Probabilistic study of end-to-end constraints in real-time systems

Cristian Maxim

THÈSE DE DOCTORAT DE
l'UNIVERSITÉ PIERRE ET MARIE CURIE

Spécialité

Informatique

École doctorale Informatique, Télécommunications et Électronique (Paris)

Présentée par

Cristian MAXIM

Pour obtenir le grade de

DOCTEUR de l'UNIVERSITÉ PIERRE ET MARIE CURIE

Sujet de la thèse :

Étude probabiliste des contraintes de bout en bout dans les systèmes temps réel

soutenue le 11 decémbre 2017

devant le jury composé de :

Mme. Liliana CUCU-GROSJEAN	Directeur de thèse
M. Benoit TRIQUET	Coordinateur Industriel
Mme. Christine ROCHANGE	Rapporteur
M. Thomas NOLTE	Rapporteur
Mme. Alix MUNIER	Examinateur
M. Victor JEGU	Examinateur
M. George LIMA	Examinateur
M. Sascha UHRIG	Examinateur

Contents

Table of contents **iii**

1 Introduction **1**

1.1 General Introduction 1

1.1.1 Embedded systems 2

1.1.2 Real-time domain 4

1.1.3 Avionics industry 9

1.2 Context 14

1.2.1 Performance race 15

1.2.2 Execution time 17

1.2.3 Probabilities and statistics 21

1.3 Thesis motivation 22

1.4 Model 24

1.4.1 Useful notions from the probability theory 24

1.4.2 Probabilistic real-time system 26

2 State Of The Art **29**

2.1 Time analysis of probabilistic real-time systems 31

2.2 Probabilistic methods 38

2.3 Measurement based probabilistic analysis 41

2.4 Randomized architectures 49

2.5 Timing analysis in avionics industry 51

2.5.1 Integrated Modular Avionics (IMA) 52

2.5.2 Time analysis of avionics applications 54

2.5.3 Mixed-criticality systems 56

3 Conditions for use of EVT in the real-time domain 59
3.1 System consistency . . . 60
3.1.1 Input . . . 61
3.1.2 Software . . . 62
3.1.3 Platform . . . 63
3.2 Identical distributed data . . . 64
3.3 Independence . . . 66
3.4 Reproducibility and representativity of measurement-based approaches 69
3.4.1 The reproducibility of the WCET estimation method . . . 71
3.4.2 The reproducibility of the measurement protocol . . . 72
3.4.3 Representativity of a measurement protocol . . . 72
3.4.4 Relations between reproducibility, representativity and convergence . . . 74
3.5 Conclusion . . . 76

4 pWCET estimation methodology 77
4.1 Generalized extreme value distribution . . . 78
4.2 Generalized Pareto distribution . . . 83
4.3 Validation of statistical results . . . 88
4.4 The pWCET estimation from dependent execution times . . . 94
4.5 Small variability data . . . 96
4.6 Conclusions . . . 97

5 Experimental results 99
5.1 Analysis of benchmarks on multiprocessor architectures . . . 99
5.2 Avionics application analysis . . . 105
5.2.1 Application presentation . . . 107
5.2.2 Platform characteristics . . . 114
5.2.3 Timing analysis results . . . 127
5.3 Conclusions . . . 156

6 General conclusions 157
6.1 Contributions . . . 157
6.2 Future work . . . 158

A Statistical tests **161**

A.1 Run test 161

A.2 Kolmogorov-Smirnov test 162

A.3 Anderson-Darling Test 163

List of Figures **169**

List of Tables **171**

Nomenclature **177**

Bibliography **196**

Chapter 1

Introduction

1.1 General Introduction

Human life in the 21^{st} century is surrounded by technology. From household to transportation, from education to hobbies and from security to sports, informatics plays a major role in daily activities. Social interaction, education and health are only a few examples of domains in which the fast evolution of technology had a major positive impact on the quality of life. Businesses rely more and more on embedded systems to increase their productivity, efficiency and value. In factories, robot precision tends to replace the human versatility.

Even though connected devices like drones, smart watches, or smart houses, are becoming more popular in the last years, in the industries that deal with user security, this kind of technology have been used for many years. Avionics industry has been using computers for their products since 1972 with the production of the first A300 airplane and has reached astonishing progress with the development of the first Concorde airplane in 1976, which was considered a miracle of technology, surpassing with many years the airplanes of its time. A considering number of innovations and knowledge acquired for the Concorde are still used in the recent models like A380 or A350.

A slower start of technological evolution can be seen in the space or automotive industries, but with the start of OneWeb project [OneWeb, 2015] and the introduction of self-driving cars, these domains have encountered an acceleration phase that does not seem to take a break any time soon. In this section we present an overview on the technologies used in the aforementioned industries with an emphasis

on the real-time domain as an important part critical real-time embedded systems (CRTES).

1.1.1 Embedded systems

In our times, we are surrounded by technologies meant to improve our lives, to assure its security, or programmed to realize different functions and to respect a series of constraints. We consider them as embedded systems or often as parts of cyber-physical systems.

An **embedded system** is a microprocessor-based system that is built to control a function or a range of functions and is not designed to be programmed by the end user in the same way that a PC is [Heath, 2002]. A user can make choices concerning functionality but cannot modify the functionality of the system by adding or replacing software. Embedded systems are managed by single or multiple processing cores in the form of micro-controllers or digital signal processors (DSP), field-programmable gate arrays (FPGA), application-specific integrated circuits (ASIC) and gate arrays. These processing components are integrated with components dedicated to handling electric and/or mechanical interfacing. An embedded system's key feature is the dedication to specific functions that typically require strong general-purpose processors. For example, router and switch systems are embedded systems, whereas a general-purpose computer uses a proper OS for routing functionality. However, embedded routers function more efficiently than OS-based computers for routing functionalities. Commercial embedded systems range from digital watches and MP3 players to giant routers and switches. Complexities vary from single processor chips to advanced units with multiple processing chips.

Compared to an embedded system, which puts emphasis on the computational elements, a **cyber-physical system** is an integration of computation with physical processes [Lee and Seshia, 2011]. Majority of cyber-physical systems contains at least one embedded system. The term of cyber-physical system is very broad and encompasses various research topics from software and modeling to systems, networking, and control in computer science and engineering. A detailed concept map of these systems can be seen in Figure 1.1 [Cyber, 2010]. This taxonomy is an evolving one and new domains are added in a continuous manner.

A **real-time system** is any information processing system which has to respond to externally generated input stimuli within a finite and specified period [Burns and

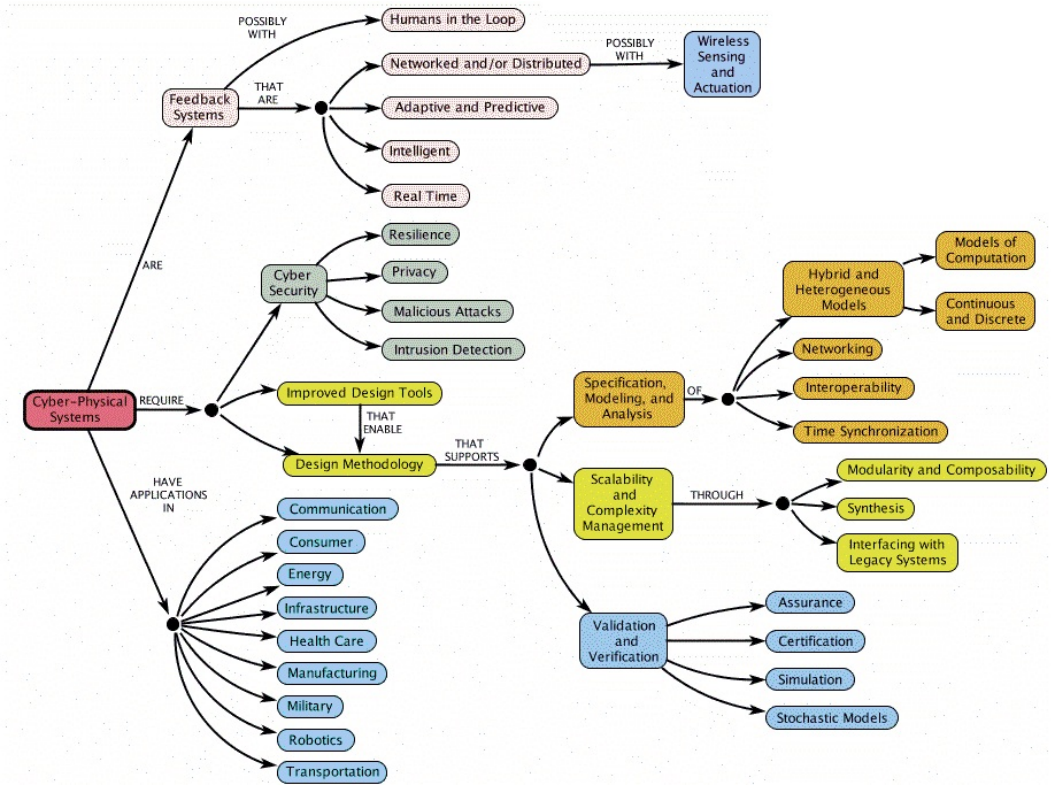

Figure 1.1: Cyber-physical systems - a Concept Map. Image made by Edward A. Lee after a taxonomy given by S. Shyam Sunder [Lee, Edward Ashford, 2012].

Wellings, 2001]. In this type of systems, the correctness depends not only on the logical result but also on the time it was delivered. A failure to respond is as bad as the wrong response. Real-time systems can be found in industries like aeronautics, aerospace, automotive or railways but also in sensor networks, image processing, multimedia applications, medical technologies, robotics, communications, computer games or household systems. According to their deadline miss importance, real-time systems can be classified as follows:

- **Hard real-time systems** - where it is absolutely imperative that responses occur within the required deadline. For these systems, an overrun in response time can be critical, leading to potential life loss and/or big financial damage. Therefore many of these systems are considered to be safety critical. If the deadline miss endangers a very expensive process, the system can be considered mission critical but not necessarily safety critical. Such examples

are the spatial missions, which reach very high budgets and in the case of an accident, even though no life would be endangered, the financier loss would be considerable.

- **Soft real-time systems** - where deadlines are important but which will still function correctly if deadlines are occasionally missed. These systems are often connected to quality-of-service and are able to tolerate deadline misses even though they are not desired.

- **Firm real-time systems** - which are soft real-time but in which there is no benefit from late delivery of service. For these systems the computation is obsolete if the response is not given in time. Forecasts systems are relevant examples in this case.

- **Weakly hard real-time systems** - where k out of m deadlines have to be met ($k < m$). These are the systems that become unstable with too many deadline misses. As an example we can mention the feedback control systems.

All these systems have a cost function associated to them. An abstract depictions of these cost functions can be found in Figure 1.2.

Between real-time systems and embedded systems there is a relation of inclusion since all real-time systems are at the same time embedded systems, but not all embedded systems present time constraints. On the other hand, there is no direct relation between embedded systems and cyber-physical systems. Most cyber-physical systems contain one or more embedded systems behaving as component parts, but their names cannot be interchanged due to difference of complexity. In this thesis we concentrate on real-time embedded systems and for ease of notations we simply name them real-time systems. We might refer to cyber-physical systems if that will be the case.

1.1.2 Real-time domain

The principles of real-time systems are first defined by Stankovic in [Stankovic, 1988] and since then the concept becomes an important part of major industries like avionics, automotive or aerospace. The basic model introduced by Liu and Layland [Liu and Layland, 1973] considers the existence of a set of tasks Γ with n tasks $\tau_i = (O_i, C_i, P_i, D_i)$, with $i = 1...n$, characterized by an offset O_i, a worst

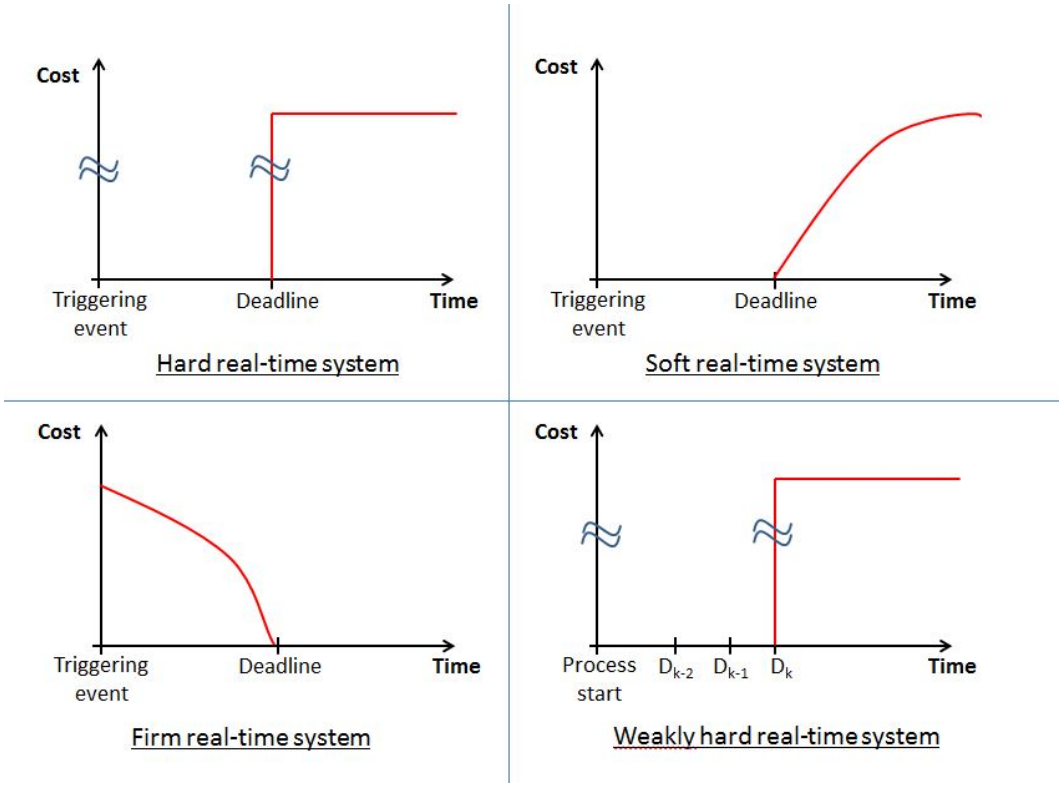

Figure 1.2: Cost functions of real-time systems.

case execution time C_i, a period P_i and a deadline D_i. To ensure the feasibility of such a system, all tasks occurrences must finish before their deadlines. In time, this model evolved giving birth to new associated notions. Here are some of the common notions:

The **Worst Case Execution Time (WCET)** of a task represents the maximum time it can take to be executed. The WCET is obtained after analysis and most of the time it cannot be accurately determined by exhausting all the possible executions. This is why, in industry, the measurements are done only on a subset of possible scenarios (the one that would generate the highest execution times) and an execution time bound is estimated by adding a safety margin to the greatest observed time. At the opposite side, we encounter the notion of **Best Case Execution Time (BCET)**, which represents the minimum time length a task can take to be executed. A detailed report regarding the WCET notion is done by Wilhelm et al in [Wilhelm et al., 2008].

The **Offset** of a task is the exact time at which that task is supposed to start

its execution while the **Jitter** is the deviation from its periodicity. While the offset occurs at once in a task's lifetime, the jitter can take place at multiple or all instances of the task. The offset can be defined by the system or programmer but the jitter is an undesired effect that occurs due to architecture components. Studies have been done in [Leung and Merrill, 1980] and [Pellizzoni and Lipari, 2005] for systems that contain offsets and in [Goossens and Devillers, 1997] and [Goossens, 2003] for offsets-free ones. The jitter was taken in account in the scheduling analysis proposed by Audsley et al. [Audsley et al., 1993]. Marti et al. [Marti et al., 2001] have identified the types of jitter that can occur in distributed real-time systems, while Nilsson et al. [Nilsson et al., 1998] took it into account for stochastic analysis.

Periodic real-time systems - are described by tasks that release their instances (jobs) in a periodical manner. Therefore, the j^{th} job of task τ_i will be released at time $O_i + j * P_i$.

Sporadic real-time systems - are different from the periodic ones by the fact that the composing tasks release jobs with a time interval equal or larger than the period P_i.

The **utilization** of a task τ_i is described by the formula $U_i = C_i/P_i$, where the system's utilization is $U = \sum_{i=1}^{n} C_i/P_i$.

The order of execution of jobs waiting at the same time for one resource is given by their **priority**. The priority can be given at task level or at job level and it can be fixed or dynamic.

If a task with higher priority is able to interrupt the execution of a lower priority, task we call this action **preemption** and the algorithm implementing it is preemptive.

The property that indicates whether a real-time system can meet its deadlines is called **schedulability**. Scheduling is used to determine the order in which the tasks are executing. In literature there are multiple **scheduling algorithms** able to schedule task of models described by different combinations of previously mentioned notions. Liu and Leyland [Liu and Layland, 1973] define the Rate Monotonic algorithm, which gives higher priorities to tasks with smaller periods. The algorithm that takes into consideration the deadlines rather than the periods in order to attribute priorities is introduced by Leung and Whitehead in [Leung and Whitehead, 1982] with the name of deadline monotonic. The Earliest Deadline First (EDF) algorithm [Liu and Layland, 1973] is a scheduling algorithm that allows any pre-

emptive task sets with utilization lower or equal than 1 to be scheduled by giving to the job with earliest deadline the highest priority and allowing it to be executed first.

Fixed priority scheduling algorithms were analyzed by Audsley et al [Audsley et al., 1993] and by Bini and Buttazo in [Bini and Buttazzo, 2004]. As a compromise between preemption overheads and low schedulability of non-preemptive systems, limited preemption scheduling was proposed in papers like [Burns, 1993], [Baruah, 2005] and [Yao et al., 2011].

Larger and more detailed surveys on real-time scheduling exists, from which we mention [Audsley et al., 1995] and [Sha et al., 2004] as complete and well-structured ones.

With the increase in application complexity, the real-time systems had been confronted with the problem of **resource sharing**. This problem occurs when two or more tasks are supposed to use the same resource and can produce an important increase in tasks response times through phenomena like priority inversion or deadlocks. To confront these problems, Sha et al [Sha et al., 1990] proposed a priority inversion protocol and Baker [Baker, 1991] introduced the stack resource policy.

Mixed-criticality systems is a hot topic in the real-time domain. It appears as an answer to the continuous increase in complexity of applications that lead to inefficient systems that were treating low critical tasks just like the high critical ones. The first model of mixed-criticality system is introduced by Vestal in [Vestal, 2007] and had further been developed and analyzed in works like [Baruah et al., 2011], [Baruah et al., 2012] and [Ekberg and Yi, 2014].

Time Analysis is a key concept that had been used in real-time systems to assign an upper bound to the WCETs of tasks or program fragments. This assignment can be achieved either by static analysis [Ferdinand et al., 2001] or by measurement based analysis [Bernat et al., 2002], [Wenzel, 2006]. Figure 1.3 proposed in [Wilhelm et al., 2008] by Wilhelm et al. depicts real-time properties for a better understanding of timing analysis.

While using similar methods in the combination of program fragments execution times, the way these estimates are obtained is fundamentally different.

Static analysis uses abstract models of the targeted hardware and computes a complete universe of all possible execution states that a program can reach starting from all possible initial states. Based on these states, an upper bound of the

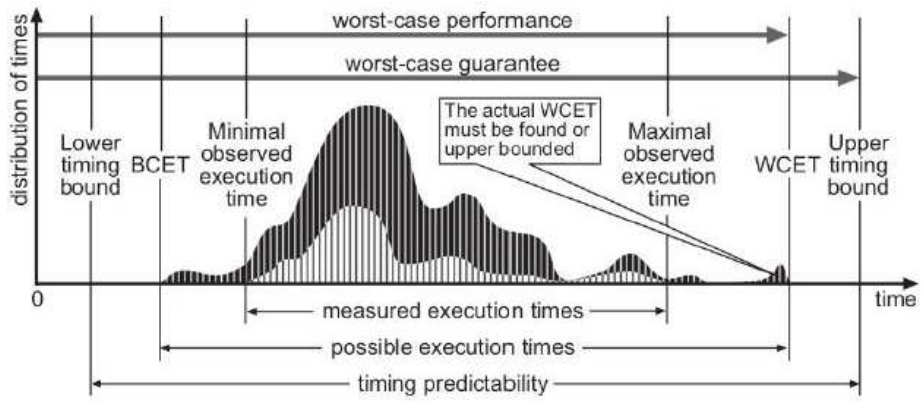

Figure 1.3: Basic notions related to timing analysis. The lower curve represents a subset of measured executions. Its minimum and maximum are the minimal observed execution times and maximal observed execution times. The darker curve, an envelope of the former, represents the times of all executions. Its minimum and maximum are the best case and worst case execution times, abbreviated BCET and WCET.

execution time can be derived.

Measurement based analysis is pretty straight forward and consist in executing each program fragment with a subset of the possible initial states and inputs. The inputs and states that are considered to stress the most the systems are usually chosen, but the maximum of the measured execution times is in general an underestimation of the WCET (see Figure 1.3). The correctness of static analysis estimations depends on the correctness of the abstract model and the creation of these models is an error-prone and laborious process especially if no precise specification of the hardware is available. This is happening often nowadays with the use of commercial of the shelf (COTS) hardware that may make static analysis a costly process. These cases may give an advantage to measurement analysis over static analysis as they may be more portable to new architectures, as it does not rely on such abstract models of the architecture. On the other hand, soundness of measurement-based approaches are hard to guarantee.

Measurement would trivially be sound if all initial states and inputs were covered. Due to their huge number, this is usually not feasible. Instead, only a subset of the initial states and inputs can be considered in the measurements.

Both static and measurement based methods, in their deterministic approaches, tend to be extremely pessimist. Unfortunately not all real-time systems can afford this pessimism and the consequent over-provisioning, and for these cases other approaches should be considered. One such approach is the use of probabilities in the time analysis or in describing the analyzed model. We detail the existing results on this topic in the following sections of this thesis.

1.1.3 Avionics industry

Amongst all branches of real-time systems, an important role is played by the Critical Real-Time Embedded Systems (CRTES) domain. CRTESs are widely being used in fields like automotive, avionics, railway, health-care, etc. The performance of CRTESs is analyzed not only from the point of view of their correctness, but also from the perspective of time.

In the avionics industry such systems have to undergo a strict process of analysis in order to fulfill a series of certification criteria demanded by the certifications authorities, being the European Aviation Safety Agency (EASA) in Europe or the Federal Aviation Administration (FAA) in United States.The avionics industry in particular and the real-time domain in general are known for being conservative and adapting to new technologies only when it becomes inevitable. For the avionics industry this is motivated by the high cost that any change in the existing functional systems would bring. Any change in the software or hardware has to undergo another certification process which cost the manufacturer money, time and resources. Despite their conservative tendency, the airplane producers cannot stay inactive to the constant change in technology and ignore the performance benefices brought by COTS processors which nowadays are mainly multi-processors. As a curiosity, most of the microprocessors found in airplanes flying actually in the world, have a smaller computation power than a modern home PC. Their chips-sets are specifically designed for embedded applications characterized by low power consumption, predictability and many I/O peripherals.

The majority of airplane accidents documented so far are due to human errors, nevertheless there have been some accidents provoked by one or multiple software issues [Wong et al., 2009]. For example, the Air France Flight 447 (31 May 2009, an Airbus A330-200) reported that the system transmitted several messages regarding discrepancies in the indicated air speed readings before the aircraft disappeared.

Even though, this accident is classified as a human error, Airbus issued an Accident Information Telex to operators of all Airbus models reminding pilots of the recommended Abnormal and Emergency Procedures to be taken in the case of unreliable airspeed indication.

On 20^{th} of December, 1994, the American Airlines Flight 965 crashed in the Andes a few minutes before its scheduled arrival time. The cause of the crashed is portrayed as a pilot error but a large part of the blame lies within the poor design of the software system. In short, the pilot failed to choose the correct radio bacon necessary to correctly calculate the aircraft trajectory and the mistake forced the plane to do a wide semicircular turn to the east. By the time the error was detected, the plane was going straight forward towards a 3000 meter mountain. A more detailed report on these errors and other catastrophic accidents in which the software played an important role can be found in [Wong et al., 2009].

Recently, an Airbus A400M doing a test flight crushed in Spain during takeoff, producing four fatalities. Investigations revealed that an incorrect installation of the engine control software during production may have caused three out of four engines to stop responding to throttle commands, thus causing the accident [A400M, 2015].

In order to avoid such accidents or to reduce drastically their probability of appearance, all safety-critical systems are legally required to undergo a *certification* process. This process is effectuated by independent authorities that will allow certified aircraft to fly.

Software certification

Thoroughly examination is done to all civil airplanes before operational use. This examination is done by independent legal authorities through specific industry standards. In order to be allowed to carry civilians, the aircraft have to obtain *certification* by passing a series of objectives mentioned in the standards. According to the criticality of the system, every component might need to pass a different number of objectives. The **criticality** is a designation of the level of assurance against failure needed for a system component. In other words, every system receives a different level of certification depending on the consequences that might arise from its failure.

The certification standards used as a reference for the development of an airplane are:

- ARP-4754 [SAE, 2010] deals with the system development

- DO-254 [RTCA, 2015] is the standard for the hardware development life cycle
- DO-178C [RTCA, 2015] gives the standard for software development life cycle.

Each component is given a different set of requirements and placed on its corresponding Safety Integrity Level (SIL) or Design Assurance Level (DAL). In the avionics industry there are five DALs, noted from A (the highest criticality) to E (the lowest criticality). The description of each DAL is found in the ARP-4761 [SAE, 1996] from which we extracted Table 1.1.

Level	**Proba.**	**Severity**	**Failure condition effect**
DAL-A	$10^{-9}/h$	Catastrophic	All failure conditions which prevent continued safe flight and landing
DAL-B	$10^{-7}/h$	Hazardous	Large reduction in safety margins or functional capabilities; higher workload or physical distress such that the crew could not be relied upon to perform tasks accurately or completely; adverse effects upon occupants.
DAL-C	$10^{-5}/h$	Major	Significant reduction in safety margin or functional capabilities; significant increase in crew workload or in conditions impairing crew efficiency; some discomfort to occupants.
DAL-D	$10^{-3}/h$	Minor	Slight reduction in safety margin; slight increase in crew workload; some inconvenience to occupants.
DAL-E	N/A	No effect	None.

Table 1.1: Description of the Design Assurance Levels from the ARP-4761 [SAE, 1996].

The concept of criticality in the avionics context is different from the theoretical concept of criticality found in literature's mixed-criticality systems [Vestal, 2007]. While mixed-criticality is seen as a "hot-topic" and new analysis and models are researched today, the criticality defined in the industrial context stayed unchanged for years under the strict supervision of the certifications authorities. Further details

regarding the differences of these two context will be presented in the State of the Art chapter (see 2.5.3).

Software development is one of the airplane development components that has encountered a continuous evolution. For the Airbus fleet, new technologies were added regularly in order to improve the safety, fuel consumption, cost savings and reliability. Airplane cockpits have changed dramatically in the last 20 years, and many piloting facilities have been introduced. As an example, between 2011 and 2016 in the Airbus A330 airplanes were introduced functions like Airborne Traffic Situational Awareness (ATSAW), Traffic Collision Avoidance System Resolution Advisory (TCAS RA), Autopilot/Flight Director Traffic Collision Avoidance System (AP/FD TCAS), On-board Airport Navigation System (OANS), GBAS Landing System (GLS), Flight Management Landing System (FLS), Continuous Descent Approach (CDA) and Runway Overrun Protection System (ROPS). All these functions increase the total number of instructions that a board computer has to execute. At the same time the computation need of the system raises and consequently the number of computers on board as well as the weight of the aircraft.

In order to avoid the aforementioned problem, aircraft manufactures introduced the concept of **Integrated Modular Avionics (IMA)** which allows multiple software system parts to be hosted on the same execution target. This comes as an answer to the conventional avionics where, for a given system, each supplier responsible for the development of one or several functions provides a computer. A precise description of the IMA concept can be found in the State of the Art chapter (see 2.5.1).

WCET in the certification process

Integration of several systems on the same hardware implies strict verifications and validation in the process of certification. Some of the constraints of the DO-178C do not only change the development process but also change the actual design choices of the system overall. For example, DO-178C requires that applicants compute the **Worst Case Execution Time** of software programs. Computing non pessimistic WCETs is a difficult task on modern processors. For custom made processors this can be achieved with a lower grade of difficulty, but for COTS hardware, where the producers focus more on performance and average execution time than on safety and WCET, this task seems cumbersome if not impossible. New techniques for timing

analysis have been considered recently, out of which the ones using probabilistic methods will be further presented in this thesis.

The WCET related constrains are found in the section 6.3.4 of the DO-178B/C standard. The objectives to be achieved by a software in order to be certified are "accuracy and consistency". In the context it was conceived (in 1992 for the DO-178B), when most of the code was written in assembler, this objective is a pertinent one. But in our days, when software development is highly partitioned between different providers, each proposing its own writing language, techniques and tools, the verification of accuracy and consistency may not be so obvious.

In the introduction of the same section (6.3), a phrase suggests that reviews and analysis alone cannot totally deal with certain subjects (e.g. WCET and stack analysis). The text pointing to this is the following:

"There may be cases where the verification objectives described in this section cannot be completely satisfied via reviews and analyses alone. In such cases, those verification objectives may be satisfied with additional testing of the software product. For example, a combination of reviews, analyses, and tests may be developed to establish the worst-case execution time or verification of the stack usage."

With the fast evolution of technologies, the avionics industry is facing the pressure of keeping the pace and each change in the software or hardware is subject to a new process of certification. The standard requires producers to assess for impact of WCET any modification in the compiler, linker or hardware.

One could note that the objective is not to determine the real WCET, or a precise upper bound, but simply to verify that the maximum execution time is consistent with the allocated time, and that the documented timing constraints (i.e. requirements) identified by the development process are verified.

Frequently, the high-level timing requirements are expressed as processing periodicity or end-to-end latency for reacting to external events. These requirements are usually answered, in the Dynamic Design phase, by allocating system functions to periodic tasks, of adequate periodicity. Most frequently, and specifically in Control-Command functions, the system requirements do not tolerate timing overrun. The software tasks are therefore designed to detect and apply some sanction when a deadline miss is observed. Generally the sanction is to kill (maybe restart) the fail-

ing task or the whole application (or partition). A WCET analysis is requested each case where a processing overrun would produce unintended results (e.g. undetected overrun) or unavailability (e.g. detected overrun and sanction).

One could note that the DO-178 does not state or recommend a method to determine the WCET. In the DO-258B is a FAQ document completing the DO-178:

FAQ#73: Are timing measurements during testing sufficient or is a rigorous demonstration of worst-case timing necessary?

R: The worst-case timing could be calculated by review and analysis of the source code and architecture, but compiler and processor behavior and its impact also should be addressed. Timing measurements by themselves cannot be used without an analysis demonstrating that the worst-case timing would be achieved, but processor behavior (e.g., cache performance) should be assessed. Using the times observed during test execution is sufficient, if it can be demonstrated that the test provides worst-case execution time.

The DO 258 considers that the source code analysis is the primary source for the worst case evaluation. The WCET identification is primarily a question of identification of the "worst case scenario". But, the compiler and processor, hardware in general (but foremost the caches and the memory latency) must be taken into account. Essentially, measurements by themselves could be sufficient if one can demonstrate that the observations are representative of the worst case scenario.

1.2 Context

Despite the fast development of technologies, the main industries dealing with safety related machines seem to be reticent into embracing new and innovative solution for their products. The three great industries to which we make reference are the avionics, space and automotive. The automotive industry is fairly open to new approaches, benefiting of the fact that there is no certification authority for cars to regulate development as in the case of airplanes. Even in these conditions, the late start given by the introduction of engine control units (ECU) and the great number of users globally makes the technological acceleration slow.

In the case of satellite industry, the long duration of space programs obliges the constructors to treat the development phase with high concern for durability and

less for performance. This duration and the high costs of a space program push back last moment innovations.

As presented in the previous section, the main stop in assimilating new technologies for the avionics industry is represented by the certification authorities. This conservationist driven experts favor passenger safety over innovation.

In the following we will present the actual technological context, the innovations proposed in the research in the last years and the way they are assimilated by the industry.

1.2.1 Performance race

Most of today's real-time theory was conceived for single-core systems, but in time the performance requirements and the source code size of applications increased and forced industries to consider more complex architectures that contain multiple computational units. Industries from the safety-critical domain such as the avionic, automotive, space, healthcare or robotic industry are dealing with exponential needs in performances and functionalities and this is steering them towards commercial-of-the-shelf (COTS) solutions that are able to offer a better average performance. The disadvantages of these architectures are that they are not predictable and that the worst case execution time does not necessary decrease with the average execution time. Figure 1.4 [Bin et al., 2014] is a good echo of the way applications and platforms are evolving.

The notions regarding actual platforms are vast and still produce confusion for those that are not experts in this branch. This is why we give a short and non-exhaustive classification and definition of actual processing platforms.

The cellular component of any real-time system is the core, an independent processing unit that reads and executes program instructions. A processor is the name usually given to a Central Processing Unit (CPU), which is the electronic circuitry within a computing system that carries out instructions of a computer program by performing the basic arithmetical, logical, control and input/output (I/O) operations specified by the instructions. A processor contains many discrete parts within itself, such as one or more memory caches for instructions and data, instructions decoders and various types of execution units for performing operations. Every CPU contains at least one core. Depending on the number of cores or processors that a system has, it can be one of the following:

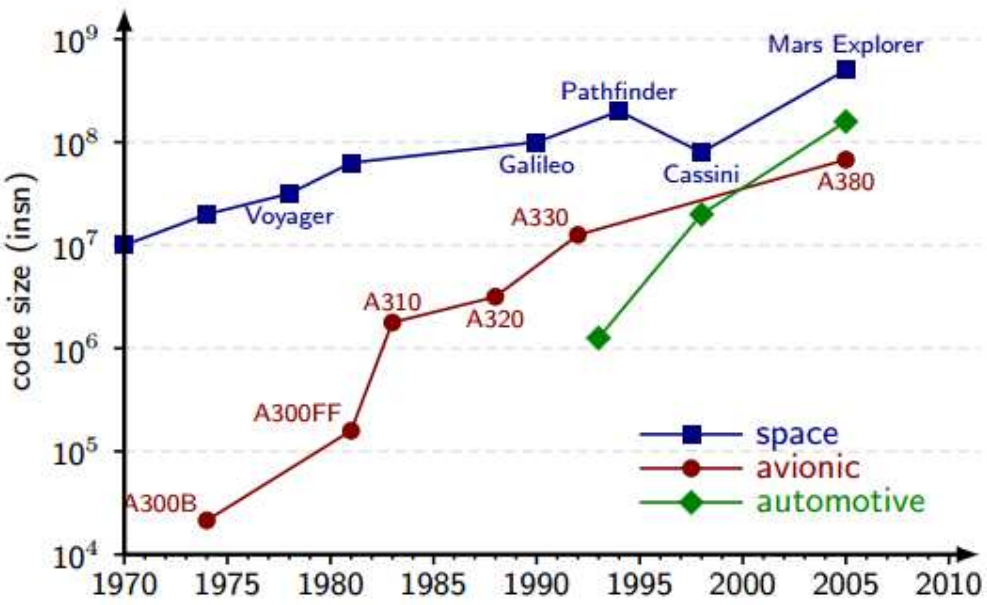

Figure 1.4: Evolution of code size in space, avionic and automotive embedded systems.

Single-core - is the name often use for processors with only one core instead of the longer and complete name of single-core processor. This is the architecture on which were designed the majority of algorithms and theories from real-time systems.

Single-processor - designs a computation system that includes a single processing chip (can include one or more cores). We then have single-core single-processor or multi-core single-processor systems.

Multi-core - represents a CPU that has multiple cores. This basically means that there is a multiplication of certain CPU components according to the number of cores. This permits the cores to work in parallel in separate operations by realizing chip-level multiprocessing.

Multi-processor - is a system that contains more than one CPU that are able to work in parallel in a simultaneous multiprocessing manner. According to the specification of composing CPUs, we may encounter single-core multi-processor or multi-core multi-processor

Many-core - is the name given to systems that have a number of cores in an order of tens or hundreds. These systems are made for highly parallelized applications

since the use of single threaded software would be slow on such equipment.

Presenting the different models existing is not the purpose of this chapter, but some of them will be detailed in future chapters as part of used architectures for the experimental part of the thesis.

Recently, a clear shift towards parallel processor can be observed. Since the 2004 when the multi-core processors were introduced, code writing and execution mindset have changed pushing developers towards a higher parallelization of applications. In multi-core processors, the computing cores are interconnected by a shared bus or a crossbar, which increases the computational power of the architecture but at the same time introduce uncertainty in the exact tracing of execution programs. Another issue regarding the WCET computation on multi-core processors appears with the usage of multiple levels of caches. Despite their clear advantage in performance, the cache memories have the undesired property of making WCET analysis harder. The use of performant cache placement/replacement protocols makes it hard for the users to identify the WCET of its executable by measurements. On the other hand, an exact analysis of the cache evolution for a program is difficult and costly. Under these circumstances (a deactivation of cache memory is not desirable) a timing analysis based on safety margins is the common practice. The main question that rises is:

How is the WCET safety margin selected?

1.2.2 Execution time

In critical real-time systems, the time factor has high importance. Understanding the worst case timing behavior of software in such systems plays a key role in reliability or correct functional behavior. The worst case execution time (WCET) needs to be guaranteed for the process of creation and verification of schedules. In research, a considerable part of publications presumes the existence of reliable WCET or of an exact bound. Even though this allows to conceive exact solutions for existing scheduling problems, in reality, most of the work done with such presumptions cannot be used in practice. As an example, the earliest deadline first (EDF) [Liu and Layland, 1973] algorithm has been proven to be optimal on preemptive uni-processors but its use in practice is still scarce due to the difficulty of implementation of such an algorithm. Providing a pessimist WCET bound to the EDF algorithm eliminates the advantage brought by its optimality.

Historically, obtaining the WCET bound has been done by following two different methodologies. On one side, **measurement** is widely used in industry while on the research front the **static analysis** is the method of choice. Both approaches have their own advantages and disadvantages and selecting one over the other one, when computing WCET bounds, is usually the result of a practical compromise in which the cost, the difficulty of implementation and the certification play a major role.

Measurement is the most used method for determining WCET bounds due to its lack of complexity and due to historical reasons in industry. The way measurements are carried in practice is by either determining the worst case input of the software under analysis or by running as many inputs as possible in order to exhaustively obtain the longest execution path. Sequentially, the software is executed on the target hardware and measurements are taken. Upon the criticality of the system a safety margin will be added to the highest measurement obtained.

All these measurement steps deal with some issues. Firstly, determining the worst case input of an arbitrary program turns up to be hard to achieve. The complexity of the program, the predictability of the hardware and the knowledge of the initial system state of the platform need to be taken into account while finding the worst case input. Alternatively, running all the inputs is impossible for a program where the number of input combinations is big. This number gets to huge values rapidly even for relatively small number of inputs. As an example, for n variables of size 32 bits the number of necessary measurement runs will be 4294967295^n. Therefore, a mid-way between the described approaches is usually preferred in practice. A sufficient great number of inputs are chosen to be executed depending on an pre-establish testing scenario that takes into account extreme cases and corners. In reality, despite all efforts at developing level or testing level, there is no guarantee that the largest execution time measurement is yielded by the worst case path or that all the precautions taken will guarantee that the worst case path will be taken.

The second issue to deal with while taking execution time measurements is the precision. Depending on the system under analysis, the measurement can be achieved by using software methods, hardware methods or a combination of the previous two. On the software side, the use of simulators is often encountered while doing timing analysis. Developing a simulator is proving to be a challenging task for nowadays' hardware, when its correctness cannot be proven or even reached.

Some simulators concentrate on specific features of the system studied and even if precise execution time for scenarios using those features can be determined, a complete analysis cannot be possible. Another method to measure execution time is the use of operation system clocks, by calling functions like `time`, `date` or `clock`. This approach is reliable under the condition that the analyzed system is using an OS that has precise hardware timing facilities. In the case of programs running on bare metal, this method is not an option. High-water marking is also a solution for capturing high execution times. This method consists in keeping the system running for a long period of time and recording the execution times observed per task at exact moments in time. While using this approach, the system can enter in a repeating cycle allowing the user to see only a limited number of execution time values per report of the total possible values that the system can produce. If we consider the effects of cached data, then the possibility of registering the WCET seems far fetched. Some tools that combine hardware with software for obtaining measurements are: oscilloscopes, logical analyzer, in-circuit emulators or processors with debug support. Figure 1.5 depicts the average behavior of execution time in the context of WCET and best case execution time (BCET). The interval between the highest observed execution time and the real WCET stays usually unknown for complex systems. Even using the input that will access the worst case path does not guarantee the encounter of the WCET. In the context of figure 1.5 this will be reflected in a shift of measurements towards right (the WCET) and a reduction of the interval in which the high-water mark can be found.

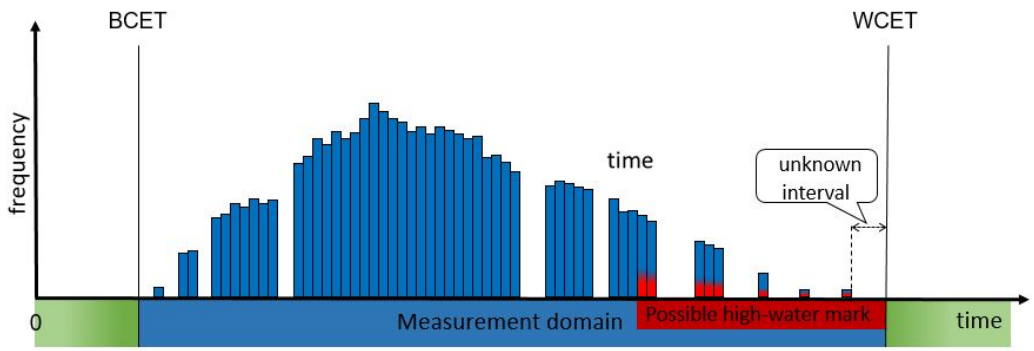

Figure 1.5: Possible execution time in the context of WCET bounds.

Static WCET analysis has been intensely studied and occasionally applied in embedded systems. Its core principle is the use of models based on the properties of the software and the hardware. This approach tends to be pessimistic by providing WCET bounds over the real WCET. This is happening because static analysis chooses the worst possible states at any certain moment. Therefore, in the case of loop bounds, the safe choice will be the one that will be hardly seen during execution. Input data dependencies are usually ignored by static analysis even though their existence will reduce the WCET. A great importance when doing static analysis must be given to the correctness of the model. For a complex program, finding the worst case input vector is a sensitive subject, as previously described for the measurement approach. Modeling with precision the platform on which the analyzed system runs is becoming harder and harder due to the complexity of the hardware, specially in the case of COTS components for which producers provide little to none information. Overall, the WCET bound estimated through static analysis is considerably greater than the measurements, in particular when performance enhancement technologies are used, like CPU pipelining, branch prediction or caches.

Hybrid methods that combine static analysis with measurements consist in partitioning the code into smaller parts through instrumentation. Specific instrumentation points (ipoints) are generated for code parts and based on the traces measured at these ipoints a flow analysis can be developed for each part of the program. Combining the program parts taking in consideration the obtained flows and the time execution distributions generates a WCET estimation. The safety of such an estimation is once again questionable. For such an analysis, the issues encountered in measurements and static analysis can be observed. Moreover, the level of instrumentation used can be a factor of uncertainty that would need to be discuss in the context of certification.

A series of tools, academic and commercial, have been developed for WCET analysis, out of which some are currently used in industry with high confidence. We will further detail some of these tools and their functionalities in the state of the art (see 2.5.2).

1.2.3 Probabilities and statistics

During the last twenty years, different solutions have been proposed to time critical system designers through a pessimistic estimation of performances of the processors (thus increased costs) while using average time behavior processors. For instance, DARPA estimates that in 2050 the construction of an airplane with current solutions will require the entire defense budget of USA [Augustine, 1997].

In the real-time domain, the arrival of multi-core processors or many-core processors as well as the increased complexity of programs have made more difficult the estimation of the worst case execution times (WCETs) of programs. The existing methods may produce estimates that are too pessimistic for some systems. As a result, new analyses based on probabilities and statistics have appeared to cope with this complexity by taking into account the fact that large values of WCET may have low probability of appearance.

The disciplines of probabilities and statistics have fundamentally changed the way science is done and the way we think about our world. New areas of mathematics and science evolved with the appearance of notions like randomness or uncertainty. In modern computer science, software engineering, and other fields, the need of taking decisions under uncertainty arises. Probability and statistics have been historically used together and their existence is strongly linked. Nevertheless, each one of these two notions have their own definition and can be used independently of the other.

Probability is the measure of the likelihood that an event will occur. It is quantified as a number between 0 and 1, where 0 indicates impossibility of the event observed to happen and 1 indicates certainty. The probability theory is the branch of mathematics dealing with probabilities. Basic examples, like tossing a coin or throwing a dice are used to describe how probabilities work while complex discrete and continuous functions stay at the basis of probability theory and are frequently used to describe events occurring in the world. Probability theory is used in finance where applications in risk assessment or modeling rely on it. In marketing and behavior finance, being able to predict consumer behaviors can bring an advantage to companies which use probability theory. In industry, producers use reliability theory to determine the life capacity of their products, the probability of failure and ultimately to take decision regarding the product's warranty. In medicine, biology and ecology, probabilities are used to analyze trends and to determine the

evolution of certain disease spread or environment changes.

Statistics is a branch of mathematics dealing with the collection, analysis, interpretation, presentation, and organization of data. Statistical population and statistical models are notions used while applying statistics. The population can be a group of actually existing objects or a hypothetical and possibly infinite group of objects. A description of the characteristics of population can be obtained by analyzing the population, or extending the analysis results obtained for one of its samples. A statistical model is a class of mathematical model, which embodies a set of assumptions concerning the sample data from a larger population. The construction of a statistical model depends on the sample used and consequently on the way the measurements are done to obtain the sample.

After the apparition of computers, statistics encountered a rebirth. Fast computation allowed mathematicians to collect and process large volumes of data which conducted to a switch from linear to nonlinear models (such as neuronal networks). From the other point of view, the use of statistics in computer science gave birth to new techniques and solutions for difficult existing problems. In software engineering, statistics can be used to test and construct models of engineering components and systems, in quality control or to study repetitive operations in manufacturing in order to set standards and detect errors. The ability to generate a statistical model of the information under analysis stays at the core of big data domain and data mining techniques. In bioinformatics statistics are used to describe biological systems and to process the large amount of information that human (and not only) DNA contains. Other domains such as neuronal networks, simulation, economics, mathematics or arts use statistics and statistical tools.

Regarding the embedded systems and the real-time domain, a larger description of the use of probabilities and statistics will be detailed in the future chapters. In the state of the art we will mention existing work while in the contribution chapter we will present the statistical methods developed for WCET analysis and the results obtained applying them.

1.3 Thesis motivation

In the actual context, where critical real-time systems are invaded by multi-core platforms, the WCET analysis using deterministic approaches becomes difficult, if not impossible. The time constraints of real-time systems need to be verified in the

context of certification. This verification, done during the entire development cycle, must take into account architectures more and more complex. These architectures increase the cost and complexity of actual, deterministic, tools to identify all possible time constrains and dependencies that can occur inside the system, risking to overlook extreme cases. An alternative to these problems is the probabilistic approach, which is more adapted to deal with these hazards and uncertainty and which allows a precise modeling of the system.

In the context of real-time embedded systems of great size, the different activities of the system (tasks, message transmission, etc.) are executed on assemblies of heterogeneous resources (different technology microprocessors, different communication networks, etc.). In order to verify the time constraints, one must be able to model and analyze formally the platform's performances and the temporal behavior of executing activities. The need exceeds often the analysis of a sole resource in an isolated manner because certain activities rely on multiple resources. In this case an analysis of *end to end constraints* must be envisioned.

As a response to these problems, a certain number of *formalisms* have been proposed in last 15 years: holistic approach (York) [Tindell and Clark, 1994], the Symta/S tool (Braunschweig) [Henia et al., 2005], the event-stream model (Ulm) [Albers et al., 2008], trajectory models (INRIA/LRI), Real-Time Calculus (ETZ) [Simalatsar et al., 2011], latency constrains (INRIA) [Cucu et al., 2008], and others. These models offer a compromise between result precision and the complexity of different analyses, and are well adapted for deterministic verifications (e.g. bounds on the response time). From our knowledge, only the work done in [Santinelli and Cucu-Grosjean, 2011] considers probabilistic analysis. This result is an extension of real-time calculus and it inherits its pessimism.

Despite the existence of certain isolation mechanisms, *dependencies* between activities at execution can often be noticed due to shared buses, simultaneous accesses of input/output memory, cache memory access, preemptions, etc. These dependencies are nowadays particularly important in the multi-core context. The need of a theory dealing with dependencies of execution times and of concrete solution, seen in a statistical tool, can be useful to the real-time community in general and to industries dealing with critical real-time systems in particular. From our knowledge, there has been no work on this subject until now.

Therefore, under this context, with the presented needs and trying to propose

a probabilistic time analysis of execution times for critical real-times systems, the present thesis was proposed supported by a CIFRE scholarship and reuniting the Inria institute of research in Paris and Airbus Operations S.A.S company in Toulouse.

1.4 Model

In this section, we present the model used in the thesis, starting with the basic notations and definitions coming from the probabilistic domain and continuing with the formal representation of a probabilistic real-time system. We dedicate special attention to the definition of probabilistic worst case execution time, a notion that often induced ambiguity and misunderstandings in the real-time domain.

1.4.1 Useful notions from the probability theory

In order to better understand the model used while performing probabilistic timing analysis, we list the basic mathematical notations and definitions to be incountered along the remaining of this document.

Definition 1 *A **probability distribution** is a table or an equation that links each outcome of an experiment with its probability of occurrence.*

In this thesis we use the notions of probability distribution, distribution function, and "distribution" with the same meaning, for both continuous and discrete functions. As an example, the distribution of a single coin flip is represented by the probability that each side of the coin has to appear, in other words 0.5 for heads and 0.5 for tails. Despite the triviality of the example, a probability distribution can be derived for any natural phenomena surrounding us, and described in the form of random variables.

Definition 2 *A **random variable** is a measurable function from a set of possible outcomes to a measurable space.*

A random variable $\mathcal{X}$ has a probability function (PF) $f_{\mathcal{X}}(\cdot)$ with $f_{\mathcal{X}}(x) = P(\mathcal{X} = x)$. The possible values of $\mathcal{X}$ belong to the interval $[x^{\min}, x^{\max}]$. In this work we associate the probabilities with the possible values of a random variable $\mathcal{X}$ using the following notation

$$\mathcal{X} = \begin{pmatrix} X^0 = X^{min} & X^1 & \cdots & X^k = X^{max} \\ f_{\mathcal{X}}(X^{min}) & f_{\mathcal{X}}(X^1) & \cdots & f_{\mathcal{X}}(X^{max}) \end{pmatrix} \tag{1.1}$$

where $\sum_{j=0}^{k_i} f_{\mathcal{X}}(X^j) = 1$.

A random variable may also be specified using its cumulative distribution function (CDF) $F_{\mathcal{X}}(x) = \sum_{z=x^{min}}^{x} f_{\mathcal{X}}(z)$.

Definition 3 *Two random variables $\mathcal{X}$ and $\mathcal{Y}$ are (probabilistically)* **independent** *if they describe two events such that the outcome of one event does not have any impact on the outcome of the other.*

Definition 4 *[Lopez et al., 2008] Let $\mathcal{X}$ and $\mathcal{Y}$ be two random variables. We say that $\mathcal{X}$ is worse than $\mathcal{Y}$ if $F_{\mathcal{X}}(x) \leq F_{\mathcal{Y}}(x)$, $\forall x$, and denote it by $\mathcal{X} \succeq \mathcal{Y}$.*

For example, in Figure 1.6 $F_{\mathcal{X}_2}(x)$ never goes below $F_{\mathcal{X}_1}(x)$, meaning that $\mathcal{X}_1 \succeq \mathcal{X}_2$. Note that $\mathcal{X}_2$ and $\mathcal{X}_3$ are not comparable. It can also be observed that $F_{\mathcal{X}_1}(x)$ upper bounds the other two random variables.

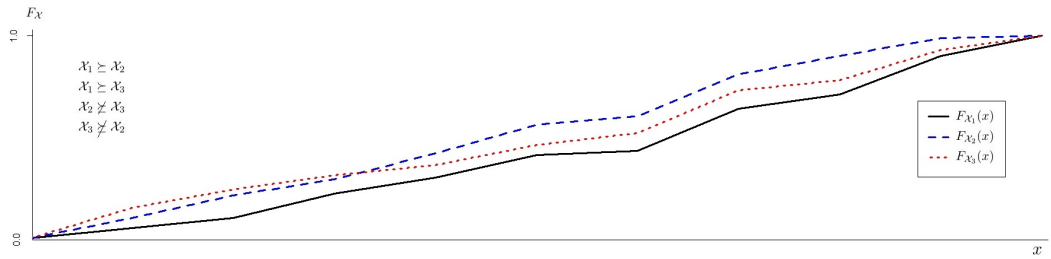

Figure 1.6: Possible relations between the CDFs of various random variables.

The term of "upper bounding" is counter-intuitive when used for cdfs, specially when the functions are plotted. The use of complementary cumulative distribution functions is a better choice when one is interested in the tail behavior of the distributions. This is the case while doing probabilistic timing analysis.

Definition 5 *The* ***complementary cumulative distribution function*** *(CCDF) of a random variable $\mathcal{X}$ represents the probability that $\mathcal{X}$ does not exceeds x. We note the CCDF of $\mathcal{X}$ as $\bar{F}_{\mathcal{X}} = 1 - F_{\mathcal{X}}$*

In Figure 1.7 we can observe the complementary cumulative distribution functions of the same random variables depicted in Figure 1.6. It can be observed that $\mathcal{X}_1$ curve is situated above $\mathcal{X}_2$ and $\mathcal{X}_3$ as the upper bounding variable.

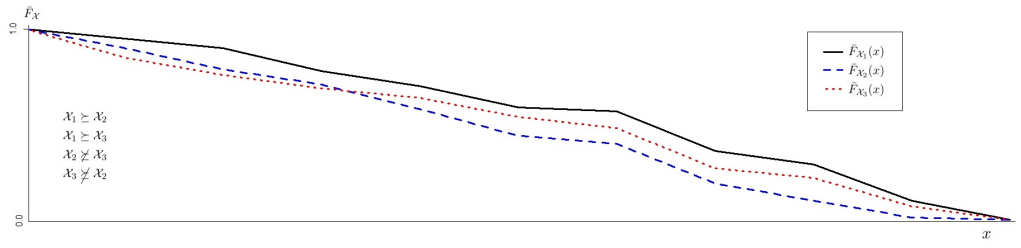

Figure 1.7: Possible relations between the CCDFs of various random variables.

The CCDF representation is chosen over the CDF one in the timing analysis performed in the following chapters of this thesis. This choice was motivated by the definition of WCET, which is a value that a program has low probability of achieving, and that approaches the bottom right most point in a CCDF distribution.

1.4.2 Probabilistic real-time system

We consider a system executing a task set $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ of n synchronous tasks processor according to a predefined scheduling policy. Without loss of generality, we consider that τ_i has a higher priority than τ_j for $i < j$. We denote by $hp(i)$ the set of tasks' indexes with higher priority than τ_i.

Each task τ_i generates an infinite number of successive jobs $\tau_{i,j}$, with $j = 1, \ldots, \infty$. All jobs are assumed to be independent from other jobs of the same task and those of other tasks.

A task τ_i is defined by the tuple (O_i, C_i, P_i, D_i) which represents the offset, worst case execution time, absolute period and absolute deadline. Inheriting from the generating task, a job $\tau_{i,j}$ is characterized by the parameters (O_i, C_i, P_i^j, D_i^j), where O_i and C_i are the task's offline and worst case execution time, P_i^j is the relative arrival time of the job ($P_i^j = O_i + (j-1) * P_i$) and D_i^j is the job's relative deadline ($D_i^j = O_i + j * D_i$).

Definition 6 *A* probabilistic real-time system *is a real-time system with at least one parameter defined by a random variable.*

In this thesis we concentrate on the probabilistic real-time systems (PRTS) having the worst case execution time defined as a random variable (probabilistic WCET) . Literature records papers dealing with systems having probabilistic inter-arrival

times or even multiple parameters described as random variables. These kinds of systems are out of the scope of our work. We dedicate the next subsection to the definitions of probabilistic execution time (pET) and probabilistic worst case execution times (pWCET).

pET and pWCET

Definition 7 ***The probabilistic execution time*** *(pET) of a job of a task describes the probability that the execution time of the job is equal to a given value.*

The pET can be derived from execution time samples in the form of a frequency function or computed from the task's characteristics. We emphasize that a pET is representative to an execution time profile of the task but not to the entire task. We consider an execution time profile of a task as the task's timing behavior under specific configuration of the system.

Each task τ_i is represented by a probabilistic worst-case execution time (pWCET) denoted by $\mathcal{C}_i$ defined as follows.

Definition 8 ***The probabilistic worst-case execution time*** $\mathcal{C}_i$ *of a task* τ_i *is an upper bound on the pETs* $\mathcal{C}_i^j$, $\forall j$ *and it may be described by the relation* $\succeq$, *as* $\mathcal{C}_i \succeq \mathcal{C}_i^j$, $\forall j$. *Graphically this means that the CDF of* $\mathcal{C}_i$ *stays under the CDF of* $\mathcal{C}_i^j$, $\forall j$ *(and equivalently the CCDF of* $\mathcal{C}_i$ *stays above the CCDF of* $\mathcal{C}_i^j$, $\forall j$*).*

The worst-case execution time $\mathcal{C}_i$ can be written as follows:

$$\mathcal{C}_i = \begin{pmatrix} C_i^0 = C_i^{\min} & C_i^1 & \cdots & C_i^{k_i} = C_i^{\max} \\ f_{\mathcal{C}_i}(C_i^{\min}) & f_{\mathcal{C}_i}(C_i^1) & \cdots & f_{\mathcal{C}_i}(C_i^{\max}) \end{pmatrix}, \tag{1.2}$$

where $\sum_{j=0}^{k_i} f_{\mathcal{C}_i}(C_i^j) = 1$.

For example for a task τ_i we might have a worst-case execution time $\mathcal{C}_i = \begin{pmatrix} 4 & 6 & 17 \\ 0.54 & 0.43 & 0.03 \end{pmatrix}$; thus $f_{C_i}(4) = 0.54$, $f_{C_i}(6) = 0.43$ and $f_{C_i}(17) = 0.03$.

We mention that following the same reasoning the probabilistic minimal inter-arrival time (pMIT) denoted by $\mathcal{T}_i$ describes the probabilistic minimal inter-arrival times of all jobs.

Hence, a task τ_i is represented by a tuple $(O_i, \mathcal{C}_i, P_i, D_i)$, or in its short form $(\mathcal{C}_i, P_i)$ when the deadline is equal to the period (the release time of a new job is equivalent with the deadline of the current job) and all the tasks start at the same moment $O_i = 0$.

Since the seminal paper of Liu and Layland [Liu and Layland, 1973] the *independence* of tasks is defined such that the *requests for a certain task do not depend on the initiation or the completion of requests for other tasks.* Moreover the schedulability analysis of independent tasks may be studied under the hypothesis that the tasks do not share any resources except for the processor. This hypothesis cannot be respected under current hardware architectures. We define the multiple notions related to independence when using probabilistic timing analysis in the contribution section (section 3.3).

Chapter 2

State Of The Art

In this chapter we present existing work on the domain of real-time systems that is relevant to the contributions of this thesis.

Aggressive hardware acceleration features like cache and deep memory hierarchies determined a wide variability of executions times. As a consequent, approaches of Worst Case Execution Time analysis and Worst Case Response Time analysis, not taking into account that large values are rare events, may produce results indicating deadline misses when, in practice, the probability of such an event occurring in the lifetime of a system is considerably small. Such over-pessimistic analysis may lead to the over-provision in the system architecture and to a reduction of the maximum number of functionalities that the system can include.

For a precise worst-case analysis the system parameters need to be considered in their worst case value or described using a safe upper-bound/lower-bound, which is not always possible. Also, there are cases when the parameters are not known until system runtime, when tasks get instantiated. For such cases, computing worst-case values becomes difficult or even impossible, while using safe bounds for the system analysis may introduce increased level of pessimism.

Another case where deterministic analysis may not be efficient is the case of event triggered systems which interact with the real world. For this kind of systems, it is not always necessary that an useful bound is placed on the arrival rate of jobs generated by interrupts from external sensors or network interfaces, which may not even have such a bound [Broster and Burns, 2004a]. The best known examples of systems with streams of jobs that arrive in a random fashion are controller area networks with faults and specifically faults generated by electromagnetic interference

(EMI) [Broster and Burns, 2004a], [Navet et al., 2000].

In practice, for the presented examples, system manufacturers are in the situation that they cannot certify a system because it is deemed infeasible by the worst case-execution analysis even though it is functioning correctly and without (or very little) faults and all experimentation, simulation and human experience confirm that it is in fact a feasible system. In the best case, manufacturers end up limiting the functionality integrated in the system, severely over-provisioning it.

An alternative approach is the use of probabilistic analysis. This is a research topic that gained ground in the last years, several works addressing the problem from different points of view. The use of probabilities in real-time systems can be also easy to grasp by the fact that systems' reliability is typically expressed in terms of probability for hardware failures, memory failures, software faults etc. An example in the time domain is represented by the maximum failure rate demanded in the avionics industry for DAL-A applications, which is the value of 10^{-9} per hour of operation. Probabilistic analysis techniques that seek to meet this requirement, rather than attempting to provide an absolute guarantee, have the potential to outperform deterministic techniques.

Probabilistic real-time systems and probabilistic real-time analysis are becoming a common practice in the real-time community, [Burns et al., 2003]. Papers related to this topic use different terms like stochastic analysis [Gardner and Lui, 1999], [Kaczynski et al., 2006], [Díaz et al., 2002], probabilistic analysis [Tia et al., 1995] or statistical analysis [Atlas and Bestavros, 1998], [Cucu-Grosjean et al., 2012] to indicate usually that the considered Critical Real-Time Embedded Systems (CRTES) have at least one parameter defined by a random variable.

In the following of this chapter we present existing work combining the real-time domain with statistics and probabilistic. The structure of this chapter is given by the impact of the probabilistic factor in the analysis or in the system being analyzed. We separate the timing analysis of systems containing a probabilistic component from the analysis that rely on probabilistic techniques, giving a short summary of such techniques. We enumerate the efforts done on the randomized architectures topic, and discussion on the precision of using probabilistic analysis on such architectures. In conclusion, we present the actual state of avionics industry and mention the work done in analyzing such systems.

2.1 Time analysis of probabilistic real-time systems

In this section we present the main timing analyses of probabilistic real-time system. The notion of probabilistic real time systems is used here in a wider context than presented in the model (see chapter 1.4) where one the system's parameter is described as a random variable. Instead, we include here results on systems or analysis methods that have a probabilistic component. The notion of stochastic is also used in the following description in concordance with the vocabulary employed by the cited authors. A stochastic process is a process with a randomly defined evolution. In mathematics the notions of stochastic process and random process are considered interchangeable. We can consider a stochastic process to be probabilistic if the probability function that describes that process is known.

Probabilistic real-time systems can be classified in three main categories according to the parameters that exhibit a random behavior. In the last years, systems with probabilistic *execution time* have been often analyzed motivated by their close resemblance to the real world. The *inter-arrival time* can also be modeled as a random variable giving birth to another topic of work. Systems with *multiple probabilistic parameters* are, for now, a theoretical exercise for researchers in the real-time domain.

The first paper introducing probabilistic distributions for the description of execution times of tasks had associated to large values of execution times low probabilities [Tia et al., 1995] as illustrated in Figure 2.1. In this work, the authors present an analysis for semi-periodic tasks, which means that they have periodic releases but their execution times vary. The analysis is called Probabilistic Time Demand Analysis (PTDA) and computes the worst case probability that a task in the system misses its deadline by bounding the total amount of processor time demanded by all higher priority tasks. The limitation of this work come from the pessimism of the analysis and from the limitation of the number of random variables to be combined. The pessimism is brought by the bound put on the processor time by the higher priority tasks. Due to the exponential explosion when combining random variables, the authors limited the analysis algorithm at combining maximum 10 random variables, while in the case of a larger number of release jobs the analysis is carried using the central limit theorem to approximate the response time.

In [Gardner and Lui, 1999], the authors present a stochastic analysis for real time systems that have probabilistic execution time. The analysis, called STDA

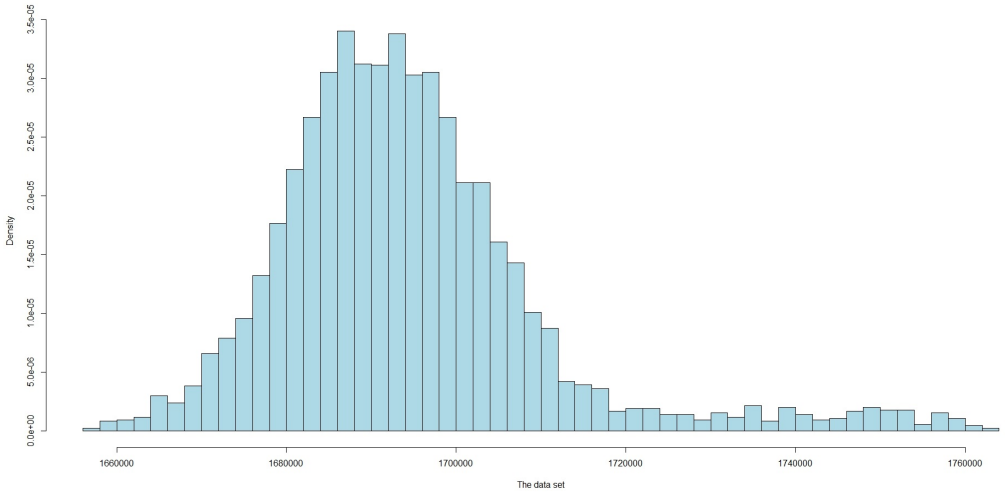

Figure 2.1: Distribution of execution times.

(Stochastic Time Demand Analysis), computes a lower bound on the probability that jobs in each task meet their deadlines. The proposed analysis is the basis for future developments, like the work of Diaz et al. [Díaz et al., 2002] who refined the analysis into an exact one.

In [Díaz et al., 2002], [Kim et al., 2005], [Lopez et al., 2008] the authors present an exact analysis for real-time systems that have random execution times. The execution time is represented as a general random variable and the priorities may be job-level or task-level. The analysis is proven to be bounded in time and exact for both cases when the system utilization is lower or greater than one. Due to the cost of convolution, the proposed analysis can be applied only for small task systems - this problem is later studied in [Refaat and Hladik, 2010] and [Maxim et al., 2012b]. Also, the system model on which the analysis can be applied is restrictive in the sense that, except for the execution time, it does not allow for other sources of variability, such as variable minimum inter-arrival time, variable deadline, etc.

In [Diaz et al., 2004] the authors further refine their analysis by bringing into discussion the concepts of pessimism and optimism, relations between two random variables, as well as truncating the tails of execution time probability distribution and moving probabilities from small values to large values of the execution time distribution, which is a pessimistic way of reducing the analysis cost, later known as re-sampling. The paper also treats the application of the pessimism concept

on the blocking in shared resources and on the priority assignment policy of the system under analysis. The authors also state that Audsley's' algorithm [Audsley, 1991, Audsley, 2001], which is optimal in the deterministic case, is also optimal in the stochastic case.

In [Refaat and Hladik, 2010], the authors propose a refinement of the existing stochastic analysis by means of re-sampling of the values in the worst case execution time distribution. That is to say, the worst case execution time distribution is reduced in size by selecting a subset of values to be kept from the original set of values, removing the unselected values and re-distributing their probability mass to the selected values. The proposed re-sampling techniques have two major shortcomings: *a)* the samples to be kept are randomly selected by assigning a selection probability to each values of the original distribution, and *b)* the probability mass of the unselected values goes entirely to the largest value of the distribution, the worst case execution time, thus greatly increasing the introduced pessimism, as opposed to re-distributing the probability mass to the values that are kept. These issues have been addressed in [Maxim et al., 2012b], by distributing the samples in a fair manner in order to keep reduce introduce a low amount of pessimism while reducing the number of values in the re-sampled random variable in order to reduce the computation needed to convolve the probabilistic execution times.

In [Burns et al., 2003] and [Bernat et al., 2005], the authors provide a probabilistic analysis framework for systems with tasks that have stochastic execution times, given as random variables following some probability distribution derived from measurement. The particularity of this work is the fact that the basic blocks which are analysed can be correlated. The correlations between blocks is solved by use of copulas, a mathematical tool that investigates dependence structures between random variables. When dependencies are not known, copulas produce upper and lower bounds of a joint distribution function of two correlated distribution functions such that it incorporates *any* possible dependences between the two distribution functions.

In [Manolache et al., 2004], the problem of uni-processor scheduling of tasks with stochastic execution times is studied. The scheduling policies taken into consideration are non-preemptive, and tasks have precedence constraints between them. The tasks' execution times are given as generalized probability distributions and assumed independent from one another, but the periods of tasks are assumed to

be harmonic, i.e., the period of one task is a common multiple of all periods of its predecessor tasks.

In [Hu et al., 2001], the authors are concerned about the feasibility of a system as a whole as opposed to the feasibility of separate tasks. The considered system is formed by periodic tasks that have probabilistic execution times given by random variables. Apart from these details, the description of the system under analysis may be improved, for example the algorithm for computing response time distributions is not presented, nor it is explained if jobs are discarded at deadline miss or if they are allowed to continue their execution - this specification has an impact on the nature of the analysis. Also, the motivating examples might not be valid, since the response time distributions presented are different than the ones that would be obtained by applying the analysis framework in [Díaz et al., 2002] and subsequent, which have been proved to be safe.

Multiple papers have been published on the topic of fault analysis in Control Area Networks (CAN) where the faults have a probabilistic arrival. We mention here, the work presented in [Axer and Ernst, 2013] which extend on the methodologies introduced by Diaz et al. [Díaz et al., 2002], [Kim et al., 2005], [Lopez et al., 2008]. Also, in [Navet et al., 2000] it is introduced the notion of Worst Case Deadline Failure Probability and present a probabilistic model for the errors that can occur in a CAN network, especially the errors caused by electro-magnetic interference which have a probabilistic nature. In [Broster et al., 2002] the authors present an analysis framework dealing with random faults on CAN. The faults arrive according to a Poisson distribution, but the analysis can only cope with a single stream of faults, i.e., all faults are equivalent, so they are considered as instances of the same process. In [Zeng et al., 2009b], [Zeng et al., 2009a] and [Zeng et al., 2010] the authors present different techniques, be it stochastic or statistic, to predict response times distributions of messages on a CAN based on simulation data on a reference CAN bus system.

In [Atlas and Bestavros, 1998], Rate Monotonic (RM) scheduling is extended in a statistical fashion in the sense that tasks have variable execution times and admission control of a job in the system is based on the likelihood of it finishing execution before its deadline. The conducted simulations show that Statistical Rate Monotonic Scheduling (SRMS) performs better than RM scheduling in overload conditions but not before overload occurs.

The problem of scheduling systems with probabilistic execution time is tackled in [Maxim et al., 2011] where an adaptation of Audsley's algorithm [Audsley, 1991] is done in order to optimally assign priorities to tasks. The authors also proposed a tree search algorithm for the Average Priority Assignment Problem (APAP) for which a greedy algorithm like Audsley's does not perform well.

The shift towards multi-processors architectures brought along analysis of probabilistic systems on such platforms. In [Manolache et al., 2002], the authors present a schedulability analysis for multiprocessor systems with tasks characterized by probabilistically distributed execution times. The parameters are having arbitrary distribution which are approximated to Coxian distributions for complexity reduction. In [Nissanke et al., 2002], the authors present a probabilistic framework for analyzing global performance issues in multiprocessor scheduling environments. The tasks are considered to have probabilistic execution times and probabilistic arrival times, and also the number of processors available at a time instance can be variable. The possible number of tasks in the system at a moment in time is computed as a probabilistic quantity, and the analysis is performed based on the execution requirements of each task and its laxity. The laxity of each task is computed based on the execution time and deadline, hence it is also a probabilistic quantity. The authors provide formulas for the computation of failure rates and success rates on a wide system scale, and for particular tasks.

In [David and Puaut, 2004] the authors provide a framework for obtaining the probabilistic execution times (pETs) of a program. This work uses static analysis to obtain probabilistic distributions of execution times, by associating to any path of a task a certain probability. In its initial form, the paper ignores the hardware on which system runs and concentrates solely on the task set that is analyzed.

The topic of probabilistic inter-arrival time is relatively a new research topic. Its lack of interest from the community is caused by the fact that industry did not confront itself with such behavior in real life. Recently, some designers started to introduce randomized events in their systems in order to avoid abnormal situations. As an example, some automotive manufacturers have randomized the sampling frequency for the reverse parking ultrasound sensor in order to avoid the situation when two vehicles reverse back-to-back and they both having the same sampling frequency reduces the efficiency of their parking sensors [Buttle, 2012]. By randomizing the sampling frequency, the jobs that are generated by the sensor have a

random arrival pattern. These jobs belong to a task that can be seen as a sporadic task with its period equal to the minimum inter-arrival time (MIT) amongst its jobs. Knowing that job arrivals are random, then we may describe the MITs by distributions, i.e., the arrival distribution of the generated jobs, which gives a more accurate description of the generated jobs.

In [Broster and Burns, 2004a] and [Broster and Burns, 2004b], the authors provide an intuition of how the random arrival model presented in previous work can be applied to fixed priority preemptive uni-processor scheduling. The main difference between faults arrivals in controller area networks and random job arrivals is that, regardless of their sources, the generated faults are all considered equivalent and so only one stream of faults applied in the analysis. In the case of one processor, the jobs with random behavior can be generated by multiple sources like external interrupts, network interfaces, etc, and have different characteristics. The previous analysis needs to be generalized in order to handle multiple streams of random arrivals.

In [Cucu and Tovar, 2006], a framework is presented for computing response time distributions in the case when in the system there are tasks that have random arrivals, given as independent discrete random variables. The rest of the parameters of the tasks are deterministic. The output of the analysis is the response time probability distribution of the first release of an analyzed task, considering that all tasks are released synchronously. The analysis is bounded in time, being polynomial in the value of the deadline. That is, the analysis stops when the job under consideration reaches values of response time that are equal to its deadline. As it is the case of the analysis proposed by Broster et all [Broster and Burns, 2004b, Broster and Burns, 2004a], this assumption restricts the system model to only those systems where jobs are evicted at deadline, and excluding the systems where jobs are allowed to continue execution even after their deadline.

The case of probabilistic minimum inter-arrival time is discussed in [Maxim et al., 2012a] from the point of view of the properties that such a parameter might have. An initial conclusion is that MITs exhibit a behavior characteristic to a Weibull distribution due to the short tail observed in the measurements.

In research, the exercise of multiple probabilistic parameters has been considered for the first time by Lehoczky in [Lehoczky, 1996]. The author presents an analysis for tasks that have arrivals according to a Poisson distribution process with rate λ

and exponentially distributed execution times. The paper does not deal with the case in which the parameters are described by another distribution and because of its tight presumptions it limits its applicability to more general task-sets.

In [Kaczynski et al., 2007] and [Kaczynski et al., 2006], the authors present an analysis for hybrid probabilistic systems which may include periodic, sporadic and aperiodic tasks. Tasks' execution times are given as execution time profiles (ETP) given as random variables. The sporadic and aperiodic tasks are considered to have arrival profiles (AP) as well given by random variables, specifically representing the number of task activations during a fixed time interval. All random variables' distributions are considered known. The solution proposed is based on a Polling Server extending the work presented in [Díaz et al., 2002]. A method for obtaining ETPs for servers used to encapsulate hybrid task-sets is developed and presented. The method is simulated and shown to have a high level of accuracy both compared to the worst case analysis and to the probabilistic analysis presented in [Díaz et al., 2002] where tasks' periods are considered as deterministic.

In [Abeni et al., 2012] and [Manica et al., 2012], the authors present an analysis framework for tasks with probabilistic execution times and random arrivals. The task system is running on a preemptive uni-processor according to a Constant Bandwidth Server based on Earliest Deadline First.

The analysis of a system containing probabilistic execution times, minimum inter-arrival time and deadlines is presented in [Maxim and Cucu-Grosjean, 2013]. This work relies on re-sampling to improve the complexity of the analysis that compute response time distributions of the tasks scheduled on one processor under a task-level fixed-priority preemptive scheduling policy.

We mention in this chapter, under the notion of probabilistic real-time systems those systems executing on platform having probabilistic caches (cache memory for which the behavior can be described as a random variable). For such systems, the technique called static probabilistic timing analysis (SPTA) is introduced in [Davis et al., 2013a] in order to analyze probabilistic cache related preemption delays (pCRPD), providing an upper bound on the 1-CDF of the probabilistic worst case execution time distribution function (pWCET) of a task while also taking into account the effect of one or more preemptions at arbitrary points in the tasks execution. In [Davis et al., 2013b] the problem of static probabilistic timing analysis is taken further to the case of multi-core processors where the existing SPTA is no

longer applicable due to the interactions that exist at cache level between cores. The authors formalize the problem and provide intuitions for several paths that could be employed to find a solution, but the problem is still open up to date. In [Hardy and Puaut, 2013], the authors introduce a static probabilistic timing analysis (SPTA) for systems with faulty caches. This analysis stems from the observation that the technological progress leads to system components that are more and more prone to failures. The precise component targeted by the mentioned work is the instruction cache, with future work directed towards other micro-architecture components with SRAM cells such as data caches and branch predictors. In this work, the cache uses the least recently used (LRU) replacement policy, and the only source of probability comes from the intrinsic probability of cache blocks failing. The probability of a permanent cache block failure is considered known and a failure actually happening in the systems implies the (permanent) disabling of the faulty block with an impact of the WCETs of the tasks in the system. The authors propose a technique to compute a probabilistic bound on the WCET in the presence of permanent faults. The technique is shown to be tight while remaining safe, due to the fact that it is based on static analysis, which is guaranteed to always find the longest execution path and hence the largest value of the execution time.

2.2 Probabilistic methods

For the case when the probabilistic law of the data under analysis is known, further techniques can be applied to determine properties of that data. Such is the case of computing deadline miss probability when knowing the random variable describing the execution times. In practice, the knowledge of the functions describing a system's parameters is approximated from the measurements, constructed from the system's functionality or even made up for the sake of research. This problem makes previous presented work hard to apply in real systems. Nevertheless, determining a pWCET distribution from a set of data is possible using extreme value theory (EVT). There have been a series of papers presenting and using this theory in the context of time analysis for real-time systems and we summarize the notable ones in the next section (see chapter 2.3). In order to better grasp the concept EVT without entering into details (see chapter 4), we present the basics of this theory and its use in science in general and in the real-time domain in particular.

Extreme value theory is a branch of statistics dealing with the extreme deviations

from the median of probability distributions. In other words, EVT provides the statistics of extreme events of a stochastic process. Extreme events are considered those events that have a behavior which deviates from the average behavior of the population. While the central limit theorem describes the probability law followed by average samplings, EVT consists of a set of well elaborated statistical theory for extreme values. Figure 2.2 gives a visual description of the place where extreme events can be found in distribution.

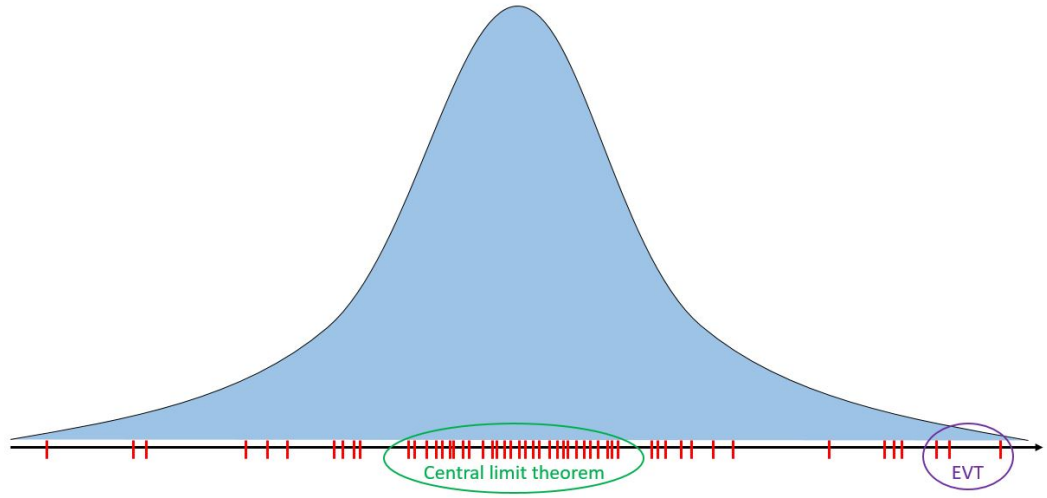

Figure 2.2: Space of interest of the central theorem compared to the extreme value theorem.

Extreme value theory has applications in various fields. In finance, EVT can be used to describe the distribution of income in an analyzed domain of activity or to predict the maximal daily lost in the value-at-risk analysis. In hydrology, EVT is applied to predict extreme floods in order to allow the authorities to install measures of protection. The result of such predictions is presented in the form of maximal flow expected once every 100 years.

The domain in which EVT development strove is the meteorology, where natural phenomena can be seen as events and the ability to better understand and predict their extremes can prove life saving. A few examples of such phenomena are: extreme winds, frosting, heavy precipitations, heat waves, hurricanes, droughts and extreme climate changing. The theory of extremes as it has been used in meteorology has been extended to the real-time domain for predicting worst case execution

times of analyzed programs for a certain probability. We detail in the following chapter (2.3) the main results on this theory.

In this section we mention only the work that modified the theory of extremes in the course of history bringing it to what it is now, a well elaborated statistical theory. For a detailed review of applications we advise the reader to consult [Faragó and Katz, 1990].

The theory of extremes is first formalized by Fisher and Tippett in [Fisher and Tippett, 1928]. The authors resemble the distributions of Gumbel, Fréchet and negative Weibull to describe the maxima for a single process. Previous researchers, as Fuller in 1914, have used some of these three distributions in applications without a precise formalization. According to Kotz and Nadarajah [Kotz and Nadarajah, 2000], extreme value distributions can be traced back to work done by Bernoulli in 1709. The classification of the three laws of distributions have been further refined and consolidated in [Gnedenko, 1943], [Haan, 1970], [Haan, 1976] and [Weissman, 1978]. For ease of notation, in the rest of this thesis, we use the term "Weibull" to describe the negative Weibull distribution.

In [Leadbetter et al., 1983], the authors provide the necessary and sufficient conditions for a set of independent and identically distributed (i.i.d.) random variables with common cumulative distribution function (cdf) to belong to the domain of attraction of one of the three extreme value cdfs (Gumbel, Fréchet and negative Weibull). Based on these conditions, multiple publications appeared in which specific sequences of iid observations ae tested in order to find the domain of attraction of the three distributions they belong to. For the Gumbel type we mention [de Oliveira and Gomes, 1984], [Marohn, 1998a] and [Marohn, 1998b]. The Weibull type is tested in [Tiku and Singh, 1981] and [Shapiro and Brain, 1987]. More general tests have been provided in [Galambos, 1982], [Öztürk and Korukogu, 1988], [Castillo et al., 1989] and [Hasofer and Wang, 1992]

Split according to the three types of distributions, we list some of the applications in which EVT is used. Therefore, the Gumbel distribution has been applied to fire protection and insurance problems and the prediction of earthquake magnitudes, to model extremely high temperatures, and to predict high return levels of wind speeds relevant for the design of civil engineering structures. The Fréchet distribution has been applied to estimate probabilities of extreme occurrences in stock index and to predict the behavior of solar proton peak fluxes. The Weibull

distribution has been used to model failure strengths of load-sharing systems and window glasses, for evaluating the magnitude of future earthquakes, for partitioning and floorplanning problems, to predict the diameter of crops for growth and yield modeling purposes, for the analysis of corrosion failures of lead-sheathed cables at the Kennedy Space center, to predict the occurrence of geomagnetic storms and to estimate the occurrence probability of giant freak waves in the sea area around Japan.

The three types of distributions can be combined in a single distribution called generalized extreme value theory (GEV). For a given set of data, the block maxima method is used to obtain the random variables fitting to the GEV distribution. This method consist in choosing the maximum values from a series of equal time interval (the annual maxima in the case of seasonal events). Most of the work done on this topic took place in the 90's having researchers like Dietrich, Hüsler, Dupuis or Castillo as contributors.

In [Balkema and De Haan, 1974] and [Pickands III, 1975], the authors evidentiate the link between GEV distribution and the survivor function of the Generalized Pareto (GP) distribution. This distribution can also be used in modeling the tail behavior of a given distribution. This is done using the peak over threshold method in which all the values over a certain threshold are chosen in order to identify the generalized pareto distribution parameters for the data under analysis. Some practical applications of GP distribution include the estimation of the finite limit of human lifespan, the modeling of high concentrations in short-range atmospheric dispersion and the estimation of flood return levels for homogeneous regions.

For a complete introduction in the domain of extreme value theory and an practical guide for applying the existing techniques regarding EVT, we guide the reader to Cole's book "An introduction to statistical modeling of extreme values" [COLES, 2001].

In the real-time domain, both GEV and GP distributions have been used to model the behavior of execution times. We present in the following sections the notable papers on this subject.

2.3 Measurement based probabilistic analysis

In critical real-time systems (CRTS), being able to predict the behavior of all components is very important. This is why, most of the manufacturers using CRTS

favor basic hardware for which well written software is easily analyzed using static methods. In such cases, the execution times from static analysis and the ones obtained by measurements tend to have similar values, with a slight pessimism on the static side. With the evolution of hardware and the multi-core platforms piercing in the embedded systems industry, this similarity disappears giving place to a big discrepancy between static analysis results and measurements. On the measurements side, new technologies, concentrating in increased average performance, generate execution time distributions shifted to the left compared to the real probability law of the system, as a result of optimism given by the platform. On the other side, static analysis may overestimate the WCET producing values much higher that the real WCET of the system, or even fail to compute the WCET due to the high complexity of the platform and/or the program. The trivial method of increasing the highest execution time obtained by measurement with a certain percentage might not be safe, especially when the value of that percentage is decided upon from historical reasons on the simple justification that "it always worked like that". Finding a mathematical computed upper bound of a system's WCET from measurements came as a logical solution. This has been done in the last two decades using the statistical methods and probability theory. The same way the central limit theorem is used for modeling average behavior of events, the extreme value theory can be used for modeling extreme events such as high execution times.

Doing a parallel between meteorological phenomena and a system's execution time, we can consider the observed execution times as the meteorological measurements in a given period of time, the highest execution times (including the high water-mark) as the rare events happening in the given interval of time, and the WCET as the most extreme event which meteorologist try to predict in the future. Such meteorological phenomena that fits the comparison can be heavy rain, storms, floods or drought. Similarly, earthquake can respect the aforementioned criteria. For all this natural events, extreme value theory (EVT) is used to predict critical events and its corresponding probability of happening.

The first papers proposing EVT as a solution for analysis real-time systems appeared in the years 2000-2002 from University of York. In [Burns and Edgar, 2000], the authors consider for the first time determining the system model using extreme value theory, inspired by the inaccurate approximation of the Gaussian distribution on observed data. In this paper only one example is used in the experimental section

and the generalized extreme value distribution is used as a targeted distribution. The premises of such a modeling are not specified and no statistical test that the analyzed data need to pass are mentioned. Moreover, the GEV parameters are estimated from the raw data and as a result the pWCET estimation of the given system does not take in account dependencies coming from the periodicity of the system.

In [Edgar and Burns, 2001], the Gumbel distribution is chosen for modeling the pWCET curve and further statistically estimate the feasibility of a task set. Selecting only one of the three EVT distributions is not correct in the context in which there is no formal proof showing that the behavior of execution time of all systems in the worst case context is modeled as that exact distribution. Indeed, from our knowledge, there is no proof of the fact that the extremes in a execution time sample it always fits on a Gumbel distribution. Even more, in the contribution section we present practical examples for which the Weibull or the Fréchet distributions are the ones obtained as fitting distributions for the observed data. As in [Burns and Edgar, 2000], the discussed paper uses the raw data to be fitted on the EVT distribution instead of using a method for maxima picking as the theory exceeds. Even though the level of confidence for the obtained results is calculated, no goodness of fit test is used to support the Gumbel fitting.

A detailed work on probabilistic methods can be found in Edgar's Ph.D. thesis [Edgar, 2002] which extends on the previous mentioned papers and inherits the same weaknesses: use of Gumbel instead of GEV, fitting of raw data instead of using a maxima selecting method, generation of models with distributions not exceeding the probability of 10^{-5} or limitations of EVT on statistical independent data. This work has a great role in the domain of probabilistic analysis of real-time systems, being the fist work that consider the idea of probabilistic worst case execution time despite its formalization in future publications.

The issue of fitting raw data on EVT models encountered in Edgard's work is corrected in [Hansen et al., 2009] with the use of block maxima method and the use of EVT in its original design. The use of a goodness of fit (GOF) hypothesis test is also considered in this work. The paper adapts the analyzing methods to produce results in the form of probability of failure in a given time period, which is inline with the results obtained in the testing community. The main issue with this work is represented by the way the block size is chosen for the block maxima method. An iterative process in which the block size is doubled and the results are verified

using the GOF test is used for determining the GEV parameters to be used for the pWCET distribution. This method suffers from lack of accuracy and demands a high number of observations for obtaining a reliable model.

In [Maxim et al., 2016], we propose a series conditions that a system under analysis has to respect such that the pWCET estimation using EVT to be reliable and certifiable. Therefore, the concepts of reproducibility and representativity of the measurement protocol are introduced. It is considered that a measurement protocol with such properties is capable to produce data samples with the same statistical properties as the real WCET distribution of the system. A detailed description of these concepts can be found in contributions chapter of this thesis.

In the years following the introduction of pWCET estimation, several authors reported problems regarding the use of EVT in timing analysis. The problem of realism in statistical analysis of WCET is tackled in [Griffin and Burns, 2010], where several misconceptions about EVT are listed. The authors point out the capacity of EVT to produce a continuous distribution of pWCET for systems containing a finite number of states and, as a result, a discrete distribution of possible execution times. This is proven not to be a real issue for statistical timing analysis in [Cucu-Grosjean et al., 2012], but rather a common misunderstanding in the real-time community where the clear difference between pET and pWCET is still to be done in 2010. Another issue spotted in [Griffin and Burns, 2010] is related to the necessity of EVT to use i.i.d. samples when dependency often arise from the hardware features. A confusion between functional dependencies and statistical dependencies is made when discussing this issue. Both [Edgar, 2002] and [Maxim et al., 2016] present a clear difference between these two notions and while the second one is a requirement for EVT use the first one is not. Even more, the use of block maxima in the case of GEV has the role of eliminating direct dependencies, while new statistical method are able to analyze dependent data using the GP distribution.

In [Lima et al., 2016], the use of Gumbel distribution is challenged and the suggestion of GEV usage is made. The GEV distribution has already been used before in papers like [Cucu-Grosjean et al., 2012] or [Wartel et al., 2013], but because most of the analyzed data is fitting a Gumbel the use of GEV is not made clear. A analytical look on randomized cache is done in [Lima et al., 2016] leading to the conclusion that randomization is not necessary for the use of EVT based analysis, but in systems with low execution time variability it can be helpful.

For systems having a small number of states due to fully deterministic platforms and/or a software with a relative number of paths, EVT can hardly be used because of the law variability of the observed execution times. To combat this problem, [Lima and Bate, 2017] proposes the IESTA method which artificially injects variability in the form of a known distribution in a execution time sample in order to allow the EVT methods to model the system's behavior. Rendering systems with low variability analyzable comes with the costs of confidence loss, in other words the timing analysis using EVT on an IESTA processed sample produce a upper bounding area for the pWCET instead of a curve.

In the last decade, the rising interest in real-time systems evolution gave birth to multiple research projects. In the context of this thesis we mention the PROARTIS European project [Proartis, 2013] and its follow up project, PROXIMA [Proxima, 2016]. The goal of Proartis project is to develop new tools and techniques that would make timing analysis of critical real-time systems approachable on platforms using faster computer hardware features. The mean of achieving this goal is the use of statistical techniques based on probability theory in order to deal with the system's uncertainty. During the project, randomization of cache accesses to memory is proposed as a condition for the use of EVT. This premise may be useful but not necessary [Lima et al., 2016]. The platform used in Proartis is the Leon3 FPGA board which perfectly suited the use of hardware randomization despite its practical limitations for the industrial partners. Most of the theory and techniques proposed in this project apply for single-core architectures, even though the project had as a clear purpose to make the transition from single-core architectures to multi-cores ones. In the context of Proartis, notions like measurement-based probabilistic timing analysis (MBPTA), statical probabilistic timing analysis (SPTA), time composability or hardware randomization has been used or developed, and a series of papers have been written where EVT is used for timing analysis.

In [Cucu-Grosjean et al., 2012], MBPTA using EVT is proposed as a alternative choice for computing WCET bounds in industrial applications. The advantage of this technology is the fact that a mathematical proof of the WCET bounds is given unlike the common practice of adding an ad hoc percentage of the maximum observed execution time as a safety margin. The presented technology is tested on the EEMBC benchmark suite [Poovey et al., 2007] as a first step before its use on industrial applications. The authors also propose the GOF exponential tail test

as an alternative for the Chi-Square test which performs incorrectly for extreme values. In the paper we can find an algorithm that computes the minimum number of times a program needs to run in order to produce a sufficiently large sample of execution times to be fed to the timing analysis. This algorithm is later discarded due to its questionable necessity. It is true that industrial designers prefer to reduce the amount of time used in the testing phase, but the time spent on executing the analyzed systems is a small price to pay for an increase accuracy of results. Even more, EVT is conceived to perform well even with reduced size data, due to the difficulty of observing extreme phenomena in nature, and therefore it can be applied to any execution trace size. Nevertheless, a larger trace increases the probability of observing maximas or unusual patterns in the system execution. In conclusion, the number of runs a system needs to execute is relative to the developer capacity of producing them and to the requested results' accuracy.

The MBPTA techniques are tested on a avionics application in [Wartel et al., 2013] as a result of AIRBUS collaboration in the Proartis project. In this paper extreme value theory is used for computing the pWCET distribution of a series of Arinc 653 applications running on randomized hardware. The novelty of this paper is the use of CRPS as a measure of convergence for the estimated distributions instead of a GOF test. The authors concentrate on the Gumbel fitting as a particular case for the samples used.

A full description of EVT applicability in the Proartis project can be found in [Cazorla et al., 2013]. Also, this work considers that the worst case execution time of a task obtained in isolation is not becoming larger in the presence of the operating system (OSs). Such operating system is called compositional. Besides the mentioned papers, centered around the probabilistic timing analysis, the Proartis participants propose papers on other topics as hardware randomization [Kosmidis et al., 2013b,Kosmidis et al., 2013a], static probabilistic timing analysis [Davis et al., 2013a], scheduling [Baldovin et al., 2013b] or the impact of operating systems on probabilistic timing analysis [Baldovin et al., 2013a]. A questionable presumption taken in this project was the fact that probabilistic timing analysis can not be achieved without randomization. We detailed in the next section 2.4 the background of this presumption and its necessity.

Proartis project is followed up by the Proxima project [Proxima, 2016], which extended previous work on multi-core platforms and on mixed-critical systems. This

project proposes improved timing analysis independently of the architecture used. Therefore, in Proxima there are used architectures like the FPGA Leon3 board from Gaisler, the AURIX board from Infineon, the P4080 PowerPC from Freescale and the MPPA Manycore from Kalray. From the timing analysis point of view, such an wide choice of platforms imposed the description of measurement protocols, the development of techniques that would take in account dependencies between and the definitions of new notions related to execution time representativity. From the architectures point of view, the presumption of randomization necessity was taken and, hardware randomization (cache and bus) is used for the FPGA architectures while software randomization is considered for COTS products.

From the work achieved in Proxima, we mention here the following topics: comparing static analysis with statistical methods in [Maxim et al., 2015], the review of SPTA in order to eliminate the optimism found in previous versions of SPTA [Altmeyer et al., 2015], the use of lossy compression as an alternative to SPTA [Griffin et al., 2014], the description of a framework for evaluating MBTA [Lesage et al., 2015b], the use of path coverage techniques in order to observe the execution path that produces the highest execution time and eventually the WCET [Kosmidis et al., 2014b,Ziccardi et al., 2015] and the development of SPTA analysis for programs containing multiple paths [Lesage et al., 2015a].

Another use of EVT in real-time system can be observed in [Lu et al., 2011], where a new sampling method called Simple Random Sample (SRS) is proposed in order to improve the block maxima method. The supposed benefit of this sampling method is to guarantee the independence of samples fed to the GEV parameter estimation. As in previous papers, only the Gumbel distribution is presumed for the WCET behavior and the inappropriate Chi-Square test is used to verify the goodness of fit. The same authors considered the use of evt for obtaining probabilistic worst case response time (WCRT) estimations [Lu et al., 2012]. The system used is considered to be a black-box and the data fed to the analyzing method is represented by response times, not execution time as seen in existing papers. The downside of this method is that the system needs to be built first in order to be analysed, at which point the estimates provided by the analysis are of little use, it can only confirm that the system is feasible but in order for the system to be build feasible decisions need to be made at the beginning of the design process otherwise manufacturers risk to end up in the situation of only building unfeasible systems and the analysis would

only confirm that the systems are unfeasible. The Gumbel only and Chi-Square test issues are inherited in this papers.

In [Liu et al., 2013], the authors use GP for estimating the probabilistic WCRT just as it is done in [Lu et al., 2012]. The paper proposes the use exponential distribution as a special case of GP (the shape parameter is 0) and compare the results with the estimations using the Gumbel distribution as it is done in Lu's paper. This method is proposed because it generates more optimistic results than the GEV method, but in the real-time context being optimist is more problematic than overestimating execution/response time. An estimation that doesn't upper bound the real WCET is more dangerous than an estimation that is too far above the real WCET. The applicability of the proposed method is restricted to independent data, which is harder to achieve for response time than for execution time and the peak over threshold method doesn't perform well in eliminating dependencies.

Lisper and al. [Santos et al., 2011] have studied the compositionality of the probabilistic measurement-based approaches. Such approaches require an independence hypothesis between the probability distributions in order to allow their combination by convolution operations. The authors of [Santos et al., 2011] have presented interesting results by indicating that the lack of independence has a low impact on the combination of two sequentially executed programs. Nevertheless the paper does not proceed at statistical testing of the independence of the execution times of the programs under study.

The topic of dependent data used by EVT is tackled in [Melani et al., 2013]. The authors give a characterization of dependencies and propose a method to decompose dependency with the introduction of notions like dependence distance and independence bound.

In [Guet et al., 2016] and [Santinelli et al., 2017], a statistical timing analysis tool called diagXtrm is proposed by the authors. In the fist paper can be found details on the features of the tool by presenting the i.i.d. tests used, the stationarity detection, block and threshold selection as well as GEV and GP parameter estimation. The second paper concentrates on the result accuracy and reliability by defining a number of hypothesis to be checked in order to verify if the result is valid. An initial testing of the theory that stay at the basis of this tool is done in [Berezovskyi et al., 2014] on execution times obtained on CUDA Kernels. The tool is tested on Graphics Processor Units in [Berezovskyi et al., 2016]. Even though

probabilistic timing analysis without human intervention is desirable, estimation of parameters for GEV and GP distributions is a process that can hardly be automatized due to the multiple parameters that can interfere in the process (sample size, maxima size, dependencies, stationarity, etc.) and the lack of monotonicity of the shape parameter evolution.

2.4 Randomized architectures

We discuss in this section the effect of imposed random behavior upon a system's architecture. A series of models and technologies are proposed that use randomization at one or multiple levels of the platform, (e.g. memory, bus or instruction). We present here the work done in the real-time domain and its impact on timing analysis.

The first paper suggesting the use of randomization in real-time systems is proposed by the authors of [Quinones et al., 2009]. The authors suggest the use of randomized caches replacement policies inspired from a policy introduced by Belady [Belady, 1966]. This suggestion comes as an alternative to standard cache replacement policies (LRU, pseudo-LRU and FIFO) which suffer from lack of predictability, a necessary property for real-time systems. Also, the proposed technique has the advantage of reducing performance anomalies encountered in the other policies. Despite its average performance reduction, the randomized replacement policy has the advantages of eliminating dependencies from access history and in allowing higher execution time values to be observed during measurements.

In the context of Proartis and Proxima projects, the use of randomization in hard real-time systems is highly promoted. Randomized caches placement/replacement policies are adapted for real-time systems [Cazorla et al., 2012, Cucu-Grosjean et al., 2012, Davis et al., 2013a, Kosmidis et al., 2013a, Kosmidis et al., 2013c, Kosmidis et al., 2013d, Kosmidis et al., 2014a] as an environment for static probabilistic timing analysis (SPTA) and measurement-based probabilistic timing analysis (MBPTA). In order to further randomize the system, a bus arbitration policy that relies on randomized-permutations is proposed in [Jalle et al., 2014]. For those systems, on which hardware intervention is difficult or impossible (e.g. COTS hardware), randomization at software level is later suggested [Kosmidis et al., 2016].

Caches are fast and small memories that bridge the latency between the CPU and main memory by storing parts of the main memory. Data is stored in cache

lines of the size of memory blocks from the main memory. This process has the effect of increasing performance by profiting from spatial and temporal locality. The data in cache is placed or modified according to different policies, characteristic to each hardware producer. From the timing analysis point of view, cache memories increase the difficulty of reliably estimating WCET bounds. Being able to correctly calculate the timing behavior of the system through static analysis is challenging because of multiple system features like the cache size, the placement and replacement policies used, the program's complexity and the initial cache state.

Randomizing the cache has two benefits:

- Gives a totally randomized behavior to the cache which allows the use of known statistical methods for the computation of a discrete probability distribution over the instruction's possible execution times and to make instructions independent which will allow their probability distribution composition through the the convolution operation [Davis et al., 2013a].

- Increases the observable space of a system's states and make observation of extreme execution times possible and, as a consequence, allows the use of MBPTA based on EVT in order to derive a WCET bound [Cucu-Grosjean et al., 2012].

Despite these benefits, the necessity of cache randomization is questioned in [Reineke, 2014], where a comparison between LRU replacement policy and random replacement policy is done. The presented result shows weak performance of randomization on certain corner cases or dummy examples. Nevertheless, randomization (and MBPTA in particular) is offered as a solution for complex systems where exact prediction of the occurring states can not be made.

In [Lima et al., 2016], the authors point out the fact that the second benefit of randomization occurs only under certain conditions. The paper shows that the use of MBPTA is based on a strong link between cache size and the program's number of executed instructions. In other words, a small program will charge all its instructions in a cache independently of the cache replacement policy and the execution times observed will benefit from the cache use and have a small variability. On the other hand, a program having a large number of instructions will always be at a disadvantage by the random replacement policy (which does not take advantage of the spatial and temporal locality) and will make the use of cache more costly than

direct access to main memory. This case will produce execution time traces with low variability and high execution times. Overall, the conclusion of [Lima et al., 2016] is that randomization is useful but "not strictly necessary for ensuring analyzability".

In [Maxim et al., 2016], the property of reproducibility in randomized architectures is questioned. A series of properties necessary for the use of EVT are presented and the idea that randomization does not respect them is indicated. A further explanation of this presumption is presented in the contribution section of this paper (see section 3.4).

We do not extend on the benefits and disadvantages of bus randomization and software randomization due to the lack of information concerning these methods, but we bring a conclusion on the use of randomization in general.

In this thesis we present MBPTA results from analysis of avionics applications in particular, but also of existing benchmarks . The analysis is done mostly on deterministic behavior of the system, but comparison with randomized cache is also presented. We demonstrate with this occasion that MBPTA analysis can be done with success on systems behaving exactly as in the conditions for which they were conceived. We also question the confidence and possibility of certification of these results through the problem of representativity.

In practice, randomization is seen as a tool for testing, but its use in active systems is hard to imagine due to the performance penalty that it brings. Moreover, the fact that new and complex architectures introduce automatically variability in the measured execution times making the use of MBPTA possible mitigates against randomized technologies.

2.5 Timing analysis in avionics industry

The aviation industry is one of the first domains that use real-time systems. The time requirements in avionics come as a necessity for passenger and aircraft security. Inside an aircraft, one can find many embedded systems with different purpose, criticality and characteristics. A part of these systems are subject to time constrains and their understanding is very important for the aircraft's functioning. Examples of such systems can be found in the applications used for flight warning, flight control, weight and balance computation, collision avoidance, etc.

The first Airbus aircraft (A300) relayed on systems constructed according to the *federated architecture* paradigm, which consisted on fully dedicated computational

units for every component of the system. These computers, called *Line Replaceable Units*, are conceived by a system supplier to allow local optimization. Timing analysis of such components was straight forward due to the determinism of the architecture and the small size of the code executing on it.

With the expansion of the airplanes market, the number of functionalities demanded by airlines increased, making the federated architectures difficult to sustain. From the electronics perspective, the computation power of computers increased, allowing them to perform a high number of instruction in a short interval of time. As a consequence of these two evolutions, the apparition of an integrated approach for avionics became inevitable. In 1995, the cockpit of the Boeing 777 aircraft contained components that respected the Integrated Modular Avionics (IMA) concept. We further detail this concept and its impact on the timing analysis.

2.5.1 Integrated Modular Avionics (IMA)

The IMA concept consists in the implementation of several functions sharing the same computer resources like processing resource (CPU time), memory or input/output capacity. Figure 2.3 depicts the transition process from federated architecture to IMA. The sharing of resources implies careful implementation since avionics industry requires separation of applications. In order to avoid interference between application, IMA relies on the concepts of partitioning. A partition is the scheduling unit of a static schedule computed offline. The partitioning is threefold:

- **Spatial partitioning**: allocates dedicated memory areas to each partition and blocks other partitions to perform accesses to memory outside of their areas. In other words, restricted access to memory areas is ensured. The spatial partitioning occurs between different avionics applications as well as between avionics applications and core software. The memory protection is provided by mechanism implemented in the processor (MMU thanks to page tables and BATs) and in the CPU board chipset (Dedicated Memory Controller Protection Registers).

- **Temporal partitioning**: guarantees temporal isolation between different partitions. This is done by allocating CPU time to each partition according to its need. The allocation is made through a mechanism called SLICER that schedules partitions based on static configuration files and guarantees them

uninterrupted access to common resources during assigned time periods of partitions. The hyper-period of all partitions is known as Major Frame (MAF), and it is divided into a number of Minor Frames (MIF). All MIFs have the same duration and period, and contain a number of partitions. When a given partition is active, only its processes can be scheduled. More than one partition can be allocated to the one and the same application within a given MAF. The exclusive access of partitions to all required resources is implemented with success in single-core architectures, but harder to imagine in multi-core paradigms when parallel execution of partitions would be problematic. In order to avoid time anomalies caused by contentions and resource sharing, new partitioning techniques that enforce separation are proposed [Jean et al., 2012, Boniol et al., 2012], but the topic still has to evolve. A key role in achieving temporal partitioning is represented by the WCET estimation of the processes executing inside partitions.

- **Communication partitioning**: isolates the communications between partitions and resources via I/Os modules. The sharing of IO devices involves two aspects: the sharing of the processing means used to perform IO data processing (i.e. activities done before actually driving the physical line), and the sharing of the physical line itself.

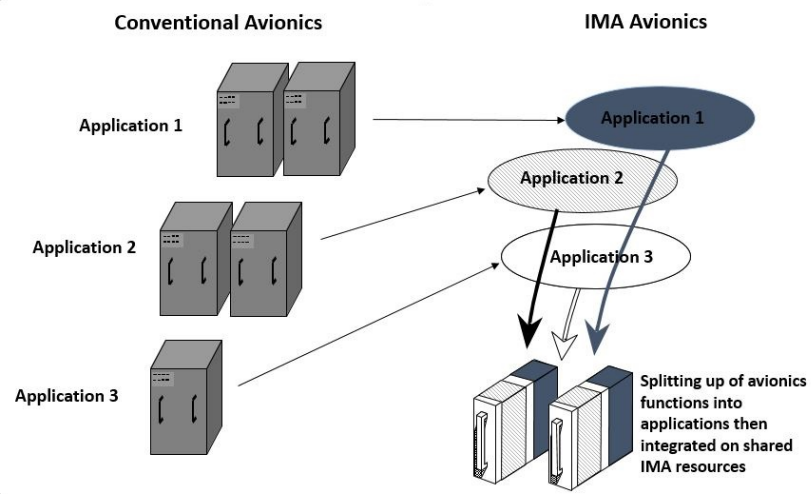

Figure 2.3: Example of transition from federated architecture to Integrated Modular Avionics.

An advantage brought by the IMA concept is represented by the independence of the software applications from other applications and from the operating systems and hardware. This way, software developers can focus on their products without needing to know the whole system's architecture. Another advantage is given by the modularity aspect that allows easy replacement, adding or elimination of software applications at the same time allowing the existing one to achieve a high level of maturity. From the financial point of view, the use of IMA contributes to weight reduction of the aircraft, which implies less energy consumption.

2.5.2 Time analysis of avionics applications

In practice, calculating the exact amount of time is allocated for each partition is done through a complex process by the software architect. Time must be cautiously attributed to each partition according to the criticality level of the processes executing inside it and the WCET bounds of these processes. The end of a partition attracts the suspension of all executing processes and might cause a system restart with critical consequences for the aircraft. As a consequence, applications having a high criticality need to be subject to a strict timing analysis in order to be certified. The WCET in avionics industry is done through static analysis computed by certification authorities validated tools.

Static timing analysis (STA) for deterministic microarchitectures is divided in two main parts:

1. *Low-level analysis* , which determines execution-time bounds for basic blocks based on an accurate model of the underlying microarchitecture.

2. *Path-level analysis*, which determines an upper bound on the execution time of the program as a whole, based on constraints on the control flow, e.g. loop bounds, and the execution-time bounds for basic blocks determined by low-level analysis.

Some example of tools capable of doing STA are:

- The **aiT** is a commercially available tool from AbsInt [Ferdinand and Heckmann, 2004], directly analyzes binary executables while taking cache and pipeline behavior into account. Historically, this has been the tool of choice for Airbus.

- The **Bound-T** tool of Tidorum [Holsti et al., 2000] is a finish originated analyzer that determines an upper bound on the execution time of a subroutine, including called functions.
- The **Heptane** tool of IRISA [Colin, 2001], is a open-source static WCET analysis tool released under GPL license. This tool computes upper bounds on execution times for programs written in C by analyzing the source code or the binaries.
- The **SymTA/P** tool of TU Braunschweig [Staschulat and Ernst, 2004] is based on the idea of combining platform independent path analysis on source code level and platform dependent measurement methodology on object code level, using an actual target system. The main benefit is that this hybrid analysis can easily be re-targeted to a new hardware platform.
- The **RapiTime** tool of Rapita Systems Ltd. [RapiTime, 2006] derives WCET bounds based on measurements. This tool targets the automotive electronics, avionics and telecommunications industries.
- The **Otawa** tool from Irit [Cassé and Sainrat, 2006] is an academic tool accessible as a C++ library.

A detailed description of static timing analysis techniques and tools is out of the scope of this thesis. For a complete image of this topic we guide the reader to already existing reviews [Wilhelm et al., 2008].

Most of the existing tools are used on deterministic platforms for which a model is created. This is the case of single-core architectures. The translation from single-core to multi-core processors introduce unexpected interferences that can violate partitioning [Fuchsen, 2010], becoming an impediment to further assimilation of such architectures. Nevertheless, the high performance potential and the temptation of parallelization make this topic an interesting one with considerable efforts in both industry and research.

One method proposing a WCET analysis in multi-core processors consists in analyzing applications in isolation (as it is done in single-core) and joining them while observing the effects they have on each other. If such an approach can be considered for those applications containing a low number of processes, for complex systems the number of permutations that is required might reach unaffordable costs in time and effort.

In the Proxima project, the use of probabilistic methods are proposed as a mean of estimating WCET bounds of applications from end-to-end measurements. The details of approach is described in the contribution chapter.

2.5.3 Mixed-criticality systems

One advantage of the IMA concept is the possibility to incorporate on the same computer applications from different criticality levels. Robust partitioning allows cohabitation of software of multiple criticality levels. The definition of criticality in the avionics industry is presented in section 1.1.3, but in research this notion took a different signification through its use in mixed-criticality systems [Vestal, 2007].

Vestal's model for mixed-criticality systems consists in a task set to which a finite number of criticality levels is associated. Each task has a certain criticality being able to release jobs in any level equal or lower than its own. According to the level in which a task's jobs are executing, its WCET will differ. A time violation occurs when a job needs more time to execute than its allocated time at the level in which the system is. In the case of a time violation, the system's criticality will increase with one level and all the task will adapt (task with a lower level of criticality will be dropped while those being able to execute in the new level will generate jobs with the corresponding WCET characterization). For a detailed description of this model, we advise the reader to consult the original paper proposing it [Vestal, 2007].

We present in this section the main divergences between the notions of criticality as it is used in industry and the one from Vestal's model:

1. In Vestal's model criticality applies to a task while in industry the criticality is given to a function.

2. In the classical model, multiple WCET values are attributed to higher criticality tasks while in industry each application needs to have only one WCET certified. The certification of multiple WCET would mean an extra cost for the aircraft producer. Also, if multiple WCETs would exist for an application in industry, and its smallest WCET value will be certified what would be the point of certifying the other larger values?

3. In research, a better CPU usage is presumed to be obtained through the use of smaller WCET corresponding to tasks executing under low criticality. In reality, the implementation of a system that allows different partition allocations

according the dynamic changes in the system is difficult.

4. In Vestal's model, failure in timing assumption of high criticality tasks result in dropping lower criticality tasks. In avionics, and in the IMA concept in particular, spatial isolation does not allow failures in a function to affect any other function.

5. The idea of mode change in the case of time violations is inconceivable in avionics, since functions are given a certain criticality according to its SIL and any change of its criticality is subject to a new certification procedure.

In this thesis we do not concentrate on the study of mixed-criticality system due to the industrial environment of the thesis. Therefore the further mentioning of criticality will refer to the motion encountered in industry.

Chapter 3

Conditions for the use of EVT in the real-time domain

In this chapter we establish the environment in which our work is done. The use of EVT in any domain comes with a series of restrictions for the data being analyzed. In our case the data being analyzed consists in execution time measurements. We resort to this choice due to EVT's ability to be used on any numerical values. Besides the statistical conditions of independence and identical distribution of data, we define the necessary conditions for a real-time systems to produce analyzable data. The key reasoning of these conditions relies on the system consistency and on the measurement protocol.

The final goal of EVT use is to obtain the probability distribution of an extreme event occurring in the future based on already observed events. In our case, the event occurring in the future is the observation of the WCET and the already observed events are represented by a sample of measured execution times. In other words, the probability distribution of the WCET is a statistical model of the analyzed system based on past executions. The process of modeling the system's behavior is a complex one, sensitive to the used data and relying on human intervention at times. The main steps of using EVT are data collection, data selection and modeling. The success of the modeling process depends on the selection stage which, on its own, is dependent on the collection one. In this chapter, we concentrate on the collection process, while in the next chapter we present the other two steps.

3.1 System consistency

In practice, it is noticeable that the higher the measured execution time is, the smaller its probability of occurrence is. In reality, the WCET is not easy to measure, and the analysis tools can either overestimate the WCET (static analysis), or underestimate it (taking into consideration only measurements), or predict it with a certain probability of occurrence (measurement-based probabilistic timing analyses - using EVT). When applying statistic methods on execution time samples, the provenance of these systems is highly important. We consider an analyzable system as a compact structure formed of three elements: *input*, *software* and *platform*. Each estimated pWCET distribution depends on the three elements of the system and the modification of any of them results in a system change for which the already estimated pWCET is not characteristic. A depiction of a system's structure seen from the probabilistic analysis point of view is presented in figure 3.1. We numerated the components and we detail them separately.

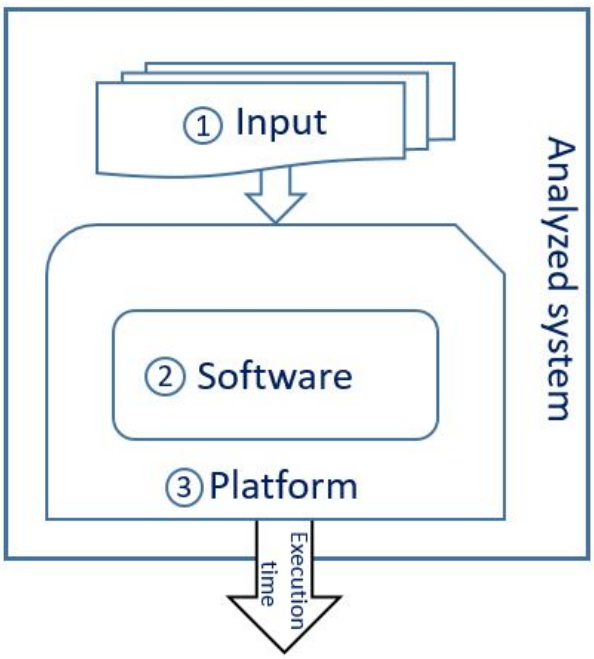

Figure 3.1: The structure of an analyzable system.

3.1.1 Input

When analyzing a system, there is a direct dependence between the input used and the execution times measured. A basic approach, used in industry, is to identify the input vector that forces the system to have the worst case behavior. The average and maximum values of the execution times obtained using this input vector are higher than the equivalent measurements when any other input vector is used. Meanwhile, this approach does not guarantee the observation of the real WCET of the system. Finding this input vector is a difficult work that depends on the knowledge of the program and even of the hardware on which it runs. Methods that rely on model checking and genetic algorithms [Wenzel et al., 2005] are proposed to be used with static analysis. Heuristics that explore the input space [Ermedahl et al., 2009] can be found for the use of measurement analysis.

In the context of statistical analysis, having the input vector is an advantage. In this thesis, a great part of the experimental results are obtained for systems in which this input vector is known and the variability of the system comes mostly from the hardware platform. The advantage of using statistical methods consists in the ability to extrapolate the system's behavior when not all the elements of the system are known. Treating the system as a black box is an approach often encountered in research, but in such case, the analysis results can be validated only if the system behaves similarly at run-time and during the analysis. For the cases when the system is seen as a black box and no information is given about the input, EVT is only able to provide a result as good as the used inputs are. In other words, relying on random generated inputs is not a good choice, especially for critical real-time embedded systems. The following reasons and examples support our claim:

- *Order of inputs*: the order of inputs influences the order of execution times. The GEV distribution is sensitive to the order of the data that is being fed with. Therefore, selecting randomly the inputs every time we are running a series of experiments produces different traces (value-wise or/and order-wise) corresponding to different pWCET estimations.

- *Input space* : finding the interval in which the accepted input values by the analyzed program are situated is importance. Setting a limit on the minimum and maximum used inputs also limits the GEV analysis to that interval. Any other input value outside that interval cannot be guaranteed to be smaller

than the pWCET obtained for the fixed interval.

- *Software influence*: depending on the number of branches a program has, and the weight of each branch, the execution times obtained from a random selected inputs can be transformed in a different distribution . As an example, in the case of a program testing the primality of a number, by randomly choosing the inputs, around half of the executions will take the same time, used to verify if the input number is even. Therefore, a randomly distributed input is transformed in light tail distribution with little variability. On such cases, variability cannot be reached by randomizing all possible inputs, but rather by intentionally feeding the program an input meant to exercise as many branches of the program as possible.

In the context of multi-core hardware, where uncertainty arises from the architecture, the risk of obtaining a small number of different execution times is reduced and the interest of applying statistical methods increases. It is true that using the input vector meant to worst case behavior of the system is pessimistic when compared to behavior of the system in normal conditions (any other input vector), but for the case of critical real-time systems being pessimistic is desired over the chance of underestimation of the WCET.

3.1.2 Software

From a probabilistic point of view, a program can be represented as a generator of execution time profile sets (ETPs). Combining all these ETPs, we obtain an absolute domain of execution times. In practice, such a domain is hard or impossible to determine through measurement for complex programs running on non-deterministic hardware. Discovering which ETP sets have a higher influence on the pWCET estimation would allow us to concentrate on their analysis in order to produce a reliable pWCET without knowing the entire domain of execution time.

The key in highlighting the influential ETP sets stays in the structure of the program and the representation of the input-output relations. Every probabilistic analysis should start with the definition of the domain of analysis and the decision of the interval in which our program's inputs appear. The choice of this interval defines the number of ETPs we need to consider for analysis. The program's semantics is afterwards projected on probabilities, each path and its weight are processed in

order to compute the ETP sets and its influence on the total domain of execution times.

By using a worst case input vector, the chances of finding an influential ETP are maximized. In the following sections we define the necessary conditions for maximizing the domain coverage through measurement protocol. Hence, we lift the WCET analysis from the level of program/platform and we focus on the compositional features of the ETP processing for obtaining pWCET.

When the system is treated as a black-box and there is no information concerning the software, the EVT results depend on the inputs used and their capacity of accessing all the software's paths. The statistical methods proposed in this thesis are capable of distinguishing samples coming from evidently different paths. Also, by using maxima picking strategies we ensure the isolation of samples coming from the dominant paths.

3.1.3 Platform

COTS hardware are becoming an option for industries everywhere due to their reduced cost and increase in performance. This comes with a disadvantage concerning static timing analysis and WCET estimation. At the same time, the use of complex platforms (as it is the case of COTS ones) gives an advantage to statistical methods through the variability introduced in the measured execution times. Therefore, for such architectures, a software executing multiple times the same code and using the same input will produce different execution times. This is happening because of the multiple performance enhancer technologies (e.g. caches, branch predictor, pipelines) that give the system an unpredictable behavior. Rendering such an platform deterministic by deactivating all these features is not an option.

The methods proposed in this thesis rely on the variability introduced by the platform (hardware). EVT cannot be used on samples containing a limited number of unique values. Therefore, for each results we specify the hardware used, keeping in mind that the pWCET estimation obtained for one system will not be valid if the platform changes.

3.2 Identical distributed data

All probabilistic theory rely on the concept of random variable which represents a quantity whose outcome is uncertain. In practice, when using statistical methods, we deal with observations of such random variables. We consider x_i as the measured observation of the random quantity X_i. Until measured, X_i can take any value in its *sample space*. Obviously, some values are more likely to be seen than others which translates in the fact that each value has attached a probability of appearance. Thus X_i is assumed to have a probability distribution.

In extreme value theory the definition of GEV and GP is done based on sequences of independent and identically distributed random variables. In practice, this means that an event has to be observed multiple times in the form of samples that can further help in inferring characteristics on the process that generated the data. For EVT the characteristics we are looking for are represented by the behavior of extreme observations. In the case of the central limit theorem, the evolution of the average behavior is studied.

Further, we split the properties of independence and identical distributed, detailing them separately through definition, interpretation in the case of execution time observations and verification through statistical tests.

Definition 9 *We say that two random variables X and Y are* ***identically distributed*** *iff $P(X < x) = P(Y < x), \forall x \in \mathbb{R}$.*

In other words, we consider two random variables to be identically distributed if they have the same probability distribution. On the other hand, we cannot say that two observations are identically distributed since they are merely manifestations of events described by random variables. If we know for sure that the same event produced a set of observations, we can say that any two subsets of this set are coming from identically distributed random variables. As a consequence, we consider that a data set (a sample) is identically distributed if any two randomly picked subsets (sub-samples) can fit to the same distribution.

In the real-time context, when the data under analysis is represented by execution time measurements, the process for which we need to infer characteristics is the system's execution. When the system is treated as a black-box, it is tempting to consider a trace of its execution times identically distributed for the simple fact that they were all obtained from a single system. We admit that this can be done

if the system under analysis is always executed in the same scenario (unique input, no modification of software or hardware between runs) and the program used has a single path. These characteristics are not known through the nature of the black-box approach and the execution time set used for pWCET estimation needs to be confirmed as identically distributed.

Even though, for the experimented part of this thesis, we have sufficient knowledge on the system under analysis, we develop a method that tests the property of identical distribution of data. This method is based on the Kolmorgorov-Smirnov (KS) test, which is nonparametric and is capable of comparing two samples in order to conclude if they come from the same population (with a specific distribution). Nonparametric tests are also called distribution-free tests because they don't assume that your data follow a specific distribution. We mention that this test can also be used to compare a sample with a reference probability distribution as a goodness of fit test.

In our case, testing data to determine if it is identically distributed, two randomly picked samples of the data are compared. The null hypothesis of the test is that the samples follow the same distribution. The result of the test is the p-value, according to whom we decide if the null hypothesis is rejected or not. Classically, if the p-value is grater than 0.05 the null hypothesis cannot be rejected meaning that the two samples can be considered as following the same distribution. This test can produce false negatives, while for those cases where p-value is greater than 0.05, there is no doubt on the result. Since there is no such thing as a truly random generator, we rely on a pseudo-random method (from the R software) for picking the tested samples. In order to avoid borderline cases, where for two iteration of the test the results are different we perform the procedure 100 times and we will consider the data as being identically distributed for a success rate of 90% or higher. We use here the theoretical argument that for a population that is not obtained by the same procedure 0% of the test will succeed (100% failures). This can be the case for a program with multiple pats and an input capable to take at least two of them. The borderline case might come when multiple paths with similar behavior are exercised.

The code of this algorithm is written in R software using the procedure ks.test() from the stats package, and can be found in the Appendix of the thesis.

3.3 Independence

The measurement-based approaches are widely used in the real-time embedded systems industry where the concept of *high water mark* (HWM) is considered as the largest observed execution time. Lifting its utilization to systems with different components requires compositionality while two different components may have different values for the HWM of a program executed on those components. Moreover, timing anomalies may prevent the HWM of a program to be obtained by the combination of the HWMs of the program on the components.

Before indicating how such measurement protocol may be proposed, we provide firstly the (necessary) definitions for independent programs, statistical independence, and probabilistic independence.

Definition 10 *We consider two programs $Prog_1$ and $Prog_2$ to be* **independent** *iff any execution of $Prog_1$ may be done before or after any execution of $Prog_2$ without any impact on their execution times.*

Two programs that are in any other situation than those covered by the definition of independent programs given previously, are *dependent*.

Consider for instance the program $Prog_{ex1}$ described in Table 3.1 and $Prog_{ex2}$ described in Table 3.2. These two programs are kept simple in order to ease the understanding.

The two programs $Prog_{ex1}$ and $Prog_{ex2}$ are dependent as $Prog_{ex1}$ produces a (positive integer) value for the global variable var_global_2 that is then used as an input by $Prog_{ex2}$. For instance each time $var\_global_1 = 1$, then $Prog_{ex1}$ has an execution time equal to 3 time units and $Prog_{ex2}$ has an execution time equal to 6 time units. For $var\_global_1 = 2$, then $Prog_{ex1}$ has an execution time equal to 4 time units and $Prog_{ex2}$ has an execution time equal to 10 time units.

For these two programs we may obtain both statistical dependent execution times or statistical independent execution times.

Definition 11 *Two probability distributions $\mathcal{C}_1$ and $\mathcal{C}_2$ are* **independent** *iff*

$$P(\{\mathcal{C}_1 = c_1\} \cap \{\mathcal{C}_2 = c_2\}) = P(\mathcal{C}_1 = c_1) \cdot P(\mathcal{C}_2 = c_2)$$

For instance the probability distributions of the execution times $pET(Prog_{ex1})$ and the probability distributions of the execution times $pET(Prog_{ex2})$ are *probabilistically dependent* as there is a relation between the probability of appearance

Table 3.1: Body of program $Prog_{ex1}$.

$Prog_{ex1}$ (var_global_1);	
$value = var\_global_1$;	// *execution time = 1 time unit*
for i = 1 **to** var_global_1	// *the loop cost is in the instr*
$value = value + 1$;	// *execution time = 1 time unit*
endfor	
$var\_global_2 = 2^* value$;	// *execution time = 1 time unit*

Table 3.2: Body of program $Prog_{ex2}$.

$Prog_{ex2}$ (var_global_1, var_global_2);	
$value = var\_global_2$;	// *execution time = 1 time unit*
for i=1 **to** var_global_2	// *loop cost is in the instr*
$value = value + 1$;	// *execution time = 1 time unit*
endfor	
$avg\_global = \frac{value+var\_global_1}{2}$;	// *execution time = 1 time unit*

of an execution time for $Prog_{ex1}$ and the probability of appearance of an execution time for $Prog_{ex2}$. If one wants to estimate the probability distributions of the execution times of these two programs executed sequentially then, given their probabilistic dependence, a complex probabilistic operation will be necessary to take into account the conditional probabilities.

Sometimes these dependences are not strong and simple probabilistic operations are possible [Santos et al., 2011]. Nevertheless, if one is able to provide a pWCET estimation $pWCET(Prog_{ex1})$ for $Prog_{ex1}$ and a pWCET estimation $pWCET(Prog_{ex2})$ for $Prog_{ex2}$, then these two probability distributions are independent by the definition of a worst case bound [Cucu-Grosjean, 2013].

Definition 12 *A set A is* **statistically independent** *iff its elements are generated in a random manner (i.e., the value generated at one instant only depends on the generator and not on the values generated before).*

For instance $A_{ex} = \{8, 18, 21, 24, 28, 30\}$ is statistically independent. We have generated A_{ex} using an on-line random generator[1].

[1] http://www.infowebmaster.fr/outils/generateur-nombre-aleatoire.php, (on-line form), but the

Statistically independent execution times for dependent programs. We consider the independent set A_{ex} as input to obtain independent execution times for our two programs, $Prog_{ex1}$ and $Prog_{ex2}$. If we consider var_global_1 to take the values from A_{ex} then the set of execution times of $Prog_{ex1}$ is $\mathcal{C}_{Prog_{ex1}} = \{10, 20, 23, 26, 30, 32\}$ which is statistically independent.

In order to obtain, for $Prog_{ex2}$, a set of statistically independent execution times we consider the values of var_global_2 to take the values from (another) statistically independent set $A_{bis} = \{1, 45, 59, 75, 88, 90\}$. We obtain a set of statistically independent execution times for $Prog_{ex2}$ equal to $\mathcal{C}_{Prog_{ex2}} = \{3, 47, 61, 77, 90, 92\}$. These two sets of execution times are statistically independent, while the programs are dependent.

Statistically dependent execution times for dependent programs. Moreover if we use a set of dependent elements like $B = \{1, 2, 3, \cdots, 8\}$, then we may obtain statistical dependent sets for the execution times of $Prog_{ex1}$ and $Prog_{ex2}$. In this case the execution times of $Prog_{ex1}$ are $\{3, 4, 5, 6, 7, 8, 9, 10\}$ and the execution times of $Prog_{ex2}$ are $\{6, 10, 14, 18, 22, 26, 30, 34\}$. These two sets are statistically dependent, while the programs are dependent.

Therefore, we observe that **the measurement protocol has a direct impact on the statistical independence of the execution times**. For two programs with pWCET estimates we may proceed at a convolution (or other composition) of their bounds [Cucu-Grosjean, 2013]. Nevertheless, the existence of this bound is not only requiring statistical independence but also the convergence of the WCET measurement-based estimation (see following section for more details).

The statistical independence may require appropriate tests when the sets of execution times are large. We have considered here two types of tests: visual or using analytical formulas.

For a visual confirmation of a dependent set of execution times, we use the *lag test* which indicates a (particular) pattern for dependent data and a graph of unrelated points for random data. Figure 3.2 depicts a set of independent data on the left and a set of auto-correlated data on the right. A lag is a fixed time distance (e.g. in a $X_1, X_2, .., X_n$ data set X_1 and X_3 have lag 2). Lag plots can be generated for any lag, but the most commonly used lag is 1. A plot of lag 1 is a plot of the values of X_i versus X_{i-1}. For instance in Figure 3.2 the abscissa axis corresponds

reader may use any other such generator.

to X_{i-1}, while the ordinate to X_i.

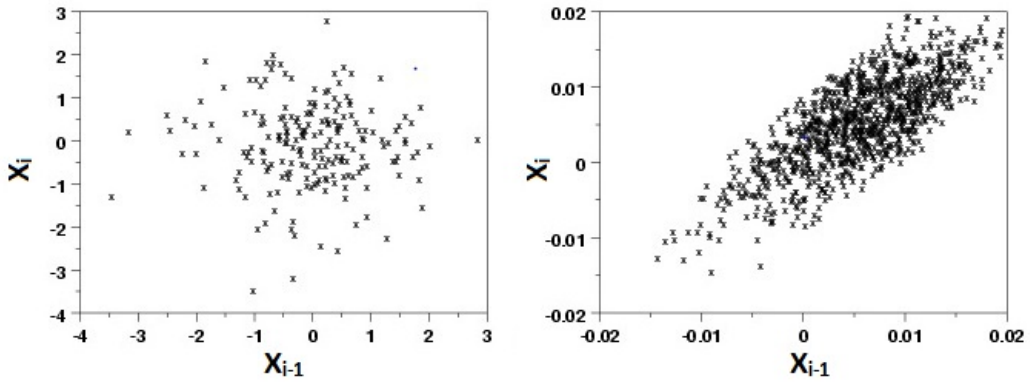

Figure 3.2: Representation of dependences using the lag test. On the left figure we have independent data and on the right figure dependent data.

In order to confirm the visual perception of statistical dependence (or to infirm it), one may use the Wald-Wolfowitz test, a.k.a. the *run test* to quantify the data dependence. Its mathematical description is provided in the Appendix of the thesis.

3.4 Reproducibility and representativity of measurement-based approaches

The measurement-based approaches, in general, and the probabilistic measurement-based approaches, in particular, propose WCET estimates using the execution times of the program on the given platform (see Figure 3.3). More precisely let $C_1^i, C_2^i, \cdots, C_n^i$ be n consecutive executions of a program on a processor starting from a given scenario of execution S_i. A scenario of execution for a program on a processor is defined by a set of states corresponding to different execution time variability factors. A scenario of execution could correspond, for instance, to the pair (path of the program, state of the cache) or to any other information related to the execution of the program.

For a scenario S_i we may define a probabilistic execution time $\mathcal{C}_i$ as an empirical probability distribution of the execution time of that program for the given processor.

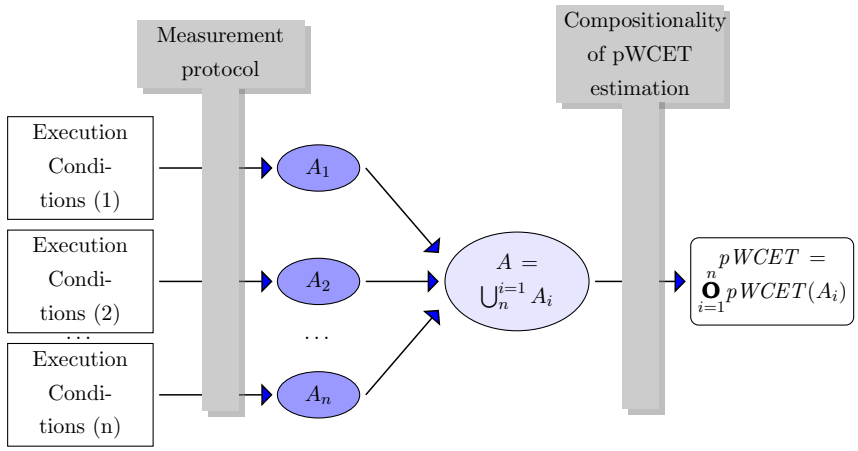

Figure 3.3: The protocol of a (p)WCET estimation from different scenarios of execution conditions.

In the remainder of this thesis we consider that the processor is fixed in sense that we estimate the pWCET of a program on a processor from the execution time measurements of the program on that given processor.

In this section we identify and characterize the *convergence*–a key feature for the compositionality of a measurement-based WCET estimation process. Any measurement-based WCET estimation process has two main parts: (i) the measurement protocol and (ii) the WCET estimation method.

The convergence of a measurement-based WCET estimation process for a program on a processor is defined by the existence of a finite set of execution times provided by a measurement protocol such that the associated measurement-based WCET estimation method provides a unique WCET estimation of that program on the given processor. A more formal definition is provided in Definition 13.

Definition 13 *Given A an absolute domain of execution times, a measurement-based WCET estimation process $pWCET$ is* ***convergent*** *if for any ascending chain of subsets $A_i \subset A$ (obtained using its measurement protocol) converging to A (i.e., $A_i \subseteq A_{i+1}, \forall i$ and $\lim_i A_i = A$) the associated chain of WCET estimations (obtained using its WCET estimation method) is almost constant, equal (or sufficiently close) to the WCET estimation of A (i.e., $\exists t \geq 0\ \forall j \geq t$ such that $pWCET(A_j) \approx pWCET(A)$).*

The convergence of a measurement-based WCET estimation requires several properties to be satisfied. We identify in this document a (non-exhaustive) list of

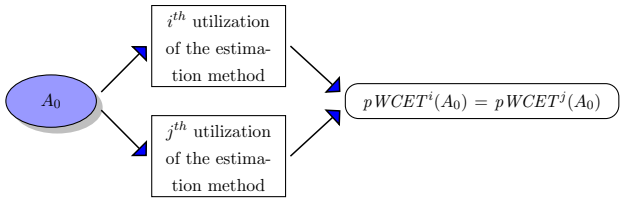

Figure 3.4: The WCET estimation is the same for different utilizations i, j of a reproducible WCET estimation method from the exactly same ordered set of execution times.

these mandatory properties (their order of presentation is not relevant):

- The reproducibility of the WCET estimation method (defined Section 3.4.1);
- The reproducibility of the measurement protocol (defined Section 3.4.2);
- The representativity of the measurement protocol (defined Section 3.4.3).

We present in Section 3.4.4 the relations between these three properties and the convergence.

3.4.1 The reproducibility of the WCET estimation method

We may note that the measurement-based WCET estimation method is used several times over subsets of ETPs, e.g., A_i in Definition 13. If two different utilizations of the estimation method on (exactly) the same subset A_0 of execution times provide different WCET estimates than the measurement-based WCET estimation diverges and it cannot provide a reliable result.

Definition 14 *A measurement-based WCET estimation method pWCET is* ***reproducible*** *iff for any two utilizations i and j the estimates $pWCET^i(A_0)$ and $pWCET^j(A_0)$ $(i \neq j)$ are the same.*

In Figure 3.4 we depict the notion of reproducibility of a pWCET estimation. For example, the EVT-based pWCET estimation method (introduced in [Edgar and Burns, 2001]) is reproducible as long as the order of the elements in A_0 of the execution times is not modified.

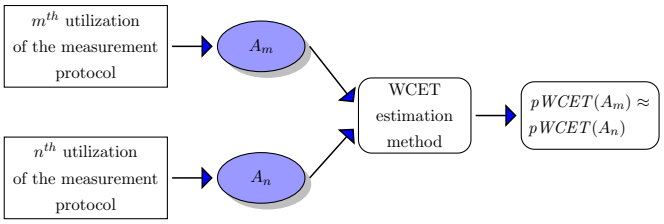

Figure 3.5: Different utilizations of a reproducible measurement protocol provides WCET estimates that are equal (or sufficiently close).

3.4.2 The reproducibility of the measurement protocol

The measurement protocol is an essential step in the measurement-based WCET estimation. We now focus on characterizing this step w.r.t. the convergence property.

Definition 15 *A measurement protocol P is **reproducible** iff for two different utilizations P_m and P_n with the same execution conditions (status of the processor, program and external factors), the obtained set of execution times A_m and respectively A_n correspond to equal (or sufficiently close) WCET estimates for the utilization of the same WCET estimation method $pWCET$, i.e., $pWCET(A_m) \approx pWCET(A_n)$.*

In Figure 3.5 we depict the reproducibility of a measurement protocol. Note that a completely randomized measurement protocol may not be reproducible with respect to the EVT-based pWCET estimation method. For instance, given a randomized cache replacement policy, if both the seed of the random generator and the places in caches are randomly picked, then the architecture execution times may not be equivalent as different associated pWCET estimates may be obtained with EVT-based pWCET estimation methods.

We also note that the randomization of only the input values for a program is not a reproducible measurement protocol either when considering an EVT-based pWCET estimation method [Lima et al., 2016, Lu et al., 2011]. This absence of the reproducibility is due to the sensitivity of EVT-based pWCET estimation method to the order of the execution times.

3.4.3 Representativity of a measurement protocol

We now present a second feature of the measurement protocol which contributes to ensuring the convergence property.

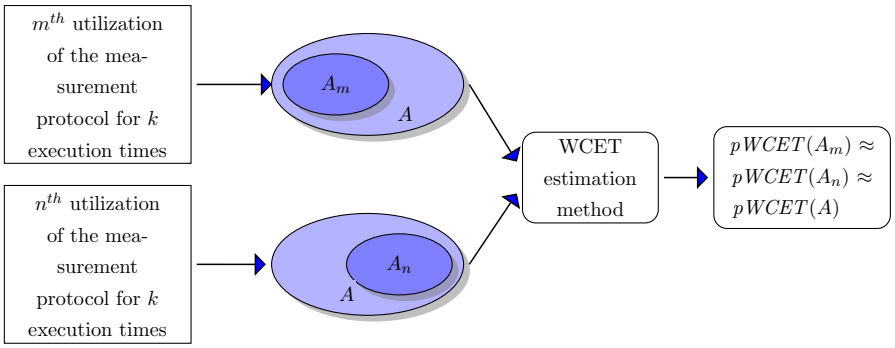

Figure 3.6: A representative measurement protocol provides equivalent subsets of execution times.

Definition 16 *A measurement protocol is* ***representative*** *iff there exists a number k of execution times for a measurement protocol such that*

$$pWCET(A_{k'}) \approx pWCET(A), \forall A_{k'}: A_k \subseteq A_{k'} \subseteq A \tag{3.1}$$

for any $A_k \subseteq A$ with $|A_k| = k$.

In Figure 3.6 we depict the representativity of a measurement protocol, where we denote by A the ETP of the WCET estimation, while A_m and A_n denote some subsets of execution times of cardinal k.

Using the notations of Figure 3.6, we may indicate that measurements obtained using randomized replacement policies (with the method provided in [Cucu-Grosjean et al., 2012]) seem to present a representativity of the HW-randomization measurement protocol for $m = 6$ utilizations of the protocol for $k = 1000$. Note that the original set has 500 execution times in the presence of Mälardalen benchmarks [Gustafsson et al., 2010]. Nevertheless there is currently no proof that such representativity may be extended to other classes of programs.

Note that the random picking of program inputs is not by default a representative measurement protocol. However, such protocol should define a representativity property with respect to the pWCET estimation method.

To our best knowledge, there exists no proof for the representativity of a measurement protocol for any given set A.

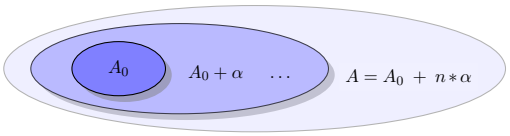

Figure 3.7: The absence of the reproducibility of a measurement protocol may prevent A_0 to converge to A.

3.4.4 Relations between reproducibility, representativity and convergence

We enumerate the relations between the concepts defined previously:

- The reproducibility of the WCET estimation method is a mandatory property for the reproducibility of the measurement protocol. Indeed if the WCET estimation method is not reproducible than for two same sets of execution times provided by a measurement protocol, the WCET estimates could be non-equal.

- The reproducibility of a WCET estimation process requires both the reproducibility of the WCET estimation method and the reproducibility of measurement protocol. Indeed if the WCET measurement protocol is not reproducible, the WCET estimate will be modified for each new measurement even in presence of a reproducible WCET estimation method.

- The reproducibility of the WCET estimation process and the representativity of the measurement protocol are *mandatory properties* for the convergence of a WCET estimation process. In Figure 3.7 we illustrate a convergence principle by slowly increasing an initial set of execution times by α elements. The absence of the reproducibility of the measurement protocol makes the measurement-based WCET estimation process unable to converge. Namely, let X and Y with $|X| = |Y| = \alpha$ be two disjoint input sets produced by the measurement protocol at step $n-1$. If $pWCET(A_0 + (n-1)\alpha)$ produces two different results (when adding to $A_0+(n-2)\alpha$ either X or Y) then the set $A = X \cup Y \cup A_0+(n-2)\alpha$ diverges since $pWCET$ may produce two different WCET estimates.

In Figure 3.8 we illustrate a measurement-based WCET protocol with relations between the three properties. For instance from execution conditions (1) the

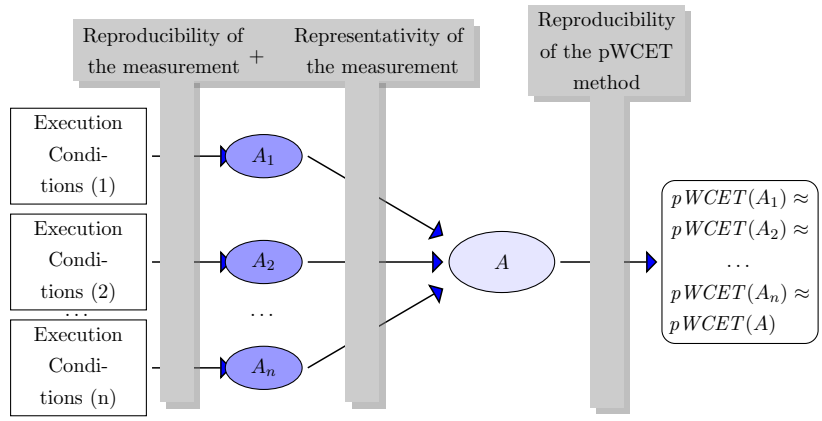

Figure 3.8: The impact of the reproducibility and the representativity on the convergence of a measurement-based WCET estimation.

measurement reproducibility ensures that an unique set of execution times A_1 is obtained.

The relations described previously are stated in presence of any WCET estimation method. If the WCET estimation method is transitive, then a stronger relation between the representativity and the reproducibility of a measurement protocol may be established.

Theorem 1 *If a measurement protocol is representative, then the measurement protocol is reproducible for any set of execution times with a cardinal larger or equal to k.*

Proof: We prove the reproducibility of a measurement protocol by contradiction. We suppose that the measurement protocol is not reproducible for any set of execution times with a cardinal larger or equal to k. This implies that there exist two utilizations $i \neq j$ of the measurement protocol A^i and A^j, with $|A^i| \geq k$ and $|A^j| \geq k$, such that

$$pWCET(A^i) \neq pWCET(A^j) \tag{3.2}$$

From the definition of the representativity we have that

$$pWCET(A^i) \approx pWCET(A) \tag{3.3}$$

and

$$pWCET(A^j) \approx pWCET(A) \tag{3.4}$$

From the transitivity of the relation $\approx$ and Equations (3.3) and (3.4) we obtain

$$pWCET(A^i) \approx pWCET(A^j) \tag{3.5}$$

We obtain the contradiction between Equation (3.2) and Equation (3.5) indicating that our initial hypothesis is not correct, thus we prove that the measurement protocol is reproducible.

3.5 Conclusion

In this chapter we presented the i.i.d. condition necessary for the classical use of EVT on data samples, as well as the tests used to verify this conditions. A clarification of the independence notion is also given.

In the absence of appropriate testing, the compositionality property may be introduced by the measurement protocol producing the execution times. The independence of a set of execution times may be obtained with an appropriate measurement protocol even in the presence of dependent programs. Therefore, we have provided the first intuitive mandatory properties for the convergence of measurement-based WCET estimation processes: reproducibility and representativity. The reproducibility describes the stability of the result w.r.t. different executions of the process. The representativity describes the existence of a (small enough) number of input measures that leads to the correct global result of the process. The provided examples are described in the context of probabilistic approaches but we expect these properties to remain true for any measurement-based WCET estimation process.

We identify an important thread of future work in providing proofs of compositionality for the existing measurement-based methods following the framework we have introduced in this chapter. In particular, we would like to study methodologies for proving and detecting convergence via reproducibility and representativity of measurement-based WCET estimation.

Chapter 4

pWCET estimation methodology

In this chapter we present our methods used for the pWCET estimation through the use of EVT. The main steps of using EVT are data collection, data selection and modeling. While in the previous chapter we detailed on the data collection step, in this chapter we concentrate on the selection and modeling process. Before presenting the mathematical definitions of the main distribution functions used in dealing with extremes, we give an insight in the central limit theorem. This notions can help for a better understanding of EVT and allow us to reference CLT for its similarities with EVT.

The central limit theorem (CLT) indicates that the mean of a large sample from a distribution has an approximate normal distribution. This is formalized as follows:

Theorem 2 ***(Central Limit Theorem)*** *Let $X_1, X_2, .., X_n$ be a sequence of independent identically distributed random variables with mean $E[X_i] = \mu$ and variance $V(X_i) = \sigma^2 > 0$.*
Let: $S_n = \sum_{i=1}^{n} X_i$ be the sum of the given random variables
Then: the following formula is verified $\frac{S_n - n\mu}{\sqrt{n\sigma^2}} \xrightarrow{D} N(0,1)$ as $n \to \infty$,
where $\xrightarrow{D}$ represents the convergence in distribution and $N(0,1)$ is the standard normal distribution.

Intuitively, the CLT says that if we collect multiple samples of a sequence of i.i.d. random variables and we compute the mean of these observation, the values obtained will belong to a normal distribution. The mean of the sample is an estimate and

the distribution of an estimate is called a *sampling distribution*. The power of CLT relies on the fact that it stays true for any probability distribution of the random variables under study.

The central limit theorem is a very important tool for thinking about sampling distributions - it tells us the shape (normal) of the sampling distribution, along with its center (mean) and spread (standard error). This allows us to infer about the average behavior of an event under study. If the event being monitored is the time taken by a program to execute (inside a system), then by using CLT on multiple sets of measurement, we are able to compute the average value of all possible execution times with a certain confidence.

There exists different versions of CLT which apply for non identical distributed data or data that contains dependencies. Its use is widely spread in various science branches like finance, computer science, engineering or medicine.

The same way CLT deals with behavior of mean, extreme value theory (EVT) deals with the behavior of maxima from a set of random variable. This is useful in the context of WCET, where understanding the evolution of the grater values of measured execution times is very important. The way CLT is able to estimate the shape, mean and spread of a mean sampling, EVT is able to compute the shape, scale and location of a maximum sampling.

EVT is composed of two limit theorems used for the study of extremal properties. We present these two theorems the way we use them for the estimation of a pWCET.

4.1 Generalized extreme value distribution

We are interested in the statistical behavior of $M_n = \max(X_1, X_2, .., X_n)$, where $X_1, .., X_n$ is a sequence of independent and identically distributed random variables having a common distribution F. The distribution function F is unknown and as a consequence the distribution on M_n ($P(M_n \leq x) = \{F(x)\}^n$) cannot be derived exactly. By using the extreme data observed we are able to estimate families of models for F^n, the same way that CLT justifies the approximation of mean samples with a mean distribution. EVT theory provides the asymptotic behavior of M_n as n increases.

Theorem 3 ***(Fisher-Tippett Types Theorem)*** *Let $X_1, X_2, .., X_n$ be independent random variables with the same probability distribution, and $M_n = max(X_1, X_2, .., X_n)$.*

If there exists sequences of constants $a_n > 0$ *and* b_n*, such that* $Pr\left\{\frac{M_n - b_n}{a_n} \leq x\right\} \rightarrow G(x)$*, as* $n \rightarrow \infty$*, for some non-degenerate distribution* G*, then* G *has as one of the following distributions:*

I. *Gumbel*

$$G(x) = \exp\left\{-\exp\left(-\frac{x-b}{a}\right)\right\}, \quad -\infty < x < \infty;$$

II. *Fréchet*

$$G(x) = \begin{cases} 0, & x \leq b, \\ \exp\left(-\left(\frac{x-b}{a}\right)^{-\alpha}\right), & x > b; \end{cases}$$

III. *Weibull*

$$G(x) = \begin{cases} \exp\left\{-\left(-\left(\frac{x-b}{a}\right)\right)^{\alpha}\right\}, & x < b \\ 1, & x \geq b. \end{cases}$$

for parameters $a > 0, b$ *and, in the cases of families II and III,* $\alpha > 0$*.*

Thus Theorem 3 states that *if* the distribution of the rescaled maxima $\frac{M_n - b_n}{a_n}$ converges, then the limit $G(x)$ is one of the three types, whatever the distribution of the variables. Collectively, these three classes of distribution are termed the *extreme value distributions*. In analogy with the CLT, regardless of the distribution of F, the three distributions are the only possible limits for the distribution of rescaled maxima $\frac{M_n - b_n}{a_n}$.

The most common distributions and the domain of attraction they belong to are presented in Table 4.1.

Domain	**Gumbel** $\xi = 0$	**Fréchet** $\xi > 0$	**Weibull** $\xi < 0$
Law	Normal	Cauchy	Uniform
	Lognormal	Pareto	Beta
	Exponential	Student	
	Gamma		

Table 4.1: The most common laws distributed by attraction domain.

Figure 4.1 presents a representation of the evolution of GEV differentiating the three families.

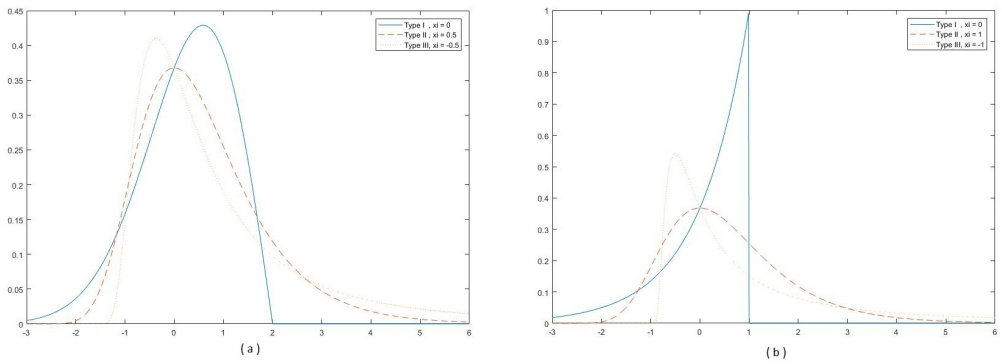

Figure 4.1: Examples of the GEV distributions with $\sigma = 1$ and $\mu = 0$. We mention that the intervals presented on the y axis are not the same for the two graphics.

Adopting one of the three types of distribution in order to estimate its parameters is method often encountered in early applications. In the real-time domain, there are papers in which Gumbel family is assimilated with the behavior of execution times of a system. This approach is prone to mistakes, since an initial exact choice cannot be made for events for which the behavior is not proved to always belong to a certain distribution law. This is the case for execution times where, from our knowledge, there is no proof of the distribution that they follow. Other weaknesses of this approach are that all future inferences on the estimated model depend on the type of distribution adopted and that an incorrect choice invalidates the obtained results.

Although the formulas of the three laws are different, they can be combined into a single parametrization containing one parameter ξ that controls the "heaviness"of the tail, called the shape parameter. This law is called the **generalized extreme value** (GEV) family of distributions and it is obtained by introducing a location, μ and scale, σ parameters:

$$G(x) = \exp\left\{ -\left[1 + \xi\left(\frac{x-\mu}{\sigma}\right)\right]_+^{-\frac{1}{\xi}} \right\} \tag{4.1}$$

where $-\infty < \mu < \infty, \sigma > 0$ and $-\infty < \xi < \infty$. The location parameter, μ determines where the distribution is concentrated, the scale parameter, σ determines its width. The shape parameter ξ determines the rate of tail decay (the larger ξ, the heavier the tail), with:

- $\xi > 0$ indicating the heavy-tailed (Fréchet) case

- $\xi = 0$ indicating the light-tailed (Gumbel, limit as $\xi \longrightarrow 0$) case

- $\xi < 0$ indicating the truncated distribution (Weibull) case

If we take into account the GEV, then the extremal theorem may be reformulated as follows: the asymptotic behavior of the maximum of a sufficiently large sample is a GEV distribution. In the same way as for the CLT, a max-stability property makes possible the convergence of the maxima and it allows to find the distribution it converges to.

In reality, we deal with a set of observations, in our case these observations are represented by multiple measurements of the execution of a program. The strategy used to perceive the data as observations of n random variables is by grouping them in blocks of equivalent length. This method is called **block maxima** and a trivial representation of its functioning can be seen in Figure 4.2. As mention in theorem 3, the convergence of G is achieved for $n \to \infty$. Therefore, one might think deciding on the block size (value of n) is trivially done by choosing a the greater value possible. Nevertheless, this choice is not obvious when we deal with a limited number of observations. A equilibrium point must be found between a block size too large that produces a number of maxima to small leading to large estimation variance, and a block size too small that can lead to bias in estimation and extrapolation.

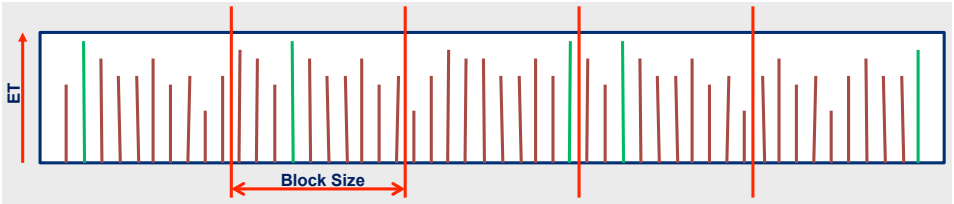

Figure 4.2: Block maxima method: the largest value for each block is kept.

Classically, when EVT is used in meteorology and the data is represented by measurements of natural phenomena, the block size is pragmatically chosen of size equal to a year worth of observation. When dealing with execution time measurements, such a reasoning can not be done and the selection of block size relies solely on the size of the data set and the estimation results. In this thesis, we propose an iterative testing of block size values and a selection based on two methods: visual confirmation of the fitting through quantiles plots and return level plots, and an automated method that uses goodness of fit (GOF) tests.

In order to estimate the three parameters of GEV from a set of maxima produced

through the block maxima method, we use the maximum likelihood estimation (MLE) method. This is a totally analytic maximization procedure that estimates the parameters of a statistical model from a set of observations. This is done by finding the parameter values that maximize the the likelihood of making the initial observations by the model with given the parameters. This method is highly used in practice and is implemented in many statistical tools (e.g.: R software packages, Matlab libraries, etc). In our case, we rely on the fevd() function from the extRemes package and the gev.fit() function from the ismev package. Both implementations are found in R software and have the same backbone, based on MLE. The use of both functions came as a result of continuous development of our global method of pWCET estimation.

To decide on the block size used for the final pWCET estimation, we verify multiple options by iteratively estimating the corresponding GEV parameters. Depending on the data size, we test as many values as possible for the block size smaller than $m/4$, where m is the data size. Once the GEV parameters were estimated for all the tested block sizes, we proceed to the selecting procedure. This selection is done in two ways:

1. **Visaul verification:** This approach relies on the use of return plots to decide upon accepting a block size or not. This method is subjective and requires "educated" human intervention. Regardless of these weaknesses, the use of graphical representations remains the method of choice for many statisticians when a decision needs to be taken. This approach is very efficient on eliminating block sizes that do not produce a good estimation. In Figure 4.3, we present an example of return level plots of the GEV distribution with different shape parameters (a), and an example of return plot of a GEV estimated model with negative shape and the execution times observations from which it was obtained (b).

2. **Goodness of fit testing:** We consider the use of Anderson-Darling GoF test to verify how close the estimated GEV model is from the maxima used in obtaining the GEV parameters. Other tests can be adapted for the same purpose, out of which we mention the Kolmogorov-Smirnov test, the Shapiro-Wilk test or the Cramér-von Mises test. We decided on the Anderson-Darling test because it is able to give more weight to the tails than the other ones [Stephens, 1974].

Once the block size is chosen and the GEV parameters are estimated, a model is created from which further extrapolation can be done. One result that can be taken from this method is computing the probability with which a certain event can happen. In the case of execution time, modeling the behavior of extreme observations will allow us to determine what the probability is for a certain execution time to be seen. The reverse can also be considered by identifying the maximum execution time with a given probability p of apparition.

The power of the described method consists in the fact that an estimation distribution can be calculated for very low probabilities as it is usually requested for safety critical embedded systems. As an example, in avionics industry, the failure rates are expressed in 10^{-x} per flight hour, where x can go up to 9 for DAL A systems.

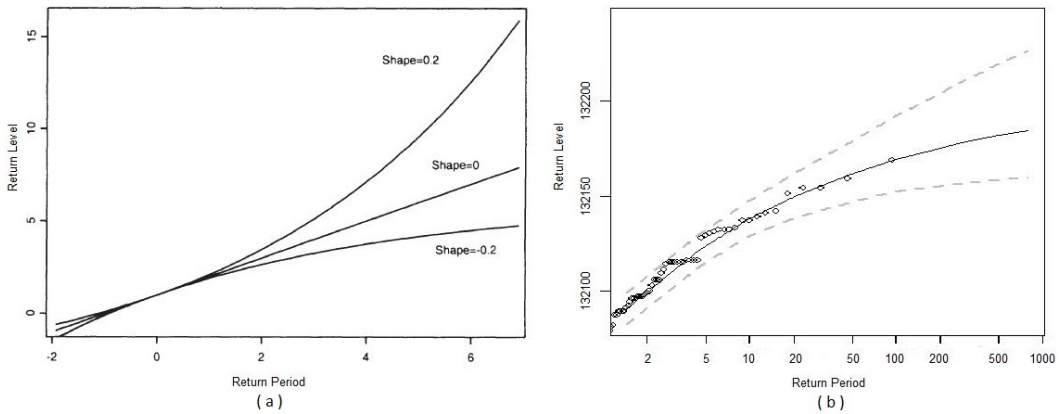

Figure 4.3: Examples of return plots for GEV: (a) Return plot for three models with different shape values. (b) Return level plot containing the model and the observed values for a model having negative shape and 95% confidence interval.

4.2 Generalized Pareto distribution

Besides the difficulty of choosing the good block size, the block maxima method has the weakness of ignoring valuable data if multiple extremes are found in the same block. This kind of scenario can be found in measurements of execution times when an abnormal functioning of the system can influence successive executions and increase their execution time. Despite efforts of isolation inside a system, program executions can still be subject to influences (OS, shared resources, etc) that can be

noticed in group at certain moments of the measuring. Even in a fully independent system with an unknown behavior multiple elevated execution times can be measured in the same block under normal condition.

In order to counteract the wastefulness of block maxima approach, the estimation based on threshold model can be used. The basic idea of this approach is to regard as extreme events the values that exceed some high threshold u. It is proven that these values have the property of belonging to a special family of distribution called **Generalized Pareto** (GP) distribution [Hosking and Wallis, 1987].

GP studies the behavior of the values exceeding u, a pre-chosen threshold sufficiently large to assure the asymptotic ground of the analysis. This method is introduced by Pickands and its advantage is that it uses the highest extremes available in a data set compared to GEV. In order to converge to a GP distribution, a set of i.i.d. random variables satisfies theorem 3.

Let $X_1, .., X_n$ be an sequence of independent and identically distributed random variable X, with $X_1, .., X_n$ having common distribution function F, and $M_n = max(X_1, X_2, .., X_n)$. We suppose that F satisfies the GEV theorem i.e. for n sufficiently large

$$P(M_n < x) \approx G(x)$$

with $G(x)$ member of the GEV family having ξ, μ, σ, the shape, the location and the scale parameters.

Let $u \in \mathcal{R}$ be the chosen threshold with $N_u = card\{i : i = 1, .., n, X_i > u\}$ the number of exceedances above u among the $(X_i)_{i \leq n}$ and let $Y_i = X_i - u > 0$ be the corresponding exceedances. We define F_u, the distribution of the values X_i exceeding u, conditional to the distribution F and the threshold u as follows:

$$F_u(y) \;=\; P(X - u \leq y | X > u) \;=\; \frac{F(y+u) - F(u)}{1 - F(u)}, \quad y \geq 0 \tag{4.2}$$

The Pickands-Balkema-de Haan theorem provides the asymptotic behavior of the distributions F_u; their intensities are approximated by the Generalized Pareto (GP) distribution and their frequencies by a Poisson point process. The GP distribution is expressed as a two parameters distribution (shape and scale):

$$H_{\xi,\sigma}(y) = \begin{cases} 1 - \left[1 + \frac{\xi y}{\tilde{\sigma}}\right]^{\frac{1}{\xi}} & \text{if} \quad \xi \neq 0 \\ 1 - \exp\left[-\frac{y}{\tilde{\sigma}}\right] & \text{if} \quad \xi = 0 \end{cases} \tag{4.3}$$

defined on $y : y > 0$ and $(1 + \xi y \setminus \widetilde{\sigma}) > 0$ where ξ and $\widetilde{\sigma} > 0$ are the shape parameters and scaling function (depending on the threshold u) of this function. The ξ parameter is equal to the ξ of the equivalent GEV distribution, while $\widetilde{\sigma} = \sigma + \xi(u - \mu)$ can be computed based on the threshold value and on the three GEV parameters.

Theorem 4 ***(The Pickands-Balkema-de Haan)*** *For distributions $F(x) = P(X \leq x)$, the GP distribution is the limiting distribution for the distribution of the excesses, as the threshold tends to τ_F (the upper bound of the distribution function). Formally, we can find a positive measurable function $F(u)$ such that:*

$$\lim_{u \to \tau_F} \sup_{0 \leq y \leq \tau_F - u} |F_u(y) - H_{\xi,\sigma(u)}(y)| = 0 \tag{4.4}$$

if and only if F is in the maximum domain of attraction of the extreme value distribution H_ξ i.e. $F \in MDA(H_\xi)$.

Definition 5 *(MDA) A distribution F is in the maximum domain of attraction of a distribution H, $F \in MDA(H)$, if for independent and identically distributed $X_1, X_2, ..X_n$ with probability distribution function F and $M_n = max(X_1, X_2, .., X_n)$, then we can find sequences of real numbers $a_n > 0$ and b_n such that the normalized sequence $(M_n - b_n)/a_n$ converges in distribution to H, where $M_n = max(X_1, X_2, .., X_n)$:*

$$\lim_{n \to \infty} \mathcal{P}\left(\frac{M_n - b_n}{a_n} \leq x\right) \;=\; \lim_{n \to \infty} F(a_n x + b_n)^n \;=\; H(x)$$

The way GEV can be separated in the three families of distributions, Gumbel, Frecét and Weibull, GP can also be seen as a combination of three families of distributions according to the value of ξ. Therefore, for $\sigma \to \infty$ GP corresponds to an exponential distribution with parameter $y \setminus \widetilde{\sigma}$, for $\widetilde{\sigma} > 0$ it corresponds to a Pareto distribution and for $\widetilde{\sigma} < 0$, we obtain a Beta distribution. Two examples of these three distributions can be seen in Figure 4.4.

According to the Pickands-Balkema-De Haan theorem, the distribution function F_u of the exceedance can be approximated by a GP distribution with the parameters ξ and $\tau = \tau(u)$ to be estimated.

In practice, the choice of the threshold u is difficult and the estimation of the parameters ξ and τ is a question of compromise between bias and variance. A lower u increases the sample size N_u but the bias grows since the tail satisfies less well

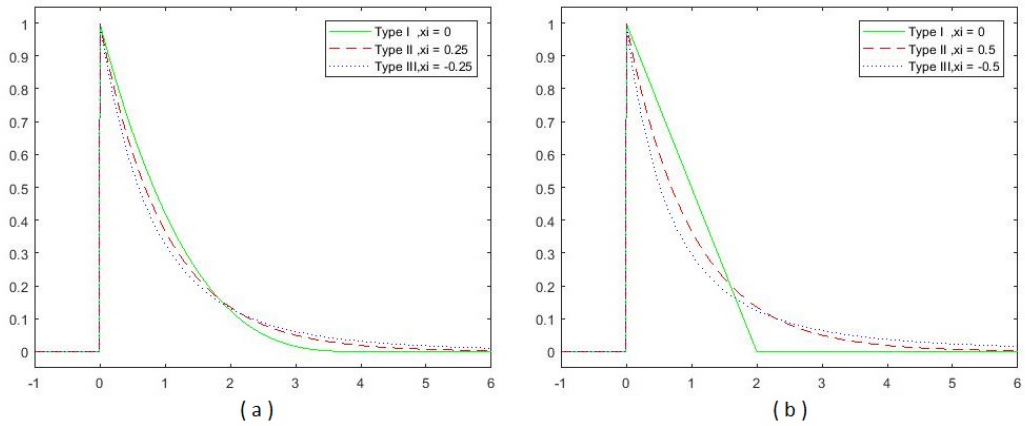

Figure 4.4: Examples of the GP distribution with $\widetilde{\sigma} = 1$ and threshold=0.

the convergence criterion (Equation(4.4)), while if we increase the threshold, fewer observations will be used and the variance will increases.

The method used to select the values over a tested level is called **peak over threshold** (PoT) and a graphical representation can be seen in Figure 4.5.

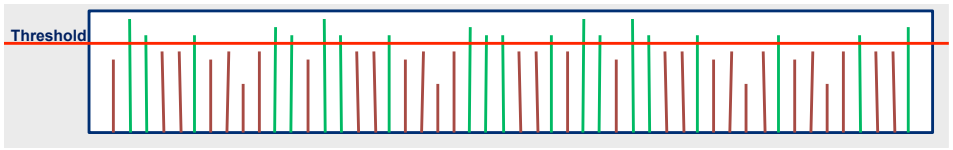

Figure 4.5: PoT keeps all values above a given threshold.

Generally, u is chosen graphically using the linearity of the sample mean excess function by plotting $\{(u, e_n(u)), X_{n:n} < u < X_{1:n}\}$ where $X_{1:n}$ and $X_{n:n}$ are the first and n_{th} order statistics of the studied sample and $e_n(u)$ is the sample mean excess function defined by:

$$e_n(u) = \frac{\sum_{i=1}^{n}(X_i - u)^+}{\sum_{i=1}^{n} 1_{X_i > u}}; \tag{4.5}$$

Thus $e_n(u)$ is the sum of the excesses over the threshold u divided by the number of data points which exceed the threshold u. It is an empirical estimate of the mean excess function which is defined as $e(u) = E[X - u|X > u]$. If the empirical plot seems to follow a reasonably straight line with positive gradient above a certain value of u, then this is an indication that the excesses over this threshold follow a GP with positive shape parameter. The weakness of this procedure is the fact that

a linearity of $e(u)$ can sometimes be hard to observe for a large enough number of observations leading to a variance increase.

Another method of deciding upon a threshold value is to estimate the model for a range of thresholds. Above a level u_0 at which the asymptotic motivation for the generalized Pareto distribution is valid, estimates of the shape parameter ξ should be approximately constant, while estimates of $\widetilde{\sigma}$ should be linear in u.

We resort to this second selection technique for determining a pWCET distribution starting from a sample of data. We explore all the threshold levels u from an interval. We decide upon this interval by plotting the GP parameter estimates (shape and scale) against a considerably large number of threshold, compared to the size of the analyzed data. We are looking for those points in which the parameters do not manifest a variability and for which a small confidence interval can be noticed. In figure 4.6, we present such a plot realized on real set of execution times with values from 1000 cycles to 1800 cycles and for which we can conclude that the thresholds to be verified are in the interval [1500-1600]. A common practice is to select the smaller value of such an interval. In our method, we test multiple values uniformly distributed from this interval. This decision is motivated by the need in precision necessary while working with critical real-time systems.

The parameters estimations are done by using the MLE method. And the final decision upon the threshold level is taken similarly as for the GEV method by looking at the return plot and applying GoF tests.

We mention the fact that the models obtained for GEV and GP are comparable at the tail level, the region of the distributions which interests us the most. Even though GP represents an estimation of the exceedance, by keeping the selected values from the PoT method in their initial form (without eliminating the threshold) we should be able to observe similar distribution as GEV. This rational is true in the case of a large enough sample of data and under condition of correct block size and threshold selection. In order to back up these selections and confirm the pWCET estimation when working with execution times, we propose a validation procedure based on the comparing between GEV and GP methods. We present this procedure in the next chapter.

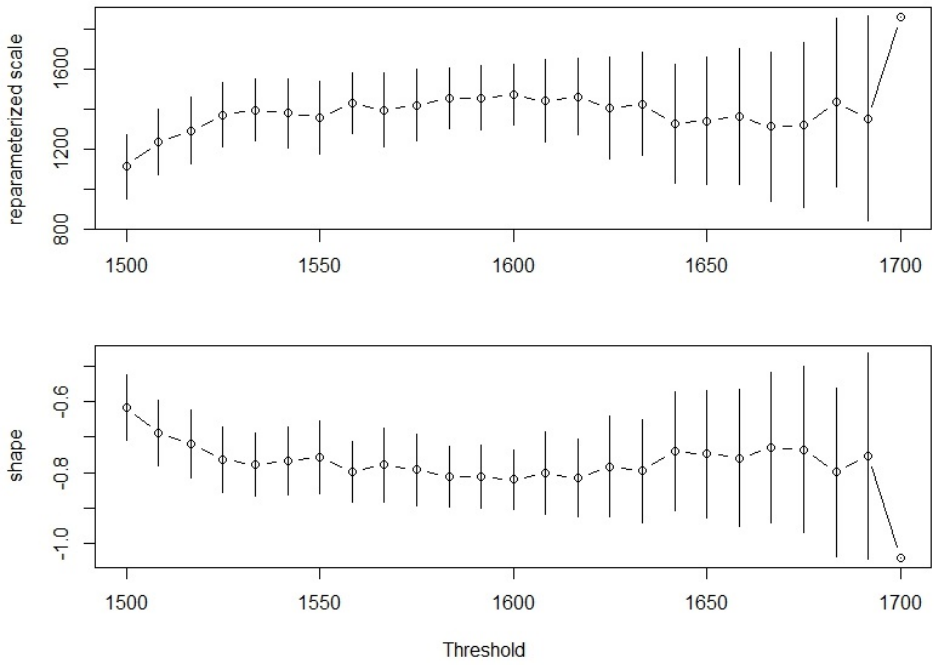

Figure 4.6: Example of parameter estimates against threshold.

4.3 Validation of statistical results

Testing a hypothesis through a statistical test is often done by using existing functions implementing statistical tests and the result is then interpreted by the user based on his/her own experience. We present in this section **a validation principle based on a voting procedure** increasing the confidence in such implementation (or to detect invalid results), beside the usual associated errors of the original statistical tests. Moreover, we deal in this chapter with the case where two independently obtained theoretical results (GEV and GP) exist for testing the same hypothesis. All tests are applied in parallel and independently, and the results are then compared. This validation method is presented with the purpose to be used on probabilistic worst case execution time estimation, but it can be applied on any kind of data for which its extreme behavior rises interest. In this section, we detail the principle starting from a sample of independent and identically distributed execution times, and in the next section we adapt it for the case when the data contains dependencies.

After applying the GEV method (see section 4.1) and the GP method (see section 4.2), we are able to obtain two models of extremes extracted from the same data

from which we are able to extrapolate upon the behavior of the system producing the data. The extrapolation that we are interested in refers to the distribution of the tail of GEV and GP distributions. These sections are able to tell us with what probability a certain execution time value will be encountered, and if we use return value equations, we will be able to predict in how much time (how many runs) we can see that value for the estimated probability.

Comparing GEV and GP pWCET estimates. The comparison of the GEV and GP curves is done using the distance between the two distributions computed with the continuous ranked probability score defined as $CRPS(GEV, GP) = \Sigma_{z=x_{min}}^{z=x_{max}}[f_{GEV}(z) - f_{GP}(z)]^2$. We consider in our experiments GEV and GP as sufficiently close when $CRPS(GEV, GP) \leq \epsilon$ with $\epsilon \approx 10^{-12}$. Other possible values of ϵ, based on, for instance, the criticality level the pWCET estimation, may be decided. In order to decrease the error introduced by such estimation, we recommend calculating the pWCET estimate as a combination of GEV and GP results. A joint pWCET estimate is obtained by choosing, for each probability, the smallest value between GEV and GP. The global view of the validation method can be seen in Figure 4.7.

A tool implementing this method is detailed in [Gogonel, 2014] and it is available on line at `inria-rscript.serveftp.com`[1].

The final result of the GEV and GP methods are represented by the equivalent distributions. In theory, the two distributions are approximately equal for a large number of blocks and a high threshold. In practice, we are limited by the number of observations being analyzed. Therefore, depending on the block size and threshold chosen and on the exactitude of the fitting, the two distributions might differ. To assure that the difference is minimal and that by choosing any of the two results we should have the same pWCET estimation, we use the CRPS function. If the result given by CRPS after the comparing of the two distribution is under a pre-established mean absolute error ϵ, we consider that the results are comparably close and that it makes no difference which one of them we accept. We take the decision of keeping the more optimist estimation. This choice is motivated by the fact that for a positive shape value (Fréchet for GEV and Beta for GP) the estimations can be high and keeping the smallest values will still produce safe WCET bounds.

In order to demonstrate the process of estimation we present a comparison be-

[1]The web page requires a secured connection using the login aoste and password aoste.

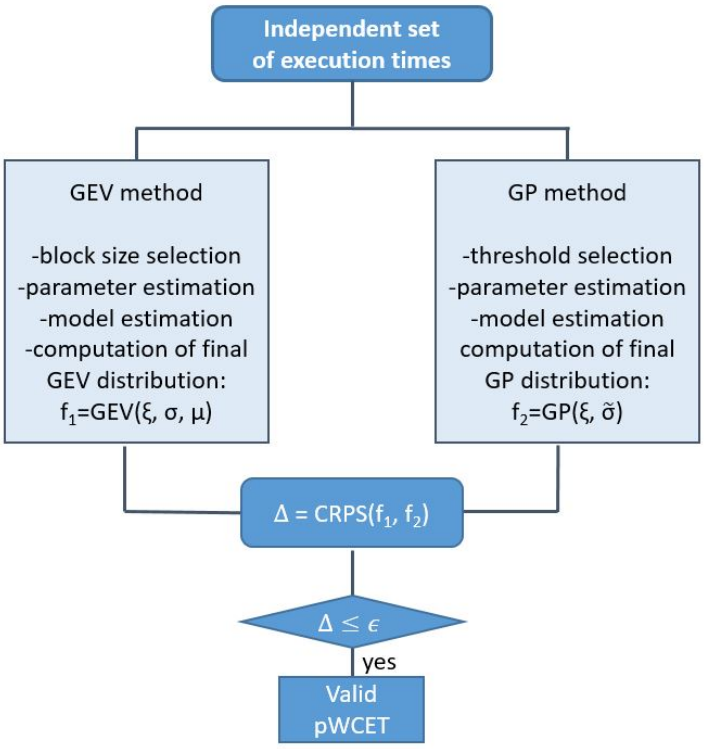

Figure 4.7: A global view of the pWCET estimation using GEV and GP.

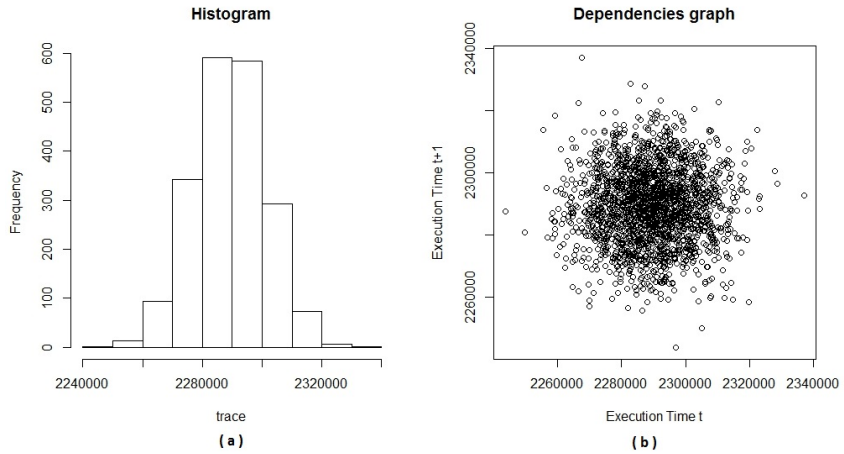

Figure 4.8: Histogram plot (a) and lag test (b) of the data used for the comparison with existing methods.

tween our work and three existing methods existing in the literature. We are using a data set of 2000 execution times, obtained from executing an avionics application on COTS hardware. Every software run has the same input and the cache replacement

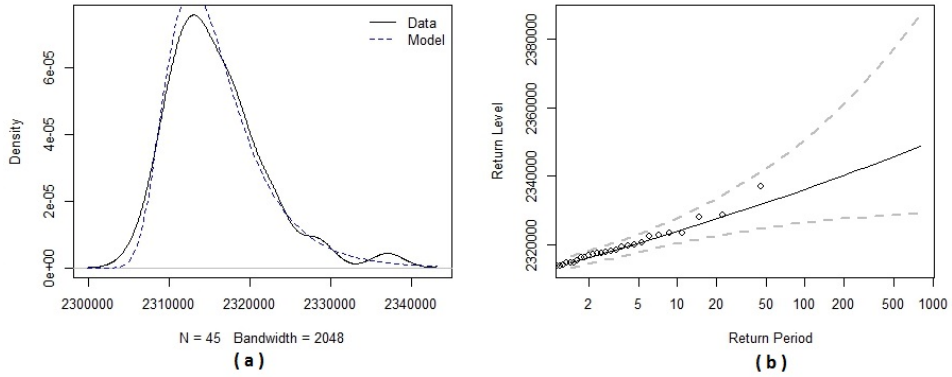

Figure 4.9: The estimated model (a) and the return level plot (b) obtained using our method on the data used for the comparison with existing methods.

protocol is LRU. Even though all execution times are obtained following the same procedure, the hardware inserts a considerably amount of variability that allow us to apply EVT for the estimation of pWCET. The data is measured in CPU cycles and is comprised in the interval [2243736,2336992], while the standard deviation is $s = 12116.12$. The variability of the execution times and the result of lag test can be seen in Figure 4.8. The i.i.d. tests for the used data are passing and the GEV distribution is chosen over GP distribution. A depiction of the estimated model over the used data and the return plot can be seen in Figure 4.9. Our method chooses a block size of 45 (selecting 45 values from the data as maxima), and estimates a GEV distribution having the shape parameter parameter $\xi = 0.05413$, indicating a Fréchet distribution. The choice of the execution times trace is totally random, the only criteria used was that the data passes the independence test. The purpose of this comparison is to spot the weak point of existing techniques and to show how we avoid them.

Comparison to the existing EVT-based estimations

- *Comparison to the seminal work of Edgar and Burns [Edgar and Burns, 2001]*: The work described in [Edgar and Burns, 2001] fit a Gumbel curve that fits the set of execution time traces by using the entire set of data (no block maxima method is applied) . Our method's estimation (in red) is compared with the curve obtained using the method of Edgar and Burns (in black in Figure 4.10). By fitting the raw data on the Gumble distribution, theorem 3 is not applied correctly. The only case in which this can be done is when we know that the data used is composed of extremes from a larger set of measurements. The

weakness of fitting the data directly to Gumbel distribution rises from the fact that this choice must be backed up. In the example given, we show that the best fit comes from a GEV with a positive shape, producing a heavier tail than a Gumbel distribution. Therefore, Edgar and Burn's method introduces a source of uncertainty: when the data is a better fit to a Weibull distribution, the method will be pessimistic, while data fitting a Fréchet distribution produces an optimistic method. Even though it introduces an eventual over provisioning of the system, being pessimistic can be acceptable compared to an optimistic estimation when the risk of deadline misses might be increased.

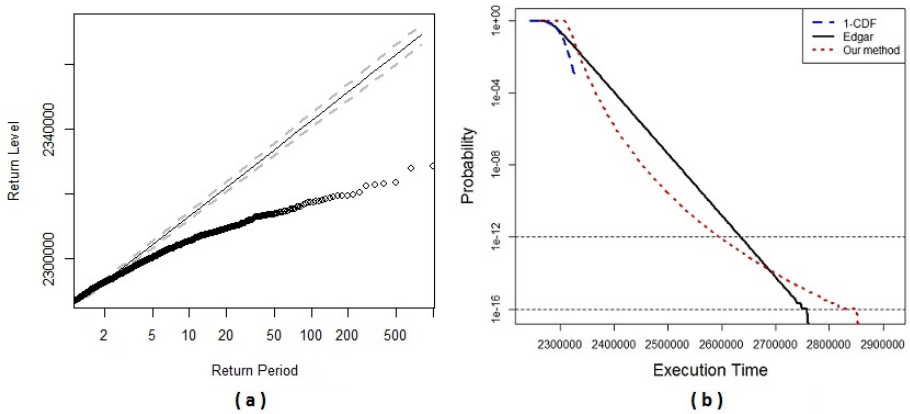

Figure 4.10: Comparison with Edgar's work: (a) return level plot for Edgar's method, (b) our pWCET estimation (in red) against the estimation obtained according to adversary method.

- *Comparison to the work of Hansen et al. [Hansen et al., 2009]*: The work described in [Hansen et al., 2009] searches for the first Gumbel curve that fits the set of execution time traces by using block maxima reasoning. Our method finds the curve of Hansen et al. and compares it to a tighter fit on the GEV distribution (see Figure 4.11). The same weaknesses as in Edgar and Burn's method appear in Hansen's method due to a fitting to Gumbel distribution. Hansen's method would coincide with ours when the data under analysis fits best a Gumbel distribution. Even though the Gumbel model is the one encountered most often when applying EVT, this is not an obvious choice for execution times. The results presented in [Lu et al., 2011, Berezovskyi et al., 2014, Berezovskyi et al., 2016] keep also the first Gumbel.

- *Comparison to the work of Cucu-Grosjean et al. [Cucu-Grosjean et al., 2012]*:

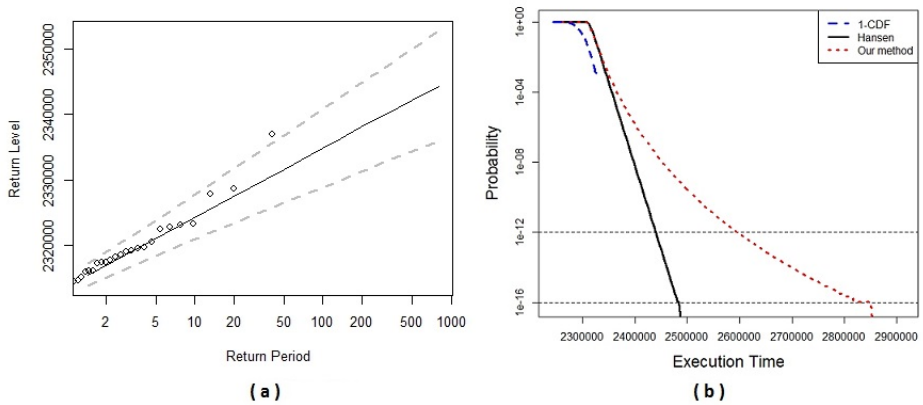

Figure 4.11: Comparison with Hansen's work: (a) return level plot for Hansen's method, (b) our pWCET estimation (in red) against the estimation obtained according to adversary method.

The work described in [Cucu-Grosjean et al., 2012] searches for the Gumbel curve that fits the set of execution time traces while iterating through block sizes that are multiples of 50. In reality fixing the block size to only a limited number of values might not be very accurate. Our method improves on the one proposed in Cucu's paper by exploring a higher number of block sizes and by eliminating the limitation on a Gumbel distribution. Cucu's method produces for our example a result closer to ours (see Figure 4.12), but uses a smaller number of maxima which increases the variance.

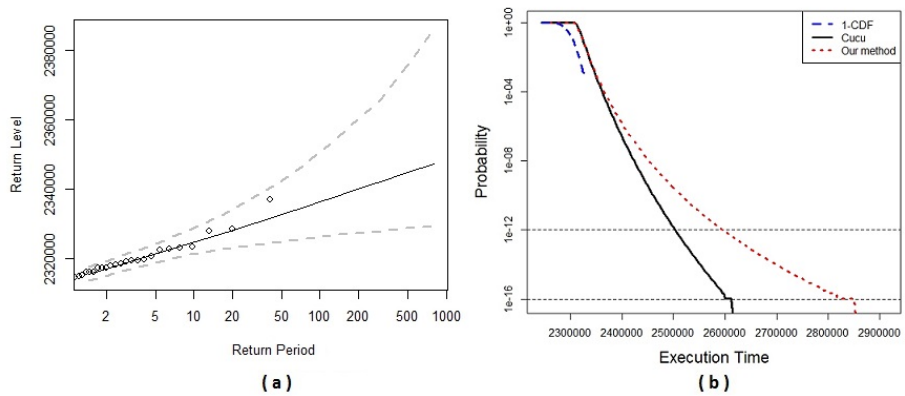

Figure 4.12: Comparison with Cucu's work: (a) return level plot for Cucu's method, (b) our pWCET estimation (in red) against the estimation obtained according to adversary method.

Theoretical convergence of the two EVT branches to the same result

EVT indicates that *the GEV and the GP estimations should theoretically provide the same curve* [COLES, 2001]. The convergence of a pWCET estimation is based on the comparison between the two pWCET estimates obtained independently by the two branches. Their fitting indicates that we are acceptably close to the theoretical curve.

4.4 The pWCET estimation from dependent execution times

We propose previously a voting procedure for the pWCET estimate from dependent execution times while using two independent results based on GEV and GP. This joint utilization of the two methods is differentiating the current contribution from existing ones by the fact that it considers the possibility of execution times behavior to be represented by the any GEV distribution. This excludes the limitation to Gumbel-only solutions.

The (statistical) dependences that a set of values may experience in general are (i) *local dependences*, where successive values are dependent in time, but values farther apart are independent; (ii) *long term trends dependences*, where the underlying distribution changes gradually over time; and (iii) *seasonal variation dependences*, where the underlying distribution changes periodically through the time. The voting procedure handles all three general cases by extending the method described in Section 4.3 (see Figure 4.13) to dependent execution times.

GEV for dependent execution times In general the block maxima is considered to ensure the independence of the remaining execution times [Lu et al., 2011] even if the original set of execution times is dependent. For those cases when the execution times left after the block maxima step are dependent, we use an extension of GEV for dependent data. This extension is built by calculating a fourth parameter: *the extremal index* θ describing the dependence degree of the execution times [COLES, 2001]. Lower is θ, higher is the correlation between the execution times. For independent execution times $\theta = 1$. The GEV is calculated then as follows:

$$G^{\theta}(x) = \exp\left\{ -\left[1 + \xi\left(\frac{x-\mu^{\star}}{\sigma^{\star}}\right)\right]_{+}^{-\frac{1}{\xi}} \right\}^{\theta} \tag{4.6}$$

where $\mu^\star = \mu - \frac{\sigma}{\xi}(1-\theta^{-\xi})$ and $\sigma^\star = \sigma\theta^\xi$. where μ is the a location parameter, σ is the scale parameter and ξ is the shape parameter.

GP for dependent execution times The GP version described in Section 4.2 requires always an extension for dependent execution times as threshold exceedances occur in groups [Griffin et al., 2015], implying that one large value is likely to be followed by another. We consider declustering of the execution times, that corresponds to filtering the dependent observations to obtain a set of threshold excesses that are independent. This filtering is done as follows.

We first split the set of execution times in clusters. A cluster is obtained as a set of $k+2$ consecutive observations such that $X_t < \nu$ and $X_{t+i} > \nu, 1 \leq i \leq k, X_{t+k+1}$, where ν is a threshold value that we vary from the highest level to the lowest.

For each cluster we keep those execution times larger than the threshold of that cluster. We obtain a set of independent execution times and the GP for independent execution times is then applied.

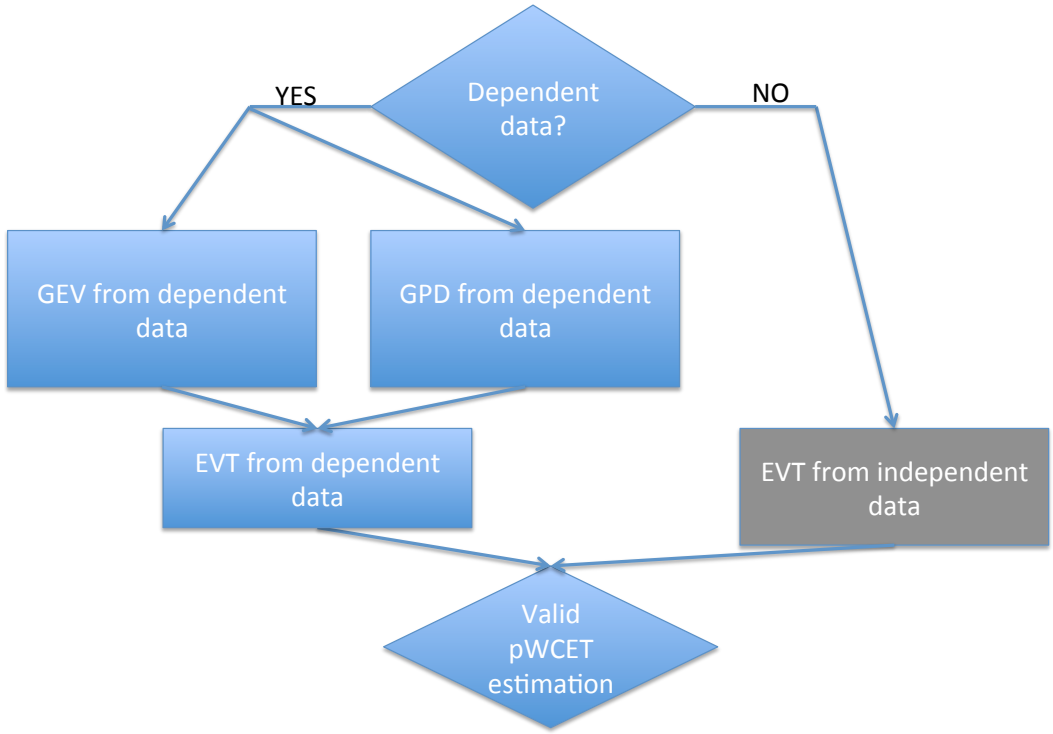

Figure 4.13: The two branches of EVT for dependent data and their relation with EVT for independent data.

The convergence of the two methods is ensured by EVT that indicates that the

GEV and the GP estimations should theoretically provide the same curve. Once fitted, one may conclude that the obtained curve is acceptably close to the theoretical curve.

4.5 Small variability data

In the case of execution times with small variability (see Figure 4.14), the pWCET estimation is modified with respect to previous method. More precisely if the execution times are grouped in two or three sub-sets, the pWCET will be obtained as the joint pWCET estimate of GEV and GP by choosing for each probability the smallest value between GEV and GP. This choice is motivated by the fact that the previous version provides pessimistic pWCET estimation through its GP branch. Indeed the existence of few sub-groups of possible values "forces" GP to keep in general the values from the largest value sub-set (sub-set 1 in Figure 4.14). GEV is not sensitive to this grouping effect and the joint pWCET estimation will be mainly based on GP. The choice of the smallest value between GEV and GP is kept.

For detecting this specific case we introduce a new utilization for the statistical test of k-means algorithm [Hartigan, 1975] to check automatically the small variability of execution times. The k-means algorithm is an algorithm clustering n objects based on attributes into k partitions, where $k < n, k \in \mathcal{Z}, k > 0$. The aim of the k-means algorithm is to divide N data points into K disjoint subsets S_j containing data points such that the sum-of-squares criterion $J = \sum_{j=1}^{K} \sum_{n \in S_j} |x_n - \mu_j|^2$ is minimized where x_n is a vector representing the n^{th} data point and u_j is the geometric centroid of the data points in S_j. The steps of the k-means algorithm are the following:

- **Step 1** We initialize k as an intuitive number of clusters
- **Step 2** We put any initial partition that classifies the data into k clusters as follows:
    1. We take the first k training sample as single-element clusters;
    2. We assign each of the remaining $(N - k)$ sample to the cluster with the nearest centroid. After each assignment, we recompute the centroid of the gaining cluster;

- **Step 3** We take each sample in sequence and compute its distance from the centroid of each of the clusters. If a sample is not in the cluster with the closest centroid, we switch it to that cluster and update the centroid of the cluster gaining the new sample and the cluster losing the sample.
- **Step 4** We repeat Step 3 until testing distances of the samples to centroid does not cause a new assignment.

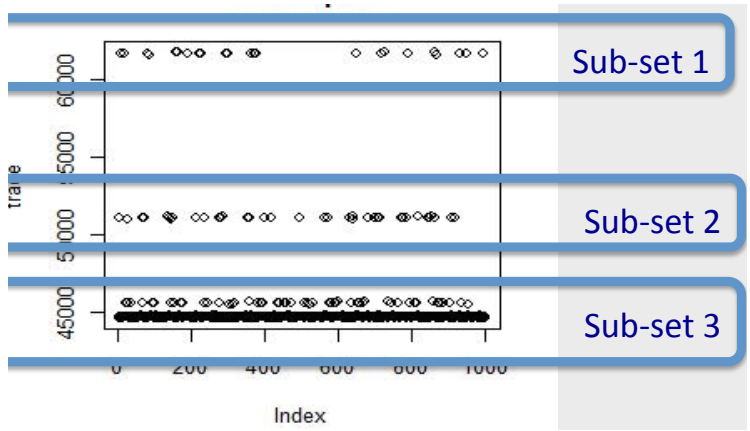

Figure 4.14: Set of execution times with small variability.

4.6 Conclusions

In this chapter we presented the two methods (GEV and GP) used for pWCET estimation when the used samples are independent. A validating procedure is introduced in order to decide on the consistency of the two estimations. We also propose solutions for the cases when the data under analysis present manifest dependence or low variability. Some of the technical details of these methods can be found in the Appendix of the thesis. Result of using these methods can be found in the part of the thesis containing the practical results (Chapter 5)

Chapter 5

Experimental results

In this section we present our results while studying the WCET estimation the proposed methods on different systems. We focus mainly on the estimations obtained for software provided by the Airbus company. The experiments are performed for multiple platforms and different configurations. Also, during three years, the period of the thesis, we encountered different stages of development for the methods used. Therefore, results might not always follow the same type of representation or the same level of detail. Nevertheless, we will explain at every moment the changes in presentation and the reasons of their occurrence.

5.1 Analysis of benchmarks on multiprocessor architectures

A first stage in developing a functioning pWCET estimation tool is its testing on existing benchmarks. We use the Mälardalen Benchmark [Gustafsson et al., 2010] and executed them on an Aurix board.

In this section we detail only the pWCET estimation for the program *prime* from this benchmark suite. This choice is made in order to better exemplify the properties of representativity and reproducibility (see section 3.4). The program *prime* verifies the primality of a number given as input. We observe here a direct link between the inputs used for this program and the execution times measured for the corresponding executions. This program is used for a more detailed presentation as its structure allows injecting dependences directly through the values of the number checked (to be prime or not).

The Aurix board is composed of 3 cores: 2 TC1.6P and one TC1.6E. The major blocks of the Aurix processor are shown in Figure 5.1. During the experiments, we use only one core and the programs are executed in isolation in bare metal mode (no other programs are on the other cores and no operating system is used).

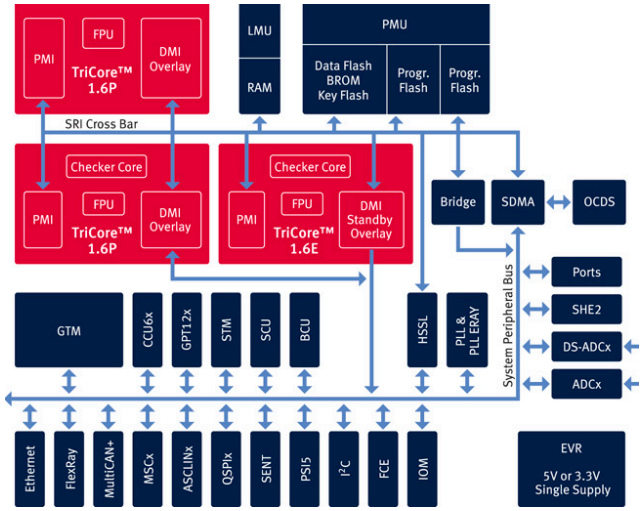

Figure 5.1: Tricore Aurix Block Diagram.

Independent execution times. We execute the *prime* program for 100 times with input values for the number (to be checked as prime) randomly picked from the set $\{13, \cdots, 5995\}$. We obtain independent execution times given in Figure 5.2. The execution times vary from 0 to 500 cycles. A representation of the lag test for this set of execution times is given in Figure 5.3. The lack of pattern in this figure is also confirmed by the run test results with a p-value at $0.16 > 0.05$ indicating the independence of the execution times. In this case the method presented in Section 4.3 is applied for the pWCET estimation and we can read the value 526 cycles for a probability of 10^{-12} from the resulting distribution.

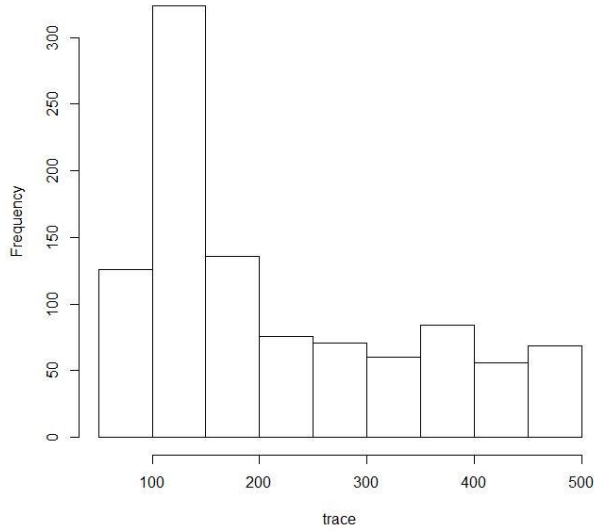

Figure 5.2: Distribution of independent execution times for the program *prime* on the Aurix architecture.

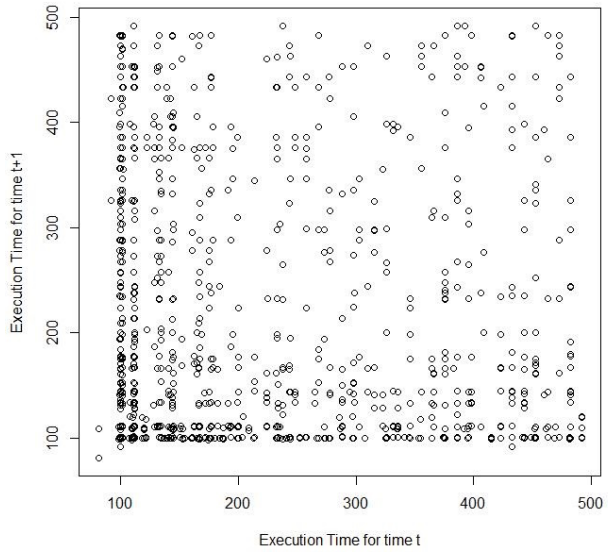

Figure 5.3: Lag plot for independent execution times for the program *prime* on the Aurix architecture.

Figure 5.4: pWCET estimation of the program *prime* on the Aurix architecture from the measurements of Figure 5.3.

As suggested in section 3.1, the analysis results of a system depend on the consistency of that system. In other words, we cannot pretend that the pWCET curve obtained for the previous example applies for any usage of the system. The simple change of the input interval automatically changes the execution time profile of the system. Also, the change in the input order or in the way the inputs are chosen, despite belonging to the same interval might influence the pWCET estimation. This occurs because the condition of reproducibility is not satisfied and as a consequence, the representativity criteria is violated. We exemplify this with another set of inputs that produces execution times containing statistical dependence.

Dependent execution times. For the second set of experiments with *prime* on Aurix board we consider dependent input values for the number (to be checked) from the set $\{2010, \cdots, 5000\}$. We obtain a set of execution times described in Figure 5.5, which is subject to dependences. The lag plot in Figure 5.6 indicates patterns and the p-value for the run test ($8.77e-22 < 0.05$) confirms the weak dependences. We calculate the value $\theta = 0.77$ confirming also the utilization of EVT for dependent execution times (as described in Section 4.4). We obtain a pWCET estimate of 518 for a probability of 10^{-12}, as seen in Figure 5.7.

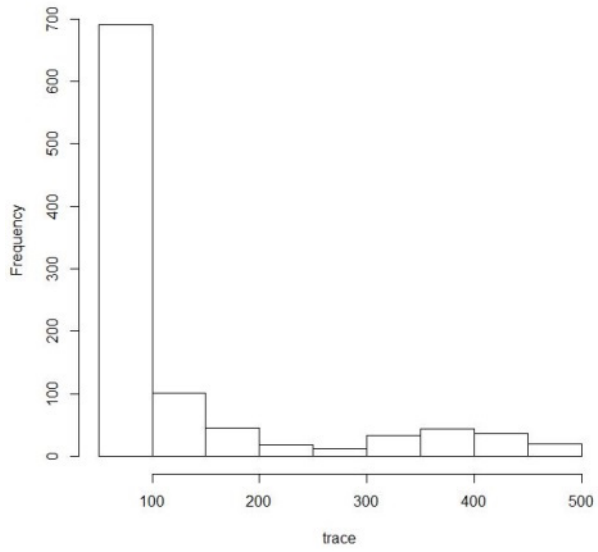

Figure 5.5: Dependent execution times for the program *prime* on the Aurix architecture with independent input values.

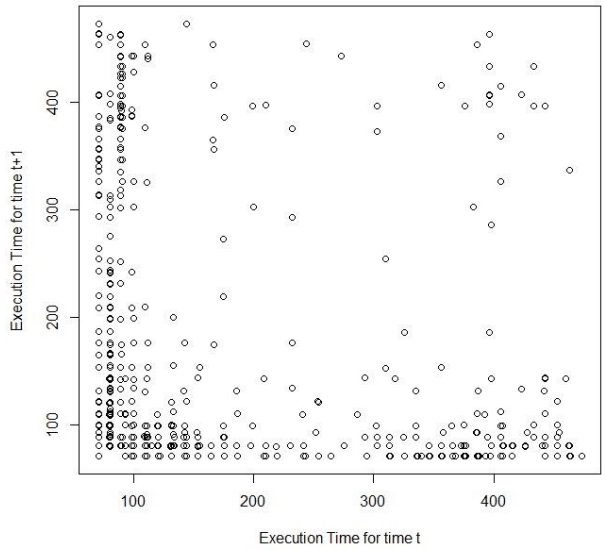

Figure 5.6: Lag test for dependent execution times for the program *prime* on the Aurix architecture while using independent input data.

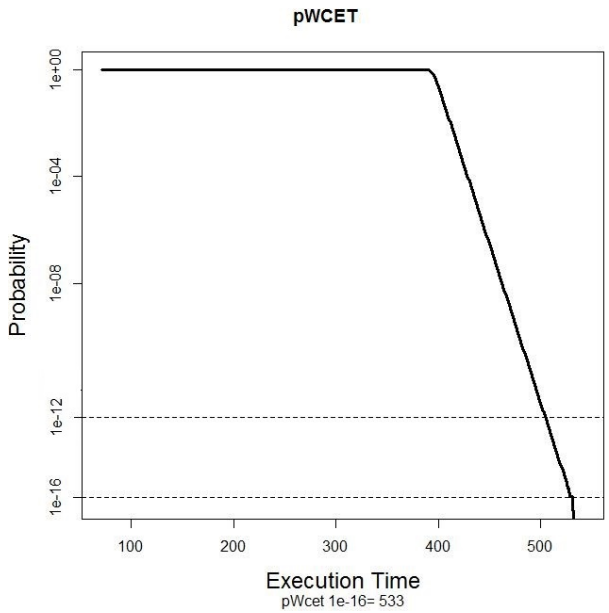

Figure 5.7: pWCET estimation from dependent execution times of the program *prime* on the Aurix architecture.

Even though the pWCETs at probability 10^{-12} are relatively close (1,5% difference), we cannot rely on any of these values or on an interval containing these values as a general result for pWCET analysis of the given system. Nevertheless, we can guarantee the first pWCET curve as an estimation for the use of independent and randomly picked inputs from the interval $\{13, \cdots, 5995\}$ for the *prime* executing on the Aurix platform. The same conclusion can be taken for systems having inputs with the same types of dependence as in the second example (and coming from the same interval).

This example is used as a justification for the best use of our method. Therefore, EVT is able to guarantee the timing analysis of a system in one of the following conditions:

- The analysis is done on a system behaving in the same way as at run-time (same platform, same software) and using the same input vector for which certification is needed. The use of a worst case input vector is accepted.

- The analysis can be done on basic blocks of the software executing in the targeted platform. Combining the results of multiple basic blocks analyses

can be done through convolution. The result is capable of upper bounding the end to end analysis with the cost of injected pessimism.

The above argumentation applies for the cases in which the architecture introduces enough uncertainty and variability in the execution time measurements. In the case of a predictable platform the use of probabilistic analysis is unnecessary and/or inappropriate. By using complex hardware architecture seen as black boxes, the variability necessity is satisfied. Such complex architecture are represented by COTS hardware. This motivates in the next section the utilization of different COTS hardware for obtaining measurement of execution times.

In Table 5.1 we provide pWCET estimations on Aurix board for other programs of the Mälardalen Benchmark. We use an evaluation formula to quantify the ratio between the pWCET estimate and the maximum observed value $maxET$. This ratio is calculated as follows:

$$eval = \frac{pWCET - maxET}{maxET} \tag{5.1}$$

Trace	ind	Method	pWCET	max	eval
fir	*indep*	*GEV*	162022	135457	0.20
lcdnum	*dep*	*GEV*	2577	1171	1.20
minver	*indep*	*GEV*	61501	43380	0.42
qurt	*indep*	*GP*	34478	28517	0.21
recursion_mutual	*indep*	*GEV*	762	346	1.20
select	*dep*	*GEV*	1689	1325	0.27
sqrt_input_inputnorm	*indep*	*GEV*	43702	38553	0.13
sqrt_input_random6	*indep*	*GP*	43702	38553	0.13

Table 5.1: pWCET estimation on Aurix for some Mälardalen Benchmark programs.

5.2 Avionics application analysis

In this section we detail the pWCET estimation of two Airbus applications. We provide below a presentation of the applications as well as the hardware used for obtaining execution time measurements for the analysis. We present here only a part

of all the data analyzed during the three years of thesis, due to space limitation and confidentiality reasons.

Despite the benefits that the IMA concept brings to the avionic industry, a number of problems arise due to the complexity increment of putting together different applications with mixed criticality levels into a single computer:

Hardware complexity: The IMA concept requires using powerful processors in order to cope with the performance requirements of the different applications. On modern single core processors, counting cycles does not work anymore as it is currently the case for some timing analysis techniques. Moreover caches, pipelines, branch predictors and all features that enhance performance on average, by exploiting execution history, make that there is much more to the execution time of a piece of code than only its execution paths. The large amount of state that a modern processor incorporate leads to combinatorial explosion when trying to enumerate all possible histories for the execution of even simple pieces of code. There are strength-reduction techniques such as abstract interpretation but these usually require knowledge of all events in the computer, which is not compatible with features hard to get rid of, such as dynamic RAM, paginated MMU, copy-back caches or I/O interrupts. Introducing multicore processors to the discussion does not solve any of these single core issues but add even more sources of interferences across cores.

Software complexity: timing analysis is non-trivial on monolithic sequential programs of limited size, it becomes difficult on large programs, is not easy to break down in order to benefit from program modularity, and there are few techniques for dealing with multithreaded programs.

Incremental qualification: the timing analysis of an application should be independent regardless of the execution environment in which this application runs. Unfortunately, the timing behavior of a program can be affected due to interferences generated by sharing resources, which is obviously the case when switching to multicore processors. Thus, static timing analysis of several programs that share a computer is difficult or even impossible in this context.

A larger description of the IMA concept can be found in section 2.5.1. We recall that one of the key features of IMA is partitioning and that the unit of partitioning is called partition. IMA incremental certification is essentially based on the following partitioning properties: spatial, temporal and communication.

5.2.1 Application presentation

A great part of avionics application is created with automatically generated code . This is composed of linear code executed periodically by a sequencer. This procedure increases the predictability of the application and favors static analysis. Despite this code structure, due to hardware complexity or OS interferences, we can still observe variability in the execution time measurements of these applications.

The avionics case study used in this thesis is comprised of two real A653 applications hosted on IMA Line-Replaceable Module (LRM): **Weight and Balance Back-up Computation (WBBC)** is part of the flight control system of the aircraft. Its main functions are:

- Computation of an independent estimation of center of gravity position, to secure the aircraft from center of gravity excursions
- Supply to the whole aircraft computers, some weight and center of gravity values to monitor
- Supply to the whole aircraft computers, some weight and center of gravity values that could be used as backup
- Generation of warnings and caution functions to prevent from center of gravity excursions

Flight Control Data Concentrator (FCDC) is part of the flight control system of the aircraft. Its main functions are:

- Concentrating data from primary and secondary flight control units.
- Concentrating data from Weight and Balance Backup Computation System.
- Generating Warnings to be displayed by the Flight Warning System.
- Generating maintenance messages to the Centralized Maintenance System (CMS).
- Transmitting messages to the Digital Flight Data Recorder from primary and secondary Flight Control Units.

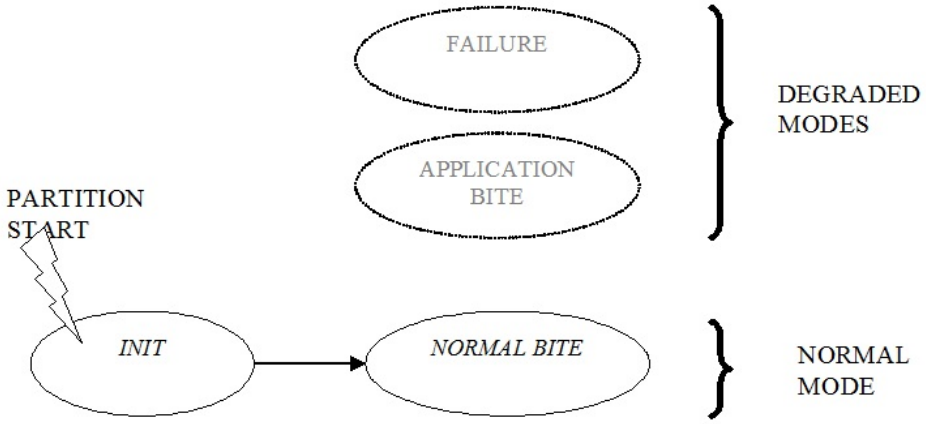

Figure 5.8: WBBC functional modes.

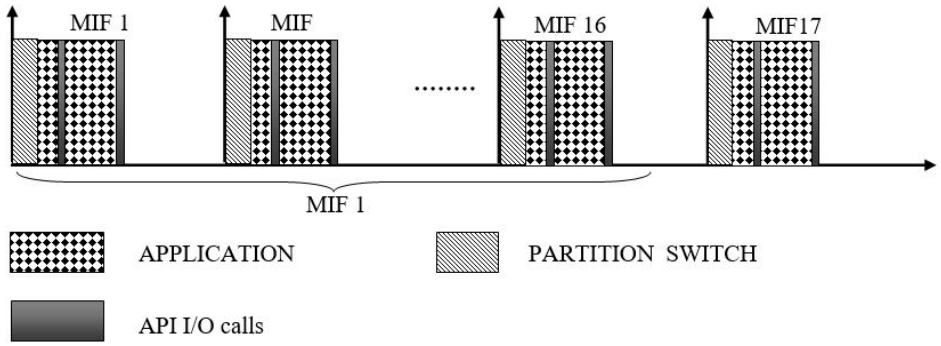

Figure 5.9: WBBC Dynamic Behavior.

MIFs	Cycle C1	Cycle C2	Cycle C3	Cycle C4	Cycle C5
1	C1	C2_1	C3_1	C4_1	C5_1
2	C1	C2_2	C3_2	C4_2	C5_2
3	C1	C2_1	C3_3	C4_3	C5_3
4	C1	C2_2	C3_4	C4_4	C5_4
5	C1	C2_1	C3_1	C4_5	C5_5
6	C1	C2_2	C3_2	C4_6	C5_6
7	C1	C2_1	C3_3	C4_7	C5_7
8	C1	C2_2	C3_4	C4_8	C5_8
9	C1	C2_1	C3_1	C4_1	C5_9
10	C1	C2_2	C3_2	C4_2	C5_10
11	C1	C2_1	C3_3	C4_3	C5_11
12	C1	C2_2	C3_4	C4_4	C5_12
13	C1	C2_1	C3_1	C4_5	C5_13
14	C1	C2_2	C3_2	C4_6	C5_14
15	C1	C2_1	C3_3	C4_7	C5_15
16	C1	C2_2	C3_4	C4_8	C5_16

Figure 5.10: WBBC SCADE Nodes Cycles Scheduling.

WBBC

WBBC software has four functional modes, as shown in Figure 5.8. Each functional mode consists in a specific subset of WBBC functions to be fulfilled. Only one mode can be active at a time.

- *INIT*. Call services to create A653 resources and start the partition processes.
- *NORMAL_BITE*. Normal execution, i.e. the execution time without any internal error conditions, of functional logic .
- *APPLICATION_BITE*. Degraded mode of WBBC functions.
- *FAILURE*. Failure mode following internal faults detection within WBBC software.

Degraded modes of WBBC software, i.e. FAILURE and APPLICATION BITE, are uninteresting from the execution time point of view and, therefore they are out of the scope for applying our methods. In these two modes a great part of software functions are dropped and only a sequential and predictable code is kept which from the execution time point of view doesn't manifest any variability. WBBC case study consists in running processes of NORMAL_BITE mode. In this mode, WBBC executes the A653 processes called APPLICATION. The APPLICATION process sequences I/Os acquisitions and emissions and execution of automated generated SCADE logic cycles code. Figure 5.9 illustrates how those processes are scheduled within a Major Frame composed of 16 Minor Frames periods. Note that MIF 17 corresponds to the MIF 1 of the next MAF.

APPLICATION is a periodic A653 process, with priority set to highest priority. It is composed out of Sampling Ports Acquisitions, Input Payload Unformatting, SCADE nodes Execution, Output payloads Formatting and Sampling Ports Emissions. The SCADE nodes scheduled during execution of the APPLICATION component of WBBC are organized in 5 major cycles, from C1 to C5 with periodicity constraint set to $C_n = 2^{n-1}$ MIFs, subdivided in minor cycles. Such a periodicity constraint defines the 16 MIFs required by the FCDC application: one each MIF, one every two MIFs, etc. The complete major and minor cycles allocation is described in Figure 5.10. In other words, during each MIF a pre-established sequence of SCADE code is executed. For example, during MIF9, the code of C1,C2_1,C3_1, C4_1 and C5_9 is executed. The 16 MIFs are executed in a round robin manner implemented by a switch-case code. Our main goal is to determine a pWCET curve for each MIF starting from each one's set of measurements. According to the developer strategy, all MIFs can contain approximately the same amount of code or no. This is not the case for WBBC, and we can say that the MIFs are not distributed uniformly.

The complete WBBC software is composed of 100 klines of code without comments, 13,000 lines being produced manually. WBBC application communication layer is through the AFDX network, and few discrete signals.

The input data used for the WBBC application is represented by a set of input vectors that were studied and generated inside AIRBUS in order to stress the application as much as possible and to obtain the produce the maximum observed execution times. These inputs stay the same all long of the execution and are used by

the blocks any time a MIF/MAF starts. Therefore the analysis of WBBC program is made under the same input conditions.

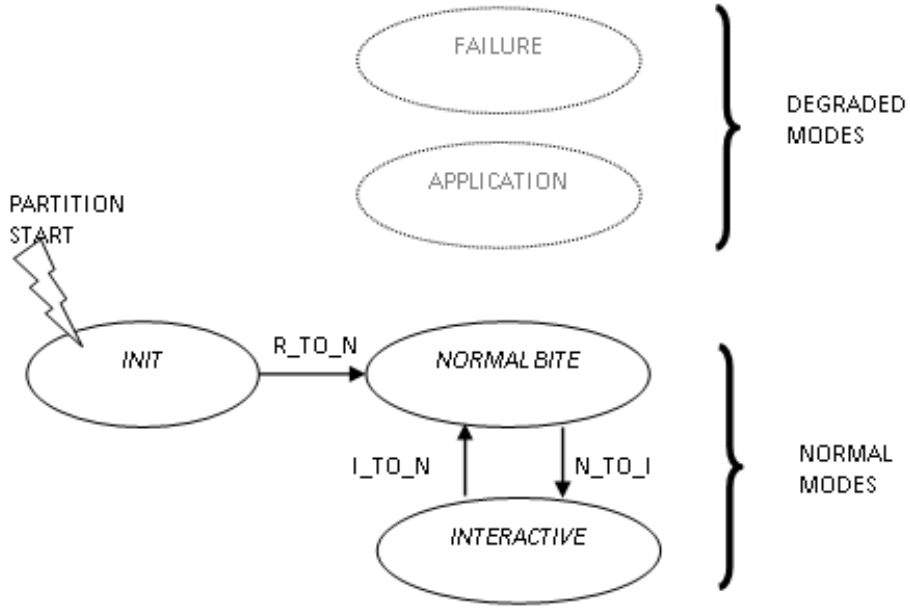

Figure 5.11: FCDC functional modes.

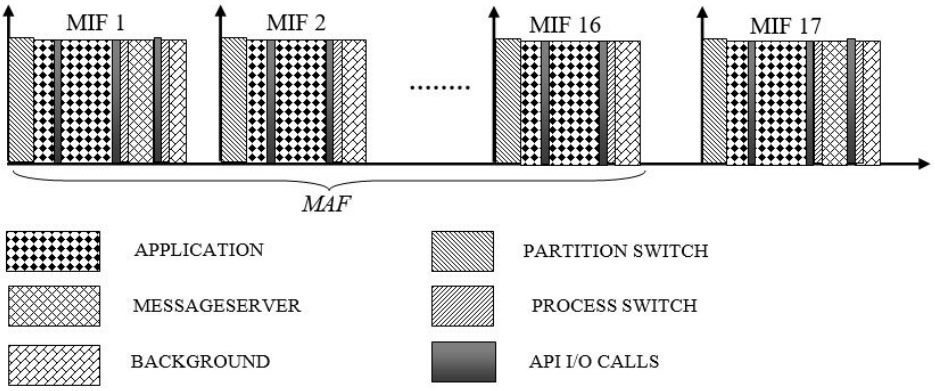

Figure 5.12: FCDC Dynamic Behavior.

MIFs	Cycle C1	Cycle C2	Cycle C3	Cycle C4	Cycle C5
1	C1	C2_1	C3_1	C4_1	C5_1
2	C1	C2_2	C3_2	C4_2	C5_2
3	C1	C2_1	C3_3	C4_3	C5_3
4	C1	C2_2	C3_4	C4_4	C5_4
5	C1	C2_1	C3_1	C4_5	C5_5
6	C1	C2_2	C3_2	C4_6	C5_6
7	C1	C2_1	C3_3	C4_7	C5_7
8	C1	C2_2	C3_4	C4_8	C5_8
9	C1	C2_1	C3_1	C4_1	C5_9
10	C1	C2_2	C3_2	C4_2	C5_10
11	C1	C2_1	C3_3	C4_3	C5_11
12	C1	C2_2	C3_4	C4_4	C5_12
13	C1	C2_1	C3_1	C4_5	C5_13
14	C1	C2_2	C3_2	C4_6	C5_14
15	C1	C2_1	C3_3	C4_7	C5_15
16	C1	C2_2	C3_4	C4_8	C5_16

Figure 5.13: FCDC SCADE Nodes Cycles Scheduling.

FCDC

FCDC software has five functional modes, as shown in Figure 5.11. Each functional mode consists in a specific subset of FCDC functions to be fulfilled. Only one mode can be active at a time:

- *INIT*. Call services to create A653 resources and start the partition processes.
- *NORMAL_BITE*. Normal execution, i.e. the execution time without any internal error conditions, of functional logic .
- *APPLICATION_BITE*. Degraded mode of FCDC functions.
- *FAILURE*. Failure mode following internal faults detection within FCDC software.
- *INTERACTIVE_BITE*. Mode to handle interactive BITE mode, ground only.

Degraded modes of FCDC software, i.e. FAILURE and APPLICATION BITE, are uninteresting from the execution time point of view and therefore out of the scope for applying our methods. In these two modes a great part of software functions are dropped and only a sequential and predictable code is kept which from the execution time point of view doesn't manifest any variability. FCDC case study consists in running processes of NORMAL_BITE mode. In this mode, FCDC executs the following A653 processes:

- APPLICATION process, sequences I/Os acquisitions and emissions and execution of automated generated SCADE logic cycles code.
- MESSAGESERVER process, manages failure messages reporting to the Centralized Maintenance System (CMS).
- BACKGROUND process, manages dialog sessions with Centralized Maintenance System.

Figure 5.12 illustrates how those processes are scheduled within a Major Frame composed of 16 Minor Frames periods. Note that MIF 17 corresponds to the MIF 1 of the next MAF. APPLICATION is a periodic A653 process, with priority set to 10 (highest priority), and its associated deadline timer set to one time window duration. MESSAGESERV is a 16 MIFs periodic A653 process with priority set to 5. It consists in a simple process, managing failure reports to the CMS. BACKGROUND is an aperiodic A653 process, with priority set to 1 (lowest priority), and no deadline. It is in charge of interactive menu management as a background task.

The SCADE nodes scheduled during execution of the APPL component of the APPLICATION A653 process are organized in 5 major cycles, from C1 to C5 with periodicity constraint set to $C_n = 2^{n-1}$ MIFs, subdivided in minor cycles. Such a periodicity constraint defines the 16 MIFs required by the FCDC application: one each MIF, one every two MIFs, etc.. The complete major and minor cycles allocation is described in Figure 5.13. In other words, during each MIF a pre-established sequence of SCADE code is executed. For example, during MIF13, the code of C1,C2_1,C3_1, C4_5 and C5_13 is executed. The 16 MIFs are executed in a round robin manner implemented by a switch-case code. Our main goal is to determine a pWCET curve for each MIF starting from each one's set of measurements. In the case of FCDC the MIFs are relatively equal in charge.

The complete FCDC software is composed of 1.5 million lines of code without comments, 30,000 lines being produced manually and 95% of this code belonging to the APPLICATION library.

For each application, an input meant to produce the worst-case scenario is used. A team of applications specialists inside Airbus was the responsible of producing this input. The use of an unique input vector allows us to concentrate on the time variability produced by the architecture.

The analysis of A653 applications focuses on the analysis of the main applicative periodic A653 processes associated to a hard deadline.

5.2.2 Platform characteristics

Our method can be used for any measured execution times, while the results and the confidence in them depend on the data used. We tested our methods on different types of data ranging from artificially generated values (to verify certain properties) to traces of execution times from real application execution. In this thesis we present the results obtained by analyzing execution times coming from the presented applications (WBBC and FCDC) running on two different platforms: Gaisler's Leon3 Processor and Freescale's P4080 PowerPC. In this section we present the properties of these two platforms and the way they are used for our measurements.

The Leon3 Processor

We use the Leon3 Processor in two different ways: default behavior and randomized behavior. By default behavior we understand the configuration offered by the hardware manufacturer that has all performance enhancing techniques activated (cache memory, branch predictors, pipeline etc.). We call this configuration as the "normal" one. We also use this processor in a randomized configuration as described in the PROXIMA project. This randomized configuration is achieved by modifying the cache replacement protocol from LRU to random and adapting other features of the hardware to be in concordance with this change. We start by presenting the architecture in normal mode following by an clarification upon the use of randomization.

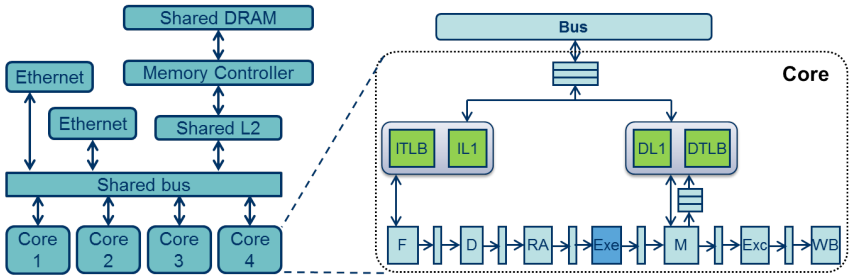

Figure 5.14: Schematic of the FPGA architecture.

The Leon3 processor is a field-programmable gate array (FPGA), which means that it can be configured by the user. We use this possibility to modify the architecture in the randomization section. The basic architecture behaves as shown in Figure 5.14. The processor features a 4-core multicore with a shared bus, a shared (unified for data and instructions) UL2 cache, a memory controller and DRAM memory. First level instruction (IL1) and data (DL1) caches are private per core. A number of Ethernet controllers are also in place.

The core architecture consists of a pipeline with the following stages: fetch (F), decode (D), register access (RA), execution of non-memory operations (Exe), DL1 access (M), Exceptions (Exc) and write back (WB). The operations occurring in each stage are as follows:

- Fetch stage: The IL1 is accessed (and the ITLB on a IL1 miss) to obtain the next instruction to be executed. Branches are predicted to be always taken.
- Decode stage: Instructions are decoded. This stage is, in essence, an extra delay in the pipeline.
- Register access: Instructions read their input registers with fixed latency.
- Execute stage: Non-memory instructions are executed with a fixed latency that depends solely on the type of operation, except FDIV and FSQRT instructions, whose latency is input data dependent. Memory operations compute their addresses.
- Memory stage: Load instructions access the DL1 (and DTLB on a DL1 miss). Indeed, they also access the write buffer. Store operations are placed in the write buffer for their offline processing. If the write buffer is full the pipeline will be blocked.

- Exception stage: Exceptions are managed here.
- Write-back stage: Results (if any) are sent to the register file.

We describe the main components of this architecture that are able to produce variability in execution time measurements:

Instruction Cache (IL1) IL1 is set-associative and implements modulo placement and both, LRU and random replacement policies. IL1 latencies are as follows. On a hit the instruction will be served after IL1hit cycles. On a miss, one of the cache lines in its set will be selected for eviction, and the request will be forwarded to the shared bus after translating the address in the ITLB (its impact in latency is analyzed in next subsection). Miss latency will depend on a series of conditions, like bus arbitration latency, UL2 hit/miss outcome, memory controller latency in case of a UL2 miss, bus latency to send the cache line back to the core, etc... To minimize latency, instruction streaming is performed. Instructions are fetched from memory starting at the missed address until the end of the cache line. Control flow changes might suffer a penalty because of this, since the fetch of a streamed line needs to be completed before the control flow change can take effect.

The Instruction Cache implements cache locking on a per line basis. The lock bit can be set by performing a diagnostic write to the instruction tag on the cache offset of the line to be locked. Locking prevents the cache line to be replaced by the replacement algorithm, but the locked cache line will be updated on a read-miss and will remain in the cache until the line is unlocked.

Instruction Cache freezing can be enabled via the cache control register. In the frozen state, the cache is accessed and kept in sync with the main memory as if it was enabled, but no new lines are allocated on read misses.

Cache flushing is also implemented and takes one cycle per cache line, during which the IU will not be halted, but during which the caches are disabled. When the flush operation is completed, the cache will resume the state (disabled, enabled or frozen) indicated in the cache control register.

Instruction TLB (ITLB) On a IL1 miss or store hit (the cache is write-through), the ITLB is accessed to translate virtual into physical addresses (if the MMU is enabled). The ITLB is fully-associative and implements LRU and random replacement policies (configurable at implementation time).

On an ITLB hit, IL1 access proceeds normally to the bus. On an ITLB miss, the IL1 access cannot be served until the MMU completes address translation. The

processor will perform a hardware table walk that:

- looks up the MMU context's root pointer
- traverses a three-level page table for the selected context
- in the case of writes, the hardware table walk mechanism will set the page table entry's modified (M) bit.

Thus, an ITLB entry is evicted (according to placement/replacement policies) and the virtual to physical address mapping is entered into the TLB. Once the ITLB miss is served, the IL1 access can proceed.

Queues In the processor several request queues may exist to decouple different hardware blocks. For instance, fetched instructions may be placed in a queue so that they can be consumed by the decode stage. If any later stage stalls long enough, such queue may get full, thus introducing backpressure into the fetch stage, which may get stalled until at least one entry is released in the queue. Thus, whether a queue gets full or not depends solely on the behavior of the different stages, which in turn progress faster or slower depending on deterministic (and fixed latency) events such as the execution in some functional units or probabilistic events such as hit/miss in any cache-like structure (IL1, ITLB, DL1, etc.).

There are two queues known to be susceptible to the problems described above; the instruction FIFO for the floating-point controller and the write-buffer. The instruction FIFO in the floating-point controller accumulates floating-point instructions and keeps them in the FIFO until the floating-point controller pipeline (that acts in parallel with the integer unit pipeline) is ready to accept a new instruction. When the FIFO is full then the integer unit pipeline will need to stall until a slot in the floating-point controller FIFO becomes available.

The write-buffer can hold two words of write data and it is used so that a memory write operation can complete immediately without waiting for the delay required for the write to complete on the bus. When the write-buffer is full, any store and any load instruction will be stalled until the write-buffer has slots available.

Data Cache (DL1) DL1 behaves similarly to IL1 for load accesses with the exception that those read accesses check in parallel both the DL1 and the write buffer. Note that the write buffer contains data from older stores that have not been sent to the DL1 yet. Additionally the data cache is subject to invalidation for

data snooped from the system bus, and it is also subject to an invalidation-on-write policy.

Store accesses are processed in order from the write buffer when no load access is to be served.

Uncacheable requests are sent straightforward to memory affecting neither IL1 nor ITLB contents. Thus, they pay some arbitration delay in the bus.

Cache locking, freezing and flushing considerations that were made for the IL1 are also valid for the Data Cache. Instruction and Data caches can be locked and flushed separately.

Data TLB (DTLB) Its behavior is completely analogous to that of the ITLB.

Write Buffer The core is equipped with a write buffer that prevents from stalling the pipeline on each store operation. This is particularly important because stored data are not typically consumed by subsequent instructions and thus, it can be stored off-line without stalling other activities as long as data consistency is kept (loads check the write buffer and stores occur in order).

On a store operation, its data are written into the next free entry of the write buffer, which is implemented typically as a circular queue. If the write buffer is full, then the corresponding pipeline stage is stalled (M), thus creating some backpressure.

The write buffer does not support write combining, therefore every entry of the buffer is sent to the shared bus as a separate write request.

Core-to-bus Queue Requests sent from the core are queued in order in the core-to-bus queue. If two requests occur simultaneously in the IL1, DL1, and MMU subsystems (exactly in the same cycle), which occurs very rarely, preference must be given to one type of requests. Higher priority is given to IL1 requests, which therefore get queued first.

Bus The bus implements round-robin arbitration among pending requests from the cores and the Ethernet interfaces. Once a core is granted access to the bus, it will keep it busy until its request is fully served or until it receives a SPLIT response. No further bus request will be accepted when the bus is busy. If the request is a memory access and SPLIT responses are not supported, no further requests will be accepted, even if they are UL2 accesses that could hit and be served right away. If SPLIT responses are supported, memory requests will not keep the bus busy, and other competing cores can issue and be served data that hits in UL2, for instance.

On a bus transmission, the sender sends 1 or multiple beats. Each beat is a 32-bit word that can be sent at once. If multiple beats are sent (2, 4, 8 or 16), they are sent in a burst. AMBA specification supports 8-beat, 16-beat, and undefined length bursts. The processor will be configured to not exceed a maximum burst size. The particular number (4 or 8 for cache line fills, 2 for load double) is to be defined, but it will be limited.

It is also the case that Ethernet transactions may need special treatment depending on their timing characteristics. This will be known at a later stage when applications are ported onto this platform. If special treatment is needed, it will most likely be managed by software means. However, any decision on this will be taken after month 18.

Shared L2 Cache (UL2) The shared UL2 cache can be partitioned on a per-way basis, thus allocating different cache ways to each core. The UL2 cache implements the same placement and replacement policies as the IL1 and DL1 caches. It serves only one request at a time. On a miss it requests data to the memory controller and waits for the answer to respond to the corresponding core through the bus. While serving a request, the UL2 cache keeps the bus busy by introducing "wait states" (a feature of the particular AMBA bus used).

When split transactions are enabled, "wait states" will not be issued and instead the UL2 cache will answer the core with a SPLIT response, freeing the bus until the data fetch from memory has completed. UL2 will process several requests simultaneously: the one coming from the bus and those being served in memory. The number of requests per core is automatically limited since a core cannot issue a new access on the bus until the current access has completed.

The UL2 cache features both, write-through and write-back (also known as copy-back) policies. Under write-back policy write operations update UL2 contents but are not forwarded to memory until a dirty cache line is evicted. Under write-through policy write operations are forwarded to memory regardless of whether the data are in UL2.

Shared Memory Controller The memory controller only processes one request at a time. The time to process such request depends on the type of request (read or write) and the type of the previous request (read or write). Note that currently the memory controller can only process one request at a time. Whenever split transactions are implemented in the bus, it will be possible to receive up to

one request from each core (any core stalls until its pending request is serviced).

All the presented features have an impact on the period taken by a program to execute. This impact can be restricted for some features by deactivating them (as it is the case of cache architectures) or the impact can be calculated if a fully understanding of the system is possible. Nevertheless, deactivating some of the component of the architecture implies a loos in performance.

Randomized architectures

Since it is an FPGA architecture, the Leon3 can be modified and adapted according to the user's needs. Inside the PROXIMA project, the idea of fully randomizing the hardware is proposed as a environment for the use of probabilistic analysis. We specify that **our methods do not require randomized hardware in order to be used**. Furthermore, randomization can violate the condition of reproducibility of the measurement protocol. On the other hand, statistical analyses are the only to study execution time behavior for randomized architectures.

This possibility of using a randomized architecture an a critical real-time system is unlikely due to the performance loss it brings. Nevertheless, my involvement in the PROXIMA project from both Inria and Airbus parts allowed me to perform probabilistic analysis on measurements obtained from randomized hardware. Therefore, in this thesis we will present results obtained in this context.

The randomized hardware used by us is obtained by using a random replacement protocol on the IL1, DL1, ITLB and DTLB. This is translated by the fact that once a cache miss is encounter by one of these four components a cache line or an ITLB entry will be randomly evicted. This approach introduces a certain level of randomization in a program's execution (depending on the program). A higher level of randomization might be achieved on the Leon3 process by performing changes at the bus level or at the UL2 level. These manipulations are out of the scope of this thesis.

The motivation promoted for using randomized hardware in the real-time domain is that cache configurations that have low chances of being observed using the normal behavior would have a higher chance to be observed with such an architecture. This increases the chances of observing extreme execution times but there is no guarantee upon observing the WCET. For the measurement approach timing analysis observing higher execution times can be useful, but by performing proba-

bilistic timing analysis using EVT, a high number of extreme values can artificially induce pessimism.

The 4080 PowerPC

The Freescale's QorIQ P4080 is a potential candidate of the next generation processor for future avionic applications, because Freescale is a long-term provider in the avionic domain and it collaborates with avionics manufacturers to facilitate their certification of systems using multicore processors. In this context, Airbus keeps a long term relationship with Freescale about PowerPC based systems. Moreover, predecessors of the QorIQ line have been used in multiple aircraft applications. Therefore, we selected the P4080 as the target platform to evaluate the experiments, and we present the P4080 structure and features in this section.

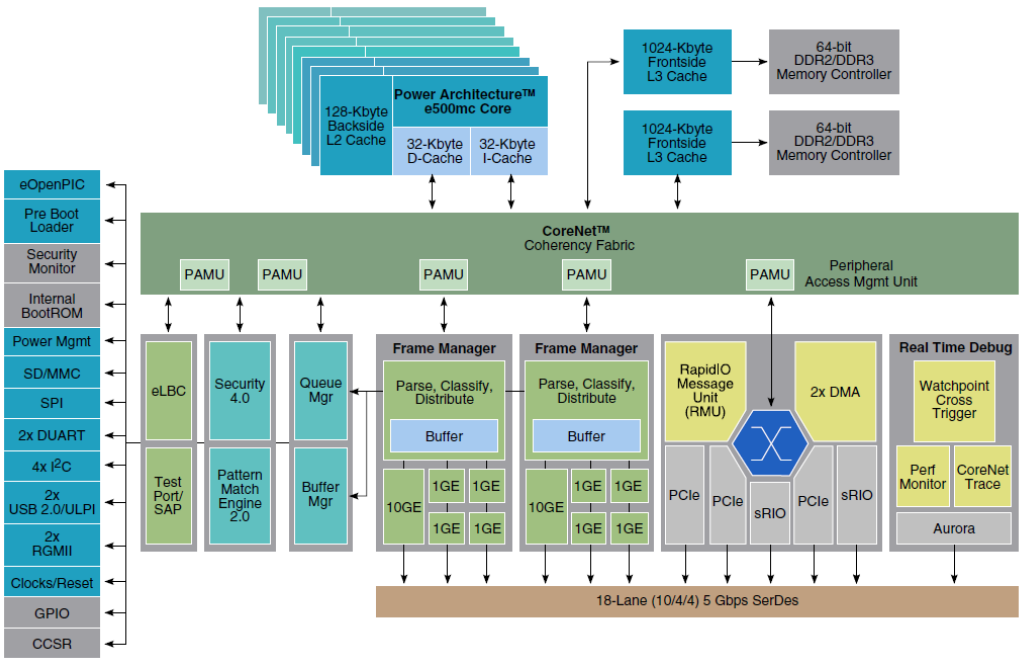

Figure 5.15: P4080 Block Diagram.

The P4080 Development System is the 8-core Freescale's QorIQ platform. It belongs to the P4 series which is a high performance networking platform, designed for backbone networking and enterprise level switching and routing. As shown in Figure 5.15, the system is composed of eight e500mc PowerPC cores coupled with

private L1 data+instruction caches and a private L2 unified cache. A shared memory architecture including two banks of L3 cache and two associated DDR controllers is built around the CoreNet fabric. Additionally, several different peripherals are implemented around the CoreNet fabric. The CoreNet fabric is a key design component of the QorIQ P4 platform. It manages full coherency of the caches and provides scalable on-chip, point-to-point connectivity supporting concurrent traffic to and from multiple resources connected to the fabric, eliminating single-point bottlenecks for non-competing resources. This eliminates bus contention and latency issues associated with scaling shared bus/shared memory architectures that are common in other multi-core approaches.

Execution timing of e500mc cores. The e500mc core is a pipelined, superscalar processor. The core has six execution units: one each for branch, one for load/store, one for floating-point operations, one for complex integer operations, and two for simple arithmetic operations.

The common pipeline stages are as follows:

- Instruction fetch: Includes the clock cycles required to request an instruction and the time the memory system takes to respond to the request. Fetched instructions are latched into the instruction queue (IQ) for consideration by the dispatcher.

- Decode/dispatch stage: This stage fully decodes each instruction; most instructions are dispatched to the issue stage. In principle, the latency of this stage is fixed for each instruction type.

- Issue stage: This stage reads source operands from rename registers and register files and determines when instructions are latched into reservation stations feeding execution units. The latency of this stage is fixed except for the selection of the next instructions to be issued for execution, as out-of-order execution may delay the execution of an instruction an arbitrary number of cycles letting younger instructions to proceed.

- Execute stage: It is comprised of individual non-blocking execution units that operate in parallel. Each execution unit has a reservation station that must be available for an instruction issue to occur. In most cases, instructions are issued both to the reservation station and to the execution unit simultaneously. Most instructions have a fixed execution latency, but there are some notable

exceptions as load/store instructions and some arithmetic operations. Further details are provided later.

- Complete and write-back stages: They maintain the correct architectural machine state and commit results to the architecture-defined registers in the proper order. If completion logic detects an instruction containing an exception status or a mispredicted branch, all following instructions are canceled, their execution results in rename registers are discarded, and the correct instruction stream is fetched. This stage introduces execution time variation due to misspeculations and due to stalls whenever the oldest instruction is not complete and no further instructions can be decoded/dispatched (backpressure).

This core allows fetching four instructions per cycle, decoding/dispatching two instructions per cycle, issuing two instructions per cycle (as long as they do not require the same reservation station), and completing two instructions per cycle in order.

Intra-Core Cache Memories. Each core has private first level data (DL1) and instruction (IL1) caches memories, as well as a private second level (L2) unified (for instructions and data) cache with these features:

- DL1 and IL1:
  - 32KB, 8-way set-associative, with pseudo-LRU replacement policy.
  - Cache locking is enabled. Locking can be performed on a per cache line basis. Locking is performed by a set of touch and lock set instructions, so it is a feature that, if used, is triggered by the software.
- DL1 only:
  - Write-through policy, non-blocking stores. Blocking only occurs when any pipeline stalls puts some backpressure that cannot be mitigated by the load/store buffer.
  - The L1 data cache supports a MESI (Modified/Exclusive/Shared/Invalid) cache coherence protocol per cache line.
- L2:

- 128KB 8-way with pseudo-LRU replacement policy.
- The L2 cache supports a MESI cache coherence protocol per cache line.
- Write-Back policy.
- Arbitrary way partitioning across instructions and data is allowed. For example, software could configure L2 to reserve 3 ways for instructions and 5 ways for data.
- Supports direct stashing of datapath architecture data into L2 to make it operate as a scratchpad, which may be useful for **R3.2-1.9** and **R2.2-1.7**.

Those caches can be emptied by means of software instructions and implement modulo placement.

Intra-Core TLBs. Analogously to the fact that each core has its own L1 and L2 cache memories, they also have private data (DTLB), instruction (ITLB) and second level translation lookaside buffers (L2TLB) that serve as Memory Management Unit (MMU).

The DTLB and ITLB are each composed of two subarrays: an 8-entry fully-associative array for variable-sized pages, and a 64-entry 4-way set-associative array for fixed sized pages that provide virtual to physical memory address translation for variable-sized pages and demand-paged fixed pages respectively. These arrays are maintained entirely by the hardware with LRU replacement.

The L2TLB contains a 64-entry, fully-associative unified (instruction and data) array that provides support for variable-sized pages. It also contains a 512-entry, 4-way set-associative unified TLB for 4KB page size support. These second-level L2TLB is maintained completely by the software.

Shared L3 Cache Memory. The P4080 has two L3 caches shared across all cores. Each L3 cache is connected to one of the two memory controllers in place. The main characteristics of the L3 caches are as follows:

- 1MB, 32-way set-associative, 64 byte/line each L3 cache, implementing pseudo-LRU replacement.
- Each L3 cache is connected to an independent memory controller as shown in Figure 5.15.
- L3 caches can be configured as write-back or write-through.

- They operate at 800MHz and can serve two full cache line reads per cycle.

Each cache way of each L3 cache can be configured in one of the following three modes: (i) disabled, (ii) as a regular cache, and (iii) as a scratchpad. If disabled, it has no influence in the timing of the programs being run. Otherwise, L3 caches may affect the timing behaviour. Note that in the same caches some cache ways can be configured as regular caches and others as scratchpads simultaneously.

Interconnection Network. The P4080 implements the CoreNet coherency fabric (CCF) as its interconnection network to communicate cores, platform-level caches, memory subsystems, peripheral devices, and I/O host bridges in the system. The CCF enables the implementation of coherent, multicore systems.

The CCF includes the following distinctive features:

- Multiple in-flight transactions with:
  - Concurrency of transactional progress through the system.
  - Out of order completion.
- Sustainable bandwidth: four transactions/cycle.
- Low latency data path for platform cache data.
- Sustainable read bandwidth: 128 bytes per cycle.
- Power Architecture coherency semantics.
- Accelerated operation for non-coherent accesses.
- Transaction ordering support.
- Address map support.
  - 32 local access windows (LAWs).
  - DDR memory and SRAM interleaving support.
- 64 byte coherency granules.
- Logical partitioning support.
  - Address-based, secure isolation of partitions and their resources.
  - Coherency subdomain assignment and snoop limiting per LAW.

  - Inter-partition sharing of address ranges.

- Support for stashing.

Details about the timing of the CCF are scarce. It is unclear what happens when the number of transactions arriving simultaneously exceeds the bandwidth of the CCF. However, based on the characteristics above, it is clear that its timing behavior can easily resemble that of a full crossbar given the high bandwidth. Furthermore, it may be the case that bandwidth can be exceeded only sporadically during very short time periods due to the high bandwidth and the limited amount of traffic that the different components connected can generate. This is particularly true because any request traversing the CCF may likely take longer to get an answer due to the high latency of the components attached. Thus, the amount of interference that could eventually occur in this component is limited and its impact in execution time is largely below that of the latency of those components processing those requests.

As for L3 caches, partitions must be defined by the software layers.

PAMU and MMU. MMU is the hardware mechanism that controls all the address-based accesses initiated by the cores. PAMU is a similar mechanism that controls all address-based accesses initiated by the DMA-capable peripherals.

On a core-initiated transaction, the program's Execution Address is extended with context IDs to make the virtual address. Through the MMU, the virtual address is verified and translated into the real address, which is the 36-bit local physical address. In case of a cache miss or non-cacheable access, the real address is directed to CoreNet and searched through the Local Access Window (LAW) registers.

The local address map is defined by a set of 32 LAWs. Each of these windows maps a programmable region of the local address space to a specified target interface, which is either a local memory interface (DDR, eLBC, internal SRAM) or a system interface (PCIe, SRIO). This allows the internal interconnections of the device to route a transaction from its source to the proper target. The LAW registers perform no address translation.

In terms of timing the PAMU itself has little effect. What really can affect timing are those devices attached to the PAMU. Most of them may get blocked on an access. However, the PAMU provides means to support partitions so that activity from different partitions can be segregated and prioritized.

Memory Controller. There are two Memory DDR SDRAM controllers which

control processor and I/O interactions with system memory. The controller allows as many as 32 pages to be opened simultaneously. The amount of time (in clock cycles) the pages remain open is programmable with specific registers. Partitioning can be implemented only to some extent according to the documentation. In principle it is only doable segregating activity across different memory controllers. For instance, one may divide the system into several partitions and allow only one of them to use a particular memory controller, thus removing interferences from other partitions.

As in the other components, partitions must be defined by the software layers.

Even though it is more performant than the Leon3, the P4080 platform inflicts a higher variability in he execution time structure. This is a disadvantage for static analysis but is helpful for our method because it allows us to better estimate the behavior f the system under analysis. We are using this platform without disabling any of the features described previous. This allows us to fully benefit from the performance of the hardware and to get as close as possible to the configuration in which this hardware would be used by the avionics industry.

5.2.3 Timing analysis results

The applications selected for the case study rely on the IMA concept. For these specific applications, the execution is separated in 16 isolated branches, which generate as many Unit of Analysis (UoA). These 16 branches (UoA1, UoA2, $\cdots$,UoA16) contain sequential code automatically generated by the SCADE environment. The application is executed 1000 times in order to produce the equivalent number of execution times per UoA.

For each application, an input meant to produce the worst-case scenario is used. A team of applications specialists inside Airbus was the responsible of producing this input. The use of an unique input vector allows us to concentrate on the time variability produced by the architecture.

Platform / App.	Leon3-N	Leobn3-R	P4080
WBBC	✓	✓	✓
FCDC	x	✓	✓

Table 5.2: Summary of results for avionics case study.

The obtained results are split in three categories according to the used hardware architecture: Leon3 Normal (Leon3-N), Leon3 (Leon3-R) Randomized and P4080.

The experiments done in the avionics case study are summarized in the Table 5.2. We mention that the applications are executed in isolation on one core of the hardware and the measurements are done in processor cycles. We consider that the methods proposed by us can apply to execution times obtained for application running in a multi-core context, under the condition that the contenders used for testing are the exact ones used at run-time. Any change in the system composition, including the applications executing on the other cores, will invalidate existing analysis. Due to low variability, we were not able to successfully compute a pWCET estimation for the execution times collected for the FCDC application on Leon3 in normal mode.

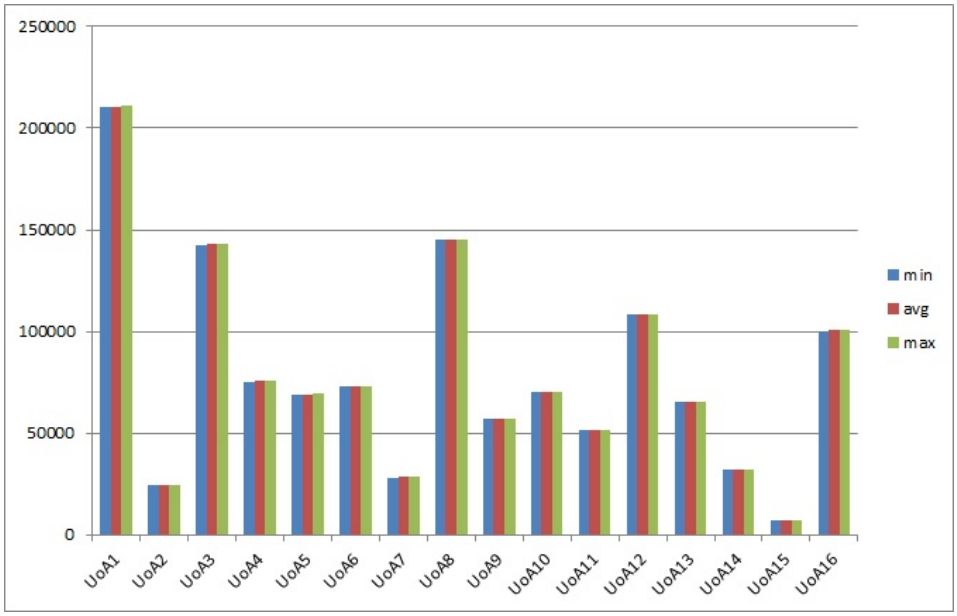

Figure 5.16: Execution time for WBBC on Leon3 in normal mode.

WBBC on Leon3-N

Figure 5.16 shows the statistics of execution times for WBBC on the FPGA platform. We observe small variations coming from an unknown or non-handled sources of jitter. The standard deviation that describes this variations is presented in Figure 5.17. Low variability makes fitting on an EVT distribution harder to achieve, but not impossible. UoA1 represents the path of WBBC application that has the

highest chance of producing the WCET. Due to lack of space, we mostly present in thesis results and graphics from the analysis of this code. The analysis results for the other 15 UoAs is summarized and presented in the form of graphics and tables.

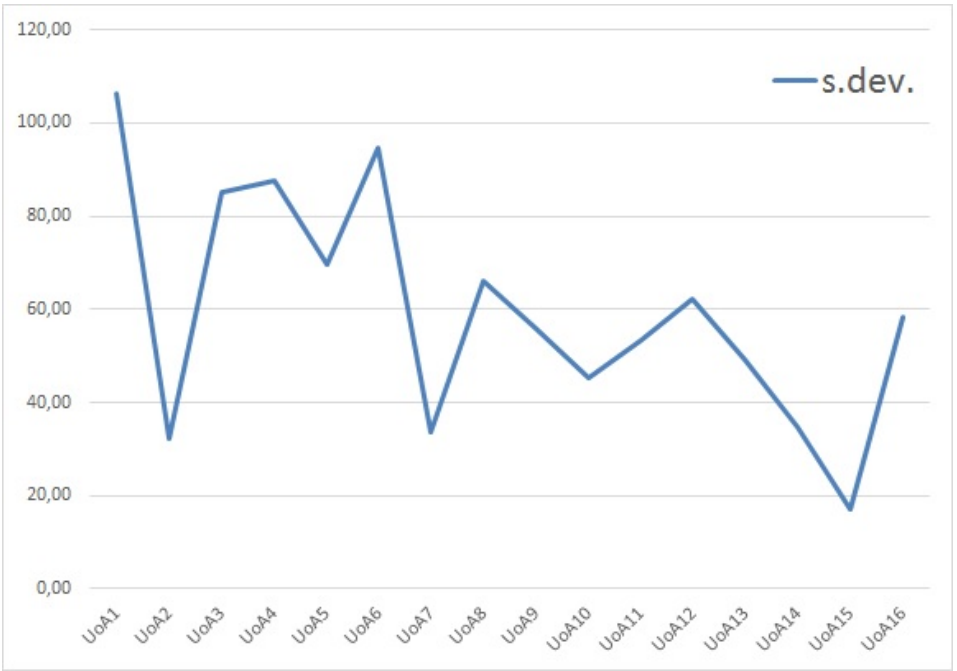

Figure 5.17: Standard deviation of the execution time obtained for the WBBC application on Leon3 in normal mode.

Table 5.3 shows the events monitored by the performance counters (PMCs) on the Leon3 for each UoA. Since the system is executed in COTS mode under similar conditions (i.e., inputs, executable), as expected, the PMCs do not change from one run to another. The PMCs measured are: instruction cache miss (icmiss), data cache miss (dcmiss), number of stores and the amount of FPU operations. The results show that the variability in execution times for this scenario must be from other unidentified reasons (possibly memory refreshes).

	icmiss	dcmiss	store	fpu
UoA1	683	729	7466	0
UoA2	168	61	576	19
UoA3	749	211	3443	248
UoA4	402	134	1968	93
UoA5	334	124	2300	32
UoA6	412	139	1432	55
UoA7	186	61	791	0
UoA8	886	182	2039	237
UoA9	354	94	1020	74
UoA10	427	106	1049	98
UoA11	304	102	1218	10
UoA12	588	210	1956	308
UoA13	359	147	1311	208
UoA14	192	66	935	0
UoA15	90	27	276	0
UoA16	400	194	3872	0

Table 5.3: PMC readings for WBBC on Leon3-N.

From statistical point of view, we examined each trace in order to verify if identical distributed and independence test pass. The results are presented in Table 5.4. We notice that a majority of the cycles are passing both tests despite low variability. Since all the execution times per UoA are obtained in the same manner, it is expected that the identically distributed test passes. For some of the code the independence test does not pass. This result is obtained due to low variability. In this case the measurements is composed of very few values out of which one or two are predominant, making the test of identically distributed to perceive this data as a unique value and therefore dependent. For these cases, we use the procedure presented in section 4.4 to compute the pWCET curve.

	i.d.	ind.
UoA1	✓	✓
UoA2	✓	✓
UoA3	✓	✓
UoA4	✓	✓
UoA5	✓	✓
UoA6	✓	✓
UoA7	✓	✓
UoA8	✓	✓
UoA9	✓	✓
UoA10	✓	✓
UoA11	✓	x
UoA12	✓	✓
UoA13	✓	x
UoA14	✓	✓
UoA15	✓	x
UoA16	✓	✓

Table 5.4: Identically distributed (i.d.) and independence tests results.

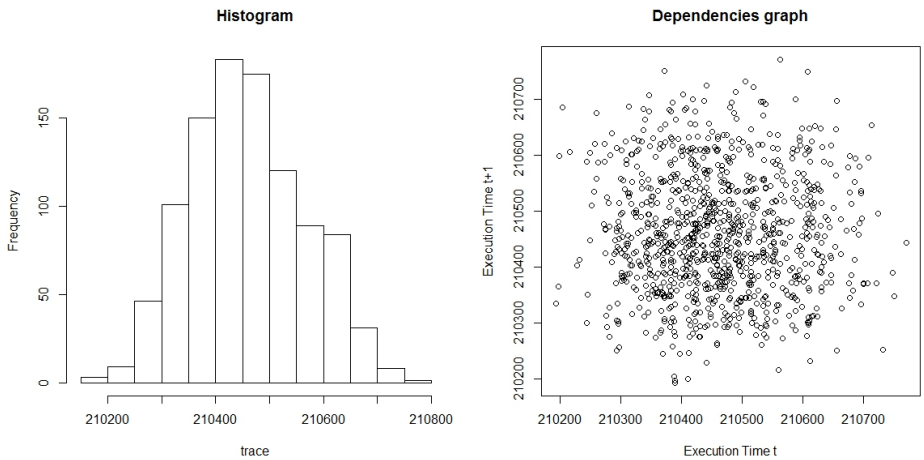

Figure 5.18: Graphical description of the UoA1 execution times obtained on the Leon3-N platform. On the left: histogram of the data. On the right: lag test result.

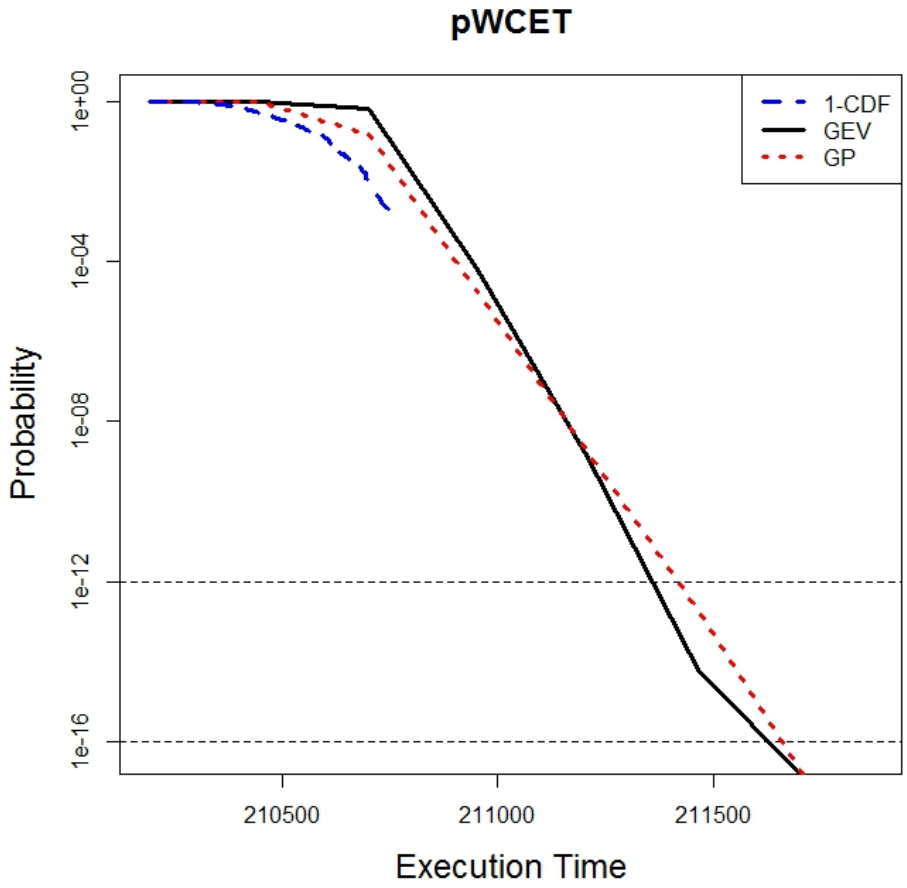

Figure 5.19: pWCET estimation for UoA1 of WBBC application executing on the Leon3 processor in normal mode.

A graphical description of the UoA1 containing the histogram and the Lag test can be seen in Figure 5.18. The pWCET estimation using our method can be seen in Figure 5.19. We depicted both GEV and GP estimations to exemplify the fact that they produce the same pWCET curve. The estimations at probability lower than 10^{-2} are the same. A summary of the estimation for all 16 UoA can be seen in Table 5.5. We notice a safety margin between 0.1% and 0.15% compared to the maximum observed execution time (MOET).

	MOET	P10e-12	eval
UoA1	210770	210910	0,07%
UoA2	24672	24689	0,07%
UoA3	143144	143870	0,51%
UoA4	75883	76293	0,54%
UoA5	69306	69365	0,09%
UoA6	73353	73392	0,05%
UoA7	28427	28449	0,08%
UoA8	145427	145500	0,05%
UoA9	57202	57234	0,06%
UoA10	70517	70624	0,15%
UoA11	51652	51668	0,03%
UoA12	108698	108811	0,10%
UoA13	65606	65613	0,01%
UoA14	32105	32299	0,60%
UoA15	7348	7355	0,09%
UoA16	100626	100706	0,08%

Table 5.5: pWCET estimations for WBBC execution on Leon3 in normal mode.

WBBC on Leon3-R

For this section, the experiments were done by running the WBBC application in isolation on one of the cores of the randomized Leon3 processor. The randomization technique is proposed in the PROXIMA project [Quinones et al., 2009]. We present these results in order to show the impact of such a platform on the execution time behavior.

The execution times statistics are shown in Figure 5.20. We notice an increase in both average and maximum values of the observations. This can be better seen in Figure 5.21 where a slowdown factor of execution times obtained on the randomized platform are presented. We notice that the maximum execution times tend to increase by about 8-12% (and a bit less for the average values).

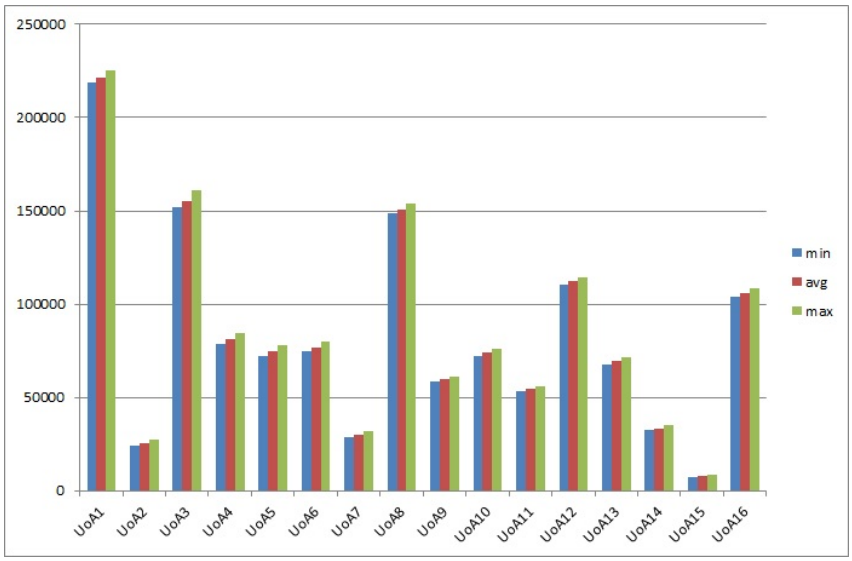

Figure 5.20: Execution time for WBBC on Leon3 in randomized mode.

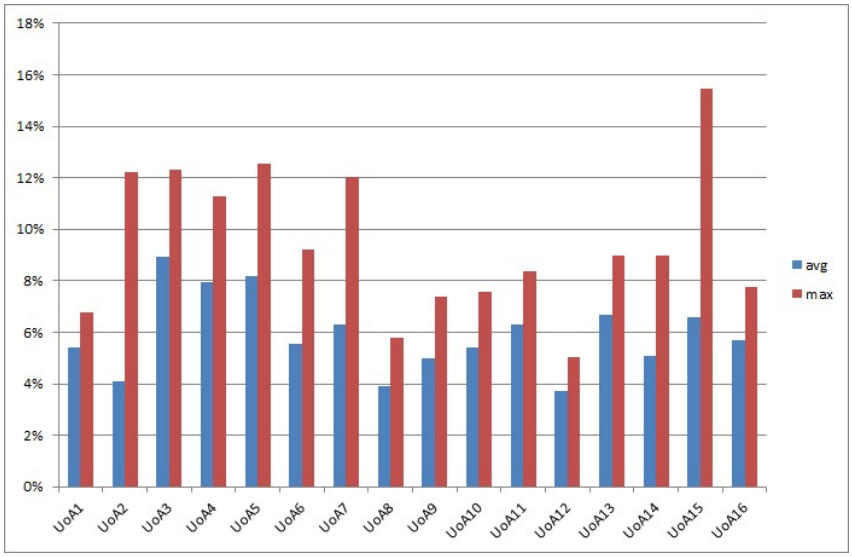

Figure 5.21: Comparing avg and max for WBBC, LEON3 in normal mode and in randomized mode.

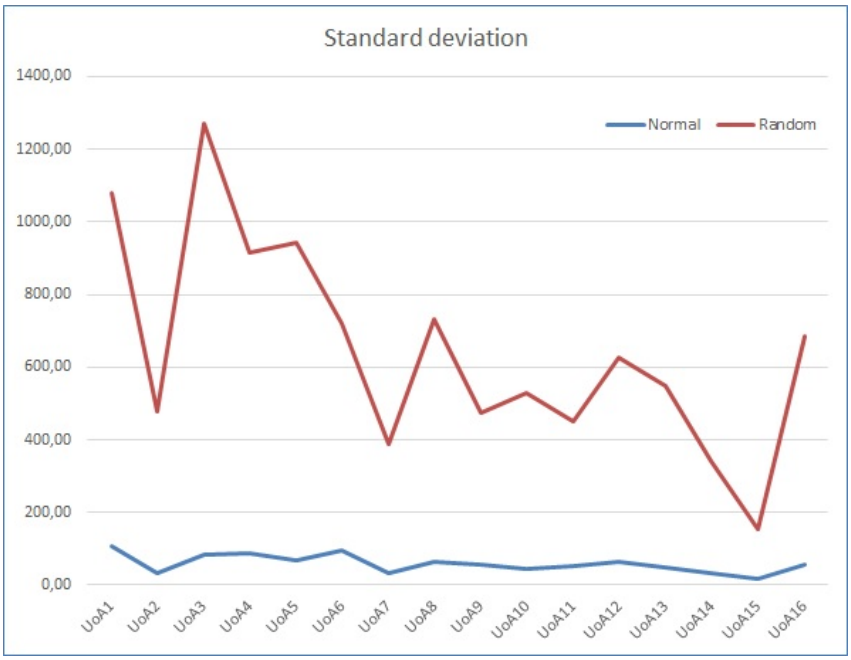

Figure 5.22: Standard deviation of the execution time obtained for the WBBC application on Leon3 in randomized mode (compared with normal mode).

The effect of randomization upon the variability of obtained execution times can be seen in Figure 5.22. Compared with the execution times obtained on Leon3 in normal mode, the standard deviation for the randomized mode is highly superior.

The performance monitoring counters reveal an increase in the number of misses on both instruction and data caches. The complete measurements can be found in Table 5.6.

For this case study, all the tests are passing which means that EVT for independent data can be applied on each UoA. However, because of the randomization factor, the experiments cannot be reproduced identically, and the collected measurements differ from one experiment to another. In some cases, different measurements produce different statistical tests results.

In order to present an example of execution time distribution corresponding to these measurements and to their probabilistic estimation we will concentrate on UoA1 of the WBBC application. This UoA was chosen because it contains the highest execution time observed for the application.

	icmiss	dcmiss	store	fpu
UoA1	687-712	736-764	7466	0
UoA2	165-182	61-65	576	19
UoA3	797-850	211-226	3443	248
UoA4	418-453	134-144	1968	93
UoA5	342-387	124-132	2300	32
UoA6	418-443	139-147	1432	55
UoA7	188-200	61-65	791	0
UoA8	893-923	182-199	2039	237
UoA9	360-377	94-103	1020	74
UoA10	432-453	106-115	1049	98
UoA11	303-320	102-111	1218	10
UoA12	591-612	210-225	1956	308
UoA13	365-386	147-155	1311	208
UoA14	188-200	66-70	935	0
UoA15	87-95	27-28	276	0
UoA16	405-430	194-203	3872	0

Table 5.6: PMC readings for WBBC on Leon3-R.

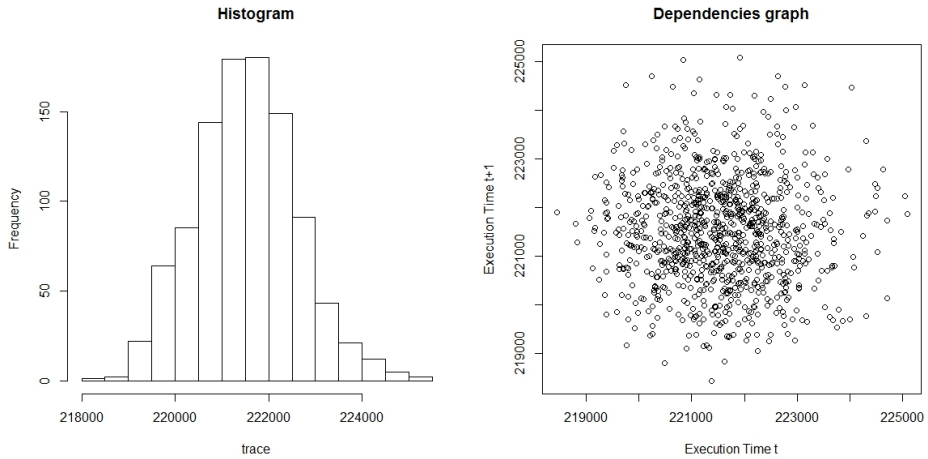

Figure 5.23: Graphical description of the UoA1 execution times obtained on the Leon3-R platform. On the left: histogram of the data. On the right: lag test result.

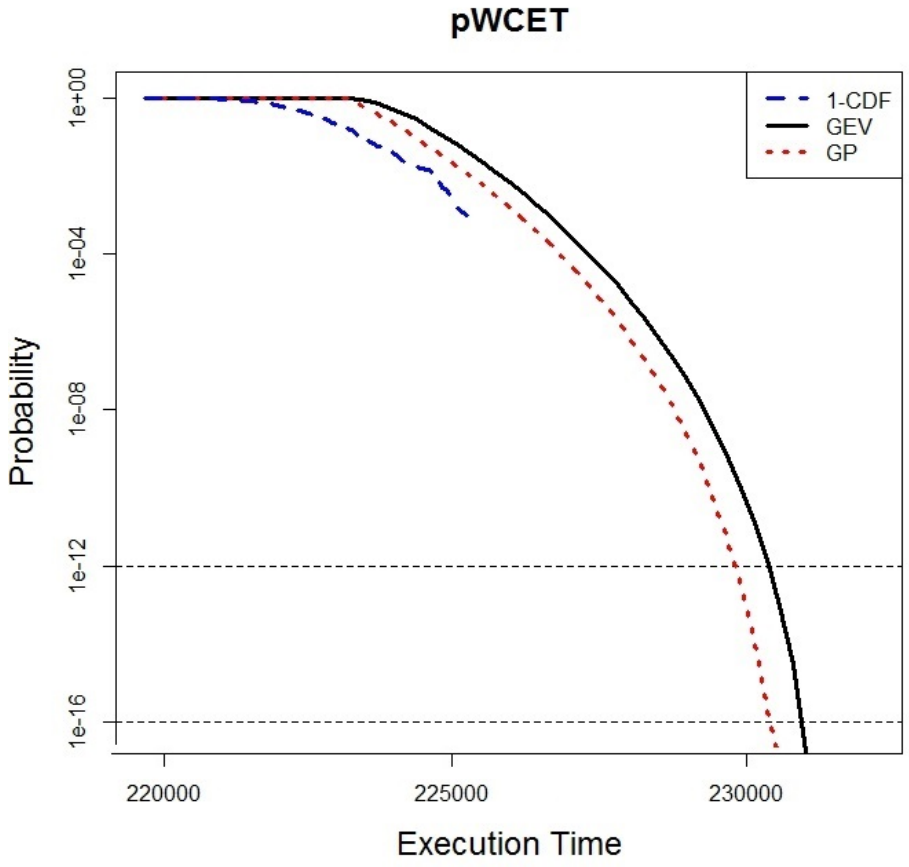

Figure 5.24: pWCET estimation for UoA1 of WBBC application executing on the Leon3 processor in randomized mode.

Figure 5.23 shows the histogram of execution times obtained for the WBBC application on the hardware randomized architecture and the lag test result (no particular pattern is observed, in concordance with the independence tests).

Figure 5.24 shows the pWCET curve for UoA1 of WBBC, the blue line corresponds to the empirical distribution of observed execution times and the red and black distributions represent pWCET estimations for the given sample.

The full summary of the probabilistic analysis can be found in Table 5.7 and Figure 5.25. They present the values observed for threshold probabilities in report with the maximum observed execution time. In line with the pWCET graphically illustrated in Figure 5.24, these results show that the pWCET prediction tightly upper-bound the actual observations.

	MOET	P10e-3	P10e-6	P10e-9	P10e-12
UoA1	225084	225920	227305	228690	229798
UoA2	27692	28559	30970	33381	35828
UoA3	160798	161545	165794	170043	174494
UoA4	84445	85639	87676	89606	91643
UoA5	78007	79256	82540	85824	89109
UoA6	80111	80297	82630	84962	87397
UoA7	31851	32261	33905	35508	37152
UoA8	153839	154392	157051	159520	162179
UoA9	61439	61865	63016	64244	65395
UoA10	75866	76547	77871	79196	80520
UoA11	55973	56290	56850	57340	57900
UoA12	114164	116225	119177	122130	125083
UoA13	71487	72245	73677	75199	76631
UoA14	34982	35200	36261	37323	38385
UoA15	8484	8559	9177	9795	10413
UoA16	108423	109380	110593	111806	113155

Table 5.7: pWCET estimations of WBBC executing on Leon3 in randomized mode.

WBBC on P4080

We present the results obtained for this platform independently. The comparison with the Leon3 processor is inappropriate due to the difference in performance between the two platforms. The P4080 processor performs the Leon3 under the same conditions of execution but has a higher variability producing a larger number of different observations.

Figure 5.26 shows the statistics of execution times for WBBC on the P4080. We observe larger variations between min-max durations than for the Leon3. This was expected given the greater complexity of the processor.

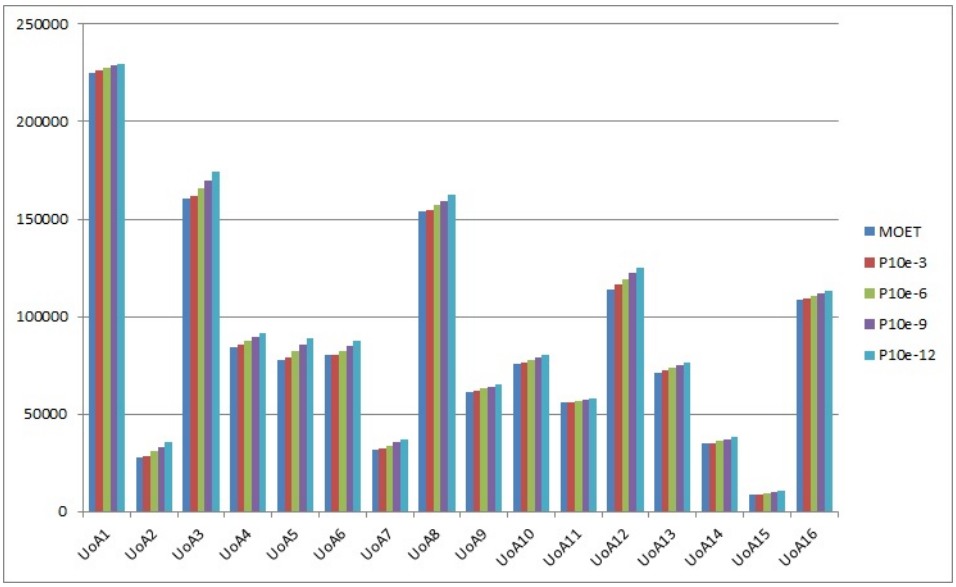

Figure 5.25: pWCET probabilities of exceedance for WBBC, LEON3 with HW Randomization.

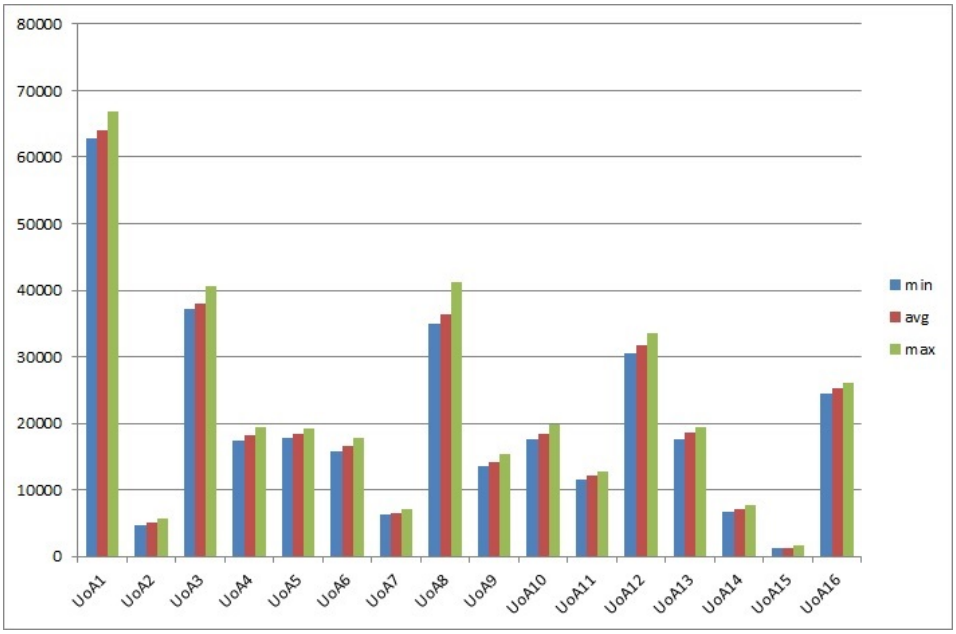

Figure 5.26: Execution time for WBBC on the P4080 hardware.

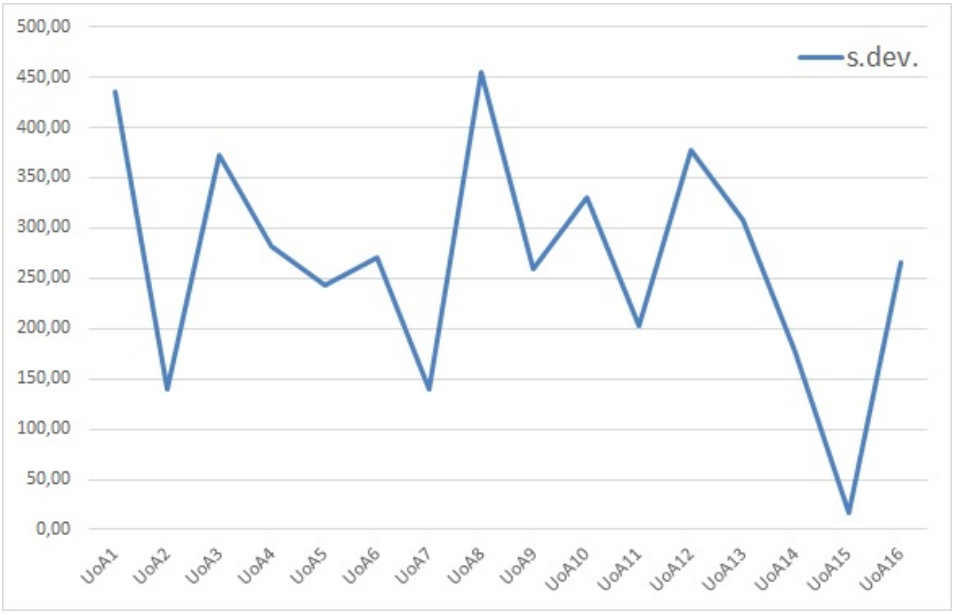

Figure 5.27: Standard deviation of the execution time obtained for the WBBC application on P4080 platform.

The variability in execution time can also be observed in Figure 5.27 where we present the standard deviation of each set of measured execution times. This difference varies from one cycle to another due to their code sizes. We decided to present the variability for each platform by itself because the results on P4080 are in a different range than those on Leon3. The observed variability is similar to that observed for the FPGA. The variability follows mostly the execution time.

The PMCs obtained on the P4080 show variability for the L1 caches misses and L2 cache hits. Results can be seen in Table 5.8. The amount of hit/miss in the L1 and L2 caches are different than expected. There are very low values in the L1 Data cache miss difficult to explain. This kind of behavior might be common for COTS hardware that lack documentation or are too complex to be fully understood. We consider these values as part of the behavior of the platform. The probabilistic analysis includes the effects of these uncertainties by analyzing the system as a whole.

	L1 I miss	L1 D miss	L2 hit
UoA1	277-292	252-257	2430-2534
UoA2	55-64	8	278-335
UoA3	338-354	67-69	1641-1767
UoA4	169-183	36	899-1031
UoA5	141-156	33-34	886-1029
UoA6	174-189	19-21	710-871
UoA7	71-82	4	338-439
UoA8	399-411	31-32	1351-1537
UoA9	157-167	12	595-727
UoA10	215-226	14-16	657-779
UoA11	124-134	42684	510-712
UoA12	262-275	47-48	961-1097
UoA13	152-162	34-35	589-705
UoA14	79-89	6	428-508
UoA15	17-26	42430	93-157
UoA16	151-165	41-45	768-1214

Table 5.8: PMC readings for WBBC on Leon3-R.

From statistical point of view, we examined each trace in order to verify if identical distributed and independence test pass. We observe that the results obtained on the P4080 platform have a higher tendency to pass the statistical tests and this is caused by the higher variability in execution which can be observed in the PMCs. The statistical test results can be seen in Table 5.9. The UoA15 is the only one that does not pass the identically distributed test due to its low variability. For the rest of the UoA we are able to apply our method, including for dependent data.

As decided prior, we present graphical results only from the UoA1, and a table containing the pWCET at 10^{-12} for all UoAs. Therefore, Figure 5.28 describes the measurement sample obtained after executing the WBBC execution on the P4080 platform. The histogram seems to indicate a normal behavior of the execution times which indicates a randomness in the data. This is confirmed by the Lag test (on the right of the figure). Also, we can observe the existence of a higher number of extreme values than for the previous data analyzed.

	i.d.	ind.
UoA1	✓	✓
UoA2	✓	x
UoA3	✓	✓
UoA4	✓	✓
UoA5	✓	✓
UoA6	✓	✓
UoA7	✓	✓
UoA8	✓	✓
UoA9	✓	✓
UoA10	✓	✓
UoA11	✓	✓
UoA12	✓	✓
UoA13	✓	✓
UoA14	✓	✓
UoA15	x	x
UoA16	✓	✓

Table 5.9: Identically distributed (i.d.) and independence tests results.

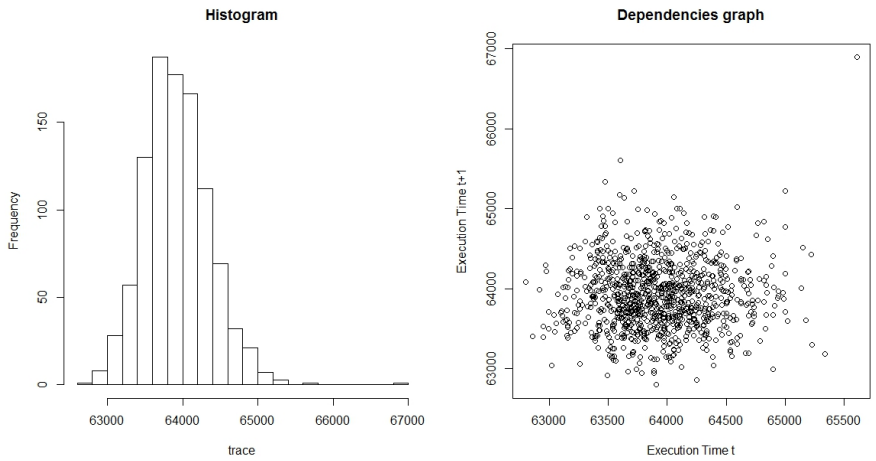

Figure 5.28: Graphical description of the UoA1 execution times obtained on the P4080 platform. On the left: histogram of the data. On the right: lag test result.

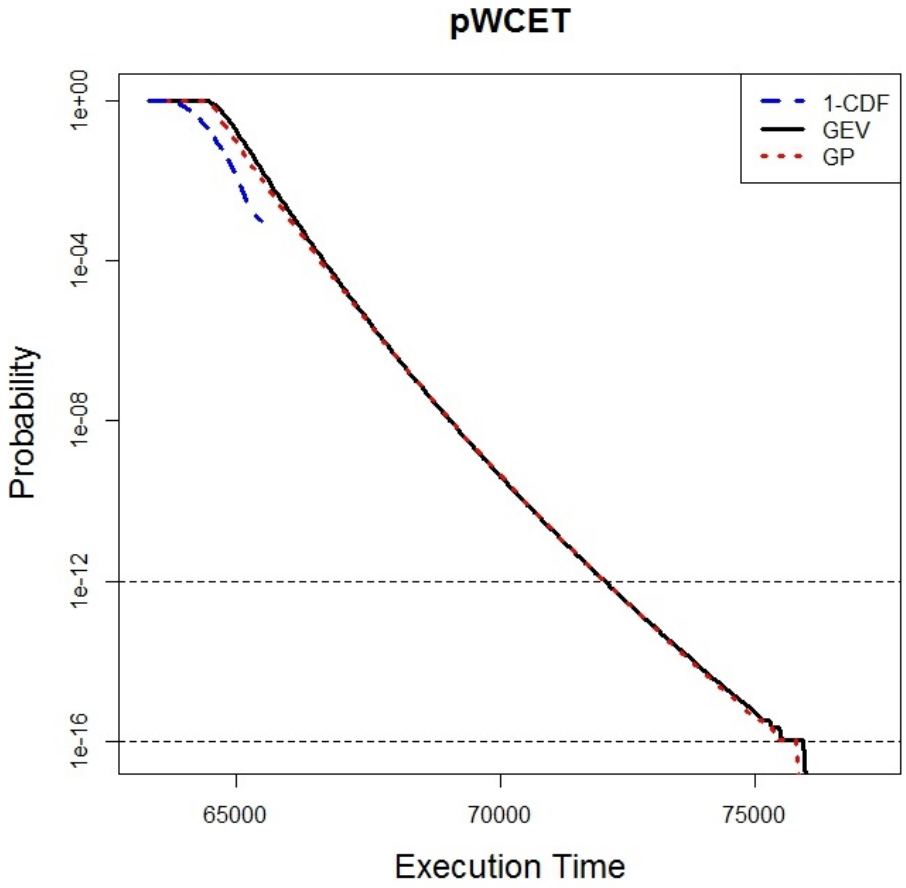

Figure 5.29: pWCET estimation for UoA1 of WBBC application executing on the P4080 platform.

In Figure 5.29, the estimated pWCET curve for UoA1 of the WBBC application can be seen. The blue line corresponds to the empirical distribution of observed execution times and the red and black distributions represent pWCET estimations for the given sample. The two curves coincide respecting the validation principle proposed in section 4.3.

The full summary of the probabilistic analysis can be found in Table 5.10 and Figure 5.30. They present the values observed for threshold probabilities in report with the maximum observed execution time. In line with the pWCET graphically illustrated in Figure 5.29, these results show that the pWCET prediction tightly upper-bound the actual observations. For certain UoAs, the MOET is higher than

the estimation at 10^{-3} or 10^{-6} (e.g. UoA8). We consider that this occurs because a rare measurement has been observed in the first 1000 measurements. Its appearance does not influence our analysis which relies on multiple extreme events for the estimation.

	MOET	P10e-3	P10e-6	P10e-9	P10e-12
UoA1	66894	66591	68289	69986	71684
UoA2	5771	5840	6401	6962	7514
UoA3	40704	40273	41802	43278	44808
UoA4	19497	19629	20472	21290	22132
UoA5	19237	19805	20743	21683	22622
UoA6	17785	18016	18999	19982	20965
UoA7	7186	7452	7983	8512	9043
UoA8	41127	38682	40191	41756	43321
UoA9	15489	15752	16932	18113	19293
UoA10	19739	20249	21363	22479	23594
UoA11	12810	13123	13591	14058	14525
UoA12	33646	33762	35161	36561	37960
UoA13	19490	21058	23056	25054	27053
UoA14	7798	7997	8411	8825	9240
UoA15	1653	0	0	0	0
UoA16	26135	26755	27784	28846	29874

Table 5.10: pWCET estimations for WBBC execution on the P4080 processor.

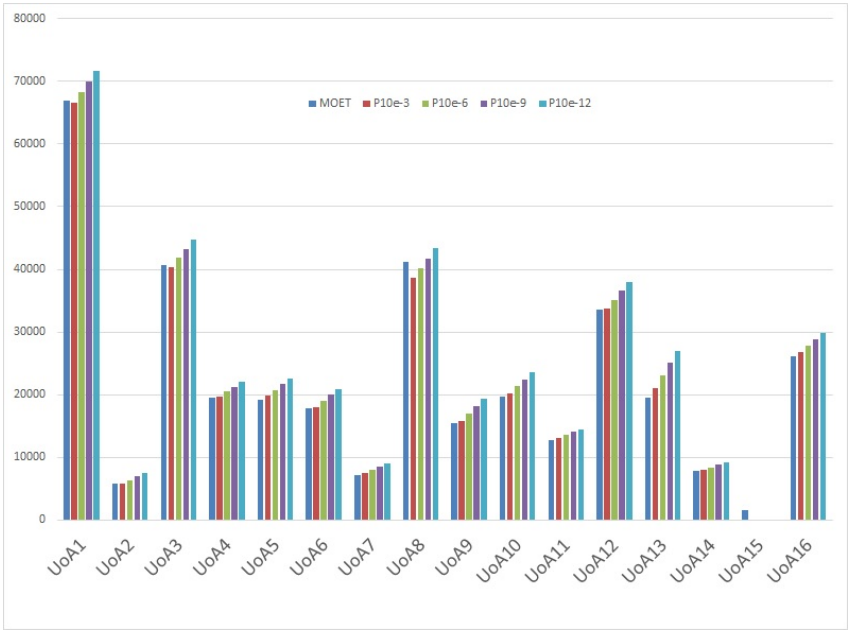

Figure 5.30: pWCET probabilities of exceedance for WBBC executing on the P4080 platform.

FCDC on Leon3-N

In this subsection, we present the results obtained with the Leon3 platform executing in normal mode. The sets of experiments have been carried out for the avionics FCDC application in isolation.

Figure 5.31 shows the statistics of execution times for WBBC on the FPGA platform. We observe small variations coming from an unknown or non-handled sources of jitter. The standard deviation that describes this variations is presented in Figure 5.32. Low variability makes fitting on an EVT distribution harder to achieve. In the case of FCDC application the variability is very small compared to the average of the samples and as a consequence most of the UoAs are not passing the i.i.d. tests. Even more, for those samples that pass the test fitting is hard to achieve. This is the reason we do not present any probabilistic analysis results for this section.

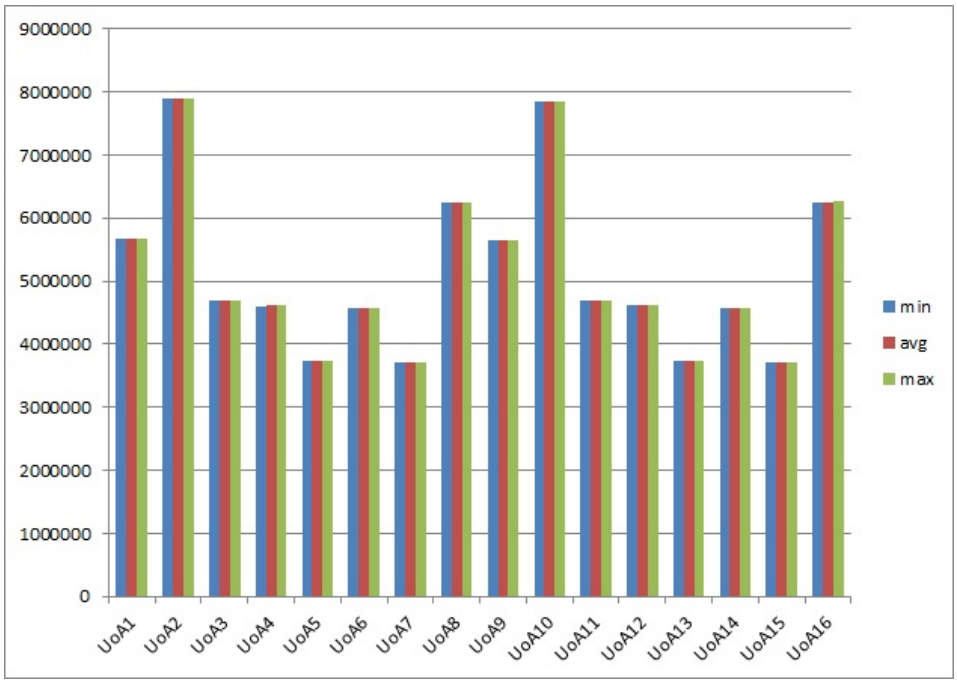

Figure 5.31: Execution time for FCDC on Leon3 in normal mode.

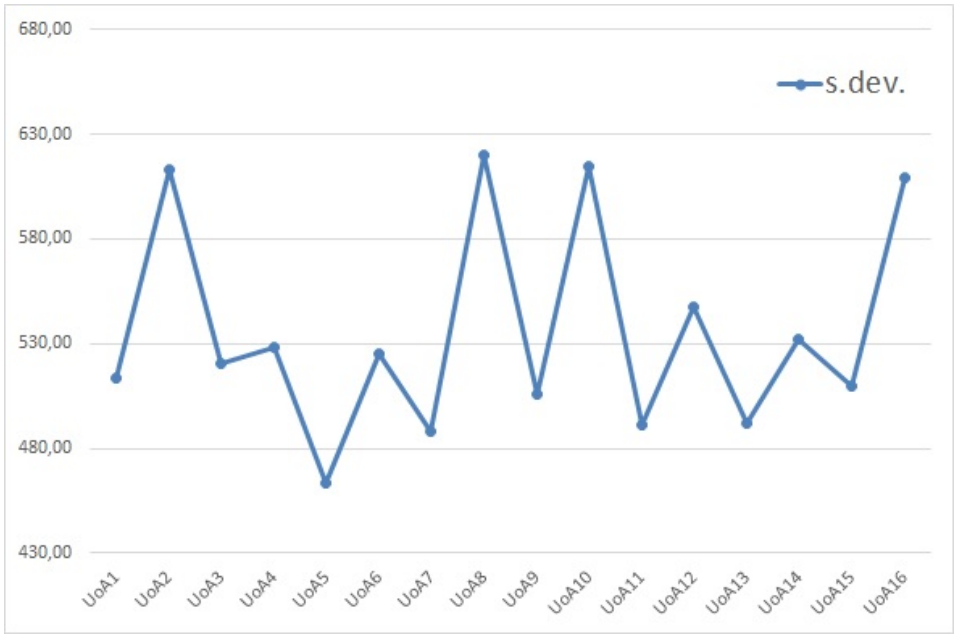

Figure 5.32: Standard deviation of the execution time obtained for the FCDC application on Leon3 in normal mode.

	icmiss	dcmiss	store	fpu
UoA1	27693-27694	7598	120171	195
UoA2	40218-40219	12058	158246	787
UoA3	20515-20516	8323	113431	339
UoA4	19401-19402	7528	134277	204
UoA5	15640-15641	6441	103748	189
UoA6	19132-19133	7655	132124	213
UoA7	15479-15480	6457	102908	201
UoA8	24244-24245	11489	194139	204
UoA9	27527-27528	7588	120045	189
UoA10	39888-39889	12066	156820	787
UoA11	20390-20391	8326	113388	339
UoA12	19483-19484	7520	134231	204
UoA13	15657-15658	6436	103749	189
UoA14	19224-19225	7646	132138	213
UoA15	15496-15497	6452	102912	201
UoA16	24340-24341	11480	194236	204

Table 5.11: PMC readings for FCDC on Leon3-N.

Table 5.11 shows the events monitored by the performance counters on the FPGA for each UoA. This is one of the reason we encounter small variability in this case study. Since the system is executed in normal mode, under similar conditions (i.e., inputs, executable), most of the PMCs do not change from one run to another. The results show that the variability in execution times for this scenario must be from other unidentified reasons.

Table 5.12 presents the i.i.d. results and also the variability reported to the average mean of the sample used (standard deviation devised by mean value). The variability column might explain the low rate of success of the i.i.d. tests.

	i.d.	ind.	variability
UoA1	x	x	9,06E-05
UoA2	✓	✓	7,77E-05
UoA3	x	x	1,11E-04
UoA4	x	x	1,15E-04
UoA5	x	x	1,24E-04
UoA6	x	x	1,15E-04
UoA7	x	x	1,32E-04
UoA8	x	x	9,92E-05
UoA9	x	x	8,94E-05
UoA10	x	x	7,83E-05
UoA11	x	x	1,05E-04
UoA12	✓	✓	1,19E-04
UoA13	✓	✓	1,32E-04
UoA14	x	x	1,16E-04
UoA15	✓	✓	1,37E-04
UoA16	✓	✓	9,73E-05

Table 5.12: Identically distributed (i.d.) and independence tests results for the FCDC application running on the Leon3 processor in normal configuration.

FCDC on Leon3-R

For this particular application, hardware randomization has a high impact on the variability of the execution times. Even though the application under analysis and the input vector used is the same, the hardware configuration changed. Therefore we cannot state that randomization helps in the use of measurement based probabilistic timing analysis. We deal here with a total different system and the pWCET estimation obtained in this section cannot be guaranteed for the same application running on the Leon3 processor in normal mode. Therefore, we restrict ourselves from comparing any distribution or measurement from the previous section and this one. Nevertheless, we compare the performance result of the different configuration as an motivation for this decision.

For this section, the experiments were done by running the FCDC application in

isolation on one of the cores of the randomized Leon3 processor. The randomization technique is proposed in the PROXIMA project. We present these results in order to show the impact of such a platform on the execution time behavior.

The execution times statistics are shown in Figure 5.33. We notice an increase in great both average and maximum values of the observations. This can be better seen in Figure 5.34. In other words, the randomized cache is from 3 to 4.5 slower than the use of the processor in its normal configuration.

The effect of randomization upon the variability of obtained execution times can be seen in Figure 5.35. Compared with the standard deviation of the execution times obtained on Leon3 in normal mode, the standard deviation for the randomized mode is highly superior.

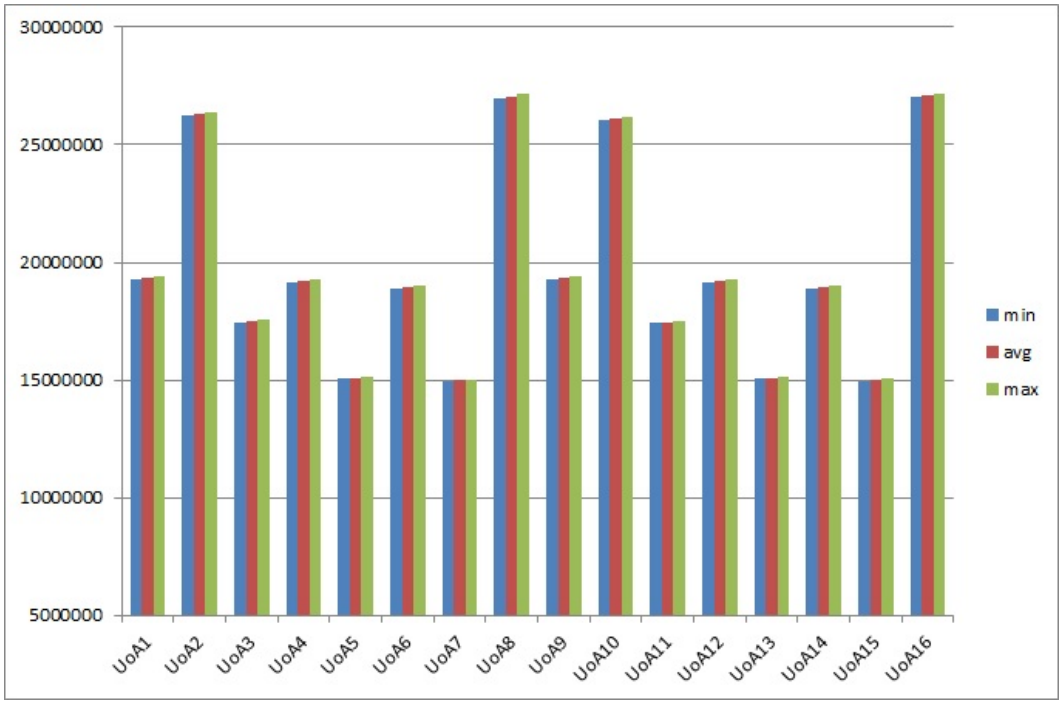

Figure 5.33: Execution time for FCDC on Leon3 in randomized mode.

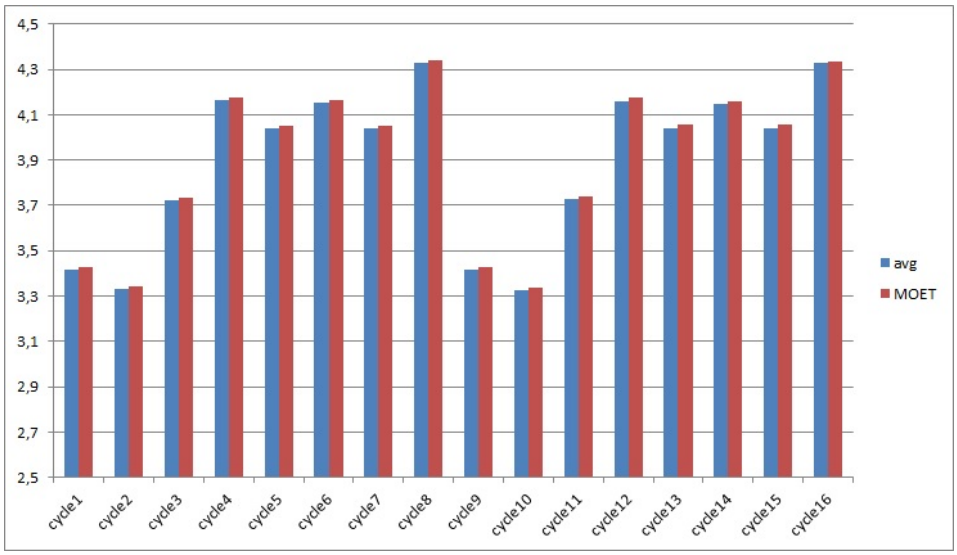

Figure 5.34: Comparing avg and max for FCDC, LEON3 in normal mode and in randomized mode.

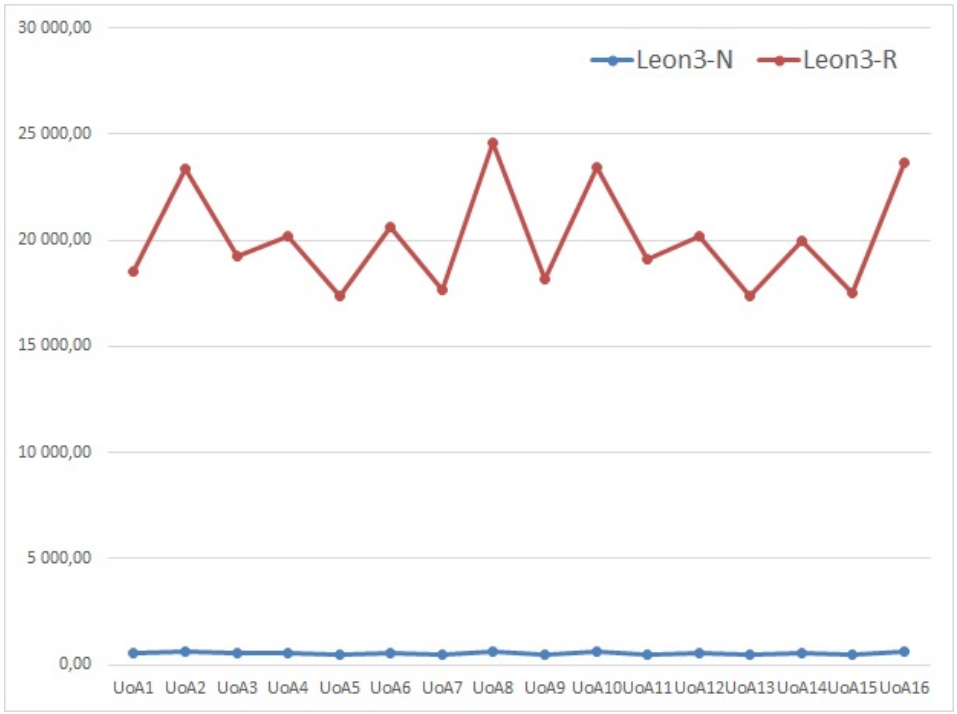

Figure 5.35: Standard deviation of the execution time obtained for the FCDC application on Leon3 in randomized mode (compared with normal mode).

An explication for this variability comes from the number of cache miss performed by the application during its execution. The performance monitoring counters reveal an increase in the number of misses on both instruction and data caches. The complete measurements can be found in Table 5.13.

	icmiss	dcmiss	store	fpu
UoA1	28373-28595	7857-7987	120171	195
UoA2	41566-41859	12483-12668	158246	787
UoA3	21231-21466	8591-8731	113431	339
UoA4	20255-20516	7734-7864	134277	204
UoA5	16238-16428	6608-6714	103748	189
UoA6	19999-20237	7847-7980	132124	213
UoA7	16073-16290	6613-6739	102908	201
UoA8	25547-25883	11771-11906	194139	204
UoA9	28214-28426	7854-7984	120045	189
UoA10	41211-41527	12483-12635	156820	787
UoA11	21098-21348	8597-8739	113388	339
UoA12	20333-20592	7727-7839	134231	204
UoA13	16257-16464	6608-6718	103749	189
UoA14	20085-20332	7851-7990	132138	213
UoA15	16096-16301	6617-6718	102912	201
UoA16	25639-25966	11767-11880	194236	204

Table 5.13: PMC readings for FCDC on Leon3-R.

For this case all the tests are passing which means that EVT for independent data can be applied on each UoA. However, because of the randomization factor, the experiments cannot be reproduced identically, and the collected measurements differ from one experiment to another. In some cases, different measurements produce different statistical tests results.

In order to present an example of execution times distribution corresponding to these and to their probabilistic estimation, we concentrate on UoA16 of the FCDC application. This UoA is chosen because it contains the highest execution time observed for the application.

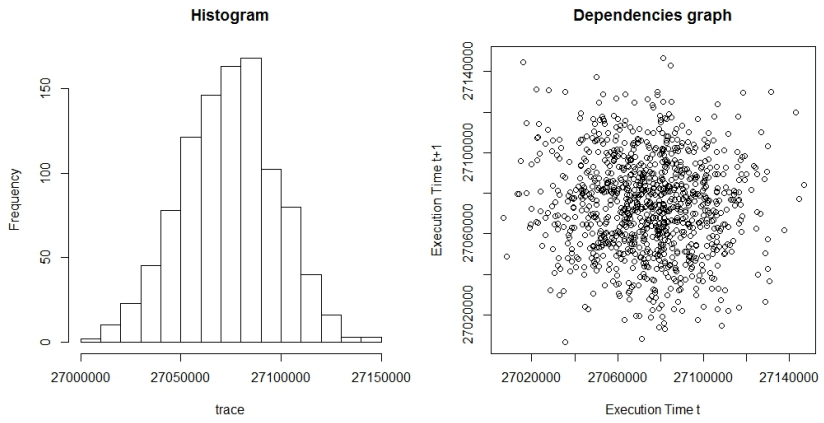

Figure 5.36: Graphical description of the UoA16 execution times obtained on the Leon3-R platform. On the left: histogram of the data. On the right: lag test result.

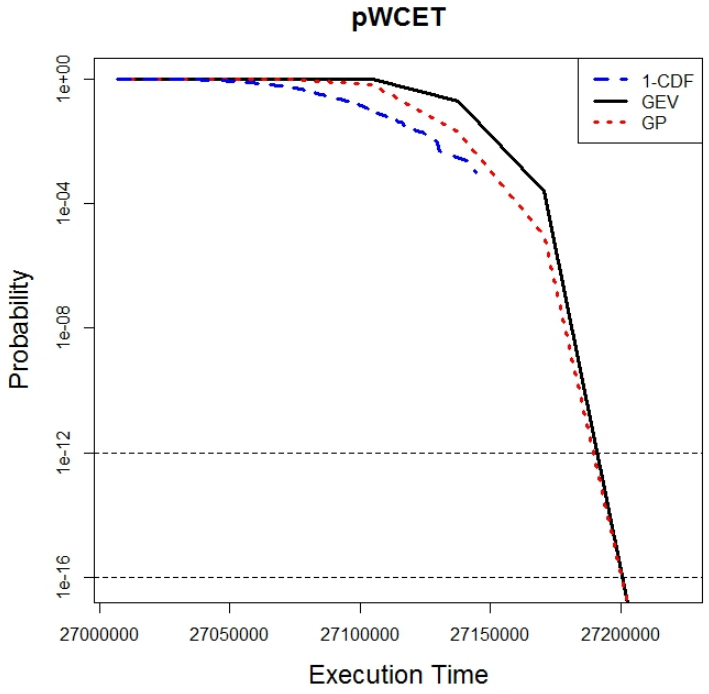

Figure 5.37: pWCET estimation for UoA16 of FCDC application executing on the Leon3 processor in randomized mode.

Figure 5.36 shows the histogram of execution times obtained for the FCDC

application on the hardware randomized architecture and the lag test result (no particular pattern is observed , in concordance with the independence tests).

Figure 5.37 shows the pWCET curve for UoA1 of FCDC, the blue line corresponds to the empirical distribution of observed execution times and the red and black distributions represent pWCET estimations for the given sample.

The full summary of the probabilistic analysis can be found in Table 5.14 and Figure 5.38. They present the values observed for threshold probabilities in report with the maximum observed execution time. In line with the pWCET graphically illustrated in Figure 5.37, these results show that the pWCET prediction tightly upper-bound the actual observations, with a margin at probability 10^{-12} ranging from 0,17% to 4% more than the maximum observed execution time.

	MOET	P10e-3	P10e-6	P10e-9	P10e-12
UoA1	19429761	19437805	19445848	19453892	19461935
UoA2	26369775	26512541	26655306	26798072	26940837
UoA3	17545826	17724161	17902497	18080832	18259168
UoA4	19262409	19318539	19374670	19430800	19486930
UoA5	15152684	15234720	15316757	15398793	15480830
UoA6	19016224	19190488	19364753	19539017	19713281
UoA7	15036100	15099838	15163576	15227313	15291051
UoA8	27134968	27173336	27211705	27250073	27288442
UoA9	19401119	19423702	19446284	19468867	19491449
UoA10	26190654	26266973	26343292	26419611	26495931
UoA11	17536373	17675156	17813938	17952721	18091503
UoA12	19271312	19282181	19293049	19303918	19314787
UoA13	15160959	15292313	15423668	15555022	15686376
UoA14	19020000	19035482	19050964	19066446	19081928
UoA15	15050219	15173089	15295958	15418828	15541698
UoA16	27146625	27157863	27169102	27180340	27191578

Table 5.14: pWCET estimations ofr FCDC executin on Leon3 in normal mode.

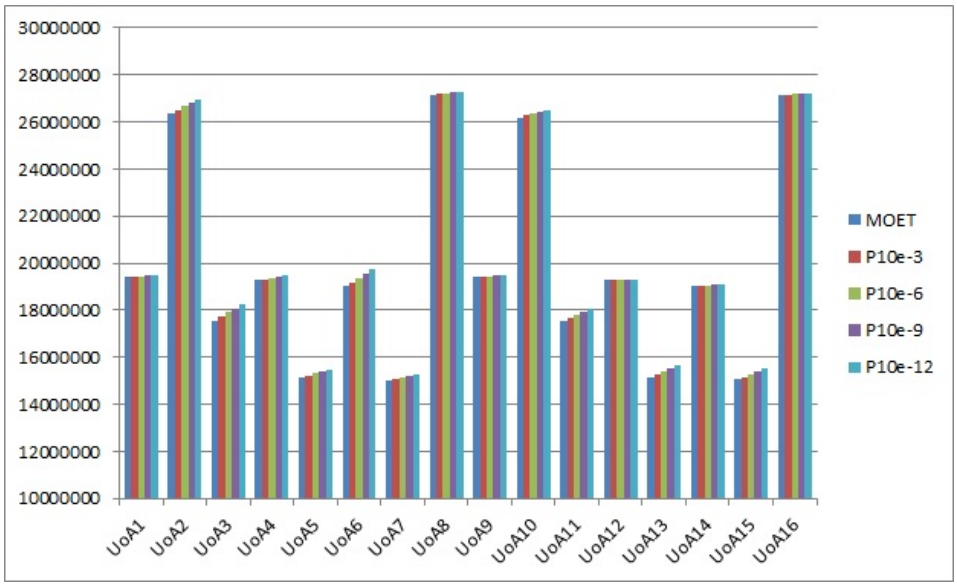

Figure 5.38: pWCET probabilities of exceedance for FCDC, LEON3 with HW Randomization.

FCDC on P4080

In this section, we present the same set of results obtained by executing the FCDC application on the P4080 platform running in COTS mode.

Figure 5.39 shows the execution time statistics. We observe less relative variability than for the WBBC. This suggests the variability sources are local and have no impact on long term. And on long term, every scenario tends to be comparable. Due to the small variability some of the UoAs do not pass the independence test but all of them are identically distributed which allow us to apply our methods for timing analysis. An example of how the measured execution times of the UoA are spread can be seen in Figure 5.40. On the left is the histogram of the data while on the right we plotted the data in the order of observation.

It can be observed that data obtained on this platform tend to be grouped on different levels. For this scenario, we use the method presented in section 4.5 that deals with data that has small variability.

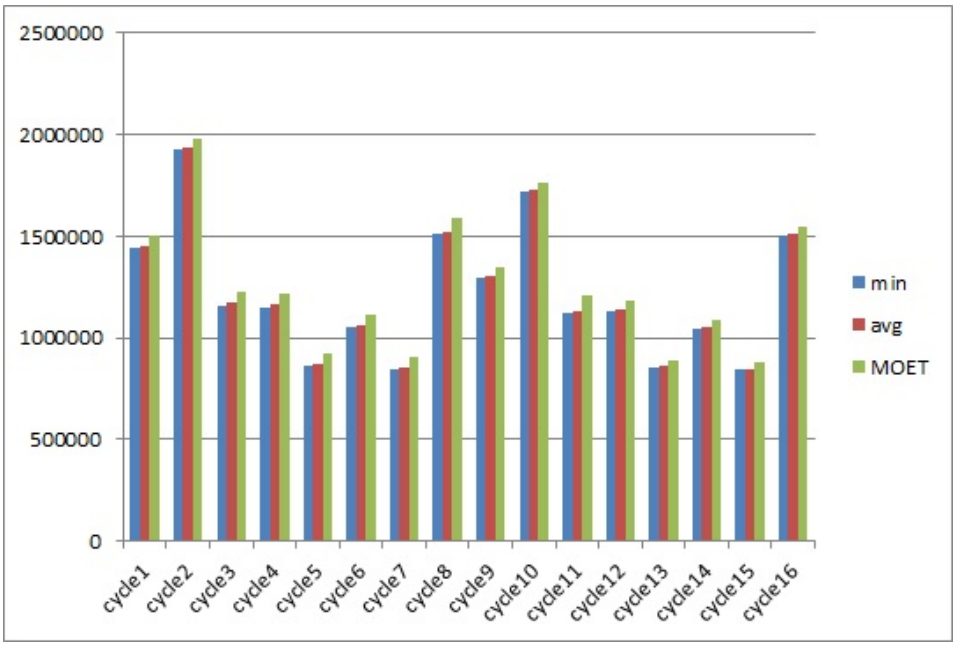

Figure 5.39: Execution time for FCDC on the P4080 platform.

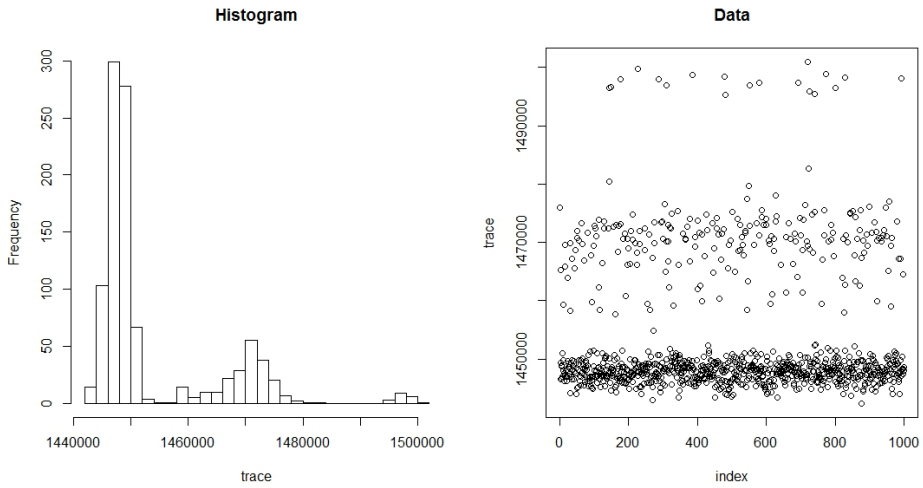

Figure 5.40: Graphical description of the UoA1 execution times obtained on the Leon3-R platform. On the left: histogram of the data. On the right: plot of the data.

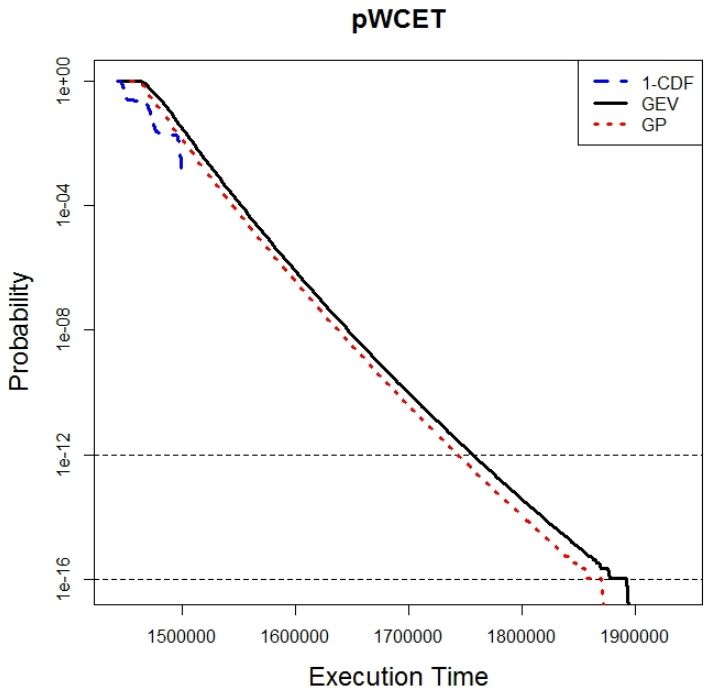

Figure 5.41: pWCET estimation for UoA1 of FCDC application executing on the P4080 platform.

In Figure 5.41 we present an example of pWCET curve obtained for the UoA1.

5.3 Conclusions

In this chapter we presented the experimental results obtained during this thesis. The data used is collected from an avionics application and executed on a series of hardware configurations in order to observe multiple behaviors. In this context, we used probabilistic analysis to obtain a mathematically motivated upper bounds on the worst case execution times of the samples observed. Our methods are adaptable to different behaviors: independent data, dependent data and data with small variability.

Chapter 6

General conclusions

Motivated by the continuous evolution of technology and by economical decisions, the avionics industry is being forced to consider the use of COTS hardware as a viable solution for future airplanes. Such a solution promotes performance with the cost of predictability. Due to strict certifications constraints, every component of an aircraft has to pass a series of tests during the verification and validation procedure. Bounding the worst case execution time of a program on a given platform is an important component of the certification process. In this context, timing analysis that relies on statistics and probability theory is seen as an option for determining WCET bounds.

6.1 Contributions

In this thesis we proposed and presented a series of probabilistic methods that can be used to determine a WCET bound for different real-time systems. We also presented the context in which these methods are used and the conditions that a system have to satisfy such that the WCET bound obtained is reliable.

Our contributions presented in this thesis are threefold:

1 Conditions for use of EVT in the real-time domain. For this contribution we give a detailed description of the different definitions for the notion of "independence", separating program independence from probabilistic and static independence. We also define three necessary conditions for the reliability of a probabilistic timing analysis: reproducibility and representativity of measurement protocol and reproducibility of the estimation method.

2 Estimation methodology for pWCET. In this part of the thesis, we adapt the existing Extreme Values Theory in order to be used for data composed of execution time measurements. Therefore, we present a methodology for choosing the block size for the block maxima method and the threshold value for the peak over threshold method in order to obtain a precise model estimation of the system under analysis. We present a validation method based on comparing the result obtained from the two different EVT distributions, GEV and GP. In the case of data containing dependences, we propose a declustering procedure in order to allow the pWCET estimation using GP distribution. We also present a method of probabilistic timing analysis based on EVT that can be used when the data has a small variability.

3 Experimental results. Finally, we test the proposed methods on a series of systems having as a common component the use of a real avionics application. Therefore, we presented the analysis results obtained for the application WBBC and FCDC running on different platforms. This allows us to observe different systems and to conclude on the adaptability of our methods on complex hardware like the P4080 PowerPC from Freescale. The presented results are a reflection of the measurements and analysis performed during this thesis and they were selected to demonstrate the use of the proposed probabilistic methods.

We agree that the use of probabilities can induce a certain level on uncertainty in the analysis results. Nevertheless, we consider that this is an affordable cost to be payed in report with the pessimism that static analysis can inflict on systems containing complex hardware (e.g. multi-core and manycore) or the budget needed to fully model such systems for static analysis.

6.2 Future work

The use of probabilities in the real-time domain is a new and developing approach, and the presented thesis is only a direction that these methodologies can take. Future work, based on related to the proposed methods, can be considered, and we have identified the following important problems:

- The use of presented methods to model the timing behavior of software running on multi-core hardware with one or multiple contenders.

- Use of probabilities to model contenders behavior in order to create a complex model of all contenders and to apply the result on the analyzed program.
- Applying Bayesian inferences in order to obtain a general system starting from measurements of multiple system configurations.
- Adapting existing methods for multiple path programs.
- The use of similar methods for the components of a real-time systems, like predicting the maximum number of cache miss in a system.
- Develop an analysis tool that fully eliminates human intervention in the decision of model choosing. This could be achieved with the development and use of more precise goodness of fit tests when deciding on the block size and threshold used.

We consider the use of probability theory in the real-time-domain as a promising solution that can work in harmony with existing, deterministic, solutions. And, as a consequence, we think that industries that have to perform timing analysis can rely on methods like the one proposed in this thesis as a verification solution for the ones they are using currently.

Appendix A

Statistical tests

A.1 Run test

The run test is a non-parametric test for the hypothesis that a set of numbers is independent. A run test is defined as a succession of similar values proceeded and followed by a different value (e.g., values that are either all above or below the mean or the median). To simplify computations, the data are first centered about their mean and the total number of runs is computed along with the number of positive and negative values. A positive run is then a sequence of values greater than zero, and a negative run is a sequence of values less than zero. We may then test if the number of positive and negative runs are distributed equally.
The statistical test hypotheses of the run test are:

H_0**:** *Data are randomly distributed* against
H_1**:** *Data are not randomly distributed.*

The associated statistical test is described by the Equation (A.1)

$$Z = \frac{R - E(R)}{\sqrt{V(R)}} \tag{A.1}$$

where R is the observed number of runs, $E(R)$ the expected number of runs and $\sqrt{V(R)}$ the standard deviation of the number of runs. The values of $E(R)$ and $V(R)$ are computed as follows:

$$\begin{aligned} E(R) &= \tfrac{2nm}{n+m} + 1 \\ V(R) &= \tfrac{2nm(2nm-n-m)}{(n+m)^2(n+m-1)} \end{aligned} \tag{A.2}$$

where n is the number of positive values and m the number of negative values in the sample.
During the tests applied to the data used in our paper we have used a significance level of $\alpha = 0.05$. This value is commonly accepted as sufficiently pessimistic and from our previous experience no example contradicting this hypothesis has been found. One may consider also the critical region of the runs test which rejects the null hypothesis if $|Z| > Z_{1-\alpha/2}$. As the computed p-value (the probability of obtaining the same value of the test or a larger one, if H_0 is true) is lower than the significance level α one should reject the null hypothesis H_0, and accept the alternative hypothesis H_1.

A.2 Kolmogorov-Smirnov test

The Kolmogorov-Smirnov (KS) test [Chakravarti and Laha, 1967] is based on the empirical distribution function (ECDF). Given N ordered data points $X_1, X_2, .., X_N$, the ECDF is defined as $E_N = n(i)/N$ where $n(i)$ is the number of points less than X_i and the $X_i$are ordered from smallest to largest value. This is a step function that increases by $1/N$ at the value of each ordered data point. The two sample KS test is a variation of this and instead of comparing an empirical distribution function to a theoretical distribution function, we compare two empirical distribution functions, as it follows : $D = |E_1(i) - E_2(i)|$ where E_1 and E_2 are the empirical distribution functions for the two samples. The hypotheses of the KS tests are

H_0: The two samples come from the same distribution,
H_1: The two samples do not come from the same distribution.

The KS two sample test statistic is defined as $D = |E_1(i) - E2_(i)|$. The significance level is α, chosen in advance (usually 0.05). The hypothesis regarding the distributional form is rejected if the test's statistic, D, is greater than the critical value obtained from a table. We can diminish the risk of rejecting the true hypothesis and in our case we better reject H_0 when H_0 is true than keeping H_0 when H_1 is true. If the other risk is more important, then we should change the order of the hypothesis.

A.3 Anderson-Darling Test

The Anderson-Darling test [Stephens, 1974] is used to test if a sample of data came from a population with a specific distribution. It is a modification of the Kolmogorov-Smirnov (KS) test and gives more weight to the tails than does the KS test. The KS test is distribution free in the sense that the critical values do not depend on the specific distribution being tested (note that this is true only for a fully specified distribution, i.e. the parameters are known). The Anderson-Darling test makes use of the specific distribution in calculating critical values. This has the advantage of allowing a more sensitive test and the disadvantage that critical values must be calculated for each distribution. Currently, tables of critical values are available for the normal, uniform, lognormal, exponential, Weibull, extreme value type I, generalized Pareto, and logistic distributions.

The Anderson-Darling test is defined as:

H_0: The data follow a specified distribution,
H_a: The data do not follow the specified distribution.

Test Statistic: The Anderson-Darling test statistic is defined as

$$A^2 = -N - S,$$

where

$$S = \sum_{i=1}^{N} \frac{(2i-1)}{N}[lnF(Y_i) + ln(1 - F(Y_{N+1-i}))]$$

F is the cumulative distribution function of the specified distribution. Note that the Y_i are the ordered data.

List of Figures

1.1 Cyber-physical systems - a Concept Map. Image made by Edward A. Lee after a taxonomy given by S. Shyam Sunder [Lee, Edward Ashford, 2012]. 3

1.2 Cost functions of real-time systems. 5

1.3 Basic notions related to timing analysis. The lower curve represents a subset of measured executions. Its minimum and maximum are the minimal observed execution times and maximal observed execution times. The darker curve, an envelope of the former, represents the times of all executions. Its minimum and maximum are the best case and worst case execution times, abbreviated BCET and WCET. . . 8

1.4 Evolution of code size in space, avionic and automotive embedded systems. 16

1.5 Possible execution time in the context of WCET bounds. 19

1.6 Possible relations between the CDFs of various random variables. . . 25

1.7 Possible relations between the CCDFs of various random variables. . 26

2.1 Distribution of execution times. 32

2.2 Space of interest of the central theorem compared to the extreme value theorem. 39

2.3 Example of transition from federated architecture to Integrated Modular Avionics. 53

3.1 The structure of an analyzable system. 60

3.2 Representation of dependences using the lag test. On the left figure we have independent data and on the right figure dependent data. . 69

3.3 The protocol of a (p)WCET estimation from different scenarios of execution conditions. 70
3.4 The WCET estimation is the same for different utilizations i, j of a reproducible WCET estimation method from the exactly same ordered set of execution times. 71
3.5 Different utilizations of a reproducible measurement protocol provides WCET estimates that are equal (or sufficiently close). 72
3.6 A representative measurement protocol provides equivalent subsets of execution times. 73
3.7 The absence of the reproducibility of a measurement protocol may prevent A_0 to converge to A. 74
3.8 The impact of the reproducibility and the representativity on the convergence of a measurement-based WCET estimation. 75

4.1 Examples of the GEV distributions with $\sigma = 1$ and $\mu = 0$. We mention that the intervals presented on the y axis are not the same for the two graphics. 80
4.2 Block maxima method: the largest value for each block is kept. . . . 81
4.3 Examples of return plots for GEV: (a) Return plot for three models with different shape values. (b) Return level plot containing the model and the observed values for a model having negative shape and 95% confidence interval. 83
4.4 Examples of the GP distribution with $\widetilde{\sigma} = 1$ and threshold=0. 86
4.5 PoT keeps all values above a given threshold. 86
4.6 Example of parameter estimates against threshold. 88
4.7 A global view of the pWCET estimation using GEV and GP. 90
4.8 Histogram plot (a) and lag test (b) of the data used for the comparison with existing methods. 90
4.9 The estimated model (a) and the return level plot (b) obtained using our method on the data used for the comparison with existing methods. 91
4.10 Comparison with Edgar's work: (a) return level plot for Edgar's method, (b) our pWCET estimation (in red) against the estimation obtained according to adversary method. 92

4.11 Comparison with Hansen's work: (a) return level plot for Hansen's method, (b) our pWCET estimation (in red) against the estimation obtained according to adversary method. 93
4.12 Comparison with Cucu's work: (a) return level plot for Cucu's method, (b) our pWCET estimation (in red) against the estimation obtained according to adversary method. 93
4.13 The two branches of EVT for dependent data and their relation with EVT for independent data. 95
4.14 Set of execution times with small variability. 97

5.1 Tricore Aurix Block Diagram. 100
5.2 Distribution of independent execution times for the program *prime* on the Aurix architecture. 101
5.3 Lag plot for independent execution times for the program *prime* on the Aurix architecture. 101
5.4 pWCET estimation of the program *prime* on the Aurix architecture from the measurements of Figure 5.3. 102
5.5 Dependent execution times for the program *prime* on the Aurix architecture with independent input values. 103
5.6 Lag test for dependent execution times for the program *prime* on the Aurix architecture while using independent input data. 103
5.7 pWCET estimation from dependent execution times of the program *prime* on the Aurix architecture. 104
5.8 WBBC functional modes. 108
5.9 WBBC Dynamic Behavior. 108
5.10 WBBC SCADE Nodes Cycles Scheduling. 109
5.11 FCDC functional modes. 111
5.12 FCDC Dynamic Behavior. 111
5.13 FCDC SCADE Nodes Cycles Scheduling. 112
5.14 Schematic of the FPGA architecture. 115
5.15 P4080 Block Diagram. 121
5.16 Execution time for WBBC on Leon3 in normal mode. 128
5.17 Standard deviation of the execution time obtained for the WBBC application on Leon3 in normal mode. 129

5.18 Graphical description of the UoA1 execution times obtained on the Leon3-N platform. On the left: histogram of the data. On the right: lag test result. 131
5.19 pWCET estimation for UoA1 of WBBC application executing on the Leon3 processor in normal mode. 132
5.20 Execution time for WBBC on Leon3 in randomized mode. 134
5.21 Comparing avg and max for WBBC, LEON3 in normal mode and in randomized mode. 134
5.22 Standard deviation of the execution time obtained for the WBBC application on Leon3 in randomized mode (compared with normal mode). 135
5.23 Graphical description of the UoA1 execution times obtained on the Leon3-R platform. On the left: histogram of the data. On the right: lag test result. 136
5.24 pWCET estimation for UoA1 of WBBC application executing on the Leon3 processor in randomized mode. 137
5.25 pWCET probabilities of exceedance for WBBC, LEON3 with HW Randomization. 139
5.26 Execution time for WBBC on the P4080 hardware. 139
5.27 Standard deviation of the execution time obtained for the WBBC application on P4080 platform. 140
5.28 Graphical description of the UoA1 execution times obtained on the P4080 platform. On the left: histogram of the data. On the right: lag test result. 142
5.29 pWCET estimation for UoA1 of WBBC application executing on the P4080 platform. 143
5.30 pWCET probabilities of exceedance for WBBC executing on the P4080 platform. 145
5.31 Execution time for FCDC on Leon3 in normal mode. 146
5.32 Standard deviation of the execution time obtained for the FCDC application on Leon3 in normal mode. 146
5.33 Execution time for FCDC on Leon3 in randomized mode. 149
5.34 Comparing avg and max for FCDC, LEON3 in normal mode and in randomized mode. 150

5.35 Standard deviation of the execution time obtained for the FCDC application on Leon3 in randomized mode (compared with normal mode). 150
5.36 Graphical description of the UoA16 execution times obtained on the Leon3-R platform. On the left: histogram of the data. On the right: lag test result. 152
5.37 pWCET estimation for UoA16 of FCDC application executing on the Leon3 processor in randomized mode. 152
5.38 pWCET probabilities of exceedance for FCDC, LEON3 with HW Randomization. 154
5.39 Execution time for FCDC on the P4080 platform. 155
5.40 Graphical description of the UoA1 execution times obtained on the Leon3-R platform. On the left: histogram of the data. On the right: plot of the data. 155
5.41 pWCET estimation for UoA1 of FCDC application executing on the P4080 platform. 156

List of Tables

1.1 Description of the Design Assurance Levels from the ARP-4761 [SAE, 1996]. 11

3.1 Body of program $Prog_{ex1}$. 67
3.2 Body of program $Prog_{ex2}$. 67

4.1 The most common laws distributed by attraction domain. 79

5.1 pWCET estimation on Aurix for some Mälardalen Benchmark programs. 105
5.2 Summary of results for avionics case study. 127
5.3 PMC readings for WBBC on Leon3-N. 130
5.4 Identically distributed (i.d.) and independence tests results. 131
5.5 pWCET estimations for WBBC execution on Leon3 in normal mode. 133
5.6 PMC readings for WBBC on Leon3-R. 136
5.7 pWCET estimations of WBBC executing on Leon3 in randomized mode. 138
5.8 PMC readings for WBBC on Leon3-R. 141
5.9 Identically distributed (i.d.) and independence tests results. 142
5.10 pWCET estimations for WBBC execution on the P4080 processor. . 144
5.11 PMC readings for FCDC on Leon3-N. 147
5.12 Identically distributed (i.d.) and independence tests results for the FCDC application running on the Leon3 processor in normal configuration. 148
5.13 PMC readings for FCDC on Leon3-R. 151
5.14 pWCET estimations ofr FCDC executin on Leon3 in normal mode. . 153

Nomenclature

AP Arrival Profile

AP/FD TCAS Autopilot/Flight Director Traffic Collision Avoidance System

APAP Average Priority Assignment Problem

ASIC Application-specific integrated circuits

ATSAW Airborne Traffic Situational Awareness

BCET Best Case Execution Time

CAN Control Area Networks

CCDF complementary cumulative distribution function

CCF CoreNet coherency fabric

CDA Continuous Descent Approach

CDF Cumulative Distribution Function

CLT Central Limit Theorem

CMS Centralized Maintenance System

COTS Commercial of the shelf

CPU Central Processing Unit

CRPS continuous ranked probability score

CRTES Critical real-time embedded systems

DAL Design Assurance Level

dcmiss data cache miss

DDR Double Data Rate memory

DL1 Level 1 data cache

DSP Digital signal processors

DTLB Data Transaction Lookaside Buffer

EASA European Aviation Safety Agency

ECU Engine Control Units

EDF Earliest Deadline First

eLBC Enhanced Local Bus Controller

EMI Electromagnetic Interference

ETP Execution Time Profile

EVT Extreme Value Theory

FAA Federal Aviation Administration

FCDC Flight Control Data Concentrator

FDIV Floating Point Division

FLS Flight Management Landing System

FPGA Field-programmable gate arrays

FPU Floating-point unit

FSQRT Floating Square Root

GEV Generalized Extreme Value

GLS GBAS Landing System

GOF Goodness of Fit

GP Generalized Pareto

HWM High Water Mark

i.i.d. Independent and Identically Distributed

I/O input/output

icmiss instruction cache miss

IL1 Level 1 instruction cache

IMA Integrated Modular Avionics

ipoint instrumentation point

ITLB Instruction Transaction Lookaside buffer

KS test Kolmorgorov-Smirnov test

L2TLB Level 2 Transaction Lookaside Buffer

LAW Local Access Windows

Leon3-N Leon 3 processor executing in normal configuration

Leon3-R Leon 3 processor executing in randomized configuration

LRM Line-Replaceable Module

LRU Least Recently Used

MAF Major Frame

MBPTA Measurement-Based Probabilistic Timing Analysis

MDA Maximum Domain of Attraction

MESI Modified/Exclusive/Shared/Invalid

MIF Minor Frames

MIT Minimum Inter-Arrival Time

MLE Maximum Likelihood Estimation

MMU Memory Management Unit

MOET Maximum Observed Execution Time

OANS On-board Airport Navigation System

OS Operating System

PCIe Peripheral Component Interconnect Express

pCRPD Probabilistic Cache Related Preemption Delays

pET probabilistic Execution Time

PMC Performance Counters

pMIT probabilistic Minimal Inter-Arrival Time

PoT Peak over Threshold

PRTS Probabilistic real-time systems

PTDA Probabilistic Time Demand Analysis

pWCET probabilistic Worst Case Execution Time

RM Rate Monotonic

ROPS Runway Overrun Protection System

SIL Safety Integrity Level

SPTA Static Probabilistic Timing Analysis

SRAM Static random-access memory

SRIO Serial Rapid I/O

SRMS Statistical Rate Monotonic Scheduling

SRS Simple Random Sample

STA Static timing analysis

STDA Stochastic Time Demand Analysis

TCAS RA Traffic Collision Avoidance System Resolution Advisory

UL2 Level 2 unified cache for data and instructions

UoA Unit of Analysis

WBBC Weight and Balance Back-up Computation

WCET Worst Case Execution Time

WCRT Worst Case Response Time

Bibliography

[A400M, 2015] A400M (2015). Airbus defence and space press release: Statement regarding accident information transmission (ait) to a400m operators as follow up to aot of 19 may. 2015 (cited on page 20).

[Abeni et al., 2012] Abeni, L., Manica, N., and Palopoli, L. (2012). Efficient and robust probabilistic guarantees for real-time tasks. *Journal of Systems and Software*, 85(5):1147–1156.

[Albers et al., 2008] Albers, K., Bodmann, F., and Slomka, F. (2008). Advanced hierachical event-stream model. In *Real-Time Systems, 2008. ECRTS'08. Euromicro Conference on*, pages 211–220. IEEE.

[Altmeyer et al., 2015] Altmeyer, S., Cucu-Grosjean, L., and Davis, R. (2015). Static probabilistic timing analysis for real-time systems using random replacement caches. *Real-Time Systems*, 51(1):77–123.

[Atlas and Bestavros, 1998] Atlas, A. and Bestavros, A. (1998). Statistical rate monotonic scheduling. In *19th IEEE Real-Time Systems Symposium (RTSS 1998)*.

[Audsley, 1991] Audsley, N. (1991). Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Technical Report YCS 164, University of York.

[Audsley, 2001] Audsley, N. (2001). On priority assignment in fixed priority scheduling. *Information Processing Letters*, 79(1):39–44.

[Audsley et al., 1993] Audsley, N., Burns, A., Richardson, M., Tindell, K., and Wellings, A. J. (1993). Applying new scheduling theory to static priority preemptive scheduling. *Software Engineering Journal*, 8(5):284–292.

[Audsley et al., 1995] Audsley, N. C., Burns, A., Davis, R. I., Tindell, K. W., and Wellings, A. J. (1995). Fixed priority pre-emptive scheduling: An historical perspective. *Real-Time Systems*, 8(2-3):173–198.

[Augustine, 1997] Augustine, N. R. (1997). *Augustine's laws*. AIAA.

[Axer and Ernst, 2013] Axer, P. and Ernst, R. (2013). Stochastic response-time guarantee for non-preemptive, fixed-priority scheduling under errors. In *Proceedings of the 50th Annual Design Automation Conference*, DAC '13, pages 172:1–172:7, New York, NY, USA. ACM.

[Baker, 1991] Baker, T. P. (1991). Stack-based scheduling of realtime processes. *Real-Time Systems*, 3(1):67–99.

[Baldovin et al., 2013a] Baldovin, A., Graziano, A., Mezzetti, E., and Vardanega, T. (2013a). Kernel-level time composability for avionics applications. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 1552–1554. ACM.

[Baldovin et al., 2013b] Baldovin, A., Mezzetti, E., and Vardanega, T. (2013b). Limited preemptive scheduling of non-independent task sets. In *Proceedings of the Eleventh ACM International Conference on Embedded Software*, page 18. IEEE Press.

[Balkema and De Haan, 1974] Balkema, A. A. and De Haan, L. (1974). Residual life time at great age. *The Annals of probability*, pages 792–804.

[Baruah, 2005] Baruah, S. (2005). The limited-preemption uniprocessor scheduling of sporadic task systems. In *Real-Time Systems, 2005.(ECRTS 2005). Proceedings. 17th Euromicro Conference on*, pages 137–144. IEEE.

[Baruah et al., 2012] Baruah, S., Bonifaci, V., D'Angelo, G., Li, H., Marchetti-Spaccamela, A., Megow, N., and Stougie, L. (2012). Scheduling real-time mixed-criticality jobs. *Computers, IEEE Transactions on*, 61(8):1140–1152.

[Baruah et al., 2011] Baruah, S. K., Burns, A., and Davis, R. I. (2011). Response-time analysis for mixed criticality systems. In *Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd*, pages 34–43. IEEE.

[Belady, 1966] Belady, L. A. (1966). A study of replacement algorithms for a virtual-storage computer. *IBM Systems journal*, 5(2):78–101.

[Berezovskyi et al., 2016] Berezovskyi, K., Guet, F., Santinelli, L., Bletsas, K., and Tovar, E. (2016). Measurement-based probabilistic timing analysis for graphics processor units. In *Architecture of Computing Systems - ARCS 29th International Conference*, pages 223–236.

[Berezovskyi et al., 2014] Berezovskyi, K., Santinelli, L., Bletsas, K., and Tovar, E. (2014). WCET measurement-based and extreme value theory characterisation of CUDA kernels. In *22nd International Conference on Real-Time Networks and Systems*, page 279.

[Bernat et al., 2005] Bernat, G., Burns, A., and Newby, M. (2005). Probabilistic timing analysis: An approach using copulas. *Journal of Embedded Computing*, 1(2):179–194.

[Bernat et al., 2002] Bernat, G., Colin, A., and Petters, S. M. (2002). Wcet analysis of probabilistic hard real-time systems. In *Real-Time Systems Symposium, 2002. RTSS 2002. 23rd IEEE*, pages 279–288. IEEE.

[Bin et al., 2014] Bin, J., Girbal, S., Pérez, D. G., Grasset, A., and Merigot, A. (2014). Studying co-running avionic real-time applications on multi-core cots architectures. In *Embedded Real Time Software and Systems conference*.

[Bini and Buttazzo, 2004] Bini, E. and Buttazzo, G. C. (2004). Schedulability analysis of periodic fixed priority systems. *Computers, IEEE Transactions on*, 53(11):1462–1473.

[Boniol et al., 2012] Boniol, F., Cassé, H., Noulard, E., and Pagetti, C. (2012). Deterministic execution model on cots hardware. *Architecture of Computing Systems–ARCS 2012*, pages 98–110.

[Broster and Burns, 2004a] Broster, I. and Burns, A. (2004a). Applying random arrival models to fixed priority analysis. In *the Proceedings of the Work-In-Progress of the 25th IEEE Real-Time Systems Symposium (RTSS04)*.

[Broster and Burns, 2004b] Broster, I. and Burns, A. (2004b). Random arrivals in fixed priority analysis. In *1st International Workshop on Probabilistic Analysis Techniques for Real-time and Embedded Systems (PARTES2004)*.

[Broster et al., 2002] Broster, I., Burns, A., and Rodriguez-Navas, G. (2002). Probabilistic analysis of can with faults. In *Real-Time Systems Symposium, 2002. RTSS 2002. 23rd IEEE*, pages 269 – 278.

[Burns, 1993] Burns, A. (1993). *Preemptive priority based scheduling: An appropriate engineering approach*. Citeseer.

[Burns et al., 2003] Burns, A., Bernat, G., and Broster, I. (2003). A probabilistic framework for schedulability analysis. In *Third International Embedded Software Conference (EMSOFT 2003)*, pages 1–15.

[Burns and Edgar, 2000] Burns, A. and Edgar, S. (2000). Predicting computation time for advanced processor architectures. In *Real-Time Systems, 2000. Euromicro RTS 2000. 12th Euromicro Conference on*, pages 89–96. IEEE.

[Burns and Wellings, 2001] Burns, A. and Wellings, A. J. (2001). *Real-time systems and programming languages: Ada 95, real-time Java, and real-time POSIX*. Pearson Education.

[Buttle, 2012] Buttle, D. (2012). Real-time in the prime-time. In *Keynote talk at the 24th Euromicro Conference on Real-Time Systems (ECRTS)*.

[Cassé and Sainrat, 2006] Cassé, H. and Sainrat, P. (2006). Otawa, a framework for experimenting wcet computations. In *3rd European Congress on Embedded Real-Time Software*, volume 1.

[Castillo et al., 1989] Castillo, E., Galambos, J., and Sarabia, J. M. (1989). The selection of the domain of attraction of an extreme value distribution from a set of data. In *Extreme Value Theory*, pages 181–190. Springer.

[Cazorla et al., 2012] Cazorla, F., Quinones, E., Vardanega, T., Cucu-Grosjean, L., Triquet, B., Bernat, G., Berger, E., Abella, J., Wartel, F., Houston, M., Santinelli, L., Maxim, D., Kosmidis, L., and Lo, C. (2012). Proartis: Probabilistically analyzable real-time system. *ACM Transactions on Embedded Computing Systems*.

[Cazorla et al., 2013] Cazorla, F. J., Quiñones, E., Vardanega, T., Cucu, L., Triquet, B., Bernat, G., Berger, E. D., Abella, J., Wartel, F., Houston, M., Santinelli, L., Kosmidis, L., Lo, C., and Maxim, D. (2013). Proartis: Probabilistically analyzable real-time systems. *ACM Trans. Embedded Comput. Syst.*, 12(2s):94–114.

[Chakravarti and Laha, 1967] Chakravarti, I. M. and Laha, R. G. (1967). Handbook of methods of applied statistics. In *Handbook of methods of applied statistics*. John Wiley & Sons.

[COLES, 2001] COLES, S. (2001). *An introduction to statistical modeling of extreme values*. Springer.

[Colin, 2001] Colin, A. (2001). Heptane webpage. *URL: http://www. irisa. fr/solidor/work/heptane-demo/heptane. html*.

[Cucu et al., 2008] Cucu, L., Pernet, N., and Sorel, Y. (2008). Periodic real-time scheduling: from deadline-based model to latency-based model. *Annals of Operations Research*, 159(1):41–51.

[Cucu and Tovar, 2006] Cucu, L. and Tovar, E. (2006). A framework for response time analysis of fixed-priority tasks with stochastic inter-arrival times. *ACM SIGBED Review*, 3(1).

[Cucu-Grosjean, 2013] Cucu-Grosjean, L. (2013). Independence - a missunderstood property of and for real-time systems. In *the 60th anniversary of A. Burns*.

[Cucu-Grosjean et al., 2012] Cucu-Grosjean, L., Santinelli, L., Houston, M., Lo, C., Vardanega, T., Kosmidis, L., Abella, J., Mezzeti, E., E., Q., and Cazorla, F. (2012). Measurement-based probabilistic timing analysis for multi-path programs. In *the 24th Euromicro Conference on Real-Time Systems (ECRTS12)*.

[Cyber, 2010] Cyber (2010). `http://cyberphysicalsystems.org/`.

[David and Puaut, 2004] David, L. and Puaut, I. (2004). Static determination of probabilistic execution times. In *the Euromicro Conference on Real-Time Systems(ECRTS)*.

[Davis et al., 2013a] Davis, R. I., Santinelli, L., Altmeyer, S., Maiza, C., and Cucu-Grosjean, L. (2013a). Analysis of probabilistic cache related pre-emption delays. In *IEEE Euromicro Conference on Real-Time Systems (ECRTS13)*.

[Davis et al., 2013b] Davis, R. I., Whitham, J., and Maxim, D. (2013b). Static probabilistic timing analysis for multicore processors with shared cache. *the 4th International Real-Time Scheduling Open Problems Seminar (RTSOPs2013), in conjunction with ECRTS2013*.

[de Oliveira and Gomes, 1984] de Oliveira, J. T. and Gomes, M. I. (1984). Two test statistics for choice of univariate extreme models. In *Statistical Extremes and Applications*, pages 651–668. Springer.

[Diaz et al., 2004] Diaz, J., Lopez, J., M., G., Campos, A., Kim, K., and Lo Bello, L. (2004). Pessimism in the stochastic analysis of real-time systems: Concept and applications. In *25th IEEE International Real-Time Systems Symposium (RTSS 2004)*, pages 197–207.

[Díaz et al., 2002] Díaz, J. L., García, D. F., Kim, K., Lee, C.-G., Lo Bello, L., López, J. M., Min, S. L., and Mirabella, O. (2002). Stochastic analysis of periodic real-time systems. In *Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS 2002)*, page 289.

[Edgar and Burns, 2001] Edgar, S. and Burns, A. (2001). Statistical analysis of WCET for scheduling. In *22nd IEEE International Real-Time Systems Symposium (RTSS 2001)*, pages 215–224.

[Edgar, 2002] Edgar, S. F. (2002). *Estimation of worst-case execution time using statistical analysis*. PhD thesis, University of York.

[Ekberg and Yi, 2014] Ekberg, P. and Yi, W. (2014). Bounding and shaping the demand of generalized mixed-criticality sporadic task systems. *Real-time systems*, 50(1):48–86.

[Ermedahl et al., 2009] Ermedahl, A., Fredriksson, J., Gustafsson, J., and Altenbernd, P. (2009). Deriving the worst-case execution time input values. In *Real-Time Systems, 2009. ECRTS'09. 21st Euromicro Conference on*, pages 45–54. IEEE.

[Faragó and Katz, 1990] Faragó, T. and Katz, R. W. (1990). Extremes and design values in climatology.

[Ferdinand and Heckmann, 2004] Ferdinand, C. and Heckmann, R. (2004). ait: Worst-case execution time prediction by static program analysis. *Building the Information Society*, pages 377–383.

[Ferdinand et al., 2001] Ferdinand, C., Heckmann, R., Langenbach, M., Martin, F., Schmidt, M., Theiling, H., Thesing, S., and Wilhelm, R. (2001). Reliable and

precise wcet determination for a real-life processor. In *Embedded Software*, pages 469–485. Springer.

[Fisher and Tippett, 1928] Fisher, R. A. and Tippett, L. H. C. (1928). Limiting forms of the frequency distribution of the largest or smallest member of a sample. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 24, pages 180–190. Cambridge University Press.

[Fuchsen, 2010] Fuchsen, R. (2010). How to address certification for multi-core based ima platforms: Current status and potential solutions. In *Digital Avionics Systems Conference (DASC), 2010 IEEE/AIAA 29th*, pages 5–E. IEEE.

[Galambos, 1982] Galambos, J. (1982). A statistical test for extreme value distributions. *Nonparametric Statistical Inference*, pages 221–230.

[Gardner and Lui, 1999] Gardner, M. and Lui, J. (1999). Analyzing stochastic fixed-priority real-time systems. In *5th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*.

[Gnedenko, 1943] Gnedenko, B. (1943). Sur la distribution limite du terme maximum d'une seris aleatoire. *Annals of Mathematics*, 44:423–453.

[Gogonel, 2014] Gogonel, A. (2014). Evt copernic pwcet estimator for real-time systems. `http://inria-rscript.serveftp.com/`.

[Goossens, 2003] Goossens, J. (2003). Scheduling of offset free systems. *Real-Time Systems*, 24(2):239–258.

[Goossens and Devillers, 1997] Goossens, J. and Devillers, R. (1997). The non-optimality of the monotonic priority assignments for hard real-time offset free systems. *Real-Time Systems*, 13(2):107–126.

[Griffin and Burns, 2010] Griffin, D. and Burns, A. (2010). Realism in statistical analysis of worst case execution times. In *10th Intl. Workshop on Worst-Case Execution Time Analysis*, pages 49–57.

[Griffin et al., 2015] Griffin, D., Lesage, B., Bate, I., Soboczenski, F., and Davis, R. I. (2015). Modelling fault dependencies when execution time budgets are exceeded. In *the 23rd International Conference on Real Time Networks and Systems, RTNS*, pages 65–74.

[Griffin et al., 2014] Griffin, D., Lesage, B., Burns, A., and Davis, R. I. (2014). Static probabilistic timing analysis of random replacement caches using lossy compression. In *22nd International Conference on Real-Time Networks and Systems, RTNS*, page 289.

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

[Gustafsson et al., 2010] Gustafsson, J., Betts, A., Ermedahl, A., and Lisper, B. (2010). The Mälardalen WCET benchmarks – past, present and future. In *the International Workshop on Worst-case Execution-time Analysis*.

[Haan, 1976] Haan, L. d. (1976). Sample extremes: an elementary introduction. *Statistica Neerlandica*, 30(4):161–172.

[Haan, 1970] Haan, L. F. M. (1970). On regular variation and its application to the weak convergence of sample extremes.

[Hansen et al., 2009] Hansen, J., Hissam, S., and Moreno, G. (2009). Statistical-based WCET estimation and validation. In *9th International Workshop on Worst-Case Execution Time (WCET) Analysis*.

[Hardy and Puaut, 2013] Hardy, D. and Puaut, I. (2013). Static probabilistic worst case execution time estimation for architectures with faulty instruction caches. In *Proceedings of the 21st International conference on Real-Time Networks and Systems (RTNS2013)*, pages 35–44. ACM.

[Hartigan, 1975] Hartigan, J. A. (1975). Clustering algorithms. *John Wiley & Sons*.

[Hasofer and Wang, 1992] Hasofer, A. M. and Wang, Z. (1992). A test for extreme value domain of attraction. *Journal of the American Statistical Association*, 87(417):171–177.

[Heath, 2002] Heath, S. (2002). *Embedded systems design*. Newnes.

[Henia et al., 2005] Henia, R., Hamann, A., Jersak, M., Racu, R., Richter, K., and Ernst, R. (2005). System level performance analysis–the symta/s approach. *IEE Proceedings-Computers and Digital Techniques*, 152(2):148–166.

[Holsti et al., 2000] Holsti, N., Langbacka, T., and Saarinen, S. (2000). Using a worst-case execution time tool for real-time verification of the debie software. *EUROPEAN SPACE AGENCY-PUBLICATIONS-ESA SP*, 457:307–312.

[Hosking and Wallis, 1987] Hosking, J. R. and Wallis, J. R. (1987). Parameter and quantile estimation for the generalized pareto distribution. *Technometrics*, 29(3):339–349.

[Hu et al., 2001] Hu, X., Zhou, T., and Sha, E.-M. (2001). Estimating probabilistic timing performance for real-time embedded systems. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 9(6):833 –844.

[Jalle et al., 2014] Jalle, J., Kosmidis, L., Abella, J., Quiñones, E., and Cazorla, F. J. (2014). Bus designs for time-probabilistic multicore processors. In *Proceedings of the conference on Design, Automation & Test in Europe*, page 50. European Design and Automation Association.

[Jean et al., 2012] Jean, X., Faura, D., Gatti, M., Pautet, L., and Robert, T. (2012). Ensuring robust partitioning in multicore platforms for ima systems. In *Digital Avionics Systems Conference (DASC), 2012 IEEE/AIAA 31st*, pages 7A4–1. IEEE.

[Kaczynski et al., 2007] Kaczynski, G., Lo Bello, L., and Nolte, T. (2007). Deriving exact stochastic response times of periodic tasks in hybrid priority-driven soft real-time systems. In *the 12th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'07), Greece.*

[Kaczynski et al., 2006] Kaczynski, G. A., Bello, L. L., and Nolte, T. (2006). Towards stochastic response-time of hierarchically scheduled real-time tasks. In *Proceedings of 11th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2006)*, pages 453–456.

[Kim et al., 2005] Kim, K., Diaz, J. L., Lo Bello, L., Lopez, J. M., Lee, C.-G., and Min, S. L. (2005). An exact stochastic analysis of priority-driven periodic real-time systems and its approximations. *IEEE Trans. Comput.*, 54(11):1460–1466.

[Kosmidis et al., 2013a] Kosmidis, L., Abella, J., Quiñones, E., and Cazorla, F. J. (2013a). A cache design for probabilistically analysable real-time systems. In

Proceedings of the Conference on Design, Automation and Test in Europe, pages 513–518. EDA Consortium.

[Kosmidis et al., 2013b] Kosmidis, L., Abella, J., Quinones, E., and Cazorla, F. J. (2013b). Multi-level unified caches for probabilistically time analysable real-time systems. In *Real-Time Systems Symposium (RTSS), 2013 IEEE 34th*, pages 360–371. IEEE.

[Kosmidis et al., 2014a] Kosmidis, L., Abella, J., Quiñones, E., and Cazorla, F. J. (2014a). Efficient cache designs for probabilistically analysable real-time systems. *IEEE Transactions on Computers*, 63(12):2998–3011.

[Kosmidis et al., 2014b] Kosmidis, L., Abella, J., Wartel, F., Quiñones, E., Colin, A., and Cazorla, F. J. (2014b). PUB: path upper-bounding for measurement-based probabilistic timing analysis. In *26th Euromicro Conference on Real-Time Systems, ECRTS*, pages 276–287.

[Kosmidis et al., 2013c] Kosmidis, L., Curtsinger, C., Quiñones, E., Abella, J., Berger, E., and Cazorla, F. J. (2013c). Probabilistic timing analysis on conventional cache designs. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013*, pages 603–606. IEEE.

[Kosmidis et al., 2013d] Kosmidis, L., Quinones, E., Abella, J., Vardanega, T., and Cazorla, F. J. (2013d). Achieving timing composability with measurement-based probabilistic timing analysis. In *Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2013 IEEE 16th International Symposium on*, pages 1–8. IEEE.

[Kosmidis et al., 2016] Kosmidis, L., Vargas, R., Morales, D., Quiñones, E., Abella, J., and Cazorla, F. J. (2016). Tasa: toolchain-agnostic static software randomisation for critical real-time systems. In *Computer-Aided Design (ICCAD), 2016 IEEE/ACM International Conference on*, pages 1–8. IEEE.

[Kotz and Nadarajah, 2000] Kotz, S. and Nadarajah, S. (2000). *Extreme value distributions: theory and applications*. World Scientific.

[Leadbetter et al., 1983] Leadbetter, M., Rootzén, H., and Lindgren, G. (1983). Extremes and related properties of random sequences and processes.

[Lee and Seshia, 2011] Lee, E. A. and Seshia, S. A. (2011). *Introduction to embedded systems: A cyber-physical systems approach.* Lee & Seshia.

[Lee, Edward Ashford, 2012] Lee, Edward Ashford (2012). `http://cyberphysicalsystems.org/`.

[Lehoczky, 1996] Lehoczky, J. (1996). Real-time queueing theory. In *10th of the IEEE Real-Time Systems Symposium (RTSS96)*, pages 186–195.

[Lesage et al., 2015a] Lesage, B., Griffin, D., Altmeyer, S., and Davis, R. I. (2015a). Static probabilistic timing analysis for multi-path programs. In *IEEE Real-Time Systems Symposium, RTSS*.

[Lesage et al., 2015b] Lesage, B., Griffin, D., Soboczenski, F., Bate, I., and Davis, R. I. (2015b). A framework for the evaluation of measurement-based timing analyses. In *the 23rd International Conference on Real Time and Networks Systems, RTNS*, pages 35–44.

[Leung and Merrill, 1980] Leung, J. Y.-T. and Merrill, M. (1980). A note on preemptive scheduling of periodic, real-time tasks. *Information processing letters*, 11(3):115–118.

[Leung and Whitehead, 1982] Leung, J. Y.-T. and Whitehead, J. (1982). On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance evaluation*, 2(4):237–250.

[Lima and Bate, 2017] Lima, G. and Bate, I. (2017). Valid application of evt in timing analysis by randomising execution time measurements. In *the 20th IEEE Real-Time and Embedded Technology and Application Symposium, RTAS*, pages 187–197.

[Lima et al., 2016] Lima, G., Dias, D., and Barros, E. (2016). Extreme value theory for estimating task execution time bounds: A careful look. In *28th Euromicro Conference on Real-Time Systems*, pages 200–211.

[Liu and Layland, 1973] Liu, C. L. and Layland, J. W. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61.

[Liu et al., 2013] Liu, M., Behnam, M., and Nolte, T. (2013). Applying the peak over thresholds method on worst-case response time analysis of complex real-time systems. In *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2013 IEEE 19th International Conference on*, pages 22–31. IEEE.

[Lopez et al., 2008] Lopez, J., Diaz, J. L., E., J., and Garcia, D. (2008). Stochastic analysis of real-time systems under preemptive priority-driven scheduling. *Real-time Systems*, 40(2):180–207.

[Lu et al., 2011] Lu, Y., Nolte, T., Bate, I., and Cucu, L. (2011). A new way about using statistical analysis of worst-case execution times. In *in the WiP sesion of the Euromicro Conference on Real-Time Systems*.

[Lu et al., 2012] Lu, Y., Nolte, T., Bate, I., and Cucu-Grosjean, L. (2012). A statistical response-time analysis of real-time embedded systems. In *Real-Time Systems Symposium (RTSS), 2012 IEEE 33rd*, pages 351–362. IEEE.

[Manica et al., 2012] Manica, N., Palopoli, L., and Abeni, L. (2012). Numerically efficient probabilistic guarantees for resource reservations. *Emerging Technologies and Factory Automation (ETFA), 2012 17th IEEE International Conference on*.

[Manolache et al., 2002] Manolache, S., Eles, P., and Peng, Z. (2002). Schedulability analysis of multiprocessor real-time applications with stochastic task execution times. In *Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design*, ICCAD '02, pages 699–706, New York, NY, USA. ACM.

[Manolache et al., 2004] Manolache, S., Eles, P., and Peng, Z. (2004). Schedulability analysis of applications with stochastic task execution times. *ACM Trans. Embed. Comput. Syst.*, 3(4):706–735.

[Marohn, 1998a] Marohn, F. (1998a). An adaptive efficient test for gumbel domain of attraction. *Scandinavian Journal of Statistics*, 25(2):311–324.

[Marohn, 1998b] Marohn, F. (1998b). Testing the gumbel hypothesis via the pot-method. *Extremes*, 1(2):191–213.

[Marti et al., 2001] Marti, P., Fuertes, J. M., Fohler, G., and Ramamritham, K. (2001). Jitter compensation for real-time control systems. In *Real-Time Systems Symposium, 2001.(RTSS 2001). Proceedings. 22nd IEEE*, pages 39–48. IEEE.

[Maxim et al., 2016] Maxim, C., Gogonel, A., Asavoae, I., Asavoae, M., and Cucu-Grosjean, L. (2016). Reproducibility and representativity-mandatory properties for the compositionality of measurement-based wcet estimation approaches. *CRTS 2016*, page 17.

[Maxim et al., 2012a] Maxim, C., Gogonel, A., Maxim, D., Cucu, L., et al. (2012a). Estimation of probabilistic minimum inter-arrival times using extreme value theory. In *the 6th Junior Researcher Workshop on Real-Time Computing (JRWRTC2012)*.

[Maxim et al., 2011] Maxim, D., Buffet, O., Santinelli, L., Cucu-Grosjean, L., and Davis, R. (2011). Optimal priority assignments for probabilistic real-time systems. In *the 19th International Conference on Real-Time and Network Systems (RTNS2011)*.

[Maxim and Cucu-Grosjean, 2013] Maxim, D. and Cucu-Grosjean, L. (2013). Response time analysis for fixed-priority tasks with multiple probabilistic parameters. In *proceeding of the* 34^{th} *IEEE Real Time Systems Symposium, 2013. RTSS'2013.*

[Maxim et al., 2012b] Maxim, D., Houston, M., Santinelli, L., Bernat, G., Davis, R. I., and Cucu-Grosjean, L. (2012b). Re-sampling for statistical timing analysis of real-time systems. In *Proceedings of the 20th International Conference on Real-Time and Network Systems*, pages 111–120. ACM.

[Maxim et al., 2015] Maxim, D., Soboczenski, F., Bate, I., and Tovar, E. (2015). Study of the reliability of statistical timing analysis for real-time systems. In *the 23rd International Conference on Real Time and Networks Systems, RTNS*, pages 55–64.

[Melani et al., 2013] Melani, A., Noulard, E., and Santinelli, L. (2013). Learning from probabilities: Dependences within real-time systems. In *Proceedings of 2013 IEEE 18th Conference on Emerging Technologies & Factory Automation, ETFA*, pages 1–8.

[Navet et al., 2000] Navet, N., Song, Y.-Q., and Simonot, F. (2000). Worst-case deadline failure probability in real-time applications distributed over controller area network. *J. Syst. Archit.*, 46(7):607–617.

[Nilsson et al., 1998] Nilsson, J., Bernhardsson, B., and Wittenmark, B. (1998). Stochastic analysis and control of real-time systems with random time delays. *Automatica*, 34(1):57–64.

[Nissanke et al., 2002] Nissanke, N., Leulseged, A., and Chillara, S. (2002). Probabilistic performance analysis in multiprocessor scheduling. *Computing Control Engineering Journal*, 13(4):171 – 179.

[OneWeb, 2015] OneWeb (2015). `http://oneweb.world`.

[Öztürk and Korukogu, 1988] Öztürk, A. and Korukogu, S. (1988). A new test for the extreme value distribution. *Communications in Statistics-Simulation and Computation*, 17(4):1375–1393.

[Pellizzoni and Lipari, 2005] Pellizzoni, R. and Lipari, G. (2005). Feasibility analysis of real-time periodic tasks with offsets. *Real-Time Systems*, 30(1-2):105–128.

[Pickands III, 1975] Pickands III, J. (1975). Statistical inference using extreme order statistics. *the Annals of Statistics*, pages 119–131.

[Poovey et al., 2007] Poovey, J. et al. (2007). Characterization of the eembc benchmark suite. *North Carolina State University*.

[Proartis, 2013] Proartis (2010-2013). `http://www.proartis-project.eu/`.

[Proxima, 2016] Proxima (2013-2016). `http://www.proxima-project.eu/`.

[Quinones et al., 2009] Quinones, E., Berger, E. D., Bernat, G., and Cazorla, F. J. (2009). Using randomized caches in probabilistic real-time systems. In *Real-Time Systems, 2009. ECRTS'09. 21st Euromicro Conference on*, pages 129–138. IEEE.

[RapiTime, 2006] RapiTime, W. (2006). tool homepage.

[Refaat and Hladik, 2010] Refaat, K. S. and Hladik, P.-E. (2010). Efficient stochastic analysis of real-time systems via random sampling. In *IEEE Euromicro Conference on Real-Time Systems (ECRTS 2010)*, pages 175–183.

[Reineke, 2014] Reineke, J. (2014). Randomized caches considered harmful in hard real-time systems. *Leibniz Transactions on Embedded Systems*, 1(1):03–1.

[RTCA, 2015] RTCA (2015). Radio technical commission for aeronautics (rtca) and european organisation for civil aviation equipment (eurocae). do-178c: Software considerations in airborne systems and equipment certification. 2011 (cited on pages 20, 39).

[SAE, 1996] SAE (1996). Sae international. aerospace recommended practices 4761 - guidelines and methods for conducting the safety assessment process on civil airborne systems and equipment. 1996 (cited on page 20).

[SAE, 2010] SAE (2010). Sae international. aerospace recommended practices 4754a - development of civil aircraft and systems. 2010 (cited on page 20).

[Santinelli and Cucu-Grosjean, 2011] Santinelli, L. and Cucu-Grosjean, L. (2011). Toward probabilistic real-time calculus. *ACM SIGBED Review*, 8(1):54–61.

[Santinelli et al., 2017] Santinelli, L., Guet, F., and Morio, J. (2017). Probabilistic real-time guarantees: There is life beyond the i.i.d. assumption. In *the 20th IEEE Real-Time and Embedded Technology and Application Symposium, RTAS*, pages 199–208.

[Santos et al., 2011] Santos, M., Lisper, B., Lima, G., and Lima, V. (2011). Sequential composition of execution time distributions by convolution. In *Proc. 4th Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS 2011)*, pages 30–37.

[Sha et al., 2004] Sha, L., Abdelzaher, T., Årzén, K.-E., Cervin, A., Baker, T., Burns, A., Buttazzo, G., Caccamo, M., Lehoczky, J., and Mok, A. K. (2004). Real time scheduling theory: A historical perspective. *Real-time systems*, 28(2-3):101–155.

[Sha et al., 1990] Sha, L., Rajkumar, R., and Lehoczky, J. P. (1990). Priority inheritance protocols: An approach to real-time synchronization. *Computers, IEEE Transactions on*, 39(9):1175–1185.

[Shapiro and Brain, 1987] Shapiro, S. and Brain, C. (1987). W-test for the weibull distribution. *Communications in statistics. Simulation and computation*, 16(1):209–219.

[Simalatsar et al., 2011] Simalatsar, A., Ramadian, Y., Lampka, K., Perathoner, S., Passerone, R., and Thiele, L. (2011). Enabling parametric feasibility analysis in real-time calculus driven performance evaluation. In *Proceedings of the 14th international conference on Compilers, architectures and synthesis for embedded systems*, pages 155–164. ACM.

[Stankovic, 1988] Stankovic, J. A. (1988). *Real-time computing systems: The next generation*. Department of Computer and Information Science, University of Massachusetts Amherst.

[Staschulat and Ernst, 2004] Staschulat, J. and Ernst, R. (2004). Multiple process execution in cache related preemption delay analysis. In *Proceedings of the 4th ACM international conference on Embedded software*, pages 278–286. ACM.

[Stephens, 1974] Stephens, M. A. (1974). Edf statistics for goodness of fit and some comparisons. *Journal of the American statistical Association*, 69(347):730–737.

[Tia et al., 1995] Tia, T., Deng, Z., Shankar, M., Storch, M., Sun, J., Wu, L., and Liu, J. (1995). Probabilistic performance guarantee for real-time tasks with varying computation times. In *IEEE Real-Time and Embedded Technology and Applications Symposium (ETFA 1995)*.

[Tiku and Singh, 1981] Tiku, M. and Singh, M. (1981). Testing the two parameter weibull distribution. *Communications in Statistics-Theory and Methods*, 10(9):907–918.

[Tindell and Clark, 1994] Tindell, K. and Clark, J. (1994). Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing and microprogramming*, 40(2-3):117–134.

[Vestal, 2007] Vestal, S. (2007). Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*, pages 239–243. IEEE.

[Wartel et al., 2013] Wartel, F., Kosmidis, L., Lo, C., Triquet, B., Quinones, E., Abella, J., Gogonel, A., Baldovin, A., Mezzetti, E., Cucu, L., Vardanega, T., and Cazorla, F. (2013). Measurement-based probabilistic timing analysis: Lessons from an integrated-modular avionics case study. In *the 8th IEEE International Symposium on Industrial Embedded Systems (SIES)*.

[Weissman, 1978] Weissman, I. (1978). Estimation of parameters and large quantiles based on the k largest observations. *Journal of the American Statistical Association*, 73(364):812–815.

[Wenzel, 2006] Wenzel, I. (2006). *Measurement-based timing analysis of superscalar processors*. na.

[Wenzel et al., 2005] Wenzel, I., Rieder, B., Kirner, R., and Puschner, P. (2005). Automatic timing model generation by cfg partitioning and model checking. In *Design, Automation and Test in Europe, 2005. Proceedings*, pages 606–611. IEEE.

[Wilhelm et al., 2008] Wilhelm, R., Engblom, J., Ermedahl, A., Holsti, N., Thesing, S., Whalley, D., Bernat, G., Ferdinand, C., Heckmann, R., Mitra, T., et al. (2008). The worst-case execution-time problem, overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems (TECS)*, 7(3):36.

[Wong et al., 2009] Wong, W. E., Debroy, V., and Restrepo, A. (2009). The role of software in recent catastrophic accidents. Technical report, University of Texas at Dallas.

[Yao et al., 2011] Yao, G., Buttazzo, G., and Bertogna, M. (2011). Feasibility analysis under fixed priority scheduling with limited preemptions. *Real-Time Systems*, 47(3):198–223.

[Zeng et al., 2009a] Zeng, H., Di Natale, M., Giusto, P., and Sangiovanni-Vincentelli, A. (2009a). Statistical analysis of controller area network message response times. In *Industrial Embedded Systems, 2009. SIES '09. IEEE International Symposium on*, pages 1 –10.

[Zeng et al., 2009b] Zeng, H., Di Natale, M., Giusto, P., and Sangiovanni-Vincentelli, A. (2009b). Stochastic analysis of can-based real-time automotive systems. *Industrial Informatics, IEEE Transactions on*, 5(4):388 –401.

[Zeng et al., 2010] Zeng, H., Di Natale, M., Giusto, P., and Sangiovanni-Vincentelli, A. (2010). Using statistical methods to compute the probability distribution of message response time in controller area network. *Industrial Informatics, IEEE Transactions on*, 6(4):678 –691.

[Ziccardi et al., 2015] Ziccardi, M., Mezzetti, E., Vardanega, T., Abella, J., and Cazorla, F. J. (2015). EPC: extended path coverage for measurement-based probabilistic timing analysis. In *2015 IEEE Real-Time Systems Symposium, RTSS*, pages 338–349.