2516821.2516842.
  - 59 D. Hardy and I. Puaut. Static Probabilistic Worst Case Execution Time Estimation for Architectures with Faulty Instruction Caches. *Springer Real-Time Systems*, 51(2):128–152, March 2015. doi:10.1007/s11241-014-9212-x.
  - 60 C. Hernandez, J. Abella, F. J. Cazorla, J. Andersson, and A. Gianarro. Towards making a LEON3 multicore compatible with probabilistic timing analysis. In *Proceedings of the Data Systems In Aerospace Conference (DASIA)*, May 2015.
  - 61 C. Hernandez, J. Abella, A. Gianarro, J. Andersson, and F. J. Cazorla. Random Modulo: A New Processor Cache Design for Real-time Critical Systems. In *Proceedings of the Design Automation Conference (DAC)*, pages 29:1–29:6, 2016. doi:10.1145/2897937.2898076.
  - 62 K. Höfig. *Failure-Dependent Timing Analysis - A New Methodology for Probabilistic Worst-Case Execution Time Analysis*, pages 61–75. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. doi:10.1007/978-3-642-28540-0\_5.
  - 63 T. Hsing. On Tail Index Estimation Using Dependent Data. *The Annals of Statistics*, 19(3):1547–1569, 1991. URL: <http://www.jstor.org/stable/2241962>.
  - 64 M. Ivers and R. Ernst. *Probabilistic Network Loads with Dependencies and the Effect on Queue Sojourn Times*, pages 280–296. Springer Berlin Heidelberg, 2009. doi:10.1007/978-3-642-10625-5\_18.
  - 65 J. Jalle, L. Kosmidis, J. Abella, E. Quinones, and F. J. Cazorla. Bus Designs for Time-probabilistic Multicore Processors. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 50:1–50:6, 2014. URL: <http://dl.acm.org/citation.cfm?id=2616606.2616668>.
  - 66 T. Kelter, H. Falk, P. Marwedel, S. Chattopadhyay, and A. Roychoudhury. Static Analysis of Multi-core TDMA Resource Arbitration

- Delays. *Springer Real-Time Systems*, 50(2):185–229, March 2014. doi:10.1007/s11241-013-9189-x.
- 67 L. Kosmidis, J. Abella, E. Quinones, and F. J. Cazorla. Efficient Cache Designs for Probabilistically Analysable Real-Time Systems. *IEEE Transactions on Computers*, 63(12):2998–3011, December 2014. doi:10.1109/TC.2013.182.
  - 68 L. Kosmidis, J. Abella, E. Quiñones, and F. J. Cazorla. A cache design for probabilistically analysable real-time systems. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 513–518, March 2013. doi:10.7873/DATE.2013.116.
  - 69 L. Kosmidis, J. Abella, E. Quiñones, and F. J. Cazorla. Multi-level Unified Caches for Probabilistically Time Analysable Real-Time Systems. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, pages 360–371, December 2013. doi:10.1109/RTSS.2013.43.
  - 70 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 *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pages 276–287, July 2014. doi:10.1109/ECRTS.2014.34.
  - 71 L. Kosmidis, D. Compagnin, D. Morales, E. Mezzetti, E. Quiñones, J. Abella, T. Vardanega, and F. J. Cazorla. Measurement-Based Timing Analysis of the AURIX Caches. In *Proceedings of the Workshop on Worst-Case Execution Time Analysis (WCET)*, 2016.
  - 72 L. Kosmidis, C. Curtsinger, E. Quiñones, J. Abella, E. Berger, and F. J. Cazorla. Probabilistic timing analysis on conventional cache designs. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 603–606, March 2013. doi:10.7873/DATE.2013.132.
  - 73 L. Kosmidis, E. Quiñones, J. Abella, G. Farrall, F. Wartel, and F. J. Cazorla. Containing Timing-Related Certification Cost in Automotive Systems Deploying Complex Hardware. In *Proceedings of the Design Automation Conference (DAC)*, pages 22:1–22:6, 2014. doi:10.1145/2593069.2593112.
  - 74 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 *Proceedings of the Euromicro Conference on Digital System Design (DSD)*, pages 401–410, August 2014. doi:10.1109/DSD.2014.50.
  - 75 L. Kosmidis, E. Quiñones, J. Abella, T. Vardanega, and F. J. Cazorla. Achieving timing composability with measurement-based probabilistic timing analysis. In *Proceedings of the IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing (ISORC)*, pages 1–8, June 2013. doi:10.1109/ISORC.2013.6913193.
  - 76 L. Kosmidis, E. Quiñones, J. Abella, T. Vardanega, C. Hernandez, A. Gianarro, I. Broster, and F. J. Cazorla. Fitting processor architectures for measurement-based probabilistic timing analysis. *Microprocessors and Microsystems*, 2016. doi:10.1016/j.micpro.2016.07.014.
  - 77 L. Kosmidis, T. Vardanega, J. Abella, E. Quiñones, and F. J. Cazorla. Applying Measurement-Based Probabilistic Timing Analysis to Buffer Resources. In *Proceedings of the Workshop on Worst-Case Execution Time Analysis (WCET)*, pages 97–108, 2013. doi:10.4230/OASICS.WCET.2013.97.
  - 78 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)*, pages 1–8, November 2016. doi:10.1145/2966986.2967078.
  - 79 K. Lahiri, A. Raghunathan, and G. Lakshminarayana. LOTTERYBUS: a new high-performance communication architecture for system-on-chip designs. In *DAC*, pages 15–20, 2001. doi:10.1109/DAC.2001.156100.
  - 80 S. Law and I. Bate. Achieving Appropriate Test Coverage for Reliable Measurement-Based Timing Analysis. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pages 189–199, July 2016. doi:10.1109/ECRTS.2016.21.
  - 81 M. R. Leadbetter, G. Lindgren, and H. Rootzen. Conditions for the convergence in distribution of maxima of stationary normal processes. *Stochastic Processes and their Applications*, 8(2), 1978.
  - 82 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*, December 2017. doi:10.1007/s11241-017-9295-2.
  - 83 B. Lesage, D. Griffin, S. Altmeyer, and R. I. Davis. Static Probabilistic Timing Analysis for Multi-path Programs. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, pages 361–372, December 2015. doi:10.1109/RTSS.2015.41.
  - 84 B. Lesage, D. Griffin, R. I. Davis, and S. Altmeyer. On the application of Static Probabilistic Timing Analysis to Memory Hierarchies. In *Proceedings of the Real-Time Scheduling Open Problems Seminar (RTSOPS)*, 2014.
  - 85 B. Lesage, D. Griffin, F. Soboczenski, I. Bate, and R. I. Davis. A Framework for the Evaluation of Measurement-based Timing Analyses. In *Proceedings of the International Conference on Real-Time Networks and Systems (RTNS)*, pages 35–44, 2015. doi:10.1145/2834848.2834858.
  - 86 Y. Liang and T. Mitra. Cache modeling in probabilistic execution time analysis. In *Proceedings of the Design Automation Conference (DAC)*, pages 319–324, June 2008.
  - 87 G. Lima and I. Bate. Valid Application of EVT in Timing Analysis by Randomising Execution Time Measurements. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2017.

- 88 G. Lima, D. Dias, and E. Barros. Extreme Value Theory for Estimating Task Execution Time Bounds: A Careful Look. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, July 2016.
- 89 Y. Lu, T. Nolte, I. Bate, and L. Cucu-Grosjean. A New Way About Using Statistical Analysis of Worst-case Execution Times. *SIGBED Rev.*, 8(3):11–14, September 2011. doi:10.1145/2038617.2038619.
- 90 Y. Lu, T. Nolte, I. Bate, and L. Cucu-Grosjean. A trace-based statistical worst-case execution time analysis of component-based real-time embedded systems. In *Proceedings of the IEEE Conference on Emerging Technologies Factory Automation (ETFA)*, pages 1–4, September 2011. doi:10.1109/ETFA.2011.6059190.
- 91 R. Mancuso, R. Pellizzoni, M. Caccamo, L. Sha, and H. Yun. WCET(m) Estimation in Multi-core Systems Using Single Core Equivalence. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pages 174–183, July 2015. doi:10.1109/ECRTS.2015.23.
- 92 C. Maxim, A. Gogonel, I. Asavae, M. Asavae, and L. Cucu-Grosjean. Reproducibility and representativity: mandatory properties for the compositionality of measurement-based WCET estimation approaches. *SIGBED Review*, 14(3):24–31, 2017. doi:10.1145/3166227.3166230.
- 93 A. Melani, E. Noulard, and L. Santinelli. Learning from probabilities: Dependences within real-time systems. In *Proceedings of the IEEE Conference on Emerging Technologies Factory Automation (ETFA)*, pages 1–8, September 2013. doi:10.1109/ETFA.2013.6648013.
- 94 E. Mezzetti, M. Fernandez, A. Bardizbanyan, I. Agirre, J. Abella, T. Vardanega, and F. J. Cazorla. EPC Enacted: Integration in an Industrial Toolbox and Use Against a Railway Application. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2017.
- 95 E. Mezzetti, N. Holsti, A. Colin, G. Bernat, and T. Vardanega. Attacking the sources of unpredictability in the instruction cache behavior. In *Proceedings of the International Conference on Real-Time Networks and Systems (RTNS)*, 2008.
- 96 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. doi:10.4230/LITES-v002-i001-a001.
- 97 a. Milutinovic, j. Abella, i. Agirre, M. Azkarate-Askasua, E. Mezzetti, T. Vardanega, and F. J. Cazorla. *Software Time Reliability in the Presence of Cache Memories*, pages 233–249. Springer International Publishing, Cham, 2017. doi:10.1007/978-3-319-60588-3\_15.
- 98 S. Milutinovic, J. Abella, and F. J. Cazorla. Modelling Probabilistic Cache Representativeness in the Presence of Arbitrary Access Patterns. In *Proceedings of the IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing (ISORC)*, pages 142–149, May 2016. doi:10.1109/ISORC.2016.28.
- 99 S. Milutinovic, J. Abella, and F. J. Cazorla. On the assessment of probabilistic WCET estimates reliability for arbitrary programs. *EURASIP Journal on Embedded Systems*, 2017(1):28, April 2017. doi:10.1186/s13639-017-0076-8.
- 100 S. Milutinovic, J. Abella, E. Mezzetti, and F. J. Cazorla. Measurement-based Cache Representativeness on Multipath Programs. In *Proceedings of the Design Automation Conference (DAC)*, pages 123:1–123:6, 2018. doi:10.1145/3195970.3196075.
- 101 S. Milutinovic, E. Mezzetti, J. Abella, T. Vardanega, and F. J. Cazorla. On uses of extreme value theory fit for industrial-quality WCET analysis. In *Proceedings of the IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 1–6, June 2017. doi:10.1109/SIES.2017.7993402.
- 102 M. Panic, J. Abella, C. Hernandez, E. Quiñones, T. Ungerer, and F. J. Cazorla. Enabling TDMA Arbitration in the Context of MBPTA. In *Proceedings of the Euromicro Conference on Digital System Design (DSD)*, pages 462–469, August 2015. doi:10.1109/DSD.2015.68.
- 103 M. Panić, J. Abella, E. Quiñones, C. Hernandez, T. Ungerer, and F. J. Cazorla. Adapting TDMA arbitration for measurement-based probabilistic timing analysis. *Microprocessors and Microsystems*, 52:188–201, 2017. doi:10.1016/j.micpro.2017.06.006.
- 104 B. Paskdeloup. Static probabilistic timing analysis of worst-case execution time for random replacement caches. Technical report, Inria, 2014.
- 105 J. Pickands. Statistical Inference Using Extreme Order Statistics. *Ann. Statist.*, 3(1):119–131, January 1975. doi:10.1214/aos/1176343003.
- 106 E. Quiñones, E. D. Berger, G. Bernat, and F. J. Cazorla. Using Randomized Caches in Probabilistic Real-Time Systems. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pages 129–138, July 2009. doi:10.1109/ECRTS.2009.30.
- 107 F. Reghenzani, G. Massari, and Fornaciari W. chronovise: Measurement-Based Probabilistic Timing Analysis framework. *Journal of Open Source Software*, 3(28), June 2018. doi:10.21105/joss.00711.
- 108 J. Reineke. Randomized Caches Considered Harmful in Hard Real-Time Systems. *Leibniz Transactions on Embedded Systems*, 1(1):03–1–03:13, 2014. doi:10.4230/LITES-v001-i001-a003.
- 109 A. Rukhin, J. Soto, J. Nechvatal, E. Barker, S. Leigh, M. Levenson, D. Banks, A. Heckert, J. Dray, S. Vo, A. Rukhin, J. Soto, M. Smid, S. Leigh, M. Vangel, A. Heckert, J. Dray, and L. E. Bassham. Statistical test suite for random and pseudorandom number generators for cryptographic applications, NIST special publication, 2010.
- 110 L. Santinelli, F. Guet, and J. Morio. Revising Measurement-Based Probabilistic Timing Analysis. In *Proceedings of the IEEE Real-Time*

- and *Embedded Technology and Applications Symposium (RTAS)*, April 2017.
- 111 L. Santinelli and Z. Guo. *On the Criticality of Probabilistic Worst-Case Execution Time Models*, pages 59–74. Springer International Publishing, Cham, 2017. doi:10.1007/978-3-319-69483-2\_4.
  - 112 L. Santinelli, J. Morio, G. Dufour, and D. Jacquemart. On the Sustainability of the Extreme Value Theory for WCET Estimation. In *Proceedings of the Workshop on Worst-Case Execution Time Analysis (WCET)*, pages 21–30, 2014. doi:10.4230/OASIcs.WCET.2014.21.
  - 113 M. Santos, B. Lisper, G. Lima, and V. Lima. Sequential Composition of Execution Time Distributions by Convolution. In *Proceedings of the Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS)*, pages 30–37, November 2011. URL: <http://www.es.mdh.se/publications/2215->.
  - 114 C. Scarrott and A. MacDonald. A review of extreme value threshold estimation and uncertainty quantification. *REVSTAT-Statistical Journal*, 10(1):33–60, 2012.
  - 115 M. Schlansker, R. Shaw, and S. Sivaramakrishnan. *Randomization and associativity in the design of placement-insensitive caches*. Hewlett Packard Laboratories, 1993.
  - 116 K. P. Silva and R. Silva de Oliveira L. F. Arcaro. On Using GEV or Gumbel Models when Applying EVT for Probabilistic WCET Estimation. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, 2017.
  - 117 M. Slijepcevic, M. Fernandez, C. Hernandez, J. Abella, E. Quiñones, and F. J. Cazorla. pT-NoC: Probabilistic Time-Analyzable Tree-Based NoC for Mixed Criticality Systems. In *Proceedings of the Euromicro Conference on Digital System Design (DSD)*, 2016.
  - 118 M. Slijepcevic, C. Hernandez, J. Abella, and F. J. Cazorla. Boosting Guaranteed Performance in Wormhole NoCs with Probabilistic Timing Analysis. In *Proceedings of the Euromicro Conference on Digital System Design (DSD)*, pages 440–444, August 2017. doi:10.1109/DS.2017.71.
  - 119 M. Slijepcevic, L. Kosmidis, J. Abella, E. Quiñones, and F. J. Cazorla. DTM: Degraded Test Mode for Fault-Aware Probabilistic Timing Analysis. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pages 237–248, July 2013. doi:10.1109/ECRTS.2013.33.
  - 120 M. Slijepcevic, L. Kosmidis, J. Abella, E. Quiñones, and F. J. Cazorla. Time-analyzable non-partitioned shared caches for real-time multicore systems. In *Proceedings of the Design Automation Conference (DAC)*, pages 1–6, June 2014. doi:10.1145/2593069.2593235.
  - 121 J. E. Smith and J. R. Goodman. A Study of Instruction Cache Organizations and Replacement Policies. In *Proceedings of the 10th Annual International Symposium on Computer Architecture, ISCA '83*, pages 132–137, New York, NY, USA, 1983. ACM. doi:10.1145/800046.801648.
  - 122 J. E. Smith and J. R. Goodman. Instruction Cache Replacement Policies and Organizations. *IEEE Transactions on Computers*, C-34(3):234–241, March 1985. doi:10.1109/TC.1985.1676566.
  - 123 Z. Stephenson, J. Abella, and T. Vardanega. Supporting industrial use of probabilistic timing analysis with explicit argumentation. In *Proceedings of the IEEE International Conference on Industrial Informatics (INDIN)*, pages 734–740, July 2013. doi:10.1109/INDIN.2013.6622975.
  - 124 N. Topham and A. Gonzalez. Randomized cache placement for eliminating conflicts. *IEEE Transactions on Computers*, 48(2):185–192, February 1999. doi:10.1109/12.752660.
  - 125 N. Tracey, J. Clark, K. Mander, and J. McDermid. An automated framework for structural test-data generation. In *Proceedings 13th IEEE International Conference on Automated Software Engineering*, pages 285–288, October 1998. doi:10.1109/ASE.1998.732680.
  - 126 N. Tracey, J. A. Clark, and K. Mander. The way forward for unifying dynamic test-case generation: The optimisation-based approach. *Proceedings of the IFIP International Workshop on Dependable Computing and Its Applications (DCIA)*, 1998.
  - 127 D. Trilla, C. Hernandez, J. Abella, and F. J. Cazorla. Resilient random modulo cache memories for probabilistically-analyzable real-time systems. In *IEEE International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pages 27–32, July 2016. doi:10.1109/IOLTS.2016.7604666.
  - 128 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 *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 397–402, March 2015.
  - 129 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 *Proceedings of the IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 241–248, June 2013. doi:10.1109/SIES.2013.6601497.
  - 130 J. Wegener and F. Mueller. A Comparison of Static Analysis and Evolutionary Testing for the Verification of Timing Constraints. *Real-Time Systems*, 21(3):241–268, November 2001.
  - 131 J. Wegener, H. Sthamer, B. F. Jones, and D. E. Eyres. Testing real-time systems using genetic algorithms. *Software Quality Journal*, 6(2):127–135, June 1997. doi:10.1023/A:1018551716639.
  - 132 I. Wenzel, R. Kirner, B. Rieder, and P. Puschner. Measurement-Based Timing Analysis. In *Leveraging Applications of Formal Methods, Verification and Validation*, pages 430–444, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.
  - 133 R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdin-

- and, 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*, 7(3):36:1–36:53, May 2008. doi:10.1145/1347375.1347389.
- 134 N. Williams. WCET measurement using modified path testing. In *Proceedings of the Workshop on Worst-Case Execution Time Analysis (WCET)*, volume 1 of *OpenAccess Series in Informatics (OASICS)*, 2005. doi:10.4230/OASICS.WCET.2005.809.
- 135 N. Williams and M. Roger. Test generation strategies to measure worst-case execution time. In *ICSE Workshop on Automation of Software Test*, pages 88–96, May 2009. doi:10.1109/IWAST.2009.5069045.
- 136 M. Ziccardi, E. Mezzetti, T. Vardanega, J. Abella, and F. J. Cazorla. EPC: Extended Path Coverage for Measurement-Based Probabilistic Timing Analysis. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, pages 338–349, December 2015. doi:10.1109/RTSS.2015.39.

## A Appendix: Measurement Protocols

In this appendix, we briefly discuss *measurement protocols* and test vector generation, since they underpin approaches to measurement-based timing analysis. We use the term *scenario* of operation to mean a potentially repeating sequence of runs of a program, starting from a feasible initial hardware state and progressing via a valid evolution of the program’s input values. To run a program over a particular scenario of operation, a measurement protocol needs to provide information about the initial hardware configuration, which is used to set up the initial hardware state, and a sequence of input values, referred to as a *test vector*, which are input to the program as it iterates over a number of runs. The aim of a *measurement protocol* is to exercise the program in ways that are *relevant* to the parameter being measured. For example, if the parameter being measured is a code coverage metric, then the measurement protocol would aim to use a set of scenarios and test vectors designed to exercise paths that cover all of the statements (or all of the conditions) in the code. In the case of the WCET, the set of scenarios and test vectors need to be designed to exercise the longest paths through the code, assuming that those paths can be identified in some way. The major difficulty in designing an appropriate measurement protocol is in choosing which scenarios and hence which test vectors to use.

Search-based techniques were successfully applied to the problem of automatic test data generation for structural code coverage by Tracey et al. [125] in 1998. While measurement protocols designed for code coverage can potentially provide a useful starting point for the WCET problem, in general even MC-DC coverage is insufficient. Further, full path-coverage is typically unattainable due to issues of tractability, although some programs for high integrity systems may be simple enough that all paths can be covered. Structural coverage offers a more attractive starting point for hybrid measurement-based analysis which records execution times at the level of simple functions or sub-programs as discussed by Deverge and Puaut [42] in 2005; however, there are still issues with how these execution times are combined due to dependences on the previous history of execution.

Search techniques developed by Wegener et al. [131] in 1997, Tracey et al. [126] in 1998, Wegener and Mueller [130] in 2001, and multi-criteria optimisation developed by Bate and Khan [10] in 2011 have also been investigated in the context of test data generation with the aim of finding input values that result in large execution times. The basic idea is to use an evolutionary algorithm to mutate or evolve a population of test data, with the fitness function determined by the execution time of the program with that data as input. Multi-criteria optimisation works in a similar way, but takes into account additional criteria such as the number of cache misses as well as the execution time.

In 2005, Williams [134] proposed a static analysis tool that seeks to determine a set of test vectors that exercise every path. This was later extended in 2009 by Williams and Roger [135] with the aim of avoiding the need for full path coverage, for example by maximising loop counts.

In 2008, Wenzel et al. [132] described a tool for measurement-based timing analysis that uses a combination of test data reuse, random search, heuristics and model checking. It partitions the program into user defined segments (to ease path complexity), with instructions inserted at segment boundaries to ensure a consistent hardware state. In 2011, Bunte et al. [21] showed that commonly used metrics for functional testing including statement coverage, decision coverage, and MC/DC coverage are *insufficient* to obtain safe WCET estimates. Instead, they propose a balanced path metric which ensures that all feasible pairs of basic blocks are exercised in combination. This metric is shown to be much more effective than the common code coverage metrics, but is still not completely safe. Later in 2011, Bunte et al. [24] explored the combination of model checking, used to produce a set of test vectors that provide basic block coverage, and a genetic algorithm, which aims to modify these test vectors to maximise execution times. They evaluated this combined approach using the relative safety metric developed previously [21].

Recent work by Law and Bate [80] in 2016, aimed at maximising loop bounds and achieving path coverage at the level of individual functions. The techniques proposed make use of simulated annealing in combination with fitness functions that target branch coverage and loop counts as well as execution time. They show that this approach is more effective in providing WCET approximations than fitness functions based solely on maximising execution time.

While these methods are effective in finding large execution times, there are no guarantees that test data which results in the worst-case execution time will be found.

Further related work in 2017 by Braams et al. [20] presented EDiFy, a measurement-based framework that aims to derive the execution time distribution of a program via exhaustive evaluation of the program inputs. Since the execution time distribution depends on the distribution of input values, the input value distribution is assumed to be provided for each independent input variable and also the conditional distribution for any dependent variables. EDiFy addresses issues of tractability via a combination of static analysis and an anytime algorithm. Static analysis is used to reduce the state-space by pruning irrelevant input variables, and clustering variable ranges where the execution time is guaranteed to be the same. The anytime algorithm makes use of a logarithmic traversal function over the variable ranges. This ensures rapid convergence and an early tight approximation of the execution time distribution. Execution times are obtained by running the program with the selected input values.

Although we have touched upon issues of test data generation and measurement protocols in the above discussion a comprehensive review of research in this area is outside of the scope of this survey. As far as we are aware, to date there has only been very limited work done on the problem of defining appropriate measurement protocols to support MBPTA. The majority of works on MBPTA (see Section 4) aim at analysing single paths and focus on obtaining sufficient observations for the path under analysis. They then rely on additional knowledge to identify the worst-case paths or existing functional testing to provide sufficient path coverage. A pWCET distribution for the program is then constructed using an envelope over the pWCET distributions for individual paths, i.e. the per-path method (see Section 2.3). Cucu-Grosjean et al. [34] note that full path coverage is a pre-requisite to obtaining sound results from MBPTA. This is backed up by the empirical work of Lesage et al. [85] which shows that omitting some paths can quickly degrade the estimated pWCET distribution output by MBPTA leading to optimistic (i.e. unsound) results. Hybrid methods (see Section 5) go some way to addressing the path coverage problem; however, they are limited in their applicability. Further research is clearly needed to define appropriate measurement protocols that can fully support MBPTA methods, while also addressing issues of representativity.