PUCRS

ESCOLA POLITÉCNICA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA
MESTRADO EM ENGENHARIA ELÉTRICA

GUILHERME ISAIAS DEBOM MACHADO

ANALYSIS OF THE EXTREME VALUE THEORY ON THE ESTIMATION OF PROBABILISTIC WCET

Porto Alegre
2021

PÓS-GRADUAÇÃO - *STRICTO SENSU*

Pontifícia Universidade Católica
do Rio Grande do Sul

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
ESCOLA POLITÉCNICA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

GUILHERME ISAIAS DEBOM MACHADO

ANALYSIS OF THE EXTREME VALUE THEORY ON THE ESTIMATION OF PROBABILISTIC WCET

Porto Alegre
2021

GUILHERME ISAIAS DEBOM MACHADO

ANALYSIS OF THE EXTREME VALUE THEORY ON THE ESTIMATION OF PROBABILISTIC WCET

Dissertação apresentada como requisito para a obtenção do grau de Mestre pelo Programa de Pós-Graduação em Engenharia Elétrica da Escola Politécnica da Pontifícia Universidade Católica do Rio Grande do Sul.

Área de concentração: Sinais, Sistemas e Tecnologia da Informação.

Linha de pesquisa: Sistemas de Computação, Controle e Automação.

Orientador: Prof. Dr. Fabian Luis Vargas

Porto Alegre

2021

Pontifícia Universidade Católica do Rio Grande do Sul
ESCOLA POLITÉCNICA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA - PPGEE

ANALYSIS OF THE EXTREME VALUE THEORY ON THE ESTIMATION OF PROBABILISTIC WCET

CANDIDATO: GUILHERME ISAIAS DEBOM MACHADO

Esta Dissertação de Mestrado foi julgada para obtenção do título de MESTRE EM ENGENHARIA ELÉTRICA e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia Elétrica da Pontifícia Universidade Católica do Rio Grande do Sul.

DR. FABIAN LUIS VARGAS - ORIENTADOR

BANCA EXAMINADORA

DR. JARBAS ARYEL NUNES DA SILVEIRA - PPGETI - UFC

DR. CÉSAR AUGUSTO MISSIO MARCON - PPGCC - PUCRS

PUCRS
Av. Ipiranga, 6681 - Predio 32 - Sala 507 | CEP 90619-900 | Porto Alegre, RS - Brasil
Fone: (51) 3320-3540 | E-mail: engenharia.pg.eletrica@pucrs.br | www.pucrs.br/politecnica

AGRADECIMENTOS

Diante da conclusão deste grande desafio agradeço ao soberano Deus, por me preservar a vida e me conceder o privilégio de realizar este trabalho, por sua fidelidade e pela graça de ter encontrado muitas pessoas especiais.

Agradeço especialmente aos meus familiares, meu pai Teodoro, minha mãe Nilda, meu irmão Jonatas e minha cunhada Amanda, por serem incentivadores nesse grande desafio.

Ao professor e orientador Dr. Fabian Luis Vargas por me receber em seu grupo de pesquisa, por viabilizar a realização deste trabalho.

Ao professor Celso Maciel da Costa pelo privilégio de tê-lo como mestre desde o meu ingresso na graduação.

A todos os professores e colaboradores da PUCRS, meu muito obrigado por toda a ajuda e colaboração.

RESUMO

Quando sistemas de tempo real são desenvolvidos para aplicações críticas, o tempo de execução é um requisito tão importante quanto o resultado computado. Por este motivo, o tempo máximo de execução de um sistema de tempo real deve obrigatoriamente ser determinado durante a fase de projeto. Estimar o tempo de execução de sistemas complexos impacta diretamente no tempo e nos custos da análise durante o desenvolvimento do sistema. Neste contexto, esta dissertação tem por objectivo avaliar a possibilidade do método MBPTA (do inglês: Measured-Based Probabilistic Timing Analysis) se basear na Teoria de Valores Extremos (EVT - Extreme Value Theory) para estimar o tempo de execução do pior caso probabilístico (pWCET - Probabilistic Worst-Case Execution Time) de uma aplicação em uma plataforma de hardware simulado.

Para tanto, utilizou-se o processador MIPS rodando dois algoritmos como estudo-de-caso: Bubble Sort e Filtro FIR. Estes algoritmos têm o WCET estimado através do método de análise de tempo determinístico estático (Static Deterministic Timing Analysis – SDTA). Neste trabalho, o MBPTA é estimado através de duas técnicas distintas: Block Maxima (BM) e Peak Over Threshold (POT), as quais são combinadas com EVT para a estimação final do WCET. Os valores de WCET obtidos por MBPTA/BM e MBPTA/POT com EVT são comparados contra os valores de referência, obtidos através do método SDTA.

Os resultados obtidos sugerem que a técnica BM fornece resultados confiáveis mais facilmente do que POT. Embora POT seja mais complexa, as análises sugerem que esta técnica possui mais precisão que BM, especialmente quando não são repetidos os valores de entrada da aplicação analisada.

Palavras-chave: Worst-Case Execution Time (WCET); WCET Estimation; Measured-Based Probabilistic Timing Analysis (MBPTA); Extreme Value Theory (EVT); Block Maxima (BM); Peak Over Threshold (POT); Aplicação de tempo real;

ABSTRACT

Real Time systems developed for critical applications require a proper execution time as important as the correct computed outcome. Owing to this, the maximum execution time of a Real Time System shall be determined by design. Estimate the execution time of complex systems affects time and analysis cost directly during the system development. In this sense, this dissertation aims to assess the possibility of applying MBPTA (Measured-Based Probabilistic Timing Analysis) based on EVT (Extreme Value Theory) to estimate the pWCET (Probabilistic Worst-Case Execution Time) of a given application in a given hardware platform.

With this purpose, this dissertation makes use of the MIPS processor executing two algorithms as case studies: Bubble Sort and FIR Filter. These algorithms have WCET estimated by SDTA (Static Deterministic Timing Analysis). This work applies MBPTA with two different approaches: Block Maxima (BM) and Peak Over Threshold (POT), which are combined with EVT to estimate the final WCET. Then it compares the obtained WCET values by MBPTA/BM and MBPTA/POT with EVT to reference values, obtained by the SDTA method.

The obtained results suggest that the BM approach presents a reduced complexity implementation as compared to the POT approach. Nevertheless, besides the POT higher complexity, this approach is more accurate than the BM, especially when input data values are not repeated.

Keywords: Worst-Case Execution Time (WCET); WCET Estimation; Measured-Based Probabilistic Timing Analysis (MBPTA); Extreme Value Theory (EVT); Block Maxima (BM); Peak Over Threshold (POT); Real-time application;

List of Figures

List of Tables

List of Abbreviations and Acronyms

ACET	Average-Case Execution Time
AH	Always-Hit
AM	Always-Miss
BCET	Best-Case Execution Time
BM	Block Maxima
CDF	Cumulative Distribution Function
CFG	Control Flow Graph
CHMC	Cache Hit/Miss Classification
CRPS	Continuous Rank Probability Score
DM	Direct-Mapped
DRAM	Dynamic Random Access Memory
ECDF	Empirical Cumulative Distribution Function
EPC	Extended Path Coverage
ETP	Execution Time Profile
EVT	Extreme Value Theory
FA	Fully-Associative
FIR	Finite Impulse Response
FM	First-Miss
GEV	Generalized Extreme Value
GPD	Generalized Pareto Distribution
i.i.d.	independent and identically distributed
IDE	Integrated Development Environment
ILP	Integer Linear Programming
IPET	Implicit Path Enumeration Technique
MARS	MIPS Assembler and Runtime Simulator
MBPTA	Measured-Based Probabilistic Timing Analysis
MBTA	Measured Based Timing Analysis
MIPS	Microprocessor without Interlocked Pipelined Stages
OS	Operating System

PDF	Probabilistic Distribution Function
pET	probabilistic Execution Time
POT	Peak Over Threshold
PUB	Path Upper-Bounding
PUBaa	Path Upper-Bounding Address Aging
PUBam	Path Upper-Bounding Address Merging
pWCET	probabilistic Worst-Case Execution Time
QQ-plot	quantile-quantile graph
r.v.	random variable
RTOS	Real Time Operating System
SA	Set-Associative
SDTA	Static Deterministic Timing Analysis
SPTA	Static Probabilistic Timing Analysis
WCET	Worst-Case Execution Time

Contents

1 Introduction

Developing computational systems requires predictability about the environment and system operation. For several applications, it is an imperative requirement to ensure execution accuracy and reliability. The occurrence of unpredictable delays in real-time systems used for aerospace, defense and automotive applications, for instance, might jeopardize the equipment integrity as well as human safety.

Worst-Case Execution Time (WCET) is an important parameter to ensure accuracy under critical conditions. Knowing the WCET since the early stages of the design allows the system to act in acceptable time for all execution possibilities. There are several techniques to find a system WCET and those techniques require software and hardware to be known and likewise predictable. Software characteristics as iterations and conditional jumps impact execution time straightforwardly. Moreover, hardware characteristics as cache memory and shared resources like buses also affect WCET (CUCU-GROSJEAN et al., 2012).

System complexity might jeopardize the process of finding the WCET. Predicting all execution cases in intricate systems increases the time and overall design cost. Several hardware components, such as cache memory, have random behavior, which varies the execution time for each execution case.

Alternatively, it is possible to find the execution time probability for different executions cases and set a probability distribution of the system. It is possible to estimate the probabilistic Worst-Case Execution Time (pWCET) using that distribution and defining the worst scenario for some low enough probability.

GIL, Samuel Jimenez et al. (2017) show the open challenges regarding the pWCET definition. Having a more critical view on the state of the art of the current literature, this paper points out theoretical problems about Extreme Value Theory (EVT) application vulnerability, such as sample set representativeness and the software and hardware characteristics correlation with some minimal sample amount for correct analysis.

LIMA, George; BATE, Iain (2017) propose the Indirect Estimation in Statistical Time Analysis (IESTA) technique, alternatively to hardware randomization to compensate systems intrinsic uncertainties. This approach aims at trying to solve system uncertainty problems, nevertheless it disregards the pWCET definition technique uncertainties.

When applying MBPTA with EVT to estimate pWCET, the inquiry is auspicious since there is a considerable lack of information about the efficiency and applicability of this estimation method for real time systems. Owing to this, this present-day work proposes a methodology to measure Extreme Value Theory efficiency to estimate pWCET for critical real-time systems. The objective is to discover and explore EVT limitations, also correlating EVT and system's characteristics with the analysis of the computed results uncertainty.

2 Objectives

This work aims to propose a methodology to analyze the quality of pWCET estimates of hard real time systems, obtained according to EVT, and point out the process limitations this methodology might present. In order to do that, it is possible to define some specific goals as follows:

- (1) Select algorithms for different applications, as Vector Ordering, Matrix Multiplication, Finite Impulse Response filter (FIR) and Image Processing algorithms, for instance;
- (2) Assembly those algorithms for execution in MIPS architecture processor;
- (3) Statically analyze each application assembly code to determine the Worst-Case Execution Time (WCET). This value is defined as the "reference value";
- (4) Apply a Measured-Based Probabilistic Timing Analysis (MBPTA) to those application codes. This process is to be realized under the use of different parameter values. It is possible to specify this process in activities as follows:
  - (4.1) Simulate every single code to obtain sets of execution time observations for each code.
  - (4.2) Reach the minimum acceptable observations number (as described later in 4.1.5.)
  - (4.3) Apply the Extreme Value Theory (EVT) in the sample sets according to the Block Maxima (BM) approach.
  - (4.4) Apply the EVT in the sample sets according to the Peak Over Threshold (POT) approach.
- (5) Investigate how the input data affect outcome data and how these input values influence the outcome accuracy and reliability of the estimated WCET for both approaches (BM and POT).
- (6) Analyze results from different scenarios considering the same algorithm but different amounts of data under analysis in order to verify whether and how the input data affects outcomes.
- (7) Validate results and look for patterns or situations that incur invalid pWCET estimates. To do so, the statically determined WCET as obtained in step (3) is assumed as the "reference value".

3 Preliminaries

This chapter grapples with Real Time Operating System (RTOS), Worst-Case Execution Time (WCET) and Probabilistic Worst Execution Time (pWCET) concepts.

3.1 Real Time Systems

For a Real Time Operating System (RTOS) the execution accuracy does not rely strictly on the logical results, since the execution time to reach the logical result is an essential parameter (VARGAS; GREEN, 2015). Time restriction underlines the main difference between RTOS and general Operating System (OS).

Time requirements are indispensable for some controlling sets. Owing to this, developing a real time system could be the best choice concerning control systems design. Costa (2010) affirms that real time systems work for several activities, such as control science experiments, medical images, industrial control process, robotics, aviation, and so forth.

Hardware characteristics have an impact directly on the real time system execution. Moreover, choosing hardware components considering time restrictions is quite imperative. Reliable and fault-tolerant hardware avoid errors and allow the system to manage predictable errors. Furthermore, hardware speed must be accordingly suitable with the design time restrictions (SHAW, 2003).

Real time system's tasks have a time restriction parameter named deadline. This parameter bounds the maximum time to execute a task with no error. Disregarding the task deadline may cause system damage (BUTTAZZO, 2012). The lost deadline hazard varies for each system, and, due to this, the literature classifies RTOS in Soft and Hard RTOS.

Disregarding deadline in Soft real time systems, which incurs failures, is proportional to a good system execution (OLIVEIRA; FRAGA; FARINES, 2000). By the way, to lose deadline in Soft RTOS implies soft hazards.

On the other hand, loose deadlines in Hard RTOS may cause catastrophic damage. For some activities such as airplane control, for instance, Hard RTOS is imperative. In this scenario, an unpredicted fault may damage all equipment and crew. Furthermore, disregarding tasks deadline could affect the environment around the control system. The RTOS time restrictions are commensurable with regard to system complexity and sensitivity (OLIVEIRA; FRAGA; FARINES, 2000).

However, Data Dependencies and History Dependencies may affect execution time. Bernat, Colin and Petters (2002) say that Data Dependencies are related not only to the algorithm under execution, but also refer to hardware architecture implementation, which may affect the division and multiplication execution time, for instance. History Dependencies refer to cache memory, pipeline and branch prediction algorithms that influence the code execution time. Furthermore, the clock difference between the processor and peripherals units might, for instance, deviate execution time for instructions as load and store.

3.1.1 History Dependencies

Abella et al. (2014) explain that cache memory loads content into fixed-size lines. Moreover, there are many different cache designs in current processors and in literature. Furthermore, regarding the most common cache architectures, it is possible to classify in Direct-Mapped (DM) caches, Fully-Associative (FA) caches and Set-Associative (SA) caches.

In DM caches, each memory address content has only one possible location in cache. A mapping function chooses the location. In FM caches, any memory address content might be stored in any cache address. In this case, as soon as the cache loads a new data, the replacement function decides which location to replace. This cache might use several replacement policies.

SA cache, contrariwise, incorporates both concepts. This method splits all memory and cache addresses in unlike sets. Each set-in memory is direct-mapped to some set-in cache, as in DM ones. Nevertheless, as in FM (First-Miss) cache, any memory address in a set might occupy any address on its mapped cache set. Moreover, SA applies both mapping and replacement functions.

It is important to point that cache and its policies affect the execution time. Therefore, cache usage increases execution complexity and decreases execution predictability. In other words, a given algorithm executing in a given architecture, even computing the same logical result, might spent different times for executing when applying different cache techniques.

3.2 The Worst-Case Execution Time (WCET)

The program *Worst-Case Execution Time* (WCET) is the time upper bound to execute the program code in a specific processor. WCET is essential to real time system schedulability analysis and time warranty, especially on hard real time systems (STARKE, 2012). The instructions execution time is no longer constant though.

Oliveira, Santos e Deschamps (2006) affirm that RTOS time analysis may consider other time parameters. The Best-Case Execution Time (BCET) is the lowest time to complete a task execution. On the other hand, the Average-Case Execution Time (ACET) is the average time to fulfill task execution. By the way, BCET and WCET are both upper and shorter time bounds for task execution.

Current WCET analysis techniques are too pessimistic, once the result is the absolute execution time upper bound. Further complex cases require several simplifications. Bernat, Colin and Petters (2002) affirm that finding a WCET by measuring the test cases set may disregards the true worst-case. In industry, most engineers add a safety margin to the computed WCET aiming to compensate the reliability fault and the uncertainty problems that measured samples may cover in the worst-case.

ABELLA et al. (2014) classify some methods to find WCET in Static Deterministic Timing Analysis (SDTA), Static Probabilistic Timing Analysis (SPTA) and Measurement-Based Probabilistic Timing Analysis (MBPTA).

3.2.1 Static Deterministic Timing Analysis (SDTA)

Static Deterministic Timing Analysis (SDTA) employs a detailed system model to derive a safe WCET upper bound (WILHELM et al.,2008). SDTA techniques demand a hardware structural knowledge. Cache analysis, for instance, requires a placement and replacement function expertise.

ABELLA et al. (2014) classify SDTA methods in low-level analysis and high-level analysis. Low-level ones focus on processor architecture and high-level analysis determines the worst execution case, among all the possible paths in a program.

1) Low-Level Analysis: Regarding cache in low-level analysis requires observing several characteristics that may affect WCET estimation. Considering the type of cache associativity behavior during execution is imperative. Theiling, Ferdinand e Wilhelm (2000) introduce the Cache Hit/Miss Classification (CHMC):

- Always-Hit (AH): all fetched data results in a cache hit.
- First-Miss (FM): It does not classify the first occurrence neither as hit nor as miss, though considering any other as hit.
- Always-Miss (AM): all fetched data results in a cache miss.
- Non-classified: it does not classify the occurrences as hit nor as miss.

There are three techniques to define CHMC category, the Must analysis, the Persistence analysis and the May analysis. Those methods apply a static analysis based on abstract interpretation. Must analysis defines if cache is always keeping the same memory block at a given program point. Thenceforth, in this case, CHMC is always-hit. Persistence analysis identifies if cache evicted a memory block soon after being fetched. Consequently, in this case, CHMC is first-miss. May analysis detects if a memory block may be in cache at given program point. If it does not, the CHMC is always-miss. On other hand, if those techniques could not classify the CHMCs, it is non-classified (ABELLA et al., 2014).

2) High-Level Analysis: Implicit Path Enumeration Technique (IPET) is the most common high-level analysis technique. IPET solves the WCET calculation problem as an Integer Linear Programming (ILP) formulation, expressing, using a linear constraints set, the program structure and all the possible execution paths (WILHELM et al., 2008). This technique obtains an upper bound WCET, considering all the programs basic blocks and the subsequent function maximum summation:

$$\sum_{i \in BasicBlocks} Ti * fi$$

Ti is the timing information of the basic block *i*, constant in the ILP problem. *Ti* considers cache effect, CHCMs and cache and memory latencies. *fi* is the basic block execution number, variable in the ILP problem. The ILP solver provides a safe upper-bound function of all the possible execution times. See Appendix A for a detailed example of computing WCET applying IPET.

3.2.2 Measured Based Timing Analysis (MBTA)

According Wilhelm et al. (2008) Measured Based Timing Analysis (MBTA) simulates or executes basic blocks code in hardware. The basic blocks are tasks code or tasks code snippets and every simulation or

execution time is measured. Vary input data allow to cover all the execution paths and discover the worst-case. MBTA defines the BCET as the shortest execution time measured while the WCET as the longest execution time measured one. Furthermore, every measure sample compounds an occurrence distribution, this distribution allows one to find a given execution time occurrence probability.

Figure 1 depicts a MBTA example. The distribution displays the occurrence number of each time. The white distribution describes a single case samples occurrence, all samples considering the same input data. The dark distribution describes all cases occurrences, considering all samples.

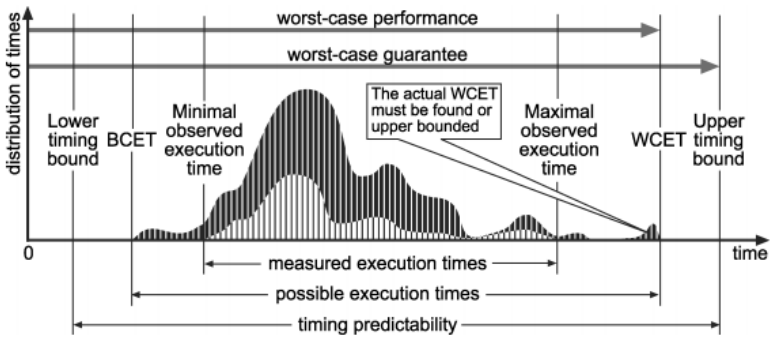

Figure 1 - Example of a Measured Based Time Analysis. Reference: Wilhelm et al. (2008, p. 3).

The Static Analysis dismisses task execution or its simulation on hardware. Static Analysis regard all possible executions path in the task code. A Control Flow Graph (CFG) may describe those paths. After the code analysis, the method relates all iterations upper bounds with the proper basic block on CFG. Match execution time with iteration bounds for every basic block and calculate the worst execution time for every path in code is the way to find the WCET.

The Static method defines every execution time bounds and ensures that the found WCET value will never exceed. This method allows a safe schedule analysis in hard RTOS. However, if a given task depends on some input data, the execution time could be eventually undeterminable. Furthermore, a higher analysis code complexity may incurs a significant increase on computational effort and designs costs, because execution paths increase exponentially as long as the conditional jumps in code increase (WILHELM et al.,2008).

3.3 The Probabilistic Worst-Case Execution Time (pWCET)

Gil et al. (2017) affirm that Static Analysis is unpractical for complex hardware components, although MBTA finds a WCET, the true worst-case execution time may be not found. The Probabilistic Worst-Case Execution Time intends to solve this MBTA problem.

The probabilistic Execution Time (pET) is a Probabilistic Distribution Function (PDF), which describes the execution time probability to a given system. Figure 2 shows a Distribution pET example. In this example, there is a probability of 14% that the code executes in 1.0 ms, for instance. Figure 3 depicts a Cumulative Distribution Function (CDF), which is a cumulative sum of PDF function. In this regard, the “1 – CDF” is the Exceedance Probability, i.e., the probability that the WCET exceeds a given computed value. For example, in

Figure 3 there is a probability of only 10% that the code executes up to 2 ms, or a probability of 90% that the code executes under 2.0 ms, for instance (CDF function). CDF function shows the probability of execution in less or equal a given time. In other words, according to Figure 3, 90% of all execution possibilities spent 2ms or less. On the other hand, the 1 - CDF function shows that 10% of all execution possibilities exceed 2ms. Using this distribution is imperative for computing execution time behavior.

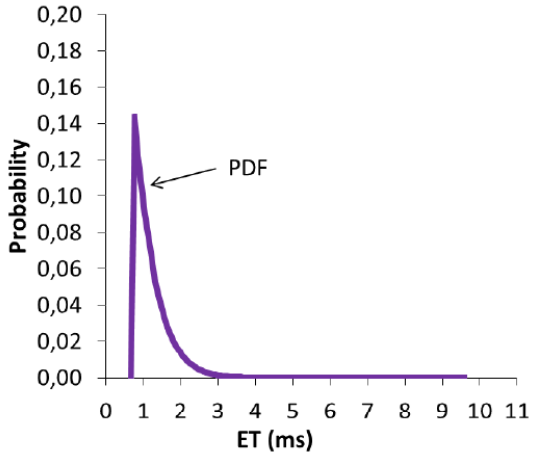

Figure 2 - System Execution Time PDF. Reference: Kosmidis et al. (2014).

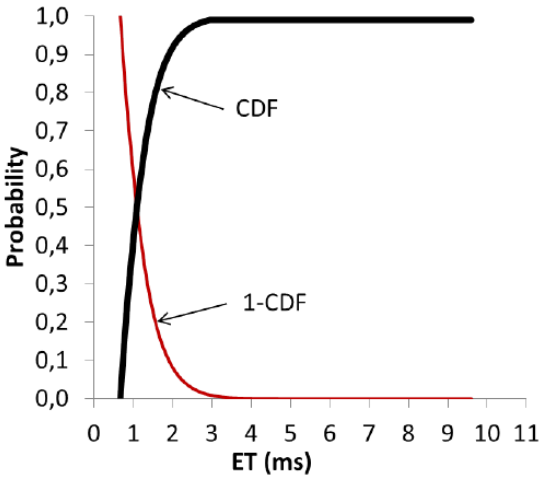

Figure 3 - CDF and 1 – CDF. Reference: Kosmidis et al. (2014).

With pET is possible to find some probability p for a given time value t. Additionally, it is possible to find some time value for a given probability. Furthermore, it is possible to relate WCET information to pET distribution and to define the probabilistic Worst-Case Execution Time (pWCET).

It is imperative compounds an empirical probability distribution with all measured execution times to find the pWCET. From this Empirical Distribution, it is possible to define an Empirical Cumulative Distribution Function (ECDF). EVT compares ECDF to an existing probability function. Figure 4 shows a pWCET estimate. In this case, the probability is $p = 10^{-16}$ for a pWCET equal to 9.5 ms. Note that Figure 4 depicts a tail amplification of the “1 – CDF” function presented in Figure 3.

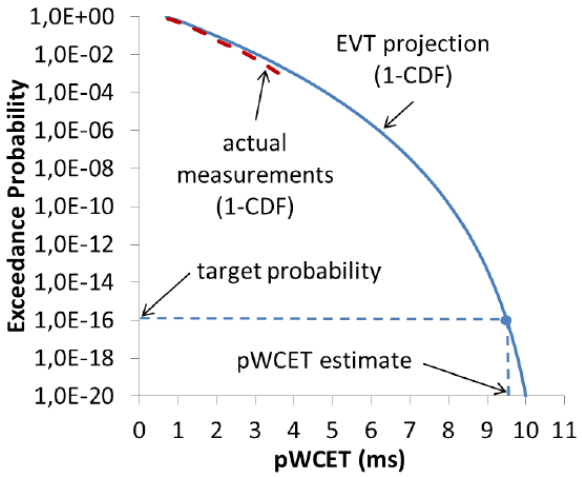

Figure 4 - pWCET estimative example ($p = 10^{-16}$) . Reference: Kosmidis et al. (2014).

The pWCET parameter is the worst-case execution time for a given probability threshold. Furthermore, pWCET researches fall into Static Method and Analytic Method (Davis, Burns and Griffin, 2017).

3.3.1 Static Probabilistic Timing Analysis (SPTA)

The Analytic Method, also called Static Probabilistic Timing Analysis (SPTA), is applicable when a system component or the environment affect the random behavior or time probability. For instance, a random cache replacement (ABELLA et al., 2014). Cucu-grosjean et al. (2012) explain that SPTA derives system model probabilities. Nevertheless, this method requires a considerable information amount about execution behavior, even reducing the required information about program and platform.

SPTA method analyses software in structural level and instruction level and uses a hardware behavior model to estimate the worst-case time behavior in a pWCET distribution. This distribution may contain cases for all possible input data, software and hardware states and executions paths in code. Therefore, SPTA does not execute the real hardware and, due to that, it depends largely on the hardware model accuracy. Abella et. al. (2014) affirm that SPTA is not totally able to analyze multi-path programs and it is more pessimistic than the measure based method.

SPTA may use random variables for probabilistic time behavior, expressed by the pair: <timing vector, probability vector>, also named Execution Time Profile (ETP). The timing vector enumerates all operation latencies while the probability vector lists the occurrence of associated probability. Hence, for an operation Ai, ETP(Ai) = $< \vec{t_i}, \vec{p_i} >$ where $\vec{t_i} = \{t_i^1, t_i^2, t_i^3, \dots, t_i^N\}$ and $\vec{p_i} = \{p_i^1, p_i^2, p_i^3, \dots, p_i^N\}$, whereas $\sum_{j=1}^{N_i} p_i^j = 1$.

SPTA always assumes that previous instructions executions do not influence the instruction ETP. So, assuming that the execution times probabilities for each instruction are independent, SPTA deploys the convolution (*) of probability distributions for each instruction execution time along the flow. The result is a probability distribution that describes the timing behavior of all execution path. In other words, if X and Y are random variables that describe instructions x and y execution time, the convolution $Z = X * Y$ is:

$$P\{Z = z\} = \sum_{k=0}^{k=+\infty} P\{X = k\}P\{Y = z - k\}$$

Abella et. al. (2014) show the forthcoming example: if an instruction x has $t_x = \{1, 10\}$ and $p_x = \{0.9, 0.1\}$, while the instruction y has $t_y = \{2, 10\}$ and $p_y = \{0.5, 0.5\}$ the convolution is:

$$Z = ETP_{(X*Y)} = X * Y = \begin{pmatrix} 1 & 10 \\ 0.9 & 0.1 \end{pmatrix} * \begin{pmatrix} 2 & 10 \\ 0.5 & 0.5 \end{pmatrix}$$

$$= \begin{pmatrix} 3 & 11 & 12 & 20 \\ 0.45 & 0.45 & 0.05 & 0.05 \end{pmatrix} = < \{3,11,12,20\}, \{0.45,0.45,0.05,0.05\} >$$

3.3.2 Measurement-Based Probabilistic Timing Analysis (MBPTA)

The Measurement-Based Probabilistic Timing Analysis (MBPTA) computes software execution time directly over hardware. For this purpose, the input data and vector may comprehend a relevant set of code execution paths and different software and hardware states that affect the time behavior (ABELLA et al., 2014).

MBPTA, unlike SPTA, does not require a significant information about hardware behavior, as memory, bus and cache probability times, for instance. This method is more attractive for industry, since it finds execution time probabilities by collecting end-to-end run samples on target hardware. Cucu-grosjean et al. (2012) affirm that MBPTA estimates pWCET trough an observed execution times collection. However, literature does not specify the required amount of samples and whether there are inferences due to the amount. The EVT has been applied to MBPTA in order to provide the execution time probability that a program exceeds a given threshold, based on Complementary Cumulative Distribution Function (CCDF), or "1 – CDF": the Exceedance Probability of the observed collection.

Instead of taking the worst-case obtained and add a safe margin, this method employs the static analysis of EVT based observations to estimate the pWCET distribution. EVT requires that all execution time samples are described as independent and identically distributed (i.i.d.) random variables. In other words, all samples must represent the same r.v. behavior and, further, a collected sample outcome cannot influence other samples outcomes. However, i.i.d. observed execution times obtained on a given processor does not make that processor MBPTA compliant. MBPTA has its own conditions beyond those EVT requirements (CAZORLA et al., 2013).

For MBPTA applicability all execution time variations sources in the system must be statically (i.i.d.) or probabilistic upper-bounded (CUCU-GROSJEAN et al., 2012). This is imperative to correctly compute latencies in analysis.

Abella et. al. (2014) explain that having ETPs at the dynamic instructions level fulfills all MBPTA requirements. At cache level, every dynamic cache access must be defined by a hit and miss probability. However, MPBTA does not require an upper bound probability derivation for cache access, once MBPTA estimates pWCET based on observations and does not derive pWCET estimates by convolution like SPTA.

3.4 Comparison

All the methods previously mentioned may be applied to deal with the worst scenario of a given system, nevertheless, each procedure has advantages and constraints. Furthermore, even WCET and pWCET metrics have their distinguished characteristics.

A WCET value represents the worst-case of the worst scenario that a given system may face, so this implies a high reliability and accuracy, whereas it requires the knowledge of every system singularity that may affect execution, increasing complexity, effort and costs to define the worst-case. In the meantime, the pWCET conciliate complexity and accuracy adding failures probabilities to analysis, since the probability models may be less intricate than the system's model. Although the lower accuracy the pWCET is worth when the failure probability is known and minimum. Table I shows a brief comparison between WCET and pWCET metrics:

Table I - Comparison between WCET and pWCET

Metric	Main characteristics	
	Drawbacks	Advantages
WCET	Requires too much data High effort and cost involved	Accurate Reliable
pWCET	Estimation as outcome Accept failures (small probabilities)	Build a model for prediction Less effort and cost involved

Static Deterministic Timing Analysis (SDTA) compute WCET according to the previously provided system model and its characteristics, this procedure requires considerable effort and costs. On the other hand, Measured Based Timing Analysis (MBTA) compute WCET measuring the system execution, the worst measured scenario is regarded as WCET. Therefore, SDTA is more accurate than MBTA, however it may apply some margin offset for safety and it involves less cost and effort.

Static Probabilistic Timing Analysis (SPTA) also requires a provided model of the system to compute its pWCET, but this procedure regards the probability from this model, so the accuracy depends on the model. Measurement-Based Probabilistic Timing Analysis (MBPTA), on the other hand, measures the system in order to compute its probability model. Based on the probability model, MBPTA defines the pWCET considering a desired probability threshold.

Table II shows a brief comparison between the methods SDTA and MBTA, it also compares SPTA and MBPTA:

Table II - Comparing SDTA to MBTA and SPTA to MBPTA

Method		Main characteristics	
		Drawbacks	Advantages
WCET	SDTA	Intricate	Accurate Reliable
	MBTA	Uncertain coverage	Simpler
pWCET	SPTA	Depends on the provided model accuracy	Based on provided probability model of the system and its components
	MBPTA	Depends on system execution measurements Unknown accuracy	Based on empirical model of the system May estimate the execution for many probability thresholds

Considering these different methods, this work aims to apply and investigate MBPTA efficiency exploring its vulnerabilities, since this is the most promising and challenging method currently.

4 State of the Art

This section presents the state-of-art approach to find WCET named: EVT. It is worthy to point that the proposed work is to analyze the performance of MBPTA with EVT because this technique has been applied in several scenarios involving extreme bounds presenting accurate and reliable outcomes. Also, it regards single path programs only, for applying EVT to compute multi-path WCET, see APPENDIX B – EVT for multi-path WCET.

Further, this section presents a comparison among EVT and two different approaches to compute the worst-case: Path Upper-Bounding (PUB) and Extended Path Coverage (EPC). For more details about these methods see APPENDIX C – Path Upper-Bounding (PUB) and APPENDIX D – Extended Path Coverage (EPC).

4.1 Extreme Value Theory (EVT)

MBPTA technique delimits WCET upper bounds based on statistical analysis of execution times, which uses the EVT, a statistical theory developed to estimate uncommon events probabilities. Several fields apply EVT nowadays, such as engineering, financial, earth science and traffic predict. Hydrology also applies EVT to estimate an abnormal flooding event probability, for instance, defining the maximum flow rate which a given flow gate must support. Estimated pWCET is commonly associated to a desired probability threshold. Critical code executing on complex processors might apply this technique with a lower cost than in static methods (SILVA; ARCARO; OLIVEIRA, 2017).

Lima and Bate (2017) define EVT in the following steps: (a) collect a desired random variable samples; (b) select the maximum samples in (a); (c) find an statistical model to fit the selected samples in (b); (d) validate the model and (e) determine an upper limit (probability threshold) based on the model found in (c) since that model was validated in (d).

The literature presents two techniques that handle empirical samples to determine pWCET: Block Maxima (BM) and Peak Over Threshold (POT). The EVT initial objective is to collect the proper values that compound the distribution tail of the system behavior model and fit this model into a Generalized Extreme Value (GEV) distribution. After collecting samples, EVT selects the maximum values, grouping all samples into equal blocks size and uses the block maxima method to obtain a block maxima with the maximum values of all the blocks. This approach is known as the Block Maxima (BM). Alternatively, EVT may select maximum values, picking all the samples above a given threshold (CUCU-GROSJEAN et al., 2012). This approach is known as the Peak Over Threshold (POT). There are also two types of software codes that EVT can be applied: single path programs and multiple-path programs. The approaches BM and POT as well as the single- and multiple-path type programs will be described along with this section.

G. Lima, D. Dias, and E. Barros. (2016) show a technique to visualize how much the EVT distribution fits on the ECDF. This technique is the quantile-quantile graphs (QQ-plot). It is, essentially speaking, a plot

of the empirical quantile observed values against the target distribution quantiles. This graph shows a distribution quantiles in one axis, while it similarly shows the compared distribution quantiles in the other axis, so if those distributions are equal, the graph will show a straight line, like x = y. The Figure 5 shows a quantile-quantile graph example, in which the x axis shows the model quantiles while the y axis shows the empirical quantiles.

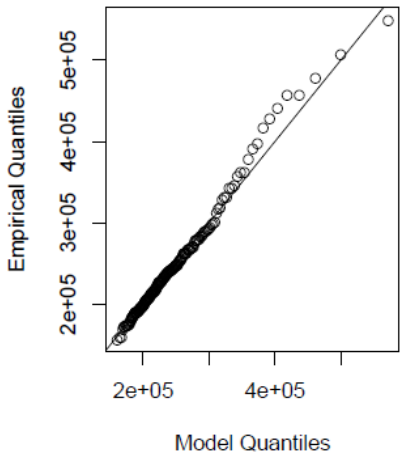

Figure 5 - Quantile-quantile graph example. Reference: G. Lima, D. Dias, and E. Barros. (2016).

Some scenarios may require a higher samples number to analyze different block numbers or block sizes to fit the model. In this case, the only constraints are the spending time and the costs to collect all the measured execution time (LIMA; DIAS; BARROS, 2016).

Silva, Arcaro and Oliveira (2017) describe EVT application on MBPTA as the fit of the collected samples with a known EVT distribution, this samples must be the hardware execution times, previously observed and gathered to this analysis. Furthermore, GEV may express these distributions behavior. However, even small variations on GEV parameters might result in significant pWCET variations, which turns the fitting process harder.

Cucu-grosjean et al. (2012) bring a detailed work about EVT. In this work, a software was executed on a given processor and observed 1000 times. From the observations, 1000 execution time samples are collected. Regarding it, it is possible to model those samples with independent and identically distributed (i.i.d.) random variables, so it can find the program pWCET through an Empirical Cumulative Distribution Function (ECDF). The inverse ECDF tail shows the pWCET possible values, which allows estimating a pWCET for a defined probability threshold. However, this approach requires a good ECDF tail model similar to the system behavior, which would need a significant observations amount to fit the low probability model. In this example, the samples number is enough to describe the real system behavior, and the maximum confidence for 1000 observations is p(execution time > WCETestimated) = 10^{-3}. In other words, the probability of WCETestimated being larger than a given execution time sample is 0,999, whereas the probability of a execution time sample being larger than the WCETestimated is 1 – 0,999 = 10^{-3}.

EVT is a mathematical method to estimate the extreme values probability of known rare events. This technique results in the maximal (or minimal) distribution function values of n samples collected and modelled

with random variables. The EVT method is similar to Central Limit Theory, but estimating the extremes instead of the average (EDGAR; BURNS, 2001).

Cucu-grosjean et al. (2012) brings the following theorem:

"**Theorem 1:** Let $\{X_1X_2, \dots, X_n\}$ be a sequence of i.i.d. random variables and let $M_n = max\{X_1X_2, \dots, X_n\}$. If F is a non-degenerate distribution function and there is a sequence of pairs of real numbers $(a_n,\ b_n)$ such that $a_n\ \geq 0$ and $\lim_{n\to\infty}\left(\frac{M_n - b_n}{a_n} \leq x\right) = F(x)$, then F belongs to either the Gumbel, the Frechet or the Weibull family".

This theorem provides the main EVT result. The distribution function F describes the common function of n random variables. In this case, random variables used to model program execution time. There are two main hypotheses required for this theorem: random variables are independent and identically distributed and the real numbers sequence $(a_n,\ b_n)$ must exist.

To define pWCET is imperative to collect the system execution time samples. Gil et al. (2017) claim that the process objective is to analyze the extremes samples (maximum). There are two techniques to select the extremes samples, a block Maxima (BM) selection or just picking all the samples higher than a defined threshold, Peak-over-threshold (POT) technique.

Cucu-grosjean et al. (2012) hold the view that the block maxima theoretical basis affirms that if some data fits a known CDF, then the block maxima of those data fits the same known CDF.

The maximum values selection affects EVT outcomes directly as far as the total observations number. Regarding BM method, the block size might embarrass the distribution fitting. Cucu-grosjean et al. (2012) emphasize that the larger the block size, the better it is to fit the tail. However, the block size is directly correlated to the number of blocks, considering that the total samples number is constant. For instance, 1000 samples may be divided into 10 blocks of 100 samples or into 100 blocks of 10 samples or even into 1 block of 1000 samples. The samples number in final block maxima is exactly suited to the same number of blocks. In other words, increasing the block size decreases the final block maxima.

To apply EVT under BM approach, the maxima block must converge with one of three possible distributions: Gumbel, Weibull or Frechet. There are three parameter do describe those distributions: shape (ξ), scale (σ) and location (μ).

The Generalized Extreme Value (GEV) comprehends those three distributions. Also, determining GEV parameters proves the real number sequence $(a_n,\ b_n)$ existence. Using GEV, the shape parameter value indicates which distribution to use. If $\xi < 0$ the distribution is Weibull, if $\xi > 0$ the distributions is Fréchet and $\xi = 0$ indicates the distribution Gumbel. Peak-Over-Threshold approach uses the Generalized Pareto Distribution (GPD) instead.

Cucu-grosjean et al. (2012) define GEV Cumulative Distribution Function as follows:

$$F_\xi(x) = \begin{cases} e^{-(1+\xi\frac{x-\mu}{\sigma})^{\frac{1}{\xi}}}, & \xi \neq 0 \\ e^{-e^{-\frac{x-\mu}{\sigma}}}, & \xi = 0 \end{cases}$$

According to Gil et al. (2017), further to determine GEV parameters is imperative to select the appropriate CDF (Gumbel, Frechet or Weibul), using a goodness-of-fit test. However, some tests may be

inappropriate for fitting extreme values. Edgar and Burns (2001) indicate that Gumbel distribution fits well the WCET estimation problem.

4.1.1 Empirical Distribution Validation

The theorem 1 requires that the random variables sequence must be independent and identically distributes (i.i.d.). Cucu-grosjean et al. (2012, 3) defines the Independence Random Variables concept as:

"Two random variables X and Y are 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."

On other hand, the identically distributed Random Variables concept is (CUCU-GROSJEAN ET AL., 2012, 3):

"A sequence of random variables is identically distributed if all random variables have the same probability distribution."

Moreover, a random variables sequence meets the i.i.d. requirements when they are independent and have the same distribution function. For instance, a sequence of dice rolls, each roll as a random variable, is i.i.d. as the obtained outcomes of independent events and the r.v. describes the same event.

4.1.2 Block Maxima (BM)

Literature presents a technique that handles empirical samples to determine pWCET called Block Maxima (BM). Block Maxima technique allocates all the samples in basic blocks, whose the higher sample of each basic block defines a new block called *maxima*. The number of observations in the maxima block is equal to the number of basic blocks. However, literature suggests that the ideal sample number in each basic block must be enough to describe the program behavior properly (ABELLA et al., 2014).

After filtering the maximum values from basic blocks and gathering them in the Maxima Block, this data is used to determine the distribution and its parameters values that better fits those maximum values, in order to compute pWCET. Moreover, the samples amount used on the analysis may affect the fitting characteristics (CUCU-GROSJEAN et al., 2012).

Based on this assumption, this work aims to assess how the samples amount, especially the number of blocks and their sizes, can prejudice this method outcomes. Literature doesn't inform if there is an optimal point neither a correlation between the size of the basic blocks and the Maxima Block that affects the computed pWCET.

4.1.3 Peak Over Threshold (POT)

Literature also presents the Peak Over Threshold (POT) technique to compute pWCET based on empirical samples. Peak Over Threshold technique, on other hand, is simpler than BM. Instead of splitting samples into blocks and filter the maximum value. It just selects all the samples over a given limit (threshold) value. The selected observations compose a set that feeds the MBPTA process. Therefore, this threshold value affects directly the samples number used under analysis and, hence, the outcome. Choosing a lower threshold, the most of samples either all samples may be regarded in MBPTA. On the other hand, choosing a higher threshold may force the process to regard a small number of samples, for instance.

Therefore, this work also aims to assess how the number of samples affects the results obtained by POT. Literature doesn't inform how to choose the best threshold value and the minimum number of samples required to compute the pWCET properly.

4.1.4 Differences between BM and POT

There are some open challenges and open concepts to applying both techniques to determine a program WCET. It is important to notice that several program characteristics, such as the number of paths, memory access inside loops and vector or matrix sizes, for instance, may influence the distribution function. As well as the relationship between the program behavior and the proper number of output observations that are produced by the program (CUCU-GROSJEAN et al., 2012). Both techniques handle empirical samples previously collected and both approaches may compute the pWCET based on GEV and, hence, based on Fréchet, Gumbel or Weibull distribution family, according to MBPTA process.

The difference between BM and POT is basically the process to samples to feed the MBPTA process. POT picks all samples whose value is higher than a given threshold to analysis whereas BM divides all samples into basic blocks and then picks the highest value of each basic block to analysis. Although literature affirms that each basic block, on BM technique, must contain samples enough to describe the program behavior, there is a lack about how to validate the basic block before to pick the maximum value. Also, it is worthy to notice that BM is more complex than POT (CUCU-GROSJEAN et al., 2012).

4.1.5 Single Path WCET

Applying EVT may require some assumptions as independent executions times and identically distributed (i.i.d.). Furthermore, the program under analysis must run in isolation with no system calls. Although it is not a realistic scenario, this assumption must be guaranteed for the success of the current analysis method (CUCU-GROSJEAN et al., 2012).

Regarding a fully deterministic system executing with identical start conditions, a single execution path has to yield a unique execution time. However, there are interference effects due to hardware features, such as Cache hit/miss and DRAM refresh cycles, and the impossibility of ensuring identical starting conditions every

time an execution begins. There are no statistical guarantees for that execution time variability, though some approaches have been taken place, in the past, as for applying EVT to represent this nature of the variability (CUCU-GROSJEAN et al., 2012).

Furthermore, if every jitter source is independent, the response times converge on average to a closed form distribution. According to Central Limit Theorem, this is the normal distribution due to the finite variance of the input distributions in most cases. On other hand, if independent random variables cannot represent jitter sources, those sources upper bound must be considered as constant (CUCU-GROSJEAN et al., 2012).

A. Input Data Dependence

Using a single-path with fixed data on a simple and known architecture is possible to compute exactly the execution times distribution when observing this path hardware behavior with static probabilistic model. This allows evaluating the EVT behavior and increasing the method confidence to apply to more complexes architectures.

When performing a measurement-based analysis, it is imperative to provide a program input data suite to stress the system and to induce the worst-case behavior. The input data may affect the execution time, even when it does not affect the execution path. For instance, multiplying some numbers could spend more or less cycles than other numbers. Moreover, data structures contain pointers, which may refer to different memory addresses depending on the input values, hence it affects cache latencies and memory access pattern.

Generally speaking, it is impossible to provide or even to statically define the worst-case data input at deployment time. Therefore, the platform design must have no effect from input data. Alternatively, there are approaches that force operations, such as divisions, to take a pessimistic safe assumption, which could be their upper-bound execution time. Paolieri et al. (2009) introduce the Worst-Case Mode, a technique that delays all resource requests until it detects their maximum execution time. This approach is transparent to execute programs when they are active and it may be configurable by software. There is a similar solution for data-dependent memory access, in this case, accounting the input data effects through flags. For instance, if a given input data dependent memory operation executes once, there is no reason to use cache for this instruction.

B. Platform Requirements

The MBPTA, in EVT approach, provides a pWCET estimate for a given exceedance probability. This approach requires that all execution time variation resources in the system must have a probabilistic behavior that measurements on platform can detect the events that vary within the execution time (CUCU-GROSJEAN et al., 2012).

A possible approach is to analyze all hardware resource with too high latency and randomize the timing behavior to upper bound when incurred pessimism is acceptable (CAZORLA et al., 2013). Applying this ensures that only random events might affect the execution time. Moreover, any detected frequencies of execution time shows probabilities with confidence level provided by EVT. Therefore, it is possible to use the observed system behavior at the test to predict the behavior during a given operation.

In standard not-time randomized architecture the execution time frequencies detected in tests do not serve as actual probabilities. In other words, standard not-time randomized architecture cannot apply this approach because only not randomized events may affect execution time. Therefore, it may not ensure the execution times occurrence probability of the observed behavior in future (CAZORLA et al., 2013).

C. Independent and Identically Distributed Observations

It is necessary to ensure that each execution runs independently in order to hold i.i.d. properties. A feasible approach is to force the hardware state to always before a run starts. At path level, identical distribution requires the latencies to be chosen independently to compose the total execution time (CUCU-GROSJEAN et al., 2012). A probability must be associated to every path execution time possibilities. It is imperative that an execution observed timing performance does not influence other executions timing performances.

D. Minimum number of observations

Griffin e Burns (2010) affirm that the results could not bound execution time for every points when using Gumbel distribution function, or even any continuous function, when approximating a discrete function to an execution time value produced by a given program. Figure 6 illustrates this limitation to fitting discrete and continuous functions.

In this case, the discrete function has values at 5, 10 and 15. The defined Gumbell function (continuous line) meets all the points when fitting with the discrete function. However, the uncovered points cannot be disregarded since MBPTA collects a finite number of execution time observations. The dashed line shows the possible values between the collected values.

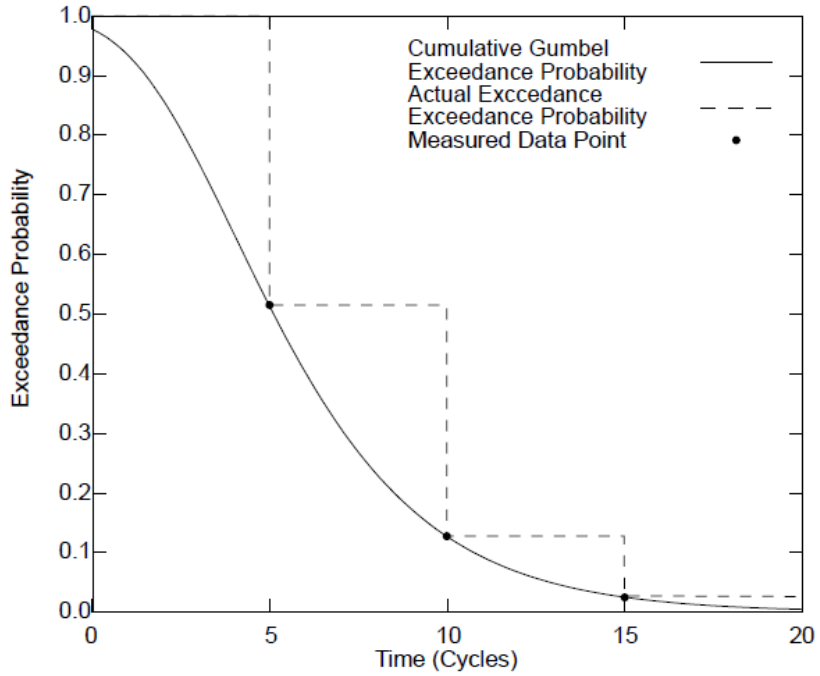

Figure 6 - Illustration of unsafe values due to the implicit assumption fitting discrete and continuous functions. Reference: Griffin e Burns (2010).

A different solution proposing to solve this limitation is to add an offset to the continuous function in order to cover all the possible values in the unmeasured points. Figure 7 shows how the offset can cover all the possible values.

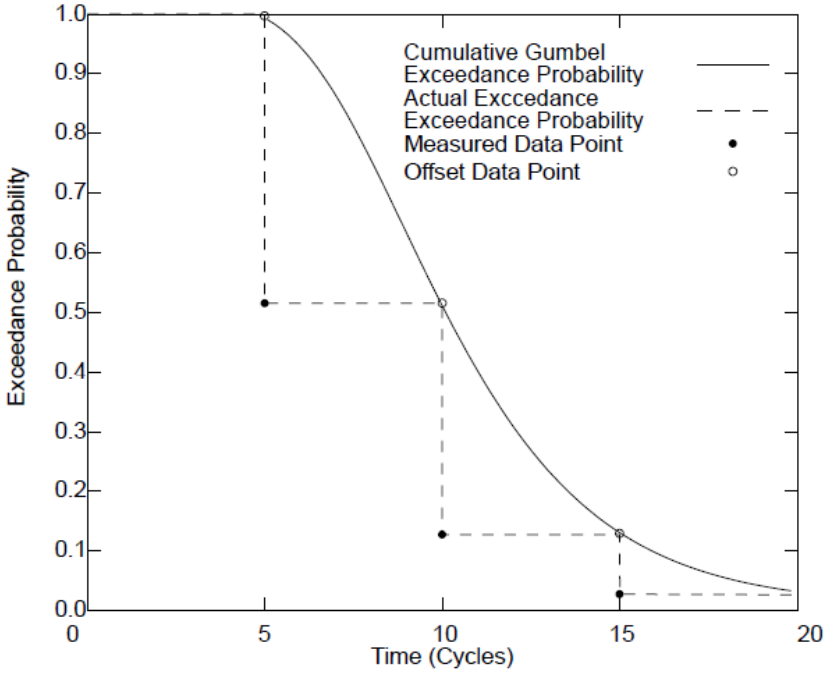

Figure 7 - An illustration of how to offset the Gumbel distribution to guarantee safe values. Reference: Griffin e Burns (2010).

To get a high confidence level in a distribution, it is required a sufficient observations number to produce an authentic execution time description of the program. Cucu-grosjean et al. (2012, 5) define it as follow:

"For a given sequential program P and an architecture A, n is the minimum number of observations if there does not exist $m \in \mathbb{N}$ where $m > n$ such that the Gumbel CDF obtained for n observations is an upper bound for the WCET of P and also exceeds the CDF obtained for m observations."

Cucu-grosjean et al. (2012) obtain the minimum observations number contrasting the EVT tail projection deviation regarding a increasing executions number. The iterative process is described as follows:

1) Run $N_{current} + N_{delta}$ times the program.
2) Project, with EVT, a tail for $N_{current}$ executions and another for $N_{current} + N_{delta}$.
3) If the two EVT distributions have a difference above a given threshold, it is necessary to repeat the step 2 considering $N_{current} = N_{current} + N_{delta}$ and making N_{delta} more runs. If the difference is below the threshold, it ends the process.

Since N_{delta} is constant, the proportional effect on new runs reduce when increasing$N_{current}$, which ensures the convergence algorithm. However, a strict convergence for some local minima is not ensured. Changing the algorithm may compensate this, stopping the process after some consecutive iterations instead of stopping as soon as the difference between distributions is under the threshold. Cucu-grosjean et al. (2012) considered 5 consecutive executions below the threshold to be a sufficient one to deal.

Bradley. (1968) defines a Continuous Rank Probability Score (CRPS), a metric to compare EVT distributions, or functions:

$$CRPS = \sum_{i=0}^{+\infty} [fx(i) - fy(i)]^2$$

In this case, fx and fy represent a distribution functions that operate in the same value domain. The difference threshold depicts the significance level in hypothesis tests. Reducing the threshold value increases the confidence result, although this requires a higher observations number.

E. EVT Step-by-step

Cucu-grosjean et al. (2012) describe the steps to apply EVT in single-path programs in order to obtain a stable result:

1) Collect observations: the author initially suggests a sample set with approximately 100 execution times observed. Afterwards, in each round the N_{delta} must be added as the previously described process.
2) Grouping: This step may use a technique, as Block Maxima or Peak-Over-Threshold, to convert the measured frequency distribution into a suitable EVT distribution.
3) Fitting: It uses the empirical distribution, 1 – ECDF actually, to estimate the GEV distribution, hence it defines the GEV parameters as the calculation previously described.
4) Comparison: It compares, according to Continuous Rank Probability Score metric, the current round distribution to the previous round.
5) Converge: If the difference between distributions does not reach a given threshold, as defined by CRPS metrics, the process starts a new round at step 1. Otherwise, to consider the distribution

converged, it is necessary to collect more samples doing a few more rounds, the author suggests 5 consecutive rounds in this step.

6) Tail extension: this step defines the final GEV parameter values for probabilities under the threshold.

4.2 Comparison

The Extreme Value Theory is widely discussed in literature since it is applied in other science fields. On the other hand, the EVT success in other areas don't ensure this theory covers time analysis with no vulnerabilities. Furthermore, EVT create a probability model to predict failures without inspecting or change the algorithm code.

Path Upper-Bounding method changes the code in order to equal the size for all paths. This method ensures that a sample represents the WCET no matter the input values or the executed path. Since all paths are equal to the worst one, the MBPTA might focus on other uncertainties, as cache and buses delays, for instance. Although, PUB requires a detailed inspection to the code in order to edit it.

Extended Path Coverage measures each basic block and, instead of creating a probability model for the whole program, it creates models for every basic block, which are computed into path's model and, hence, the program's model. Similar to PUB this method requires code inspection in order to all basic block independently. Table III shows the advantages and drawbacks of EVT, PUB and EPC:

Table III - advantages and drawbacks of EVT, PUB and EPC:

Method	Main characteristics	
	Drawbacks	Advantages
EVT	Depends on input values May not execute the worst path	Widely applied No code inspection
PUB	Detailed code inspection and edition Recently discussed in literature	All paths are equal to the worst Focus on hardware variabilities
EPC	Detailed code inspection Recently discussed in literature	Measure all basic blocks

5 Proposed Methodology

The proposed methodology in this work aims to assess pWCET estimates by Measured Based Probabilistic Timing Analysis (MBPTA), based on EVT. Path Upper-Bounding (PUB) and Extended Path Coverage (EPC) methods are not considered in this study because these approaches have been cited recently in Literature whereas EVT has been mentioned in several researches. Further, EVT has been applied in different scenarios involving extreme limits presenting accurate and reliable results.

In this methodology the MBPTA process uses samples of execution time of a given program. These samples can be collected executing as well as simulating the program. According to EVT these samples might be filtered and gathered to form an empirical distribution which describes the algorithm behavior. Based on

that, this methodology looks for limitations in the samples filtering techniques Block Maxima (BM) and Peak Over Threshold (POT) and also measure how much it affects the pWCET outcome.

BM technique splits all samples in basic blocks. The sample with the maximum value of every basic block form the Block Maxima, which contains all considered data on EVT analysis. This methodology compares EVT analysis for a given algorithm applying different input parameters and number of samples. Executing BM approach, this methodology tries to define the best basic block size. As higher is the number of blocks, lower is the number of samples in these blocks, decreasing the block data quality and, maybe, decreasing outcome accuracy also. For this reason, this work aims to find an optimal block size for BM.

POT filters all samples based on its threshold value to EVT analysis. The threshold is the key for this technique, since a too low threshold might select all samples with no criteria, whereas, a high threshold might filter just a few samples to be analyzed, which may affect the outcome as well. For this reason, this work aims to define an optimal threshold value for POT.

Therefore, this methodology uses different observations numbers and different input parameters values in order to exercise and to point out the vulnerabilities that these EVT techniques may have. Exercising different cases with distinct numbers of samples, blocks and thresholds may draw a clear conclusion about the advantages and drawbacks of these techniques.

Furthermore, this methodology consists of selecting a given algorithm, assemble the code to MIPS architecture, analyzing statically to obtain a WCET, and applying MBPTA to obtain a pWCET. In both analyses, the execution time is measured by executed instructions, as to avoid that hardware relate uncertainties affect the outcome. In other words, this may enhance only the applied techniques uncertainties. Owing to this, this work does not regard execution cycles.

5.1 Specification

This methodology may be summarized in four steps: 1) Context Definition, 2) WCET Definition, 3) Sample Collection, and 4) pWCET Process Analysis, as depicted in Figure 8:

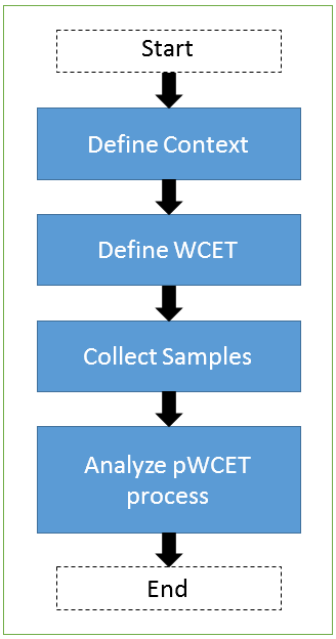

Figure 8 - Methodology steps. Reference: Author.

1 – Context Definition: It is the software and hardware selecting process. The chosen algorithm is so important as the chosen hardware platform. Once this algorithm is executed in the target hardware, both affect the execution time behavior and, hence, the later analysis. The number of paths, the existence of loops and the complexity are some algorithm's characteristics that may be regarded in this deciding step. On the other hand, when choosing the hardware platform is opportune to consider the time uncertainty due to buses and used peripherals, cache memory usage, number of cores in processor, for example.

2 – WCET Definition: It consists of analyzing the defined context and finding its absolute worst-case execution time. To achieve that, both statically analysis and measured-based analysis may be applied, the available tools and analysis complexity should be considered in the Context Definition. This step compute an absolute WCET value, it defines the "reference value" that will be used to validate and compare the pWCET determined by the BM and POT approaches thereafter.

3 – Sample Collection: This step the collects execution time observations of the chosen context. Also, it defines the minimum samples number for the context analysis. This minimum number is computed through Continuous Rank Probability Score (CRPS), as described in 4.1.5. Further, the CRPS determine the smaller probability threshold for the pWCET in the end of the analysis. The collected samples may be filtered by BM and POT methods and, then, analyzed to calculate the pWCET.

The literature does not explain whether the samples collecting process should control the used input values. Owing to this there are two alternatives, control the used input values and don't repeat them, collecting only one sample for input, or don't control the used values, repeating them and assuming the risk of do not execute the worst-case path.

4 – pWCET Process Analysis: it defines the probability distribution that describe the pWCET. Besides that, this step also filters the collected samples according to BM or POT approaches and computes the pWCET according to EVT. Comparing the reference value obtained in step 2 with the calculated pWCET it is possible point how accurate and effective the MBPTA process is for the context.

Figure 9 depicts this step, which applies the BM and POT techniques in the collected observations to estimate the pWCET. There are some input parameters exercised in this step. In this work the parameters are named N, M and L. N is the number of samples before the filtering processes BM and POT. M is the number of observations in the Maxima Block, it is equal to the number of basic blocks in which all the samples are allocated. This parameter refers to the Block Maxima approach only. L is the *timing* limit value that POT uses to filter values before the result computation. This limit number refers to Peak Over Threshold only.

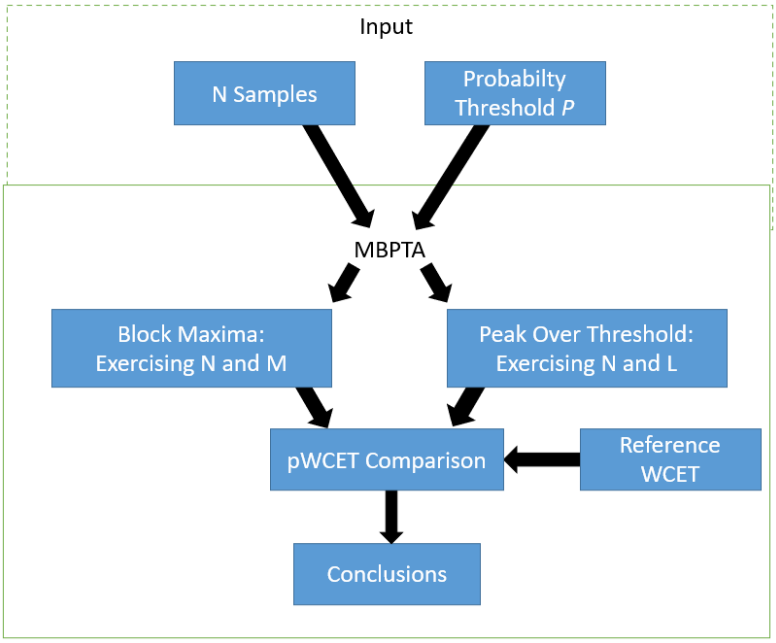

Figure 9 - pWCET Process Analysis step. Reference: Author.

As mentioned above, the M parameter in BM approach defines the number of basic blocs and hence, the number of observations in the maxima block. Together, these basic blocks contain all the collected observations. However, separately, each basic block requires a sample number enough to describe the program behavior. In other words, changing the M parameter may demand more samples and, hence, change the N parameter too. Figure 10 shows the BM approach flow. This methodology applies different values for M and N.

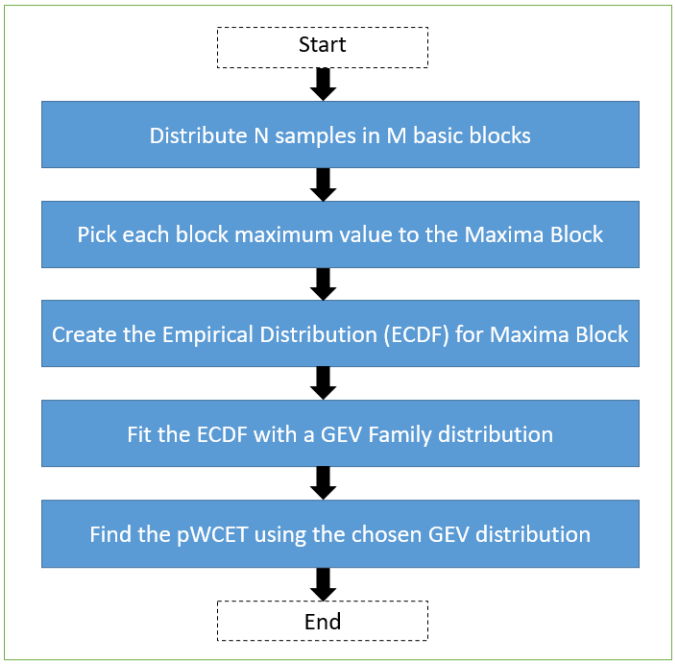

Figure 10 - Block Maxima approach flow. Reference: Author.

The L parameter in POT approach defines the timing limit value for the observations in analysis. However, changing the L parameter changes the samples number under analysis too, which can affect the behavior's depiction. This methodology exercises different values for L parameter. Figure 11 shows the POT approach flow.

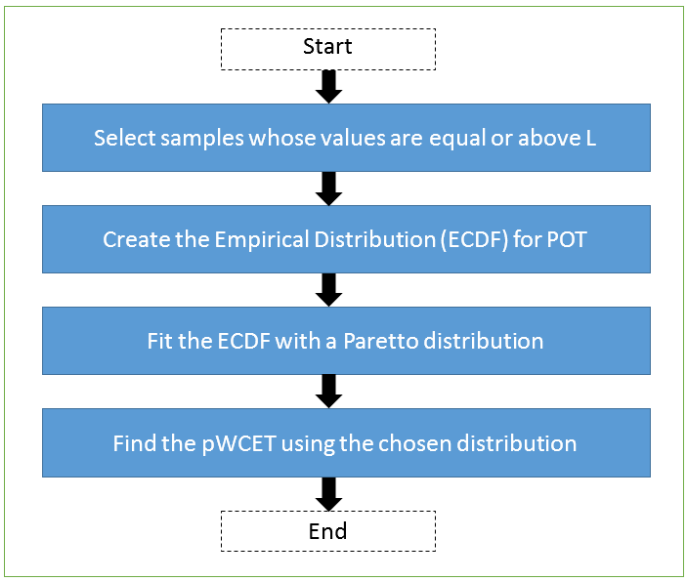

Figure 11 - Peak Over Threshold approach flow. Reference: Author.

5.2 Implementation

The execution context defined to for this methodology is basically executing the algorithms Bubble Sort and Finite Impulse Response filter (FIR) on MIPS processor. For this reason, it is required to assemble these codes for MIPS architecture processor and, hence, run on simulators such as Mars (MISSOURY STATE UNIVERSITY, 2017). This context regards only MIPS single core since it is more predictable and less complex than multi core architectures.

The WCET computation step found the static WCET, manually.

Every time an assembled code is simulated in Mars, for MIPS processor, its execution time is collected as a sample of execution time. Once it collects one sample per simulation, it is required to run the algorithm several times, since MBPTA requires a considerable number of samples. After that, the analysis filter all collected samples according to the techniques criteria.

The N parameter might vary for each algorithm analysis, since different programs might require a different number of samples. The M and L parameters values are specific for each approach, for this reason, their values are defined separately for BM and POT.

It is possible to compute all the EVT calculation by Matlab tool, as it has libraries and functions targeting EVT calculation. To illustrate it, there are two examples of Matlab code in APPENDIX E – Matlab code examples.

There is a function for the Empirical Cumulative Distribution Function (ECDF), called ecdf( ) , which receives as parameter an array containing the samples for analysis. This function returns two arrays (x and y axis) describing the ECDF of the provided data. It is worthy to point the BM and POT filtering process are applied manually before the EVT computation on Matlab.

For computing the Generalized Extreme Value (GEV) distribution, in BM approach, the gevfit( ) function receives the same parameter than ecdf( ), the filtered data for EVT analysis. However, this function finds the Generalized Extreme Value (GEV) distribution and its parameters that better fit to the empirical data. Also, gevfit( ) returns the reference values for the parameters shape (ξ), scale (σ) and location (μ).

Thereafter, it is possible to generate the Cumulative Distribution Function (CDF) using the gevcdf( ) function. This function receives an array with the desired values of the x-axis and the computed GEV parameters ξ, σ e μ. Then it returns the y-axis GEV values for the input x-axis range. The Figure 12 shows the flow with the functions and data involved in this computation.

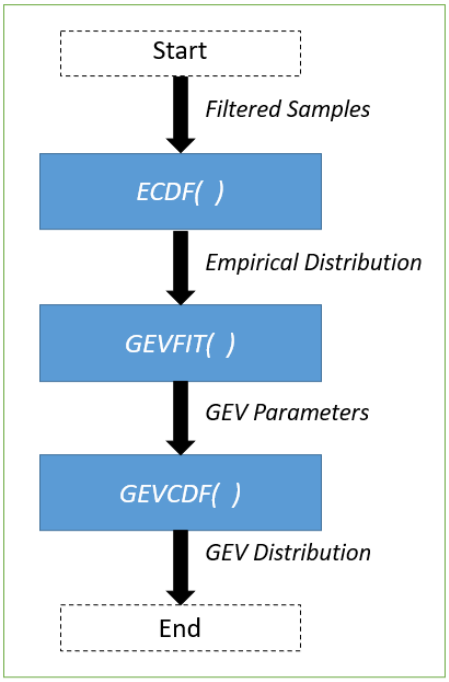

Figure 12 – Flow of Matlab functions to compute GEV Distribution.

Besides that, it is possible to compare the GEV computed distribution to the empirical one, ECDF, and also, it is possible to find extreme values uncovered by ECDF. Furthermore, once the GEV function is defined, it is easy to identify the probability value (y axis) for extreme values (x axis). In other words, it can identify when the function assumes the value equal to the desired probability threshold.

For computing the Grand Pareto (GP) distribution, in POT approach, Matlab provides gpfit and gpcdf functions. The gpfit( ) function is similar to gevfit( ), whereas gpcdf( ) function is similar to gevcdf( ) and returns the y-axis values regarding the computed GP distribution. The Figure 13 shows the flow with the functions and data involved in this computation.

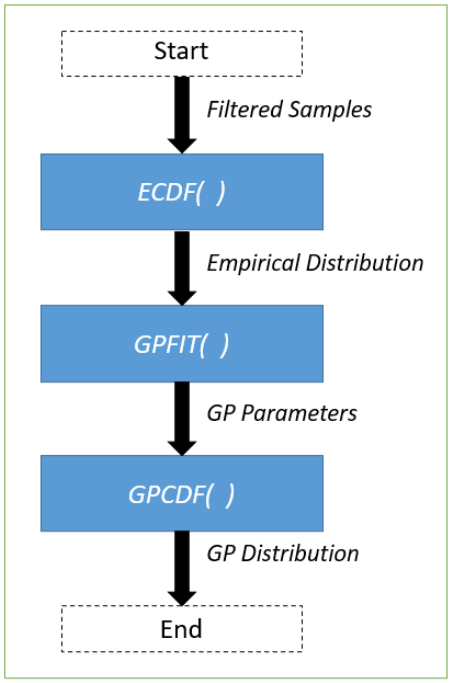

Figure 13 - Flow of Matlab functions to compute GP Distribution.

There examples of Matlab code are in APPENDIX E – Matlab code examples.

6 Validation & Evaluation

The difference among results obtained analyzing a given algorithm must evidence the proposed methodology efficiency. Thereafter, it is possible to point out characteristics that may disturb the EVT results for pWCET. The results validation regards the statically defined WCET. The result accuracy depends on the difference between the obtained value and the reference one, statically computed (SDTA). Aiming to get different results, it analyzes different scenarios considering different number of samples and different sample filtering parameters. Therefore, the outcome accuracy may point how the number of samples and the filtering process affect each scenario.

6.1 The Simulator MARS

The chosen simulator to this work is MARS (MIPS Assembler and Runtime Simulator), developed by Missouri State University. MARS is a lightweight interactive development environment (IDE) for programming in MIPS assembly language, intended for educational-level use with Patterson and Hennessy's Computer Organization and Design. MARS was initially developed for teaching MIPS architecture aiming to clarify and facilitate the understanding of this architecture by the simulation of the entire processor. It can be used through its Integrated Development Environment (IDE) or even from command line. Also, MARS

simulates MIPS-32 instruction set, covering 155 basic instructions of the MIPS-32, approximately 370 pseudo-instructions, the 17 syscall functions and more. Furthermore, its IDE provides program editing and assembling, it also supports interactive debugging. It allows the user to easily set/remove execution breakpoints and to inspect the execution, viewing and editing register and memory contents (MISSOURI STATE UNIVERSITY, 2020).

It is important to enhance that MARS simulates MIPS architecture according to Patterson and Hennessy's descriptions, using the same registers and flags. It also contains a virtual cache memory with some features as counting cache-miss and cache-hit occurrences. Similarly, it has a functional virtual memory for program code and for data. Further, it allows the user to change register and memory values during the execution.

Mars afford to use tools that explore the program behavior. The Instruction Counter tool provides the total number of executed instructions during the whole program. It also displays the executed number regarding different instructions types. There are tools that scan cache memory and main memory access, which allows setting different configurations and memory block sizes, and then it provides the access rates for each memory block during simulation. The Branch Prediction tool is worthy to be pointed, also. Moreover, MARS contains several functionalities and proper tools to learn and understand MIPS single core behavior.

6.1.1 MARS Constraints

Besides all functionalities, it is important to point some constraints in MARS. It disregards the time spent to execute instructions, it only computes the next instruction to simulate and do not mind how much time it takes to do that. Hence, there is no pipeline simulation and it is not possible to detect stalls when accessing memory, for instance. Although it has memory tools to report access rates, it ignores completely buses delays and asynchronous executions.

MARS has Delayed Branching functionality, which aims to reproduce processor behavior on branching. However, this functionality forces the simulator to execute the next line after a branch instruction, independently whether the branch will take or not.

6.2 Bubble Sort Case

The first analyzed context is an ordering vectors algorithm named Bubble Sort. This is a simple algorithm, widely employed in literature. In this case, Bubble Sort must ordering an eight positions vector. Each position may assume values among 0 to 8. The Figure 14 shows the analyzed assembly code:

```
1  .data
2  vec:        .word 8, 7, 5,6, 4, 3, 2, 1
3  vecsz:      .word 8
4
5  .text
6  main:       la $a0, vec
7              lw $a1, vecsz
8              j bubble
9
10 bubble:     li  $s0, 0
11 eloop:      bge $s0, $a1, end
12             li  $s1, 0
13 iloop:      bge $s1, $a1, endiloop
14             sll $t1, $s0, 2
15             sll $t2, $s1, 2
16             add $t2, $a0, $t2
17             lw $t3, 0($t2)
18             lw $t4, 4($t2)
19             blt $t4, $t3, swap
20             addi $s1, $s1, 1
21             j iloop
22 swap:
23             sw $t3, 4($t2)
24             sw $t4, 0($t2)
25             addi $s1, $s1, 1
26             j iloop
27
28 endiloop:
29             addi $s0, $s0, 1
30             j eloop
31 end:
32             li $v0, 10
33             syscall
```

Figure 14 - Bubble Sort assembly code. Reference: Author.

The assembled algorithm code has 10 basic blocks. It is important to enhance that jumps or branches delimit every basic block, either for executing a branch instruction or being called for an instruction in other block. Figure 15 shows the Control Flow Graph (CFG) of the assembled code of this algorithm. Assembling the code may replace some instructions and change the instructions number in each basic block.

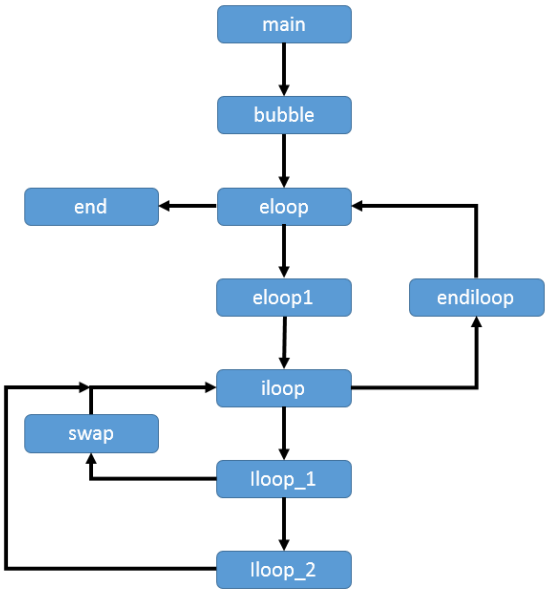

Figure 15 - CFG of Bubble Sort assembled code. Reference: Author.

6.2.1 Static Analysis

The worst-case statically computed regarded the assembled code, similarly to the CFG. It applied the Implicit Path Enumeration Technique (IPET) due to this outcome reliability and accuracy. According to this technique, the number of a block execution times is equal to the summation of all times the execution enters this block and also equal to the summation of all times the execution get out this block. With this assumption, it is possible to determine the maximum execution times for each block from the iterations blocks with known upper bounds.

The IPET analysis for the Bubble Sort algorithm resulted in 826 executed instructions in the worst scenario, since this work regards executed instructions as time unit, the WCET is 826. For a detailed IPET computation for this program, read APPENDIX A - IPET Example: Bubble Sort Case.

6.2.2 MBPTA: Collecting Samples

This algorithm has two variables, the vector and the vector size, vec and vecsz in Figure 14 (line 2 and 3). In this example, vecsz has value 8 and vec has the values 8, 7, 6, 5, 4, 3, 2, and 1. During the analysis this algorithm is executed several times regarding different values for vec, however the vector size is always the same. Considering the vector size and all the values that each vector position can assume, it has 9^8 (or 43.046.721) different possible cases.

Every time the algorithm is executed, the time required to execute the code becomes a sample. During the collecting step, every execution sample is unique, since the input value is unique. In other words, this step never repeats the input vector values. Considering 10000 unique samples, it has a coverage of 0,0232% of all possible scenarios.

Since it is infeasible to execute the algorithm 9^8 times, it is necessary to define the minimum but enough number of samples. The numbers of observations regarded in this analysis follows the Continuous Rank Probability Score (CRPS) explanation in 4.1.5. It basically creates distributions with different number of samples and compare them. As more the number of samples used to compute these distributions, less is the difference between them. To reach the minimum number of samples the difference should be less than the desired probability threshold.

In this scenario the chosen value as desired probability threshold is 10^{-6}, since it is small but reachable also. The initial number of samples is $N_{current} = 3500$, arbitrarily chosen. This choice must consider the time and costs effort to collect samples. Starting CPRS with a high number of samples may induce an useless effort, since the minimum number can be smaller. On the other hand, the chosen number for delta is $N_{delta} = 100$, arbitrarily chosen. This choice must regard the time and costs effort to compute and compare the CRPS distributions, since small the delta number is, bigger is the number of computations and comparisons until the desired threshold be reached. The following table shows the CRPS analysis, the computation regards the equation $CRPS = \sum_{i=0}^{+\infty}[fx(i) - fy(i)]^2$ according to 4.1.5.

Table IV - Samples number ($N_{current}$) on analysis for WCET. Reference: Author.

Rounds	Ncurrent	Ncurrent + NDelta	Max Difference (CRPS)
1	3500	3600	1.35E-06
2	3600	3700	1.54E-08
3	3700	3800	2.45E-07
4	3800	3900	6.66E-08
5	3900	4000	8.06E-09
6	4000	4100	5.95E-12
7	4100	4200	9.92E-07
8	4200	4300	1.08E-10
9	4300	4400	1.41E-09
10	4400	4500	2.60E-07
11	4500	4600	6.27E-07

The last value above the desired threshold 10^{-6} occurred when $N_{current} = 3500$, no one of the following rounds computed differences exceeding the threshold. Therefore, it points an enough number of samples starting from 3600 observations. Furthermore, *it* is possible to point these differences do not decrease linearly, which is a good reason to execute a few more rounds after finding the minimum number of samples in order to assure the Max difference still smaller than the threshold. Although CRPS indicates 3600 samples as a safe number of samples for analyzing, this scenario regards 4500 observations, which means a safe number and may provide more details than 3600 samples. The chosen number represents a coverage of 0,01045% of all possible executions case for this algorithm.

6.2.3 MBPTA: Block Maxima Approach

Since the literature does not specifies the criteria for splitting samples in basic block, in this work it divides samples in basic blocks according the order they were collected. For instance, regarding blocks with 100 samples, the first basic block would contain the 1st to 100th samples.

This Block Maxima analysis regards only 4500 samples and compares different basic blocks sizes, as depicted in table below:

Table V - Different block sizes for Basic Blocks and Maxima Block. Reference: Author.

Size of basic Blocks	1	10	15	18	45	50	90	100	250	300	450	4500
Number of Basic Blocks	4500	450	300	250	100	90	50	45	18	15	10	1
Size of Maxima Block	4500	450	300	250	100	90	50	45	18	15	10	1

The first scenario considered basic block as one sample size and, therefore, the maxima block contains all 4500 collected samples. On the other hand, the last one regards 4500 basic blocks, which implies in only one sample for Maxima block: the highest value sample.

The Maxima block must contain enough samples to represent the execution behavior properly, specially the extreme values (CUCU-GROSJEAN et al., 2012). Regarding this, the next figure shows the histogram for each maxima blocks considering the different sizes.

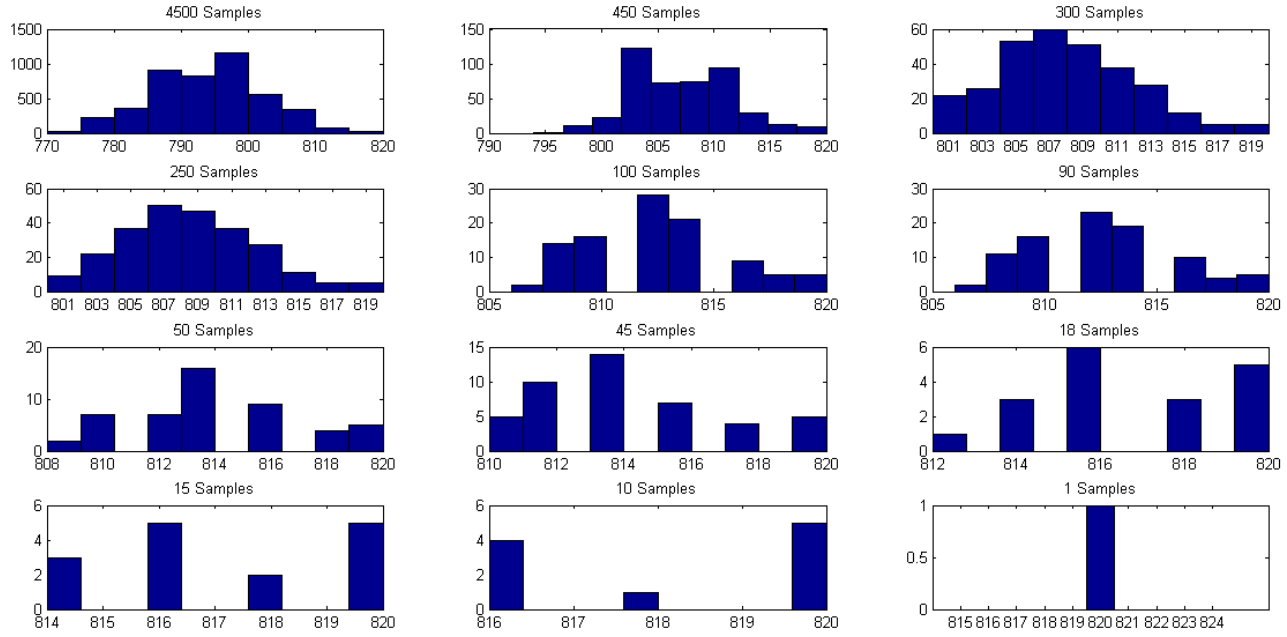

Figure 16 - Maxima block histograms (X axis means *execution time*, Y axis means *occurrences*). Reference: Author.

By analyzing histograms, it is possible to realize how the maxima block representability decreases as the samples number becomes lower. Under 90 samples, the histograms seem to be describing a completely

different algorithm than the one with more than 90 samples, since the number of repetitions is dramatically reduced. Therefore, pWCET outcome analysis must count on this characteristic.

After split samples in basic blocks and pick their maximum values to compound the maxima block, it is possible to apply EVT methodology. The Matlab function gevfit( ) receives an array containing the maxima block samples. This function returns three parameters calculated according to Generalized Extreme Value methodology, the parameters *location*, *scale* and *shape*. These parameter values for each maxima block size are displayed in Table VII. The pWCET outcome analysis must count that in some cases gevfit returned warning messages.

Table VI - Gevfit function response for different maxima block sizes.

Maxima Block Size	**Location (μ)**	**Scale (σ)**	**Shape (ξ)**
4500	791.488	8.211	-0.262
450	805.297	4.353	-0.193
300	807.309	3.903	-0.196
250	807.974	3.724	-0.188
100	811.163	2.985	-0.162
90	811.284	3.044	-0.174
50	813.091	3.078	-0.277
45	813.218	2.571	-0.124
18*	817.395	2.910	-1.117
15*	817.812	2.706	-1.237
10*	819.070	1.182	-1.271
1	820	0	0

*Gevfit returned the message: Maximum likelihood estimation did not converge. Function evaluation limit exceeded. Maximum likelihood converged to a boundary point of the parameter space. Confidence intervals and standard errors cannot be computed reliably.

Computing EVT distribution requires only the parameter values, thereafter the computed distribution is compared to the ECDF empirically defined. Figure 17 displays the quantile-quantile plots for all scenarios with different maxima block sizes.

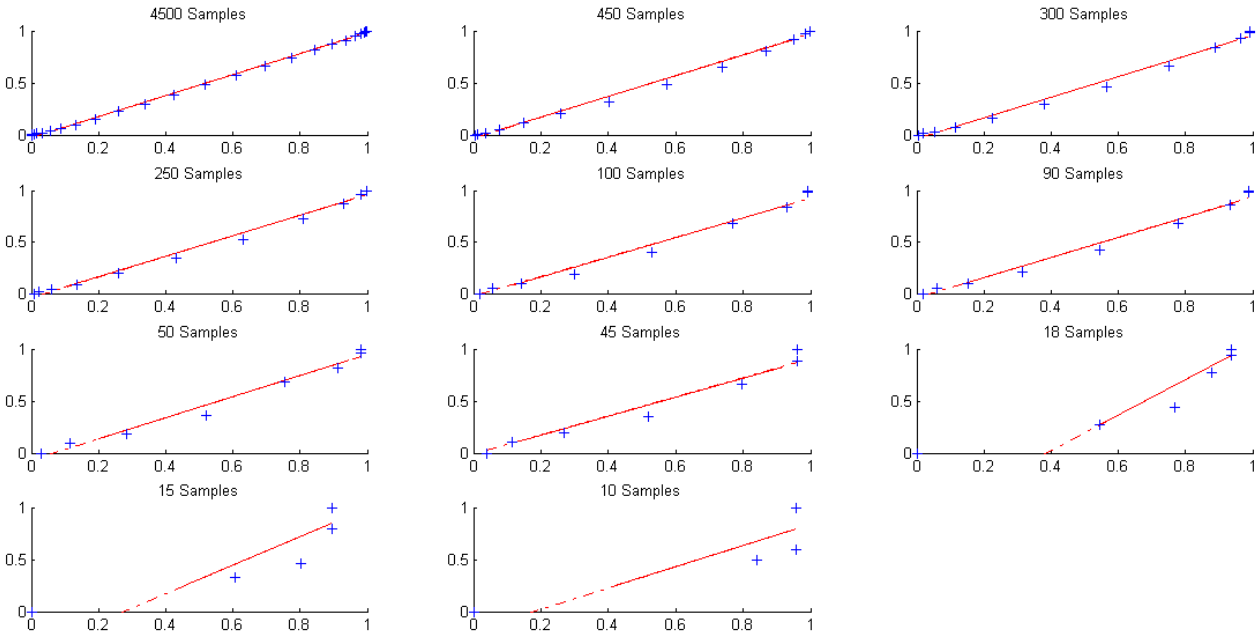

Figure 17 - Maxima block quantile-quantile plots. Reference: Author.

These QQ-plots represent the symmetry between EVT and ECDF distributions. The blue points are the real comparison whereas the red line is computed to represent these points. When all points in the plot can be described as X = Y, the distributions fitting is perfect. However, when the computed distribution does not represent the empirical data properly, there is a bad fitting, which is evidenced by the asymmetry, when the points are furthermost ($X \neq Y$). Keeping this in mind it is possible to analyze the QQ-plots and deduce the fitting comparison gets worse when the Maxima Block has less samples. Also, it points a bad fitting for 18 samples or less samples whereas the other scenarios seem close from symmetry.

Based on the QQ-plots it is important to point out that 4500 samples scenarios is the most symmetric, on the other hand, a good enough symmetry may be obtained with 100 samples, for example. Therefore, based only in these plots it is possible to conclude that there is no need to consider so many samples for computing the EVT distribution.

Like histograms, the quantile-quantile plots point that fitting the empirical (i.e., the MBPTA) and the EVT distributions is harder as long as the number of samples in maxima block is lower. It is important to enhance that it is not even possible to define a distribution for a unique value, as the one sample maxima block case. These differences between distributions are also shown in the following table:

Table VII - Difference between ECDF MBPTA (Block Maxima) and EVT distribution.

Maxima Block Size	**Max**	**Comparison (Max)**	**Average**	**Comparison (Average)**
4500	0.042	0%	0.019	0%
450	0.088	110%	0.035	84%
300	0.103	145%	0.041	116%
250	0.102	143%	0.041	116%

100	0.128	205%	0.053	179%
90	0.122	190%	0.053	179%
50	0.161	283%	0.05	163%
45	0.164	290%	0.062	226%
18	0.325	674%	0.103	442%
15	0.337	702%	0.12	532%
10	0.357	750%	0.163	758%

According to the table, the best fitting occurs for 4500 samples. In this scenario, 0.042 is the maximum difference between the computed EVT distribution and the empirical data collected as samples, when calculating the average difference, the outcome is less than a half of the maximum difference. Owing to this, the 4500 samples scenario is regarded as reference values for comparison to all scenarios in the columns Comparison (Max) and Comparison (Average).

Decreasing the number of samples in maxima block increases considerably the distributions differences. For instance, the average difference for 100 samples is 179% higher than the difference for 4500 samples, whereas the maximum difference value is 205% higher than the maximum for 4500. The 450 samples scenario, when compared to the 4500 samples case, points out that decreasing the Maxima Block Size in 90%, the difference between distributions increases 110%.

Finally, Table VIII shows the pWCET outcome for each Maxima Block case. The probability threshold value is 10^{-6} and the reference values for WCET, statically computed in 6.2.1, is 826 executed instructions:

Table VIII - Bubble Sort on Block Maxima approach.

Maxima Block Size	**pWCET(10^{-6})**	**Difference |pWCET – WCET|**	**Difference/WCET**
4500	818	8	0.97%
450	822	4	0.48%
300	823	3	0.36%
250	823	3	0.36%
100	824	2	0.24%
90	824	2	0.24%
50	823	3	0.36%
45	**826**	**0**	**0.00%**
18	820	6	0.73%
15	820	6	0.73%
10	820	6	0.73%
1	820	6	0.73%

According to the pWCET results, the higher difference between the WCET and the outcome occurs for Maxima Block size equal to 4500 samples. This difference decreases as long as the sample number decreases. For 45 samples the difference is zero, the found pWCET value is equal to the reference WCET.

Considering the histograms (Figure 16) and the QQ-plots (Figure 17), the scenarios with 90 or less samples do not satisfy the requirements described on literature for a reliable outcome. It is worthy to point that

gevfit function (Table VI) returned the following message for 18 or less samples: ‘Confidence intervals and standard errors cannot be computed reliably’.

While scenarios when the Maxima Block has small size seem clearly unreliable, the scenarios with the higher size of Maxima Block, 4500 samples, resulted in a pWCET of 818 executed instructions. It represents the largest difference between the computed pWCET and the reference WCET. Besides that, this outcome value means that there is a probability of 10^{-6} of the execution time exceeds 818, which seems accurate, acceptable and trustable even though this pWCET does not cover 0,9685% of all possible values (from 818 to 826).

The 45 samples scenario outcome, on the other hand, has no difference to the reference. Although it covers 100% of all possible values, it indicates that there is a probability of 10^{-6} of the execution time exceeds 826, which is not true. It is worthy to enhance that even when the outcome seems more accurate, the reliability still been the most imperative requirement in the analysis.

Considering the outcome values and the Block Maxima analysis, it suggests that the best results, with reliable outcomes and smaller differences, arise regarding 300 to 90 samples in analysis. In other words, regarding 2% to 7% of the total samples in EVT analysis with Block Maxima (BM) approach increases the outcome accuracy.

6.2.4 MBPTA: Pick Over Threshold Approach

Similarly to Basic Block Approach, this analysis with Pick Over Threshold regards 4500 samples amount. However, instead of splitting samples and defining blocks sizes, this approach defines threshold values to determine whether regards or not a given sample in analysis. Therefore, the number of samples under analysis depends on the chosen threshold value. The following table shows the threshold values and the analyzed samples amount for each case.

Table IX - Threshold values and respective number of samples. Reference: Author.

Threshold	770	780	790	800	805	810	815	820
Samples	4500	4352	3350	1330	442	167	22	5

This picking process select all samples whose value is equal or higher than the threshold. The first threshold value, 770, picks all 4500 samples, since it is the minimum value in the collected samples. On the other hand, the last threshold, 820, selects only five samples, once this is the maximum value collected on the samples.

The following image shows the histogram of the analyzed samples for each scenario:

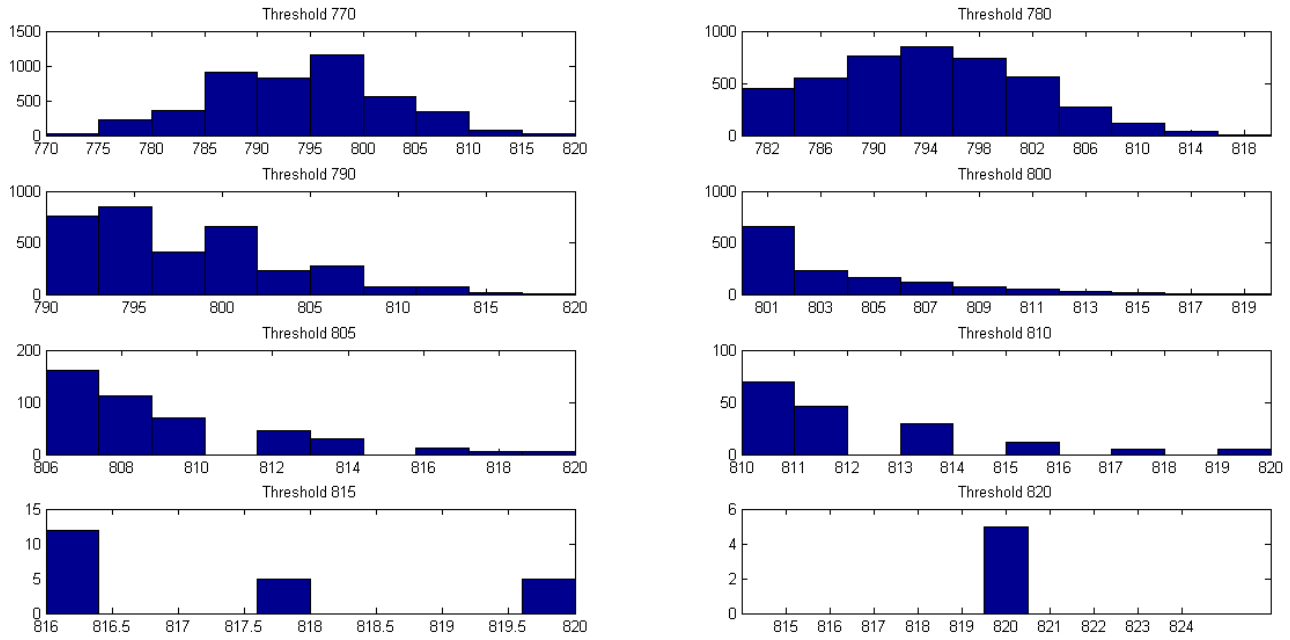

Figure 18 - Histograms for different threshold values (X axis means time, Y axis means occurences). Reference: Author.

As the Block Maxima approach, the Pick Over Threshold requires to converge values between the collected samples and an EVT computed distribution. However, instead of computing a distribution from GEV family, POT compute from Pareto Distribution family. Matlab provides gpfit() function for Pareto distributions, which is similar to gevfit. This function returns two parameters: Scale and Shape. The following table displays this parameter values for each case.

Table X - Pareto parameter values for different thresholds.

Threshold	**Scale** (σ)	**Shape** (ξ)
770*	$2.165\ 10^{-3}$	-2.6401
780*	$1.581\ 10^{-3}$	-1.9289
790*	$1.944\ 10^{-3}$	-2.3712
800*	$1.866\ 10^{-3}$	-2.2764
805*	$1.672\ 10^{-3}$	-2.0399
810*	$1.958\ 10^{-3}$	-2.3885
815*	$1.386\ 10^{-3}$	-1.6907

*Gpfit returned the message: Maximum likelihood has converged to a boundary point of the parameter space. Confidence intervals and standard errors cannot be computed reliably.

It is imperative to enhance that there is no good convergence for any scenario. Therefore, no one outcome is reliable. Also, it is not possible to determine a distribution with only one value, as the 820-threshold case. The following quantile-quantile plots confirm these bad convergences.

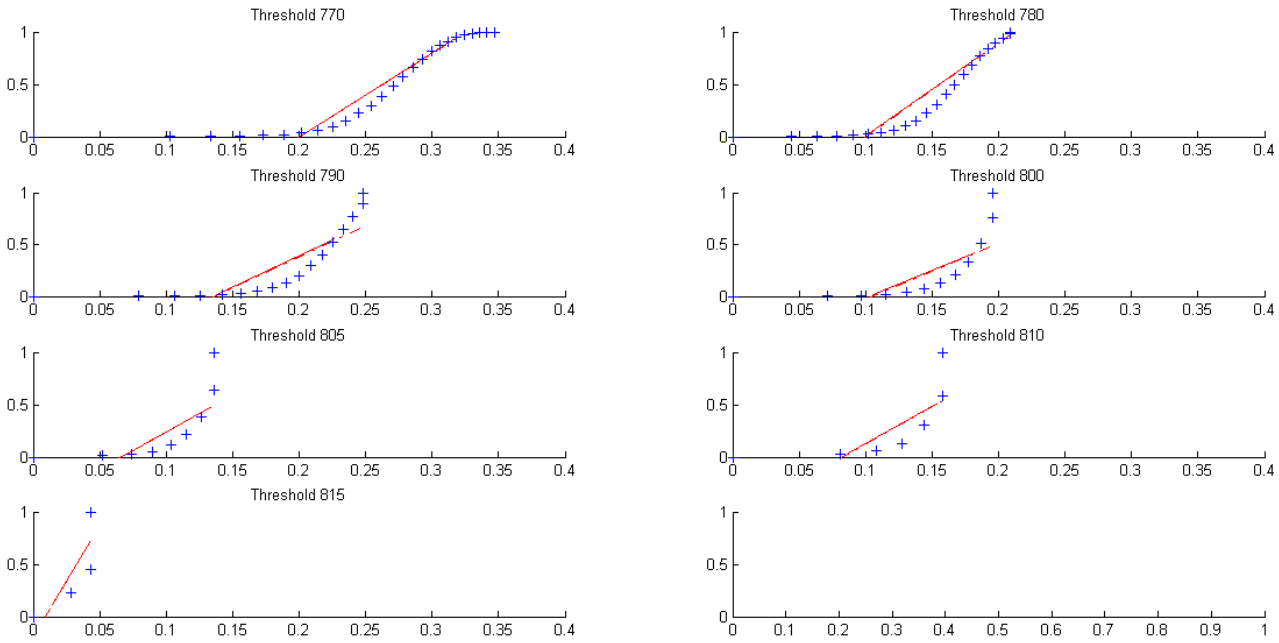

Figure 19 - Quantile-quantile plots for different threshold values. Reference: Author.

The quantile-quantile plots proves the large difference between empirical distributions and Pareto's. It is important to point that a good fitting shows a straight line (X = Y) from 0 to 1. However, no one of plots above not even reached 0.5 on the X axis. This lack of confidence is also evidenced on the outcomes, displayed on the following table:

Table XI - Bubble Sort on Pick Over Threshold approach.

Threshold	**pWCET(10^{-3})**	**Difference |pWCET – WCET|**
770	820	6
780	820	6
790	820	6
800	820	6
805	820	6
810	820	6
815	820	6

Unlike Block Maxima, the Peak Over Threshold analysis seems to be unreliable and useless according to histograms, QQ-plots and the gevfit result. However, the evidenced lack of confidence and invalid outcome may be caused, for example, by some unknown Matlab limitation, by some context characteristics as the chosen algorithm complexity and even by the collecting samples methodology. So, it is important to investigate and understand why POT doesn't work properly in this context. Future works can consider this case.

6.3 Finite Impulse Response Filter Case

Other context regarded in this work is the FIR Filter, which is employed in digital filters. This algorithm receives two parameters, N and K in Figure 20. The first is the number of samples and the second is the filter order. The N variable may assume values from 1 to 64, whereas the K variable may assume only the even values from 1 to 64. The figure below shows the analyzed assembly code, in this case N and K variables have values 8 and 16 respectively:

```
1  .data
2  N:          .word 8
3  K:          .word 16
4  SCALE:      .word 5
5  SCALE32:    .word 5
6
7  .text
8  li  $a0, 0x1001
9  sll $a0, $a0, 16
10 move  $a1, $a0
11 addi $a1, $a1, 0x08
12 move  $a2, $a0
13 addi $a2, $a2, 0x1c
14
15 lw  $s0, N
16 lw  $s1, K
17
18 sll $s0, $s0, 2
19 sub $s2, $s0, 4
20 add $s0, $s0, $a2
21 sll $s1, $s1, 2
22 add $a3, $a1, $s1
23 addiu $t5, $zero, 0x7FFF
24 addiu $t6, $zero, 0x8000
25
26 loop:
27 mult $zero, $zero
28
29 loopRe:
30 lh $t0, 0($a0)
31 lh $t1, 2($a0)
32 lh $t2, 0($a1)
33 lh $t3, 2($a1)
```

```
33 lh $t3, 2($a1)
34 madd $t0, $t2
35 msub $t1, $t3
36 lh $t0, 4($a0)
37 lh $t1, 6($a0)
38 lh $t2, 4($a1)
39 lh $t3, 6($a1)
40 madd $t0, $t2
41 msub $t1, $t3
42 addiu $a1, $a1, 8
43 bne $a1, $a3, loopRe
44
45 addiu $a0, $a0, 8
46
47 mflo $v0
48 mfhi $v1
49 sub $a0, $a0, $s1
50 sub $a1, $a1, $s1
51 mult $zero, $zero
52 srl $v0, $v0, 5
53 sll $t2, $v1, 5
54 or $v0, $v0, $t2
55 slt $t2, $t5, $v0
56 movn $v0, $t5, $t2
57 slt $t2, $v0, $t6
58 movn $v0, $t6, $t2
59 sh $v0, 0($a2)
60
61 loopIm:
62 lh $t0, 0($a0)
63 lh $t1, 2($a0)
64 lh $t2, 0($a1)
65 lh $t3, 2($a1)
```

```
66 madd $t1, $t2
67 madd $t0, $t3
68
69 lh $t0, 4($a0)
70 lh $t1, 6($a0)
71 lh $t2, 4($a1)
72 lh $t3, 6($a1)
73 madd $t1, $t2
74 madd $t0, $t3
75 addiu $a1, $a1, 8
76 bne $a1, $a3, loopIm
77
78 addiu $a0, $a0, 8
79 mflo $v0
80 mfhi $v1
81 sub $a0, $a0, $s2
82 sub $a1, $a1, $s1
83 addiu $a2, $a2, 4
84 srl $v0, $v0, 5
85 sll $t2, $v1, 5
86 or $v0, $v0, $t2
87 slt $t2, $t5, $v0
88 movn $v0, $t5, $t2
89 slt $t2, $v0, $t6
90 movn $v0, $t6, $t2
91 bne $a2, $s0, loop
92 sh $v0, -2($a2)
93 syscall
```

Figure 20 - Assembled code of the FIR Filter. Reference: Author.

The FIR Filter code has seven different basic blocks compounding the program. The following image shows the Control Flow Graph of this algorithm.

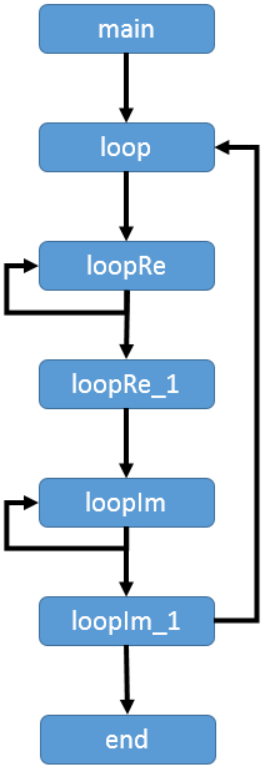

Figure 21 - CFG of FIR Filter code. Reference: Author.

6.3.1 Static Analysis

Similarly to the Bubble Sort Case, the static analysis applies IPET to the algorithm in order to compute the WCET. The APPENDIX A - IPET Example: Bubble Sort Case presents a detailed example of this method and all the involved computation. For FIR Filter case, IPET outcomes defines WCET as 59,223 (executed instructions). This high WCET estimation number is consequence of the loop repetition, since both inner loops upper bound is 1984 times.

6.3.2 MBPTA: Collecting Samples

This algorithm has two variables, N and K in Figure 20. The first is the number of samples and the second is the filter order. The N variable may assume values from 1 to 64, whereas the K variable may assume only the even values from 1 to 64 (32 different values). Considering the possible values each variable can assume, there are 32x64 possible cases, which means 2048 different cases.

Since it is feasible to collect 2048 samples, it is possible to consider two scenarios: A and B. In the scenario A, it collects one sample for each input possibility, which means that it analyzes 2048 samples at all. The scenario B allows input values repetition and, hence, may regard a higher number of samples under analysis.

Regarding unique samples, with no input value repetition, may affect the analysis, since the total of 2048 samples implies in less samples under analysis, shorter basic blocks and maxima block, which might not be enough to apply EVT techniques properly. Notice that a sample is unique due to its input values, N and K, and not due to its outcome. Therefore, the collection of unique samples may contain repeated outcome values, although it has no samples with repeated input values.

Scenario A has 2048 unique samples that means this scenario covers 100% of all possible cases. It might be useful to evaluate whether EVT's accuracy decreases when considering a shorter number of samples on analysis. Regarding all samples outcomes, the highest observed result value is 59223, which is according to the statically analysis result.

Scenario B has 5000 samples randomly collected with no control in input values. Thus, it might repeat some input values, although it may not use all possible values. The highest observer result value is 57431, below than the static worst-case result. Nevertheless, the total number of collected samples represents over than 244% of the required number to cover all cases. It might be useful to observe EVT results comparing both scenarios.

Since both scenarios have samples enough to cover all possible cases, it is not required to estimate the minimal number of samples as explained in 4.1.5. However, it must be regarded to define a safe probability threshold. The following table shows the CRPS analysis for scenario A, the computation regards the equation $CRPS = \sum_{i=0}^{+\infty}[fx(i) - fy(i)]^2$ according to 4.1.5. It starts considering 1024 samples ($N_{current}$), adding 64 (N_{delta}) in each round of comparison.

Table XII - Number of samples ($N_{current}$) on analysis (A). Reference: Author.

Rounds	Ncurrent	Current + Delta	Max Difference (CRPS)
1	1024	1088	1.93E-08
2	1088	1152	1.59E-08
3	1152	1216	1.30E-08
4	1216	1280	1.08E-08
5	1280	1344	9.04E-09
6	1344	1408	7.60E-09
7	1408	1472	6.45E-09
8	1472	1536	5.51E-09
9	1536	1600	4.72E-09
10	1600	1664	4.07E-09
11	1664	1728	3.53E-09
12	1728	1792	3.08E-09
13	1792	1856	2.69E-09
14	1856	1920	2.37E-09
15	1920	1984	2.09E-09
16	1984	2048	1.85E-09

The table shows that maximal difference decreases linearly throughout the analysis. The difference stands consistently less than 1.10^{-8} when $N_{current}$= 1280 samples and further. This value still the same even when increasing N_{delta} to 128. The same CRPS analysis, for scenario B, is shown in the following table. It starts with 3500 samples and adds 100 in each round.

Table XIII - Number of samples on analysis (B). Reference: Author.

Rounds	Ncurrent	Current + Delta	Max Difference (CRPS)
1	3500	3500 - 3600	6.30E-07
2	3600	3600 - 3700	2.42E-07
3	3700	3700 - 3800	7.53E-11
4	3800	3800 - 3900	1.49E-06
5	3900	3900 - 4000	1.71E-07
6	4000	4000 - 4100	1.47E-07
7	4100	4100 - 4200	2.56E-08
8	4200	4200 - 4300	8.47E-08
9	4300	4300 - 4400	2.28E-08
10	4400	4400 - 4500	4.00E-12
11	4500	4500 - 4600	5.95E-06
12	4600	4600 - 4700	5.88E-07
13	4700	4700 - 4800	5.02E-08
14	4800	4800 - 4900	1.12E-07
15	4900	4900 - 5000	2.82E-08

Unlike the scenario A, the maximal difference does not decrease linearly. Actually, Table XIII depicts some values above 1.10^{-6} (3800 and 4500 samples). Although having almost 250% more samples than A, the scenario B has a less reliable sample collection. Nevertheless, increasing the samples amount may facilitate statistical analysis, but may affect reliability as side effect. In addition, it is important to enhance the difference between these scenarios are the number of samples and the control of input values, which may incur the reliability issue.

6.3.3 MBPTA: Block Maxima Approach

Since the literature do not specifies the criteria for splitting samples in basic block, in this work it divides samples in basic blocks according the order they were collected. For instance, regarding blocks with 100 samples, the first basic block would contain the 1st to 100th samples.

This Block Maxima analysis regards 2048 samples for scenario A (Table XIV) and 5000 samples for scenario B (Table XV):

Table XIV - Different block sizes for Basic Blocks and Maxima Block (A). Reference: Author.

Size of basic Blocks	1	10	20	25	40	50	100	200	250	500	1000	5000
Number of Basic Blocks	5000	500	250	200	125	100	50	25	20	10	5	1

Size of Maxima Block	5000	500	250	200	125	100	50	25	20	10	5	1

Table XV - Different block sizes for Basic Blocks and Maxima Block (B). Reference: Author.

Size of basic Blocks	1	2	4	8	16	32	64	128	256	512
Number of Basic Blocks	2048	1024	512	256	128	64	32	16	8	4
Size of Maxima Block	2048	1024	512	256	128	64	32	16	8	4

Literature affirms that basic blocks must contain enough samples to represent the execution behavior properly. Figure 22 and Figure 23 show histograms for maxima blocks with different sizes.

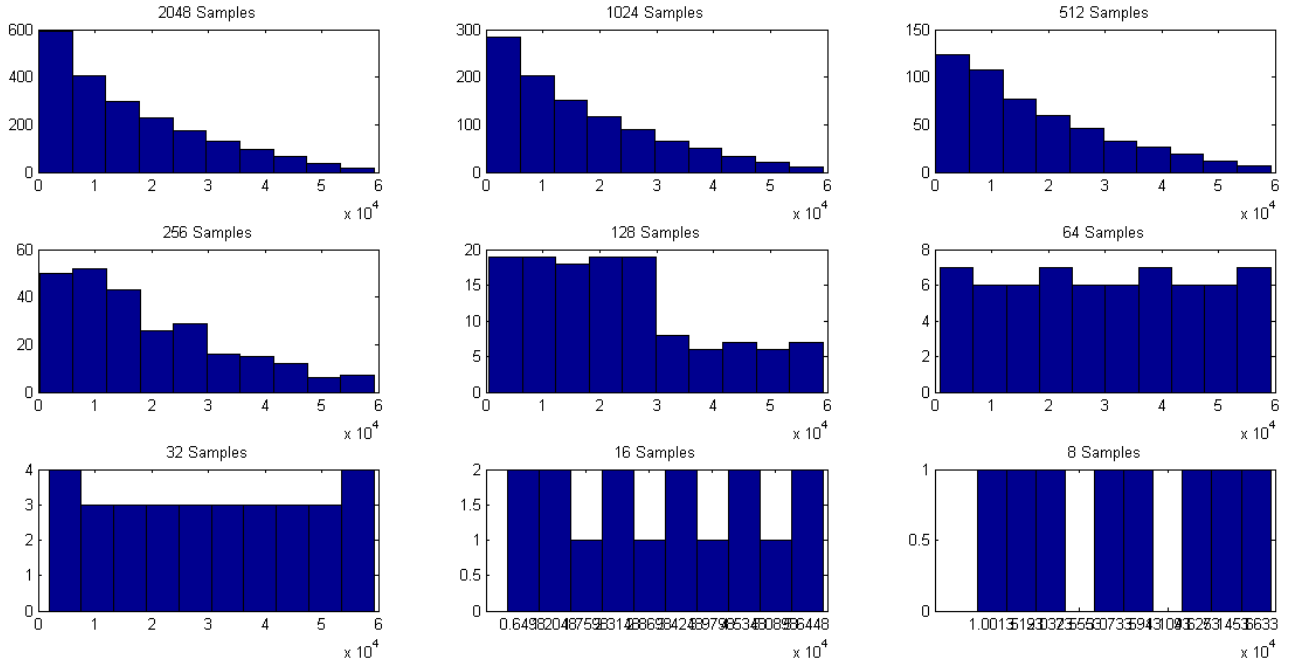

Figure 22 - Maxima blocks histograms for A (X axis means time, Y axis means occurences). Reference: Author.

According to these histograms, the maxima block representability decrease as the samples number becomes lower. From 128 samples to below, the histograms do not represent the execution behavior properly, since it has less information. For this reason, the image disregard the histogram related to the maxima block with four samples. These histograms characteristics likely affect the pWCET outcome.

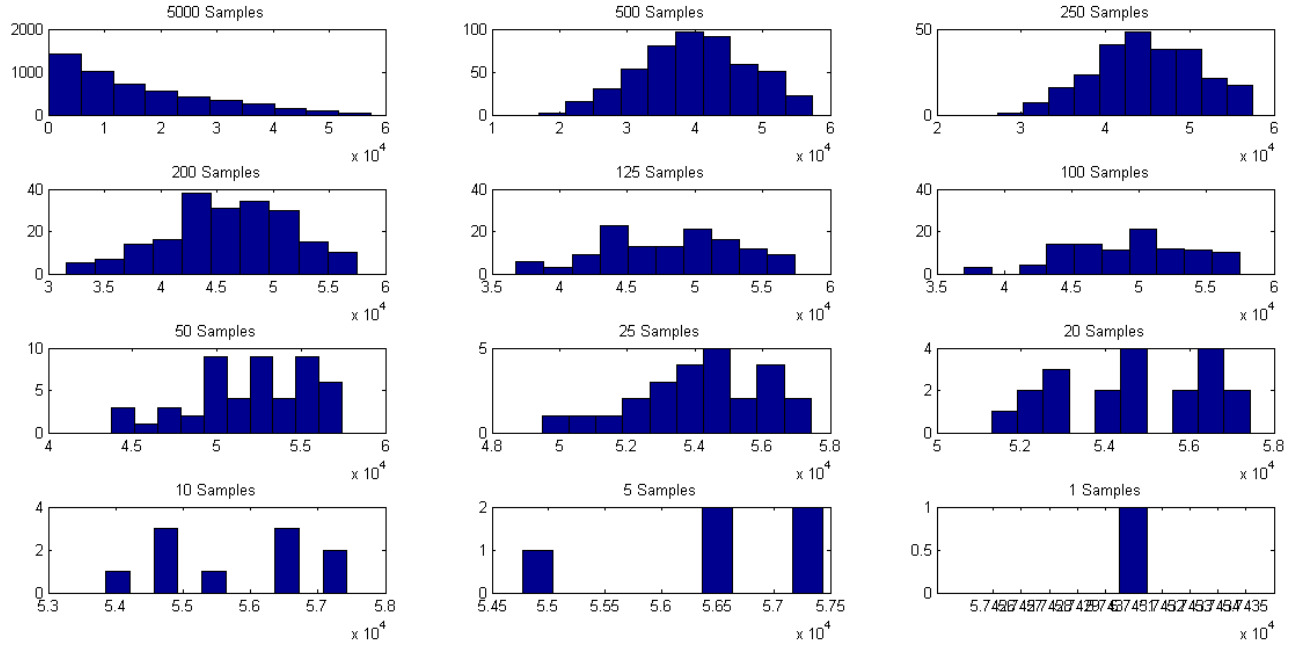

Figure 23 - Maxima blocks histograms for B (X axis means time, Y axis means occurences). Reference: Author.

Similarly, to the histograms in last image, the maxima block representability decrease when regarding less samples. On the other hand, unlike the scenario A, from 125 samples to below, the histograms do not represent the execution behavior properly, since it has less information. These histograms characteristics likely affect the pWCET outcome.

After that, it is possible to apply EVT methodology, using Matlab. The function gevfit( ) receives an array containing the maxima block samples and returns three parameters calculated according to Generalized Extreme Value methodology. These parameters are location, scale and shape and the following tables show their values for each maxima block size regarding A and B. The pWCET outcome analysis must count that in some cases gevfit returned warning messages.

Table XVI - Gevfit function response for different maxima block sizes (A).

Maxima Block Size	**Location (μ)**	**Scale (σ)**	**Shape (ξ)**
2048	8456.389	8003.180	0.33435
1024	8896.437	8246.997	0.31210
512	9793.856	8753.928	0.26727
256	11633.035	9818.336	0.18123
128	15444.609	12050.860	0.02751
64	25382.261	17986.367	-0.44034
32	25861.017	17993.445	-0.44201
16	26852.916	18023.875	-0.44890
8	29014.667	18195.410	-0.48175
4*	39645.564	25626.445	-1.30898

Table XVII - Gevfit function response for different maxima block sizes (B).

Maxima Block Size	**Location (μ)**	**Scale (σ)**	**Shape (ξ)**
5000	8279.365	7800.338	0.32319
500	37499.584	8320.073	-0.36310
250	42833.202	6346.083	-0.36504
200	44311.495	5853.875	-0.37807
125	46611.243	5182.819	-0.40576
100	48083.872	4746.951	-0.44483
50	51181.760	3776.319	-0.55532
25	53787.696	2282.692	-0.55729
20	54270.530	2012.918	-0.55511
10*	55844.692	1753.467	-1.10538
5*	56553.402	1150.2383	-1.31067
1*	57431	0	0

*Gevfit returned the message: Maximum likelihood estimation did not converge. Function evaluation limit exceeded. Maximum likelihood has converged to a boundary point of the parameter space. Confidence intervals and standard errors cannot be computed reliably.

Although EVT distribution requires only these parameters values on its definition, comparing the defined distribution and the ECDF (empirically defined) might be useful to observe how these distributions fit. Figure 24 and Figure 25 display the quantile-quantile plots for different maxima block sizes in A and B.

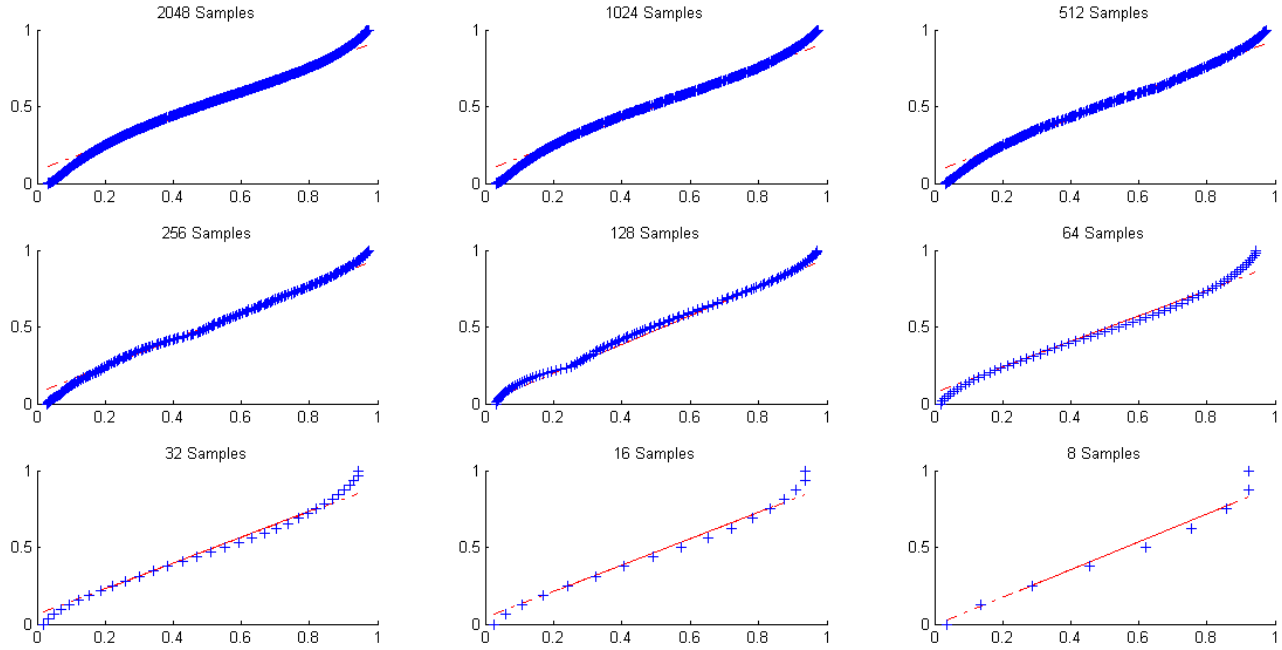

Figure 24 - Maxima block quantile-quantile plots (A). Font: Author.

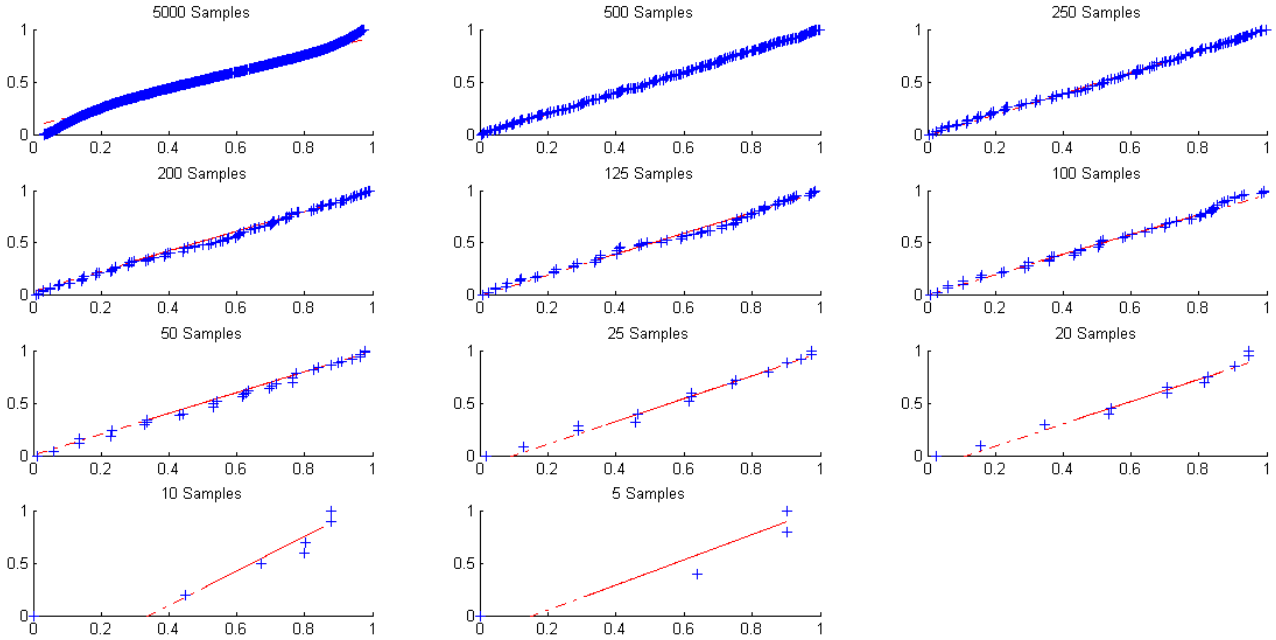

Figure 25 - Maxima block quantile-quantile plots (B). Reference: Author.

Like histograms, the quantile-quantile plots point that when regarding a lower number of samples in maxima block, the analysis is affected. In this case, it influences how the empirical and the EVT distributions fit with small samples amount. It is important to enhance that it is not even possible to define a distribution for a unique value, as the one sample maxima block case.

Besides the quantile-quantile plots, the maximal difference between distributions might be a useful and indicating information. The following tables display the differences between these distributions:

Table XVIII - Differences between ECDF MBPTA(Block Maxima) and EVT distribution (A).

Maxima Block Size	**Max**	**Comparison (Max)**	**Average**	**Comparison (Average)**
2048	0.058	100%	0.031	100%
1024	0.056	97%	0.030	97%
512	0.053	91%	0.028	90%
256	0.050	86%	0.023	74%
128	0.041	71%	0.019	61%
64	0.072	124%	0.037	119%
32	0.080	138%	0.039	126%
16	0.096	166%	0.045	145%
8	0.131	226%	0.072	232%
4	0.304	524%	0.160	516%

Table XIX - Differences between ECDF MBPTA(Block Maxima) and EVT distribution (B).

Maxima Block Size	Max	Comparison (Max)	Average	Comparison (Average)
5000	0.059	100%	0.031	100%
500	0.026	44%	0.007	23%
250	0.040	68%	0.012	39%
200	0.045	76%	0.013	42%
125	0.073	124%	0.022	71%
100	0.054	92%	0.024	77%
50	0.070	119%	0.026	84%
25	0.137	232%	0.046	148%
20	0.135	229%	0.068	219%
10	0.250	424%	0.124	400%
5	0.238	403%	0.110	355%
1	-	-	-	-

As quantile-quantile plots, this difference values also show the disparity due to different maxima block sizes. Decreasing the number of samples in maxima block increases considerably the distributions differences. Regarding A, the lowest difference is 0.041, which occurs when Maxima block has 128 samples collected from 128 basic blocks, each basic block have 16 samples. Comparing all maxima block sizes in this scenario, 128 seems ideal.

Besides that, differences in scenario B does not seems linear. Although, the lowest difference value is 0.026, when Maxima has 500 samples collected from 500 basic blocks, each basic block having 10 samples. Since the lowest difference value is 0.041 in A, the lowest difference in B is considerably better. Furthermore, the 250 samples case shows the difference value 0.040, which is lower than A also.

It is worthy to enhance that the better options according to Table XVIII and Table XIX are the following Maxima block size: 128, 200 and 250 samples. That means the better options have the following basic blocks size: 10, 16 and 20.

Finally, Table XX and Table XXI show the pWCET outcome for each maxima case. The probability threshold value is 10^{-8} for A and 10^{-6} for B, the reference value, the statically defined WCET, is 59223 executed instructions, as detailed in 6.3.1.

Table XX - FIR Filter on Block Maxima approach, scenario A.

Maxima Block Size	pWCET(10^{-8})	Difference |pWCET – WCET|	Difference/WCET
2048	225527	166304	280.81%
1024	210630	151407	255.66%
512	184534	125311	211.59%
256	146892	87669	148.03%
128	107117	47894	80.87%
64	64277	5054	8.53%
32	64647	5424	9.16%
16	65196	5973	10.09%
8	65429	6206	10.48%

4	**59220**	**3**	**0.01%**

According to Table XX, as larger the maxima block, the worse the pWCET estimation is. The difference between the pWCET(10^{-8}) and the static WCET is 166304 for 2048 samples. On the other hand, when Maxima block has only 4 samples, the difference is 3. In addition, it is important to point that Gevfit function returned an error message for the Maxima with 4 samples (Table XVI). Thus, the lowest outcome difference is 5054 and occurs for 64 samples.

The histogram comparison (Figure 22) shows that maxima block represents the algorithm execution behavior when having 128 samples or more. Regarding this assumption, the best outcome difference is 47894, for 128 samples. Table XVIIITable XX also shows that 128 samples have the best fitting between the EVT defined and the empirical distributions, the maximal difference is 0.041 and the average difference is 0.019. When comparing the 64 samples case, the maximal difference is 75.6% higher and the average difference is 94.7% higher than 128 samples case. Therefore, different analysis pointed the 128 samples case as more reliable and assured. However, the EVT results pointed the 64 samples as more accurate.

Considering this cases, it is possible to deduce that the best samples amount in Maxima block for this analysis is among 3% and 6% of the total collected samples.

The following table shows the EVT outcomes for scenario B:

Table XXI - FIR Filter on Block Maxima approach, scenario B.

Maxima Block Size	**pWCET(10^{-6})**	**Difference |pWCET – WCET|**	**Difference/WCET**
5000	209131	149908	253.12%
500	58547	676	1.14%
250	58820	403	0.68%
200	58658	565	0.95%
125	58608	615	1.04%
100	58261	962	1.62%
50	57837	1386	2.34%
25	57796	1427	2.41%
20	57819	1404	2.37%
10	57430	1793	3.03%
5	57430	1793	3.03%
1	-	-	

According to Table XXI, the worst outcome occurs for 5000 samples, when the difference between the EVT distribution and the ECDF is 149908. Despite that, the best-obtained result value is 58547, which means a difference of 676, for 500 samples. The difference worsens as the Maxima block has fewer samples.

The histograms analysis (Figure 23) depicts that maxima block represents the algorithm execution behavior with 200 samples or more. Besides that, the Gevfit function returned an error message for 10, 5 and 1 sample (Table XVII), so its outcomes might be ignored.

The fitting analysis (Table XIX) also pointed the 500 samples case as the best one. The maximal difference is 0.026 and the average difference is around 0.007. The 250 samples case is the second best. However, when comparing it, the maximal difference is 53.8% higher and the average difference is 71.4% higher than the 500 samples case. Thus, all analysis and the EVT outcomes point 500 samples case as more reliable and accurate.

Regarding these cases, by deduction, the best samples amount in Maxima block for this analysis is among 5% and 10% of the total collected samples.

6.3.4 MBPTA: Pick Over Threshold Approach

Like the Basic Block Approach, this Pick Over Threshold (POT) approach analyzes two scenarios: (A), only unique samples, and (B), repeated samples. Notice that there is an input control when collecting samples in A, for this cause it executes no input values twice. Besides that, A has 2048 samples and 100% of coverage. In other hand, B has 5000 samples but there is no assurance of coverage since it might repeat some input values.

POT selects samples based on a threshold value. The number of samples under analysis depends on the chosen threshold value. Furthermore, it regards only samples that have values above the threshold value. The collected samples values in A vary from 80 to 59223, in B the values vary from 80 to 57431.

The following table depicts the regarded thresholds in this approach. In addition, the image shows the number of samples under analysis. In other words, the figure shows how many samples have values above the given threshold:

Table XXII - Threshold values and number of samples for scenarios A and B. Font: Author.

Threshold	80	14500	29000	36250	43500	50750	54375	58000
Samples (A)	2048	906	365	204	94	30	12	2
Samples (B)	5000	2149	839	443	183	48	17	0

The lowest threshold (80) ensure the total coverage of samples for both cases, which means all samples under analysis. Besides that, the highest threshold (58000) filters only two samples for A and no samples for scenario B. Figure 26 displays the POT histograms for scenario A whereas Figure 27 shows the histograms for scenario B.

According to Figure 26, the histograms from threshold 80 to threshold 50750 represent the same executing behavior. In other hand, thresholds 54375 and 50000 clearly have no information to represent the algorithm executions properly.

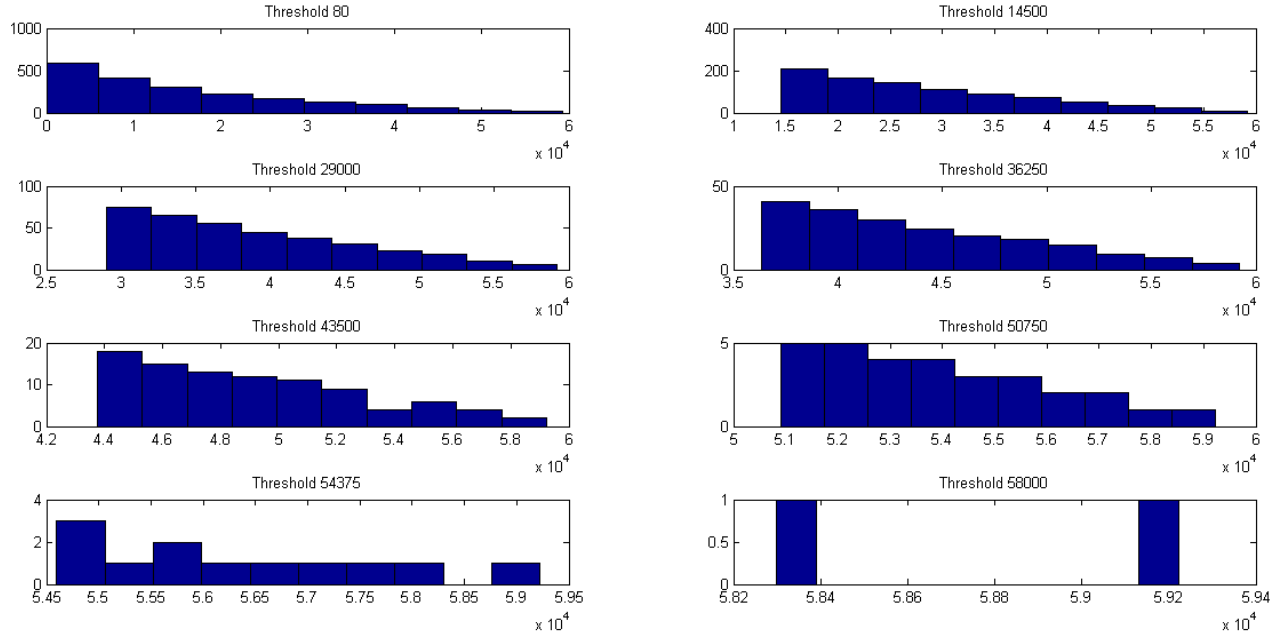

Figure 26 - Peak Over Threshold histograms for A (X axis means time, Y axis means occurences). Reference: Author.

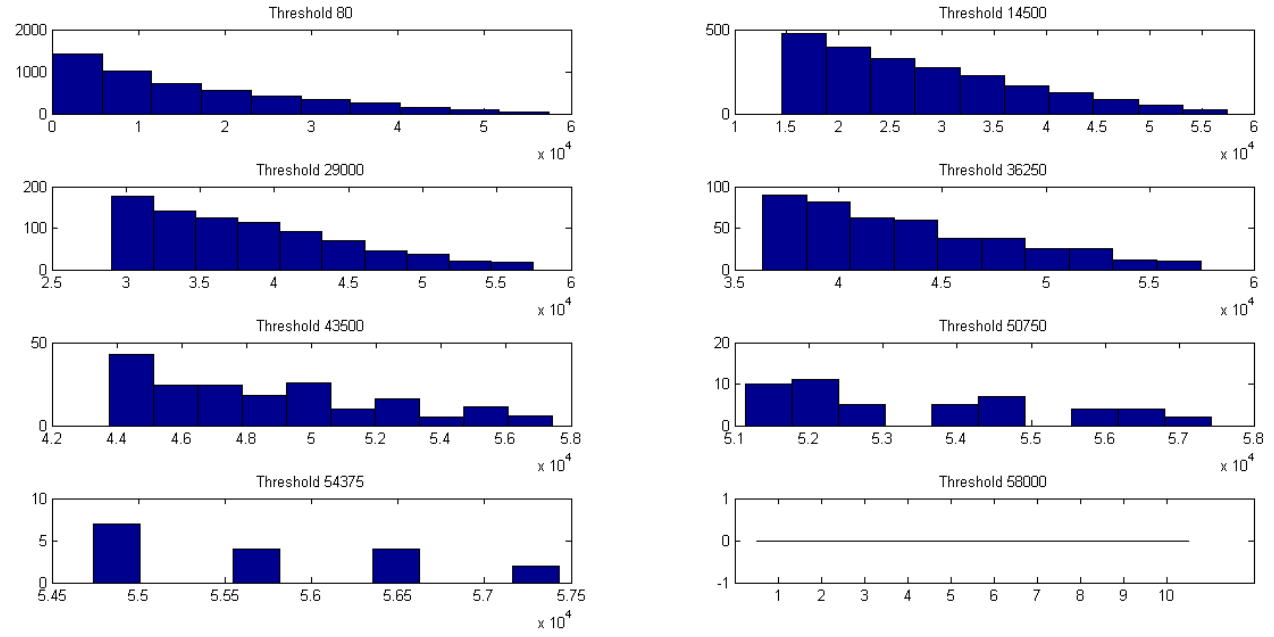

Figure 27 - Peak Over Threshold histograms for B (X axis means time, Y axis means occurences). Reference: Author.

According to Figure 27, histograms point that cases with threshold 43500 and higher might not be reliable. Furthermore, the threshold 58000 case has no sample.

As the block maxima approach, the Pick Over Threshold requires converging values between the collected samples and Pareto Distribution family. Instead of gevfit, Matlab afford gpfit( ) function for Pareto

distributions. This function find the proper Pareto distribution that best fits the given samples, also it returns two parameter Scale and Shape. The table below displays this parameter values for each case for both scenarios:

Table XXIII - Pareto parameter values for different thresholds.

Threshold	**Scenario A**		**Scenario B**	
	Scale (σ)	**Shape** (ξ)	**Scale** (σ)	**Shape** (ξ)
80	20907.891	-0.29750	20374.667	-0.30330
14500*	43819.635	-0.73930	43063.200	-0.74930
29000*	68855.758	-1.16270	66839.168	-1.16380
36250*	81848.796	-1.38200	85321.420	-1.48560
43500*	90660.376	-1.53080	100991.401	-1.75850
50750*	122908.724	-2.07540	104063.626	-1.81200
54375*	95431.500	-1.61140	99566.584	-1.73370
58000*	113501.888	-1.91650	-	-

*Gpfit returned the message: Maximum likelihood has converged to a boundary point of the parameter space. Confidence intervals and standard errors cannot be computed reliably.

It is important to enhance that there is only one good convergence, since the function returned error message for every cases except the first. Therefore, these outcomes must not be reliable. Further, the quantile-quantile plots (Figure 28 and Figure 29) confirm that. Regarding these images, there is a good fitting between the defined Pareto distribution and the empirical only when regarding all samples, in threshold 80 case. It happens for both scenarios A and B.

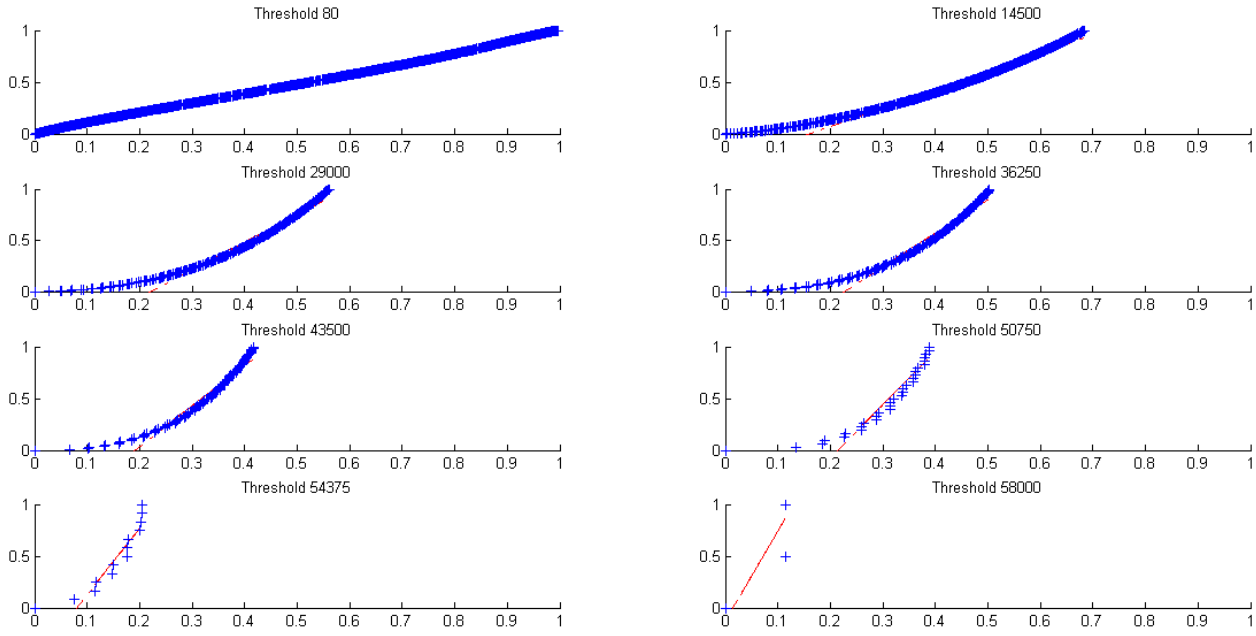

Figure 28 - Quantile-quantile plots for different threshold values (A). Font: Author.

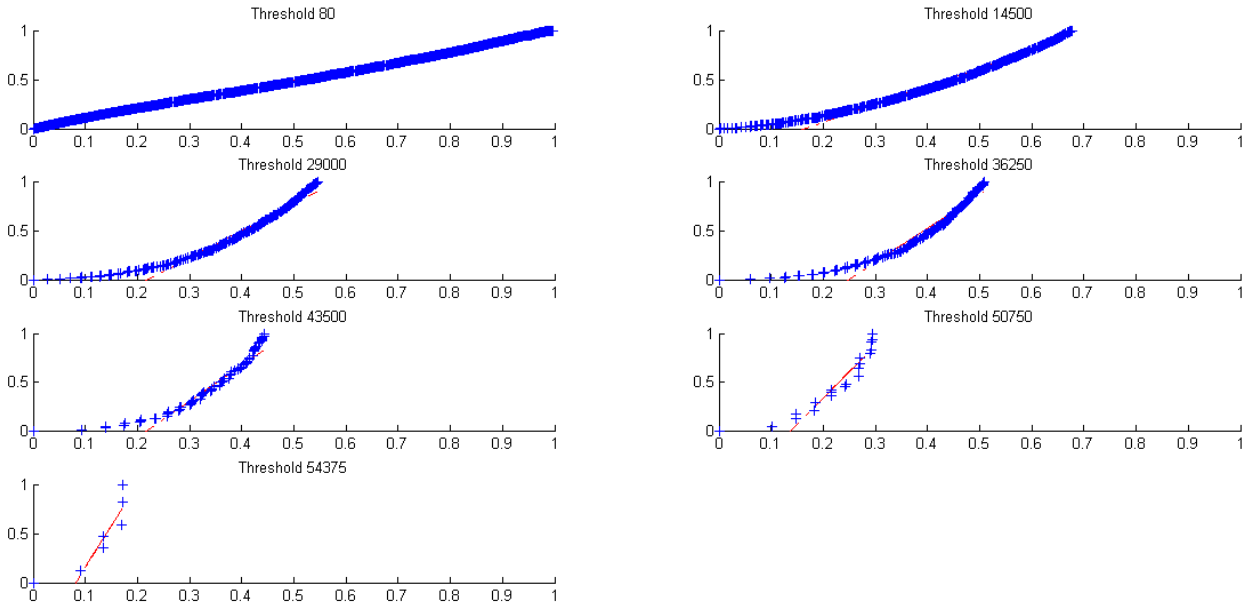

Figure 29 - Quantile-quantile plots for different threshold values (B). Font: Author.

As the quantile-quantile plots point, there are considerable differences between the empirical distributions and those defined by Pareto family model. The following tables show the maximal and the average difference for every case regarding scenarios A and B.

Table XXIV - Differences between ECDF MBPTA (Peak Over Threshold) and EVT distribution (A).

Threshold	Max	Comparison (Max)	Average	Comparison (Average)
80	3.3E-02	100%	1.0E-02	100%
14500	6.5E-02	198%	-7.0E-02	-687%
29000	1.1E-01	336%	-1.0E-01	-1032%
36250	1.2E-01	354%	-1.3E-01	-1302%
43500	9.6E-02	292%	-1.9E-01	-1883%
50750	1.2E-01	360%	-2.0E-01	-1957%
54375	1.3E-10	0%	-3.5E-01	-3449%
58000	4.7E-09	0%	-4.2E-01	-4173%

Table XXV - Differences between ECDF MBPTA (Peak Over Threshold) and EVT distribution (B).

Threshold	Max	Comparison (Max)	Average	Comparison (Average)
80	3.2E-02	100%	9.9E-03	100%
14500	7.5E-02	235%	-7.4E-02	-755%
29000	1.1E-01	348%	-1.1E-01	-1098%
36250	1.4E-01	438%	-1.2E-01	-1176%
43500	1.2E-01	376%	-1.5E-01	-1572%
50750	5.9E-02	185%	-2.8E-01	-2855%
54375	6.3E-10	0%	-3.5E-01	-3596%
58000	-	-	-	-

Like the quantile-quantile plots, these differences values show the disparity due to different threshold values. Decreasing the number of samples in analysis increases considerably the distributions differences. In both scenarios, the threshold 80 seems the best case. It has a maximal difference around 0.03 and an average difference around 0.01.

Considering these cases, it is possible to deduce that considering all samples is the best option for Peak Over Threshold approach. Since the most reliable cases regards 100% of the total collected samples. Table XXVI shows the EVT outcomes for scenario A, besides, Table XXVII depicts the EVT outcomes for scenario B. The probability threshold value is 10^{-8} for A and 10^{-6} for B, the reference value, the statically defined WCET, is 59223 executed instructions, as detailed in 6.3.1.

Table XXVI - FIR Filter on Peak Over Threshold approach, scenario A.

Threshold	**pWCET(10^{-8})**	**Difference |pWCET – WCET|**	**Difference/WCET**
80	61273	2050	3.46%
14500	58912	311	0.53%
29000	59203	20	0.03%
36250	59218	5	0.01%
43500	59221	2	0.003%
50750	59222	1	0.002%
54375	59222	1	0.002%
58000	59222	1	0.002%

Table XXVII - FIR Filter on Peak Over Threshold approach, scenario B.

Threshold	**pWCET(10^{-6})**	**Difference |pWCET – WCET|**	**Difference/WCET**
80	58915	308	0.52%
14500	57143	2080	3.51%
29000	57412	1811	3.06%
36250	57428	1795	3.03%
43500	57430	1793	3.03%
50750	57430	1793	3.03%
54375	57430	1793	3.03%
58000	-	-	

According to Table XXVI the threshold 80 case has the less accurate result and the outcomes value is more accurate when analyzing fewer sample. Therefore, for scenario A the most reliable outcome according other analysis is also the worst in accuracy according the results. It is important to enhance those samples in A represents 100% of coverage of all possible results. It might affect mainly with higher thresholds. The threshold 58000 case, for instance, has only two samples and one is the worst-case that is 59223. In addition, it is worthy to point that 61273 is not accurate but it is safe for worst-case, since its largest than 100% of possible values.

In scenario B, unlike A, the most accurate result is also the most reliable. The threshold 80 case has the small difference between the outcome and the statically defined worst-case. Furthermore, other analysis also pointed this case as most trustable.

It is important to enhance that 58915 is smaller than 59223. However, when considering every possible cases, there is only one case largest than that value, which is the WCET 59223. Therefore, 58915 covers 2047 of 2048 cases, that means a hit probability of 99.951%. In other words, this result fails for 1 of 2048 cases, that means a failing probability of 0.000488, which is slower than the probability threshold (10^{-3}) regarded on analysis.

6.4 Comparison

This section introduced two different contexts on which EVT is applied to estimate the pWCET, both consider the same processor platform (MIPS) but different algorithms: Bubble Sort and Finite Impulse Response (FIR) Filter. For both contexts the minimum number of samples is calculated according to CRPS and two approaches for filtering samples are applied: BM and POT.

In Bubble Sort case, the analysis with BM approach presents reliable and accurate outcomes when Maxima Block has 90 or more samples. Furthermore, the bests scenarios are respectively 90 and 100 samples, which evidenced the difference between the pWCET and the reference value, only 0.24%. On the other hand, analyzing with POT approach does not succeed in the same way. In this case, no scenario has reliable outcomes, since Matlab can't compute the distribution properly.

In FIR filter case, the analysis considers two scenarios when collecting samples: controlling input values (scenario A) and no controlling input values (scenario B).

For scenario A, when applying BM, the reliable outcomes are obtained for Maxima Blocks with 128 or more samples. However, these outcomes are significantly inaccurate, some of them estimate more than 250% when compared to the reference WCET. The best reliable pWCET estimation is evidenced for Maxima Block with 128 samples, nevertheless, this outcome is 80.87% higher than the reference value, which means this approach is not so efficient in this case. When considering outcomes for POT approach, it also has several unreliable scenarios which can't be calculated by Matlab. However there is one reliable scenario, when the threshold is 80. In this scenario the estimation is 3.46% higher than the reference WCET, which means that POT may be better for this context or even when controlling the input data.

For scenario B, in BM approach, scenarios with Maxima Block with 200 or more samples generated reliable outcome. Although the 5000 samples case estimates a pWCET 250% higher than the reference, the other cases are considerable accurate. The best result is found for Maxima Block with 250 samples, in this scenario the difference between the pWCET and the reference WCET is only 0.68%. For POT approach, similarly to A, only the threshold 80 computed reliable outcomes, on the other hand, for B the result is even more accurate. The difference between the calculated pWCET and the reference WCET in this scenario is only 0.52%. The following table show a comparison among the different analysis with BM and POT approaches:

Table XXVIII – Comparing outcomes from BM and POT approaches.

	Block Maxima	Peak Over Threshold
Bubble Sort	Reliable outcomes Small difference is 0.24%	No reliable outcomes
FIR Filter (A)	Few reliable outcomes Small difference is 80.87%	Only one reliable outcome Difference is 3.46%
FIR Filter (B)	Reliable outcomes Small difference is 0.68%	Only one reliable outcome Difference is 0.52%

Based on this table it is possible to conclude that POT is considerably more intricate than BM, since only two scenarios resulted in reliable information. On the other hand, BM seems simpler, and more accurate than POT. Also, it is worthy to point that the input data control for collecting samples may prejudice BM accuracy significantly whereas POT accuracy is lightly affected.

7 Conclusion

This work presented a methodology to analyze pWCET estimates of hard real time systems and pointed out the process limitations related to that. With this purpose, it regarded two algorithms: Bubble Sort and FIR Filter. Furthermore, this work used the MIPS architecture processor, single core, to assemble and run the selected algorithms.

Firstly, this methodology statically computed the WCET of both algorithms. After that, it applied the Measured Based Probabilistic Timing Analysis (MBPTA) approach to re-compute the WCET with different parameter values. This methodology used MARS, a MIPS simulator, to collect MBPTA samples. At the end, both outcomes, the one provided by statically computing and the one obtained by the MBPTA, were compared.

It is worth noting that the number of samples and their empirical distribution affect the probability threshold. For this reason, the present work considered the probability according to CRPS calculation and not the standard for avionic systems (10^{-9}).

The methodology applied the Extreme Value Theory (EVT) in the sample sets following the Block Maxima approach, comparing the outcomes by different input parameters. For the Bubble Sort analysis, the most reliable and accurate outcomes have a Maxima block from 2% to 7% of the total samples. Presenting only 0.24% of difference between the pWCET and the reference value.

On the other hand, for FIR Filter analysis controlling input data, Block Maxima computed only one reliable but not accurate outcome, it differs 80.87% when compared to the reference. When the analysis does not control the input data, the reliability and accuracy get better, the difference between the best outcome and the reference value is 0.68%. Further, FIR Filter analysis indicated that from 3% to 10% of all samples, the Maxima block provided the best outcomes.

This methodology also applied EVT following the Peak Over Threshold (POT) approach, diversifying the input parameters. The outcomes in Bubble Sort pointed POT as irrelevant and unreliable, all cases had a bad calculation and no case reached a safe fitting behavior. This work doesn't investigate what causes this problem, letting this topic as a future work.

Besides that, POT analysis depicted accurate and reliable outcome values for FIR Filter, especially when collecting samples with no input control. This scenario pointed that, as more selected samples in analysis, more reliable and accurate are the results, it computed a value with 0.52% of difference when compared to reference value. In addition, when collecting unique samples, with input control, accuracy and reliability slightly decreased, the best outcome had a difference of 3.46% between the reference value and the result.

Based on this work EVT is a useful and trustable technique to define pWCET. Block Maxima technique seems easier and more reliable since both algorithms had proper outcome values. On the other hand, the large number of samples required to feed a good Maxima Block is an important limitation to point. Further, this work does not investigate whether sample ordering when creating basic blocks affect or not the result.

On the other hand, Peak Over Threshold seems more complex and limited than Block Maxima. This technique presented no reliable outcome for Bubble Sort study-case, further, this work could not point what is the cause. Therefore, it is an important problem to point as limitation. However, POT produced reliable and accurate results for FIR Filter case, even for analysis controlling input data, when Block Maxima failed.

8 Future Work

As future work, the following topics can be addressed in order to extend and explore the proposed work, they are:

- Investigate why Peak Over Threshold failed for Bubble Sort case. The following questions may be a good start:
  - Are there limitations in Matlab functions for EVT? How they could affect the analysis?
  - Is the CRPS computation applicable for all scenarios?
  - Considering more samples can fix the problem?
- Apply this work methodology for different algorithms, e.g. Selection Sort, Merge Sort, and heapsort, in order to assure whether the algorithm complexity affects EVT reliability. Also, compare Block Maxima and Peak Over Threshold accuracy.
- Inspect whether the samples ordering affect or not Block Maxima technique.

References

ABELLA, Jaume et al. On the Comparison of Deterministic and Probabilistic WCET Estimation Techniques. **2014 26th Euromicro Conference On Real-time Systems**, Madrid, Spain, v. 1, n. 26, p.266-275, jul. 2014. IEEE. DOI: 10.1109/ECRTS.2014.16

BERNAT, Guillem; COLIN, Antoine; PETTERS, Stefan M.. WCET analysis of probabilistic hard real-time systems. **23rd Ieee Real-time Systems Symposium, 2002. Rtss 2002**., Austin, Texas, Usa, p.1-10, dez. 2002. IEEE Comput. Soc. DOI: 10.1109/REAL.2002.1181582

BRADLEY., James Vandiver. **Distribution-free statistical tests**. Englewood Cliffs, N.j: Prentice-hall, 1968. 388 p.

BUTTAZZO, Giorgio C.. **Hard Real-Time Computing Systems**: Predictable Scheduling Algorithms and Applications. Pisa: Kluwer Academic Publishers, 2011. 524 p.

CAZORLA, Francisco J. et al. PROARTIS: Probabilistically Analysable Real-Time Systems. **Acm Transactions On Embedded Computing Systems**, New York, Ny, Usa, v. 12, n. 2, p.1-26, 1 maio 2013. Association for Computing Machinery (ACM). http://dx.doi.org/10.1145/2465787.2465796.

CAZORLA, Francisco J. et al. Upper-bounding Program Execution Time with Extreme Value Theory. **13th International Workshop On Worst-case Execution Time Analysis (wcet 2013)**., Germany, p.64-76, jul. 2013. DOI: 10.4230/OASIcs.WCET.2013.64

COSTA, Celso Maciel da. Sistemas Operacionais – **Programação concorrente com Pthreads.** Porto Alegre: Edipucrs, 2010. 212 p.

CUCU-GROSJEAN, Liliana et al. Measurement-Based Probabilistic Timing Analysis for Multi-path Programs. **2012 24th Euromicro Conference On Real-time Systems**, [s.l.], p.1-11, jul. 2012. IEEE. DOI: 10.1109/ECRTS.2012.31

DAVIS, Robert I.; BURNS, Alan; GRIFFIN, David. On the Meaning of pWCET Distributions and their use in Schedulability Analysis. **Rtops - Real-time Scheduling Open Problems Seminar**. Dubrovnik, Croácia, p. 1-4. 27 jun. 2017. Disponível em: <http://www.cister.isep.ipp.pt/rtsops2017/RTSOPS17_proceedings.pdf>. Acesso em: 04 nov. 2018.

EDGAR, S.; BURNS, A.. Statistical analysis of WCET for scheduling. **Proceedings 22nd Ieee Real-time Systems Symposium (rtss 2001) (cat. No.01pr1420)**, London, Uk, v. 22, n. 1, p.215-224, dez. 2001. IEEE Comput. Soc. DOI: 10.1109/REAL.2001.990614

GIL, Samuel Jimenez et al. Open Challenges for Probabilistic Measurement-Based Worst-Case Execution Time. **Ieee Embedded Systems Letters**, [s.l.], v. 9, n. 3, p.69-72, set. 2017. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/les.2017.2712858.

GRIFFIN, David; BURNS, Alan. Realism in Statistical Analysis of Worst Case Execution Times. **10th International Workshop On Worst-case Execution Time Analysis (wcet 2010)**, Dagstuhl, Germany, v. 15, n. 1, p.44-53, jul. 2010. DOI: 10.4230/OASIcs.WCET.2010.44

KOSMIDIS, Leonidas et al. PUB: Path Upper-Bounding for Measurement-Based Probabilistic Timing Analysis. **2014 26th Euromicro Conference On Real-time Systems**, Madrid, Spain, p.1-12, jul. 2014. IEEE. http://dx.doi.org/10.1109/ecrts.2014.34.

LIMA, George; BATE, Iain. Valid Application of EVT in Timing Analysis by Randomising Execution Time Measurements. **2017 Ieee Real-time And Embedded Technology And Applications Symposium (rtas)**, [s.l.], p.187-197, abr. 2017. IEEE. http://dx.doi.org/10.1109/rtas.2017.17.

LIMA, George; DIAS, Dario; BARROS, Edna. Extreme Value Theory for Estimating Task Execution Time Bounds: A Careful Look. 2016 28th **Euromicro Conference On Real-time Systems (ecrts)**, [s.l.], p.1-12, jul. 2016. IEEE. http://dx.doi.org/10.1109/ecrts.2016.20.

MISSOURI STATE UNIVERSITY (Missouri). **Mars**: MIPS Assembler and Runtime Simulator. Disponível em: <https://courses.missouristate.edu/KenVollmar/mars/>. Acesso em: 15 fev. 2020.

OLIVEIRA, B.; SANTOS, M. M.; DESCHAMPS, F. Cálculo do tempo de execução de Códigos no Pior Caso (WCET) em aplicações de tempo real: um estudo de caso. **Revista Eletrônica de Sistemas de Informação**, v. 5, n. 1, p. 1-10, 2006. DOI: https://doi.org/10.21529/RESI.2006.0501002

OLIVEIRA, Rômulo Silva de; FRAGA, Joni da Silva; FARINES, Jean-marie. **Sistemas de Tempo Real**. Florianópolis: Ufsc, 2000.

PAOLIERI, Marco et al. Hardware support for WCET analysis of hard real-time multicore systems. **Proceedings Of The 36th Annual International Symposium On Computer Architecture - Isca '09**, Austin, Tx, Usa, v. 36, n. 9, p.57-68, jun. 2009. ACM Press. DOI: https://doi.org/10.1145/1555815.1555764

SHAW, Alan C.. **Sistemas e Software de Tempo Real**. São Paulo: Bookman, 2003.

SILVA, Karila Palma; ARCARO, Luis Fernando; OLIVEIRA, Romulo Silva de. On Using GEV or Gumbel Models When Applying EVT for Probabilistic WCET Estimation. **2017 Ieee Real-time Systems Symposium (rtss)**, [s.l.], p.220-230, dez. 2017. IEEE. http://dx.doi.org/10.1109/rtss.2017.00028.

STARKE, Renan Augusto. **UMA ABORDAGEM DE ESCALONAMENTO HETEROGÊNEO PREEMPTIVO E NÃO PREEMPTIVO PARA SISTEMAS DE TEMPO REAL COM GARANTIA EM MULTIPROCESSADORES**. 2012. 200 f. Dissertação (Mestrado) - Curso de Pós-Graduação em Engenharia de Automação e Sistemas, Universidade Federal de Santa Catarina, Florianópolis, 2012.

THEILING, Henrik; FERDINAND, Christian; WILHELM, Reinhard. Fast and Precise WCET Prediction by Separated Cache and Path Analyses. **Real-time Systems**, [s.l.], v. 18, n. 2/3, p.157-179, abr. 2000. Springer Nature. http://dx.doi.org/10.1023/a:1008141130870.

VARGAS, Fabian; GREEN, Bruno. Preliminaries on a Hardware-Based Approach to Support Mixed-Critical Workload Execution in Multicore Processors. In: **SECOND INTERNATIONAL CONFERENCE ON ADVANCES IN COMPUTING, CONTROL AND NETWORKING**, 2., 2015, Bangkok, Thailand. Second International Conference on Advances In Computing, Control And Networking. Bangkok: IRED, 2015. p. 23 - 27. DOI : 10.15224/978-1-63248-073-6-05

WILHELM, Reinhard et al. The worst-case execution-time problem—overview of methods and survey of tools. **Tecs**, [s.l.], v. 7, n. 3, p.1-53, 1 abr. 2008. Association for Computing Machinery (ACM). http://dx.doi.org/10.1145/1347375.1347389.

ZICCARDI, Marco et al. EPC: Extended Path Coverage for Measurement-Based Probabilistic Timing Analysis. **2015 Ieee Real-time Systems Symposium**, San Antonio, Tx, Usa, v. 1, n. 1, p.338-349, dez. 2015. IEEE. http://dx.doi.org/10.1109/rtss.2015.39.

APPENDIX A - IPET Example: Bubble Sort Case

This section aims to explain all steps on IPET computation for the item 6.2 Bubble Sort Case. The following image shows the chosen algorithm's CFG.

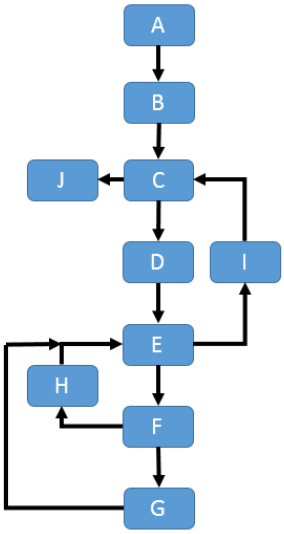

Figure 30 - Buble Sort CFG

According to IPET technique, every basic block has two parameters, time (T_i) and number of execution (N_i). To define the basic block WCET is required to multiply this block execution times (N_i) and its spent time for a unique execution (T_i). The WCET is the maximum summation of the outcomes of every block.

$$WCET = \sum_{i \in BasicBlocks} N_i * T_i$$

$$WCET = N_A * T_A + N_B * T_B + N_C * T_C$$
$$+N_D * T_D + N_E * T_E + N_F * T_F + N_G * T_G$$
$$+N_H * T_H + N_i * T_i + N_J * T_J$$

All basic block times are known, since instruction number is regarded as time unit in this case. The following figure shows the time for all blocks.

Basic Block		T_i
main	A	5
bubble	B	1
eloop	C	2
eloop1	D	1
iloop	E	2
iloop_1	F	7
iloop_2	G	2
swap	H	4
endiloop	I	2
end	J	2

Figure 31 - The basic blocks times.

Although the time for all blocks are known, the number of execution have to be computed. IPET define some rules for this computation:

- The first block executes only once ($N_A = 1$).
- The last block executes only once ($N_J = 1$).
- The number of times the execution enters a given block is the same number that the execution leaves the block.

Based on that it is possible the compute the block's number. It regards the numbers of execution for every block, as N_A, and also the number of transitions among the blocks, as N_{AB}. The IPET equations for this CFG are described below:

$$N_A = N_{AB}$$

$$N_B = N_{AB} = N_{BC},$$

$$N_c = N_{BC} + N_{IC} = N_{CD} + N_{CJ},$$

$$N_D = N_{CD} = N_{DE},$$

$$N_E = N_{DE} + N_{HE} + N_{GE} = N_{EF} + N_{EI},$$

$$N_F = N_{EF} = N_{FG} + N_{FH},$$

$$N_G = N_{FG} = N_{GE},$$

$$N_H = N_{FH} = N_{HE},$$

$$N_I = N_{EI} = N_{IC},$$

$$N_J = N_{CJ},$$

The first (N_A) block and the last (N_J) block are executed only once. Besides that, the outer loop blocks (N_D, N_I) upper bound is 8 times whereas the inner loop (N_F) upper bound is 64.

$N_A = 1, \quad N_B = 1, \quad N_D = 8, \quad N_I = 8, \quad N_F = 64,$

$1 = N_{AB},$

$N_B = 1 = N_{BC},$

$N_c = 1 + N_{IC} = N_{CD} + N_{CJ},$

$8 = N_{CD} = N_{DE},$

$N_E = 8 + N_{HE} + N_{GE} = N_{EF} + N_{EI},$

$64 = N_{EF} = N_{FG} + N_{FH},$

$N_G = N_{FG} = N_{GE},$

$N_H = N_{FH} = N_{HE},$

$8 = N_{EI} = N_{IC},$

$1 = N_{CJ},$

After that is possible to replace some variables values and simplify the equation:

$N_A = 1,$

$N_B = 1,$

$N_C = 9,$

$N_D = 8,$

$N_E = 72,$

$N_F = 64,$

$N_I = 8,$

$N_G = N_{FG} = N_{GE},$

$N_H = N_{FH} = N_{HE},$

The number of executions of the blocks N_G and N_H depends on how mess the vector values are. Analyzing the bubble sort algorithm is possible to infer that the worst scenario has decreasing values in vector (8, 7, 5, 6, 4, 3, 2 and 1, for example). In this worst-case, the algorithm swaps the values 28 times during

execution to order the vector. This implies $N_H = 28$ and, hence $N_G = 36$. The Figure 32 shows the number of executions for all blocks in the worst-case:

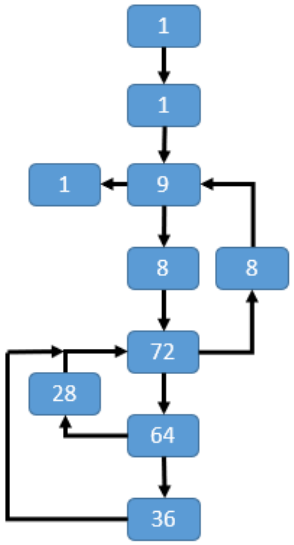

Figure 32 - The number of executions for each block in the worst scenario. Reference: Author

The Table XXIX displays the IPET result and all found values throughout this static analysis computation considering the worst-case. This analysis resulted in 826 executed instructions in the worst scenario, since this work regards executed instructions as time unit, the WCET is 826.

Table XXIX - Static analysis for WCET. Reference: Author.

Basic Block		T_i	N_i	Total
main	A	5	1	5
bubble	B	1	1	1
eloop	C	2	9	18
eloop1	D	1	8	8
iloop	E	2	72	144
iloop_1	F	7	64	448
iloop_2	G	2	36	72
swap	H	4	28	112
endiloop	I	2	8	16
end	J	2	1	2
			Total	826

APPENDIX B – EVT for multi-path WCET

Regarding multiple-path program, the EVT application is direct, although there are three essential conditions:

1) I.i.d. properties are requisite to the resultant distribution from multiple paths.

2) Each path requires a sufficient number of execution time samples.

3) The result refers to the observed paths set alone.

A. Independent and identically distributed observations

As in single-path programs, it is important to hold the i.i.d. property for multi-path programs, choosing random inputs in the measurement runs and grouping them, or testing all inputs and choosing the outcomes randomly when grouping. There is a necessary assumption in which it exists a direct and traceable correlation among the taken path, the observation and the input data and state.

B. Minimum Observations Number

EVT does not regard different paths frequencies in the measured samples, since the worst-case path dominates the outcome in block maxima approach. Otherwise, a minimum samples number of each path may be a requirement to characterize the behavior adequately. Executing each input at least this minimum number may ensure this property.

C. Path Coverage

The EVT result is reputable for observed paths only. Untested paths might not be related to the pWCET estimates, since the execution time would not compound an identical distribution for those paths. This connects the i.i.d. concept to the path coverage achieved by input data. Furthermore, in complex applications, the loops and branches combinations increase considerably the number of possible paths during the program. For this reason, it is possible to establish a warning on the EVT result, describing it as valid only for the observed paths during execution (CUCU-GROSJEAN et al., 2012).

D. EVT Step-by-step for multi-path

The EVT application for multi-path program is very similar to single-path. There is only one difference in the first step, during the observations collection (CUCU-GROSJEAN et al., 2012). Each path requires an acceptable sample number to characterize its behavior. In other words, the number of paths multiplies the sample number required. For instance, it may be viable to start with 100P executions, where P is the number of paths. Moreover, it is possible to increase the additional number of observations in each round to $P * N\Delta$.

APPENDIX C – Path Upper-Bounding (PUB)

The Path Upper-Bound (PUB) method extends MBPTA to upper bound for any path in the program under analysis, even if the input vector does not induce the worst-case path. PUB extends the original program, adding instructions in the different program paths. Therefore, the execution time of any path upper bounds the worst-case execution time. However, it uses the extended program only for pWCET analysis time (KOSMIDIS et al., 2014).

PUB estimates the probability of any program path upper bound. Figure 33 compares both MBPTA and PUB flows, since the collection step until pWCET estimating. MBPTA provides a pWCET estimating only for analyzed paths, as the analysis result may not upper-bound non-exercised paths. On the other hand, PUB operates on the original code. It is possible to upper bound the program execution time exercising any path, since the method adds instructions in every code branch. Moreover, Kosmidis et al. (2014) propose two PUB techniques: Address Merging (PUBam) and Address Aging (PUBaa).

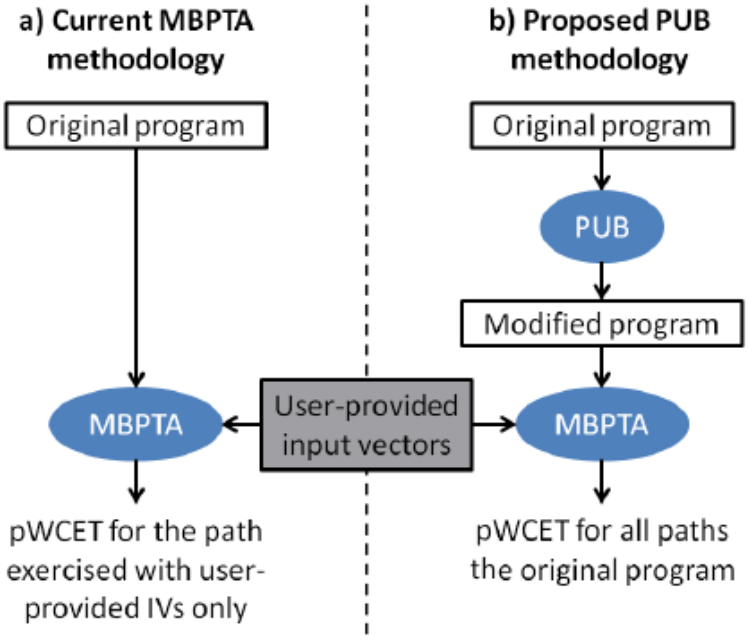

Figure 33 - MBPTA and PUB methodologies. Reference: Kosmidis et al. (2014).

<Onde está a ref no texto desta figura?>

8.1.1 Address Merging (PUBam)

This methodology consists of performing the same data accesses regarding the same order in any path of a given conditional branch. Figure 34 shows a conditional branch in which the first path accesses the addresses @A, @B and @C, while the second accesses the addresses @D and @E. In this case, the PUBam proposes to edit the code to ensure a safe worst-case for both paths. PUBam also maintains the relative order of accesses. In other words, it respects the sequences @A, @B, @C and also @D, @E. Consequently, there are two possibilities for this case: access <@A;@B;@C;@D;@E >, or <@D;@E;@A;@B;@C >, or <@A;@D;@B;@E;@C>, but not <@A;@E;@D;@C;@B>.

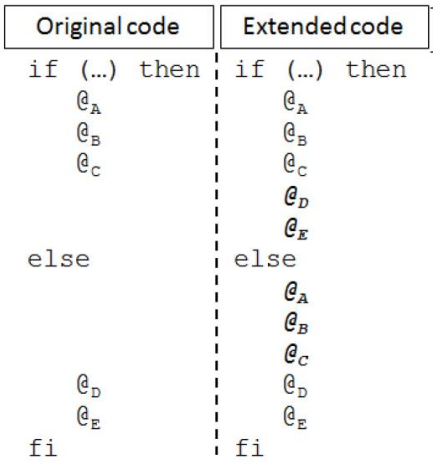

Figure 34 - Simple Code Replication. Reference: Kosmidis et al. (2014).

To optimize the solution replicating all access in all branches it is possible to avoid unnecessary access replications. In order to do that, sequences or addresses replicated in both paths must be identified and not copied. The figure 35 shows an example that illustrates how a given if-then-else can be optimized. In this code there are three addresses sequences, @A,@C and @C, occurring both in then and else paths. Due to this, there is no blank space for those address in the left side column.

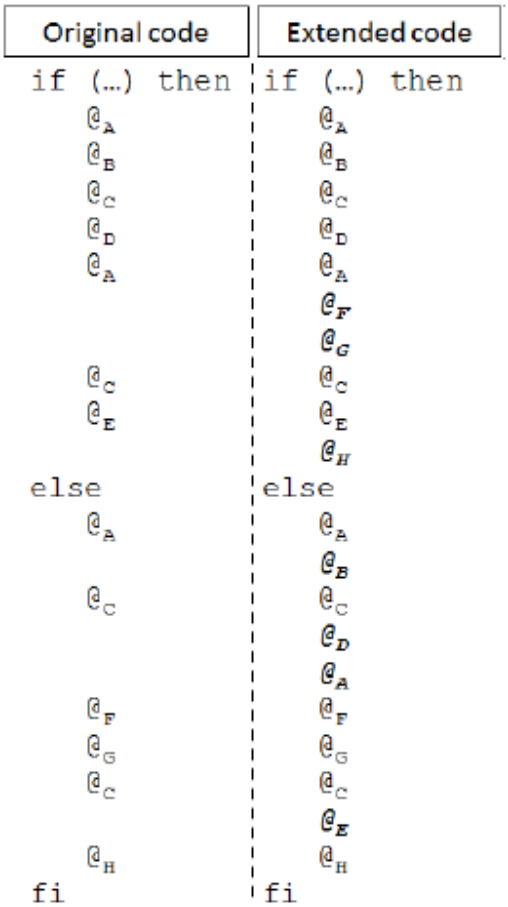

Figure 35 - Code identification and replication. Reference: Kosmidis et al. (2014).

Kosmidis et al. (2014) also consider different instructions flows as switch, loop, nested if-then-else and if-then (without else) constructs.

- If-then: Regarding this construct as an if-then-else with an empty else branch, PUB simply adds in the else branch the code in the then branch.
- Switch: Switch constructs may have more than two branches. PUB replies the sequence access as in if-then-else, however it does this for all branches.
- Nested Conditionals: For nested conditional PUB, it may be applied recursively. Figure 36 shows an example of if-then-else and if-then nested. This figure shows the original code in the first column, the inner conditional applying PUB in the second column and the final code in the last column.
- Loops: Access in any branch remains the same in loops, although data cache access may vary crosswise iterations. For this reason, branch upper bounding is trustworthy even in this case. It is noteworthy mentioning that PUB replicates access only for those that do not access the same address in all branches in each iteration. The same occurs when accessing an array content, if different branches access the same array in different positions, those accesses must be replicated by PUB.
- Infeasible paths, error codes: In several cases, there is no need to consider some branches in pWCET computation. In other words, it is possible to reduce the PUB overhead. For that, the user can instruct PUB, by annotations, to not consider instructions in some branches.
- Operation modes: Software with different, and mutually exclusive, operations modes may be upper bounded in every operation mode independently. The user might identify and indicate every operation mode code to PUB and provide a path coverage for that mode instead of covering all code paths. Switch constructs, for instance, have different modes for each case.
- Function Calls: when a given branch calls a function, PUB replicates its cache effects to the other branches. PUB may use a dummy function that access the same addresses in the same order that function call does. Alternatively, it is possible to apply a technique named Address Aging. However, if all branches call the same function, there is no need to replicate it.

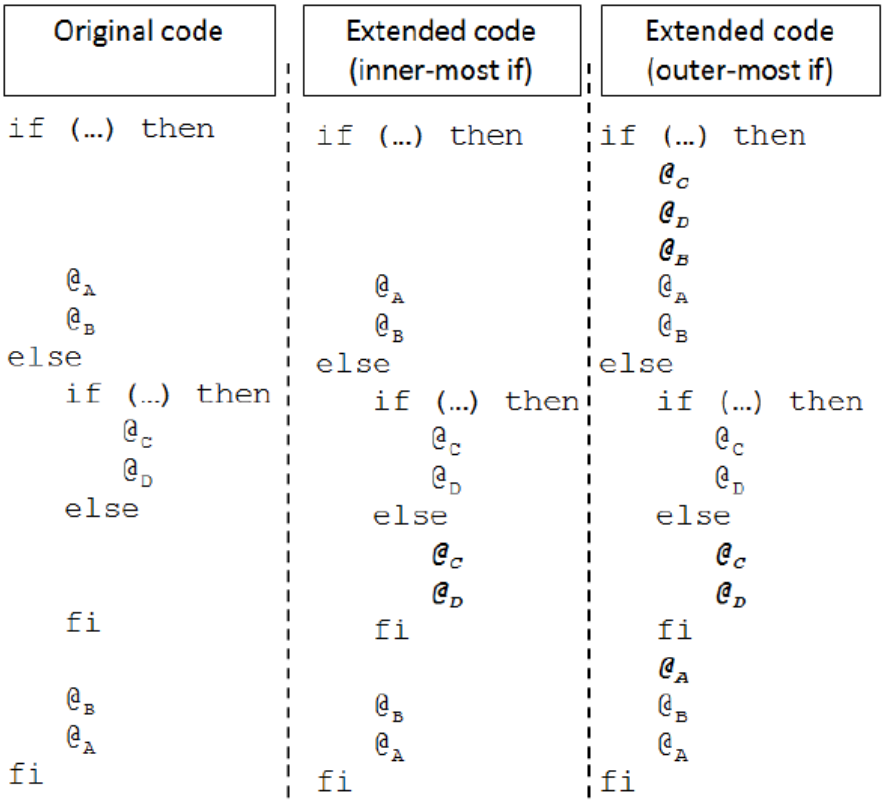

Figure 36 - Nested If code replication. Reference: Kosmidis et al. (2014).

8.1.2 Address Aging (PUBaa)

PUBam (Address Merging) is less efficient when each branch has different addresses sets to access, since PUB copies every accessed address to all branches (KOSMIDIS et al., 2014). Considering a switch with 10 cases, for instance, if each branch access two different addresses, will result in an extended switch in which all branches access 20 addresses.

To handle it, Kosmidis et al. (2014) propose the Address Aging technique (PUBaa). This method adds addresses accessed nowhere else in the code, resulting in missing and fetching useless data, instead of copying addresses on all the other branches. The access number PUBaa adds is the maximum number of any branches access. Regarding the previous example, instead of copy all the addresses, it adds two unique addresses in each branch. Therefore, every branch must have four addresses instead of 20. This still ensures the switch worst-case, since there are at least two misses occurring in every case.

PUBaa is potentially more efficient for a higher number of branches. However, for high imbalance and a low number of branches, PUBam is more efficient. PUBaa might be applied by using a data structure (dummy) and a pointer (next) to ensure that the address is accessed once (dummy[next]). Although it is also possible to implement PUBaa with hardware support, creating a special instruction that induces a cache miss, causing some data eviction, accessing memory and bringing useless content to cache memory.

APPENDIX D – Extended Path Coverage (EPC)

Ziccardi et al. (2015) present the Extended Path Coverage (EPC). This technique enhances the MBPTA process and results in a valid pWCET for the whole program, even without the worst-case inputs, either the full path coverage. EPC regards only measurements on the original program code, unlike PUB. EPC derives an execution times collection, which represents all the program paths just counting on basic block measurement sets. Therefore, it requires a basic block full coverage.

In the MBPTA approach, the result is valid only for the paths covered and executions conditions for observations collected. EPC, on the other hand, extends the observations set synthetically to get the equivalent effect of full path coverage. EPC requires collecting executions time observations, sample set, for every program basic blocks.

Basic block is a code snippet, the smallest unit of sequential code to execute. Measure basic blocks is industrially viable since the executions probing overhead might be negligible by adopting trace tools or advanced hardware debug interfaces.

It is possible to make probabilistic execution times for a basic block to be path-independent. EPC increases, probabilistically, the probabilistic execution time to balance the benefits that basic blocks may take in some cases. As a given traversal path leading to this block, due to history sensitivity of execution, for instance.

Every basic block execution time that is probabilistically path-independent may construct a whole program representative collection of execution times, even blocks that input vector do not exercise. The execution time observations over basic blocks may not be combined to obtain unobserved path execution time. By the way, path-independence is a mandatory requirement. Due to cache-level dependence and core-level dependence effects, each observation corresponds only to the executed path during the run.

EPC copes these dependencies identifying unobserved paths' probabilistic impact on the observed execution times set and it generates an artificially extended set of execution times regarding the entire program. MBPTA process might analyze this extended set to define the worst-case. The computed pWCET may be fully trustworth, once it is regarded as upper bound, due to the extended measurements obtained by thoroughly observing each possible path.

Figure 37 depicts the EPC and MBPTA processes interactions. The main EPC process steps are (a) collecting execution time samples, (b) padding basic blocks and (c) construction of execution time. The original observations and the synthetic execution times, which EPC generates, feed the MBPTA process and allow it to compute a trustworthy pWCET for any exceedance probability. This pWCET relates to all paths in the program and all possible addresses. Furthermore, MBPTA convergence criterion may require synthetic measurements or observations.

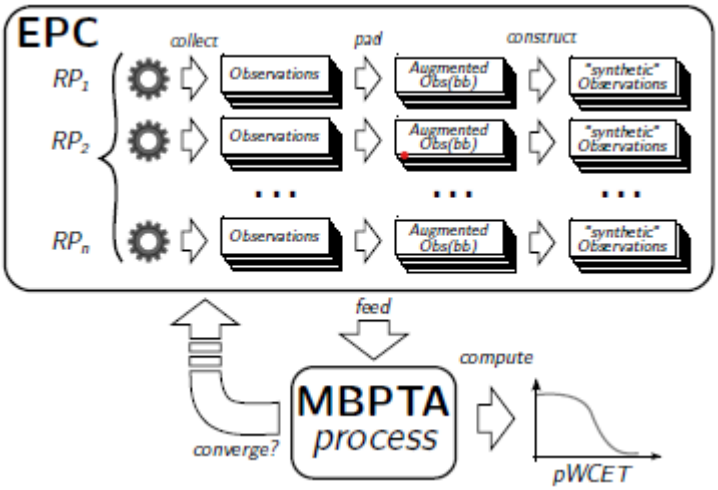

Figure 37 - EPC interaction with standard MBPTA process. Reference: Ziccardi et al. (2015).

Figure 38 shows the EPC steps applied on a simple program. This program consists of two cascading conditional code. The first step collects execution time for individual basic blocks across the paths φ0 and φ2. Afterwards, the second step augments the execution times collected for each basic block to turn them into path-independent. The third step synthetically computes execution time values for all the non-observed paths φ1 and φ3.

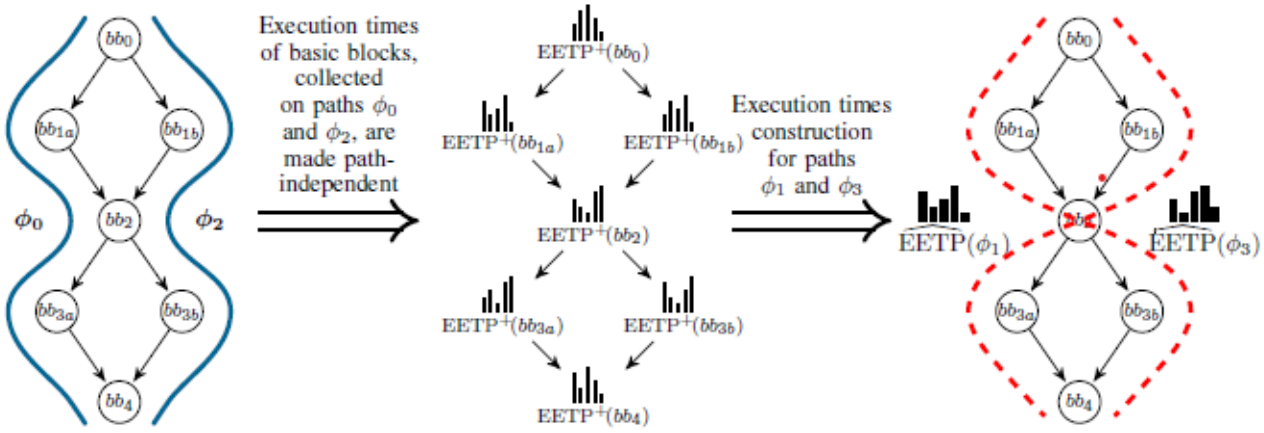

Figure 38 - EPC steps applied on a simple program. Reference: Ziccardi et al. (2015).

The first step is to collect execution times for the program. It is not required to exercise all the paths, though it must cover every basic block (bbi) in the program. After extracting timing information for each basic block, it is possible to generate an empirical execution time profile EETP(bbi), relating every basic block bbi. The second step applies observations Obs(bbi, φ) to all basic blocks and their respective EETP(bbi). Each Obs(bbi, φ) is made probabilistically path-independent, denoted Obs+(bbi, φ), by adding a probabilistic padding. Augmented observations induce augmented execution time profiles EETP+(bbi). These profiles are independent of observed paths and over-approximates every basic block timing behavior on every path in the program that passes through it. The last step combines path-independent profiles to define a synthetic execution time profile $\overline{EETP}(\varphi)$ for each non-observed path φ. Building blocks in $\overline{OBS}(\varphi i)$ computation, regarding φi as a program path, they are always valid upper bounds, since using probabilistically path-independent

EETP+(bbi) ensures it. Feeding a $\overline{EETP}(\varphi)$ complementary set to MBPTA enables it to obtain a valid pWCET for all execution paths and for any exceedance probability. EPC application ends and gathers all the artificially constructed execution times obtained under different cache placements, randomly generated, and providing them to MBPTA.

APPENDIX E – Matlab code examples

The following code computes GEV for Block Maxima approach in Matlab, the filtering process happens before this execution and is provided in a samples.csv file.

```
clc; close all; clear all;
importfile('samples.csv');

[f,x] = ecdf(data);
f2 = 1-f;
paramEsts = gevfit(data);
paramEsts

location = paramEsts(3);
scale    = paramEsts(2);
shape    = paramEsts(1);

p = 1 - gevcdf(x,shape,scale,location);

plot(x,p,x,f2);
figure;
qqplot(p,f2);
xlim([0 1]);
ylim([0 1]);

% for a better analysis decrease x step
% This example uses a range between 2570 and 2580
x = (2570:0.002:2580);

p = 1 - gevcdf(x,shape,scale,location);
```

```
p2 = 10^-9;
figure;
plot(x,p,x,p2);
```

The following code computes GP for Block Maxima approach in Matlab, the filtering process happens before this execution and is provided in a samples.csv file.

```
clc; close all; clear all;
importfile('samples.csv');

[f,x] = ecdf(data);
f2 = 1-f;
paramEsts = gpfit(data);

scale    = paramEsts(2);
shape    = paramEsts(1);

p = 1 - gpcdf(x,shape,scale);
figure;
plot(x,p,x,f2);
figure;
plot(x,f2);

figure;
qqplot(p,f2);

% for a better analysis decrease x step
% This example uses a range between 923.64 and 924
% x = (923.64:0.002:924);

p = 1 - gpcdf(x,shape,scale);

p2 = 10^-9;
```

```
figure;
plot(x,p,x,p2);
```

Pontifícia Universidade Católica do Rio Grande do Sul
Pró-Reitoria de Graduação
Av. Ipiranga, 6681 - Prédio 1 - 3º. andar
Porto Alegre - RS - Brasil
Fone: (51) 3320-3500 - Fax: (51) 3339-1564
E-mail: prograd@pucrs.br
Site: www.pucrs.br