UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Master in Innovation and Research in Informatics
High Performance Computing Specialisation

On the Analysis of the Timing Behaviour of Time Randomised Caches

Author: **Pedro Benedicte**

Advisors: **Francisco J. Cazorla**
Barcelona Supercomputing Center
Spanish National Research Council
Jaume Abella
Barcelona Supercomputing Center

Tutor: **Mateo Valero**
Computer Architecture Department, UPC
Barcelona Supercomputing Center

July 7, 2016

Acknowledgements

First of all, I want to deeply thank my advisors Fran and Jaume for their guidance and mentoring through the development of this thesis.

I also want to thank the rest of the people of the CAOS group at BSC who always offer help when is most needed.

Furthermore, I would like to acknowledge BSC for financially supporting my master studies through the Severo Ochoa scholarship.

Finally, I would like to thank my parents for their unconditional support both in my studies and in my life.

Abstract

Time Randomised caches (TRc), which can be implemented at hardware level or with software means on conventional deterministic cache designs, have been proposed for real-time systems as key enablers for Probabilistic Timing Analysis (PTA) and in particular its measurement-based variant: Measurement-Based Probabilistic Timing Analysis (MBPTA). A key parameter of MBPTA is the number of runs required to ensure representativity of the execution time measurements taken at analysis time with respect to execution times that can occur during system deployment, so that MBPTA can trustworthily be applied.

In this thesis, we propose several methods to determine whether the number of observations taken at analysis, as part of the normal MBPTA application process, capture the cache events significantly impacting execution time and Worst-Case Execution Time (WCET). If this is not the case, our techniques provide the user with the number of extra runs required so that cache events are captured ensuring trustworthiness on MBPTA provided WCET estimates. Our techniques have been evaluated using a set of synthetic benchmarks and a real avionics application.

Contents

List of Figures

List of Tables

1 Introduction

This chapter serves as an introduction to the reader to the work performed in this thesis. We start in Section 1.1 by explaining the increasing need for guaranteed computation of the real-time system industry. We further introduce how the techniques proposed in this thesis help accomplish the increasing performance requirements. Section 1.2 lists the main contributions of the thesis. We finish the chapter in Section 1.3 by presenting the structure of the rest of this document.

1.1 Motivation

Nowadays, the Real-Time Embedded Systems (RTES) industry represents an important part of the global chip market. Some predictions point out that it will drive the global chip demand in the following years [2]. RTES comprise a wide range of commercial products: from low cost commodity appliances such as microwaves to expensive and critical systems like cars or planes.

In all these systems, in addition to functional correctness, *timing correctness* plays a key part for the systems to function properly. Programs running on RTES must be completed in a limited amount of time called *deadline*. In order to know whether a specific program will fit in its assigned budget, a WCET estimate for the application is computed.

To that end, depending on how critical if not to miss the deadline, real-time systems can be classified into 3 categories:

- *Hard real-time systems:* this systems control the most critical operations, usually where a failure of the system can result in a fatality, e.g. Anti-lock Braking System (ABS) of a car, flight control system in planes... Frequent deadline misses are not desirable because of the consequences. This systems are also known as Critical Real-Time Embedded Systems (CRTES).

- *Soft real-time systems:* the system can miss several deadlines, since it will not result in a critical outcome. An example is a video decoding processor. If the processing of a frame of the video does not meet the deadline probably will not be noticeable for the final user. Even if some group of consecutive frames miss

the deadline, the user will see some distortion that, although undesirable, does not cause the whole system to stop working (the user can continue watching the movie).

- *Firm real-time systems:* the system can miss an occasional deadline and it will not cause a critical outcome, but the results will be discarded since they are of no use after the deadline. An example of this is Software-defined radio (SDR), where a missed deadline will result in some part of the audio stream not being heard by the user.

Critical Real-Time Embedded Systems is a fertile research as testified by the High-Performance Embedded Architecture and Compilation (HiPEAC) 2013's roadmap [13] that presents the relevance of the CRTES industry in Europe. It is highlighted the need for an increase in the guaranteed performance in these systems. This is so because newer functionalities require future CRTES systems that have a guaranteed performance higher than that of the current systems. As an example in the automotive industry cars are increasingly introducing more Advanced Driver Assistance Systems (ADAS) with high performance requirements.

The industry solution used until recently to increase guaranteed performance is to add more than one embedded processor per device (e.g. car, plane...), increasing significantly the number of processors per generation. For instance, a modern car contains up to 70 Electronic Control Units (ECUs) while in the following generations this number is predicted to grow half an order of magnitude [11]. When more tasks need to be implemented, the use of more individual systems is a possible solution. However, for a single task that requires a lot of single-thread performance this is not a valid solution, so a single more powerful processor must be used.

Most of the processors used until recently in CRTES are simple 8-bit and 16-bit [11] without any aggressive hardware acceleration components like pipelines, memory hierarchies, memory prefetchers or branch predictors. These hardware features are not being used because their complex behaviour complicates timing analysis, making increasingly complex to derive tight WCET estimates and providing evidence about their reliability. There are currently two timing analysis technique approaches: Static Timing Analysis (STA) and Measurement-Based Timing Analysis (MBTA):

- Static Timing Analysis: creates a mathematical model of the timing behaviour of the processor architecture. This model takes into consideration the latencies of each operation, the possible architectural components that could make this latencies change (pipeline stages, dependencies between instructions...). This is becoming increasingly difficult with new, more complex processor designs. Also some manufacturers do not provide all the implementation details, so there is some uncertainty in the mathematical model. This translates into more pessimistic assumptions to cover the uncertainty created by the lack of implementation details that are unclear or too complex, resulting in a longer (degraded) WCET.

- Measurement-Based Timing Analysis: computes the WCET by observing different execution time outcomes of the system. This phase is called analysis, while the later operational phase is called deployment. At the analysis phase several runs (R) are done changing the input sets so the timing changes. Using these timing observations and applying some statistical methods, the WCET is computed. However, with Measurement-Based Timing Analysis the conditions of the system at analysis time have to be the same of the ones at operation. Furthermore, the events that can significantly impact the execution time at operation must be captured in the analysis runs.

MBPTA [12, 33] is a variant of Measurement-Based Timing Analysis that allows to alleviate this problem. MBPTA introduces randomisation in certain architecture components that can generate different timing outcomes. This way, since the different timing outcomes will be seen with a certain probability (due to their randomness), it is possible to compute the probability that the worst case will be seen in the analysis runs.

For instance, this is a problem when trying to use a cache in a CRTES: the mapping of the objects in memory will impact the mapping of these objects in the different cache sets. The possible conflicts can affect the number of misses that the cache will have, having a noticeable impact on the program execution time. Thus, it is critical that this scenario (cache conflicts) is seen in the analysis runs for MBTA to be reliable.

However, often the mapping of the objects in memory is not easily controllable by the user, but done automatically by the operating system. Furthermore, assuming that the mapping can be specified, the task of manually forcing the scenario where cache conflicts are seen relies on the tester's skill.

To address this problem, in the context of MBPTA, time randomised caches have been proposed [26] to deal with this representativity problem. These caches introduce randomisation, by either software or hardware means, to the mapping of the objects in the cache. This means that the set where a certain object will be mapped has a specific and known probability.

1.2 Contribution

In this thesis we tackle the proposed problem of computing the probability that a cache conflict will occur with a specific cache configuration when allocating a specific number of objects of different sizes, for both software and hardware time randomised caches. The correct resolution of this problem will enable MBPTA to produce reliable WCET from the analysis runs.

For computing this probability, we propose 3 different mechanisms:

- The first consists in developing a probability tree with all the possible outcomes of the object allocation in the different sets.
- Since the previous method does not scale with the number of objects or sets, we propose a second model based on the mathematical interpretation of the multinomial coefficient. This method also has some computation restrictions when the number of objects to allocate is high.
- Finally, we use a method based on Monte Carlo simulations, which gives results in reasonable time with any number of objects or sets.

Furthermore, since increasing the number of analysis runs increases the probability of a cache conflict to be seen, we propose a method to derive the increased number of runs needed for the MBPTA analysis runs to be representative of those at operation time.

We used this methods to test both hardware and software time randomised caches. For each of these cache types, we tested them using synthetic object sequences as well as object sizes from a real avionics application.

Our models for computing the probability that a cache conflict will occur assume that all objects allocated in the cache are equally important, that is, will be actively requested by the processor.

1.3 Structure of the Thesis

The rest of this thesis is organised as follows:

Chapter 2 presents the background about timing analysis techniques and cache memories needed to understand the thesis.

Chapter 3 explains the main problem and the current solution for hardware time randomised caches, as well as explaining its limitations.

Chapters 4 and 5 analyse hardware time randomised caches and software time randomised caches respectively. For each one, we propose a solution based on a probability tree generation, a mathematical solution based on the multinomial coefficient and a statistical solution based on Monte Carlo simulations.

Chapter 6 presents the related work.

Finally, chapter 7 presents the main conclusions and the future work to be done in this area.

The work done in this thesis has been published in 3 international conferences, two as a research manuscript and one poster, found in the Appendix A: Published work.

2 Background

In this section we introduce the background of the two main topics needed to understand the contributions of this thesis: timing analysis techniques and cache memories.

2.1 Timing analysis techniques

In the hard real-time systems domain, the system (that is the specific hardware and software to be deployed) must pass some strict certification tests, for both functional validation as well as timing validation. Depending on the specific industry domain, a different certification process is applied, such as the DO-178B [36] in the aerospace industry or ISO 26262 [17] in the automotive industry.

Timing analysis techniques are used in order to verify if the applications that run on real-time systems fulfill their timing constraints. Since both the hardware and the software of these systems is increasing in complexity, it is getting more and more difficult to correctly – and in a time and cost efficient manner – verify that these constraints are being met.

This section explains the concept of timing analysis and introduces the two most used techniques in the industry: Static Timing Analysis (STA) and Measurement-Based Timing Analysis (MBTA), as well as an emerging variant of the latter: Measurement-Based Probabilistic Timing Analysis (MBPTA).

The execution time of a program is affected by various components, such as the hardware where it is executed. As the hardware becomes more complex, executing the same program in different hardware increases the variability in the execution time.

The input set of the application also affects the execution time of a program. The input data may change the control flow of a program, with each path taking a different execution time. This can happen because the execution paths have a different length or execute different instructions that take different time in the processor to execute.

Furthermore, in a system with other programs running at the same time, those other programs can affect the execution time since the resources are being shared. Multicore contention is the subject of intense study [15].

In Figure 1 we can see that the execution time of a task running on isolation can vary. Actually, it is really difficult to derive the complete execution time distribution

of a specific task on a specific hardware since all the possible factors that could have an impact on the execution time should be explored, which is unfeasible.

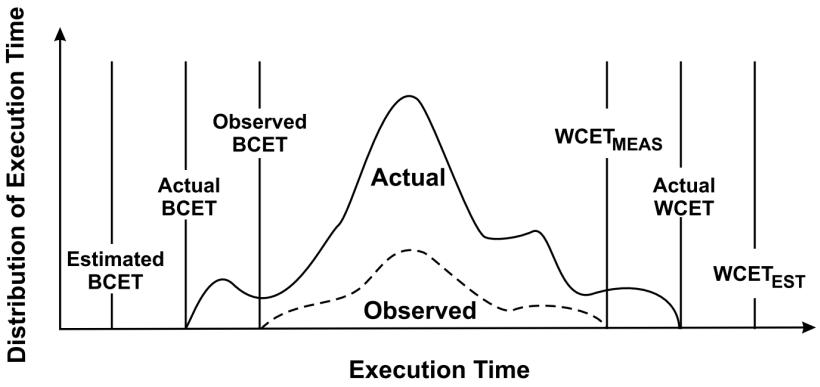


Figure 1: Execution time distribution [30].

On the left we have the Best-Case Execution Time (BCET), which is the minimum execution time of the task. On the right, we find the Worst-Case Execution Time (WCET), which is the highest execution time the task can have. Trying to obtain these values by just running the task many times does not result in satisfactory results. This is so because usually corner scenarios are quite uncommon and probably will not be seen in the number of runs performed. Moreover, it is hard for the end user to make worst-case scenario experiments. However, by observing the execution time we can find a minimum and maximum execution time (Observed BCET and $WCET_{MEAS}$ in Figure 1), that may or may not be close enough to the real BCET and WCET depending on various factors like the number of runs made, the type of distribution or the input sets selected.

Between the BCET and the WCET we find the execution time distribution. Given a specific execution time distribution, different industries have different goals. For instance, the high-performance computing industry has as a goal to move the whole execution time distribution to the left, because they want to reduce the average execution time. However, in the real-time systems industry, the main objective is to reduce the WCET, since is the critical time that makes the program successfully execute in its time frame or not.

Research on timing analysis techniques for real-time systems has two main goals: reducing the WCET and increasing reliability on derived WCET bounds under strict

time and cost constraints. The first one seeks to improve the worst-case performance of the system, allowing for longer and more complex tasks to execute while guaranteeing that they will finish in time. The second one tries to precisely compute the WCET. This is crucial since the wrong upper-bounding of the WCET can lead to a task believed to be trustworthy, taking more time and missing a deadline. On the other hand, if the WCET estimation is way over the actual WCET the system will be wasting resources and we will probably use more expensive hardware than the really needed.

The two most used analysis techniques to compute the Worst-Case Execution Time are Static Timing Analysis (STA) and Measurement-Based Timing Analysis (MBTA) [40]. In addition to these, recently some new techniques are based on the theory of probabilities. All these techniques have as a purpose to safely and tightly estimate WCET bounds.

On Static Timing Analysis the WCET is computed through analytic methods, without the need of actually executing the program on the real hardware. The possible execution paths that the application can take are analysed and then put into an abstract model of the real hardware. Using this information, the model computes an execution time for each execution path, and the final WCET is derived using these execution times.

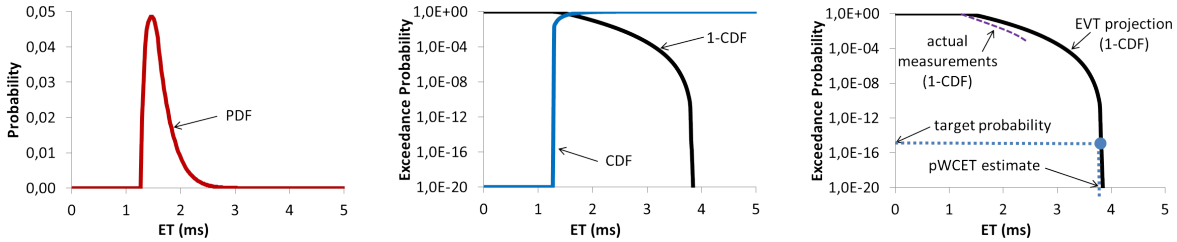
Measurement-Based Timing Analysis techniques compute the WCET by executing the program on the real hardware or a simulator and measuring the time it takes to finish. After several executions, a WCET bound is computed depending on the execution time distribution and the proper statistical methods.

Finally, Measurement-Based Probabilistic Timing Analysis techniques are based on MBTA, but use the probabilistic timing behaviour of the hardware to ensure that the WCET estimates are representative.

If timing analysis techniques can be successfully used to derive the WCET of an application in a specific system, we say that the application and the system are analysable. However, all the applications and systems cannot be analysed by all the timing analysis techniques. In the following sections we explain how the different methods compute the WCET and the limitations they present.

2.1.1 MBPTA

Measurement-Based Probabilistic Timing Analysis (MBPTA) is a new timing analysis methodology that has been actively researched in the recent years [5, 9, 10]. The main advantage of MBPTA is that it requires less information about the hardware specification than the other timing techniques, so that instead of making pessimistic estimations because of the lack of information, MBPTA can give tight and reliable guarantees of the WCET. In order to achieve this, some hardware components are changed to have a random behaviour instead of a deterministic one. Since true random behaviour has the independent and identically distributed (i.i.d.) properties, probabilistic analysis techniques can be used. For instance in a shared bus, the arbiter that decides what contender gets the bus could make this selection randomly, so that we do not need to know the previous state of the shared bus in order to know how it will behave, and the probability of each requester to use the bus can be computed.



(a) Probability distribution function (PDF)

(b) CDF and 1-CDF (logarithmic scale)

(c) Example of pWCET curve

Figure 2: Example of PDF, CDF, 1-CDF and pWCET curve.

Instead of giving a single WCET, MBPTA produces a Probability Distribution Function (PDF), which is a set of execution times each one with a given probability (Figure 2 a). These probabilities are obtained from a number of analysis runs, since it is a measurement-based timing analysis technique. The Cumulative Distribution Function (CFD) shown in Figure 2 b, and its inverse (1-CFD) are the same distribution but cumulative. In Figure 2 c, we see the inverse of the CFD in a dotted purple line (actual measurements). Once this curve is obtained, and using statistical analysis (such as Extreme Value Theory (EVT) [14, 29]), we compute the projection that this curve will have. This projection (shown as a black line in Figure 2 c) gives the probabilistic Worst-Case Execution Time (pWCET) for a given probability. For

instance in the example of Figure 2 c, the pWCET will be safely upper-bounded at 4 ms with a probability of 10^{-15} (blue dotted line).

2.2 Cache memories

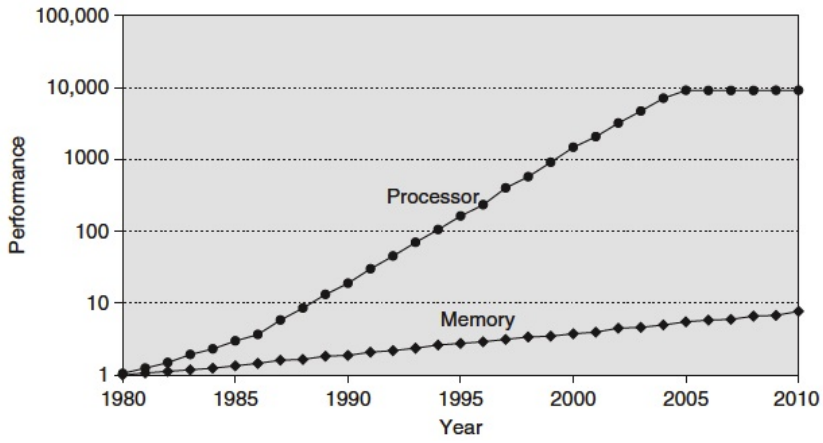


Figure 3: Evolution of the performance gap between the processor and memory [18].

In recent years the processor and the memory clock speed have been increasing at different rates (shown in Figure 3). This clock speed gap makes memory accesses to be the main performance bottleneck in processors. In order to hide this latency difference, cache memories are used. Caches are small SRAM memories that have lower access latencies than main memory (tipically implemented in DRAM). For instance, a cache access can have a latency between 4 and 50 processor cycles [3], while a memory access can have a latency of at least 50ns [3] (80,000 cycles at 1.6GHz). This difference in access latency usually results in an increased average CPU performance, so most commercial systems use caches. Since SRAM is more expensive to manufacture than DRAM, cache memories are smaller than main memory, only containing a subset of all the data in memory. Because of this, all data does not fit in the cache, meaning that is important to carefully select the data that will be stored in it, since choosing data that will not be reused will result in accesses going through cache to main memory, resulting in even higher latencies than that of systems without a cache. Caches are based on two principles that most computer programs follow: locality and temporality of data. Data locality implies that when a particular data object is accessed in a program, in the next cycles some adjacent data will be

accessed. Temporal locality happens when some data accessed is then accessed again in a brief period of time.

The way a cache typically works is the following: when a program is running, data accesses are issued by the processor. The cache is the first place where the processor requests the data. If the requested data is not in cache (cache miss), then the request is made to main memory. However, if the requested data is in the cache (cache hit), the data is just sent to the CPU from the cache, without a need of accessing the memory. The latency of a cache hit is significantly smaller than the latency of a cache miss (called miss penalty). The hit rate ($\frac{cache\ hits}{total\ accesses}$) is used to measure the performance.

2.2.1 Cache organisation

Main memory can be abstracted as a continuous chunk of data, where the position of the data in this chunk is the address. Even if main memory cannot store the whole data of the system, in this thesis we will suppose it can. Since a cache memory cannot store the whole data of the system, a subset of this data will be stored there. Since we want to take advantage of the data locality property of programs, when the cache requests some data to main memory, that data and some more continuous data will be brought to the cache. The total amount of data brought in every access is called a cache line or cache block, and it is a fixed size for the whole system execution. For organisation purposes, inside a cache every cache line will be treated as an atomic entity.

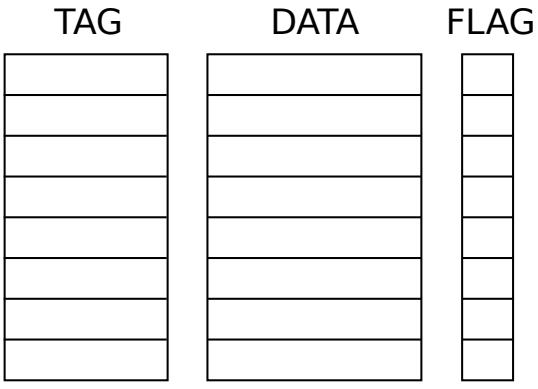


Figure 4: Basic cache structures.

A cache is formed by several cache entries, each entry containing the following

information (Figure 4):

- Tag: bits that allow to identify the cache line. Some bits of the accessed address are stored in order to compare with future data requests. If the tag bits match, the cache line stored is the same, and the access hits in cache. If the tag bits do not match, the cache line stores is not the same, and the access misses in cache.

- Data block: actual cache line stored in this cache entry. This usually occupies several bytes (common sizes are 32, 64 or 128 bytes).

- Flag bits: extra bits added to control the state of the cache entry. The number of bits depends on the features implemented in the cache. The most basic flag bit is the Valid bit, which informs if the cache entry contains valid data or not. Some other flag bits can be in charge of tracking how recently a cache line has been needed, for replacement purposes (explained later in Section 2.2.2).

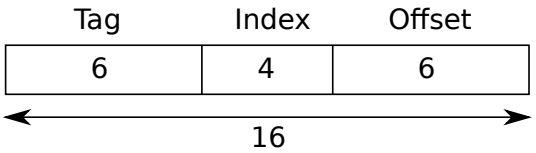


Figure 5: Memory address line.

The memory address is logically divided into 3 parts (as seen in Figure 5):

- Offset: bits that allow to identify the requested byte inside a cache line. The number of offset bits are the logarithm to the base 2 of the cache line size in bytes. For instance if the cache line is 64B, the number of offset bits is $log_2 64 = 6$.

- Index: bits used to know to what cache entry is assigned a cache block. The number of index bits are the logarithm to the base 2 of the number of cache entries. For example, if there are 16 cache entries, the number of index bits is $log_2 16 = 4$

- Tag: bits that allow to identify the cache line. It is formed by the remaining bits in the address $\#tag_{bits} = \#address_{bits} - \#offset_{bits} - \#index_{bits}$.

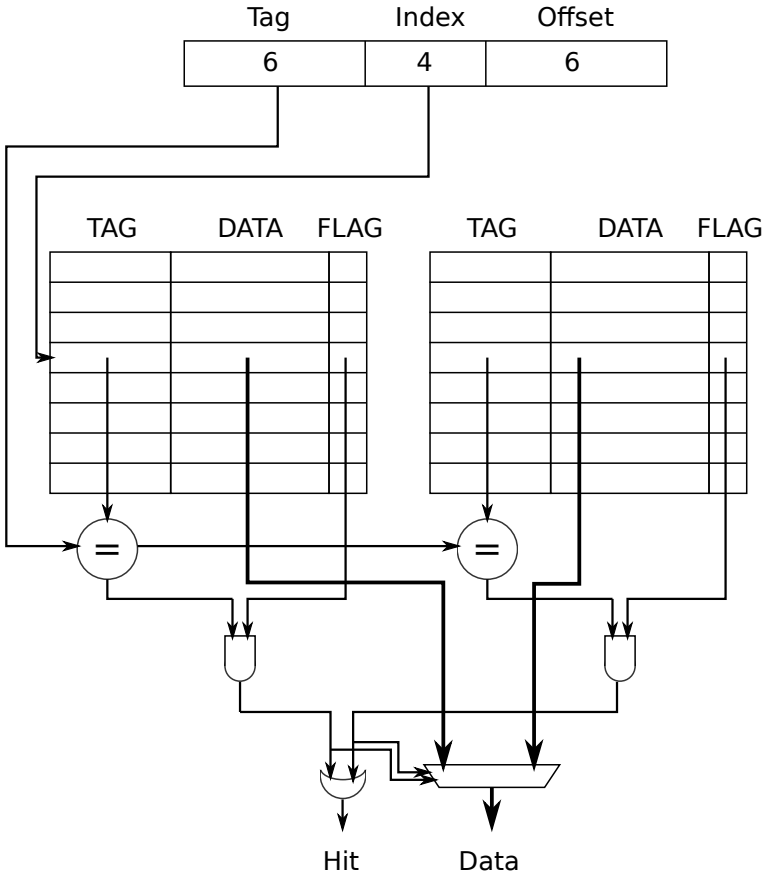


Figure 6: Set associative cache.

Cache entries are organised into sets and ways (see Figure 6). Cache sets (seen as rows in the figure are accessed using the index bits. This index bits are passed through a mapping function (usually a modulo function). Cache ways (seen next to each other in the figure) can store any cache entry that belongs to that set. If we want to store more cache lines than ways a set has, the replacement function decides which cache line is evicted (victim) in order to make space for the new cache line. A given address can only go to one possible set (the one specified by the mapping function), but can be in any of the ways of that set. This means that on a cache access, once the specific set where that access has to go has been determined, all ways have to be checked in case they contain the data. This check is done using the tag of the cache entry, as well as the valid bit.

Depending on the number of sets and ways a cache has, we can distinguish three cache types:

- Directly mapped: in a direct mapped cache there is only 1 way. In a cache with

n cache entries, there would be n sets of 1 way each one. This configuration has the lowest cost and latency, but results in higher miss rates.

- Fully associative: in this type of cache there is only 1 set. This means that if there are n cache entries, the cache will have 1 set with n ways. This configuration gives the highest possible hit rate of the three, but has a higher latency and implementation cost.

- Set associative: in a set associative cache there is a number of ways higher than 1, and a number of sets higher than 1. If there are n sets, we say it is n-way associative. These caches offer a trade-off between latency, cost and hit rate. Usually, all caches are set associative.

In Figure 6 we show an example of set associative cache comprising 2 ways and 8 sets.

2.2.2 Cache management

Placement functions On a cache with more than one set, the placement function is the one used to relate a cache address with a set. Some used placement functions are the following:

- Modulo placement: uses the index bits to directly map to a set. In an 8 set cache, if the index bits of an address are 010, the cache line corresponding to that address will be mapped to the set 2 of the cache. This implementation is simple, but has some drawbacks. For instance, in strided memory accesses if the stride is an even number, the cache lines are not regularly distributed through the cache sets. This generates conflict misses even though the whole dataset could fit in the array if another placement function was used.

- Prime modulo placement [22]: similar to the normal modulo function but the divisor is a prime number instead a power of two. This way better placement is achieved at the cost of increasing the hardware complexity.

- Random placement: a random placement policy uses a hash function that randomises the set where a cache line will be placed. However, even if randomisation is used, there is a need to be able to retrieve the same cache line we stored

in the cache. Using this hash function we guarantee that with the same RII number, a given address will always map to the same set. A cache that uses random placement can be seen in Figure 7.

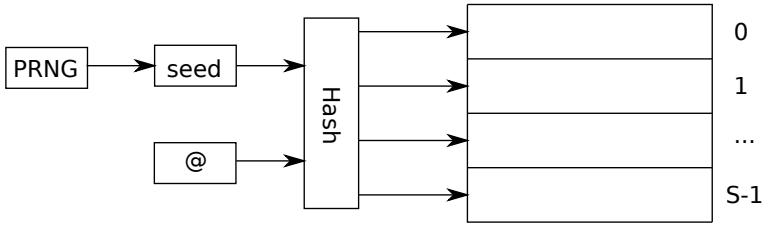


Figure 7: Cache with random placement.

Replacement policies On a cache with more than 1 way, once all the cache entries of a set are occupied and we want to allocate a new one, we must decide which cache entry to evict. This policy can have a significant impact in execution time since it will result in a higher or lower hit rate. Some examples of replacement policies are:

- LRU: the Least Recently Used replacement policy evicts the least recently used items first. Using the flag bits, it orders the different entries in a set with respect to when they were brought to the cache. It is a widely used algorithm because of its good performance and moderate cost.

- NRU: the Non Recently Used replacement policy distinguishes between two groups: recently used lines and non-recently used. The last cache lines added to the set have the recently used tag. When there is an eviction, an address from the non-recently used group is evicted. With this method, the cost of the flag bits is reduced to one bit per cache line, although some performance is lost compared to LRU.

- Random replacement [26, 33]: the random replacement policy randomly evicts a cache line. Although it does not give the best average performance, it enables PTA techniques to be used.

Currently, replacement policies are a hot topic in high-performance computer systems with several cache levels [20, 21, 34].

2.2.3 Time randomised caches

As explained in Section 2.1.1, MBPTA introduces randomisation into certain elements of the processor architecture so that pWCET can be obtained. One of the elements that introduces more time variability is the cache. Several consecutive accesses to a cache can produce all hits, all misses or a mix of both, depending on the previous instructions executed and the placement and replacement policies implemented by the cache, resulting in significantly different execution times.

Time Randomised caches (TRc) [33] introduce a random component on the placement and/or replacement policies. By randomising the cache behaviour, there is no longer need for the previous cache state in the timing analysis, allowing for MBPTA to be used. The two time randomised caches used in this thesis are hardware Time Randomised caches (hTRc) and software Time Randomised caches (sTRc), depending if the randomisation is done by hardware or software means.

Hardware Time Randomised Caches Hardware Time Randomised caches [33] (hTRc) introduce randomisation by implementing in their hardware a random placement and a random replacement function. The random placement function (Figure 7) is implemented hashing the memory address with a random seed. The output of the hash will indicate to what cache set the address is mapped. This seed is the same through the program execution, but it changes across different program executions. Since the hash is the same in a program execution, the same address can be retrieved when accessed multiple times. However, since we want to have the same address mapped to different sets through different runs, the seed is changed in between runs.

Software Time Randomised Caches Although hTRc enable Probabilistic Timing Analysis, most current commercial systems do not yet implement the placement and replacement policies needed for hTRc to be used. Software Time Randomised caches (sTRc) [26] use software randomization on top of COTS caches (modulo placement and LRU replacement). Software randomization (as seen in Figure 8) is a software layer that allocates objects in memory randomly. This random memory allocation is done between executions, so that in different executions the allocations of the same object will probably be different.

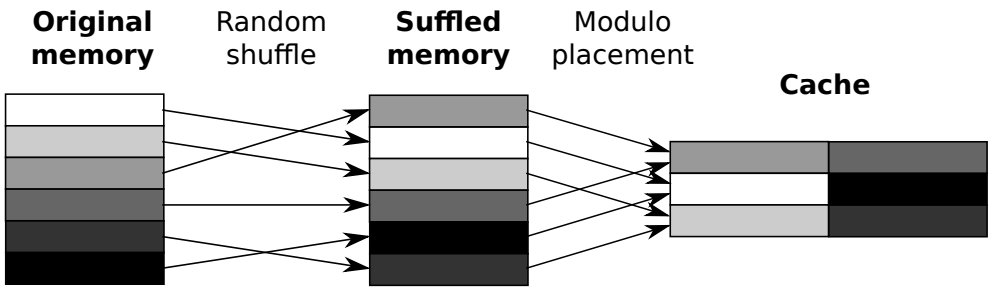


Figure 8: Software randomisation on top of conventional cache.

3 Problem statement

In this Section we introduce the problem addressed in this thesis: the representativity issue related with MBPTA when using Time Randomised caches. We explain the main problem in Section 3.1, then we comment on the impact this has on the safety standards in Section 3.2 and finally we explain the approximation previously proposed in Section 3.4.

3.1 MBPTA representativity requirements

As explained previously in Section 2.1.1, MBPTA provides a pWCET function that relates different execution times with the probability that they will be exceeded. These pWCET estimates obtained at analysis time remain valid at operation time only if the conditions in analysis time are the same or better than the ones later found on operation. The conditions that have to remain the same are the ones whose impact can significatively affect the execution time, for instance memory layout or arbitration in shared resources. Hence, for a correct application of MBPTA it is critically important to capture in the analysis-time measurements those events that can increase execution time meaningfully for a reliable application of MBPTA. These events are called Event of Interest (eoi).

In particular MBPTA imposes several requirements beyond those of EVT [10, 12], which, as used in MBPTA [12], requires a data sample of a random variable so that each execution time observation is independent and identically distributed. Additionally, MBPTA defines representativeness as the requirement in which the impact of any relevant event (eoi) affecting execution time is properly upper-bounded at analysis time, where a relevant event corresponds to any event occurring with a probability above a cutoff threshold (e.g. 10^{-9} per hour of operation). We relate such threshold to the assurance/integrity level of the task and the probability of hardware random failures allowed under such assurance/integrity level as dictated by the corresponding functional safety standards in the domain. In particular, in the context of MBPTA the cutoff threshold upper bounds the residual risk under which evidence of reliable operation is not had as detailed later in Section 3.1.1. While those events occurring with overly low probability become irrelevant for pWCET estimation

purposes, events occurring with higher probability need to be accounted for, and this requires that their effect is captured in the measurements taken at analysis time. This occurs because, while EVT predicts the combined impact and the probability of observed events, EVT cannot predict in general those events that are never observed and whose impact in execution time is larger than that of the observed ones [7].

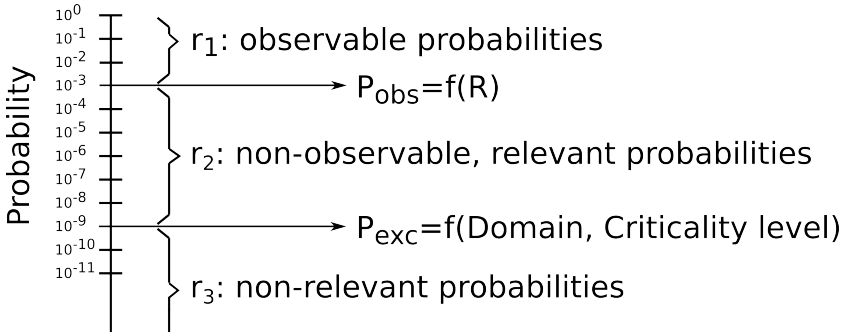


Figure 9: Range of probabilities of interest for MBPTA.

MBPTA representativeness (Figure 9) on the Event of Interest relates to two specific probabilities.

- The exceedance probability (P_{exc}) defines the lowest relevant probability for events occurring during operation. Events with smaller probability than P_{exc} are considered not relevant. P_{exc} is a function of the safety standards in the application domain and the criticality (integrity) level of the program. For instance, for commercial airborne systems at the highest assurance/integrity level (DAL-A), the maximum allowed random hardware failure rate in a system component is 10^{-9} per hour of operation [36]. Thus, we use the same threshold to upper bound the residual risk in the software verification process.

- The observable probability (P_{obs}) determines the lowest probability of occurrence of an event such that the probability of not observing it in the execution time measurements collected at analysis time is below a cutoff probability, e.g. $P_{coff} = 10^{-9}$. P_{obs} is a function of the probability of occurrence per run of the Event of Interest, P_{eoi}, and the number of runs R (observations) collected by MBPTA at analysis time:

$$P_{obs} = 1 - (1 - P_{eoi}^{R}) \tag{1}$$

For instance, for a cutoff probability of $P_{coff} = 10^{-9}$ and $R = 1,000$ runs, events with $P_{eoi} = 0.021$ will not be observed with a probability below P_{coff}, that is, $10^{-9} \geq (1 - 0.021)^{1000}$. It also follows that the higher the number of runs, the lower the P_{eoi} that can be captured. Similar to P_{exc}, P_{coff} is a function of the applicable safety standard and criticality level. P_{exc} and P_{obs} define (Figure 9) three probability ranges:

- $r1$ is the probability interval for which a probabilistic argument can be provided on the fact that events with a probability in this range are captured in R runs, i.e. the probability of not observing them is irrelevant.
- $r2$ corresponds to the probability interval for which events may not be observed, yet they are considered relevant for the correctness (i.e. non-optimism) of the pWCET estimate.
- $r3$ corresponds to the probability interval below the exceedance threshold. Events occurring during operation with such a low (or smaller) probability are regarded as irrelevant in relation with the corresponding safety standard and criticality of the function.

This thesis aims to determine P_{eoi}, taking different actions depending on its value: If $P_{eoi} \in r1$ or $P_{eoi} \in r3$ MBPTA is deemed as reliable as described above. However, when $P_{eoi} \in r2$ it is required to determine the increase in the number of runs (Δ_R) to carry out. Increasing the number of runs to $R = R + \Delta_R$ increases P_{obs} to P'_{obs} such that events with lower probability can be observed, making that $P'_{eoi} \in r1$. Hence, our proposed approach is vital to maintain confidence on MBPTA-provided pWCET estimates.

3.1.1 Cache representativity challenges

The Heart of Gold (HoG) approach [7] is the first attempt to address representativeness issues of cache related events for hardware Time Randomised caches. HoG, whose representativeness findings for hTRc were also identified in [32, 35], addresses the scenario in which the execution times, obtained from a MBPTA-compliant architecture deploying TRc [23] cause MBPTA to yield optimistic pWCET estimates. It

is noted [7] that the number of addresses competing for a set is the critical parameter affecting execution time noticeably. If competing addresses fit in the cache set – so there are up to W addresses where W is the cache associativity – then they will end up fitting in the cache set after some random evictions. Conversely, if there are more than W cache line addresses competing for the cache set space, then they do not fit and evictions will occur often, if all those cache lines are accessed often and in an interleaved fashion. This scenario where more than W cache line addresses compete for the space in a cache set is therefore the cache event of interest. We illustrate this scenario by performing an experiment where we access a number of addresses, between 1 and 17, in a loop iterating $1,000,000$ times and accessing a cache set with $W = 8$, as seen in Figure 10. Miss rates are very low when the number of addresses does not exceed the space in the set. However, the miss rate increases abruptly (4 orders of magnitude) when 9 or more contending addresses are accessed.

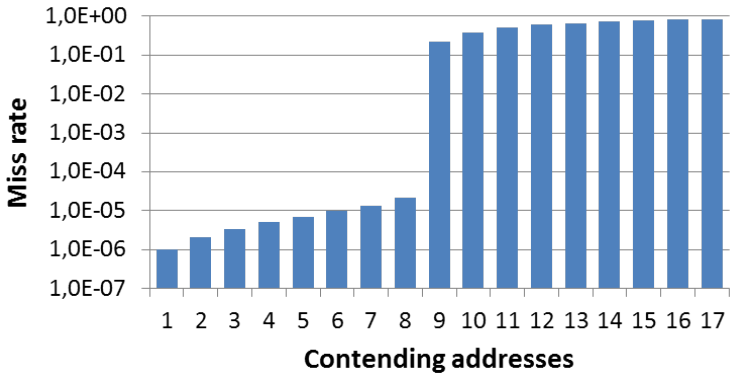


Figure 10: Miss rates for different number of addresses accessed in a round-robin fashion competing in an 8-way cache set.

Overall, for the sanity of the MBPTA results, it is crucial to determine whether execution times resulting from $W+1$ addresses competing for the same cache set can occur with a sufficiently high probability to be relevant and, in that case, make sure they are included in the observations, which defines our event of interest for the cache. In HoG authors provided an approximate formula to derive P_{eoi} on hTRc. However, its inaccuracy can be significant in some cases, since their estimation may say the analysis runs are representative when in reality they are not, so we provide means to compute the exact value of P_{eoi} in hTRc, as well as in software Time Randomised caches, where no value has been previously derived.

Let U be the number of addresses accessed by the program under analysis. In this thesis we assume that the impact on execution time of mapping any arbitrary group of K addresses to the same set – with $K \in [W+1, U]$ – is similar. This is the case, for instance, for the instruction addresses for many programs that may access them homogeneously inside a main control loop. In this case our method would require identifying those relevant addresses. In other cases where addresses are accessed heterogeneously (e.g., data accesses for control applications) a different solution would be required. Such a solution will likely require analysing program's access patterns. This problem is more complex and does not fit in the scope of this thesis, so it will be done in future work.

3.2 Relating exceedance probabilities and safety standards

Functional safety standards such as DO178B/C [36] in avionics and ISO26262 [17] in automotive relate assurance (integrity) levels with failure rates, either absolute or per hour of operation. However, software verification and testing is not explicitly related to those failure rates.

In any software verification process in the context of certification there is a qualitative step to collect "enough" evidence about software not failing during operation, where standards describe appropriate means to collect sufficient evidence for the different assurance (integrity) levels. In the context of MBPTA, pWCET estimates come along with an exceedance threshold. Such threshold upper bounds the risk of one instance of the task to overrun its assigned budget, i.e. suffering a timing violation (failure). The purpose of the exceedance threshold is not truly upper-bounding software failure rates which, in principle, are not allowed, but upper-bounding the residual risk of the software verification process.

For instance, with deterministic caches the placement of objects in memory determines which cache set each object is assigned to (e.g. based on modulo placement) resulting in a given cache layout. Conventional measurement-based practice on deterministic caches relies on the user ability to reduce the risk of not evaluating memory placements leading to bad cache layouts that produce high execution times, which can occur during operation. Such (residual) risk is only assessed qualitatively given that the user, despite making many tests, does not have a way to determine whether the

space of potential memory mappings (and the corresponding cache layouts) is truly covered. This occurs because for complex software it is hard to force a particular placement in a test run.

In the context of MBPTA and time randomised caches, the space of potential cache layouts and their impact is randomly explored: in each run, a random cache layout (mapping of objects to sets) is explored. In this way, the risk brought by unexplored placements is no longer to be controlled by the end user but it is transferred to the confidence had on the pWCET estimate obtained based on a given number of runs. As we present in this thesis, it is possible to assess the probability of a particular mapping not to be observed.

Thus, while end users need to argue qualitatively on the non-existence of unobserved placements when using non-probabilistic measurement-based methods, MBPTA allows to argue *quantitatively* on the fact that evidence shows that pWCET estimates are not exceeded with extremely high probabilities, and the residual risk (a.k.a. exceedance threshold), which can be made arbitrarily low, indicates that beyond that probability (e.g., 10^{-15} per program run) evidence is not had and so there is some residual risk of failure.

Interestingly, the certification process is the same as for conventional (non-probabilistic) practice, but replacing user's ability and unquantified residual risk by a systematic and sound approach and a quantitatively upper-bounded residual risk.

3.3 Definitions and notation to derive P_{eoi}

Basic notation	
O	Sequence of memory objects to allocate
R	Number of runs carried out by MBPTA at analysis
S	Number of sets in cache
W	Number of ways in cache
cl_b	Size in bytes of a cache line
m_b	Memory size in bytes
set_l	Number of memory lines mapped to a set

Table 1: Basic notation

In order to explain the current approximation solution (HoG) and the precise methods proposed, we introduce some basic notation. Apart from the notation in

Table 1, we also build on the following definitions:

Definition 1 (Allocation Scenario) *An allocation scenario defines how allocated objects are mapped to sets. We denote allocation scenarios as* a_i*, with* $a_i = (a_i^1, a_i^2, ..., a_i^S)$*.* a_i^j *is the number of objects allocated to set* s_j *under* a_i*.*

Definition 2 (Cardinality of an allocation scenario) *The cardinality of an allocation scenario* $|a_i|$ *is given by the number of objects allocated under it.*

Definition 3 (Maximum of an allocation scenario) *The maximum of an allocation scenario* a_i^{max}*, is given by the maximum number of objects allocated to any set in that allocation scenario.*

Definition 4 (Cache event of interest) *The cache event of interest is defined by those scenarios, called scenarios of interest, that have a set where the number of allocated objects is higher than* W*, i.e.* $a_i | a_i^{max} > W$*.*

For instance, for a 3-set 2-way cache (S=3, W=2) and a sequence with 3 single-line objects ($|\mathcal{O}| = 3$), the allocation scenarios are $\mathcal{A} = \{(0,0,3), (0,1,2), (0,2,1), (0,3,0), (1,0,2), (1,1,1), (1,2,0), (2,0,1), (2,1,0), (3,0,0)\}$. From those, the scenarios of interest are $(3,0,0), (0,3,0)$ and $(0,0,3)$.

Definition 5 (Probability of the event of interest) *The probability of the event of interest (*P_{eoi}*) is given by the addition of the probabilities for the allocation of scenarios of interest.*

Definition 6 (Path of allocation scenarios leading to a_i**)** *Given an allocation scenario* a_i*, each of the successions of allocations in the probability tree leading to* a_i *is called path to* a_i*. Each path leading to* a_i *is represented as* $pth(a_i) \in PTH(a_i)$*, where* $PTH(a_i)$ *is the set of all paths leading to* a_i*.*

For instance, in the example in Figure 11 the path leading to $a = (2,0,0)$ is $PTH(2,0,0) = \{(0,0,0), (1,0,0), (2,0,0)\}$

3.4 Heart of Gold

Determining whether at least $W+1$ objects out of the $|\mathcal{O}|$ under consideration are mapped into the same cache set requires deriving all potential mappings of the $|\mathcal{O}|$ objects into the S cache sets and the fraction of those mappings in which at least one cache set has $W+1$ objects allocated.

As shown in [7] these values can be approximated in hTRc by means of *weak compositions* theory. A weak composition of an integer n is a way of writing n as the sum of a sequence of non-negative integers [16]. We are interested in all the weak compositions of $|\mathcal{O}|$ made of exactly S parts where at least one part is higher than W and the total number of weak compositions of $|\mathcal{O}|$ is exactly S sets. When no limit is put on the values of the parts we have $WComp(|\mathcal{O}|, S, -)$. If we impose that no part can have more than W objects we have $WComp(|\mathcal{O}|, S, \leq W)$. The probability of the mappings of all objects such that one part is greater than W can be approximated as:

$$P_{eoi}^{hTRc}(|\mathcal{O}|, S, W) = 1 - \frac{WComp(|\mathcal{O}|, S, \leq W)}{WComp(|\mathcal{O}|, S, -)} \tag{2}$$

The problem of this approach is that it considers that all potential allocation scenarios have the same probability. However, in reality two scenarios can have different probabilities. For instance, Figure 11 shows all possible allocation scenarios (10) resulting from allocating 3 objects in a 3-set cache. Edges represent how scenarios are allocated while nodes show each allocation scenario. The nodes in the same level have the same number of allocated objects. In this example, the event of interest for a 2-way cache is computed with weak compositions as $P_{eoi}^{hTRc}(3,3,2) = 1 - \frac{7}{10} = \frac{3}{10}$, while in reality it is $\frac{3}{27}$, as we present in Section 4.2.1.

4 Hardware time randomised single-level caches

In the previous section, we have presented an approximation of P_{eoi}^{hTRc} using the HoG method. In this section, we propose three other methods for exactly computing the P_{eoi}^{hTRc}: one based on the probability tree generation, one based on the multinomial coefficient and another based on Monte Carlo simulations. Then, we compare all the techniques (including the previously existing HoG), and perform some experiments using different cache sizes with synthetic object mixes and a real avionics application.

4.1 Behaviour

Before introducing the models for computing the P_{eoi}^{hTRc}, first we analyse the behaviour of hardware Time Randomised caches.

As previously explained, hTRc (Figure 7) map a cache line randomly to a set. The set where a cache line is mapped will be the same for the same run, but different between runs. In a certain run, the probability of a memory address to be mapped to a specific set is $\frac{1}{S}$ (Equation 3). This is because the hash function homogeneously distributes the address space into the possible S sets, so any address has the same possibility to be mapped to the any set.

$$P_{set}^{hTRc} = \frac{1}{S} \tag{3}$$

Multi-line memory objects So far we have assumed that objects occupy a single cache line. However, in reality objects can be larger than a cache line. With hTRc, each memory element of a multi-line object is randomly assigned to a cache line. That is, hTRc assigns different elements to different sets regardless of whether those addresses belong to the same object. As a result, the allocation of a multi-line object of size l lines is equivalent to the allocation of l objects of size 1 line. Hence, the allocation of n objects $i = 0..n-1$ with sizes $\{l_i\}$ is equivalent to the allocation of $sum(\{l_i\})$ objects of size 1. This is not the case for $sTRc$ as presented in the next section.

Order of objects Furthermore, as mentioned before, since in hTRc the probability of an object to be placed in a given set $(1/S)$ does not depend on past object allocations, the order of the object allocations is irrelevant. Since multi-line memory objects are equivalent to multiple single line objects, the permutation of multi-line objects does not affect the outcome either.

4.2 Models

We propose three different exact models for computing P_{eoi}^{hTRc}: probability tree, multinomial coefficient and Monte Carlo method.

4.2.1 Probability tree

In order to compute the P_{eoi}, we first need to know the possible allocation scenarios which its corresponding probabilities of happening, and then select the allocation scenarios of interest and add their probabilities. A simple method for generating all the possible allocation scenarios with its probabilities is generate a probability tree with all the possibilities.

In Figure 11 we can see a graphical representation of this probability tree supposing a cache with 3 sets and allocating 3 objects of size 1 for hTRc. Each node represents a cache allocation scenario, with each vertical level containing all the possible allocation scenarios with a different number of objects allocated. For instance, in the first level at the top we only find one allocation scenario, the empty cache, since no objects have yet been allocated. In the second level just below the previous one, we find the three possible allocation scenarios when allocating one object in cache: $(1,0,0),(0,1,0),(0,0,1)$. The probability that each scenario has in this level is computed using as an input the probability the previous state of the cache had. We repeat this process until all the objects have been allocated, and we reach the final state: in this example the forth level with 10 possible allocation scenarios. Once in this level, we select the allocation scenarios of interest $(max(a_i) > W)$ and we add their probabilities to derive the P_{eoi}. In this example, supposing a cache with $W = 2$, the allocation scenarios of interest would be $(3,0,0),(0,3,0),(0,0,3)$.

As previously explained, the probability that a specific object will be mapped in a

given set is the same for all the sets, regardless of past object allocations. This means that given a specific allocation scenario, when allocating an object all the resulting allocation scenarios will have the same probability $P_{a_i+1} = \frac{P_{a_i}}{S}$. For instance in the example of Figure 11, given the first empty allocation scenario with $P_{a_0} = 1$, the three allocation scenarios generated from it will have the same probability $P_{a_1} = \frac{P_{a_0}}{S} = \frac{1}{3}$.

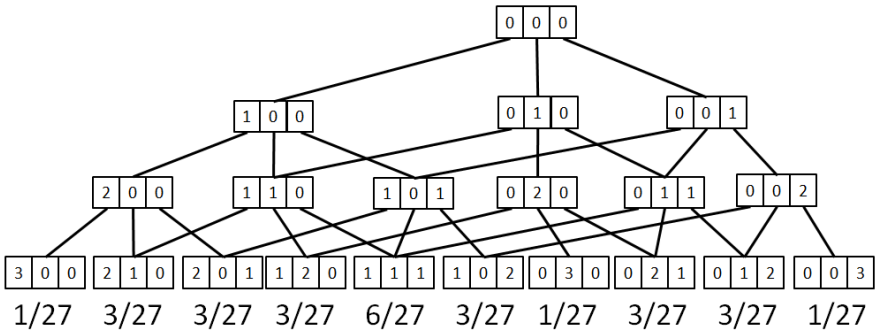


Figure 11: Probability tree for hardware randomisation with 3 objects allocated in a 3-set cache.

Although this technique gives exact results, it requires the whole probability tree to be build. The probability tree grows with the number of sets S and objects $\mathcal{O}$ to allocate (with a complexity of $O(S^{\mathcal{O}})$), having both an impact in the processor load and the memory space. Hence, if we want to compute P_{eoi} using this method for a relatively large cache (8 sets) and with a large number of objects (100 objects) it will be unfeasible in reasonable time because of the explosion of the number of allocation scenarios.

4.2.2 Multinomial coefficient

With hTRc, all the paths reaching any of the allocation scenarios with the same cardinality have the same probability. This is so because for every new allocated object the probability of being mapped to any set does not vary with the allocation of more objects. After all objects in $\mathcal{O}$ have been allocated (represented by the leaves in Figure 11), the cardinality of any of the potential resulting allocation scenarios a_i equals to the number of allocated objects, i.e. $|a_i| = |\mathcal{O}|$, and the probability of reaching any of them through one path is:

$$P^{hTRc}_{pth(|a_i|)} = \left(\frac{1}{S}\right)^{|a_i|} \quad \forall a_i \tag{4}$$

It is worth noting that not all allocation scenarios with the same cardinality are reached the same number of times, i.e. the number of paths leading to each scenario varies. For instance, in the example in Figure 11, from all those scenarios with cardinality three, $PTH(a_i)$ for $a_i = (3, 0, 0)$ comprises 1 path, while $PTH(a_j)$ for $a_j = (1, 1, 1)$ comprises 6 paths.

We approach the problem of computing the number of times a specific allocation scenario can be reached – which determines the number of paths leading to it – using the combinatorial interpretation of the multinomial coefficient [37]. The multinomial coefficient defines the possible combinations of distributing n elements over k containers, each containing exactly $y = (y^1, y^2, ..., y^k)$ elements. By replacing n by $|\mathcal{O}|$, k by S and y^j by a_i^j, the multinomial coefficient provides the number of times a_i can be reached, i.e. the number of paths from the original empty allocation scenario leading to a_i, which we call Np_{ai}. Np_{ai} is computed as follows:

$$Np_{ai} = \left(\begin{array}{c} n \\ a_i \end{array} \right) = \frac{n!}{a_i^1! a_i^2! \cdots a_i^S!} \tag{5}$$

For instance, in the example in Figure 11 (3-sets cache and 3 objects) the number of times that the possible combinations give the outcome $a_i = (1, 1, 1)$ is given by $Np_{a_i} = \binom{3}{1,1,1} = 6$.

We derive the probability of a specific allocation scenario by multiplying the number of paths leading to it by the probability of a single path:

$$P^{hTRc}_{a_i} = Np_{ai} \times P^{hTRc}_{pth(|a_i|)} = \left(\begin{array}{c} |a_i| \\ a_i \end{array} \right) \cdot \left(\frac{1}{S}\right)^{|a_i|} \tag{6}$$

In the previous example, $P^{hTRc}_{a_i}$ for $a_i = (1, 1, 1)$ is $6 \cdot \left(\frac{1}{3}\right)^3 = \frac{6}{27}$.

Overall, P^{hTRc}_{eoi} is computed by adding $P^{hTRc}_{a_i}$ for those allocation scenarios of interest $a_i | max(a_i) > W$.

$$P_{eoi}^{hTRc}(|\mathcal{O}|, S, W) = \sum_{\forall a_i | max(a_i) > W} P_{a_i}^{hTRc} \tag{7}$$

For instance in the example in Figure 11 (3-sets cache and 3 objects) and assuming that the cache has $W = 2$ ways, the allocation scenarios where the event of interest occurs are: $a_i = (3, 0, 0)$, $a_j = (0, 3, 0)$ and $a_k = (0, 0, 3)$. Adding the probabilities of each one of this scenarios gives the total probability $P_{eoi}^{hTRc} = \frac{1}{27} + \frac{1}{27} + \frac{1}{27} = \frac{3}{27}$.

When implementing this method, there are two main computationally intensive tasks. One is enumerating all the possible scenarios and selecting the ones that are scenarios of interest. The other is computing the probability of each of those scenarios. The probability computing task can be made efficient by using memoisation with the factorial values 0 to $|\mathcal{O}|$, which is the most expensive computation. However, the enumeration of all the scenarios can be significantly time consuming when the number of sets and objects increases. Although it allows the computation of a bigger number of sets and objects than the previous method did (probability tree generation), it still has limitations on certain cases with big caches and lots of objects to allocate.

4.2.3 Monte Carlo simulations

The two previous presented methods for the exact computation of the P_{eoi}^{hTRc} (probability tree generation and multinomial coefficient) generated precise although not scalable results. In order to find a scalable solution, we use a method based on Monte Carlo simulations.

The main approach we follow to derive P_{eoi}^{hTRc} is based on Monte Carlo simulations using an algorithm that processes the object sequence several times allocating each object in a random location in each simulation. Each run in which objects are randomly allocated to memory addresses (and hence sets) represents a Monte Carlo simulation. This allows deriving the probability of each allocation scenario and hence P_{eoi}^{hTRc}, for both single and multi-line objects.

While the Monte Carlo method does not provide an exact result, we can statistically derive the confidence and accuracy of the result depending on the number of simulations performed. In particular, for each estimate obtained we compute the

confidence interval ($value \pm variation$) assuming a certain degree of confidence. Determining this narrow confidence interval allows us making worst case assumptions, with little impact. That is, if any value in the range determined by the confidence interval is in $r2$ we assume that the value is not safe, and we compute the number of extra runs needed to guarantee that all values in the confidence interval fall in $r1$ with a sufficiently high probability.

Confidence	**10,000 runs**	**1M runs**	**100M runs**
0.99	0.013	0.0013	0.00013
$1 - 10^{-9}$	0.030	0.0030	0.00030

Table 2: Confidence interval width.

As an example, for an arbitrary cache event whose actual probability is $P_{eoi} = 0.5172$, Table 2 shows the width of the confidence interval for different confidence values (0.99 and $1 - 10^{-9}$) as we increase the number of Monte Carlo simulations. We observe that for $1 - 10^{-9}$ with $10,000$ runs is 0.030, which is $\pm 0.058\%$.

Based on this analysis, we perform 100 million simulations that require less than 15 minutes in a regular laptop and provide narrow confidence intervals.

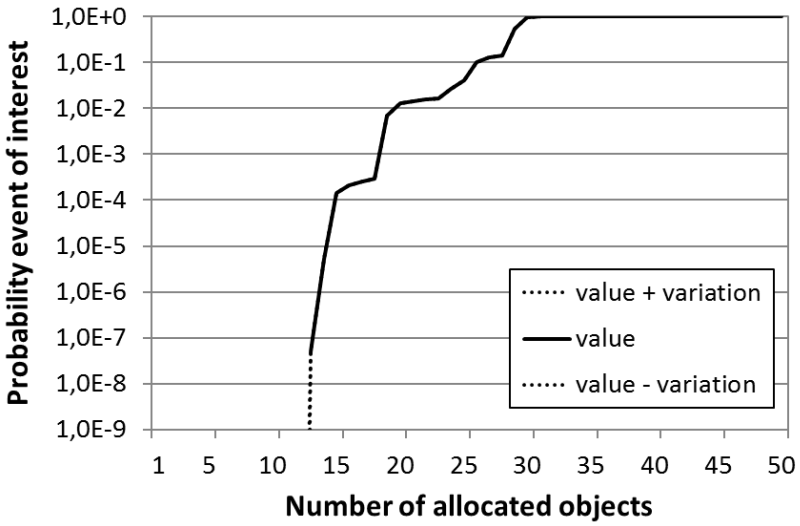


Figure 12: P_{eoi} confidence interval for the Monte Carlo simulations.

In Figure 12 we can see the P_{eoi} range given by the Monte Carlo simulations. For this example the confidence level is set to $1 - 10^{-9}$ and we have made 100 million

simulations. Since the P_{eoi} range is really small (less than 2% variation at any point), in Figure 12 we can see the three lines defining the interval overlapping graphically.

4.3 Increasing the number of runs

Given a sequence of objects $\mathcal{O}$ and a number of runs carried out at analysis time R, the probability of the event of interest P_{eoi}^{hTRc}, as computed in Equation 7 might be below P_{obs} and above P_{exc}, shown in Figure 9 as range *r2*, which challenges MBPTA reliability. With R runs done by default by MBPTA, the lowest probability of an event such that the probability of not observing it in the R runs is below a given cutoff probability P_{coff}, is given by: $(1 - P_{eoi}^{hTRc})^R \leq P_{coff}$. If we work out R we obtain $R \leq log_{(1-Peoi^{hTRc})} P_{coff}$ hence $R \leq \frac{log(P_{coff})}{log(1-P_{eoi}^{hTRc})}$. The minimum number of runs R' is:

$$R' = \frac{log(P_{coff})}{log(1 - P_{eoi}^{hTRc})} \tag{8}$$

With R' it can be guaranteed that for the specified level of confidence P_{coff} the event of interest, whose probability is P_{eoi}^{hTRc}, will be observed at analysis time.

4.4 Experimental setup

We evaluate our empirical model to obtain P_{eoi} and we compare it against the theoretical model. For that purpose we use i) synthetic object sequences and ii) object sequences coming from a real avionics case study [39].

	Small Cache	**Big Cache**
Size	2KB	128KB
Ways	4	8
Sets	8	256
Line size	64B	64

Table 3: Small and big cache configurations.

In our experiments we use two cache sizes, a small one (*sCache*) and a big one (*bCache*). *sCache* has been chosen to be particularly small to deploy all the proposed models for comparison purposes. As the cache size and/or the number of objects

increases, the probability tree generation and the multinomial coefficient models become intractable as explained before.

The *sCache* is a 2KB 4-way 64B/line cache (so with 8 sets), whereas the *bCache* size is 128KB 8-way 64B/line cache (so with 256 sets). These values are representative of first and second-level caches. For instance, caches of the Cobham Gaisler NGMP [4] or the ARM Cortex A7 [1] are within this range.

4.4.1 Comparison

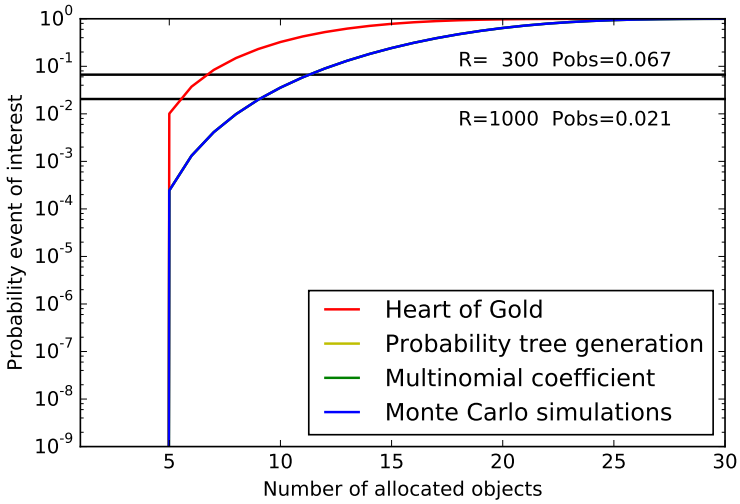


Figure 13: Comparison of all the methods proposed to compute the P_{eoi}^{hTRc}.

In this experiment we focus on a *sCache* setup and single-line objects. Figure 13 shows how the probability of the cache *eoi* varies as more objects are allocated. We implement all 4 techniques: the previously proposed approximation (HoG), and our 3 exact methods (probability tree generation, multinomial coefficient and Monte Carlo simulations). In Figure 13 we represent with horizontal lines P_{obs} for two different numbers of runs: $R = 300$ and $R = 1,000$. The former is the lowest number of runs required by the method [12] and the latter is a typical value used for many programs [12]. In the figure we can see two clearly different lines: a red one and a blue one. The red one represents the Heart of Gold approximation, while the blue one represents the three other precise techniques.

With the HoG method the probability of the eoi is higher than the exact one

obtained with the precise methods for a range of object counts. This can lead to pessimistic/optimistic results in terms of the number of runs to carry out:

1. $P_{eoi}^{hTRc} \in r3$ and $P_{eoi-HoG}^{hTRc} \in r3$: In this case both approaches conclude that the probability of the *eoi* is irrelevant.

2. $P_{eoi}^{hTRc} \in r3$ and $P_{eoi-HoG}^{hTRc} \in r2$: In reality the probability of the *eoi* is irrelevant and with HoG the user may be required to increase the number of runs.

3. $P_{eoi}^{hTRc} \in r2$ and $P_{eoi-HoG}^{hTRc} \in r2$: The user is asked to carry out, as a result of applying HoG, fewer experiments than required to ensure that the cache *eoi* is captured, since $P_{eoi-HoG}^{hTRc} > P_{eoi}^{hTRc}$.

4. $P_{eoi}^{hTRc} \in r2$ and $P_{eoi-HoG}^{hTRc} \in r1$: As in the previous case.

5. $P_{eoi}^{hTRc} \in r1$ and $P_{eoi-HoG}^{hTRc} \in r1$: Both approaches indicate that the probability of the *eoi* is captured with the runs carried out by the user.

Under cases 3) and 4) MBPTA may lead to optimistic pWCET estimates if it is applied with the Heart of Gold approach since the user is requested to perform fewer runs than needed due to an overestimated P_{eoi}^{hTRc}. For instance, when 8 objects are allocated, the HoG estimate puts the $P_{eoi-HoG}^{hTRc}$ on the safe region $r1$, while in reality P_{eoi}^{hTRc} is found in the unsafe region $r2$.

We observe no noticeable difference between the exact three methods, thus providing evidence of the accuracy of our analytical and empirical methods. As expected, the higher the number of allocated objects the higher the probability of the *eoi*. Interestingly for sequences smaller than 5 objects the probability of the cache *eoi* is below 10^{-9} that we use as the reference exceedance probability, P_{exc}. Meanwhile, when 12 objects are allocated the probability of the *eoi* is above P_{exc}. Despite the probability tree generation and multinomial coefficient methods provide exact results, its memory and execution time requirements prevent using it in the general case. We rather use it in this small example to show the accuracy of the Monte Carlo method.

4.4.2 Synthetic benchmarks

We consider randomly-generated sequences of objects of two types regarding their size. Small objects whose size is in the range $[8B, 16B, 32B, 64B, 128B]$ and big

objects whose size ranges $[256B, 1KB, 2KB, 8KB]$. By mixing objects of these two types we generate 3 types of object sequences. The *smallSeq* comprises 75% of small objects and 25% big objects, the *balSeq* comprises 50% of small objects and 50% big objects and the *bigSeq* comprises 75% of big objects and the rest of small objects. All object sequences have a size of $|\mathcal{O}| = 50$.

Small objects	32B, 64B, 128B, 256B, 512B
Big objects	1KB, 2KB, 4KB, 8KB

Table 4: Sizes for big and small objects

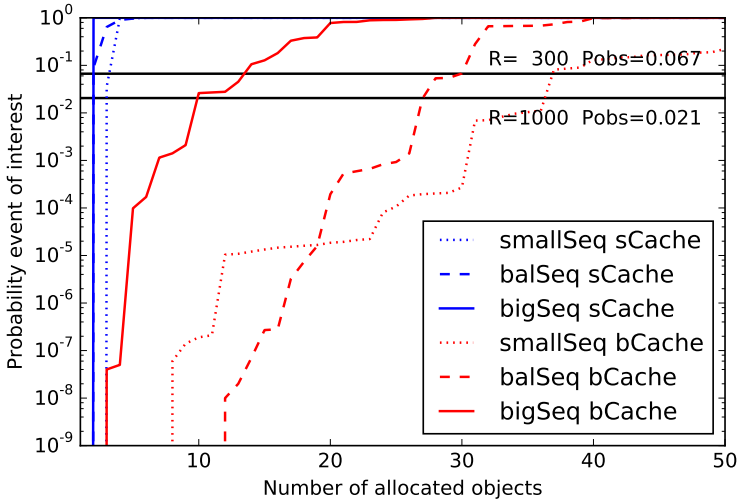


Figure 14: Results for different synthetic object sequences.

In Figure 14 we use the different object mixes previously described and show P_{eoi}^{hTRc} (obtained with the Monte Carlo simulations method) in each case as we increase the number of objects allocated. For the *scache*, with balSeq, when allocating 1 to 3 objects the P_{eoi}^{hTRc} is in range $r3$, not shown in the figure because it is below $P_{exc} = 10^{-9}$. If the number of objects allocated is within 2 and 4, P_{eoi}^{hTRc} is in the range $r2$, which can lead to a relevant event being missed in the analysis runs. When more than 5 objects are allocated, P_{eoi}^{hTRc} is in $r1$ so it is observable. Similar trends are observed for the rest of the mixes, with the rule of thumb that the higher the allocated object count, the higher the cache occupancy, and hence the higher P_{eoi}^{hTRc} is and the faster it converges to $r1$.

Number of runs. In Figure 14 we observe that with *bCache*, when less than 30 objects are allocated P_{eoi}^{hTRc} is in $r2$. In this scenario, in order to ensure that P_{eoi}^{hTRc} lies in $r1$, we use Equation 8 to increase R. For instance, if we allocate 27 objects in the balSeq we have that $P_{eoi}^{hTRc} = 0.0111685$ and $R' = 1,846$ runs would be needed to make sure that the event of interest is captured (i.e. lies in $r1$).

4.4.3 Avionics benchmarks

We applied our technique to an industrial-size avionics application [39] consisting of around 5,000 functions varying from few bytes to 300KB each. The total size of those functions is 4.7MB if they are enforced to be aligned with cache line boundaries. In this experiment the focus is on code randomization, i.e. on the instruction cache. With instruction hTRc the cache set assigned to instructions is randomised across runs by hardware means.

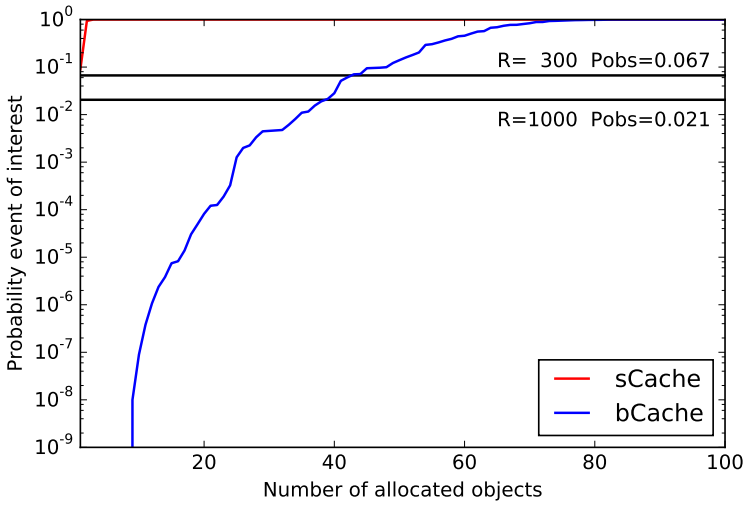


Figure 15: Results for the avionics case study.

Figure 15 shows the probability of the *eoi* for the allocation of these objects in both the *sCache* and the *bCache* setups. For the *sCache* we can see that with as low as 4 objects allocated, the probability of the *eoi* is above P_{obs}. For the *bCache* setup, we need around 40 objects to reach P_{obs}. Hence, for this particular application, the number of runs carried out $R = 1,000$ with MBPTA standard process was enough to ensure representativeness of the instruction cache events of interest.

It is also worth noting that this real case requires more object allocations than the synthetic object mixes because the average function size is smaller than the smaller objects considered in our synthetic sequences. However, even with objects this small, the number of objects that must be allocated so that the event of interest is shown in the analysis runs is largely below the total number of objects in the application.

4.5 Conclusions

In this section we have analysed the problem of computing the P_{eoi} for hTRc. We have proposed three different and precise methods for computing this probability, one of which is computationally feasible with any cache size and object sequence. Further, we have compared these precise models with the previous approximative model and we have obtained the results of computing the P_{eoi} with both synthetic benchmarks and an avionics application.

5 Software time randomised single-level caches

In this section we will propose three methods for the exact computation of the P_{eoi} for sTRc. These methods will be the same three methods already used with hTRc but adapted to the behaviour of sTRc. We also perform experiments with synthetic object mixes and a real avionics application.

5.1 Behaviour

Software Time Randomised caches (Figure 8) use a randomisation software to randomly allocate memory objects into their memory positions. On sTRc, if the memory size would be infinite, the probability of an object to be mapped to a specific set would be $\frac{1}{S}$, since the number of memory positions leading to a specific set would be the same (infinite), hence behaving like hTRc. However, in reality memory size is finite, and if an object has been allocated to a specific memory location, another object cannot be allocated there. After several objects have been mapped, the probability that an object is mapped to a set is different, since some memory addresses that mapped to some sets have already been occupied.

The number of cache lines that memory can allocate is $m_l = \frac{m_b}{cl_b}$ lines. The number of lines mapped to each set is set $l = \frac{m_l}{S}$. If a first object is mapped to memory, has the same probability to be mapped anywhere in memory, and hence the same probability to be mapped to any set. After this allocation, the number of memory lines where the second object can be allocated is $l_i = set_l - 1$, since a memory space has been occupied by the first object. Also it has a smaller probability $\frac{set_l - 1}{m_l - 1}$ of being mapped to the same set, since one memory slot that mapped to this set has already been occupied.

$$P_{set}^{sTRc}(k, l) = \frac{set_l - l}{m_l - k} \tag{9}$$

The final probability of an object being mapped to a specific set (Equation 9) depends on what sets the previous objects where allocated.

Multi-line memory objects With sTRc multi-line objects are randomized atomically, that is, the first element[1] of the object is assigned a random location in memory while the rest of the elements occupy consecutive memory positions. When modulo placement is used, multi-line objects are allocated into consecutive sets in cache.

Multi-line objects challenge the computation of whether several objects or part of them are mapped to the same set. We assume that there is always space to allocate objects in any set, that is, that the space allocated to a set is not fully consumed by allocated objects. In this scenario, the problem can be reduced to tracking the allocation scenarios and the associated probability of the first element of the object.

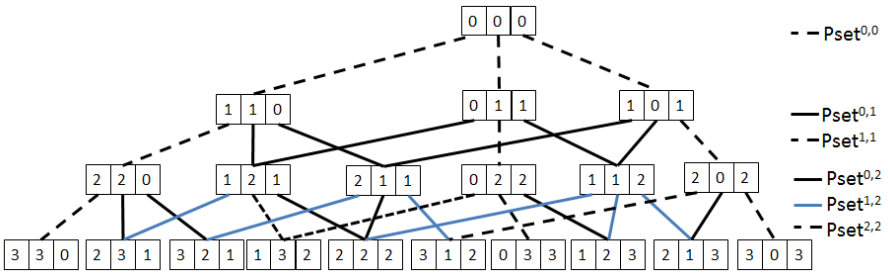


Figure 16: Probability tree for software randomisation with 3 2-line objects in a 3 set cache.

As an illustrative example, Figure 16 shows the probability tree when allocating three 2-line objects into a 3-set cache. We observe that the probability of going from an allocation scenario to another is given by the probability that the first element of the object goes to a particular line. For instance, starting from $a_i = (0, 1, 1)$ there is a probability $Pset^{0,2}$ that the second object is allocated in sets s_0, s_1, hence ending up in $a_{j1} = (1, 2, 1)$. Meanwhile there is a probability $Pset^{1,2}$ of allocating object two to s_1, s_2 and the same probability to s_2, s_0 respectively leading to $a_{j2} = (0, 2, 2)$ and $a_{j3} = (1, 1, 2)$. Note that in $Pset^{l,k}$, l counts the number of elements (lines) allocated to a given set i and k the total number of elements (lines) already allocated.

Order of objects In sTRc the order of allocation does not matter in the case of single line objects, but it does matter when allocating multiple line objects. For instance, let us assume a memory size of 30 cache lines, a 3-set cache and objects

[1]Note that an i-th element of an object is the i-th cache line it occupies.

O_1 of size 2 lines and O_2 of size 3 lines. We want to compute the probability of the allocation scenario $a_i = (2, 2, 1)$ happening.

When o_1 is allocated first, the resulting allocation scenarios are $(1, 1, 0)$ and then $(2, 2, 1)$. In this case the probability of $a_i = (2, 2, 1)$ is $\frac{10}{30} \cdot \frac{9}{28} = 0.107$. If o_2 is allocated first we have the scenarios $(1, 1, 1)$ and then $(2, 2, 1)$. In this case, the probability of a_i is $\frac{10}{30} \cdot \frac{9}{27} = 0.111$. Hence, the probability of the same allocation scenario is different depending on the order of allocation of objects. Furthermore, we observe that allocating first the big object leads to a higher probability of the allocation scenario. This happens because in the case of sTRc, allocating first the biggest objects means having less space remaining in memory, i.e. fewer memory lines available, so there is higher probability for subsequent objects to be mapped to any of them. For instance, in the previous example, when o_2 is allocated first, the denominators are 30 and 27, instead of 30 and 28, resulting in a higher probability for the same numerators.

5.2 Models

The models used for computing the P_{eoi}^{sTRc} are the probability tree, multinomial coefficient and Monte Carlo simulations.

5.2.1 Probability tree

The main idea is the same previously used with hTRc: develop the tree with all the possible object allocations on all the different sets. However, with sTRc we must take into consideration that the previous objects allocated influence the P_{eoi}^{sTRC}.

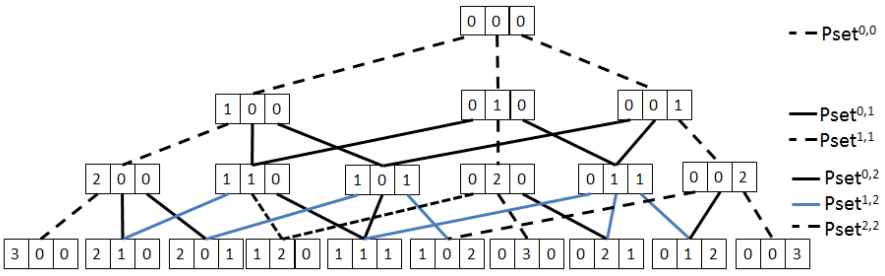


Figure 17: Probability tree for software randomisation with 3 objects allocated in a 3-set cache.

As an example, in Figure 17 we can see the probability tree when allocating 3 objects in a 3 set cache. When allocating the first object, all allocation scenarios have the same probability of happening ($\frac{1}{3}$), since no objects have been allocated. However, after the first object has been allocated, the probability that the next object will be allocated in the same set where the previous object was is smaller. This is represented in the Figure by using a different line style where the probability would be different.

As with hTRc, the probability tree generation method proposed previously does not scale with a large number of objects or sets.

5.2.2 Multinomial coefficient

With sTRc the computation of the probability of a given allocation scenario is more complex than with hTRc. This is better illustrated in Figure 17 that shows the allocation scenarios for 3 objects allocated into a 3-set cache. With hTRc, the ratio among the number of paths leading to a given a_i and the total number of paths in the tree gives the probability of that s_i. Consider the case of allocation $(3, 0, 0)$. It has a probability of occurrence of $1/27$ where 27 is the number of possible allocations.

Similar to hTRc, with sTRc we can use the multinomial coefficient to compute the number of times a specific allocation scenario occurs as well, since all possible outcomes also occur the same number of times. However, the probability of occurrence of each single scenario changes since it behaves differently from hTRc.

With sTRc each edge in the probability tree is weighed with $Pset^{k,l}$ to catch the dependence among object allocations. For a given a_i its probability of occurrence is given by the probability of all the paths leading to it, $path_i$. That is, $prob(a_i) = \sum_{j \in pth(a_i)} P_{pth_j}$. For each path, after k objects are allocated, $prob(a_s \rightarrow a_t)$ defines the probability of going from a given allocation scenario a_s (which has $k = |a_s|$ allocated objects) to another a_t after allocating object o_{k+1}. This probability is given by:

$$prob(a_s \rightarrow a_t) = Pset^{|a_s|, a_s^l} \tag{10}$$

where $|a_s| = k$ is the number of objects already allocated; s_l is the set where

the object is allocated and a_s^l is the number of objects in s_l before object o_{k+1} is allocated. For $a_i = (3,0,0)$, its probability can be derived as: $prob(3,0,0) = prob((0,0,0) \rightarrow (1,0,0)) \times prob((1,0,0) \rightarrow (2,0,0)) \times prob((2,0,0) \rightarrow (3,0,0))$ that is given by: $Pset^{0,0} \times Pset^{1,1} \times Pset^{2,2}$.

Interestingly, the probability of getting to a given allocation scenario is the same regardless of the path followed to get to it. That is, the paths, which define the order in which objects are allocated, do not impact the probability to get to a given allocation scenario.

For instance, let us assume a cache with 3 sets and a memory comprising 30 cache lines. Further let us assume that the allocation scenario we want to analyze is $(1,2,0)$. We analyze two of the paths leading to this scenario. The first path comprises the allocations $pth1 = \{(0,0,0)(1,0,0)(1,1,0)(1,2,0)\}$ and the second $pth2 = \{(0,0,0)(0,1,0)(0,2,0)(1,2,0)\}$.

The probabilities through $pth1$, computed using Equation 10, are as follows: $P_{pth1}^{sTRc} = \frac{10}{30}\frac{10}{29}\frac{9}{28}$. Each multiplicand respectively corresponds to the transition for the i allocation scenario to the $i+1$ one. Through $pth2$ the probabilities are as follows $P_{pth2}^{sTRc} = \frac{10}{30}\frac{9}{29}\frac{10}{28}$. As it can be seen, both the numerators and denominators are the same for both paths, just the multipliers appear in different order. Hence, for a given scenario, the order of allocations is irrelevant since all orders have the same occurrence probability. Based on this appreciation, we compute the probability of a given path leading to an allocation scenario a_i as shown in Equation 11.

$$P_{pth(a_i)}^{sTRc} = \frac{\prod_{i=1}^{s}\prod_{k=1}^{a_i^k}(lps - a_i^k + k)}{\prod_{j=0}^{|a_i|-1}(lpm - j)} \tag{11}$$

To better understand this equation, we use the previous example of the allocation scenario $(1,2,0)$.

The denominator is a product of the number of remaining memory addresses before each object allocation. In the example, the denominator before the first object allocation is 30 since all memory is available. Before the second object allocation, since one object has already been allocated, only 29 cache lines are available in memory.

The final product in the denominator will be $30 \cdot 29 \cdot 28 = 24,360$.

The numerator depends on the number of objects already allocated in the same set. In the example, before the first object allocation in any of the sets will be 10 since no objects have been allocated on that set. The next object allocated will be a 9 or a 10, depending if the set where this object has been allocated is the same where the previous object was allocated or not. Since the order of allocation does not matter, the final numerator for our example will be $10 \cdot 10 \cdot 9 = 900$. The final probability of occurrence of this specific scenario for one of its possible paths is $P^{sTRc}_{path(a_i)} = \frac{900}{24,360} = 0.0369$.

As for the case of *hTRc*, we can derive the probability of a specific allocation scenario by multiplying the number of paths leading to it, as shown in Equation 5 – which is the same for *hTRc* and *sTRc* – times the probability of each path, which for the case of software is as shown in Equation 11. Overall $P^{sTRc}_{a_i}$ is defined as:

$$P^{sTRc}_{a_i} = Np_{ai} \cdot P^{sTRc}_{path(|a_i|)} \tag{12}$$

Following the same example, since the allocation scenario $(1, 2, 0)$ has 3 different paths, the probability of occurrence of the allocation scenario is $P^{sTRc}_{a_i} = 3 \cdot 0.0369 = 0.1107$.

In order to calculate the probability of the events of interest occurring (a single set having more than W objects allocated), we add the probabilities of all the possible scenarios where this happens. With this P^{sTRc}_{eoi} is computed as follows:

$$P^{sTRc}_{eoi}(|\mathcal{O}|, S, W) = \sum_{\forall a_i | max(a_i) > W} P^{sTRc}_{a_i} \tag{13}$$

5.2.3 Monte Carlo simulations

As with hTRc, the multinomial coefficient method has limitations due to the enumeration of all the possible cases. Hence, we also propose a method based on Monte Carlo simulations that behaves like the one in hTRc but computing the P_{eoi} for software Time Randomised caches.

5.3 Experimental setup

We use the same setup described in Section 4.4, with a small cache and a big cache (Table 3). We have used a 20MB memory size for our experiments. Our results evidence negligible P_{eoi} variance for different memory sizes, so we do not report results for different memory sizes.

5.3.1 Synthetic benchmarks

The same synthetic object mixes generated for hTRc will be used in these experiments (small, medium and big object mixes).

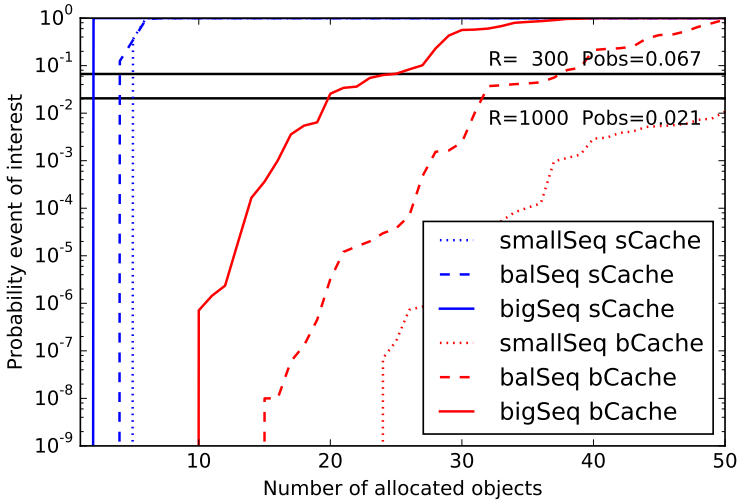


Figure 18: Results for different synthetic object sequences.

In the synthetic simulations, shown in Figure 18, we analyze the impact of cache size (*sCache* and *bCache*) and the sequence size (small, bal and big). For the *sCache* setup the probability of the eoi reaches P_{obs} for both values of R (300 and 1,000) with less than 5 allocated objects in all 3 object sequence types. This occurs because the *sCache* is small compared to the object sizes used, and with few objects the entire cache is filled. It can also be observed that, as expected, the bigger the objects are, the fewer objects are needed to reach P_{obs}. For the *bCache* setup, instead, we observe that more objects are needed in order to reach P_{obs}. Anyway, as soon as 30-60 objects are allocated MBPTA provides reliable results with its default number of runs.

Let assume the *bCache* setup, $R = 1,000$ as the number of runs under MBPTA,

an object sequence of $|\mathcal{O}| = 25$ and the sequence object *bigSeq*. For this scenario, the increased number of runs R' needed would be $R' = 1,297$.

5.3.2 Avionics benchmarks

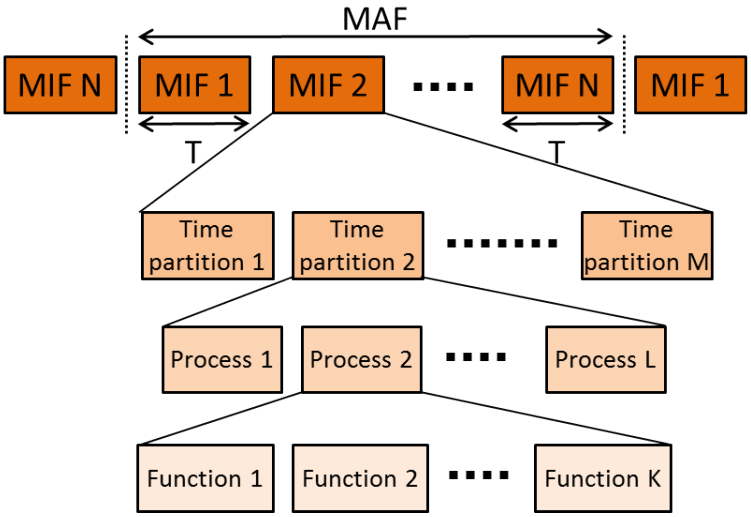


Figure 19: Scheduling of functions, processes, time partitions and MIF within a MAF.

Many avionics systems build upon the Integrated Modular Avionics (IMA) concept [8], which defines how different subsystems can be integrated onto the same hardware platform, so that size, weight and power costs can be reduced. In the context of IMA, the system architecture implements temporal and spatial partitioning to avoid undesired functional and temporal interferences across different applications – especially in the context of mixed-criticalities. Temporal partitioning is generally implemented by partitioning time into scheduling units and using a static schedule generated offline. The Major Frame (MAF) is the hyper-period of all partitions. Each MAF is divided into a number of Minor Frame (MIF) whose duration and period is identical. Time partitions are scheduled inside those MIF. Those time partitions contain processes, and each process contains a number of functions, that are the smallest unit of analysis possible. This is illustrated in Figure 19.

We analyze two different applicability approaches of our model to the avionics application. When software randomization (and so pWCET estimation) is applied (1) at MAF (full application) granularity and (2) at finer granularity i.e. at MIF, partition, process or function level.

Full application granularity

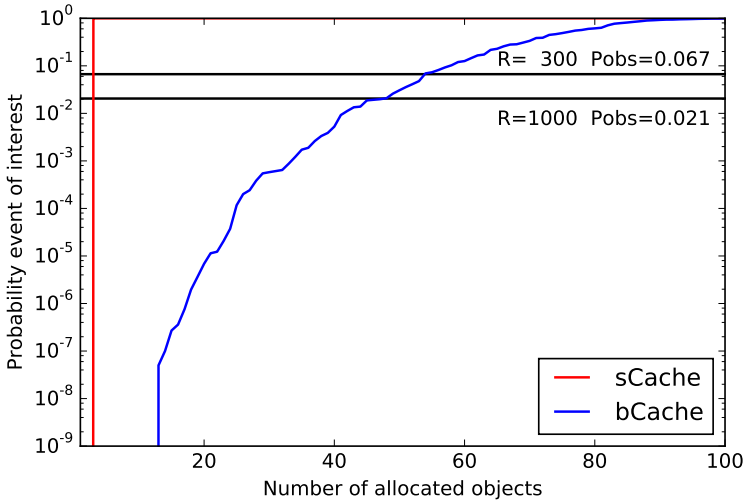


Figure 20: Results for the avionics case study.

In this case the avionics case study is analyzed with its 5,000 functions. Figure 20 shows that for the *sCache* setup with as low as 2 objects allocated, the probability of the *eoi* is higher than P_{obs}. In the *bCache* we observe that between 45 and 60 object allocations are needed to reach P_{obs}.

Analysis at fine granularity

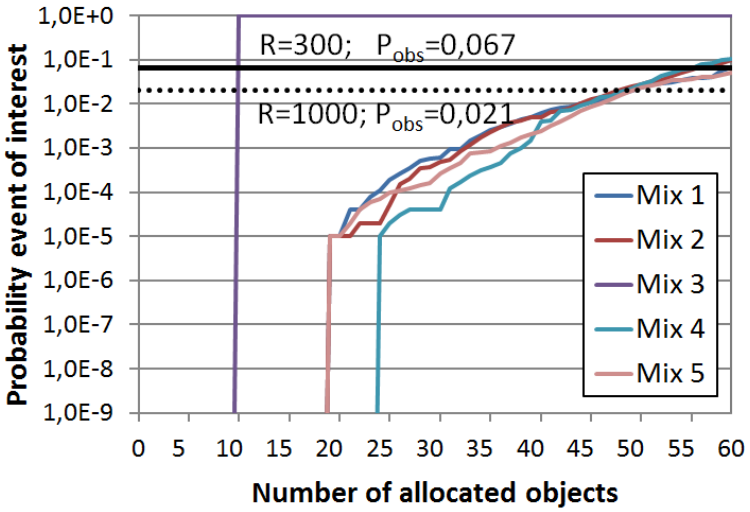


Figure 21: Results for the avionics case study for different object allocations.

Finer granularity than the full MAF may be used by end users to analyze the

timing of their applications. However, there is not a strict rule on what the right granularity is. The finer the granularity is, the higher the control exercised on the execution paths traversed, on the execution time cost of each function or process, etc. On the other hand, finer granularity also implies adding more instrumentation and exercising a stronger control on the process to collect execution time measurements. Thus, there is a tradeoff between the amount of information that can be retrieved and the cost to obtain it.

These experiments evaluate the impact of the analysis granularity on P_{eoi} to allow the users find the most convenient granularity from a software randomization point of view. For that purpose we incrementally pick functions from the case study (randomly) and compute P_{eoi} and the number of runs needed to apply MBPTA reliably (R').

First, for the sake of illustration we have sorted the 5,000 functions randomly 100 times. Then, we have obtained P_{eoi} as objects are allocated. The value of P_{eoi} for the first 5 sortings is shown in Figure 21. In general, although some differences exist depending on the objects that form the unit of analysis, we observe that all sortings reach $r1$ (so $P_{eoi} \geq P_{obs}$) with around 50 objects.

Functions	10	20	30	40	50	60
P_{eoi}	0.0000	0.0100	0.0105	0.0165	0.0549	0.1606
Range	$r3$	$r2$	$r2$	$r2$	$r2$	$r1$
R'	-	2,061	1,959	1,243	367	-

Table 5: Impact of the number of objects on P_{eoi}.

In Table 5 we summarize the average value of P_{eoi} across all 100 sortings, the range (i.e. r_1, r_2 or r_3) where that probability belongs to, and the number of runs R' that would be needed for different object counts if P_{eoi} falls in $r2$. If we assume that, for instance, $P_{exc} = 10^{-9}$ and $R = 300$, P_{obs} will be around 0.067.

As shown in the table, P_{eoi} is virtually 0 with 10 objects, meaning that the probability of the *eoi* is so low that is not relevant ($r3$). Then, from $R = 20$ to $R = 50$, P_{eoi} falls in $r2$, thus meaning that the number of runs R' needs to be higher than 300. In this example R' must be between $2,061$ (20 objects) and 367 (50 objects). Finally, P_{eoi} falls in $r1$ with 60 objects or more, so 300 runs suffice to guarantee a reliable application of MBPTA.

5.4 Conclusions

As we did in the previous section with hTRc, we have proposed three techniques to exactly compute the P_{eoi} with sTRc. These techniques have also been evaluated with the synthetic benchmarks and the real avionics application, but with the avionics application we have distinguished between two different granularities: MAF and MIF.

6 Related work

In this Section we present the related work both with Time Randomised caches as well as Measurement-Based Probabilistic Timing Analysis.

Time Randomised caches have been object of intense study in the last years. Their level of maturity has raised until they have been already prototyped in FPGA [19]. Several studies already compare the performance of measurement-based timing analysis on top of TRc and static timing analysis [6, 40], on top of time-deterministic caches. In terms of WCET, results show that MBPTA provides competitive results with respect to static timing analysis [28].

MBPTA also presents the advantage of being a measurement-based approach that can be faster adapted to new processors (systems) [40]. In terms of average performance, TRc have slight worse behaviour than deterministic caches, 12% on average [28]. Although this is not the focus of this thesis it is worth mentioning several works on Static Probabilistic Timing Analysis (SPTA) for TRc [23, 24]. In general, SPTA is in a much more immature state than MBPTA that has been evaluated with avionics and automotive case studies [27, 38, 39]. Further, an important difference with respect to SPTA is that, while SPTA requires deriving or upper-bounding the hit/miss probability of every cache access, MBPTA only requires that the probability exists.

A set of techniques in the literature shows how MBPTA handles control flow dependences and data dependences [28]. For control flow dependences, the techniques in [25, 41] show how MBPTA can be adapted to provide a pWCET estimate that upper bounds the execution time of all the execution paths of the program, even when the user-provided input vectors only exercise a subset of the paths.

Although EVT can be used with time-deterministic architectures [31], there is no guarantee that EVT captures the representative events that can occur at operation. This is possible with MBPTA [10] and is the object of this work.

Several academic works have identified the issue of representativeness of the event of interest [7, 32, 35]. Although some authors indicated that representativeness can be a risk [35], solutions have been provided later to mitigate the risk [7, 32] for TRc. For instance, HoG [7] – explained in Section 3.4 – is the first approach to solve this

representativeness issue in TRc. However, HoG relies on an approximate method to obtain P_{eoi} and so the number of runs needed, thus not removing the risk completely. Our approach solves this issue by providing an exact method to obtain P_{eoi} and so the minimum number of runs to use MBPTA.

7 Conclusions and future work

In this section we present the conclusions and the future work derived from this thesis.

7.1 Conclusions

Time Randomised caches are proposed to increase the performance of Critical Real-Time Embedded Systems. However, the use of TRc in Measurement-Based Probabilistic Timing Analysis is challenged by the fact that the execution time observations used for the prediction are those obtained at analysis time, while the predicted pWCET estimate must provide a trustworthy upper bound of deployment time. Therefore, evidence is required proving that execution time observations obtained at analysis time capture the impact of relevant events affecting execution time and that can arise at deployment.

Given the objects to be allocated for an application, we have proposed several methods to compute the probability of observing cache related events of interest in the analysis runs, for both hardware and software TRc. Whenever that probability is high enough to be relevant and low enough so that high confidence on observing the event cannot be had, our method computes the extra number of runs needed to regain confidence on pWCET estimates provided by MBPTA.

All the proposed techniques have been evaluated using both synthetic object mixes and a real avionics application. The presented results allow the user to know if the analysis runs will be representative of those at operation.

7.2 Future work

Although current Critical Real-Time Embedded Systems use a single level of cache memory or no cache memory at all, in future designs they may include multi-level cache hierarchies in order to further improve performance. If that is the case, an interesting work would be the computation of P_{eoi} for multi-level caches.

In this work we have supposed that all the memory objects are accessed the same number of times and in the same order. The methods presented in this thesis can be used in early design stages, when we still do not know the access patterns of the software. In later stages, more complex and time consuming techniques must be used

to have intro consideration the access pattern of the objects. These techniques are left as part of the future work.

Also, this work focuses on TRc as a high-performance hardware used in real-time systems to increase performance. As part of out future work, we intend to bring other high-performance architecture features to the real-time systems domain, such as prefetching.

Appendices

A Published work

Based on the work done for this master thesis, the following papers and posters have been published:

- **Modeling the Confidence of Timing Analysis for Time Randomised Caches**
  P. Benedicte, L. Kosmidis, E. Quiñones, J. Abella and F. J. Cazorla
  11th IEEE International Symposium on Industrial Embedded Systems
  Krakow (Poland), May 23-25 2016

- **Poster: On the Analysis of the Confidence on WCET Estimates for Software Randomized Caches**
  P. Benedicte, L. Kosmidis, E. Quiñones, J. Abella and F. J. Cazorla
  53rd Design Automation Conference
  Austin (United States of America), June 5-9

- **A Confidence Assessment of WCET Extimates for Software Time Randomized Caches**
  P. Benedicte, L. Kosmidis, E. Quiñones, J. Abella and F. J. Cazorla
  14th IEEE International Conference on Industrial Informatics
  Poitiers (France), July 18-21 2016

Glossary

ADAS Advanced Driver Assistance Systems. 9

BCET Best-Case Execution Time. 15

CFD Cumulative Distribution Function. 17

COTS Commercial off-the-shelf. 24

CRTES Critical Real-Time Embedded Systems. 8–10, 60

eoi Event of Interest. 26, 27, 41, 52

EVT Extreme Value Theory. 17, 26, 27, 58

HoG Heart of Gold. 28, 29, 32, 34, 41, 42, 58

hTRc hardware Time Randomised caches. 24, 28–30, 33–36, 44–46, 48, 49, 51, 52, 56

i.i.d. independent and identically distributed. 17

IMA Integrated Modular Avionics. 53

LRU Least Recently Used. 23

MAF Major Frame. 53, 54, 56

MBPTA Measurement-Based Probabilistic Timing Analysis. 3, 10, 11, 14, 16, 17, 24, 26–31, 40, 52, 58–60

MBTA Measurement-Based Timing Analysis. 9, 10, 14, 16

MIF Minor Frame. 53, 56

NRU Non Recently Used. 23

PDF Probability Distribution Function. 17

PTA Probabilistic Timing Analysis. 3, 23, 24

pWCET probabilistic Worst-Case Execution Time. 17, 18, 24, 26, 28–31, 53, 58, 60

RTES Real-Time Embedded Systems. 8

SPTA Static Probabilistic Timing Analysis. 58

STA Static Timing Analysis. 9, 10, 14, 16

sTRc software Time Randomised caches. 24, 30, 46–49, 51, 56

TRc Time Randomised caches. 3, 24, 26, 29, 58, 60, 61

WCET Worst-Case Execution Time. 3, 8–11, 15–17, 58

Bibliography

[1] ARM Cortex-A7 Processor Specification. http://www.arm.com/products/processors/cortex-a/cortex-a7.php. Accessed: 2016-05-01.

[2] Automotive Industry Drives Chip Demand. http://www.eetimes.com/document.asp?doc_id=1324718. Accessed: 2016-05-01.

[3] LZMA Benchmark Intel Haswell. http://http://www.7-cpu.com/cpu/Haswell.html. Accessed: 2016-05-01.

[4] Quad Core LEON4 SPARC V8 Processor - LEON4-NGMP-DRAFT - Data Sheet and Users Manual. http://microelectronics.esa.int/ngmp/LEON4-NGMP-DRAFT-2-1.pdf. Accessed: 2016-05-01.

[5] J. Abella, J. Castillo, F. J. Cazorla, and M. Padilla. Extreme value theory in computer sciences: The case of embedded safety-critical systems. In *International Conference on Risk Analysis*, pages 579–586. FUNDACIÓN MAPFRE, 2015.

[6] J. Abella, C. Hernandez, E. Quiñones, F. J. Cazorla, P. R. Conmy, M. Azkarate-asakasua, J. Perez, E. Mezzetti, and T. Vardanega. WCET analysis methods: Pitfalls and challenges on their trustworthiness. In *International Symposium on Industrial Embedded Systems*, pages 1–10. IEEE, 2015.

[7] J. Abella, E. Quiñones, F. Wartel, T. Vardanega, and F. J. Cazorla. Heart of Gold: Making the Improbable Happen to Increase Confidence in MBPTA. In *Euromicro Conference on Real-Time Systems*, pages 255–265. IEEE, 2014.

[8] Inc Aeronautical Radio. *Specification 651: Design Guide for Integrated Modular Avionics*. 1997.

[9] F. J. Cazorla, E. Quiñones, T. Vardanega, L. Cucu, B. Triquet, G. Bernat, E. Berger, J. Abella, F. Wartel, M. Houston, L. Santinelli, L. Kosmidis, C. Lo, and D. Maxim. PROARTIS: Probabilistically Analysable Real-Time Systems. *ACM Transactions on Embedded Computing Systems*, 12(2):z1–25, 2013.

[10] F. J. Cazorla, T. Vardanega, E. Quiñones, and J. Abella. Upper-bounding Program Execution Time with Extreme Value Theory. In *International Workshop on Worst-Case Execution Time Analysis*, pages 61–70. OASIcs, 2013.

[11] R. Charette. *This Car Runs on Code*. IEEE Spectrum, 1st edition, 2009.

[12] L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzetti, E. Quiñones, and F. J. Cazorla. Measurement-Based Probabilistic Timing Analysis for Multi-path Programs. In *Euromicro Conference on Real-Time Systems*, pages 91–101. IEEE, 2012.

[13] M. Duranton, D. Black-Schaffer, K. De Bosschere, and J. Maebe. The HiPEAC vision for advanced computing in Horizon 2020. *HiPEAC*, 2013.

[14] W. Feller. *An Introduction to Probability Theory and Its Applications*. Probability and Mathematical Statistics. Wiley, 3rd edition, 1968.

[15] G. Fernandez, J. Abella, E. Quiñones, C. Rochange, T. Vardanega, and F. J. Cazorla. Contention in Multicore Hardware Shared Resources: Understanding of the State of the Art. In *14th International Workshop on Worst-Case Execution Time Analysis*, 2014.

[16] P. Flajolet and R. Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 1st edition, 2009.

[17] International Organization for Standardization. *ISO/DIS 26262. Road Vehicles – Functional Safety*. 2011.

[18] J. Hennessy and D. Patterson. *Computer Architecture: A Quantitative Approach*. Series in Computer Architecture and Design. Morgan Kaufmann, 5th edition, 2006.

[19] C. Hernandez, J. Abella, F. J. Cazorla, J. Andersson, and A. Gianarro. Towards Making a LEON3 Multicore Compatible with Probabilistic Timing Analysis. In *Data Systems In Aerospace Conference*, 2015.

[20] A. Jaleel, W. Hasenplaugh, M. Qureshi, J. Sebot, S. Steely, and J. Emer. Adaptive insertion policies for managing shared caches. In *Proceedings of the 17th*

International Conference on Parallel Architectures and Compilation Techniques, pages 208–219. ACM, 2008.

[21] A. Jaleel, K. Theobald, S. Steely, and J. Emer. High performance cache replacement using re-reference interval prediction (RRIP). In *International Symposium on Computer Architecture*, pages 60–71. ACM, 2010.

[22] M. Kharbutli, K. Iwrin, Y. Solihin, and J. Lee. Using Prime Numbers for Cache Indexing to Eliminate Conflict Misses. In *Proceedings of the 10th International Symposium on High Performance Computer Architecture*, pages 288–299. ACM, 2004.

[23] L. Kosmidis, J. Abella, E. Quiñones, and F. J. Cazorla. A Cache Design for Probabilistically Analysable Real-Time Systems. In *Design, Automation and Test in Europe Conference*, pages 513–518. IEEE, 2013.

[24] L. Kosmidis, J. Abella, E. Quiñones, and F. J. Cazorla. Multi-level Unified Caches for Probabilistically Time Analysable Real-Time Systems. In *Real-Time Systems Symposium*, pages 360–371. IEEE, 2013.

[25] L. Kosmidis, J. Abella, F. Wartel, E. Quiñones, A. Colin, and F. J. Cazorla. PUB: Path Upper-Bounding for Measurement-Based Probabilistic Timing Analysis. In *Euromicro Conference on Real-Time Systems*, pages 276–287. IEEE, 2014.

[26] L. Kosmidis, C. Curtsinger, E. Quiñones, J. Abella, E. Berger, and F. J. Cazorla. Probabilistic timing analysis on conventional cache designs. In *Design, Automation and Test in Europe Conference*, pages 603–606. IEEE, 2013.

[27] L. Kosmidis, E. Quiñones, J. Abella, G. Farrall, F. Wartel, and F. J. Cazorla. Containing Timing-Related Certification Cost in Automotive Systems Deploying Complex Hardware. In *Design Automation Conference*, pages 1–6. ACM, 2014.

[28] L. Kosmidis, E. Quiñones, J. Abella, T. Vardanega, I. Broster, and F. J. Cazorla. Measurement-Based Probabilistic Timing Analysis and Its Impact on Processor Architecture. In *Euromicro Conference on Digital System Design*. IEEE, 2014.

[29] S. Kotz and S. Nadarajah. *Extreme Value Distributions. Theory and Applications*. Imperial College Press, 1st edition, 2000.

[30] P. Lokuciejewski and P. Marwedel. *Worst-Case Execution Time Aware Compilation Techniques for Real-Time Systems*. Embedded Systems. Springer Netherlands, 1st edition, 2011.

[31] Y. Lu, T. Nolte, I. Bate, and L. Cucu-Grosjean. A New Way About Using Statistical Analysis of Worst-Case Execution Times. In *Euromicro Conference on Real-Time Systems*, pages 11–14. ACM, 2011.

[32] E. Mezzetti, M. Ziccardi, T. Vardanega, J. Abella, E. Quiñones, and F. J. Cazorla. Randomized Caches Can Be Pretty Useful to Hard Real-Time Systems. *Leibniz Transactions on Embedded Systems*, 2(1):1–10, 2015.

[33] E. Quiñones, E. Berger, G. Bernat, and F. J. Cazorla. Using Randomized Caches in Probabilistic Real-Time Systems. In *Euromicro Conference on Real-Time Systems*, pages 129–138. IEEE, 2009.

[34] M. Qureshi, A. Jaleel, Y. Patt, S. Steely, and J. Emer. Set-Dueling-Controlled Adaptive Insertion for High-Performance Caching. *IEEE Micro*, 28(1):91–98, 2008.

[35] J. Reineke. Randomized Caches Considered Harmful in Hard Real-Time Systems. *Leibniz Transactions on Embedded Systems*, 1(1):1–13, 2014.

[36] RTCA and EUROCAE. *DO-178B / ED-12B, Software Considerations in Airborne Systems and Equipment Certification*. 1992.

[37] C. Wagner. *Basic Combinatronics*. CreateSpace Independent Publishing Platform, 1st edition, 2014.

[38] F. Wartel, L. Kosmidis, A. Gogonel, A. Baldovin, Z. Stephenson, B. Triquet, E. Quiñones, C. Lo, E. Mezzetti, I. Broster, J. Abella, L. Cucu-Grosjean, T. Vardanega, and F. J. Cazorla. Timing Analysis of an Avionics Case Study on Complex Hardware/Software Platforms. In *Design, Automation and Test in Europe Conference*, pages 397–402. ACM, 2015.

[39] F. Wartel, L. Kosmidis, C. Lo, B. Triquet, E. Quiñones, J. Abella, A. Gogonel, A. Baldovin, E. Mezzetti, L. Cucu-Grosjean, T. Vardanega, and F. J. Cazorla. Measurement-Based Probabilistic Timing Analysis: Lessons from an Integrated-Modular Avionics Case Study. In *International Symposium on Industrial Embedded Systems*, pages 241–248. IEEE, 2013.

[40] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, S. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Stachulat, and P. Stenstrom. The Worst-Case Execution Time Problem — Overview of Methods and Survey of Tools. *ACM Transactions on Embedded Computing Systems*, 7(3):36–89, 2008.

[41] M. Ziccardi, E. Mezzetti, T. Vardanega, J. Abella, and F. J. Cazorla. EPC: Extended Path Coverage for Measurement-Based Probabilistic Timing Analysis. In *Real-Time Systems Symposium*, pages 338–349. IEEE, 2015.