Microprocessors and Microsystems 89 (2022) 104419

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

Hybrid probabilistic timing analysis with Extreme Value Theory and Copulas

Levent Bekdemir [a], Cüneyt F. Bazlamaçcı [b,*]

[a] Aselsan, Inc., Ankara 06200, Turkey
[b] Department of Computer Engineering, Izmir Institute of Technology, Izmir 35430, Turkey

ARTICLE INFO

Keywords:
Worst-case execution time analysis
Measurement-based probabilistic timing analysis
Copula theory
Extreme Value Theory (EVT)
Statistical timing analysis

ABSTRACT

The primary challenge of time-critical systems is to guarantee that a task completes its execution before its deadline. In order to ensure compliance with timing requirements, it is necessary to analyze the timing behavior of the overall software. Worst-Case Execution Time (WCET) represents the maximum amount of time an individual software unit takes to execute and is used for scheduling analysis in safety-critical systems. Recent studies focus on statistical approaches, which augments measurement-based timing analysis with probabilistic confidence level by applying stochastic methods. Common approaches either utilize Extreme Value Theory (EVT) for end-to-end measurements or convolution techniques for a group of program units to derive probabilistic upper bounds for the program. The former method does not ensure path coverage while the latter suffers from ignoring possible extreme cases. Furthermore, current state-of-the-art convolution methods employed in a commercial WCET analysis tool overestimates the results because of using the assumption of worst-case dependence between basic blocks. In this paper, we propose a hybrid probabilistic timing analysis framework and modeling the program units with EVT to capture extreme cases and use Copulas to model the dependency between the units to derive tighter distributional bounds in order to mitigate the effects of co-monotonic assumptions.

1. Introduction

The most distinguishing characteristic of safety-critical real-time systems is to give correct response within their strictly defined deadline. In order to satisfy safety related requirements of these systems, timing characteristics of software units must be estimated to quantify safety confidence along with the system level requirements.

Recent advances in technology arose challenges in timing analysis. Performance enhancing features of modern complex processors such as *multi-cores*, *shared buses*, *pipelines*, *out-of-order execution*, *branch prediction* and *caches* made the execution time dependent to the execution history. These architectural improvements decrease the feasibility of conventional static analysis techniques and cause jittery response times. The variability in execution times makes statistical methods to be applicable in WCET analysis. Statistical analysis provides WCET estimates with increased confidence without the necessity of full path coverage, thus allowing us to obtain only a few measurements to estimate the worst timing behavior of the system.

Convolution of probability distributions and Extreme Value Theory (EVT) are two main approaches in statistical timing analysis domain. Current state-of-the-art convolution approaches generate new paths that might be infeasible to reach in reality, which leads to overestimation in the results. On the other hand, most of the studies in the literature applies EVT to end-to-end measurements of the analyzed programs, which suffers from the path coverage problem during the analysis runs.

The aim of this paper is to propose an enhanced hybrid probabilistic timing analysis (HYPTA) framework for commercial off-the-shelf (COTS) platforms. Static structural information is extracted from the analyzed program by dividing it into functional blocks, which also decreases the probe effect resulting from instrumentation of the source code in basic block granularity. This structural information along with the collected measurements are used to construct a probabilistic WCET (pWCET) distribution by virtually generating new paths to upper bound all possible execution scenarios.

Current state-of-the-art solutions either assume an independence between the blocks or use a conservative convolution approach named as *biased convolution* to upper bound all possible dependence types between the blocks. However, neither of them reflects the real behavior of the programs. In fact, it is possible to model the dependence between the random variables by using Copulas. Copulas are joint probability distribution functions that have uniform marginals. Hence, they allow to simulate the joint behavior of the random variables by Monte-Carlo simulation technique, which eases to derive the probability distribution

* Corresponding author.
E-mail addresses: lbekdemir@aselsan.com.tr (L. Bekdemir), cuneytbazlamacci@iyte.edu.tr (C.F. Bazlamaçcı).

https://doi.org/10.1016/j.micpro.2021.104419
Received 9 September 2020; Received in revised form 29 November 2021; Accepted 21 December 2021
Available online 10 January 2022

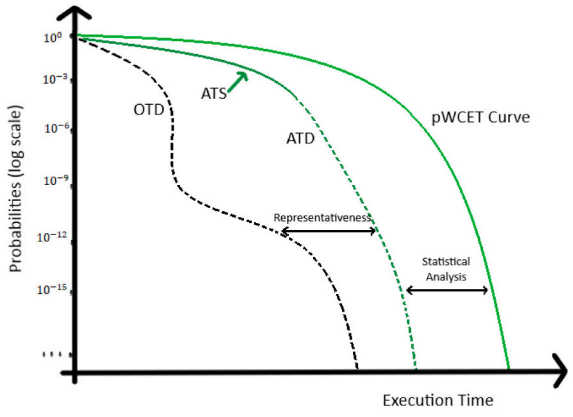


Fig. 1. Measurement-Based Probabilistic Timing Analysis (MBPTA) notions.

of their sum. Furthermore, there are very limited studies for COTS platforms in WCET domain.

In addition, each marginal of the n-dimensional probability distribution is represented by a parametric continuous Extreme Value distribution whenever possible. Since our proposed method divides the programs into functional blocks that tend to be multi-path sub-programs, it is possible to model those functions with EVT, thus allowing us to predict possible rare events based on its tail's behavior.

The proposed HYPTA framework can also be considered as an improvement over an existing popular commercial HYPTA solution named as RapiTime by Rapita Inc., which implements a conservative convolution mechanism [1]. Results of the case studies are compared with the results of the tool and widely accepted independent assumptions. It can be observed from the results that our approach decreases the overestimation by providing tighter bounds.

The remainder of the paper is organized as follows: Section 2 presents the background information and literature overview. The proposed hybrid framework with EVT and Copulas is introduced and the results of the case studies are given in Section 3. Section 4 concludes the paper including a summary of this study.

2. Background information

Probabilistic timing analysis (PTA) methods have emerged in order to mitigate some of the drawbacks of the existing solutions. Static Probabilistic Timing Analysis (SPTA) techniques seek to reduce the pessimism encountered due to having incomplete information about an underlying platform by expressing some of its behaviors probabilistically. Measurement-Based Probabilistic Timing Analysis (MBPTA) on the other hand, seeks to scientifically reason about the worst case events that are captured during the analysis phase by modeling the execution time behavior of the program based on widely accepted statistical methods. Output of the MBPTA is not a single valued WCET, but rather a probability distribution of the execution time profile of the program that is guaranteed to upper bound all analysis time observations. Those observations of course should upper bound operational execution times for reliability. Fig. 1 distinguishes the difference between such scenarios and presents fundamental notions of MBPTA.

MBPTA requires an analysis-time distribution (ATD) as input, which should always upper bound the operation-time distribution (OTD). Aiming to this purpose and to minimize the difference between the two is referred as *representativeness*. Applicability of MBPTA methods also require to represent the whole ATD with less number of samples, which are referred as *analysis-time samples (ATS)*. By using a computationally affordable testing scheme and a relatively small set of samples (ATS), pWCET distribution is calculated, which guarantees an upper bound to both OTD and ATD.

2.1. Extreme value theory

Extreme Value Theory (EVT) is considered as the building block of MBPTA [2], which is a branch of statistics that was designed to predict unusual natural events such as extreme floods, tornado outbreaks and earthquakes. These extreme events are modeled as probability distributions and EVT deals with the extreme deviations from the median of observations associated with the phenomena of interest. In line with [3] and [4], a typical implementation of EVT is described step-by-step as follows: (1) *Applicability Evidence* : identically distributed independent (i.i.d) input requirement of EVT is checked to determine whether the obtained samples are to be accepted or not, (2) *Data Selection* : Relevant values in a data set that represent the tail of the distribution are selected either by using Block Maxima (BM) or Peaks Over Threshold (PoT) approaches [5], (3) *Model Fitting* : Depending on the data selection mechanism, filtered values are either fit to a Generalized Extreme Value Distribution (GEV) or Generalized Pareto Distribution (GPD), (4) *Tail Extension* : Finally, by using the calculated parameters, the distribution of the sample tail is found. Inverse cumulative distribution function (ICDF) of the estimated probability distribution is then used to calculate extreme values for the given exceedance probability. Taking p as the desired probability of exceedance (e.g. 10^{-10}), $ICDF(p) = x$ gives the upper bound value with exceedance probability p.

The quality of the estimated distribution with EVT is a challenging entity to assess. At their recent comprehensive survey about probabilistic WCET analysis, Cazorla et al. state that no universal consensus exists to date on this issue [6]. Since the whole process is a statistical paradigm, it is reasonable to assess the reliability with some statistical tests. Some works on WCET domain [7,8] consider the estimates obtained from the EVT are reliable only if every hypothesis of the EVT is verified. On the other hand, other approaches as in [9] propose to use standard statistical tools such as Quantile–Quantile or Mean-Excess plots to evaluate the reliability of the estimated distributions.

2.2. Copula theory

In statistics, copulas are used to model the dependence of several random variables. A copula is basically a multivariate probability distribution with uniform marginals.

Let X_1 and X_2 be two different random variables with their cumulative distribution functions (cdfs) F_1 and F_2 defined as:

$$F_1(x_1) = P[X_1 \le x_1]$$

$$F_2(x_2) = P[X_2 \le x_2]$$

Let H be the joint cumulative distribution function of X_1 and X_2, defined as:

$$H(x_1, x_2) = F_{X_1,X_2}(x_1, x_2) = P[X_1 \le x_1, X_2 \le x_2]$$

Here, H represents all aspects of the joint behavior of the random variables, but it is hard to interpret the dependence structure out of H. Copulas make it possible to separate the dependence structure and the behavior of the marginals described by $F_1(x_1)$ and $F_2(x_2)$. It is worth noting that, dependence is different from correlation. Correlation is only one type of dependence, which is a straight line between two random variables.

Let I represent the interval $[0, 1]$. A d-dimensional copula C is a cumulative distribution function on I^d with uniform marginals with a general notation

$$C(\mathbf{u}) = C(u_1, u_2, \dots, u_d)$$

Sklar's theorem is the most important result regarding copulas, which describes the relationship between the joint distribution H and a copula C [10].

Let F be a d-dimensional joint distribution function with marginals $F_1, \ldots, F_d$. Then there exists a copula C on I^d such that, for all $x_1, \ldots, x_d$ in $\mathbb{R} = [-\infty, \infty]$,

$$F(x_1, \ldots, x_d) = C(F_1(x_1), \ldots, F_d(x_d))$$

or

$$C(u_1, \ldots, u_d) = F(F_1{}^{(-1)}(u_1), \ldots, F_d{}^{(-1)}(u_d))$$

where $F_1{}^{(-1)}, \ldots, F_d{}^{(-1)}$ represent the *quasi-inverse* of the marginal distribution functions.

2.3. Literature review

There are numerous research works related to measurement-based timing analysis (MBTA). In this work we primarily focus on the probabilistic variant of MBTA, which is surveyed excessively in recent papers of Cazorla et al. [6] and Davis et al. [11]. MBPTA was initially studied by Edgar and Burns [12]. In [13], the input of the program under test is randomly generated, execution time samples are collected and then modeled through Extreme Value distributions.

Hansen et al. [14] presented the use of *Block Maxima* method to estimate the pWCET through Gumbel distribution of a PowerPC platform with VxWorks RTOS. The estimated WCET values are then validated by collecting additional millions of measurements. They have shown that it is possible to safely predict the WCET values without the need for large collection of samples.

The work of Cucu-Grosjean et al. [4] is considered as the basis for MBPTA in the literature. They introduced the methodology of how to apply EVT for both single-path and multi-path programs on a simulator with random replacement cache. The authors point out that in order to satisfy the i.i.d. requirement for multi-path programs, it is reasonable to select the inputs randomly and group the observations sequentially.

Santinelli et al. [15] studies the effects of dependence between time observations for EVT on an Intel Xeon platform without a randomized cache or any type of bus. They experimentally evaluate that the execution time variability results from the underlying complex hardware architecture and is random enough for EVT applicability.

Cazorla et al. [16] defined the general properties of MBPTA with EVT for both time-deterministic and time-randomized platforms. The authors suggest to randomize the underlying platform to increase the confidence of the results as was studied later in [17]. However, they also mention that it may be possible to derive WCET estimates on time-deterministic platforms with EVT by randomly generating inputs for the multi-path programs.

Silva et al. [3] studied the pWCET estimation by implementing both GEV and Gumbel fitting methods. They used L-Moments approach to estimate the parameters for GEV and MLE approach for Gumbel. Observations are grouped with Block Maxima method. They collected few samples (10^6) to estimate the low probability execution times and tightness assessments are done by comparing the results with 10^8 samples that are taken from the system. Their method is later studied empirically on an Intel platform with Linux operating system in [18].

Years of efforts performed by G. Bernat and his team are transferred into a commercial tool as detailed in [19]. They initially developed a scope-tree representation method to estimate the WCET of the programs under test by mitigating the effects of conventional syntax tree representation [20]. Then, they introduced the state-of-the-art *biased convolution* method to derive the WCET of the whole program out of the observations of small blocks [1]. The development of pWCET tool in [21] followed. They also examined the use of Copulas for estimating an upper-bound on WCET [22]. This study forms the basis of our work. Throughout the years, the initial pWCET tool is observed to evolve into RapiTime WCET tool, which is widely used in the industry [23,24].

3. Hybrid Probabilistic Timing Analysis (HYPTA) with EVT and copulas

3.1. Current state-of-the-art

There are very few studies in HYPTA domain to mention [25–27]. All of these studies aim at increasing the path coverage for MBPTA. However, our main interest is on the RapiTime tool, which is developed by Rapita Systems Ltd.

Authors of [6] defines the RapiTime as a hybrid MBTA solution rather than hybrid MBPTA. They argue that the tool predicates the notion of pWCET, but it does not apply a predictive model to estimate the distribution. Thus, it should be considered much more as an SPTA with measurements rather than MBPTA. However, Davis et al. [19] states that the tool falls into the HYPTA category since it combines the static structural properties of the program under test with measurements, but its probabilistic approach is still questionable. Besides, one of the developers of the tool states that they follow a frequentist approach in order to determine the probability values of each measurement sample [21].

Park and Shaw [28] defined a set of rules called as the *timing schema* of the program in order to evaluate the WCET as a function of tree nodes and Bernat et al. [22] presented a probabilistic version of it. Since the execution time of the nodes are represented by random variables X_i, the problem reduces to the calculation of $(X_1 + X_2 + \cdots + X_n)$ for sequential blocks. If it is assumed that X_is are independent of each other, then the standard convolution of the distributions give the result easily. However, this assumption does not hold in reality especially for perfect positively (comonotonic) or perfect negatively (countercomonotonic) dependent cases.

Bernat et al. [22] in their study specifically aim to solve this problem by using copulas. The study proposes that the supremal convolution with the assumption of comonotonicity between blocks results in safe estimation for any type of dependence between them. Similar to supremal convolution, they branded Biased Convolution technique [1], which still relies on comonotonicity and used it in their tool RapiTime.

3.2. Open challenges for HYPTA

In one of their constituent papers [22], Bernat et al. state that the only acceptable method is comonotonic convolution for any type of dependence between basic blocks. They support this idea with some experiments, but the degree of overestimation is not mentioned. Actually the results in the paper do not seem to be overestimated because of the test scenarios that only covers the comonotonic and independent cases. However, for a countercomonotonic case (perfect negative dependence), the assumption of comonotonicity would result in a huge overestimation [1]. Consider the following program:

```
void f1(x) {
  for(int i = 0; i < x; i++) {
    //delay 1ms
  }
}
void f2(x) {
  for(int i = 0; i < 100 - x; i++) {
    //delay 1ms
  }
}
void testProgram() {
  f1(x);
  f2(x);
}
```

It is obvious that the functions $f1$ and $f2$ are negatively dependent to each other meaning that when one of them executes for (n) ms, the other one should execute for $(100 - n)$ ms, which results in $testProgram$ always executing for 100 ms. RapiTime tries to estimate the WCET of the $testProgram$ function by calculating the following expression.

$$W(testProgram) = W(f1) + W(f2)$$

Here, $W(f1)$ and $W(f2)$ are observed execution time distributions that are represented by random variables $f1$ and $f2$. Sequential addition of the random variables with comonotonic convolution results in:

$$W(testProgram) = \begin{pmatrix} p_1 & .. & p_n \\ 2 & .. & 200 \end{pmatrix}$$

The resulting distribution claims that the WCET of $testProgram$ could be 200 ms with a probability p_n or alternatively it claims that there could be an input x, which could lead the program through a worst-case path where both $f1$ and $f2$ executes for 100 ms. This assumption is far from reality and causes 100% overestimation of the actual WCET.

The second problem is related to the construction of Execution-Time Profiles (ETPs) for each basic block. The method is based on a frequentist approach meaning that the frequency of observation of each value is assigned as the probability of occurrence for each value. For example, consider the execution time observations for block A as

$X_A = \{10, 10, 11, 11, 11, 11, 12, 13, 14, 14\}$.

The ETP for block A is constructed as:

$$ETP(A) = \begin{pmatrix} 0.2 & 0.4 & 0.1 & 0.1 & 0.2 \\ 10 & 11 & 12 & 13 & 14 \end{pmatrix}$$

It is also stated that the distributions for blocks are discrete in nature. This statement holds for fine grained basic blocks since they are not exposed to huge variations. However, if the blocks are defined as functions, they are much vulnerable to input and hardware effects, thus their execution time behavior would result in an asymptotic tail.

In practice, it is not feasible to instrument every branching point within the program in order to construct ETPs for fine grained basic blocks due to increasing probe effects. The only reasonable solution for COTS platforms would be to increase the granularity of the blocks, which then brings the variance issue due to multi-path nature of coarse grained blocks.

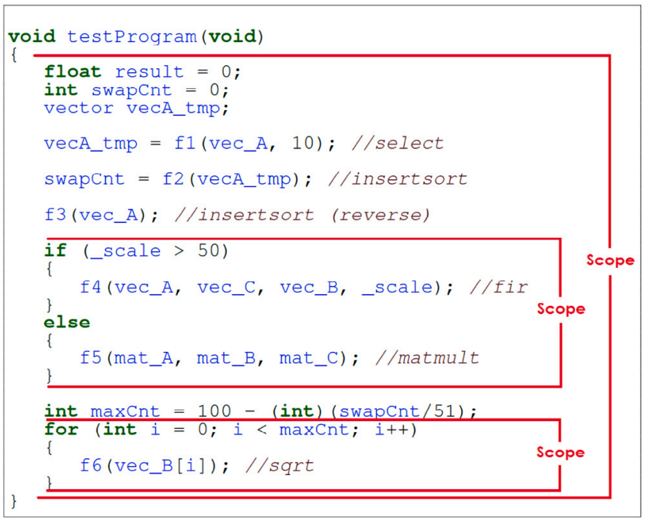
The overestimation resulting from the assumption of comonotonicity and the lack of rare event capturing when functional level instrumentation is performed has been identified as the main research focus of the present work.

3.3. Proposed method

Avdulaj in his MSc thesis [29] proposes a procedural approach to estimate the Value-at-Risk VaR for an empirical portfolio by using Extreme Value Theory and Copulas. The general steps that he followed are as follows: (1) Model each investment return with Semi Parametric Piecewise Distribution (SPD), which fits GPD to the tails and a kernel distribution to the intermediate part, (2) Transform each distribution into a uniform interval, (3) Fit a t-copula to uniform marginal distributions, (4) Generate huge number of uniform values from t-copula generator, (5) Convert uniform variates back to their original domain (Inverse Transform Sampling [30]), (6) Perform a weighted sum to estimate overall VaR, (7) Finally, the desired VaR value is calculated with the given confidence level α.

The steps after the $3rd$ one represents the Monte-Carlo Simulation steps. An analogy can easily be drawn between VaR analysis and the WCET analysis in our case. Returns of investments correspond to execution time of blocks and VaR corresponds to the overall probabilistic WCET of the program.

In our proposal, we follow a similar approach, but instead of using t-copulas to model the dependence, we use Vine Copulas. Although the t-copula allows to model symmetric tail dependencies in higher dimensions, it still relies on a single parameter. In fact, multivariate dependencies are not necessarily symmetric. Also when the dimensions become more complex, single parameter approach might fall behind. For these reasons, Vine Copula approach, which models the overall dependency by using pair-copulas and a tree model [31] is preferred.


```
void testProgram(void)
{
   float result = 0;
   int swapCnt = 0;
   vector vecA_tmp;

   vecA_tmp = f1(vec_A, 10); //select

   swapCnt = f2(vecA_tmp); //insertsort

   f3(vec_A); //insertsort (reverse)

   if (_scale > 50)
   {
      f4(vec_A, vec_C, vec_B, _scale); //fir
   }
   else
   {
      f5(mat_A, mat_B, mat_C); //matmult
   }

   int maxCnt = 100 - (int)(swapCnt/51);
   for (int i = 0; i < maxCnt; i++)
   {
      f6(vec_B[i]); //sqrt
   }
}
```

Fig. 2. Example scopes in a sample program.

We model each block of the program with Extreme Value distributions. However, there might be some cases when goodness-of-fit tests are not passed. In such cases historical simulation approach is followed, which corresponds to using frequentist ETPs. Modeling the blocks with one of the EVT distributions provides deriving predictive tail values while current state-of-the-art ETP approach only provides already observed discrete values.

The most essential step in our approach is to derive the copula model, which represents the dependency between the blocks of the analyzed program. In order to derive a copula, marginals must have the same dimensions, meaning that the observations of each block should be taken at the same time. This is an important criterion especially for conditional and iterative blocks since for a single run of the program, sequential blocks are visited once, but conditional or iterative blocks might be visited zero or several times, thus a special consideration is taken for the conditional and iterative blocks. A *scope* can either be the whole function body itself or a conditional or an iterative part of a function. Fig. 2 illustrates that a whole "if-else block" can represent a scope while a whole "for block" can represent another one. This approach is similar to Ermedahl et al. [32] to a certain extent. In [32], program blocks (basic blocks) are grouped as scopes augmented with flow facts to track loop counts or special conditions. Since our study does not aim to provide a full-fledged clustering mechanism to represent all types of programs, we are content to provide only an intuition and prerequisites in deriving scopes.

The steps of our approach are given below:

(1) Inject instrumentation points (IPoints) to the entry and exit points of functions in source code.
(2) Derive the timing schema of the program by using IPoints as explained in [33].
(3) Mark those functions that are called inside iterative and conditional blocks and also mark the scopes.
(4) Determine the random variables that represent the execution time of the functional blocks (Each node inside IPoint tree).
(5) From the most inner scope to the outer one, fit the suitable copulas for each scope that have sequential addition of random variables.
(6) Simulate next n execution of the scopes out of the copulas by using Monte-Carlo approach.
(7) Derive the inverse CDFs of each random variable.

(8) Transform each uniform margins generated from simulations into their original domain by using their inverse CDFs.
(9) Perform a sum operation for all the marginals to derive the overall distribution of the scope.
(10) Repeat steps 2 – 7 until no scope is left to be analyzed.

More details and codes about the procedure are available in [34].

3.4. Experimental evaluation

3.4.1. Experimental setup

The hardware platform used in this experimental evaluation phase is a LEON3FT based ASIC platform, which is a fault tolerant version of LEON3 SPARC V8 processor. Its general architecture is presented in Fig. 3.

LEON3FT is designed for embedded applications, combining high performance with low complexity and low power consumption. It runs at 64 MHz and supports most of the functionality of standard LEON3 processor including error detection and correction in on-chip RAM memories. First level instruction (IL1) and first level data (ID1) caches have 4 sets, 16k bytes/set and 32 bytes/line. The replacement algorithm in both caches is the Least-Recently-Used (LRU). The processor implements a 7-stage pipeline with Harvard architecture with the use of an efficient branch-prediction capability. It has a high-performance, fully pipelined Floating-Point Unit (FPU). The platform has a Memory Management Unit (MMU), but we chose to disable it since the underlying real-time operating system does not have virtual address translation capability to support MMU.

At the software side of the platform, RTEMS exists as the real-time operating system (RTOS). Traces are captured through a special mechanism, which outputs GPIO signals when instrumentation point hit occurs, and those signals are captured and timestamped by a custom made external hardware device in order to decrease the probe effects. Using such a solution is mandatory since it is necessary to obtain detailed traces instead of end-to-end measurements.

Most of the steps in this section are done in R environment, which is a freely available language and environment for statistical computing and graphics with powerful visualization capabilities [35].

3.4.2. Application procedure

In order to detail the steps, we constructed an example test program as follows:

```
void testProgram() {
  insertSortAsc(x); //Sort in ascending
      order
  insertSortDesc(x); //Sort in descending
      order
}
```

The given program is composed of two consecutive sorting functions both depending on the given input x. x is a randomly generated array that is composed of 500 float values. Since x is randomly generated for each run, their execution times would be negatively correlated. In addition, these functions are also input dependent multi-path programs, which have loops and floating point comparisons. For reproducibility purposes, the sorting function is realized by the *insertsort* program out of the Mälardalen WCET Benchmarks [36]. Statistics for 10 000 analysis runs observed by the help of functional instrumentation are given below.

It is observed from Table 1 that although both sorting functions execute for more than 260 ms, the observed maximum end-to-end execution time for testProgram is about 487 ms. This indicates that the functions are not perfectly positively correlated (comonotonic), thus the dependency type between them should be taken into account. The correlation plot given in Fig. 4 summarizes the case. Note that the values on both axes correspond to clock tick counts in which each tick represents 5 ns that is the resolution of our trace capturing unit.

testProgram function also has a self execution time, which is the execution time when the execution time of individual functions are excluded from end-to-end measurements. Since *testProgram* is composed of only 2 function calls, that self execution time is negligible, but we still take it into account.

Implementation of IPoint depends on the analysis environment. In our work, IPoint is a macro that writes the trace identifier value x to a general purpose I/O (GPIO) output register in order to capture from an external trace hardware to timestamp IPoint hits. The macro toggles the most significant bit in the register to high and eventually low to inform the external hardware that it finished writing trace identifier to the port (see Fig. 5).

Table 1
Execution time statistics of observations for *testProgram*.

Scope	*Minimum*	*Maximum*
insertSortAsc	217.4657 ms	272.0894 ms
insertSortDesc	212.7195 ms	266.9449 ms
testProgram	482.2895 ms	487.3415 ms

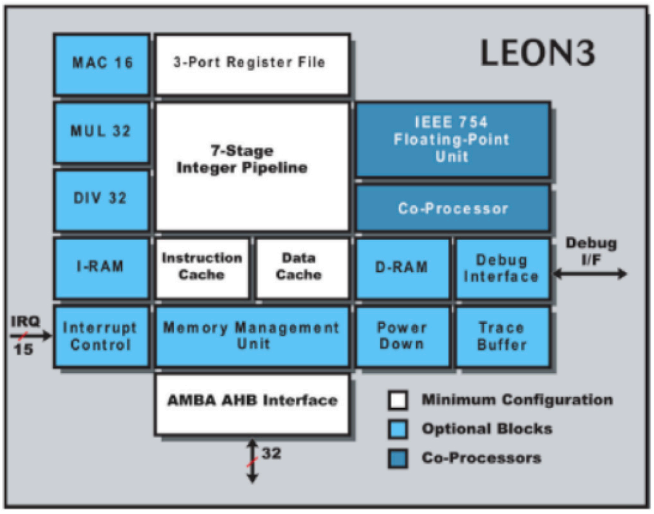


Fig. 3. LEON3 architectural blocks.

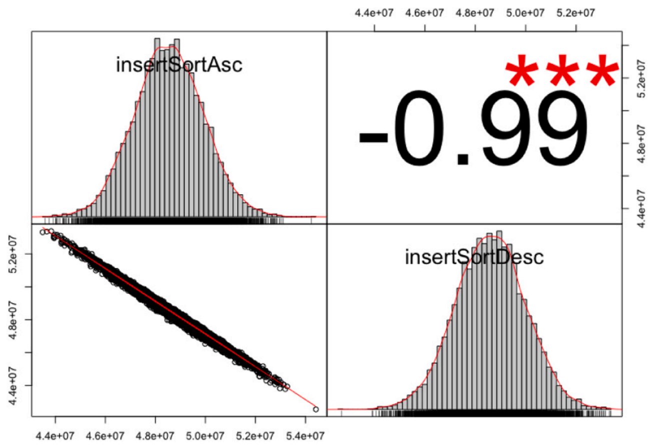


Fig. 4. Correlation plot of negatively dependent sorting functions.

```
#define IPoint(x)
{
 GPIOOutput = (x & 0x7F);
 GPIOOutput = GPIOOutput | 0x80;
 GPIOOutput = 0;
}
```

Fig. 5. Implementation of IPoint macro.

3.4.2.1. Timing schema of the program. The instrumented version of our test program is given below :

```
void insertSortAsc(x) {
  IPoint(29);
  //sort x in ascending order
  IPoint(28);
}
void insertSortDesc(x) {
  IPoint(27);
  //sort x in descending order
  IPoint(26);
}
void testProgram() {
  IPoint(31);
  insertSortAsc(x);
  insertSortDesc(x);
  IPoint(30);
}
```

The timing schema in the form of random variables and their corresponding IPoint pairs are shown in the following expressions:

$$W(insertSortAsc) \quad = W(IPoint_{29-28}) \tag{1}$$

$$W(insertSortDesc) \quad = W(IPoint_{27-26}) \tag{2}$$

$$W(testProgram_{entry}) \quad = W(IPoint_{31-29}) \tag{3}$$

$$W(insertSortAsc_{ret}) \quad = W(IPoint_{28-27}) \tag{4}$$

$$W(insertSortDesc_{ret}) \quad = W(IPoint_{26-30}) \tag{5}$$

Both $W(insertSortAsc)$ and $W(insertSortDesc)$ are equal to end-to-end execution time of the functions since they do not contain sub function calls inside. The total execution time of the $testProgram$ can be represented as:

$$W(testProgram) = \underbrace{W(testProgram_{self})}_{\text{X}} + \underbrace{W(testProgram_{sub})}_{\text{Y}} \tag{6}$$

where $W(testProgram_{sub})$ corresponds to the timing schema of the sub-function calls inside the $testProgram$ and $W(testProgram_{self})$ represents the timing schema of the main body of the program when sub-functions are excluded. Thus, they can be expressed as:

$$W(testProgram_{self}) = \underbrace{W(testProgram_{entry})}_{\text{X}} + \underbrace{W(insertSortAsc_{ret})}_{\text{Y}} + \underbrace{W(insertSortDesc_{ret})}_{\text{Z}} \tag{7}$$

$$W(testProgram_{sub}) = \underbrace{W(insertSortAsc)}_{\text{X}} + \underbrace{W(insertSortDesc)}_{\text{Y}} \tag{8}$$

3.4.2.2. Deriving the copulas. Each Eqs. (6)–(8) are composed of sub-parts that are represented by random variables X, Y and Z, which might be the nodes or leaves of the program structure tree. Each random variable represents the execution time of an inner scope of their parent scope. Thus, the problem reduces to finding the sum of n random variables where the dependency between them is unknown.

Considering Eq. (7) which have the highest number of variables, in order to derive the multivariate joint distribution $H(x,y,z) = P[X \leq x, Y \leq y, Z \leq z]$ which is necessary to finally derive the $J(t) = P[X+Y+Z \leq t]$, copulas are suitable since according to Sklar's theorem for any $(u,v,k) \sim U(0,1)$:

$$H(x,y,z) = C(F(x), G(y), M(z)) \tag{9}$$

$$C(u,v,k) = H(F^{(-1)}(u), G^{(-1)}(v), M^{(-1)}(k)) \tag{10}$$

where F, G and M correspond to the cumulative distribution functions of X, Y and Z, respectively. Additionally u, v and k should have the same dimensions, which in our case are the execution time observations for the corresponding scopes.

When $testProgram$ is run for 10 000 times, same amount of observations are collected for both $W(testProgram_{self})$ and $W(testProgram_{sub})$. Similarly, 10 000 samples are taken for both $W(insertSortAsc)$ and $W(insertSortDesc)$ when $W(testProgram_{sub})$ is visited 10 000 times. Same logic applies to the sub-scopes of the $W(testProgram_{self})$.

Vine Copula package of R is used to fit a copula to the marginal distributions, which in this case are X, Y and Z. To do this, first the marginals should be converted into uniform range. Original observations are converted into uniform distribution by using `pobs` function, which computes the pseudo-observations for the given data matrix. After the conversion, the correlation between the sub-scopes are shown in Fig. 6.

Note that the dependency between $u(pobs_{insertSortAsc})$ and $v(pobs_{insertSortDesc})$ in Fig. 6c coheres with Fig. 4. However, the dependency between the sub-scopes of $testProgram$ in Fig. 6a seems to be independent because there are no operations between the functions, which might result in any type of behavior. This situation also results in almost independence between the sub-scopes of $testProgram_{self}$ that is seen in Fig. 6b.

Next, `RVineStructureSelect` function of Vine Copula package is used to fit a suitable copula model to our program scopes. It also selects the tree structures that are appropriate for the pair-copula families. The output of the function in R environment is given in Fig. 7.

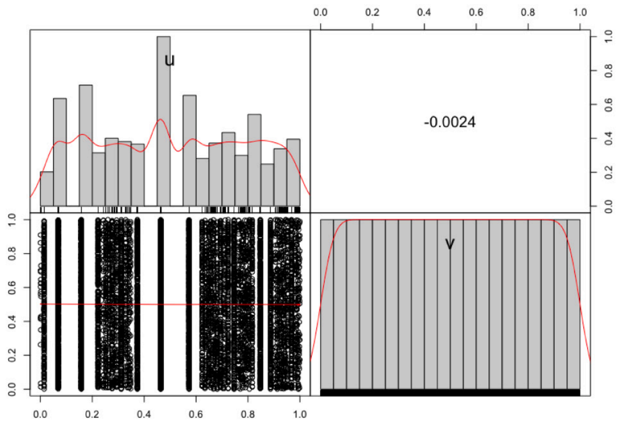
It can be observed from the output that an independence copula is fitted to the sub-scopes of $testProgram$ and $testProgram_{self}$ as expected. Gaussian copula with parameter -0.13 corresponds to the almost independent case. Important result is that a t-copula with parameter $\{-0.99\}$ is fitted to (u,v) pairs of $testProgram_{sub}$, which also coheres with the results given in Fig. 6c.

Next step is to check whether this copula structure is suitable for the given data sets. For this the `RVineGofTest` function is used which performs 15 different goodness-of-fit tests for R-Vine copula models. In our evaluation *ECP2* method is selected, which is a goodness-of-fit test based on the combination of probability integral transform (*PIT*) and empirical copula process (*ECP*) [37]. Results are not included here due to the space limitations, but all p-value results were above 0.05, which indicate that there is no significant evidence for rejecting the fitted copula for modeling the dependency between the sub-scopes of the given scopes.

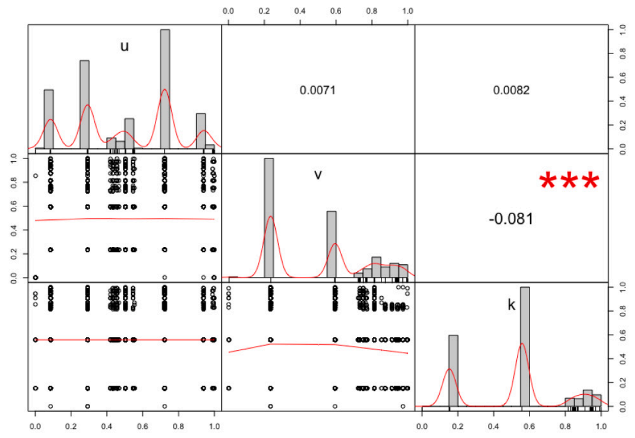
3.4.2.3. Simulation from the copulas. By using the obtained copula models, Monte-Carlo simulation steps are followed basically by generating uniform probability values randomly. To do this, `RVineSim` function is used, which generates the desired number of probability value pairs out of the given R-Vine copula model. In our experiments we generated 1.000.000 samples actually representing the next one million possible outcomes of the program scopes in accordance with the dependency model.

The results of the simulations are 1.000.000 x n matrices where n corresponds to the number of marginals of each copula. Simulated uniform margins are illustrated in Fig. 8.

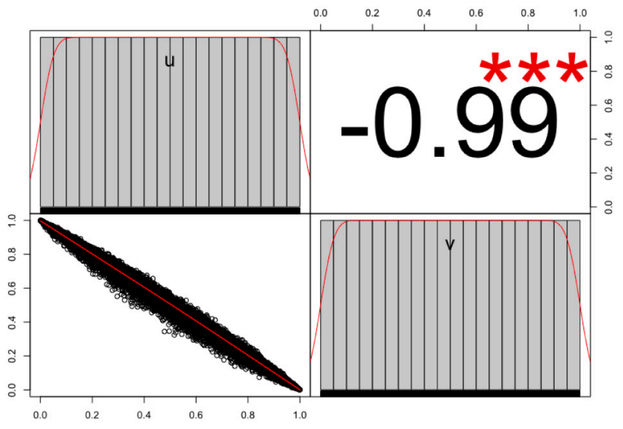
Those simulated uniform values represent possible outcome of the next execution of the corresponding scope. For example, the first line in Fig. 8c states that the next execution of $testProgram_{sub}$ would result



(a) $testProgram$



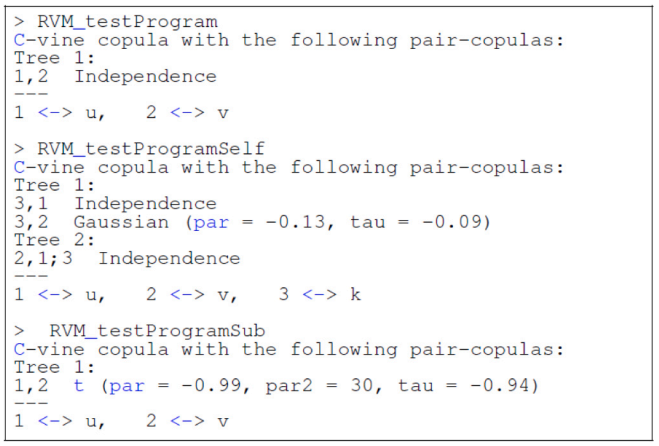
(b) $testProgram_{self}$



(c) $testProgram_{sub}$

Fig. 6. Correlation plot of the pseudo-observations.

in execution time of $insertSortAsc$ to be $F^{(-1)}(0.3318784754)$ and the execution time of $insertSortDesc$ to be $G^{(-1)}(0.675288088)$ where F and G corresponds to the CDFs of the scopes. Thus, it is important to derive the CDFs of each scope in order to construct a safe upper bound for the $testProgram$.



```
> RVM_testProgram
C-vine copula with the following pair-copulas:
Tree 1:
1,2  Independence
---
1 <-> u,   2 <-> v

> RVM_testProgramSelf
C-vine copula with the following pair-copulas:
Tree 1:
3,1  Independence
3,2  Gaussian (par = -0.13, tau = -0.09)
Tree 2:
2,1;3  Independence
---
1 <-> u,   2 <-> v,   3 <-> k

>  RVM_testProgramSub
C-vine copula with the following pair-copulas:
Tree 1:
1,2  t (par = -0.99, par2 = 30, tau = -0.94)
---
1 <-> u,   2 <-> v
```

Fig. 7. Copula fitting with RViceStructureSelect function in R.

3.4.2.4. Deriving CDFs of the segments. The total execution time of $testProgram$ that is represented with Eq. (6) can be derived by using Eqs. (9) and (10) :

$$C(u,v) = H(F^{(-1)}(u), G^{(-1)}(v)) = P[X \leq x, Y \leq y] \tag{11}$$

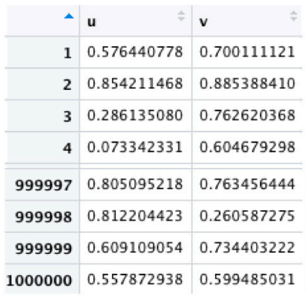
In order to construct the $P[X \leq x, Y \leq y]$ distribution, inverse CDFs $F^{(-1)}$, $G^{(-1)}$ must be found. $F^{(-1)}$ and $G^{(-1)}$ represents the inverse CDFs of the random variables X and Y, which represent the execution times of $testProgram_{self}$ and $testProgram_{sub}$, respectively. Obviously CDFs of the individual scopes are needed.

It is possible to model any random variable with a parametric or non-parametric probability distribution. However, modeling a random variable with a known parametric distribution based on observations is not always possible. Furthermore, random variables might be composed of several sub random variables as in $testProgram_{self}$ and $testProgram_{sub}$. In such cases, modeling them by using their end-to-end measurement observations would result in ignoring both extreme cases and the cases resulting from the hybrid analysis. The idea is to benefit as much as possible from the observations in each scope. For those situations where modeling a random variable with a known parametric extreme value distribution is not possible, we propose to use empirical CDFs instead, at the expense of resolution loss and incapability of prediction of capturing possible rare events.

In accordance with the explanation given above, $testProgram_{self}$ and $testProgram_{sub}$ parts of the $testProgram$ should not be modeled via an Extreme Value Distribution since they are composed of several subparts. It is instinctively known that parts of the $testProgram_{self}$ are not suitable to model with any type of Extreme Value Distribution since in reality there are no operations between the function calls and their execution time contribution is negligible. For these reasons and not to complicate the overall procedure, parts of the $testProgram_{self}$ are not considered to fit to a parametric continuous distribution, but their empirical CDFs are constructed out of historical observations made.

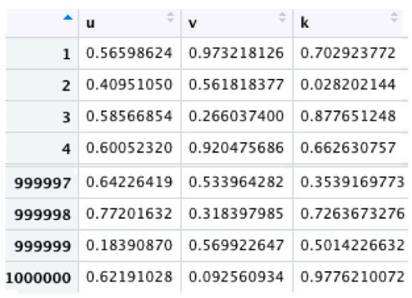
On the other hand, the sub-scopes of $testProgram_{sub}$ can be modeled by a parametric EV distribution since they are not composed of sub-scopes. The primary requirement of EVT is to derive whether the original data conforms i.i.d requirements. In line with Silva et al. [3], we followed the same approach for i.i.d tests and the results are given in the Fig. 9.

Results illustrate that both X and Y in Eq. (8) are suitable to be modeled with one of the EV distributions. X and Y can either fit to a GEV or GPD if only the rightmost tails are important. However, in this case lower tails are also significant since there might be a countercomonotonic situation where GEV/GPD would result in overestimation. Hence, in our proposed method we chose to model each random variable with Semi Parametric Piecewise Distribution which



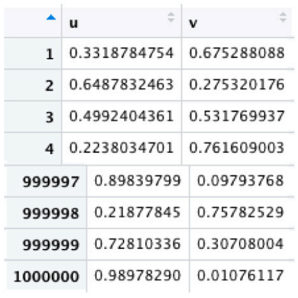
	u	v
1	0.576440778	0.700111121
2	0.854211468	0.885388410
3	0.286135080	0.762620368
4	0.073342331	0.604679298
999997	0.805095218	0.763456444
999998	0.812204423	0.260587275
999999	0.609109054	0.734403222
1000000	0.557872938	0.599485031

(a) $testProgram$



	u	v	k
1	0.56598624	0.973218126	0.702923772
2	0.40951050	0.561818377	0.028202144
3	0.58566854	0.266037400	0.877651248
4	0.60052320	0.920475686	0.662630757
999997	0.64226419	0.533964282	0.3539169773
999998	0.77201632	0.318397985	0.7263673276
999999	0.18390870	0.569922647	0.5014226632
1000000	0.62191028	0.092560934	0.9776210072

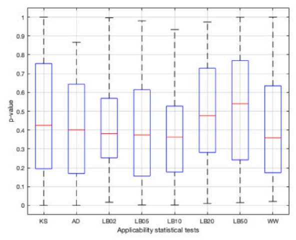
(b) $testProgram_{self}$



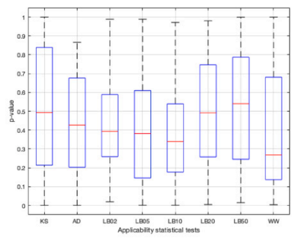
	u	v
1	0.3318784754	0.675288088
2	0.6487832463	0.275320176
3	0.4992404361	0.531769937
4	0.2238034701	0.761609003
999997	0.89839799	0.09793768
999998	0.21877845	0.75782529
999999	0.72810336	0.30708004
1000000	0.98978290	0.01076117

(c) $testProgram_{sub}$

Fig. 8. Simulated uniform margins from the fitted copula models.

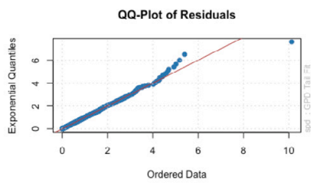


(a) insertSortAsc

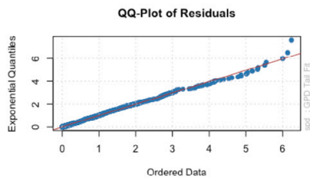


(b) insertSortDesc

Fig. 9. Statistical test results of the $testProgram_{sub}$ segments.



(a) insertSortAsc



(b) insertSortDesc

Fig. 10. QQ-Plots of the fitted piecewise GPD to $testProgram_{sub}$ segments.

fits a GPD to both upper and lower 10% tails of the distribution and fits a kernel distribution to the internals.

QQ Plots in Fig. 10 show that the fitted distributions are suitable enough to represent the whole observation data of the $insertSortAsc$ and $insertSortDesc$. Thus, `qspd` function of *spd* package serves to represent the $F^{(-1)}$ and $G^{(-1)}$ of X and Y for $testProgram_{sub}$.

Therefore, reverting back to Fig. 8c, possible next execution time of $testProgram_{sub}$ can be calculated as:

$$F^{(-1)}(0.3318784754) + G^{(-1)}(0.675288088) = 97151546 \tag{12}$$

where 97151546 is the clock tick count and it represents one possible outcome for $J(t) = P[X + Y \leq 97151546]$. Deriving the full $J(t)$ distribution is computationally intractable. In order to approximate the $J(t)$ distribution, samples must be generated as many as possible from the copula model. Applying the methodology in Eq. (12) to the whole matrix that is shown in Fig. 8c results in an approximate distribution of $J(t)$ by combining the Extreme Value Theory and Copulas.

Each random variable X and Y that represents the execution time of $insertSortAsc$ and $insertSortDesc$, respectively are modeled with an Extreme Value distribution in order to capture the rare cases instead of using empirical CDF.

3.4.2.5. Estimating the total distribution. The equations to derive the total execution time distribution for $testProgram$ becomes:

$$W_i(testProgram_{sub}) = F^{(-1)}(u_i) + G^{(-1)}(v_i) \tag{13}$$

where:

- i: $1, \dots, 1.000.000$
- $W_i(testProgram_{sub})$: is the execution time distribution of $testProgram_{sub}$
- $F^{(-1)}$: is the inverse CDF of fitted SPD model of X in Eq. (8)
- $G^{(-1)}$: is the inverse CDF of fitted SPD model of Y in Eq. (8)
- u_i: are the simulated uniforms from the copula for $insertSortAsc$
- v_i: are the simulated uniforms from the copula for $insertSortDesc$

$$W_i(testProgram_{self}) = F^{(-1)}(u_{(i)}) + G^{(-1)}(v_{(i)}) + M^{(-1)}(k_{(i)}) \tag{14}$$

where:

- i: $1, \dots, 1.000.000$
- $W_i(testProgram_{self})$: is the execution time distribution of $testProgram_{self}$
- $F^{(-1)}$: is the inverse ECDF of X in Eq. (7)
- $G^{(-1)}$: is the inverse ECDF of Y in Eq. (7)
- $M^{(-1)}$: is the inverse ECDF of Z in Eq. (7)
- $u_{(i)}$: are the simulated uniforms from the copula for $testProgram_{entry}$
- $v_{(i)}$: are the simulated uniforms from the copula for $insertSortAsc_{ret}$

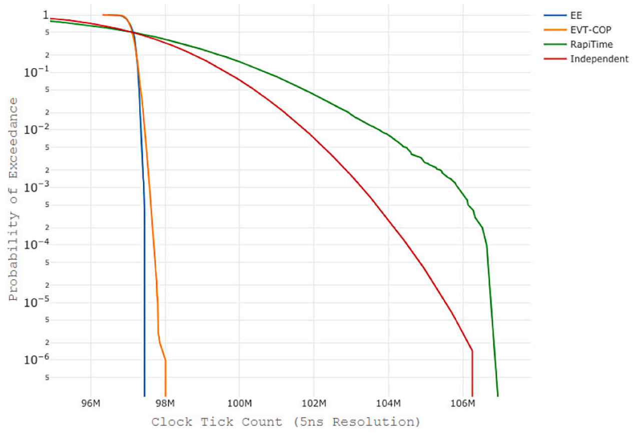


Fig. 11. Estimated pWCET distributions for $testProgram$.

- $k_{(i)}$: are the simulated uniforms from the copula for $insertSortDesc_{ret}$

Finally,

$$W_i(testProgram) = F^{(-1)}(u_i) + G^{(-1)}(v_i) \tag{15}$$

where:

- i: $1, \ldots, 1.000.000$
- $W_i(testProgram)$: is the execution time distribution of $testProgram$
- $F^{(-1)}$: is the inverse of $W(testProgram_{sub})$
- $G^{(-1)}$: is the inverse of $W(testProgram_{self})$
- u_i: are the simulated uniforms from the copula for $testProgram_{self}$
- v_i: are the simulated uniforms from the copula for $testProgram_{sub}$

Note that F, G and M in Eq. (14) and F and G in Eq. (15) are step functions hence they do not have unique inverse functions. Therefore, their *empirical quantile functions* are defined as the right inverse of the CDFs. The quantiles are fetched from the inverse CDFs with the help of `findInterval` function of R.

Evaluating the calculations given in Eqs. (13)–(15) results in a distribution shown in Fig. 11, which is compared with the observed end-to-end (EE) measurements and independent case and comonotonic case that implemented for our $testProgram$ in RapiTime tool.

Clearly, the biased convolution method (comonotonic assumption) and the standard convolution method (independent assumption) overestimate the results by a factor of 10% in the worst case. On the other hand, our approach based on EVT and Copulas provides tighter upper bound for the overall $testProgram$. The overestimation factor of 10% may not seem to be significant, but this only happened for a very simple program that has only two consecutive function calls inside. This factor is likely to increase rapidly when the program structure is composed of more complex blocks such as iterative and conditional ones.

3.5. Case study

We constructed a synthetic and reproducible benchmark program in which the functions are selected from the Mälardalen WCET Benchmarks again [36]. The code is given in Fig. 2.

The inputs vec_A, vec_B, mat_A and mat_B are randomly generated by employing *Mersenne Twister* pseudo-random number generation method, which has been proven to be “more random” than the built-in generators in many common programming languages including C [38]. In fact, vec_A and mat_A are the same objects, the only difference is their dimensions where vec_A is $1\ x\ 100$ float array and mat_A is $10\ x\ 10$ float matrix. This is also valid for vec_B and mat_B. vec_C and mat_C are initially arrays that are used to store the results.

The total execution time of this program can also be represented by Eq. (6).

However, the expressions $W(testProgram_{self})$ and $W(testProgram_{sub})$ are not easily decomposed in this case as in Eqs. (7) and (8). That is because we have conditional and iterative blocks that should be carefully examined.

In our proposed method, conditional and iterative blocks are handled as one scope at the highest level of the program scope. This approach is necessary in order to derive the copulas that models the dependency between the sub-scopes. This is best explained by the following expressions:

$$\begin{aligned} W(testProgram_{sub}) &= \underbrace{W(f1)}_{A} + \underbrace{W(f2)}_{B} + \underbrace{W(f3)}_{C} \\ &\quad + \underbrace{W(cond_{sub})}_{D} + \underbrace{W(loop_{sub})}_{E} \end{aligned} \tag{16}$$

$$\begin{aligned} W(testProgram_{self}) &= \underbrace{W(f1_{ret})}_{A} + \underbrace{W(f2_{ret})}_{B} \\ &\quad + \underbrace{W(f3_{ret})}_{C} + \underbrace{W(cond_{self})}_{D} \\ &\quad + \underbrace{W(loop_{self})}_{E} \end{aligned} \tag{17}$$

$$W(cond_{sub}) = max(\underbrace{W(f4)}_{A}, \underbrace{W(f5)}_{B}) \tag{18}$$

$$W(cond_{self}) = max(\underbrace{W(f4_{ret})}_{A}, \underbrace{W(f5_{ret})}_{B}) \tag{19}$$

$$W(loop_{sub}) = \underbrace{\overbrace{W(f6)}^{A} + \ldots + W(f6)}_{N} \tag{20}$$

$$W(loop_{self}) = \underbrace{\overbrace{W(f6_{ret})}^{A} + \ldots + W(f6_{ret})}_{N} \tag{21}$$

It is worth noting that $W(cond_{sub})$, $W(cond_{self})$, $W(loop_{sub})$ and $W(loop_{self})$ expressions are same as the RapiTime approach [22]. However, while evaluating Eqs. (16) and (17), directly replacing the expressions for conditional and iterative blocks with their corresponding equations given in (18)–(21) would result in the same approach given in [22], which is the comonotonic assumption between segments.

In our approach, however, instead of directly convolving all the expressions, a copula is fitted for the sub-scopes of $testProgram_{sub}$ and $testProgram_{self}$. After the simulation phase, the total execution time distribution is calculated by using the derived CDFs for the sub-scopes. Derivation of the CDFs for the sub-scopes are done in the same way as described in the previous section for sequential blocks (A, B and C). However, for iterative and conditional blocks a different mechanism has to be employed.

In order to derive the copula for the sub-scopes of $testProgram_{sub}$ and $testProgram_{self}$, measurements of these scopes are needed. A, B and C in Eq. (16) are constructed using end-to-end measurements of $f1$, $f2$ and $f3$. For conditional and iterative blocks, their corresponding random variables are expressed as:

$$D_i = (Obs_i(f4) \text{ or } Obs_i(f5)) \tag{22}$$

$$E_i = \sum_{j=1}^{N_i} Obs_j(f6),\ N_i = \text{current iteration count} \tag{23}$$

where: (24)

i = 1,…,10000

$Obs_i(f) = i$th observed sample of function f

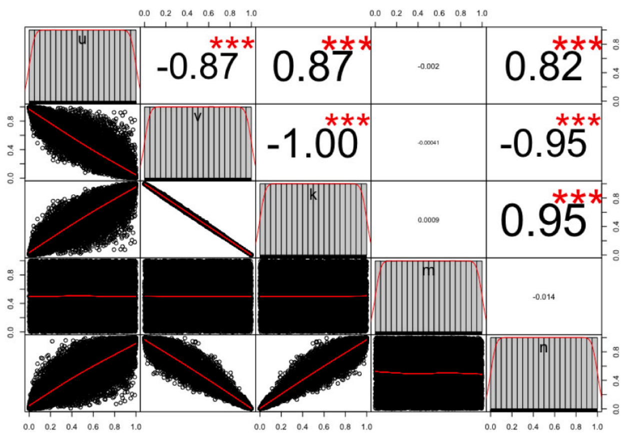
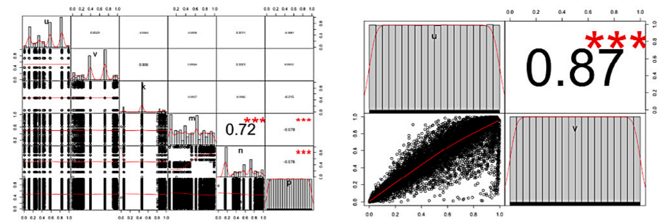


Fig. 12. Correlation plot of $testProgram_{sub}$.



(a) Correlation of $testProgram_{self}$ (b) Correlation of $testProgram$

Fig. 13. Correlation plots of the program scopes.

After constructing A, B, C, D and E for $testProgram_{sub}$, correlation between the pseudo-observations (u, v, k, m, n, respectively) of the sub-scopes are shown in Fig. 12.

The same procedures are applied to $testProgram_{self}$ and $testProgram$ and the correlation between their scopes are shown in Fig. 13.

A Vine Copula model is fitted to each $testProgram$, $testProgram_{self}$ and $testProgram_{sub}$ scope by using `RVineStructureSelect` function of R. Then goodness-of-fit test is applied to each copula with `RVineGofTest` function and all the fitted copulas turned out to be valid. For simplicity of this section, these results are omitted.

The challenging part is the derivation of a CDF for each sub-scope of the scopes. For sequential sub-scopes which correspond to A, B, and C inside $textProgram_{sub}$, same procedure can be applied as in previous section.

However, for conditional blocks, each part of the condition should be considered as a separate scope and our hybrid approach should be applied since these are separate programs. The result would be a distribution for each condition block and by using their inverses, Eq. (18) or (19) can be calculated. In this case study, condition blocks only consist of one functional block so it is valid to model them using SPD if they conform the requirements.

For iterative blocks, their CDF is calculated by the approach detailed in [1], which is based on checking whether there is independency across iterations. If so, then standard convolution is applied. Otherwise, biased convolution is applied for Eqs. (20) and (21). Instead of modeling each random variable inside the loop with their empirical CDFs (ETP), our method primarily aims to model them with SPD if possible. All functions inside our $testProgram$ is observed to conform the i.i.d requirements.

Fig. 14 shows that all the functions except $f6$ are suitable to be represented by an SPD model. According to our proposed method, if any random variable cannot be represented by an Extreme Value distribution, its empirical distribution should be used. Therefore, empirical CDF is used for $f6$ case only.

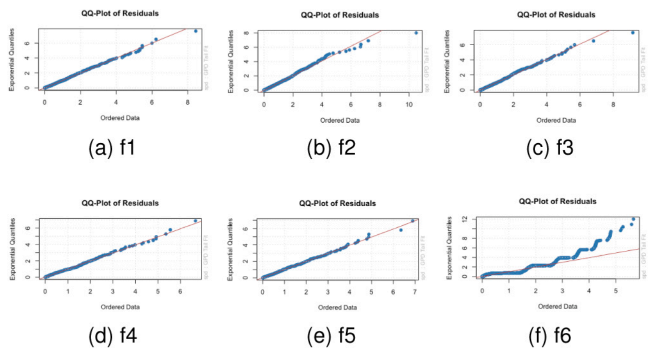


Fig. 14. QQ-Plots of piecewise GPD fit for $testProgram_{sub}$.

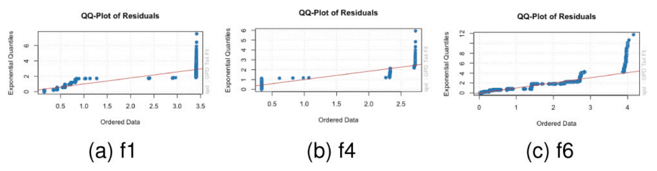


Fig. 15. QQ-Plots of piecewise GPD fit for $testProgram_{self}$.

Sub-scopes of the $testProgram_{self}$ are not suitable to be modeled by an EV distribution as shown in Fig. 15. QQ results for $f2$, $f3$ and $f5$ are missing because the `spdfit` function of R could not even estimate suitable parameters for them. Therefore, empirical CDFs are used to represent the random variables within $testProgram_{self}$.

Finally, the equations to derive the total execution time distribution for this case study becomes:

$$W_i(cond_{sub}) = max(F_A^{(-1)}(m_i), F_B^{(-1)}(m_i)) \tag{25}$$

where:

- i: $1, \dots, 1.000.000$
- $W_i(cond_{sub})$: is the execution time distribution of $cond_{sub}$
- $F_A^{(-1)}$: is the inverse CDF of fitted SPD model of A in Eq. (18)
- $F_B^{(-1)}$: is the inverse CDF of fitted SPD model of B in Eq. (18)
- m_i: are the simulated uniforms from the copula for D in Eq. (16)

$$F_{loop_{sub}} = \underbrace{F_A \circledast \cdots \circledast F_A}_{N} = F_A^{\circledast N} \tag{26}$$

$$W_i(loop_{sub}) = F_{loop_{sub}}^{(-1)}(n_i) \tag{27}$$

where:

- i: $1, \dots, 1.000.000$
- $F_{loop_{sub}}$: is the CDF of E in Eq. (16)
- F_A: is the ECDF of A in Eq. (20)
- N: is the maximum observed iteration count
- $\circledast$: is the convolution operation (standard or biased) for CDFs
- $W_i(loop_{sub})$: is the execution time distribution of $loop_{sub}$
- n_i: are the simulated uniforms from the copula for E in Eq. (16)

$$W_i(testProgram_{sub}) = F_A^{(-1)}(u_i) + F_B^{(-1)}(v_i) + F_C^{(-1)}(k_i) + W_i(cond_{sub}) + W_i(loop_{sub}) \tag{28}$$

where:

- i: $1, \dots, 1.000.000$
- $W_i(testProgram_{sub})$: is the execution time distribution of $testProgram_{sub}$

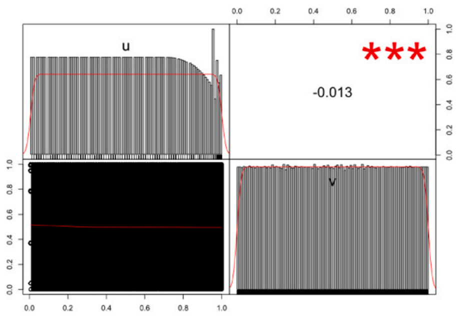


Fig. 16. Correlation across the iterations.

Table 2
Calculated pWCET values for case study.

pWCET	EVT-Cop	Independent	RapiTime	EE
10^{-2}	57.2 ms	60.2 ms	79.1 ms	54.8 ms
10^{-4}	58.1 ms	63.7 ms	95.8 ms	56.3 ms
10^{-6}	58.6 ms	67.0 ms	113.6 ms	56.3 ms
10^{-9}	58.7 ms	68.3 ms	113.6 ms	56.3 ms

- $F_A^{(-1)}$: is the inverse CDF of fitted SPD model of A in Eq. (16)
- $F_B^{(-1)}$: is the inverse CDF of fitted SPD model of B in Eq. (16)
- $F_C^{(-1)}$: is the inverse CDF of fitted SPD model of C in Eq. (16)
- $W_i(cond_{sub})$: is the execution time distribution of $cond_{sub}$ given in Eq. (25)
- $W_i(loop_{sub})$: is the execution time distribution of $loop_{sub}$ given in Eq. (27)
- u_i: are the simulated uniforms from the copula for A in Eq. (16)
- v_i: are the simulated uniforms from the copula for B in Eq. (16)
- k_i: are the simulated uniforms from the copula for C in Eq. (16)

Similar procedure is applied for $testProgram_{self}$ in Eq. (17) and its total CDF is derived. The only difference is that ECDFs are used to represent the segments because of the unsuitability of SPD for modeling them as shown in Fig. 15.

It is worth noting that the convolution operation used for loop case is the standard convolution. Fig. 16 shows correlation between the iteration index parameter and the execution time of $f6$.

u and v represent pseudo-observations of loop iteration index parameter and pseudo-observations of execution time of $f6$ respectively. No correlation between iteration index and execution time of $f6$ is observed. It means that the execution time of $f6$ does not increase when loop iteration count increases or vice-versa. In order to calculate Eq. (20) standard convolution of random variable A is sufficient. This calculation is basically sum of N random variables, which is the main problem in our work. Therefore, the same copula approach could be applied, but in order to do that 1.000.000 x N simulated uniforms should have been generated with a fitted copula. Then by using inverse CDF of $f6$, Eq. (20) could have been calculated as $testProgram$ itself. However, this is computationally intractable, therefore an adequate independence test is sufficient to decide whether to use standard convolution or biased convolution for loop blocks. Finally, solving the same equation shown in Eq. (15) yields Fig. 17.

Our main result is illustrated in Fig. 17 which is the tightest among all methods employed. Although the standard convolution is used to calculate the execution time distribution of the loop block, it is still below the independent method. The comonotonic assumption of commercial RapiTime tool is observed to overestimate the result by a huge factor.

Following table summarizes some pWCET values that are obtained from the estimated distributions.

Results in Table 2 show that the RapiTime's approach overestimates the observed end-to-end execution time more than 100% for

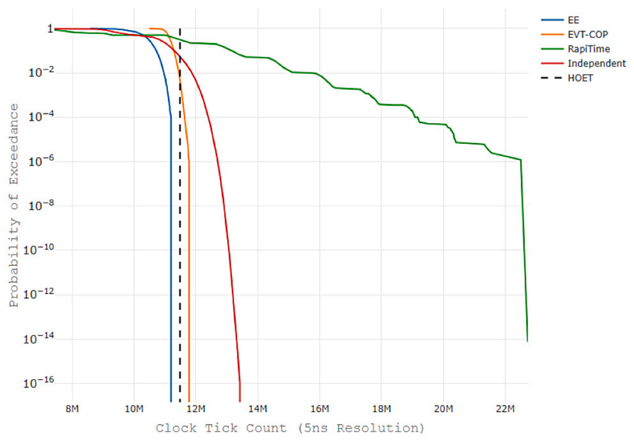


Fig. 17. Estimated pWCET distributions for case study.

pWCET(10^{-9}) case. Higher level design decisions are usually made based on this value, which might push the limits of some real-time embedded software environments having limited resources to be scheduled. In cases where overestimation leads to poor utilization, our results might prove useful since WCET is now tightly bounded up to some confidence level based on the tail modeling methodology (EVT) and the proposed dependency modeling approach (Copulas).

3.5.1. Tightness assessment

In order to provide evidence for the tightness of the proposed EVT-Copula method, 10^6 additional observations are collected from the same setup. The highest observed execution time ($HOET$) out of the 10^6 measurements for the Case Study was 57.44 ms. It shows that our EVT-Copula method is both tight and reliable compared to the Independent and Comonotonic assumptions.

Reliability is another important issue for EVT applications, however a universal consensus do not exist on this topic [6]. Results of all statistical approaches are as reliable as the applied statistical tests and can be trusted only up to some confidence level.

Fig. 17 shows $HOET$ of 10^6 observations in black dashed line. Our EVT-Copula method seems to be both safe and tight for pWCET(10^{-6}) in comparison to other approaches.

4. Conclusion

This paper proposes an enhanced HYPTA framework for time-critical applications running on COTS hardware platforms.

A literature review revealed that there is no known study to this day, which employed a hybrid approach using Copulas with EVT for timing analysis of real-time applications running on COTS platforms.

The main principle in our proposed framework is to model the dependency between the random variables using Copulas and model each random variable with a proper Extreme Value distribution whenever possible. Instead of representing the whole program as one syntax tree, a specialized version similar to the scope-tree approach is followed. In order to fit a copula for each scope of the program under test, this representation method was necessary. Basically, the program is divided into scopes and each scope is analyzed in isolation by using our proposed methodology.

The majority of studies in MBPTA domain utilize a time randomized platform (for example by altering cache behavior) to eliminate some of the prerequisites of EVT. However, in COTS platforms this is not possible at all, which leaves us with using randomized inputs for the programs under test. Similar to other measurement based methods, coverage of all hardware effects are not guaranteed with our approach and it is not possible to analyze and ensure the coverage without an analysis of the compiled object code.

The effects of the underlying hardware units such as caches, pipelines, out-of-order executions, etc. can only be analyzed by employing Static Timing Analysis techniques, which require to model every aspect of the underlying platform precisely. Lack of knowledge about the details of the platform and unpredictability resulting from modern day architectural properties forces static methods to manage the available complexity only by making very conservative/pessimistic assumptions for the basic/functional block, which in return leads to overestimation of the WCET.

Results of our experiments show that the proposed EVT with Copula method can provide much tighter results in comparison to the commercial tool named RapiTime. RapiTime employs either co-monotonic or standard convolution between blocks. In the present work instead of aiming a fully automated solution such as RapiTime, we preferred to search for a meaningful enhancement possibility over existing approaches to be applicable to COTS platforms and have succeeded in pointing out such an opportunity.

To the best of our knowledge, there is no known methodology or a commercial solution that autonomously resolves all issues and analyzes a program or a piece of code without a user intervention. Even Static Timing Analysis tools require manual annotation of loop bounds or recursion limits in source code, which decreases the usability and autonomy of timing analysis without some empirical and also pessimistic assumptions made by the user. Measurement-based approaches do not require much intervention or annotation hence making it more applicable in industry.

Our proposal may be employed as an extension to RapiTime tool, which is broadly used in industry. We introduce modeling of functional blocks with EVT whenever possible. This part hugely depends on the quality of inputs given to the analyzed program. Generation of qualified and high coverage inputs is not the main focus hence we used a classical random generation approach. If those inputs are sufficient to somehow generate enough variability in execution times of the functional blocks then they are modeled with parametric probability distributions. In the other case, we fall back to using the same approach that RapiTime applies, which is to use frequentist ETPs for those specific (non-conforming EVT prerequisites) functional blocks. Thus, we claim that our approach is suitable for industrial use as much as RapiTime.

This study aims to propose a hybrid approach to enhance current state-of-the-art solutions. The proposed method has the following limitations:

(1) Structured and relatively simple programs are currently supported.
(2) Injection of instrumentation points (IPoints) is not automated yet.
(3) Scalable by the number of IPoints that can be inserted in a program.

Future work

Despite limitations, there are opportunities for further research. The present work aims to provide a hybrid probabilistic measurement based timing analysis mechanism for COTS platforms, hence the only applicable source of execution time variability is random input generation. This randomization process can be made more intelligent by using model checking or genetic algorithms.

For the application of EVT, our procedures are based on visual checking from the plots in order to conclude that the models are fit to the data or the data conforms to applicability requirements. There exists recent studies, which calculates the applicability of EVT numerically and selects the best distribution to represent the observed data such as MBPTA-CV [2]. Furthermore, an open-source study named as chronovise [39] that implements MBPTA-CV method in C++ is available. By adapting more automatic mechanisms like chronovise, the necessity of user interaction within the proposed framework could be decreased.

This work presents a framework, which is implemented in MATLAB and R. The experiments and case studies are analyzed mostly by manual modifications or re-writing of analysis code in R or MATLAB. In order to transform the proposal into a full function automatic tool, the following actions might be carried out:

(1) An automatic C/C++ source code parser is needed in order to extract the structural information of the analyzed code and instrument the necessary points automatically.
(2) A trace parser is needed to extract the execution traces of each individual instrumentation point and associate them with the corresponding blocks inside the program.
(3) Representation of the analyzed program with our proposed scope tree approach should be automated.
(4) A fully automatic mechanism to derive the copulas and estimate the extreme value distributions by testing the applicability is needed.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

The authors would like to thank Turkish Aerospace Industries, Inc. (TAI), for their support in the starting phases of the present work.

References

[1] G. Bernat, A. Colin, S. Petters, WCET analysis of probabilistic hard real-time systems, in: 23rd IEEE Real-Time Systems Symposium, 2002. RTSS 2002, IEEE Comput. Soc, 2002, pp. 279–288, [Online]. Available: http://ieeexplore.ieee.org/document/1181582/.

[2] J. Abella, M. Padilla, J.D. Castillo, F.J. Cazorla, Measurement-based worst-case execution time estimation using the coefficient of variation, ACM Trans. Des. Autom. Electron. Syst. 22 (4) (2017) 1–29, [Online]. Available: http://dl.acm.org/citation.cfm?doid=3097980.3065924.

[3] K.P. Silva, L.F. Arcaro, R.S. de Oliveira, On using GEV or gumbel models when applying EVT for probabilistic WCET estimation, in: 2017 IEEE Real-Time Systems Symposium (RTSS), Vol. 2018-Janua, IEEE, 2017, pp. 220–230, [Online]. Available: http://ieeexplore.ieee.org/document/8277295/.

[4] L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzetti, E. Quiñones, F.J. Cazorla, Measurement-based probabilistic timing analysis for multi-path programs, in: Proceedings - Euromicro Conference on Real-Time Systems, IEEE, 2012, pp. 91–101, [Online]. Available: http://ieeexplore.ieee.org/document/6257562/.

[5] B.Y.A. Ferreira, L. De Haan, On the block maxima method in extreme value theory: PWM estimators, Ann. Statist. 43 (1) (2015) 276–298.

[6] F.J. Cazorla, L. Kosmidis, E. Mezzetti, C. Hernandez, J. Abella, T. Vardanega, Probabilistic worst-case timing analysis, ACM Comput. Surv. 52 (1) (2019) 1–35, [Online]. Available: http://dl.acm.org/citation.cfm?doid=3309872.3301283.

[7] F. Guet, L. Santinelli, J. Morio, On the reliability of the probabilistic worst-case execution time estimates, in: European Congress on Embedded Real Time Software and Systems 2016 (ERTS'16), 2016, p. 10, [Online]. Available: https://hal.archives-ouvertes.fr/hal-01289477.

[8] L. Santinelli, F. Guet, J. Morio, Revising measurement-based probabilistic timing analysis, in: 2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), IEEE, 2017, pp. 199–208, [Online]. Available: http://ieeexplore.ieee.org/document/7939040/.

[9] G. Lima, D. Dias, E. Barros, Extreme value theory for estimating task execution time bounds: A careful look, in: 2016 28th Euromicro Conference on Real-Time Systems (ECRTS), Vol. 2016-Augus, IEEE, 2016, pp. 200–211, [Online]. Available: http://ieeexplore.ieee.org/document/7557881/.

[10] A. Sklar, Fonctions de Répartition à n Dimensions et Leurs Marges, Vol. 8, Publications de L'Institut de Statistique de L'Université de Paris, 1959, pp. 229–231.

[11] R.I. Davis, L. Cucu-Grosjean, A survey of probabilistic timing analysis techniques for real-time systems, Leibniz Trans. Embedded Syst. (LITES) 6 (1) (2019) 3:1–3:60, [Online]. Available: https://ojs.dagstuhl.de/index.php/lites/article/view/LITES-v006-i001-a003/lites-v006-i001-a003-pdf.

[12] A. Burns, S. Edgar, Predicting computation time for advanced processor architectures, in: Proceedings 12th Euromicro Conference on Real-Time Systems. Euromicro RTS 2000, IEEE Comput. Soc, 2000, pp. 89–96, [Online]. Available: http://ieeexplore.ieee.org/document/853996/.

[13] S. Coles, An Introduction to Statistical Modeling of Extreme Values, in: Springer Series in Statistics, Springer London, London, 2001, [Online]. Available: http://link.springer.com/10.1007/978-1-4471-3675-0.

[14] J. Hansen, S. Hissam, G. Moreno, Statistical-based WCET estimation and validation, Wcet (January) (2009) 123–133.

[15] L. Santinelli, J. Morio, G. Dufour, D. Jacquemart, On the sustainability of the extreme value theory for WCET estimation, in: The 14th International Workshop on Worst-Case Execution Time (WCET) Analysis, no. Wcet, 2014, pp. 21–30.

[16] F.J. Cazorla, T. Vardanega, E. Quiñones, J. Abella, Upper-bounding program execution time with extreme value theory, in: International Workshop on Worst-Case Execution Time Analysis 2013 (WCET'13), Vol. 30, 2013, pp. 64–76, no. Wcet. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2013/4123.

[17] 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, W. Talaboulma, PROXIMA: Improving measurement-based timing analysis through randomisation and probabilistic analysis, in: 2016 Euromicro Conference on Digital System Design (DSD), IEEE, 2016, pp. 276–285, [Online]. Available: http://ieeexplore.ieee.org/document/7723564/.

[18] K.P. Silva, L.F. Arcaro, D.B. de Oliveira, R.S. de Oliveira, An empirical study on the adequacy of MBPTA for tasks executed on a complex computer architecture with linux, in: 2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA), IEEE, 2018, pp. 321–328, [Online]. Available: https://ieeexplore.ieee.org/document/8502513/.

[19] R.I. Davis, I. Bate, I. Broster, A. Burns, S. Hutchesson, R.-r. Plc, Transferring real-time systems research into industrial practice: Four impact case studies, 106 (7) (2018) 7:1–7:24. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2018/8995.

[20] A. Colin, G. Bernat, Scope-tree: A program representation for symbolic worst-case execution time analysis, in: Proceedings - Euromicro Conference on Real-Time Systems, IEEE, 2002, pp. 50–59.

[21] G. Bernat, A. Colin, S. Petters, pWCET: A Tool for Probabilistic Worst-Case Execution Time Analysis of Real-Time Systems, Report-University of York . . . , 2003, pp. 1–18, [Online]. Available: ftp://www.cs.york.ac.uk/reports/2003/YCS/353/YCS-2003-353.pdf.

[22] G. Bernat, A. Burns, M. Newby, Probabilistic timing analysis: An approach using copulas, J. Embedded Comput. 1 (2) (2005) 179, [Online]. Available: http://portal.acm.org/citation.cfm?id=1233763.

[23] G. Bernat, R. Davis, N. Merriam, J. Tuffen, A. Gardner, M. Bennett, D. Armstrong, Identifying opportunities for worst-case execution time reduction in an avionics system, Ada User J. 28 (3) (2007) 189–194.

[24] R.S. Ltd., Worst-Case Execution Time Analysis, RapiTime White Paper, 2008, [Online]. Available: https://www.rapitasystems.com/.

[25] L. Kosmidis, J. Abella, F. Wartel, E. Quinones, A. Colin, F.J. Cazorla, PUB: Path upper-bounding for measurement-based probabilistic timing analysis, in: 2014 26th Euromicro Conference on Real-Time Systems, IEEE, 2014, pp. 276–287, [Online]. Available: http://ieeexplore.ieee.org/document/6932609/.

[26] J. Abella, C. Hernandez, E. Quinones, F.J. Cazorla, P.R. Conmy, M. Azkarate-askasua, J. Perez, E. Mezzetti, T. Vardanega, WCET analysis methods: Pitfalls and challenges on their trustworthiness, in: 10th IEEE International Symposium on Industrial Embedded Systems (SIES), IEEE, 2015, pp. 1–10.

[27] M. Ziccardi, E. Mezzetti, T. Vardanega, J. Abella, F.J. Cazorla, EPC: Extended path coverage for measurement-based probabilistic timing analysis, in: 2015 IEEE Real-Time Systems Symposium, Vol. 2016-Janua, IEEE, 2015, pp. 338–349, [Online]. Available: http://ieeexplore.ieee.org/document/7383590/.

[28] C. Park, A. Shaw, Experiments with a program timing tool based on source-level timing schema, in: [1990] Proceedings 11th Real-Time Systems Symposium, 1990, pp. 72–81, http://dx.doi.org/10.1109/REAL.1990.128731.

[29] K. Avdulaj, Value-at-Risk based on Extreme Value Theory method and copulas. Empirical evidence from Central Europe (Master's thesis), Univerzita Karlova, 2010.

[30] F. Miller, A. Vandome, M. John, Inverse Transform Sampling, VDM Publishing, 2010, [Online]. Available: https://books.google.com.tr/books?id=OFtdXwAACAAJ.

[31] J. Dißmann, E.C. Brechmann, C. Czado, D. Kurowicka, Selecting and estimating regular vine copulae and application to financial returns, Comput. Statist. Data Anal. 59 (1) (2013) 52–69.

[32] A. Ermedahl, F. Stappert, J. Engblom, Clustered worst-case execution-time calculation, IEEE Trans. Comput. 54 (9) (2005) 1104–1122, [Online]. Available: https://doi.org/10.1109/TC.2005.139.

[33] A. Betts, Hybrid Measurement-Based WCET Analysis using Instrumentation Point Graphs (Ph.D. dissertation), University of York, 2010.

[34] L. Bekdemir, Hybrid Probabilistic Timing Analysis with Extreme Value Theory and Copulas (Master's thesis), Middle East Technical University, 2019.

[35] R Core Team, R: A Language and Environment for Statistical Computing, R Foundation for Statistical Computing, Vienna, Austria, 2019, [Online]. Available: https://www.R-project.org/.

[36] J. Gustafsson, A. Betts, A. Ermedahl, B. Lisper, The mälardalen WCET benchmarks: Past, present and future, in: International Workshop on Worst-Case Execution Time Analysis (WCET 2010), 2010.

[37] U. Schepsmeier, A goodness-of-fit test for regular vine copula models, Econometric Rev. (2013) 1–22, [Online]. Available: http://arxiv.org/abs/1306.0818.

[38] M. Matsumoto, T. Nishimura, Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator, ACM Trans. Model. Comput. Simul. 8 (1) (1998) 3–30, [Online]. Available: http://portal.acm.org/citation.cfm?doid=272991.272995.

[39] F. Reghenzani, G. Massari, W. Fornaciari, Chronovise: Measurement-based probabilistic timing analysis framework, J. Open Source Softw. 3 (28) (2018) 711, [Online]. Available: http://joss.theoj.org/papers/10.21105/joss.00711.

Levent Bekdemir received his B.S. degree in electrical and electronics engineering from Eskişehir Osmangazi University, Eskişehir, Turkey, in 2014, and the M.S. degree in electrical and electronics engineering from Middle East Technical University, Ankara, Turkey, in 2019.

He worked as an embedded software design engineer in Turkish Aerospace Industries earlier and is currently an avionics software design engineer in Aselsan. His research interests include embedded systems, computer systems architecture and hardware, and software engineering.

Cüneyt F. Bazlamaççı received his B.Sc. and M.Sc. degrees both in Electrical and Electronics Engineering from the Middle East Technical University, Ankara, Turkey, in 1988 and 1991, respectively. He received his Ph.D. degree in Computing from the University of Manchester Institute of Science and Technology, Manchester, UK, in 1996. He has served both in industry and academia for many years and is currently a professor in Computer Engineering Department, Izmir Institute of Technology, Turkey.

His major research interests include computer systems architecture — hardware and software, communication and computer networks, graph algorithms and embedded systems, in general.

Dr. Bazlamaççı is a senior member of IEEE and a member of ACM.