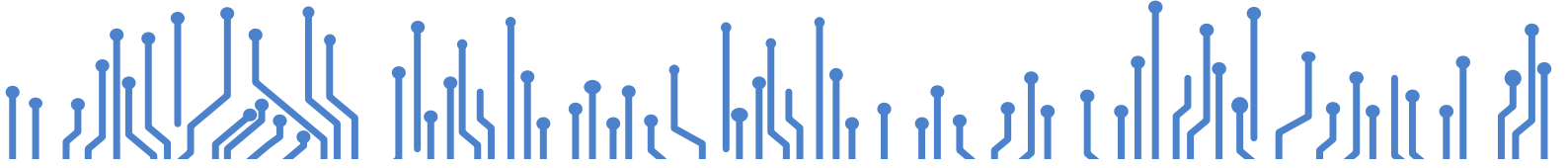


REliable Power and time-ConstraiNts-aware Predictive management of heterogeneous
Exascale systems



RECOPE

WP3 Predictive Reliability and QoS Enforcing
Methodologies

D3.2 Report on Predictive Reliability Techniques



<http://www.recipe-project.eu>



This project has received funding from the European Union's Horizon
2020 research and innovation programme under grant agreement No
801137

Grant Agreement No.: 801137
Deliverable: D3.2 Report on Predictive Reliability Techniques

Project Start Date: 01/05/2018
Coordinator: *Politecnico di Milano, Italy*

Duration: 36 months

Deliverable No:	D3.2
WP No:	3
WP Leader:	R. Canal
Due date:	31/10/2019
Delivery date:	07/11/2019

Dissemination Level:

PU	Public Use	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

DOCUMENT SUMMARY INFORMATION

Project title:	REliable Power and time-ConstraInts-aware Predictive management of heterogeneous Exascale systems
Short project name:	RECIPE
Project No:	801137
Call Identifier:	H2020-FETHPC-2017
Thematic Priority:	Future and Emerging Technologies
Type of Action:	Research and Innovation Action
Start date of the project:	01/05/2018
Duration of the project:	36 months
Project website:	http://www.recipe-project.eu

D3.2 Report on Predictive Reliability Techniques

Work Package:	WP3 Predictive Reliability and QoS Enforcing Methodologies
Deliverable number:	D3.2
Deliverable title:	Report on Predictive Reliability Techniques
Due date:	31/10/2019
Actual submission date:	07/11/2019
Editor:	J. Abella
Authors:	R. Canal, M. Fusi, F. Mazzocchetti, L. Kosmidis, F.J. Cazorla, J. Abella, M. Zapater, C. Hernandez, R. Tornero, J. Flich, G. Massari, F. Reghenzani, M. Kulcewski
Dissemination Level:	PU
No. pages:	39
Authorized (date):	07/11/2019
Responsible person:	W. Fornaciari
Status:	Final

Revision history:

Version	Date	Author	Comment
v.0.1	14/09/2019		Outline and contributions identified
v.1.0	24/10/2019		First complete version
v.2.0	7/11/2019		Final version after internal review

Quality Control:

	Who	Date
Checked by internal reviewer	R. Canal	25/10/2019
Checked by WP Leader	R. Canal	25/10/2019
Checked by Project Technical Manager	G. Agosta	5/11/2019
Checked by Project Coordinator	W. Fornaciari	7/11/2019

COPYRIGHT

©Copyright by the **RECIPE** consortium, 2018-2020.

This document contains material, which is the copyright of RECIPE consortium members and the European Commission, and may not be reproduced or copied without permission, except as mandated by the European Commission Grant Agreement no. 801137 for reviewing and dissemination purposes.

ACKNOWLEDGEMENTS

RECIPE is a project that has received funding from the European Union’s Horizon 2020 research and innovation programme under Grant Agreement No 801137. Please see <http://www.recipe-project.eu> for more information.

The partners in the project are Universitat Politècnica de València (UPV), Centro Regionale Information Communication Technology srl (CeRICT), École Polytechnique Fédérale de Lausanne (EPFL), Barcelona Supercomputing Center (BSC), Poznan Supercomputing and Networking Center (PSNC), IBT Solutions S.r.l. (IBTS), Centre Hospitalier Universitaire Vaudois (CHUV). The content of this document is the result of extensive discussions within the RECIPE ©Consortium as a whole.

DISCLAIMER

The content of the publication herein is the sole responsibility of the publishers and it does not necessarily represent the views expressed by the European Commission or its services. The information contained in this document is provided by the copyright holders ”as is” and any express or implied warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose are disclaimed. In no event shall the members of the RECIPE collaboration, including the copyright holders, or the European Commission be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of use, data, or profits; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of the information contained in this document, even if advised of the possibility of such damage.

Contents

1	Introduction	7
2	Reliability Methodology	8
2.1	General approach	8
2.1.1	OS-based models	9
2.2	CPU reliability estimation	9
2.2.1	Intel support for occupancy and usage estimation	9
2.2.2	Raw FIT estimates	10
2.3	GPU reliability estimation	10
2.3.1	NVIDIA support for occupancy and usage estimation	10
2.3.2	Raw FIT estimates	11
2.4	Reliability estimates in FPGA devices	11
2.4.1	Raw FIT estimates	11
2.4.2	Temperature measurements	12
2.4.3	Computing Resources Utilization	13
3	Timing Analysis	14
3.1	Execution Time Test Coverage Improvement for HPC	14
3.1.1	Memory-Placement Software Randomization	15
3.1.2	Code Randomization	15
3.1.3	Stack Randomization	17
3.1.4	Heap Randomization	17
3.1.5	Summary	17
3.2	Measurement-Based Probabilistic Timing Analysis for HPC	18
3.2.1	MBPTA-CV Fundamentals	18
3.2.2	MBPTA-CV Steps	19
3.2.3	Independence and Identical Distribution	20
3.3	Minimizing Network Interference	22
3.3.1	Summary	23
3.4	Related Work	23
3.5	<i>libta</i> : A C++ library for MBPTA-CV Analysis	24
3.5.1	The API	24
3.5.2	<i>libta</i> Flow	25
3.6	Conclusions and Next Steps	26
4	Thermal Modelling	28
4.1	System-on-Chip modelling	28
4.2	Server/farm modelling	30
5	RTRM Integration	32
5.1	Hardware Reliability library (<i>libhwrel</i>)	33
5.2	Timing Analysis library (<i>libta</i>)	34
6	Summary	35

1 Introduction

High-Performance Computing (HPC) systems have become ubiquitous and are no longer concentrated in supercomputing facilities and data centers. While these facilities still exist and grow, a plethora of HPC systems building on large multicores and accelerators (e.g. GPUs, FPGA-based) are nowadays deployed for a variety of applications of interest, not only for large enterprises and public institutions, but also for small and medium enterprises as well as small public and private bodies.

The proliferation of HPC systems and applications in new domains has led to new requirements related to non-functional requirements (time, power, reliability, temperature, etc) and the implementation of platforms to satisfy them [4, 23, 22, 32, 38]. In this deliverable, we provide a summary of the progress achieved in the following activities related to different QoS aspects, as part of WP3:

- Reliability methodology. Our work to predict reliability concerns, developed as part of T3.1, deals with the aging of the different hardware components of the platform to guide application deployment and resource management, leveraging aspects related to the lifetime of the platform as a whole. This work is on track.
- Timing analysis. Timing predictions are key to make an efficient use of resources while ensuring that each application is allocated enough resources to complete in time. Our work in T3.2 provides means to perform those timing predictions. This work is ahead of schedule and the method itself is already complete.
- Thermal modelling. Thermal modelling techniques to estimate chip temperature are required in order to enable the assessment of temperature-related reliability effects on chips. Our work in T3.3 provides means to simulate and model chip temperature accurately with the goal of proactively enhancing chip reliability.
- Applications characterization. This task has just started (1st of October), so there is nothing relevant to report yet. Details on this topic will be provided in D3.6 in month 30 (October 2020).
- RTRM Reliability, Timing, and Thermal Policies Development. This work is at a preliminary stage, since it spans from April 2019 to March 2021. Thus, we provide a brief summary of the (limited) work done so far.

The rest of the deliverable provides details on the progress made in each front, except on applications characterization, as discussed before.

2 Reliability Methodology

2.1 General approach

Reliability at the system level needs to take into consideration the different -heterogeneous- blocks that build it. In RECIPE, this means a system composed of a multicore CPU and several accelerators (GPU and FPGA). Overall, the system should provide information (on the reliability status of each block). For some blocks where the hardware provides more information, the system can leverage more detailed information (e.g. for the CPU, we may distinguish between the memory hierarchy and the processor core).

System reliability measurements will be done through the computation of the FIT rate (number of failures in 10^9 hours of operation). System reliability may be computed as an aggregate of multiple blocks (e.g. CPUs, caches, memories, accelerators ...). Individual -independent- FIT rates can be combined to a system by:

$$FIT = \frac{\text{number of failures}}{t_{op}}$$

$$FIT_{system} = \sum_{i=1}^N (FIT_i)$$

Similarly, the FIT rate depends on the fundamental vulnerability of the block (also known as raw FIT rate). From the raw FIT rate, there are several derating factors in the system [43]. This effectively means that any fault in the circuitry, may not be visible to the architectural state (i.e. on-chip code and values of the application being run). To take this into account, we will use the Architecture Vulnerability Factor -AVF- which measures block occupancy. In short, if there is a fault in a block we are not using (i.e. it does not hold the architectural state), then it will not affect the system (i.e. it will be masked). AVF can be factored in in the FIT rate computation as follows:

$$FIT_{system} = \sum_{i=1}^N (AVF_{block_i} * FIT_i)$$

Finally, electronic circuits degrade over time. Among other causes, temperature is a key contributor. As degradation increases the FIT rate of components, we will factor it in as an acceleration factor (A_f) of the “operation” time (HMOL model [21]). Thus:

$$top + d = top * A_f$$

and A_f is defined as:

$$A_f = e^{\frac{E_a}{k} * (\frac{1}{T_{use}} - \frac{1}{T_{stress}})}$$

where:

E_a is the activation energy (eV) of the failure mode ($E_a = 0.7eV$ for transistors)

k is the Boltzmann’s Constant ($k = 8.617x10^{-5}eV/K$)

T_{use} is use temperature (standardized at 55°C or 328°K)

T_{stress} is the stress temperature (higher than T_{use} means stress, lower means recovery) (in °K)

This model has been shown to adapt to different sources of acceleration (NBTI, HCI, TDDB, electromigration) [20, 34, 35, 44]. If so, individual FIT rates for each source should be computed and then aggregated (as seen in earlier in this section). This is a powerful model that will let us incorporate different sources of degradation into our system-wide model.

2.1.1 OS-based models

Several efforts have been spent on using system error reporting logs to keep track of system status. Thus, a fault has to manifest into a recovered error or a fatal error at the application/system level. At the system level, these errors may be caused by faulty devices (our case), program bugs/application errors (not our case) or system misconfiguration (not our case). Consequently, system error reporting is not of use for the target of system reliability estimation as the reliability information is hidden between other error causes.

Consequently, in RECIPE we propose to use detailed hardware measurements and the model described in the previous section to unequivocally predict system reliability.

2.2 CPU reliability estimation

Given the model used in RECIPE, the CPU has to provide estimates of AVF as well as temperature readings (for degradation). For AVF, we will use average block occupancy/usage to compute AVF (being 0 not used at all, and 1 used always). Any value in between 0 and 1 reflects that the block is partially occupied/used.

2.2.1 Intel support for occupancy and usage estimation

The target RECIPE platform uses Intel chips. Intel already provides several libraries that can help compute the average occupancy/usage. The following is a list of the software packages/libraries that are relevant to our work:

- Intel Resource Director Technology [11]
 - Cache Monitoring Technology (CMT)
 - Memory Bandwidth Monitoring (MBM)
 - Cache Allocation Technology (CAT)
 - Code and Data Prioritization (CDP)
 - Memory Bandwidth Allocation (MBA)
- Intel® Performance Counter Monitor [10]
 - Includes power / thermal data
 - Detailed CPU statistics

	Core	Task	CMT	MBM	L3 CAT	L3 CDP	L2 CAT	MBA
intel-cmt-cat	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Intel(R) PCM	Yes	No	Yes	Yes	No	No	No	No
Linux perf	Yes	Yes	Yes	Yes	No	No	No	No
Linux cgroup	No	Yes	No	No	Yes	No	No	No
Linux resctrl	Yes	Yes	No	No	Yes	Yes	Yes	No

Figure 1: Intel Libraries and Linux tool integration

	CMT	MBM	L3 CAT	L3 CDP	L2 CAT	MBA
Intel® Xeon® processor E5 v3	Yes	No	Yes	No	No	No
Intel® Xeon® processor D	Yes	Yes	Yes	No	No	No
Intel® Xeon® processor E3 v4	No	No	Yes	No	No	No
Intel® Xeon® processor E5 v4	Yes	Yes	Yes	Yes	No	No
Intel® Xeon® Scalable Processors	Yes	Yes	Yes	Yes	No	Yes
Intel® Atom® processor for Server C3000	No	No	No	No	Yes	No

Figure 2: Intel hardware support of monitoring libraries across generations

These software packages are directly accessible through the selected libraries. By using public libraries, we want our runtime manager to be usable in all systems. Thus, we will favor the use of public libraries and tools to extend the applicability of our results. The usage of the values given by the tools is what defines the novelty of this project.

Some of these packages are already available in Linux systems. Figure 1 describes what is available for each library and linux tool.

Finally, not all Intel chips provide the same level of hardware monitoring tools. Figure 2 lists several available Intel chips available for at each partner’s server infrastructure.

2.2.2 Raw FIT estimates

RAW FIT rates for current technology is a very well kept secret for all companies. Yet, several technology libraries/tools are able to compute it. This estimates are used widely in literature and they allow for valid comparisons of FIT rates when using the same technology. Consequently, we will use the RAW fit rates reported in [39] as they include 14nm FINFET devices (as the CPUs under analysis).

2.3 GPU reliability estimation

Given the model used in RECIPE, the GPU has to provide estimates of AVF as well as temperature readings (for degradation). For AVF, we will use average block occupancy/usage to compute AVF (being 0 not used at all, and 1 used always). Any value in between 0 and 1 reflects that the block is partially occupied/used.

2.3.1 NVIDIA support for occupancy and usage estimation

The target RECIPE platform uses NVIDIA GPUs. NVIDIA already provides several libraries and a user-space tool (NVIDIA Data Center GPU Manager (DCGM)) that can help compute the average occupancy/usage. The following is a list of the software packages/libraries that are relevant to our work:

- NVIDIA Management Library (NVML) [12]

- nvmlDeviceGetTemperature
- nvmlDeviceGetUtilizationRates

2.3.2 Raw FIT estimates

GPUs are implemented in the same underlying technology as its CPU counterparts. Consequently, we will use the same raw fit rate as the CPUs. This approach is widely used in literature and it allows for valid comparisons of FIT rates when using the same technology.

2.4 Reliability estimates in FPGA devices

FPGA devices are silicon-made and thus, reliability estimations in these devices also follow Arrhenius model. In this subsection, we report FIT numbers as described in the FPGA datasheets and vendor tools and describe how to measure FPGA device utilization and temperature. Finally, we also discuss about the accuracy and limitations of these metrics.

2.4.1 Raw FIT estimates

The failure rate is typically defined in FIT units. One FIT equals 1 failure per 1 billion device hours. For example, 5 failures expected out of 1 million components operating for 1,000 hours have a failure rate of 5 FIT. Failure rate calculation equation is shown below:

$$FIT = \frac{x^2 * 10^{-9}}{2(Num.Devices) * (Time(hours)) * Acc.Factor} \quad (1)$$

where x^2 is the Chi-squared value at a desired confidence level and $2f+2$ degrees of freedom where f is the number of failures. The Acceleration factor is calculated using the Arrhenius relationship. Table 2.4.1 show FIT values for Xilinx devices at a different process technology nodes [47].

Process Technology	Device Hours at Tj = 125°C	FIT	Device
16nm	1,143,159	10	UltraScale+TM
20nm	1,050,575	11	UltraScale+TM
28nm	1,044,069	11	7 series FPGAs and Zynq®-7000 SoCs
40nm	1,086,054	11	XC6VxXxxx
45nm	1,100,455	11	XC5VxXxxx

Intel FPGA reliability numbers are described in [13]. Unfortunately, this document, the latest available that can be found in the web, does not provide information about the Stratix 10 device that we are using in the FPGA prototype. However, as stated in that document, FIT rate and mean time between failures (MTBF) information for devices within the same family with the same process technology will be very similar. Following this advice, we can try to extrapolate the FIT numbers for the Stratix-10 using the FIT values reported for other devices. Nevertheless, the Stratix 10 uses a 14nm process and the smallest technology node reported by Intel in the reliability report [13] is 20nm. We will monitor the Intel report for any updates.

2.4.2 Temperature measurements

The RECIPE prototype includes different FPGA devices namely an Intel Stratix 10 in a ProDesign board, a Xilinx ALVEO U280 board, and the MANGO FPGA cluster that comprises several types of Xilinx devices (KU115, Zynq Ultrascale, and Virtex V200T) included in modular ProDesign boards. Each of these FPGA platforms is able to provide real-time temperature readings but not all temperature sensor will have the same accuracy.

Xilinx ALVEO U280

To obtain temperature measurements using these devices Xilinx provides a system management IP for on-chip voltage and temperature measurements. This IP generates an HDL wrapper to configure the system monitor for user-specified internal sensor channels and alarms. We have instantiated and tested this IP for the ALVEO 280 being able to obtain consistent temperature readings at runtime.

Intel/Altera Stratix 10

The on-chip voltage sensor in the Intel Stratix 10 FPGA is an 8-bit full differential ADC that provides built-in capability for on-die temperature monitoring. The internal digital temperature sensor consists of internal temperature sensing diodes (TSD) and a built-in ADC. The user can monitor the on-die temperature through the internal digital temperature sensor in the Intel Stratix 10 but for this the Temperature Sensor Intel FPGA IP have to be instantiated in the design. The Temperature Sensor IP core returns the Celsius temperature value in signed 32-bit fixed point binary format, with eight bits below binary point. This value can be converted into decimal by using two's complement operation on the signed integer portion and adding the decimal number to the unsigned 8-bit fraction. For example, if the returned value is 0xFFFFE1C0, the temperature value is -30.25°C.

Since the Intel Stratix device is implemented in a ProDesign board, temperature reads have to occur through the mmi64 interconnect provided by Prodesign. It is important to mention that in this FPGA setup, the user is forced to implement the Stratix 10 temperature monitor IP provided with the proFPGA HDL platform in every user design. The monitoring and fault detection procedure is running in the monitoring System Controller implemented in the control FPGA and microcontrollers included in the board. Note that ProDesign boards do not only include the FPGA devices for computation but additional control logic to monitor and guarantee the reliability of the system. One second after the Stratix 10 has been configured, the monitoring starts. If the IP is not implemented in the user design or an over temperature occurs due to environmental or usage conditions, the Power supply of the Stratix 10 will be turned off immediately and the temperature error LED will be activated. This mechanism prevents any damage in the FPGA device due to overheating.

FPGA devices in the MANGO cluster

As in the case of the Stratix board, FPGA boards included in the MANGO prototype are from ProDesign. This means that temperature reads have to be performed using the mmi64 interconnect. However, it is noteworthy to mention that FPGA devices in the MANGO prototype do not include on-chip temperature monitors and thus, temperature reading from these boards come from sensors implemented in these boards.

Software API

Within the RECIPE prototype the HN_daemon provides the temperature of the different HW components of the system via API functions. This functionality is mainly implemented in a specific thread (hn_temperature_monitor) that periodically accesses the HW sensors of the different components and stores the readings to later provide this values to the applications requesting them. The HN_daemon provides the temperature of the different FPGA modules, the motherboard controller, and the motherboard clock network.

When an architecture is loaded into the HN_system, the HN_daemon generates a resource_location map to set where the resources are physically placed, so a user can directly get the HW module temperature via the API functions as well as get the temperature of the HW component where a resource is mapped with no need to know its physical location. The polling interval of the readings can be specified by the user. The API also provides a set of functions to read the current temperature of the HW components, but usage of this function is strongly discouraged since accessing the HW sensors is a high time consuming task (tens of miliseconds per sensor).

2.4.3 Computing Resources Utilization

In FPGAs, similarly to other silicon devices as GPUs and CPUs, the utilization of the device has an strong impact on its degradation. To measure utilization in FPGA devices included in the RECIPE prototype we rely on the the performance monitoring counters provided by the HN_daemon. However, the information these monitors provides is significantly influenced by the acceleration mode under which FPGAs are being utilized. In particular, as already defined in D4.1, RECIPE FPGA accelerators can be employed according to the following modes: (1) full custom RTL kernels,(2) HLS-based design implementation of certain kernels, (3) library-based design and (4) software programmable vector soft-core accelerators. For (1) and (2) utilization measurements reported by the HN_daemon will occur at the granularity of the kernel, that is considering the kernels is fully utilized or not utilized at all but not at a finer granularity. For the acceleration modes (3) and (4) utilization measurements can occur at much finer granularity since in this case computations occur in differentiated computing elements which are provided with specific counters. Finally, it is important to mention that utilization values will have to be weighted considering the area resources occupancy of the different resources. Fortunately, FPGA CAD tools report these values after each implementation.

3 Timing Analysis

Timeliness is a key non-functional requirement. Timeliness can be expressed in the form of strict real-time guarantees or quality-of-service (QoS), and is quite common in HPC applications related to big data in general and sustained input processing in particular. For instance, the result of a weather prediction or large simulation modeling the propagation of hazardous substances after an accident, needs to be completed in a given short time frame in order to be useful.

So far, timing guarantees have been mostly of interest for embedded systems with some form of criticality, such as those in avionics, automotive, space, industrial processes, etc. Therefore, technology to estimate execution time bounds already exists. However, target systems for such technology are often much simpler than those in HPC, and execution conditions are also far more controlled, with single-threaded applications statically scheduled [33]. Hence, whether such technology fits the specific requirements of HPC systems is not yet studied and suitable techniques offering sufficient scalability and flexibility need to be identified and used appropriately.

This section presents a methodology for the analysis of the timing behavior of HPC applications. Our approach builds upon techniques based on memory layout randomization for increasing test coverage [18, 24], as well as measurement-based probabilistic timing analysis (MBPTA) [17, 3] to predict rare (high) timing behavior beyond those values observed in applications with a continuous flow of (big) data. The contributions of this work can be summarized as follows:

1. Exploration of execution conditions. We adapt a software randomization layer for its integration in HPC applications to test their susceptibility to memory layouts caused by different code, heap and stack allocations.
2. Worst-Case Execution Time (WCET) analysis. We analyze and fit an MBPTA technique for WCET prediction so that it can be used in the context of HPC applications running on high-performance systems.

The rest of this section is organized as follows. Section 3.1 assesses methods to increase execution time test coverage and adapts them for the needs of the HPC domain. Section 3.2 reviews a method for probabilistic WCET estimation used in the context of critical real-time embedded systems, and tailors it for its use in HPC systems. Related work is reviewed in Section 3.4. Finally, Section 3.6 summarizes the work in this section.

3.1 Execution Time Test Coverage Improvement for HPC

Predicting how much an application can take to run requires the use of representative execution time tests during the analysis phase. However, a number of execution time conditions are hard – if at all possible – to control or even to track, so that it turns out to be virtually impossible to relate execution time measurements in the analysis phase with those that may occur once the system has been deployed.

Difficulties arise from the way the Operating System (OS) places code, stack and heap data in memory, and the impact that such allocation has on cache behavior, contention in the access to shared hardware resources (e.g. the memory controller, or shared cache buses and buffers), among other effects. Therefore, performing an arbitrarily large number of experiments during analysis does not guarantee, in general, covering execution time conditions relevant during operation since

those analysis conditions may preclude, by construction, some specific memory allocations that may lead to high execution times. Hence, the lack of controllability of those effects, which are managed in non-obvious ways by the OS, defeat any attempt to predict execution times based on test campaigns that lack means to trigger specific timing conditions. In order to address this challenge, some techniques based on (memory placement) software randomization have been proposed with the aim of enabling probabilistic coverage of the different execution time conditions [18, 24]. Those techniques randomize the physical location of code and data in memory, which indirectly randomizes their placement in cache and hence, their hit/miss behavior.

3.1.1 Memory-Placement Software Randomization

This set of techniques, which we refer to as *SWrand* for short, aims at exploring arbitrary memory placements, which ultimately determine cache behavior for both, code and data. The randomization of the memory placement can alter the hit/miss behavior and the possible contention experienced in the access to shared resources -when accessed by other applications or other threads of same application; and thus, it can affect the execution time of the application under study.

There are different incarnations of SWrand, depending on whether randomization is introduced dynamically in the different runs of a single binary, or whether memory placement is performed statically at compile-time (and thus; we need to generate multiple instances of the binary each one with a different random placement). The latter, often known as *static* SWrand [25, 26], randomizes the location of the different objects at run-time, thus removing any type of indirection during execution, or code copy to different memory locations, which is against some basic principles for critical real-time embedded systems. In our context, HPC systems are not subject to the same set of strict constraints as critical real-time embedded systems since they do not have to undergo any strict functional safety certification and hence, the most flexible incarnation of SWrand – dynamic SWrand [18, 24] – can be used instead. Such dynamic SWrand is the approach we detail next, emphasizing how it fits the proposed methodology.

3.1.2 Code Randomization

SWrand copies functions' code at random memory locations during execution with the objective of randomizing their cache placement and hence, how code conflicts in first level (L1) instruction caches with OS code, dynamic libraries and other applications, and additionally with data in caches shared amongst code and data. This is illustrated with the scheme in Figure 3. The plot in the left shows how functions f_a and f_b are allocated in memory without SWrand, and f_b calls f_a . On the right, we observe how functions are randomly placed in memory and function calls replaced by indirect calls whose pointers in the caller are replaced with the newly allocated location of the callee. We refer the interested reader to the original publication for further details [24].

There are two main ways to perform code randomization: at initialization or at call time [16]. The former performs the copy of all functions to random locations and the arrangement of all pointers for indirect calls before starting the execution of the code in the `main` function. This incurs in some overhead to copy those functions that will never be called. To mitigate this overhead, the latter approach performs a lazy allocation so that pointers for indirect calls are not set at initialization and, instead, every time a function is called, it is checked whether the

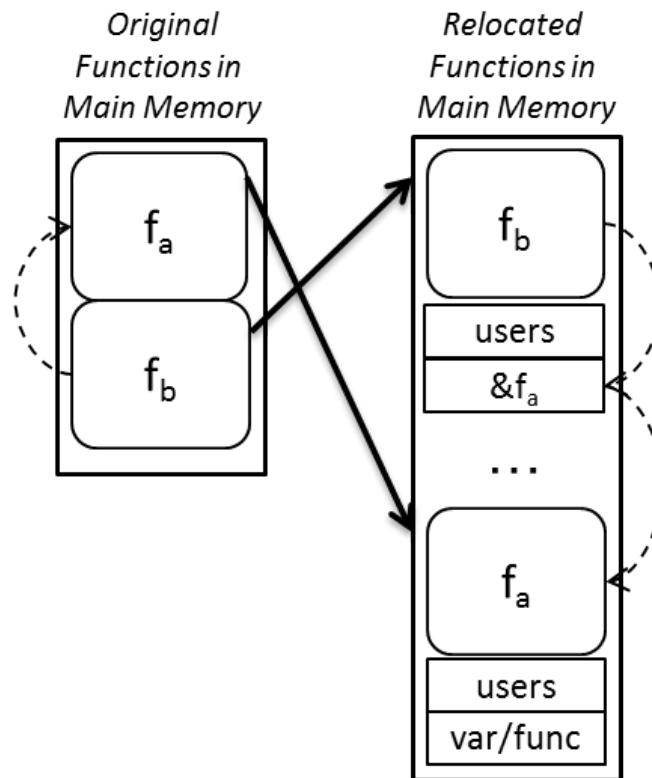


Figure 3: Schematic of code randomization.

pointer has already been initialized. If this is the case, then the function has already been called before. Otherwise, it is the first function invocation and hence, the code is copied into a random location opportunistically and the pointer is set accordingly.

Note that in both cases, some memory is allocated from the heap in a random location to copy the code in it. Such random location is not selected across the whole memory space for the sake of efficiency and, instead, it is limited to an offset of, at most, the size of one cache way of the largest cache in the system. Within such region, random placement is performed. Note that using a random allocation across a larger space does not bring further benefits since the sets where code is effectively placed in cache are obtained with the address modulo the size of the cache way.

In order to use this technique in the context of HPC applications, which may be potentially multi-threaded, we note that opportunistic allocation at call time cannot be used since one thread will set a pointer on a function call (in local memory space). Consequently, this specific memory address may not be visible by the other threads or may bring inconsistency problems needed for explicit synchronization. Instead, if code randomization is performed at initialization time before threads are spawned, we eliminate this problem by construction.

Observation 1. Code randomization must be applied at initialization time for HPC systems.

3.1.3 Stack Randomization

Dynamic stack randomization builds upon placing the function stack at a random location for each function. This can be done using indirections and allocating the stack from the heap, as in the case of code [24]. In particular, the solution builds upon having a pool of preallocated stack frames that are given to the functions called on demand. However, in the context of HPC applications, which may run multiple threads simultaneously, managing a pool of stack frames imposes the use of synchronization primitives and may decrease performance due to sharing such pool.

Static SWrand [26, 25], instead, builds upon inserting padding (e.g a random size -but constant for each binary produced dummy variable) at the beginning of the stack frame so that the location of the data used from the stack is effectively random. While such a solution imposes a fixed stack frame location for each function, and thus not random across calls, we use a dynamic variant of it, where the padding created is sized randomly on each invocation locally for each function, again limited to the size of the largest cache way in the system. Since data is kept in the local stack frame of the thread, no synchronization is needed across threads.

Observation 2. Stack randomization must be applied independently for each thread with a random shift of the stack on a function invocation for HPC systems.

3.1.4 Heap Randomization

SWrand, by default, does not consider heap randomization per se since, critical real-time embedded systems are not allowed to allocate memory dynamically. Hence, specific randomization for heap objects is not needed as a goal. However, SWrand builds upon random heap object allocation provided by Stabilizer [18], a compiler pass developed within the LLVM compiler and a runtime system based on Die-Hard[5] and Heap-Layers[6] providing random allocation of objects.

In the case of HPC applications, dynamic memory allocation is allowed. Building upon a centralized memory allocator for the heap imposes the use of synchronization. Whether such memory allocator is randomized or not is irrelevant in this respect and hence, SWrand does not bring any additional constraint and can be used for HPC applications regardless of whether they are single- or multi-threaded.

Observation 3. Heap randomization does not impose any additional constraint for HPC applications.

3.1.5 Summary

Overall, SWrand can be used for HPC applications, even if they are multi-threaded. However, specific considerations need to be taken into account for code and stack randomization, as detailed above. Once those considerations are taken into account, SWrand provides means to test any cache placement probabilistically, which calls for appropriate probabilistic means to analyze execution times obtained with SWrand-based test campaigns. The next section reviews an MBPTA method, as needed to analyze randomly sample execution time measurements, making considerations for its reliable use in the context of HPC applications.

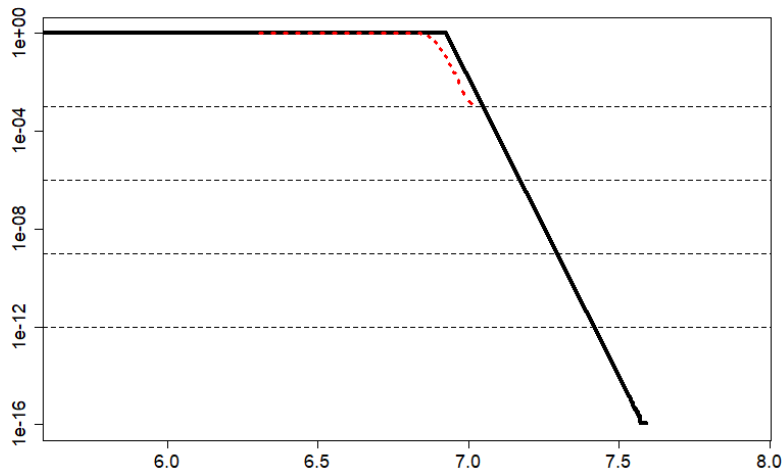


Figure 4: Example of pWCET distribution.

3.2 Measurement-Based Probabilistic Timing Analysis for HPC

A number of methods have been proposed for MBPTA [7]. Amongst those, MBPTA-CV [3] offers a detailed explanation of its implementation details as well as a publicly available implementation [1]. Therefore, we consider MBPTA-CV as the basis of our work. In this section, we review its main steps and assess their applicability in the context of HPC applications.

3.2.1 MBPTA-CV Fundamentals

MBPTA in general, and MBPTA-CV in particular, builds upon Extreme Value Theory (EVT) [9, 27], a branch of Statistics aiming at predicting rare events beyond those observed in an input sample. In the context of MBPTA, EVT is used to deliver a probabilistic WCET (pWCET) or, in other words, a distribution for high execution times so that the pWCET curve can be used to determine the probability of exceeding any particular execution time. Such pWCET curve is normally depicted as a Complementary Cumulative Distribution Function (CCDF), as the example shown in Figure 4, where the red dotted line corresponds to a sample of 1,000 execution time measurements and the black straight line the pWCET curve. The y-axis corresponds to the exceedance probability (in logarithmic scale), and the x-axis shows execution time in seconds. For instance, we observe that the probability of exceeding an execution time of 7.4 seconds is up to 10^{-12} per run.

MBPTA-CV uses the residual coefficient of variation (residual *CV*), where the *CV* for a given distribution is obtained as the ratio between its standard deviation and its mean, and the residual *CV* has been shown to determine the type of the tail of the distribution: $CV = 1$ for exponential tails, $CV \geq 1$ for heavy tails and $CV \leq 1$ for light tails [19]. It has been shown that heavy tails are not appropriate for programs with finite execution times [7], and no solution has succeeded at using light tails for pWCET estimation so far, so the only reliable type of tails applicable to any program with finite execution time generally corresponds to exponential tails.

Such residual *CV* can be estimated with the *empirical residual CV* using the observed mean and standard deviation from a sample rather than theoretical parameters, and used to produce the *CV-plot*, which estimates the *CV* for each number of exceedances of a sample. An example of a

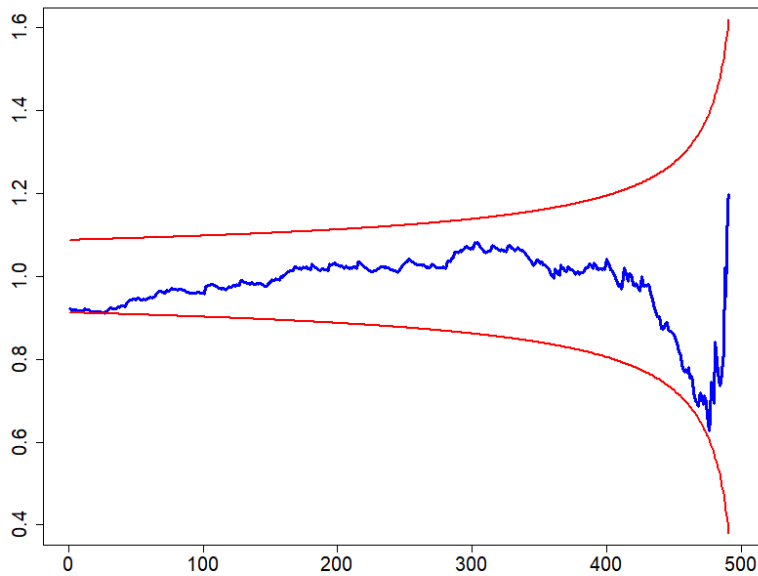


Figure 5: Example of CV-plot.

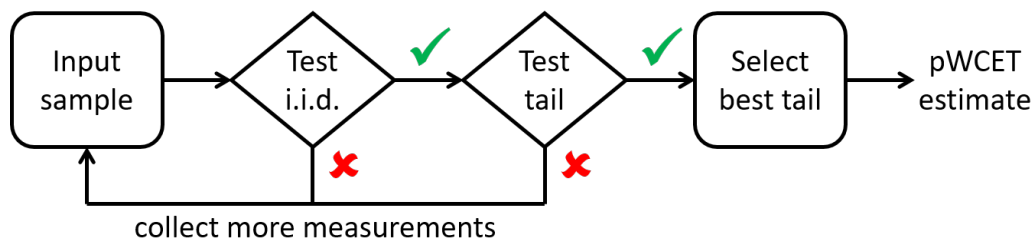


Figure 6: MBPTA-CV process to obtain pWCET estimates.

CV-plot is shown in Figure 5. The x-axis shows, for a sample of 1,000 measurements, out of which the 500 lowest values are discarded, from left to right the number of exceedances discarded. For instance, when the x-axis indicates 400, it indicates that only the highest 100 values of the sample are retained, and the blue line indicates the estimated *CV* for those exceedances. The red lines include the range of values where the exponentiality assumption for the tail cannot be rejected with 95% confidence. If the blue line is below both red lines, then exponentiality is rejected and the tail is regarded as light. Hence, in this case using exponential tails is pessimistic but reliable for pWCET estimation. However, if the blue line is above both red lines, then exponentiality is rejected and the tail is regarded as heavy, thus meaning that exponential tails cannot be used. In the case of MBPTA-CV, the CV-plot is used to guarantee that the set of highest values used for pWCET estimation can be approximated with an exponential tail reliably.

3.2.2 MBPTA-CV Steps

Next, we review the steps of the MBPTA-CV process and assess their fit for HPC applications on HPC systems.

Input Sample Generation MBPTA-CV application process starts with the collection of an execution time sample. In the default application of the method, such sample must have at least

100 execution time measurements, although, in general, the larger the sample, the higher the chances to converge and deliver a pWCET estimate. Moreover, in general, larger samples have more information about the distribution and hence, pWCET estimates obtained will be tighter.

Depending on the duration of the HPC application under analysis and the computing resources available, the number of measurements that may be collected may vary. We note that MBPTA-CV requires 100 measurements because the lowest half is discarded considering that they are not appropriate to predict high execution times, and at least 50 measurements (maxima) are needed to derive a highly reliable pWCET estimate, as needed for critical real-time embedded systems. However, as noted by the authors in [3], different applications of EVT allow the minimum number of maxima used to fit a tail to be between 10 and 50. Therefore, we note that, although the reliability and tightness of the pWCET distribution may decrease, using samples of only 20 measurements is still possible – with the lowest half being discarded – while having a sound application on MBPTA-CV. Note that, in this case, the 10 maxima should allow fitting a tail reliably. Otherwise, a larger sample would be needed until having at least 10 maxima allowing such reliable tail fitting.

Observation 4. The lowest number of execution time measurements to use MBPTA-CV for HPC applications can be decreased down to 20, instead of 100.

3.2.3 Independence and Identical Distribution

Input data must be independent and follow an identical distribution (i.i.d.), which, in practice, means that execution time measurements have been collected with the same initial conditions. For instance, this is achieved by collecting execution times for the HPC application on the same system resetting any state remaining from previous executions before each experiment, and using either the same input set or choosing the input set randomly out of a pool of relevant input sets.

In the context of HPC systems, while we can control input data and some state for the application under analysis, some other activities (e.g. periodic OS services) may not be sufficiently controlled and may affect differently each execution measurement. Moreover, such impact may be periodic, so that i.i.d. properties across measurements do not hold. However, as shown in [31], i.i.d. properties do not need to hold for all measurements but only for maxima (i.e. the subset of measurements above a given threshold used to draw the tail with EVT). This, in essence, implies that the impact of those OS services must have a sufficiently low relative impact not to make non-maxima become maxima. In general, OS services may take up to few milliseconds to execute. Hence, by analyzing applications whose execution time is some orders of magnitude higher (e.g. hundreds of milliseconds or more), the impact of OS noise becomes irrelevant in practice. Still, i.i.d. properties of maxima need to be statistically tested for a reliable application of EVT as part of MBPTA-CV.

Observation 5. I.i.d. properties may not hold for HPC applications, but, as long as execution time is large enough, i.i.d holds for maxima, which is enough for a reliable application of MBPTA-CV.

Exponential Tail Test MBPTA-CV, as explained before, builds upon the CV-plot to test whether the maxima retained for pWCET estimation are compatible with the exponential as-

sumption. In other words, such maxima is regarded as acceptable if maxima are either compatible with exponential or light tails, since both are reliably upper-bounded with exponential tails. If maxima in the sample fails to fulfill such statistical criteria, since the distribution under analysis must meet this probabilistic property by construction, a larger sample is requested until, eventually, it converges.

The particular method considered, MBPTA-CV, in fact, imposes that not only those maxima used for pWCET estimation must be compatible with the exponential assumption, but also smaller sets of maxima with at least 10 measurements¹. Since MBPTA-CV imposes the use of at least 50 maxima, then all sets of maxima between 10 and 50 elements must pass the exponentiality test (i.e. have the *CV* estimator below the top red line for all those sets of maxima). As discussed for the sample generation, fitting the tail with only 10 maxima, while less reliable, it is still acceptable based on common practice. Hence, the exponentiality test provided by the *CV*-plot is also restricted to such set of maxima, which facilitates passing it.

Observation 6. The exponential test may be less demanding for HPC applications if the smallest set of maxima to estimate the pWCET distribution is decreased below 50 measurements (being at least 10).

Select the Best Tail Once there is at least a set of maxima passing the exponentiality test, selecting the best set of maxima is an arbitrary choice statistically speaking since any set is equally statistically valid, as pointed out in [3]. In the case of MBPTA-CV, out of all sets of maxima passing the test, the one with a *CV* estimator closer to 1 (the expected value for exponential tails) is used. In the case of HPC applications, there is no particular reason to change this criteria.

Observation 7. The criteria to select the best tail remains unchanged for HPC applications.

pWCET Estimate Once determined the set of maxima to use for pWCET estimation, fitting an exponential tail is an automatic step independent of any other consideration, such as, for instance, the application under analysis or the domain of such application. However, the particular exceedance probability to consider may change across domains. In the case of critical real-time embedded systems, such probability normally relates to acceptable failure rates or residual risk as determined in the particular functional safety standard in the domain.

In the case of HPC applications, in general, no such standard exists. In general, those applications, rather than safety critical, are mission critical, thus meaning that a failure to execute properly diminishes the success of the mission. For instance, acceptable failure probabilities for safety-critical systems may be in the order of 10^{-12} per run, whereas an HPC application processing, for instance, soil data to detect appropriate locations to set oil wells may afford higher failure probabilities (e.g. 10^{-6} per run).

Observation 8. Exceedance probabilities acceptable for HPC applications may differ from those usually considered for critical real-time embedded systems.

¹Based on common practice, smaller sets of maxima (i.e. 9 measurements or fewer) are regarded as unreliable [3].

3.3 Minimizing Network Interference

pWCET estimates will hold valid if the execution time conditions at which measurements of the application under analysis were taken do represent a worst-case condition scenario. However, in the context of HPC systems this is difficult to achieve. Previous sections focus on capturing the variability of application execution time variability caused by software libraries and OS effects. Unfortunately, there are other sources of execution time variability affecting execution time that cannot be captured using those means. In particular, the effect that other applications running in the system or maintenance tasks have in the application has a significant impact in execution time. The impact of co-running applications can be mitigated at the resource manager level by preventing the allocation of computing resources within the same node for applications when a critical application is already executing in this node. However, even if we avoid these situations execution time variability can exist due to interference in the network since in the HPC system there will be other shared resources such as distributed file system that will be accessed through the same interconnection medium. At the same time, maintenance/monitoring tasks occurring in parallel with the execution of a critical application will necessarily interfere with our application at the network level.

To control the impact of network interference or even remove it we rely on the utilization of Infiniband specific quality-of-service features. Infiniband is a low-latency and high-bandwidth interconnection network commonly used in supercomputing facilities that also favours having predictable behavior in the execution time of applications. In RECIPE, the servers are connected utilizing Infiniband card adapters which can be configured to isolate critical applications. To do so, we have to exploit the Service Levels(SL) features provided by this network. Within Infiniband SL are mapped to specific virtual lanes (VL) and VLs are arbitrated using a 2-level weighted round-robin arbiter. In the first level arbitration we find the high-priority flows and in the second layer the low-priority flows. Within each arbitration layer different weights can be given to specific VLs to achieve different levels of service. Since our intention is to minimize or even remove network interference we define a service level SL=0 with the highest possible level of priority and allow only the critical application to use this service level. To do so, the following configuration has to be employed in the Infiniband subnet manager configuration file:

```
qos_max_vls 15
qos_high_limit 255
qos_vlarb_high 0:255
qos_vlarb_low 0:0,1:64,2:128,3:192,4:0,5:64,6:64,7:64
qos_sl2vl 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,7
```

The previous configuration defines a high limit of 255 and puts only VL 0 as a high priority channel. In general, the value of the high limit represents the amount of packets from the highest priority level that can be sent before a low priority level packet is sent but when this value is set to 255 it means that high-priority levels will be always prioritize. Note that this configuration may cause starvation in theory, however, this will only occur if it always exist a packet of the high critical task to be sent and this is something we can ensure it will never happen when profiling our application.

With the previous configuration we ensure that communications using VL=0, that is mapped to SL=0, will have always priority over other communications. Then, we have to allow only a critical application to use SL=0. This can be easily implemented for MPI applications when the application is launched (`mpirun -sl 0`). Although, in RECIPE, we will control QoS features at the runtime resource monitor level. At the same time, to avoid other communications to use SL 0 we have to configure the system to force other services to default to other SLs. This can be done in a qos policy file specifying the following directives:

```
qos-ulps
default : 0
any, target-port-guid MYPOR_T_HEX
end-qos-ulps
```

For instance, assuming we have in our system a Lustre distributed file system we can configure Infiniband to force all communications to default to SL 1 and thus, avoid interferences from the accesses to the file-system.

Note that D4.4 explains how to install and configure Infiniband drivers so that the previous configurations can be employed.

3.3.1 Summary

As described in this section, the use of MBPTA-CV in the context of HPC applications is viable as long as particular considerations are taken into account. Those considerations relate to choosing appropriate sample sizes, with less demanding criteria than for critical real-time embedded systems, and appropriate measurement collection protocols to achieve i.i.d. at least for maxima. Constraints for exponentiality compliance can also be decreased, and acceptable exceedance probabilities may be, in general, higher than those for critical real-time embedded systems.

3.4 Related Work

The critical real-time embedded systems domain is prolific in techniques for WCET estimation. A family of techniques is based on the static analysis and abstract interpretation of the software under analysis on a timing model of the hardware platform [45]. However, it has been shown that those methods are appropriate for very simple microcontrollers [2], far simpler than those in HPC systems.

On the other side of the spectrum, we can find measurement-based timing analysis techniques, which have been used in many real systems due to their flexibility and portability to virtually any platform [33]. Most of those techniques aim at estimating a deterministic WCET estimate that must hold under any circumstance, which often leads to a tradeoff between reliability and tightness. In general, those approaches build upon collecting execution time measurements and adding an engineering margin on top of the maximum observed execution time, whose reliability is hard – if at all possible – to assess [2].

Still in the area of measurement-based timing analysis techniques, a new family of techniques has been proposed, building on probabilistic principles to derive a pWCET rather than an absolute bound [7]. Those techniques build upon appropriate measurement collection protocols

as well as, potentially, on specific platform support – either hardware or software – to increase the representativeness of the tests used for pWCET estimation. Amongst those techniques, in this work we focus on MBPTA-CV [3], since a detailed method description and an actual implementation are publicly available.

Most MBPTA methods build upon the assumption of independent data or, at least, independent maxima. However, some works have shown that some weak dependence can also be acceptable [9, 40].

Finally, to the best of our knowledge, no specific methods have been devised to predict reliably high execution times of HPC applications. Hence, our adaptation of a mechanism – MBPTA-CV – used in another domain to fit the needs and constraints of HPC systems is a pioneering attempt to reach the goal of reliable and tight estimation of execution time bounds for HPC applications.

3.5 *libta*: A C++ library for MBPTA-CV Analysis

libta [8] implements the technique of MBPTA-CV [3] in a header-only library. The library is template-based, so it is possible to support generic data structures. Error handling is performed through the `Either` class taken from the *neither* library [30], which offers a modern way of handling errors without the need of using return codes nor exceptions.

3.5.1 The API

The API is contained with the `libta` namespace. In this section, such a namespace is not specified before class and method names for the sake of conciseness of the showed headers. The recommendation is to use `float` and `double` datatypes to process data. Using `long double` may slow down the computation significantly.

The `Request` class represents the input for the library. It wraps a `std::vector`, but it may be easily changed if needed. These are the public methods:

```
Request<T>(): Empty class constructor.
std::vector<T>::iterator begin(): Begin iterator for for-range-loops.
std::vector<T>::const_iterator cbegin() const: Const begin iterator for for-range-loops.
std::vector<T>::iterator end(): End iterator for for-range-loops.
std::vector<T>::const_iterator cend(): Const end iterator for for-range-loops.
void add_value(const T &time): Add new values to the timing array.
const std::vector<T> & get_all() const: Getter for the whole timing array.
int get_size(): Get the number of added values.
```

The `Response` class is a generic interface of the responses produced by the *libta* API. Most of the times it used to report errors, except for the `ResponsePWCET<T>` sub-class, which reports a valid timing estimate given an input probability. Figure 7 shows the inheritance diagram related to the `Response` class and the following list briefly describes the meaning of each `Response`:

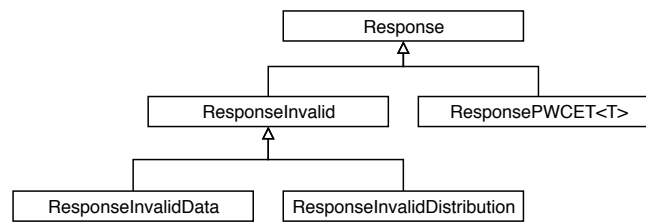


Figure 7: Inheritance diagram of Response.

Response: A generic response.

ResponseInvalid: A generic response with a string message.

ResponseInvalidData: This is returned when the a request cannot be used to estimate a distribution.

ResponseInvalidDistribution: This response is returned when it is not possible to produce a pWCET with a given distribution.

ResponsePWCET<T>: A response obtained by successfully getting a pWCET estimate.

The class `EVTDistribution<T>` represents an estimate of a heavy-tail distribution obtained by processing a `Request<T>`. A user should produce an `EVTDistribution<T>` instance using the `DistributionAnalyzer` class. This class offer the following public methods (note that the data type `EitherPWCETResponse<T>` is an alias for `Either< ResponseInvalidDistribution, ResponsePWCET<T> >`):

`EitherPWCETResponse<T> getPWCET (T probability) const:` Get the execution time that exceeds with the given probability.

`EitherPWCETResponse<T> getPWCETLow (T probability) const:`Get the execution time that exceeds with the given probability with risky assumption.

`EitherPWCETResponse<T> getPWCETHigh (T probability) const:`Get the execution time that exceeds with the given probability with safe assumption.

`T getMaxExecutionTime () const:` Get the collected maximum execution from the real values.

`const std::vector< T > getTailValues() const:` Get the real values used for the tail estimation.

`DistributionAnalyzer` is responsible of the production a heavy tail estimate through the `estimate_distribution` method, which requires a shared pointer that references a `Request` and a `const int rank_length` that tells how many points the output distribution should have. Setting a higher value of `rank_length` increases the accuracy of the result, but it slows down the production of the curve. Such a method returns a `Either< ResponseInvalidData, EVTDistribution<T> >` instance.

3.5.2 libta Flow

The standard usage of `libta` is shown in the Flow-Chart present in Figure 8. At first, a new

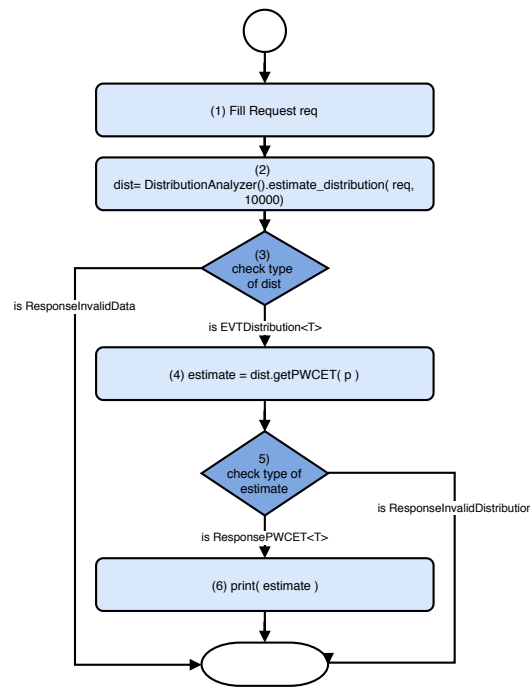


Figure 8: Flow-chart of the usage of *libta* to collect the minimum execution time that can be observed with a given probability p .

instance of `Request` must be created and filled with samples the represents the execution time of a target (step 1 in the previously mentioned Figure), then the `estimate_distribution` method of the `DistributionAnalyzer` class produces an estimate of a heavy tail distribution in a `EVTDistribution` instance if it's possible to apply the MBPTA-CV on the given input, otherwise it returns a `ResponseInvalidData` (steps 2 and 3). This may happen if one of the number of CV values valid for estimating the heavy tail is smaller or equal than 10. If a valid distribution is returned, now it is possible to ask to the distribution the minimum execution time that can be observed with input probability p by calling the `getPWCET` method (step 4). If it's possible to compute such a value, then a `ResponsePWCET` instance is returned, otherwise a `ResponseInvalidDistribution` is what the programmer will get (step 5). Note that `ResponseInvalidDistribution` is not used in the API yet, but this may change in the future.

3.6 Conclusions and Next Steps

The availability of HPC platforms nowadays has led to a plethora of HPC applications in a variety of domains and contexts. Such ubiquity of HPC has made a number of new requirements emerge. Out of those, timing guarantees are particularly important for a number of applications with real-time needs.

This work shows how some techniques based on software randomization and MBPTA, if used reliably, allow performing extensive and representative execution time test campaigns for HPC applications and predicting high execution times.

The next steps consist of evaluating this approach in, at least, one of the use cases of RECIPE (UC1 in particular) to validate the effectiveness of the solution, as well as perform a full inte-

gration of this tool in the runtime resource manager to facilitate the effective management and execution of HPC applications atop.

4 Thermal Modelling

4.1 System-on-Chip modelling

Thermal modelling within RECIPE is required to first enable the development of thermal-aware policies but also, and more importantly, to enable the evaluation of thermal effects on the long-term reliability of servers. Within WP3, thermal models of processors and servers are developed, that are then used in WP2 for the development of thermal and reliability aware management policies at the server level.

In particular, thermal stress and spatial gradients are proven to have a negative effect on the long-term reliability of Multi-Processors System on Chip (MPSoCs) [14], impacting their FIT rate [29]. Temporal Temperature Gradient (TTG) can be defined as the rate of temperature changes over time. For a given time, the rate of the temperature changes from one point to another indicates the spatial temperature gradient (STG). Both STG and TTG pose a critical impact on the system lifetime reliability, but STG is mostly affected by the power and thermal management techniques applied at the overall MPSoC system, i.e., the allocation and specific setup of all cores in the system need to be taken into consideration. In contrast, TTG is mostly affected by the core frequency and the workload running in each specific core.

Thermal cycling is the phenomena which takes place when the temperature rises up (or drops down) and goes back to the initial value (which can be defined as a thermal cycle) frequently [46]. MTTF reduction due to thermal cycling occurs due to the mismatch on the expansion coefficient between the layers of the chip, which results in thermo-mechanical stresses. Thermal cycling (TC) tends to reduce the whole system MTTF as the number of cycles or amplitudes increases. Large amplitudes are normally induced due to improper task scheduling on a single core. Number of thermal cycles increases especially by the power management techniques which frequently turn cores on and off [15].

All in all, in order to reliably estimate the MTTF of a system, we need ways of modelling and simulating temperature in a spatio-temporal way. In this sense, simulators such as the 3D-ICE tool developed as part of the previous work of EPFL can help in accurately modelling these effects [41]. However, in order for this tool to work efficiently and accurately within the framework of RECIPE, there is a need to incorporate the following enhancements:

- Enabling the simulation of arbitrary state-of-the-art cooling mechanisms, such as the ones found in current servers. In particular, within RECIPE we need to enable the simulation of both natural convection mechanisms (i.e., heatsinks), and forced convection cooling (i.e., heatsinks plus fans).
- Proposal of a methodology that will allow us to assess the impact of the main control knobs related to temperature in today's servers, which range from workload allocation, DVFS setting and fan speed control policies. This methodology needs to exploit the capabilities of our simulation tool.
- Proposal of a methodology to adequately link the thermal aspects to the reliability of the system. For this purpose, we will use the MTTF and reliability models proposed by T3.1, which will be incorporated into the policies developed by EPFL in both T3.5 in WP3 and T2.3 in WP2.

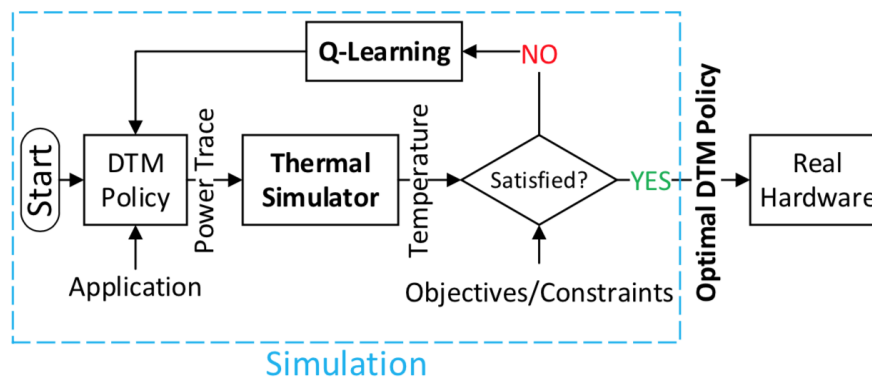


Figure 9: Thermal modelling methodology to enable DTM and reliability management

Based on the previous work released in v2.2.6 of 3D-ICE (project background), within this task of RECIPE, we have developed both the natural convection (heatspreader plus heatsink) and the forced convection (heatsink plus fan) models, using a real Thermal Test Chip (TTV), for real chips and cooling devices, using real traces. This contributions are released in v2.2.7 of 3D-ICE. We plan to release v2.2.8 in the near future, incorporating both the natural and forced convection models developed. The details of the pluggable heatsink and its integration with 3D-ICE will be described as part of Deliverable D3.5.

Furthermore, to create accurate models of the system, we need ways of validating them against real devices. For this purpose, within the RECIPE project we have also created a platform which comprises a real test chip [42] for accurate thermal characterization. In particular, this platform is based on a Thermal Test Chip (TTC), an integrated circuit containing an array of power dissipating elements and an array of temperature sensors. Our thermal platform is capable of applying a generic power dissipation pattern to the thermal test chip and measuring the corresponding temperature map, at a rate up to 1 kHz. This capability allows to measure the temperature map of an integrated circuit subject to reference power dissipation maps, and thus to design and validate thermal models. In our particular case, the chip is organised as a 4 by 4 array of individual cells, each capable of temperature sensing and power generation through a resistive element. The heating element in each cell is capable of dissipating up to 12 W. This allows to simulate the equivalent of a 16-core chip with a total power dissipation of 192 W.

The methodology for envision for enabling DTM and reliability management is the one depicted in Figure 9. The simulator will enable us perform an exploration of the impact of aggressive DTM management policies which may not only cause performance degradation and additional power consumption, but more importantly, jeopardizes lifetime reliability of the whole system. In fact, one of the main reasons that makes researchers reluctant to consider fan speed control as a key DTM approach is the lack of a transient thermal simulator for MPSoCs with proper integration of fans. The incorporation of fan models in 3D-ICE, and the use of this methodology provides a comprehensive framework for exploring thermal effects of DTM policies in a safe way.

Workload allocation, DVFS and fan speed altogether drastically increase the number of runtime design parameters to be decided by a DTM and reliability-aware policy, which leads to additional challenges to find the best values for optimal behavior of the whole system. Our methodology enables the exploration of this design space in an automated way. In particular, we envision the

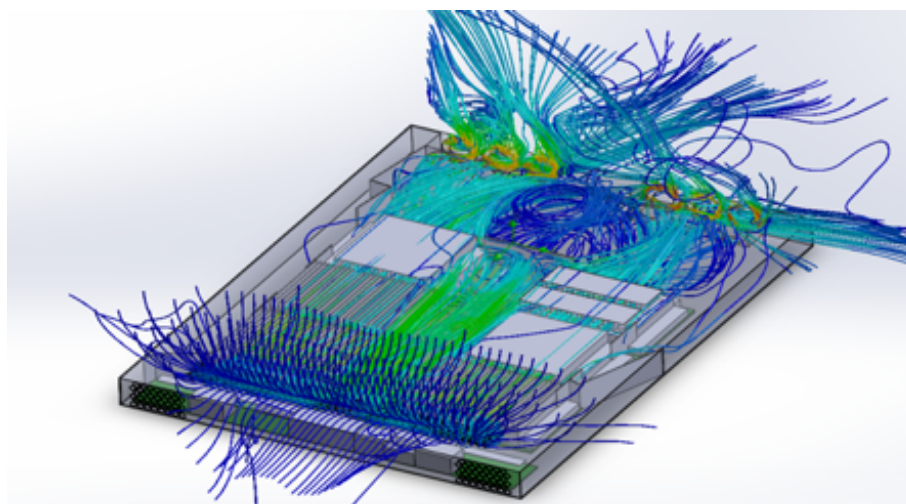


Figure 10: Analysis of the airflow inside a chassis

use of Reinforcement Learning (RL) techniques in WP2. The RL agents can explore the design space using 3D-ICE and combine that with the impact of reliability. Initial experiments [42], prove the feasibility of our approach.

4.2 Server/farm modelling

PSNC contributes to thermal models ranging from single processing units through the computing nodes and racks up to whole server rooms, including models of cooling equipment. Thermal models at processor scale benefit from Newton’s law of cooling supported by the duality between thermal and electrical phenomena. The model considers surrounding temperature, CPU power state, its thermal capacitance and resistance characteristics. The latter one reveals the impact of cooling system on CPU thermal behaviour. By these means, the rapidity and the rate of temperature changes towards reaching the stable value for a given state can be described. Additionally, proposed models utilize a well-known heat transfer formula and the law of energy conservation to describe the mutual impact of neighbouring nodes. Last but not least, the models allow incorporating power leakage function specifying the CPU power usage due to the increase of its temperature. A detailed description of the aforementioned models can be found in [36] and [37]. At higher levels, the models are used to estimate temperature of the air leaving nodes/racks and the thermal settings required to cool the servers. In this way they enable calculations of cooling equipment in efficiency and power usage. All these models are incorporated into the DCworms simulation toolkit [28], developed by PSNC, dedicated to modelling and simulation of computing infrastructures to estimate their performance, energy consumption and energy-efficiency metrics for different applications and management strategies. Moreover, these models are bidirectionally supported with the CFD simulations, in order to verify thermal state of given IT components and identify the possible hotspots. The example results of CFD airflow analysis and temperature are presented in figures 10 and 11.

At the infrastructure (i.e. data center) level we are developing a CFD-based simulation framework, build around OpenFOAM software that supports and facilitates the process of model creation and definition of simulations settings. Models and simulations results for different hardware architectures and levels may be provided. A number of thermal and power data, obtained

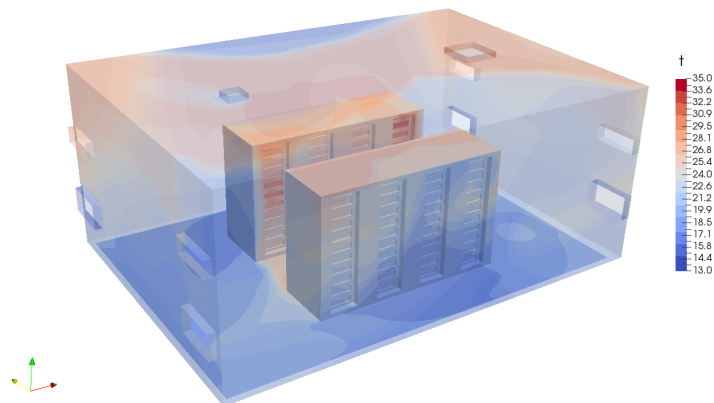


Figure 11: Analysis of the temperature inside a server room

via the experimental results, may be used to verify and tailor thermal models. These results can be also applied directly as input data into the aforementioned simulation environments. Within the project, models developed by will be combined with the output of 3D-ICE thermal simulation tools. As the tool considers the detailed specification of SoCs, it could constitute to the more coarse-grained CPU thermal models extending their capabilities, to deliver enhanced thermal policies considering the full server room.

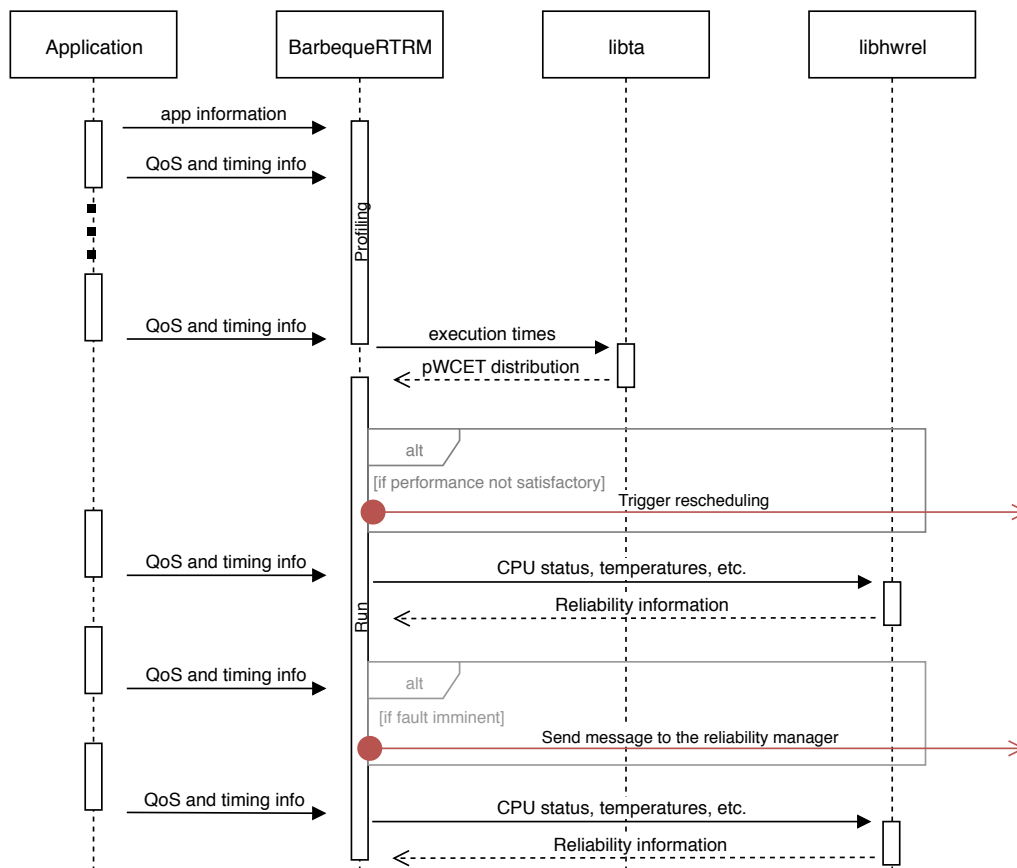


Figure 12: UML sequence diagram of the BarbequeRTRM accessing the functionalities provided by the HW reliability and the timing analysis libraries.

5 RTRM Integration

The Barbeque Run-Time Resource Manager (BarbequeRTRM) is a highly modular software written in C++. The developed interface enables the integration with hardware reliability monitors and timing analysis tools. The developer of such software is required to implement a specific interface as a C++ library and, in particular, he or she has to implement a pure virtual C++ class. The library code is required to be in C++ and with the modern standard C++11. However, the developer can easily build a wrapper in any other programming language and expose the functions using C++ (e.g. the Python language). The reason behind this choice is both to maintain the compatibility with the current BarbequeRTRM internal structure, to guarantee the interoperability with several programming languages and to have the maximum possible performance. In fact, C++ is very advantageous if the external tool (reliability monitor or timing analyzer) has to run computationally intensive algorithms: C++ could guarantee the maximum performance and scalability. This is especially important when the reliability monitor or the timing analysis tool are required to carry out the results within a certain timing deadline.

The information exchanged via the built interface and the logical steps of the resource management decisions are outlined in the sequence diagram of Figure 12. In a initial phase, the application is profiled gathering information about its timing characteristics. To do this, the

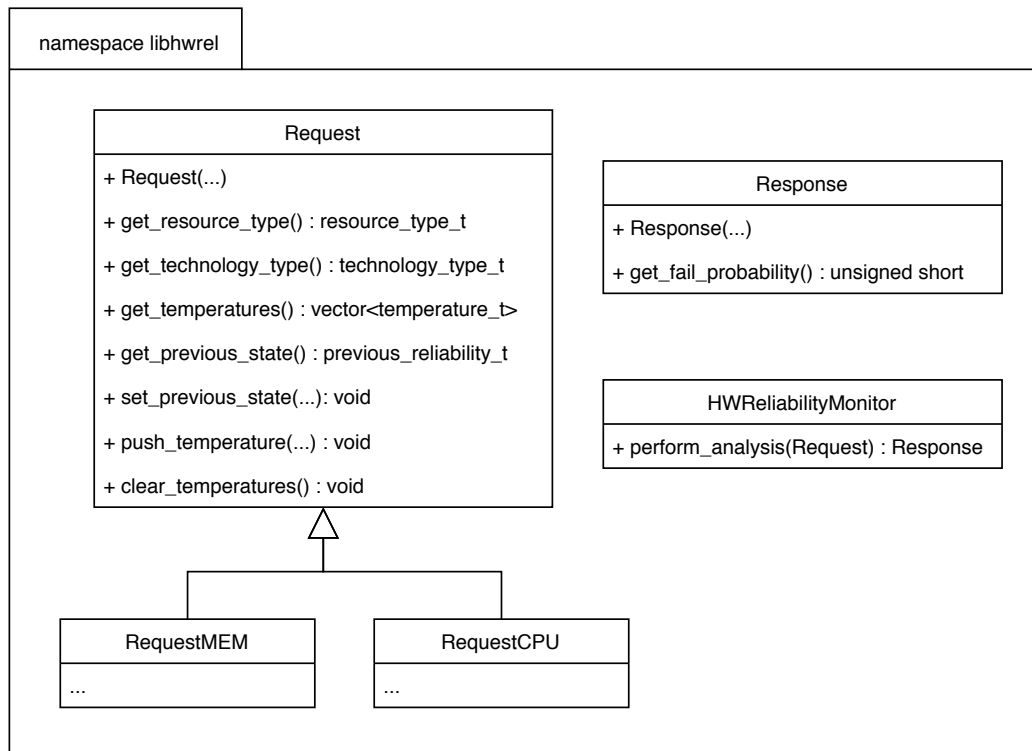


Figure 13: HW Reliability library: UML class diagram.

BarbequeRTRM acquires the execution times and calls the timing analysis tool. Then, the application runs and the timing and reliability information are continuously checked to detect any anomaly and to trigger the appropriate countermeasures.

5.1 Hardware Reliability library (libhwrel)

The goal of the hardware reliability monitor is to provide the information about the status of the main hardware components, such as CPU and memory, in order to predict a deterioration of their reliability. This information can be used to perform reactive and proactive resource management actions that: (1) reduces the ageing and wear out of the components, and (2) performs recovery actions in case of failures. The triggered mechanisms are the subject of deliverable D2.2.

The interface to hardware reliability monitor is provided as a C++ library to be implemented (libhwrel). Its UML class diagram is depicted in Figure 13. The library is encapsulated in a dedicated namespace, and three main classes are used: (1) the **Request** class that contains the request sent by the resource manager to the library, (2) the **Response** class, that is the reply from the library to the resource manager, and (3) the main class called *HWReliabilityMonitor* that performs the actual analysis taking a **Request** class as input and providing a **Response** class as output. The **Request** class contains various method to access to the resource type (CPU, GPU, Memory, etc.) and the technology used (ASIC, FPGA, etc.). Then, the resource manager registers the temperatures values of the components and the dedicated information like memory size and CPU usage (thanks to appropriate sub-classes). The output of the library (**Response** class) is, instead, just an integer representing the probability of failure in per-mille/program-run.

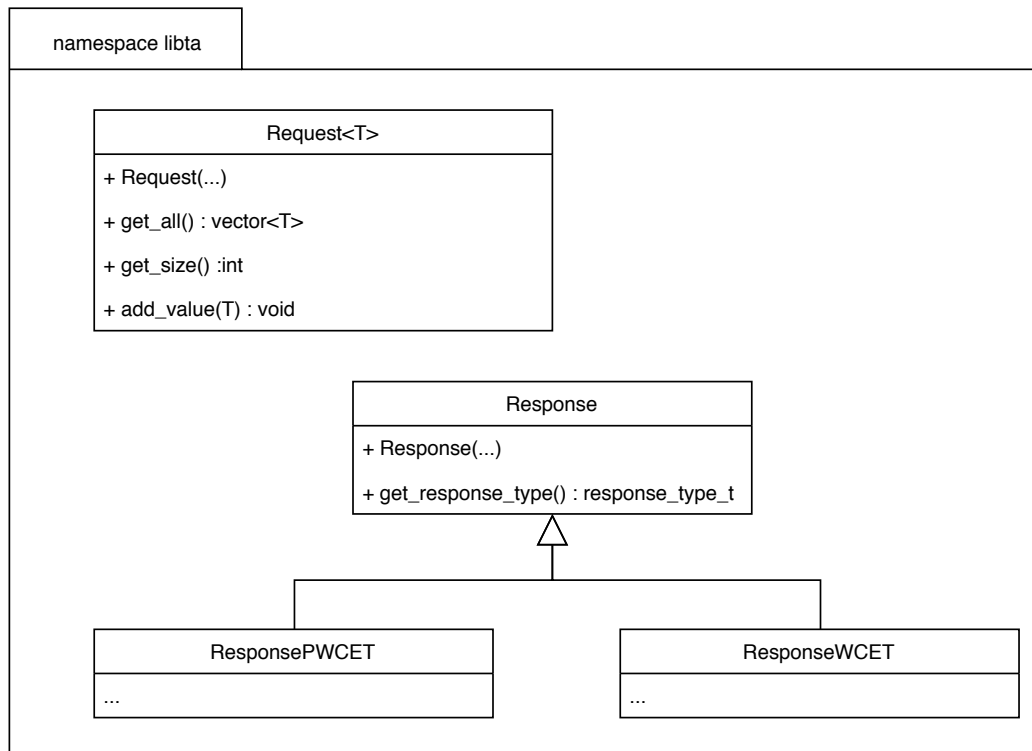


Figure 14: Timing Analysis library: UML class diagram.

5.2 Timing Analysis library (libta)

The timing analysis tool is in charge of performing statistical analyses over the set of measurements of tasks’ execution time. The tool computes the resulting statistical distribution thanks to suitable statistical algorithms. The BarbequeRTRM provides to the timing analysis tool the list of timing measurements across a pre-defined timespan. The output of the library is a statistical distribution representing this sample. The usual distribution provided by the library is the probabilistic-WCET (Worst-Case Execution Time) distribution, i.e. the statistical distribution of samples at the extremes.

The interface to the timing analysis tool is provided as a C++ library to be implemented (libta). Its UML class diagram is depicted in Figure 14. The structure is similar to the libhwrel library, having the Request, Response, and TimingAnalyzer classes with the same meaning. The Request class is a template-parameter class, so the execution times can be provided in different types, e.g. integer or float. This has been done to maintain the flexibility of using the library with any timing measurement system. Its purpose is to store the vector of execution times to be analyzed. The Response class is instead a generic class that must be specialized. Currently, we proposed two possible responses: a statistical pWCET distribution or a fixed WCET value. The actual choice between them is library dependent, i.e. if it exploits probabilistic or not WCET estimation methods.

6 Summary

The proliferation of heterogeneous HPC systems and applications in new domains has led to new requirements related to non-functional requirements (time, power, reliability, temperature, etc) and the need for platforms to satisfy them. In this deliverable, we provided a summary of the progress achieved in the following activities related to different QoS aspects, as part of WP3:

- Reliability methodology. We reported the methodology used. We showed that it is applicable to any heterogeneous system and we showed how we can integrate all reliability and degradation measurements into a single system value. We described the predictive mechanism we will integrate in the run-time manager.
- Timing analysis. Timing predictions are key to make an efficient use of resources while ensuring that each application is allocated enough resources to complete in time. We reported the mechanism to provide statistically significant timing predictions. This method is already encapsulated in a library and ready for integration in the run-time manager.
- Thermal modelling. Thermal modelling techniques to estimate chip temperature are required in order to enable the assessment of temperature-related reliability effects on chips. The models developed provide the means to know chip temperature accurately with the goal of proactively enhancing chip reliability when no hardware support (i.e. temperature reading) is available.
- Applications characterization. This task has just started (1st of October), so there is nothing relevant to report yet. Details on this topic will be provided in D3.6 in month 30 (October 2020).
- RTRM Reliability, Timing, and Thermal Policies Development. We provide the libraries defined that will serve as an interface with the models developed (task 3.1 to task 3.3). This work is at a preliminary stage, since it spans from April 2019 to March 2021. Yet, the interface has already been defined so we are on track.

This deliverable has described the current status of WP3. Novel contributions on timing, reliability and thermal models have been proposed and submitted for publication. The tasks are progressing according to the plan and we look forward to the use of the novel approaches for deriving better run time manager policies.

References

- [1] Jaume Abella. Mbpta-cv. <https://doi.org/10.5281/zenodo.1065776>, November 2017.
- [2] J. Abella et al. WCET analysis methods: Pitfalls and challenges on their trustworthiness. In *SIES*, 2015.
- [3] J. Abella et al. Measurement-based worst-case execution time estimation using the coefficient of variation. New York, NY, USA, 2017. ACM.
- [4] G. Agosta, W. Fornaciari, G. Massari, A. Pupykina, F. Reghenzani, and M. Zanella. Managing Heterogeneous Resources in HPC Systems. In *Proc. of PARMA-DITAM '18*, pages 7–12. ACM, 2018.
- [5] Emery D. Berger and Benjamin G. Zorn. DieHard: Probabilistic memory safety for unsafe languages. In *In Proceedings of the ACM SIGPLAN 2006 Conference on Programming Language Design and Implementation*, pages 158–168. ACM Press, 2006.
- [6] Emery D. Berger, Benjamin G. Zorn, and Kathryn S. McKinley. Composing high-performance memory allocators. pages 114–124, 2001.
- [7] Francisco J. Cazorla, Leonidas Kosmidis, Enrico Mezzetti, Carles Hernandez, Jaume Abella, and Tullio Vardanega. Probabilistic worst-case timing analysis: Taxonomy and comprehensive survey. *ACM Comput. Surv.*, 52(1):14:1–14:35, February 2019.
- [8] Barcelona Supercomputing Center. libta: header-only implementation of mbta-cv. <https://chef.heaplab.deib.polimi.it/source/libta>.
- [9] S. Coles. *An Introduction to Statistical Modeling of Extreme Values*. Springer, 2001.
- [10] Intel Corp. Intel processor counter monitor, 2019. <https://github.com/opcm/pcm/>.
- [11] Intel Corp. Intel resource director technology, 2019. <https://www.intel.com/content/www/us/en/architecture-and-technology/resource-director-technology.html/>.
- [12] NVIDIA Corp. Nvidia gpu monitoring tools, 2019. <https://github.com/NVIDIA/gpu-monitoring-tools/>.
- [13] Intel Corporation. Reliability Report 1H. 2017.
- [14] A. K. Coskun, D. Atienza, T. Simunic Rosing, T. Brunschwiler, and B. Michel. Energy-efficient variable-flow liquid cooling in 3d stacked architectures. In *2010 Design, Automation Test in Europe Conference Exhibition (DATE 2010)*, pages 111–116, March 2010.
- [15] Ayse Kivilcim Coskun, Tajana Simunic Rosing, and Kenny C. Gross. Temperature management in multiprocessor socs using online learning. In *Proceedings of the 45th Annual Design Automation Conference*, DAC '08, pages 890–893, New York, NY, USA, 2008. ACM.
- [16] F. Cros, L. Kosmidis, F. Wartel, D. Morales, J. Abella, I. Broster, and F. J. Cazorla. Dynamic software randomisation: Lessons learned from an aerospace case study. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pages 103–108, March 2017.

-
- [17] L. Cucu-Grosjean et al. Measurement-based probabilistic timing analysis for multi-path programs. In *ECRTS*, 2012.
 - [18] Charlie Curtsinger and Emery D. Berger. STABILIZER: Statistically sound performance evaluation. *SIGARCH Comput. Archit. News*, 41(1):219–228, March 2013.
 - [19] J. Del Castillo, J. Daoudi, and R. Lockhart. Methods to distinguish between polynomial and exponential tails. *Scandinavian Journal of Statistics*, 41(2):382–393, 2014.
 - [20] DES. Constant temperature accelerated life testing using the arrhenius relationship, 2013. <https://www.desolutions.com/blog/2013/08/constant-temperature-accelerated-life-testing-using-the-arrhenius-relationship/>.
 - [21] P. Ellerman. Calculating reliability using fit & mttf: Arrhenius htol model.
 - [22] J. Flich, G. Agosta, et al. Exploring manycore architectures for next-generation HPC systems through the MANGO approach. *Microprocessors and Microsystems*, 61:154 – 170, 2018.
 - [23] J. Flich et al. Enabling HPC for QoS-sensitive applications: The MANGO approach. In *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 702–707, March 2016.
 - [24] L. Kosmidis, C. Curtsinger, E. Quiones, J. Abella, E. Berger, and F. J. Cazorla. Probabilistic timing analysis on conventional cache designs. In *2013 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 603–606, March 2013.
 - [25] L. Kosmidis, R. Vargas, D. Morales, E. Quiñones, J. Abella, and F. J. Cazorla. TASA: Toolchain Agnostic Software Randomisation for Critical Real-Time Systems. In *ICCAD*, 2016.
 - [26] Leonidas Kosmidis, Eduardo Quiñones, Jaume Abella, Glenn Farrall, Franck Wartel, and Francisco J. Cazorla. Containing timing-related certification cost in automotive systems deploying complex hardware. In *Proceedings of the 51st Annual Design Automation Conference, DAC '14*, pages 22:1–22:6, New York, NY, USA, 2014. ACM.
 - [27] S. Kotz et al. *Extreme value distributions: theory and applications*. World Scientific, 2000.
 - [28] Krzysztof Kurowski, Ariel Oleksiak, Wojciech Piatek, Tomasz Piontek, W. Przybyszewski, Andrzej, and Jan Weglarz. Dcworms - a tool for simulation of energy efficiency in distributed computing infrastructures. *Simulation Modelling Practice and Theory*, 39:135–151, 2013.
 - [29] Clemens J.M. Lasance. Thermally driven reliability issues in microelectronic systems: status-quo and challenges. *Microelectronics Reliability*, 43(12):1969 – 1974, 2003.
 - [30] LoopPerfect. neither. <https://github.com/LoopPerfect/neither>.
 - [31] Yue Lu, Thomas Nolte, Iain Bate, and Liliana Cucu-Grosjean. A new way about using statistical analysis of worst-case execution times. *SIGBED Review*, 8(3), 2011.
 - [32] Giuseppe Massari, Anna Pupykina, Giovanni Agosta, and William Fornaciari. Predictive resource management for next-generation high-performance computing heterogeneous platforms. In *Proceedings of the 18th International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS'19)*, Jul 2019.

-
- [33] Enrico Mezzetti and Tullio Vardanega. On the industrial fitness of wcet analysis. *11th International Workshop on Worst-Case Execution-Time Analysis*, 2011.
 - [34] H. Miyamoto. Semiconductor reliability and quality handbook.
 - [35] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin. A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. In *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36.*, pages 29–40, Dec 2003.
 - [36] Wojciech Piatek, Ariel Oleksiak, and Georges Da Costa. Energy and thermal models for simulation and workload and resource management in computing systems. *Simulation Modelling Practice and Theory*, 58:40–54, 2015.
 - [37] Wojciech Piatek, Ariel Oleksiak, and Micha vor dem Berge. Modelling impact of power-and-thermal-aware fans management on data center energy consumption. *e-Energy*, pages 253–258, 2015.
 - [38] A. Pupykina and G. Agosta. Optimizing Memory Management in Deeply Heterogeneous HPC Accelerators. In *2017 46th Int'l Conf on Parallel Processing Workshops (ICPPW)*, pages 291–300, Aug 2017.
 - [39] M. Riera, R. Canal, J. Abella, and A. Gonzalez. A detailed methodology to compute soft error rates in advanced technologies. In *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 217–222, March 2016.
 - [40] L. Santinelli et al. On the sustainability of the extreme value theory for WCET estimation. In *WCET Workshop*, 2014.
 - [41] Arvind Sridhar, Alessandro Vincenzi, David Atienza, and Thomas Brunschwiler. 3d-ice: A compact thermal model for early-stage design of liquid-cooled ics. *IEEE Transactions on Computers*, 63(10):2576–2589, 2014.
 - [42] Federico Terraneo, Alberto Leva, and William Fornaciari. An open-hardware platform for mpso thermal modeling. In *International Conference on Embedded Computer Systems*. Springer, 2019.
 - [43] A. Vallero, S. Tselonis, N. Foutris, M. Kaliorakis, M. Kooli, A. Savino, G. Politano, A. Bosio, G. Di Natale, D. Gizopoulos, and S. Di Carlo. Cross-layer reliability evaluation, moving from the hardware architecture to the system level. *Microprocess. Microsyst.*, 39(8):1204–1214, November 2015.
 - [44] P Vassiliou and A. Mettas. Understanding accelerated life-testing analysis. In *2002 Annual Reliability and Maintainability Symposium.*, Dec 2002.
 - [45] R. Wilhelm et al. The worst-case execution time problem: overview of methods and survey of tools. *ACM TECS*, 7(3):1–53, 2008.
 - [46] Yun Xiang, Thidapat Chantem, Robert P. Dick, X. Sharon Hu, and Li Shang. System-level reliability modeling for mpso. In *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, CODES/ISSS '10*, pages 297–306, New York, NY, USA, 2010. ACM.

[47] Xilinx. Device Reliability Report, First Quarter 2019. 2019.