Upper-bounding Program Execution Time with Extreme Value Theory

Francisco J. Cazorla[†,‡], Tullio Vardanega[*], Eduardo Quiñones[†], Jaume Abella[†]

[†]**Barcelona Supercomputing Center** [‡]**Spanish National Research Council (IIIA-CSIC)**
[*]**University of Padova**

Abstract

In this paper we discuss the limitations of and the precautions to account for when using Extreme Value Theory (EVT) to compute upper bounds to the execution time of programs. We analyse the requirements placed by EVT on the observations to be made of the events of interest, and the conditions that render safe the computations of execution time upper bounds. We also study the requirements that a recent EVT-based timing analysis technique, Measurement-Based Probabilistic Timing Analysis (MBPTA), introduces, besides those imposed by EVT, on the computing system under analysis to increase the trustworthiness of the upper bounds that it computes.

1998 ACM Subject Classification [D2.4] **Software Engineering**: Software/Program Verification

Keywords and phrases WCET, Extreme Value Theory, Probabilistic, Deterministic

Digital Object Identifier 10.4230/OASIcs.WCET.2013.61

1 Introduction

Extreme Value Theory (EVT) can be regarded as the counterpart of Central Limit Theory [7]: where the latter studies the bulk of the population of a given distribution, EVT studies the tail of it, in other words the extreme deviations from the median of probability distributions. By analysing a sample of observations of a given random variable, EVT determines the probability of extreme events to occur, where "extreme" refers to either end of the range of the value domain of those events. EVT is widely used in many disciplines, ranging from structural engineering to Earth sciences.

EVT has also been used to provide estimates of average-case execution time [17][16] and Worst-Case Execution Time (WCET) of software programs [8][5], which is the focus of this paper. In contrast to classic static WCET analysis, EVT computes a cumulative distribution function, or probabilistic WCET (pWCET) function, that upper-bounds the execution time of the program, guaranteeing that it only exceeds the given bound with a probability lower than some threshold (e.g., 10^{-15} per run).

EVT is applied in for measurement based timing analysis (MBTA). In MBTA, execution time measurements of the timing behaviour of the program of interest are processed by specialised EVT-based analysis to generate pWCET bounds for the program that should hold at deployment time.

In order for sound results to be obtained from the application of EVT it must hold that the observations of the events of interest can be regarded as random variables that are independent and identically distributed (i.i.d.). When this property cannot be asserted a-priori, it can be verified a-posteriori by suitable statistical tests [7]. However, EVT has nothing to say on the *representativeness* of those data, that is, on the safeness of the pWCET estimate that is computed from them. Representativeness is determined by the quality of the data passed to EVT, or analogously, by relevant properties of the environment that generated those data. The pWCET estimates obtained with EVT are therefore valid only for the data population sampled for the analysis or, by extension, for the operating conditions subsumed by those data.

If representativeness is *low*, the pWCET bounds obtained with EVT do *not* provide any trustworthy prediction on the timing behaviour of the program on the target platform, but rather a description of the extreme timing behaviour that the program might have in the execution conditions exercised during the observation runs. If representativeness is high instead, more general conclusions can be drawn on the computed pWCET when the program is run on the target platform under execution

13th International Workshop on Worst-Case Execution Time Analysis (WCET 2013).
Editor: C. Maiza; pp. 61–70

OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

conditions beyond those exercised during the analysis. The primary goal of MBTA techniques is to provide pWCET estimates that hold under execution conditions that may occur during the operation of the system: whereas those conditions may not be exactly identical to those captured by the observation runs made at analysis time, they should still represent them probabilistically. How this critical property can be asserted is the subject of this paper. This ability solves one of the key problems that real-time system designers have in trying to determine the timing behaviour of a real-time system. In this regard we show the benefits that can be achieved in terms of representativeness when using a time-randomised execution platform like the one proposed in the PROARTIS project [3][10][13], in contrast to conventional time-deterministic architectures[1].

In this paper we discuss the requirements on the use of EVT in computing pWCET bounds and the representativeness of the pWCET estimates. We show that increasing representativeness requires: (1) controlling the execution conditions at analysis time; and (2) understanding the representativeness of the analysis-time execution conditions with respect to those that may occur during operation. We discuss how *measurement-based probabilistic timing analysis* (MBPTA) based on EVT [5] reduces the burden placed on the user of the method for controlling the execution conditions. To that end, MBPTA requires that the effects that can be exercised by the execution conditions on the observation runs made during analysis are: either (1) bounded from above so that they represent worst-case effects; or (2) time-randomised; or else (3) ensured to have exactly the same probabilistic distribution at analysis *and* at deployment.

Contribution: This paper establishes the principles on which EVT can be used to derive WCET estimates in time-deterministic and time-randomised architectures. In particular, it helps WCET analysis community better understand the requirements, limitations and benefits that EVT carries on the determination of pWCET bounds, also understanding the requirements that MBPTA adds on top of EVT. This will set the baseline for future works in this promising area of research.

2 An executive introduction to EVT

EVT provides a canonical theory for the (limit) distribution of normalised maxima of independent, identically distributed random variables. EVT involves non-parametric statistics, that is, EVT does not rely on data (i.e. the population under study) belonging to any particular distribution. EVT describes the behaviour of extremal events for stochastic processes that evolve dynamically in time and space. It gives the user a methodology for predicting the occurrence of rare events. Estimating distribution tails beyond the limit of available data is a complex process that requires making mathematical assumptions on the tail model. These assumptions are very difficult if at all possible to verify in practice. There is thus intrinsic risk in choosing the tail model to fit the problem at hand, which is necessary to correctly apply EVT.

We are interested in the high values that bound the pWCET, so we consider the EVT prediction for maximal values of a set of observations. There are two main approaches to EVT. The first, known as Peak over Threshold method [2], models a distribution of excess over a given threshold: EVT shows that the limiting distribution of exceedance is a Generalised Pareto Distribution or GPD. The second approach, which we use, known as Block Maxima model (BMM) [7] considers the largest (smallest) observations obtained from successive *periods* (blocks), where the selection of the block size is a critical parameter. Under BMM the asymptotic distribution of the maxima (minima) is modelled and the distribution of the standardised maxima is shown to follow one of the Gumbel, Frechet or Weibull distributions [7]. The generalised extreme value distribution (GEV) is a standard form of these three distributions.

[1] A system is time-deterministic when we can determine its state at any time t on the basis of the initial state, inputs and the time cost of the state transition triggered by those inputs (read more on this in Section 4).

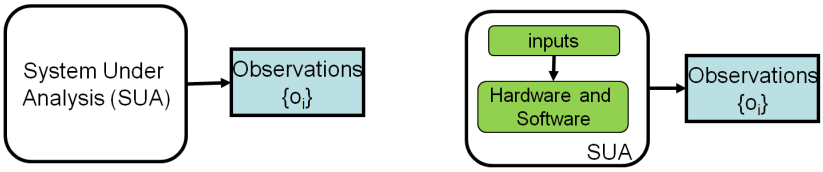

Figure 1 (a) system under analysis as assumed by EVT; (b) a computing system for analysis with EVT.

3 Requirements on the use of EVT for WCET estimation

When used to predict the extreme (hence worst-case) timing behaviour of applications running on a computing system (platform), EVT is given in input a number of *observations* taken from real execution of the system of interest. In our case, these observations are execution time measurements from runs of the program of interest, taken, under controlled execution conditions (hence, during analysis) on the target platform. From these observations EVT infers an approximation of the tail of the timing behaviour of the program that hold during actual operation.

EVT requires observations coming from the system under study to be described mathematically as random variables that are i.i.d. [7]: Two random variables are said to be independent if they describe two events such that the occurrence of one event does not have any impact on the occurrence of the other event. Two random variables are said to be identically distributed if they have the same probability distribution function. In our application of EVT, identical distribution holds when those two random variables describe the same system using the same set of parameters in the same way (whether deterministically or probabilistically), for all inputs with influence on the timing behaviour of the program, including input vectors and initial hardware and software state.

EVT does not describe how the input observations are made: it regards the system as a black box of which it is only interested in considering the external manifestations (observation of runs here) which have to be i.i.d. for theory to apply, see Figure 1(a). Observations evidently describe *some* behaviour of the system: however, as argued in this paper, it must also be ensured that the execution conditions under which those observations are taken at analysis time, do represent the execution conditions that will occur at deployment time, for which the computed pWCET estimates are required to provide a safe upper bound. This requirement imposes significant overhead on the user in: (1) understanding the execution conditions that the programs of interest may experience at deployment time; and (2) controlling the execution conditions during analysis time so that they are significantly representative. If not fulfilled properly, these two obligations can make the whole approach of deriving estimates based on EVT utterly misleading.

3.1 Defining the population under analysis

One of the most critical steps in applying EVT is understanding and capturing the population (universe) to be analysed, including the features of that population that are relevant for the analysis. In our case, the population is given by all the runs that the program of interest will perform at deployment time, and the feature of interest is the execution time of those runs. For instance, considering a program to be deployed on an aircraft, each run of that program in it during its whole operational lifetime is an individual of the population of interest. If the aircraft had a lifetime of 25 years, flying 80% of that time, and the program under analysis had an execution period of 100 ms (i.e., 10 times per second), the total population would be comprised of nearly 63×10^8 elements.

In general, the total population of events in a real-world system is inordinately large and hard to determine, so it cannot be fully enumerated at analysis time. User intervention is therefore needed to cap the population of interest to a treatable dimension, in a trustworthy, timely and cost-effective manner. This step requires understanding the sources of influence on the feature of interest of the population, which means understanding what factors in the system affect the execution time of any given run of the programs under study. We call those factors: *sources of execution time variability*.

Sources of execution time variability

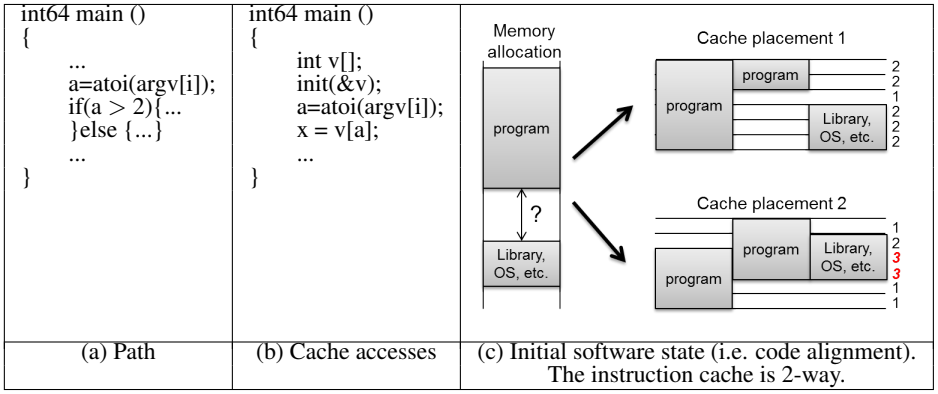

Figure 3 Examples of sources of execution time variability: programs for which the input vector affects: (a) the execution path; (b) the cache access pattern; (c) the cache jitter caused by the initial program state.

The execution time of a program is affected by several main factors, namely, the input conditions and the internal logic (state-dependent behaviour) of the platform (i.e. its hardware and software resources underneath the application layer), see Figure 1(b). Those factors represent what we call the Sources of Execution Time Variability ($SETV$) of the program. Each combination of values in the $SETV$ defines one particular *execution condition* under which a given run of the program may occur.

In our discussion here "input conditions" are understood to refer to 3 main components, see Figure 2: the input vector; the initial state of the software (for the program and the operating system, OS, underneath it); and the initial state of the hardware. Let us look at each of them in isolation.

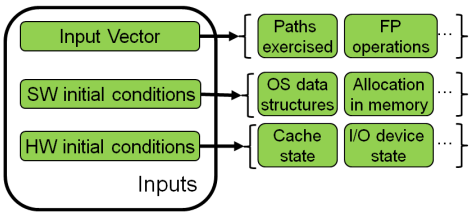

Figure 2 Main input components.

Input vector is the complete description of all the input data passed to the program which may affect the program execution behaviour. Input vectors may affect the hardware and the software state in all resources that are sensitive to history of execution. At software level, the most evident manifestation of the influence of input vectors is the execution path taken by the program during a given run. This is shown in Figure 3(a) where a is an element of the input vector. At hardware level, we find a variety of cases in which input vectors affect the timing behaviour of some components: Figure 3(b) shows the case where the memory address of a cache access (whose outcome may thus be a hit or a miss) depends on an input value.

Software initial state includes the initial state of all the read-write (state-sensitive) data structures used by the program directly or indirectly, the latter being those used by the operating system underneath it. It also includes all external aspects of the program that may have an influence on its execution-time behaviour: with processor resources such as the cache, the location in memory of the program determines the cache placement and the consequent cache conflicts of each memory access. Figure 3(c) shows a program using some external software components such as libraries or OS structures. Those components are allocated independently in memory and therefore, their relative alignment in cache can vary. As shown, different memory allocations (left) may lead to different cache placements (right). We show two particular cache placements: in the first one (top right) no cache set requires more than two lines to store all code, whereas in the second one (bottom right) two cache sets require up to 3 cache lines. Hence, if the instruction cache is 2-way set-associative, the former allocation is likely to produce better performance than the latter.

Hardware initial state includes the initial state of all the hardware resources (e.g., cache state) used by the program or any other software invoked by the program. Those states are important as the operation logic of many hardware and software components is state-dependent. The authors of [12] show that the initial state of the cache can be exploited to decrease WCET bounds by considering how cache contents may survive across subsequent disjoint executions of the program of interest.

Max population

We mentioned earlier that the execution time of a program is affected by several sources of execution time variability: $\{SETV_1, SETV_2, ..., SETV_j, ...SETV_n\}$, where source $j \in \{1, ..n\}$ can take up to k distinct values, that is: $SETV_j = \{v^j\}_k = \{v_1^j, v_2^j, ...v_k^j\}$. Consider for instance a multiplication unit whose response latency depends on its operands: the $SETV$ for this resource can take 2^m values, where m is the number of multiplications that occur in the program.

The sources of variability include all system inputs (input vectors, and SW and HW initial conditions)[2]. Controlling all the $SETV$ for each individual (i.e. measurement run) in the population is unfeasible as the target population is given by all the runs of all calls of the program under study that occur during operation.

The user has to derive a *population of maxima* (*max population*) that provides a safe upper bound of the target population, see the top part of Figure 4(a). If that is granted, then the execution times (i.e. individuals in the max population) safely upper bound the execution times in the actual population. In principle this requires analysing in detail each $SETV_j, \forall j \in \{1, ..n\}$. For each $SETV_j = \{v^j\}_k$ the user has to derive a subset of values $max(SETV_j) = max(\{v_1^j, v_2^j, ...v_k^j\}) = \{mv_1^j, mv_2^j, ...mv_{mk}^j\}) = \{mv^j\}_{mk}$. Eventually, based on the simplifying assumption of independence, we would need $max(SETV_j)$ to contain only its worst-case values. Identifying them may be so complex, however, that the user may have to resort to safe upper bounds. Moreover, to make the problem tractable, $mk < k$ should also hold as using $max(SETV_j)$ in place of $SETV_j$ serves the purpose of decreasing the size of the population of interest.

The actual cardinality of the max population is defined by the number of combinations of the max values that each $SETV$ can take during operation. If all $SETV$ are independent of one another, there is an individual in the max population for each element in the Cartesian product of all max values of all $SETV$: $\{mv^1\}_{mk} \times \{mv^2\}_{ml}, \times ... \times \{mv^n\}_{mm}$. Therefore, there is one execution time individual in the max population for a run under the execution environment defined by each of the combinations of all $SETV$. Otherwise, if some $SETV$ are not independent, then unfeasible combinations should be removed from the Cartesian product of all max values of all $SETV$. How to identify those combinations may be very hard.

3.2 Achieving i.i.d. behaviour

Ensuring that the observations submitted to EVT fulfil the required i.i.d. properties is only possible if the whole system behaves as a *random process* such that the observations drawn from it exhibit those properties. Given that the execution times of the programs under study are affected either by the input vectors passed to them or the internal logic (state-dependent behaviour) of the platform, the required i.i.d. properties on the observations drawn from the computing system must be obtained by: either (1) applying a random process on the way inputs are selected; or (2) applying random processes in the internal behaviour of the hardware/software (the platform). In both cases, WCET estimates are based on measurements (observations) taken from the execution of the program under study on the target platform. Hence, both approaches can be regarded as *measurement-based* rather than *static* as it is the case with other timing analysis techniques [18].

Next we discuss each of the two approaches in more detail, paying special attention to the requirements they place on the user as well as on the execution platform itself.

4 Deriving WCET estimates on time-deterministic systems with EVT

Current computing systems can be regarded as *time-deterministic*. Time-determinism is achieved when we can determine the state of the system at any time *t* on the basis of the initial state, the inputs and the time cost of the state transitions triggered by those inputs. The property of time determinism

[2] For the purposes of this paper we assume $SETV$ to operate independently of one another. Later in this section we show how this simplifying assumption can be removed.

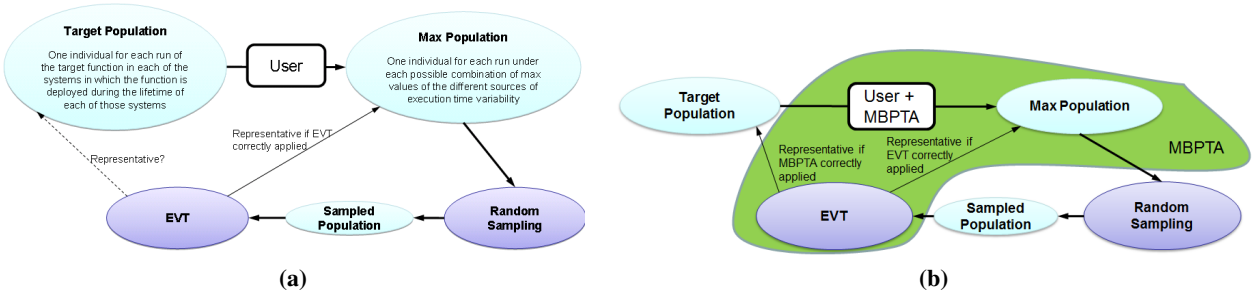

Figure 4 Application of EVT (left) and MBPTA on a time-randomised platform (right).

is disjoint from that of *functional determinism* in that a functional deterministic system can also be non time-deterministic: consider for example a system whose functionality can be fully described by a finite state machine. Further assume that the transition time from any two states *S0* to *S1* is a random value in the range [t1:t2]. This system is functionally deterministic in that it will always finish in state *S1* from state *S0*, but it is not time-deterministic because we cannot determine the exact time at which the transition from *S0* to *S1* will complete.

More specifically to our discussion, the execution time of a program on a time-deterministic system is constant across different runs that occur under the same execution conditions, i.e. the initial state in the hardware and software is fixed and the same input data are used across runs.

The behaviour of the system (the program running on the computing system) should behave like a *random process*, X_n, such that the i.i.d. statistical properties required for EVT can be had. If EVT is applied to a COTS (commercial off-the-shelf) deterministic computing system, the system cannot be changed for it to behave as X_n. In that case EVT conformance can only be sought by operating on the inputs submitted to it, as captured in Figure 2, in particular by randomising the selection of inputs using *random sampling*.

4.1 Random sampling

A sample is a set of individuals chosen from the population under study. A *random sample* is a sample chosen by random sampling from the population.

Several unbiased sampling methods exist, e.g, simple random sampling (SRS) [4]. An unbiased random selection of individuals is important so that, in the long run, the sample has the same statistical properties as the population under study. With SRS, each individual is chosen randomly and entirely by chance, hence providing independence between each picking of an individual. Further, each individual has the same probability of being chosen and each subset of k individuals has the same probability of being chosen as any other subset of k individuals. Hence, by construction, SRS constructs i.i.d. random samples.

However, as shown earlier, the target population under study cannot be fully constructed in the general case, and therefore it cannot be sampled. The max population is sampled instead. In our scenario this has an important corollary in the dimension of *representativeness of the WCET estimates computed with EVT*. If and only if the max population is a safe upper bound of the (deployment-time) target population, the WCET estimates that can be obtained with EVT actually upper bound the timing behaviour of the program of interest during operation. Hence, a critical step to achieve the required representativeness is to understand the deployment-phase target population, correctly defining the max population. Failing to do so would cause the WCET estimates obtained with EVT to lack any statistical representativeness with respect to the deployment-phase population. Under that scenario, the testing-time behaviour observed cannot be used by EVT to derive WCET estimates of the deployment-time behaviour of the program on the target system.

4.2 Deriving a safe max population

This step is one of the most complex passages in the procedure of applying EVT to derive pWCET estimates that hold at deployment time for the timing behaviour of the programs of interest. This step requires analysing in detail each of the $SETV_j$, ensuring that all the components of $max(SETV_j)$

take their respective maximum value(s), or value(s) that upper-bound their maximum. Indeed, it must be the case that the individuals leading to the pWCET must be part of the combination of $max(SETV_j)$ values or the upper bounds thereof. This condition holds if no timing anomalies occur across $SETV$, so that the combination of local worst cases in $max(SETV_j)$ leads to the global worst case. Some SETV are well understood by the user who can then control and bound them, either quantitatively or qualitatively, as we discuss below. Others are harder to enumerate and consequently difficult to bound from above.

Qualitatively-boundable SETV. The input vectors are the most challenging case of SETV, for the effect they have on the execution paths taken by the program. A poor path coverage may cause the results computed by EVT to miss the safeness quality required for worst-case timing analysis. To use EVT the user should understand what the worst-case paths in the program are and providing input vectors that exercise them. And the pWCET estimates derived with EVT would only be valid for the paths observed during analysis.

Tool support may be available to compute the coverage obtained during observation runs and report it back to the user. If a path that the tool deems possible is not yet covered in the observation runs, the tool may request that the user either acknowledges the exclusion of that path (which may be asserted as irrelevant for WCET estimation) or provides further test cases to exercise that path. This is technically possible because full path coverage at source level (possibly reduced to MC/DC [9]) is one of the prerequisites to be satisfied by functional testing in certified systems. This type of timing analysis should thus be performed in conjunction with the functional verification campaign.

Quantitatively-boundable SETV. The input vectors do not only affect the execution path followed in a program, but may also influence other resources with jitter, hence creating another $SETV$, causing an inordinate increase in their quantity. Returning to the multiplier whose response latency depends on its input values, the user should provide input vectors for each program path in which the distribution of multiplier input values upper-bounds the one that can occur during operation. If for example the multiplier takes 1 cycle when one operand is zero and 4 cycles otherwise, then the user is required to ensure that the distribution of non-zero values in the analysis input vectors upper bounds the distribution that may occur upon deployment.

Uncontrollable SETV. Controlling all $SETV$ to a level in which all their values can be known and a subset of them (max values) can be forced to occur in the computing system is generally out of reach for the user. Assume for example that the only $SETV$ is the location in memory where objects are placed (e.g., the program data and instructions, libraries, OS data and instructions, etc.) since this determines cache behaviour. In this scenario, enumerating all combinations of object placements in memory is simply unfeasible. As shown in [11], the number of memory alignments leading to different cache placements is $s^{n_{obj}-1}$, where s is the number of cache sets and n_{obj} the number of memory objects. Moreover, the user has limited control to force a given alignment. It therefore follows that some $SETV$ are hard to define, understand and control by the user.

4.3 Summary

EVT requires that the observations taken during analysis warrant independence and identical distribution. For a population of maximum values on which SRS is applied, independence can be preserved by controlling how experiments are made. This requires ensuring that: (1) no source of dependence can exist between end-to-end runs; and (2) no state-dependent effect occurs in the processor *and* no logical software-level state is allowed to pass between any two runs. Whether that dependence may exist across events or instructions within a run is irrelevant so long as observations are collected at the granularity of end-to-end runs.

The fact that each element (an individual in the max population) has the same probability of being chosen and each subset of k individuals has the same probability of being chosen for the sample as any other subset of k individuals, ensures identical distribution. However, EVT by itself does not ensure the representativeness of the max population with respect to the *actual* population, which may not be fully known by the user and hence not be sampled. EVT makes no claim on how safe

the selected "maximum population" is for the purposes of upper-bounding the actual population of system events of interest.

Ensuring that the computed pWCET estimates are safe upper bounds of the target population requires controlling all $SETV$. Our view here is that attempting to provide WCET estimates on COTS deterministic systems is fatally limited by the intricate dependences of the $SETV$ and the hardware/software support to control $SETV$ to a level in which all their values can be known, the maximum can be identified and forced in the computing system to carry out pertinent runs to feed EVT. There is therefore a risk of taking as valid for the program EVT projections whose representativeness cannot be assessed beyond the particular set of inputs used during the analysis. To apply EVT in MBTA, therefore, the user must be provided with means to derive max population safely, timely and cost-effectively.

5 Deriving WCET estimates on time-randomised systems with EVT

In order to ensure that the computed pWCET estimates are safe upper bounds to the execution time of the program at deployment time, MBPTA adds further constraints to those imposed by EVT [5], see Figure 4(b). While the EVT requirements only concern the nature of the observations, MBPTA requires controlling the inputs submitted to the program and the platform on which the runs occur, as shown in Figure 1(b). In particular, MBPTA requires that all the $SETV$ are "controlled" in one of the following ways.

1. **Safe upper-bounding of $SETV$ with no HW change.** The user provides values for each and every $SETV$ to upper-bound the effect that those $SETV$ can have on the execution time of the program during operation. For instance, for "reasonable" replacement policies, an empty initial cache state represents the worst-case state that a program may find at start up. Similarly, for path-dependent effects, the user needs to ensure that the paths of interest to pWCET estimation are traversed during the observation runs.
2. **Removal of $SETV$ with HW change.** Some $SETV$ are hard, if at all possible, to be effectively controlled by the user. For those $SETV$, the processor hardware, and to a lesser extent, the software, should be redesigned such that the response time jitter of the corresponding execution resources does not depend anymore on the relevant $SETV$. This approach is tantamount to removing the $SETV$ for those resources. We consider two ways in which this can be had[3]:
   a. **Worst case timing behaviour.** At software level this approach consists in forcing relevant software components (e.g., methods, procedures) to always take the same time to execute regardless of when they are called. This approach has been followed for OS calls [1], where the jittery part of the required activity is deferred until after the return from the call, in the interstices between the execution of application programs, taking care to not incur disturbing perturbations to hardware state left on return. At hardware level, the worst-case mode [15] is a feature forcing a hardware block to take its worst delay even if a particular request finishes earlier. Both features make the observations obtained at analysis time be a safe upper bound of the deploy-time behaviour of the program.
   b. **Time randomisation.** Forcing the worst-case response of some hardware/software components to occur at all times may degrade performance significantly (e.g., considering all cache accesses as misses). Instead, randomising the timing behaviour of a given resource, significantly improves performance of that resource and makes observations taken at analysis time to be representative of the deployment time behaviour. If enough observations of the execution of that resource are taken, the observed frequencies converge to the actual probabilities. Further, randomisation may help removing some of the $SETV$ as shown in coming section.

[3] A third way exists if the user can provide inputs that have the same distribution of values, with respect to a given characteristic under analysis, as those that can appear during operation. This is possible but exceedingly difficult.

5.1 How to achieve time randomisation

One of the main challenges in the context of MBPTA is understanding whether timing may be randomised for some hardware resources, while of course leaving functional behaviour unaffected. Time randomisation removes some of the $SETV$ that affect the execution time of programs, thereby reducing the burden on the user in applying EVT. For the sake of illustration we focus on the case of the cache, though the principles we present apply to any other resource.

We have seen earlier that the memory layout of program data and code is a $SETV$ that affects the program execution time. It is well known that the location in which program data and code are placed in memory may vary across runs (or upon composition with other software). This in turn means that the particular sets in which the different data/instructions are located vary across runs. An important consequence of that phenomenon is that the conflicts in cache that a program suffers and their effect on execution time may vary across runs. This is illustrated in the example in Figure 3(c), which shows the case of the code placement in memory and its effect on the instruction cache.

In a real-world scenario, expecting the user to control the way in which program data and code are placed in memory is not practical: tools and methods exist to do so [14], but they place some burden on the user. MBPTA can be facilitated by randomising the placement policy, and optionally the replacement policy (which is not covered in this discussion), to cause the effect of placement and replacement policies to become probabilistically analysable. The basic idea is to break the causal dependence between the particular address in memory in which a piece of data is, and the particular cache set in which it is allocated. This dependence is broken by *randomising the placement* such that the index set of individual data items is randomised and made vary across runs. In this way, making enough runs, hence taking enough observations, is sufficient for the user to provide probabilistic evidence of the effect of placement on execution time. And more importantly, in each run the user needs not control the particular location in memory in which individual data items are located. Random placement can be done at hardware level by changing the design of the cache [10] or at software level by controlling the way data and code are loaded in memory [11].

Overall, in a MBPTA-compliant platform (i.e. one that conforms to the principles we illustrated) all $SETV$ are under control, whereby: (1) their effect on the execution time observations taken from the program is known; (2) the representativeness of the observed execution times of the program as affected by those $SETV$ at analysis time is guaranteed with respect to the execution times of the program at deployment time; and (3) the observed execution times can be regarded as independent and identically distributed random variables so that EVT can be meaningfully applied.

6 Related work

Extreme-value statistics are used in [6] to model the WCET. To select highest execution times the cited authors use Block Maxima [7], for which instead the authors of [8] use the Peak Over Threshold method. In [19] the authors apply EVT to the problem of computing Worst-Case Response-Time (WCRT) of programs. Older papers focus on the application of EVT on time-deterministic platforms but do not cover the representativeness of the result obtained from EVT, that is the safeness of the pWCET estimate. The authors of those works assume that the observations collected are representative of the *target population*. In this paper we have shown the difficulties of achieving representativeness in time-deterministic platforms.

The authors of [5] present MBPTA as well as the requirements that it imposes on the hardware and software components of the system to increase the trustworthiness of the upper bounds computed by MBPTA, primarily the time randomisation of certain processor resources.

7 Conclusions

While EVT has been regarded as a powerful method to derive upper (lower) bounds on arbitrary distributions, its utilisation for deriving WCET estimates has not been well described yet. In this paper we review those characteristics of WCET estimation relevant for the use of EVT. In particular, we present how the initial population for EVT should be generated, including all sources of execution time variability (SETV), and how this process can be hopeless on time-deterministic platforms.

Conversely, we show that time-randomised platforms enable an effective use of EVT by means of MBPTA by randomising and upper-bounding the timing behaviour of some hardware and software resources so that some SETV do not need to be described by the user while WCET estimates obtained are still sound.

Acknowledgements

The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under the PROARTIS Project (www.proartis-project.eu), grant agreement no 249100. This work was partially supported by EU COST Action IC1202 "Timing Analysis On Code-Level (TACLe)", and by the Spanish Ministry of Science and Innovation under grant TIN2012-34557. Eduardo Quiñones is partially funded by the Spanish Ministry of Science and Innovation under the Juan de la Cierva grant JCI2009-05455.

References

1 A. Baldovin, E. Mezzetti, and T. Vardanega. A time-composable operating system. *WCET Workshop*, 2012.

2 J. Beirlant, Y. Goegebeur, J. Segers, and J. Teugels. *Statistics of Extremes: Theory and Applications*. 2004.

3 F.J. Cazorla, E. Qui nones, T. Vardanega, L. Cucu, B. Triquet, G. Bernat, E. Berger, J. Abella, F. Wartel, M. Houston, L. Santinelli, L. Kosmidis, C. Lo, and D. Maxim. Proartis: Probabilistically analysable real-time systems. *ACM TECS*, 2012.

4 W.C. Cochran. *Sampling Techniques*. 1977.

5 L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzeti, E. Quinones, and F.J. Cazorla. Measurement-based probabilistic timing analysis for multi-path programs. In *ECRTS*, 2012.

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

7 W. Feller. *An introduction to Probability Theory and Its Applications*. 1996.

8 J. Hansen, S Hissam, and G. A. Moreno. Statistical-based wcet estimation and validation. In *the 9th International Workshop on Worst-Case Execution Time (WCET) Analysis*, 2009.

9 Hayhurst K.J., Veerhusen D.S., Chilenski J.J., and Rierson L.K. A practical tutorial on modified condition/decision coverage. Technical report, 2001.

10 L. Kosmidis, J. Abella, E. Quinones, and F.J. Cazorla. A cache design for probabilistically analysable real-time systems. In *DATE*, 2013.

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

12 L. Kosmidis, E. Quinones, J. Abella, T. Vardanega, and F.J. Cazorla. Achieving timing composability with measurement-based probabilistic timing analysis. In *ISORC*, 2013.

13 E. Mezzetti and T. Vardanega. On the industrial fitness of wcet analysis. *WCET Workshop*, 2011.

14 E. Mezzetti and T. Vardanega. A rapid cache-aware procedure positioning optimization to favor incremental development. In *RTAS*, 2013.

15 M. Paolieri, E. Quinones, F.J. Cazorla, G. Bernat, and M. Valero. Hardware support for WCET analysis of hard real-time multicore systems. In *ISCA*, 2009.

16 P. Radojković, P.M. Carpenter, M. Moretó, A. Ramirez, and F.J. Cazorla. Kernel Partitioning of Streaming Applications: A Statistical Approach to an NP-complete Problem. In *MICRO*, 2012.

17 P. Radojković, V. Čakarević, M. Moretó, J. Verdú, A. Pajuelo, F.J. Cazorla, M. Nemirovsky, and M. Valero. Optimal Task Assignment in Multithreaded Processors: A Statistical Approach. In *ASPLOS*, 2012.

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

19 L. Yue, T. Nolte, I. Bate, and L. Cucu-Grosjean. A statistical response-time analysis of real-time embedded systems. In *the 33rd IEEE Real-time Systems Symposium*. IEEE, December 2012.