uniserver

D3.3 1st Analysis of On-Chip Caches and Dynamic Memories

Contract number	688540
Project website	http://www.uniserver2020.eu
Contractual deadline	Project Month 12 (M12): 31st January 2017
Actual Delivery Date	10th February 2017
Dissemination level	Public
Report Version	1.0
Main Authors	George Papadimitriou (UOA), Manolis Kaliorakis (UOA), Athanasios Chatzidimitriou (UOA), Dimitris Gizopoulos (UOA), Lev Mukhanov (QUB), Georgios Karakonstantis (QUB), Konstantinos Tovletoglou (QUB)
Contributors	
Reviewers	Peter Lathers (APM), Greg Favor (APM), Pedro Trancoso (UCY)
Keywords	Characterization, on-chip caches, DRAMs, voltage/frequency/refresh rate scaling, corrected errors, uncorrected errors, silent data corruptions, crashes

Notice: The research leading to these results has received funding from the European Community's Horizon 2020 Programme for Research and Technical development under grant agreement no. 688540.

Disclaimer
This deliverable has been prepared by the responsible Work Package of the Project in accordance with the Consortium Agreement and the Grant Agreement Nr 688540. It solely reflects the opinion of the parties to such agreements on a collective basis in the context of the project and to the extent foreseen in such agreements.

© 2017. UniServer Consortium Partners. All rights reserved

Acknowledgements
The work presented in this document has been conducted in the context of the EU Horizon 2020. UniServer is a 36-month project that started on February 1st, 2016 and is funded by the European Commission. The partners in the project are:

The Queen's University of Belfast (QUB)
The University of Cyprus (UCY)
The University of Athens (UoA)
Applied Micro Circuits Corporation Deutschland Gmbh (APM)
ARM Holdings UK (ARM)
IBM Ireland Limited (IBM)
University of Thessaly (UTH)
WorldSensing (WSE)
Meritorious Audit Limited (MER)
Sparsity (SPA)

More information
Public UniServer reports and other information pertaining to the project are available through the UniServer public Web site under http://www.uniserver2020.eu.

Change Log

Version	Description of change
0.1	Initial draft – Outline
0.2	Caches (UoA) and DRAMs (QUB) characterization first contents
0.3	Caches (UoA) and DRAMs (QUB) characterization final contents
0.4	Enhancements, corrections
0.5	Review by APM, UoA, QUB, UCY
0.6	Enhancements and corrections after review
1.0	Final version for delivery to EC

Table of Contents

Index of Figures

Executive Summary

This document is part of Task T3.3 “Analysis of caches and dynamic memories” as presented in the Description of Action (DoA) of the UniServer project under Work Package 3 (WP3 – “Analysis and Enhancement of Hardware Substrate Outside Nominal Conditions”). This task uses as target platform the X-Gene 2 board as was described in Task 3.1 “Definition and enhancement of the target platform” and contributes to two objectives: (a) analyse the behaviour of the processor cores and on-chip caches of the project platform under various voltage and frequency settings and stress conditions, and (b) analyse the behaviour of dynamic memories under various refresh rates, supply voltage and stress conditions.

Figure 1 visualizes the goal of this deliverable, its main results, the inputs it uses and which work packages will use its outputs. In general, this deliverable is focused on the presentation of developed tools and methodologies to analyse the behaviour of the on-chip caches and dynamic memories of the target platform (X-Gene 2) when they operate in off-nominal Voltage/Frequency/Refresh rate (VFR) conditions. Making a detailed characterization using these tools and methods, we aim to understand the critical parameters that affect the reliability of the system operating in off-nominal conditions. The outcomes of this characterization are going to be used:

- To develop dedicated programs to stress the system in its operation limits. These programs can be synthetic stress tests (“diagnostic virus”) for the cores, the on-chip caches and the DRAM as presented in T3.2 and T3.3 of WP3 respectively and real life application kernels as presented in D4.3 “StressLog Specification and Interface” which is part of T4.3 of WP4.
- To give valuable directions for the creation of the prediction model presented in T4.4.
- To develop memory failure models that can help in identifying the real issues and effect at the system software level towards the creation of fault-tolerant Hypervisor as described in T5.2 and T5.3 of WP5.
- To reveal the operational limits of the on-board memories which can be exploited for power and performance gains at a later stage in the integrated prototype in WP7.

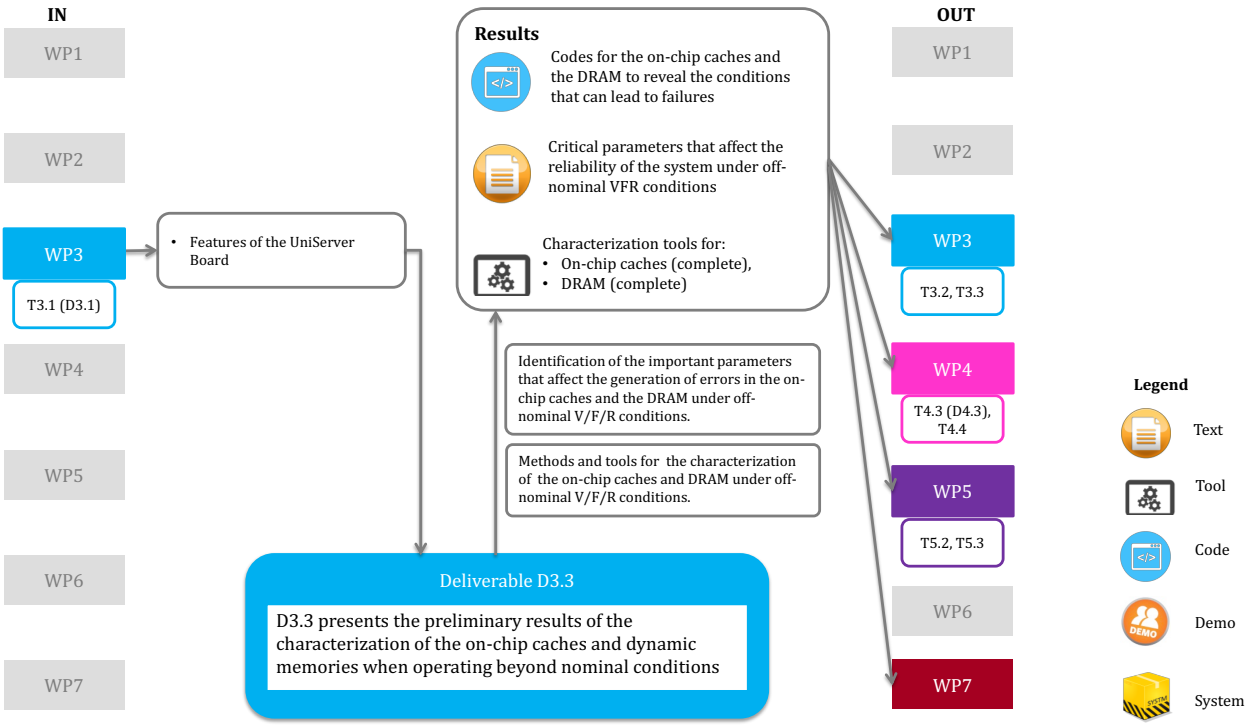

Figure 1: Inputs and Outputs of this Deliverable

The document is organized in the following sections:

- **Introduction:** It introduces the objectives of this deliverable.
- **On-chip Caches Characterization:** The section first briefly presents the related studies in the literature concerning the on-chip caches of microprocessors that operate in off-nominal conditions. Next, the details of the developed automated versatile characterization framework for the study of

the behaviour of the system operating in voltage and frequency conditions different than the nominal values are presented. Finally, we present all the results collected so far, the status of our work and our future plans.

- **Characterization of Dynamic Memories:** It presents the related research on DRAM investigating how applications with different data and accesses patterns affect the number of errors for DRAM operated at the lowered input voltage and relaxed refresh rates. In particular, initially it presents the experimental setup on the X-Gene2 board and the initial characterization results. Then an alternative setup is being presented to overcome the refresh scaling limitations of the X-Gene2 board along with a simulator that targets the characterization of access pattern distributions that seem to have an important effect on the observer error rates.

1. Introduction

UniServer targets to improve performance and energy efficiency in cloud based systems by utilizing the pessimistic Voltage/Frequency/Refresh rate (VFR) margins conventionally adopted by the hardware vendors and used as the only nominal values. However, system operation under off-nominal conditions increases the population of manifested errors compared to the operation under nominal conditions. Changing the voltage, frequency or the DRAM refresh rate may introduce errors on different parts of the chip within the core and the memory hierarchy. In depth characterization and analysis of system's behavior under nominal and scaled VFR conditions is of major importance to finally provide a reliable, high performance and energy efficient system.

The insights that will be revealed by this characterization step are valuable for many tasks and work packages of the UniServer project. They can guide decisions concerning the development of dedicated programs to stress the entire system in its operation limits under off-nominal VFR conditions. Specifically, these programs can be synthetic stress tests ("diagnostic virus") for the cores, the on-chip caches and the DRAM as presented in T3.2 and T3.3 of the DoA and real life application kernels as presented in D4.3 "StressLog Specification and Interface", which is part of T4.3 of WP4. Moreover, the output of executing all these stress programs under off-nominal conditions will feed the Predictor that is responsible to identify the best combination of VFR values in order to ensure the efficient operation of the system in terms of performance, energy and reliability (as presented in T4.4). Finally, the output of these stress programs will provide useful insights concerning the development of a fault-tolerant Hypervisor as described in T5.2 and T5.3 of WP5.

The goal of this deliverable is to present the preliminary results of the characterization of the on-chip cache memories of different levels, as well as the off-chip dynamic memories when operating beyond the nominal conditions in the X-Gene 2 platform. The features of the targeted platform were described in detail in deliverable D3.1 "Definition of the UniServer Board". Concerning the cores and the on-chip caches, a versatile automated framework was developed for system-level voltage and frequency scaling characterization of Applied Micro's X-Gene 2 micro-server family (our tool can be easily extended to the X-Gene 3 when the new system will be integrated in the UniServer workflow). The framework provides fine-grained information about the system's state by monitoring any abnormal behavior that may occur during under-scaled supply voltage and frequency conditions. In our preliminary results using our framework, we observe significant variations when we scale the voltage and frequency executing the same workload in different cores or executing different workloads in the same core. We also discuss all the related work in the literature concerning the characterization of the caches and the impact of voltage noise in systems that operate under-off nominal voltage conditions. This part of the UniServer project technical work is the first that focuses on the characterization of a commercial state-of-the-art ARMv8-based multicore CPU (X-Gene 2).

Concerning the characterization of the Dynamic Random-Access Memory (DRAM) we investigate how different data and access patterns affect the error rate when the DRAM is operated under lower supply-voltage and relaxed refresh rates. Our initial characterization results, presented in this report, reveal that the number of DRAM errors may vary significantly depending especially on the access pattern of the executed micro-benchmark.

2. On-Chip Caches Characterization

2.1. Literature Review

During the last years, the goal for improving microprocessors' energy efficiency, while reducing their power supply voltage is a major concern of many scientific studies that investigate the chips' operation limits in nominal and off-nominal conditions. In this section, we briefly summarize the existing studies and findings concerning low-voltage operation regarding cache memories.

Whilkerson et al. [1] go through the physical effects of low-voltage supply on SRAM cells and the types of failures that may occur. After describing how each cell has a minimum operating voltage, they demonstrate how typical error protection solutions start failing far earlier than a low-voltage target (set to 500mV) and propose two architectural schemes for cache memories that allow operation below 500 mV. The word-disable and bit-fix schemes sacrifice cache capacity to tolerate the high failure rates of low voltage operation. While both schemes use the entire cache on high voltage, they sacrifice 50% and 25% accordingly in 500 mV. Compared to existing techniques, the two schemes allow a 40% voltage reduction with power savings of 85%.

Chishti et al. [2] propose an adaptive technique to increase reliability of cache memories, allowing high tolerance on multi-bit failures that appear on low-voltage operation. The technique sacrifices memory capacity to increase the error-correction capabilities, but unlike previously proposed techniques, it also offers soft and non-persistent error tolerance. Additionally, it does not require self-testing to identify erratic cells in order to isolate them. The MS-ECC design can achieve a 30% supply voltage reduction with 71% power savings and allows configurable ECC capacity by the operating system based on the desired reliability level.

Bacha et al. [3] present a new mechanism for dynamic reduction of voltage margins without reducing the operating frequency. The proposed mechanism does not require additional hardware as it uses existing error correction mechanisms on the chip. By reading their error correction reports, it manages to reduce the operating voltage while keeping the system in safe operation conditions. It covers both core-to-core and dynamic variability cause by the running workload. The proposed solution was prototyped on an Intel Itanium 9560 processor and was tested using SPECjbb2005 and SPEC CPU2000-based workloads. The results report promising power savings that range between 18% to 23%, with marginal performance overheads.

Bacha et al. [4] again rely on error correction mechanisms to reduce operating voltage. Based on the observation that low-voltage errors are deterministic, the paper proposes a hardware mechanism that continuously probes weak cache lines to fine-tune the system's supply voltage. Following an initial calibration test that reveals the weak lines, the mechanism generates simple write-read requests to trigger error-correction and is capable to adapt to voltage noise as well. The proposed mechanism was implemented as proof-of-concept using dedicated firmware that resembles the hardware operation on an Itanium-based server. The solution reports an average of 18% supply voltage reduction and an average of 33% power consumption savings, using a mix set of applications.

Bacha et al. [5] exploit the observation of deterministic error distribution to provide physically unclonable functions (PUF) to support security applications. They use the error distribution of the lowest save voltage supply as an unclonable fingerprint, without the typical requirement of additional dedicated hardware for this purpose. The proposed PUF design offers a low-cost solution for existing processors. The design is reported to be highly tolerant to environmental noise (up to 142%) while maintaining very small misidentification rates (bellow 1ppm). The design was tested on a real system using Itanium processor as well as on simulations. While this study serves a different domain, it highlights the deterministic error behavior on SRAM cells.

Duwe et al. [6] propose an error-pattern transformation scheme that re-arranges erratic bit cells that correspond to uncorrectable error patterns (e.g. beyond the correctable capacity) to correctable error patterns. The proposed method is low-latency and allows the supply voltage to be scaled further that it was previously possible. The adaptive rearranging is guided using the fault patterns detected by self-test. The proposed methodology can reduce the power consumption up to 25.7%, based on simulated modeling that relies on literature SRAM failure probabilities.

There are several papers that explore methods to eliminate the effects of voltage noise, which however lean closer to the scope of T3.2 and thus, only a very brief reference is included.

Gupta et al. [7] and Reddi et al. [8] focus on the prediction of critical parts of benchmarks, in which large voltage noise glitches are likely to occur, leading to malfunctions. In the same context, several studies either in the hardware or in the software level were presented to mitigate the effects of voltage noise [11] [12] [13] [14] [15] or to recover from them after their occurrence [16]. Ketkar et al. [17] and Kim et al. [18] [19] propose methods to maximize voltage droops in single core and multicore chips in order to investigate their worst-case behavior due to the generated voltage noise effects. To conclude with, the characterization studies of commercial chips in off-nominal voltage conditions are limited to [3] [4] [5] [9] [10] [20]. Table 1 summarizes some generic attributes of the referenced works on commercial chips.

Table 1: Summary of studies on commercial chips

Reference	ISA	Processor	Fabrication
[3][4][5]	x86 - IA64 extension	Intel Itanium 9560	32 nm
[9]	CUDA	Nvidia GTX 480,580,680,780	40 nm & 28 nm
[10]	POWER ISA (POWER7)	IBM Power 750	45 nm
[20]	POWER ISA (POWER7+)	IBM Power 780	32 nm
Expected UniServer contribution			
UniServer	ARMv8	Applied Micro X-Gene 2 & X-Gene 3	28 nm & 16 nm

2.2. Experimental Setup

In this section, we present a versatile framework to study the behaviour of multicore CPUs, when they operate under scaled voltage and frequency conditions. This phase is very important to provide accurate results and behaviours of the system, in order to proceed to an extensive on-chip cache analysis. We build a voltage/frequency (V/F) scaling characterization framework on Applied Micro's (APM) X-Gene 2 micro-server family, fabricated in 28nm process technology that consists of eight ARMv8-compliant cores. The proposed infrastructure is fully aligned with all the aforementioned requirements and aims to investigate the limits of a state-of-the-art microprocessor architecture beyond the nominal conditions: it is fast, reliable and easily extensible and replicable. The characterization framework records several different types of deviations from the normal execution as proxies for the effect of voltage and frequency scaling.

As for the results that we have already collected, we also propose a new metric called *Severity Function*, to both quantify the severity and illustrate the scaling of abnormal behaviours due to voltage reduction. The metric's contribution is twofold: (1) to aggregate the results produced by multiple runs, and (2) to quantify a microprocessor's ability to operate beyond nominal conditions. To the best of our knowledge, this is the first study that presents in detail an automated infrastructure for the characterization of ARM-based systems beyond nominal conditions.

2.2.1. Platform Basics

The X-Gene 2 architecture offers high-end processing performance and capabilities. For example, the X-Gene 2 subsystem features the Power Management processor (PMpro) and Scalable Lightweight Intelligent

Management processor (SLIMpro) to enable breakthrough flexibility in power management, resiliency, and end-to-end security for a wide range of applications. The PMpro, a 32-bit dedicated processor provides advanced power management capabilities such as multiple power planes and clock gating, thermal protection circuits, Advanced Configuration Power Interface (ACPI) power management states and external power throttling support. The SLIMpro, a 32-bit dedicated processor monitors system sensors, configure system attributes (e.g. regulate supply voltage, change DRAM refresh rate etc.) and access all error reporting infrastructure, using an integrated I2C controller as the instrumentation interface between the X-Gene 2 Cores and this dedicated processor. SLIMpro can be accessed by the system's running Linux Kernel.

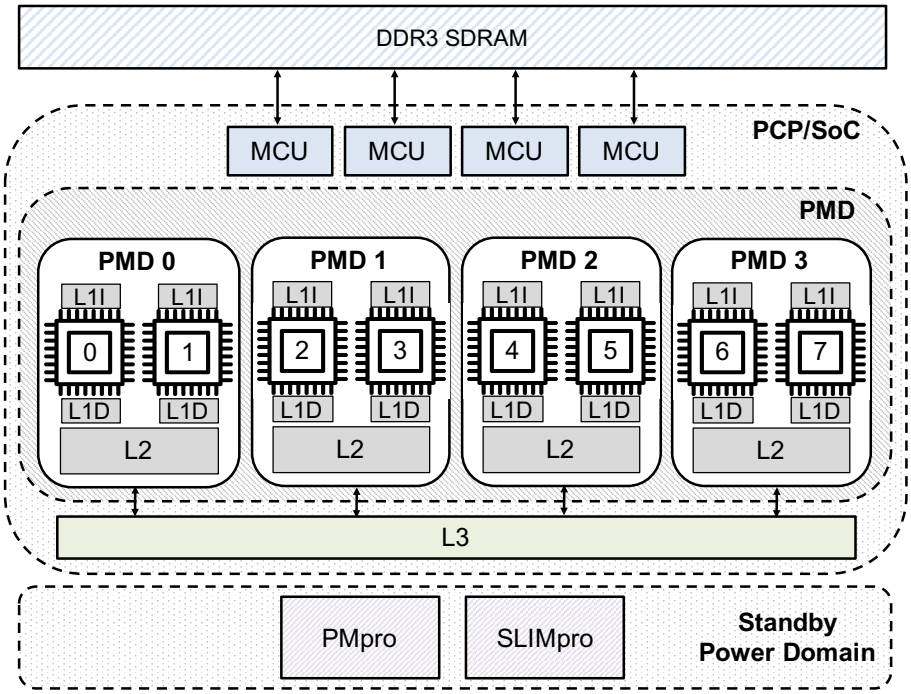

Figure 2: X-Gene 2 micro-server power domains block diagram. The outlines with dashed lines present the independent power domains of the chip.

X-Gene 2 has three independently regulated power domains (as shown in Figure 2 above):

1. PMD (Processor Module): Each PMD contains two ARMv8 cores. Each of the two cores has separate instruction and data caches, while they share a unified L2 cache. The operating voltage of all four PMDs together can change with a granularity of 5mV beginning from 980mV. While PMDs operate at the same voltage, each PMD can operate at a different frequency. The frequency can range from 300MHz up to 2.4GHz at 300MHz steps.
2. PCP (Processor Complex)/SoC: It contains the L3 cache, the DRAM controllers, the central switch and the I/O bridge. The PMDs do not belong to the PCP/SoC power domain. The voltage of the PCP/SoC domain can be independently scaled downwards with a granularity of 5mV beginning from 950mV.
3. Standby Power Domain: This includes the SLIMpro and PMpro microcontrollers and interfaces for I2C buses.

Table 2 summarizes the most important architectural and microarchitectural parameters of the APM X-Gene 2 micro-server that is used in our study. More details can be found in D3.1 - Definition of the UniServer Board, and in official APM's X-Gene 2 User's Manuals and Hardware Specification Sheets.

Table 2: Basic Characteristics of X-Gene 2.

Parameter	Configuration
ISA	ARMv8 (AArch64, AArch32, Thumb)
Pipeline	64-bit OoO (4-issue)
CPU	8 Cores, 2.4GHz
L1 Instruction Cache	32KB per core (Parity Protected) no-write allocate, PIPT
L1 Data Cache	32KB per core (Parity Protected) write-though, no-write allocate, PIPT
L2 cache	256KB per PMD (SECDED Protected) write-back, write allocate (no-write allocate for whole cache line), PIPT
L3 cache	8MB (SECDED Protected)

2.2.2. Framework Description

The primary goals of the proposed framework are: (1) to identify the target system's limits when it operates at scaled voltage and frequency conditions, and (2) to record/log the effects of a program's execution under these conditions. The framework provides the following features; it:

- compares the outcome of the program with the correct output of the program when the system operates in nominal conditions to record Silent Data Corruptions (SDCs),
- monitors the exposed corrected and uncorrected errors from the hardware platform's error reporting mechanisms,
- recognizes when the system is unresponsive and restores it automatically,
- monitors system failures (crash reports, kernel hangs, etc.),
- determines the safe, unsafe and non-operating voltage regions for each application for all frequencies, and
- performs massive repeated executions of the same configuration.

The automated framework (outlined in Figure 3) is easily configurable by the user, and can be embedded to any Linux-based system, with similar voltage and frequency regulation capabilities. As shown in Figure 3, the proposed framework consists of three phases (Initialization, Execution, Parsing).

To completely automate the characterization process, and due to the frequent and unavoidable system crashes that occur when the system operates in reduced voltage levels, we set up a Raspberry Pi board [21] connected externally to the X-Gene 2 board which behaves as a watchdog. The Raspberry is physically connected to both the Serial Port and the Power and Reset buttons of the system board to enable physical access to the system.
We discuss the several challenges that were taken into consideration for a solid development of such a framework.

Safe Data Collection. Given that a system operating beyond nominal conditions often has unexpected behaviours (e.g. file system driver failures), there is the need to correctly identify and store all the essential information in log files (to be subsequently parsed and analysed). The automated framework was developed in such a way to collect and store safely all the necessary information concerning the experiments.

Failure Recognition. Another challenge is to recognize and distinguish the system and program crashes or hangs. This is a very important feature for the Parsing Phase to easily identify and classify the final results, with the most possible distinct information concerning the characterization.

Reliable Cores Setup. Another major challenge we also face is that the characterization of a system is performed primarily by using properly chosen programs in order to provide diverse behaviours and expose all the potential deviations from nominal conditions. It is thus important to run the selected benchmarks in *reliable cores setup*. This means that the cores, where the benchmark runs, must be isolated and unaffected from the other active processes of the kernel in order to capture only the effects of the desired benchmark.

Iterative Execution. The non-deterministic behaviour of the characterization results due to several microarchitectural features makes necessary to repeat the experiments multiple times with the same configuration to eliminate the probability of misleading results.

In the following subsections, we analyse each of these functionalities grouped in the 3 distinct phases of the framework's execution (Initialization, Execution, Parsing), and describe their detailed implementation and how these challenges were overcome.

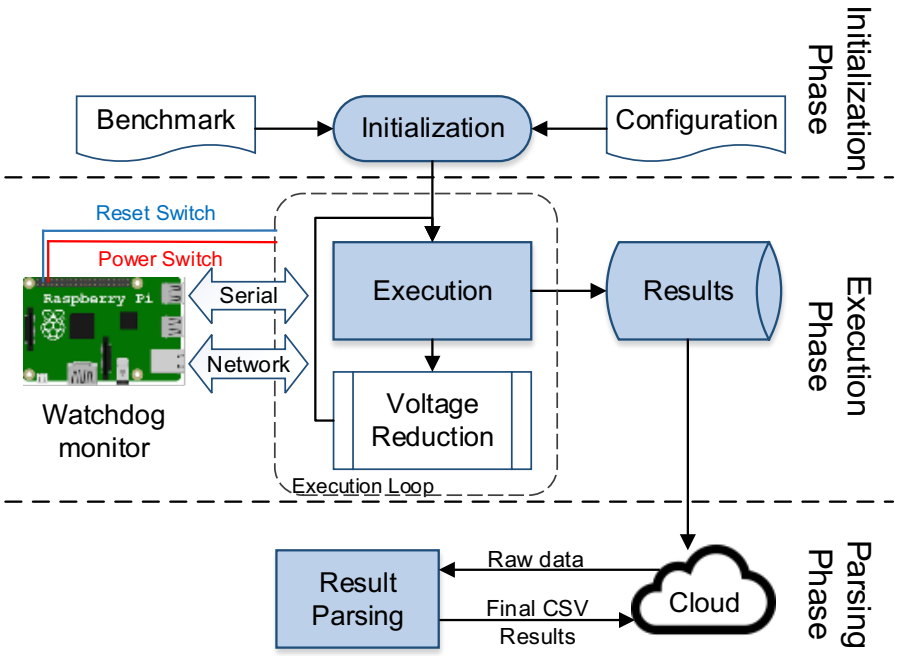

Figure 3: Framework Layout.

2.2.2.1 Initialization Phase

During the *initialization phase*, a user can declare a benchmark list with any input dataset to run in any desirable characterization setup. The characterization setup includes the voltage and frequency (V/F) values under which the experiment will take place and the cores where the benchmark will be executed; this can be an individual core, a pair of cores in the same PMD, or all of the available eight cores in the microprocessor. The characterization setup depends on the power domains supported by the chip, but our framework is easily extensible to support the power domain features of different CPU chips.

This phase is in charge of setting the voltage and frequency ranges, the initial voltage and frequency values, with which the characterization begins, and to prepare the benchmarks: their required files, inputs, outputs, as well as the directory tree where the necessary logs will be stored. This phase is performed at the beginning of the characterization and each time the system is restored by the Raspberry (for example, after a system crash) in order to proceed to the next run until the entire Execution Phase finishes. Each time the system is restored, this phase restores the initial user's desired setup and recognizes where and when the characterization has been previously stopped. This step is essential for the characterization to proceed sequentially according to user's choice, and to complete the whole Execution Phase.

This phase is also responsible to overcome the challenge of *reliable cores setup* that is responsible to ensure the correctness and integrity of our results. The benchmark must run in an "as bare as possible" system without the interference of any other running process. Therefore, reliable cores setup is twofold: first,

it recognizes these cores or group of cores that are not currently under characterization, and migrates all currently running processes (except for the benchmark) to a completely different core. The migration of system processes is required to isolate the execution of the desired benchmark from all other active processes, as much as possible. Second, given that all the PMDs in the studied system are in the same power domain, they always have the same voltage value (in case this does not hold in a different microarchitecture the proposed framework can be adapted). This means that even though there are several processes running on different cores (not in the core(s) under characterization), they have the same probability to affect an unreliable operation while reducing the voltage.

On the other hand, each individual PMD can have different frequency, so we leverage the combination of V/F states in order to set the core under characterization to the desired frequency, and all other cores to the minimum available frequency in order to ensure that an unreliable operation is due to the benchmark's execution only. When for example the characterization takes place in the PMD0 (meaning that the benchmark runs in PMD0; cores 0 and 1), the PMD0 is set to the pre-defined by the user frequency, and all the other PMDs are set to the minimum available frequency (300MHz in our case). Thus, all the running processes, except for the benchmark, are executed to the reliable cores setup.

In our setup, we also use a stripped/lightweight Linux Kernel to diminish the unnecessary kernel daemons that the majority of well-known Linux Distributions provide. Thus, the system's running processes and the common power domain of all PMDs, neither affect the benchmarks execution nor can contribute to a system's failure or error event.

2.2.2.2 Execution Phase

After the characterization setup is defined, the automated *Execution Phase* begins. The Execution Phase consists of multiple runs of the same benchmark, each one representing the execution of the benchmark with a pre-defined characterization setup. The set of all the characterization runs running the same benchmark with different characterization setups represents a *campaign*. After the initialization phase, the framework enters the Execution Phase, in which all runs take place. The runs are executed according to user's configuration, while the framework reduces the voltage with a step defined by the user in the initialization phase. For each run, the framework collects and stores the necessary logs at a safe place externally to the system under characterization, which will be then used by the parsing phase.

The logged information includes: the output of the benchmark at each execution, the corrected and uncorrected errors (if any) collected by the Linux EDAC Driver [22], as well as the errors' localization (L1 or L2 cache, DRAM, etc.), and several failures, such as benchmark crash, kernel hangs, and system unresponsiveness. The framework can distinguish these types of failures and keep logging about them to be parsed later by the parsing phase. Benchmark crashes can be distinguished by monitoring the benchmark's exit status. On the other hand, to identify the kernel hangs and system unresponsiveness, during this phase the framework notifies the Raspberry when the execution is about to start and also when the execution finishes.

In the meantime, the Raspberry starts pinging the system to check its responsiveness. If the Raspberry does not receive a completion notification (hang) in the given time (we defined as timeout condition a 2 times the normal execution time of the benchmark) or the X-Gene 2 turns completely unresponsive (ping is not responding), the Raspberry sends a signal to the Power Off button on the board, and the system resets. After that, the Raspberry is also responsible to check when the system is up again, and sends a signal to restart the experiments. These decisions contribute to the *Failure Recognition* challenge.

During the experiments, some Linux tasks or the kernel may hang. To identify these cases, we use an inherent feature of the Linux kernel to periodically detect these tasks by enabling the flag "*hung_task_panic*" [22]. Therefore, if the kernel itself recognizes a process hang, it will immediately reset the system, so there is no need for the Raspberry to wait until the timeout. In this way, we also contribute to the *Failure Recognition* challenge and accelerate the reset procedure and the entire characterization.

Note that, in order to isolate the framework's execution from the core(s) under characterization, the operations of the framework are also performed in *Reliable Cores Setup*. However, when there are operations of the framework, such as the organization of log files during the benchmark's execution that are an integral part of the framework, and thus, they must run in the core(s) under characterization, these operations are performed after the benchmark's execution in the nominal conditions. This is the way to

ensure that any logging information will be stored correctly and no information will be lost or changed due to the unstable system conditions, and thus, to overcome the *Safe Data Collection* challenge.

2.2.2.3 Parsing Phase

In the last step of our framework, all the log files that are stored during the Execution Phase are parsed in order to provide a fine-grained classification of the effects observed for each characterization run. Note that, each run is correlated to a specific benchmark and characterization setup. The categories that are used for our classification are summarized in Table 3, but the parser can be easily extended according to the user's needs. For instance, the parser can also report the exact location that the correctable errors occurred (e.g. the cache level, the memory, etc.) using the logging information provided by the Execution Phase.

Table 3: Experimental Effect Classification.

Effect	Description
NO (Normal Operation)	The benchmark was successfully completed without any indications of failure.
SDC (Silent Data Corruption)	The benchmark was successfully completed, but a mismatch between the program output and the correct output[1] was observed.
CE (Corrected Error)	Errors were detected and corrected by the hardware.
UE (Uncorrected Error)	Errors were detected, but not corrected by the hardware.
AC (Application Crash)	The application process was not terminated normally (the exit value of the process was different than zero).
SC (System Crash)	The system was unresponsive; meaning that the X-Gene 2 is not responding to pings or the timeout limit was reached.

Note that each characterization run can manifest multiple effects. For instance, in a run both SDC and CE can be observed; thus, both of them should be reported by the parser for this run. Furthermore, the parser can report all the information collected during multiple campaigns of the same benchmark. The characterization runs with the same configuration setup of different campaigns may also have different effects with different severity. For instance, let us assume two runs with the same characterization setup of two different campaigns. After the parsing, the first run finally revealed some CEs, and the second run was classified as SDC. To quantify the criticality of the effects of different experimental runs of different campaigns with the same setup, we define the "severity function" S_v, where v is the voltage value, as presented below:

$$S_v = W_{SDC} \cdot \frac{SDC}{N} + W_{CE} \cdot \frac{CE}{N} + W_{UE} \cdot \frac{UE}{N} + W_{AC} \cdot \frac{AC}{N} + W_{SC} \cdot \frac{SC}{N}$$

In this function, the parameters *SDC*, *CE*, *UE*, *AC* and *SC* can take the values from 0 to *N* (*N* is the number of runs at each voltage level), and represent the times that this effect appears to these runs. Parameters W_{SDC}, W_{CE}, W_{UE}, W_{AC} and W_{SC} represent "weights" that can be set to characterize the severity of each effect of Table 3. The higher the weight, the more critical the effect is considered by our function. For the purpose of this work, we defined the values presented in Table 4 as values for our severity function (any values for the weights can be used). These Weights represent in descending order the "intuitive" severity of each event. For example, an SDC is of higher severity than a CE or a UE. Note that the maximum value that can be achieved by this function concerning these weights is 19, which represents the case of a crash (that has value a equal to 16), followed by some corrected (value = 1) and uncorrected errors (value = 2).

[1] A reference output, provided by a normal execution in nominal voltage conditions.

Table 4: Weights Used in our Experiments.

Weight	Value
W_{SC}	16
W_{AC}	8
W_{SDC}	4
W_{UE}	2
W_{CE}	1
W_{NO}	0

At the end of the parsing step, all the collected results concerning the characterization (according to Table 3) and the severity function of each run are reported in CSV files.

2.3. Results

We present indicative examples of the results that can be generated by the characterization framework. We present our first findings in three different chips: TTT, TFF, and TSS. The TTT part is the "normal" part. The TFF is the fast corner part, which has high leakage but at the same time can operate at higher frequency. The TSS part is also corner part which has low leakage and works at lower frequency.

2.3.1. Results Provided by TTT Chip at 2.4GHz

The framework can reveal for each core of the CPU and the evaluated program three different regions of operation when we reduce the voltage. These are the *safe* and *unsafe* operating regions and the region in which the system is unresponsive. For our experiments, we used the benchmarks: *Linpack*, which is a widely-used high-performance benchmark [23] and 12 benchmarks from the SPEC CPU2006 benchmark suite [24] with the reference and test input dataset, as shown in Table 5 below. All of them ran on a single core, while the remaining six cores are reliable (one of the cores is paired with the "unreliable" and is thus also unreliable; see explanation in section 2.2.2). In order to present the non-deterministic behaviour of such experiments, we ran each campaign three times. Note that, concerning the statistical safe analysis, it is needed more than three iterative executions. Our results shown below, present an example of three iterative executions in order to show the non-deterministic behaviour of these experiments and to present some examples of the severity function.

Figure 4, Figure 5, and Figure 6 present the three campaigns for the Linpack, hmmer and gcc benchmark (1^{st}, 2^{nd}, 3^{rd}), respectively, in the case that are executed in each individual core running at 2.4GHz, while the rest of the PMDs operate on the reliable cores setup. In both benchmarks, we can notice the three regions of operation according to the collected results. The regions are:

- *Safe* region (green): The characterization runs that correspond to this zone had a NO (normal operation) without SDCs, errors or crashes.
- *Unsafe* region (yellow): The characterization runs that correspond to this zone manifested an abnormal behavior (SDC, CE, UE, AC) but not a system crash.
- *Crash* region (grey): This region is created by the first characterization runs that lead to a system crash.

Table 5: Benchmarks that were used in the characterization phase

Benchmark	Dataset	Chip
Linpack	-	TTT
hmmer	ref	TTT
gcc	ref	TTT
bwaves	test	TTT, TSS
cactusADM	test	TTT
dealII	test	TTT
gromacs	test	TTT
leslie3d	test	TTT
mcf	test	TTT, TSS
milc	test	TTT
namd	test	TTT
soplex	test	TTT
zeusmp	test	TTT, TSS

We observe significant variation among the three runs, and also significant core-to-core static variation when they execute the same benchmark. From these results, we observe that cores 4 and 5 of the particular chip we used are more robust than the others. Moreover, it is important to notice the width of the *Safe* region in the three benchmarks that is up to 11.2% lower than the nominal voltage value (980mV). This reduction of voltage from the nominal value corresponds to power gains up to more than 21%. Finally, the width of the *Unsafe* region ranges from 0mV up to 40mV. This illustrates that with the development of appropriate mitigation techniques the power gain can reach the 28.3%.

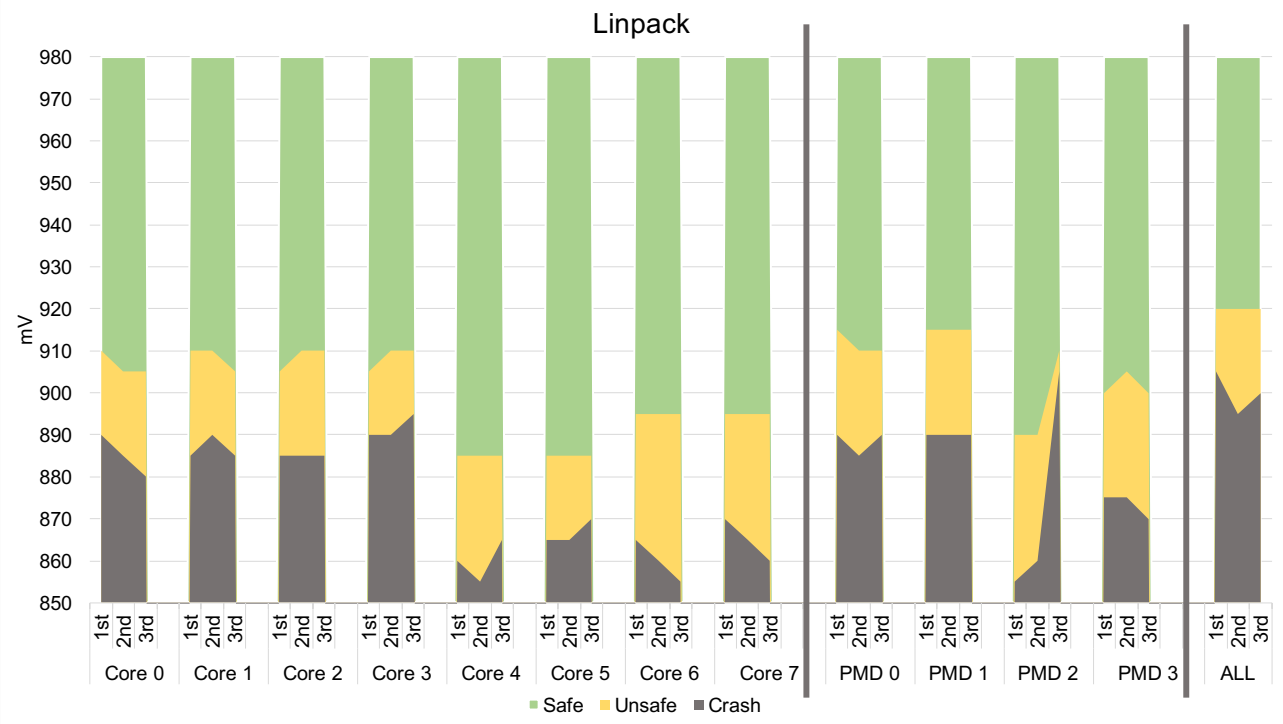

Figure 4: Linpack benchmark.

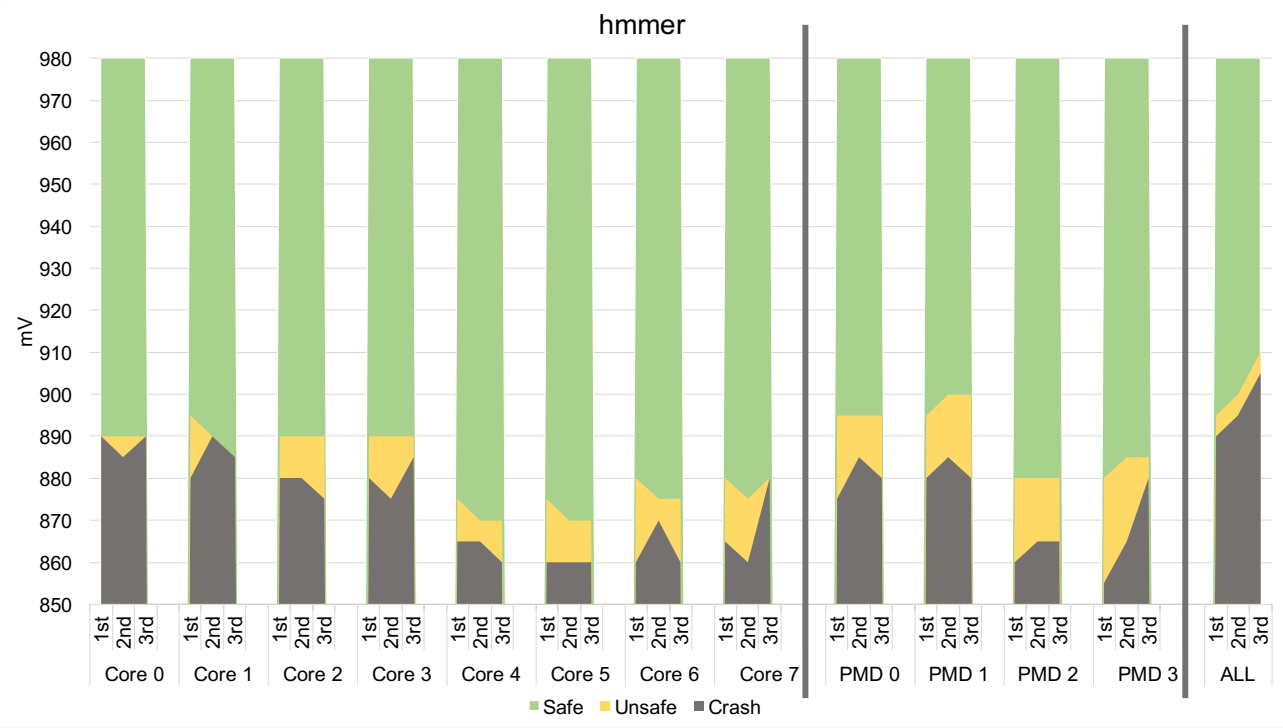

Figure 5: hmmer benchmark.

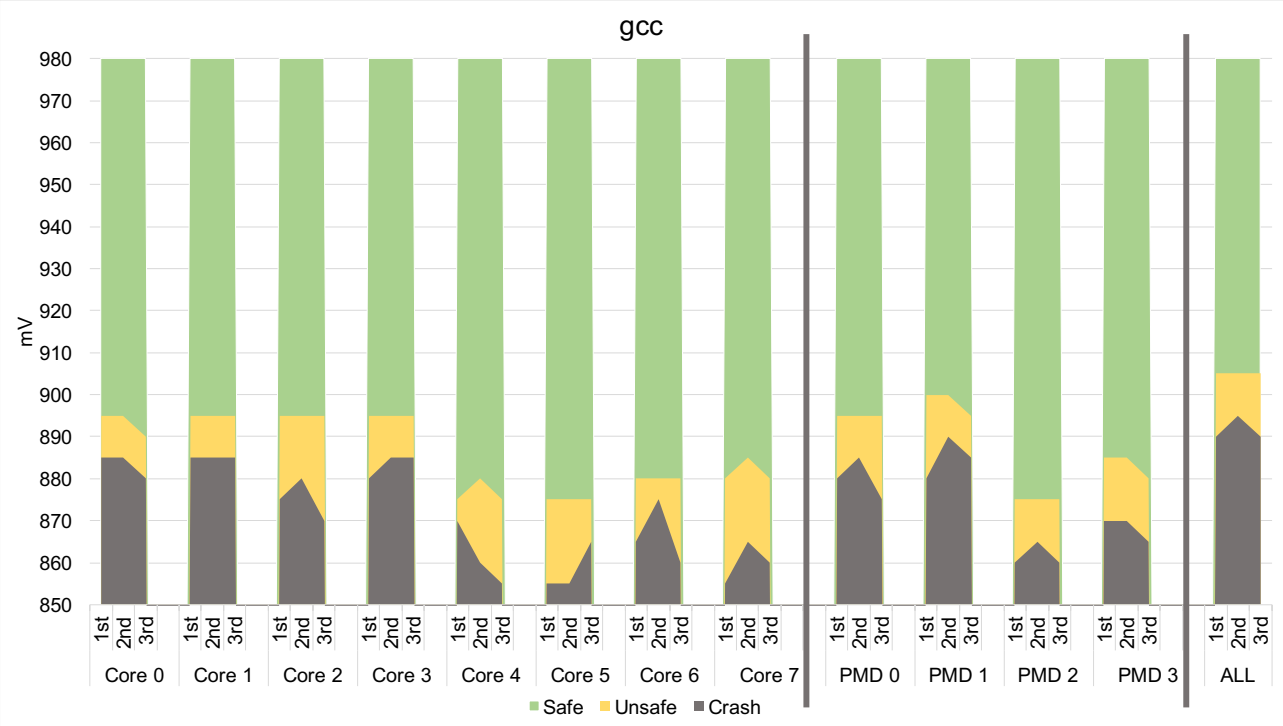

Figure 6: gcc benchmark.

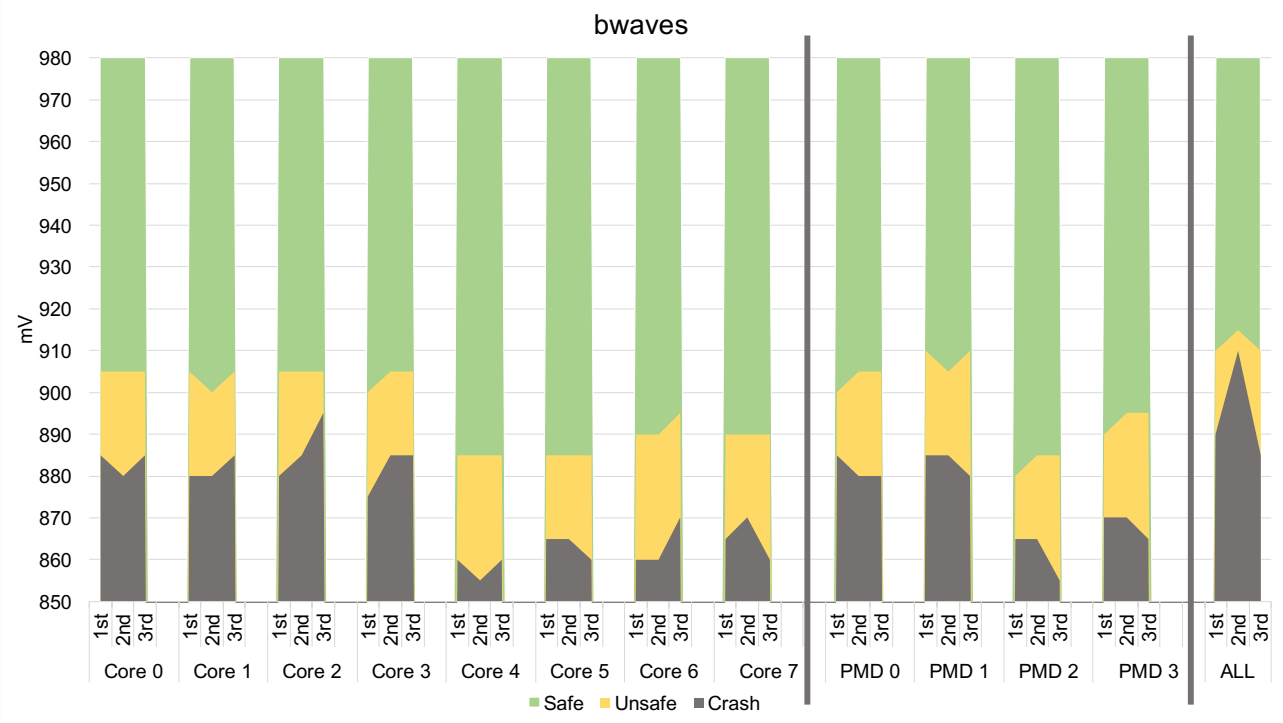

Figure 7: bwaves benchmark.

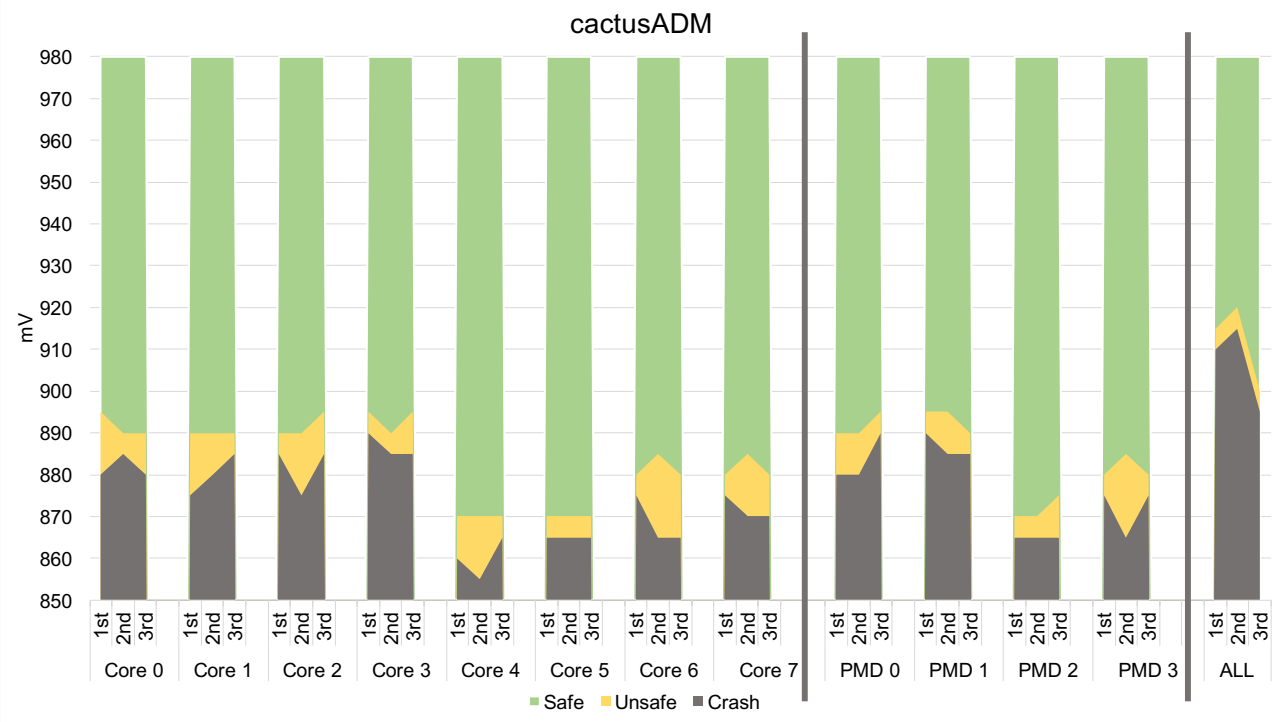

Figure 8: cactusADM benchmark.

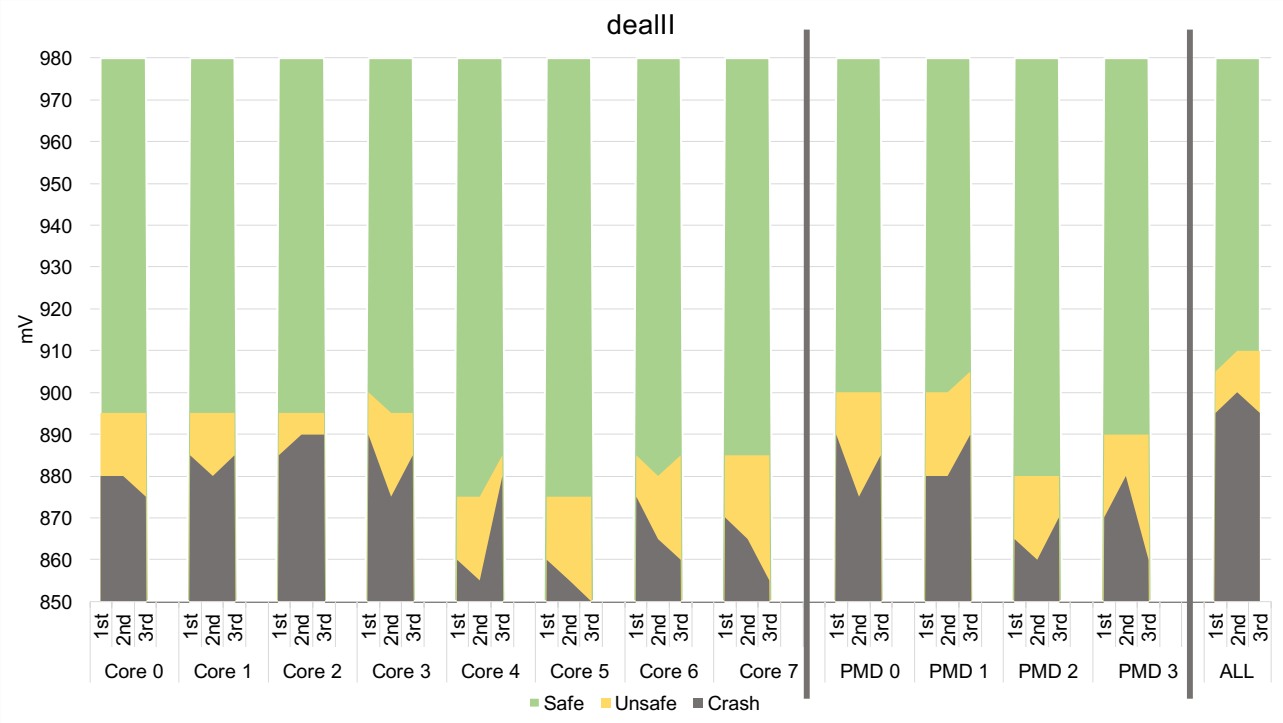

Figure 9: dealII benchmark.

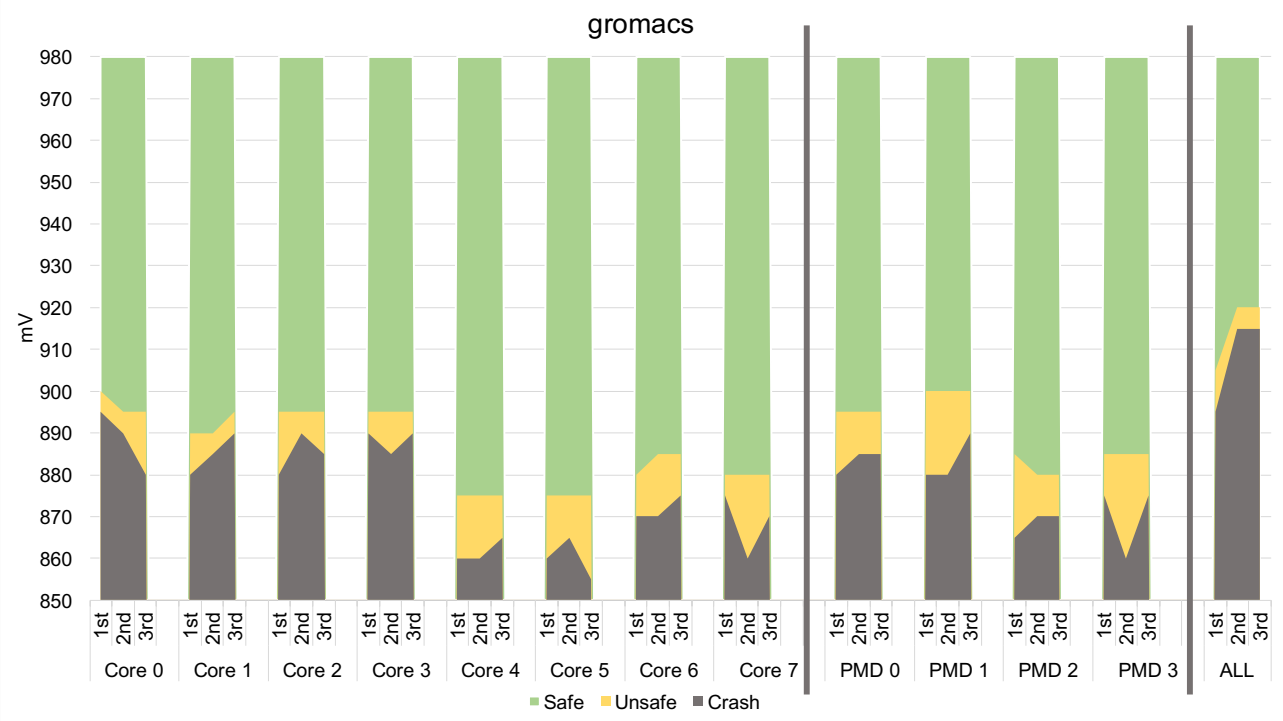

Figure 10: gromacs benchmark.

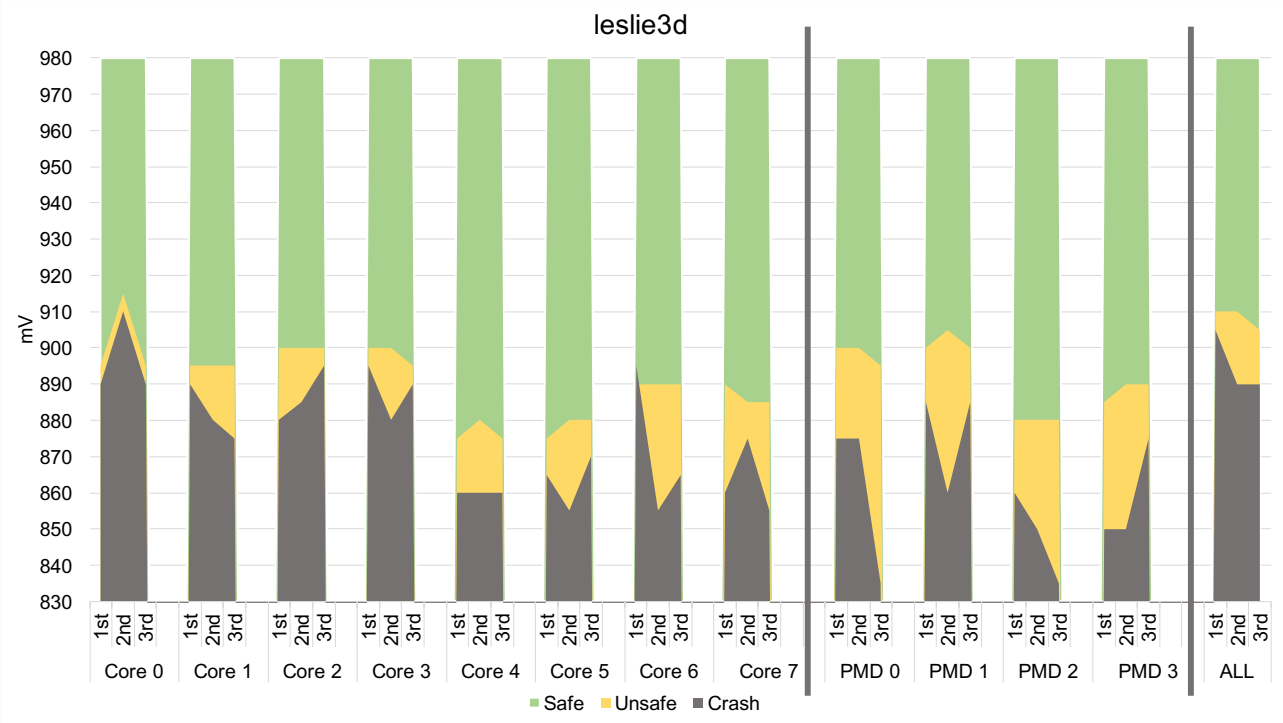

Figure 11: leslie3d benchmark.

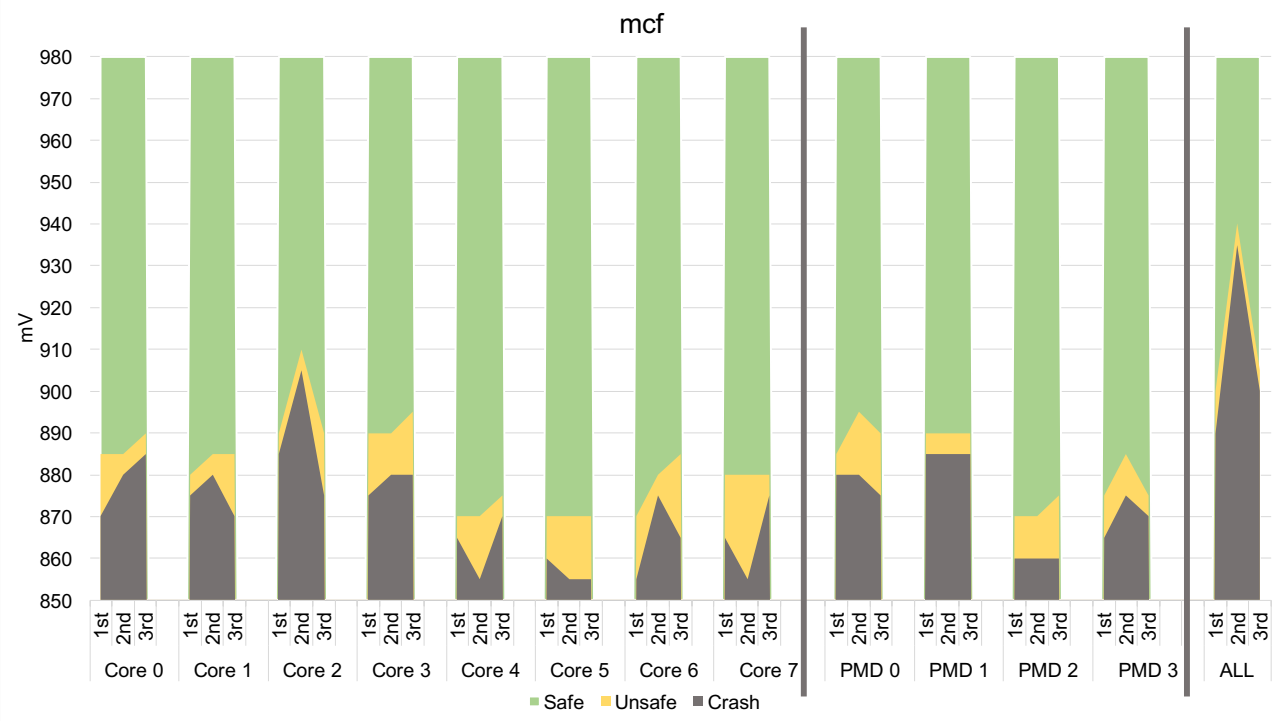

Figure 12: mcf benchmark.

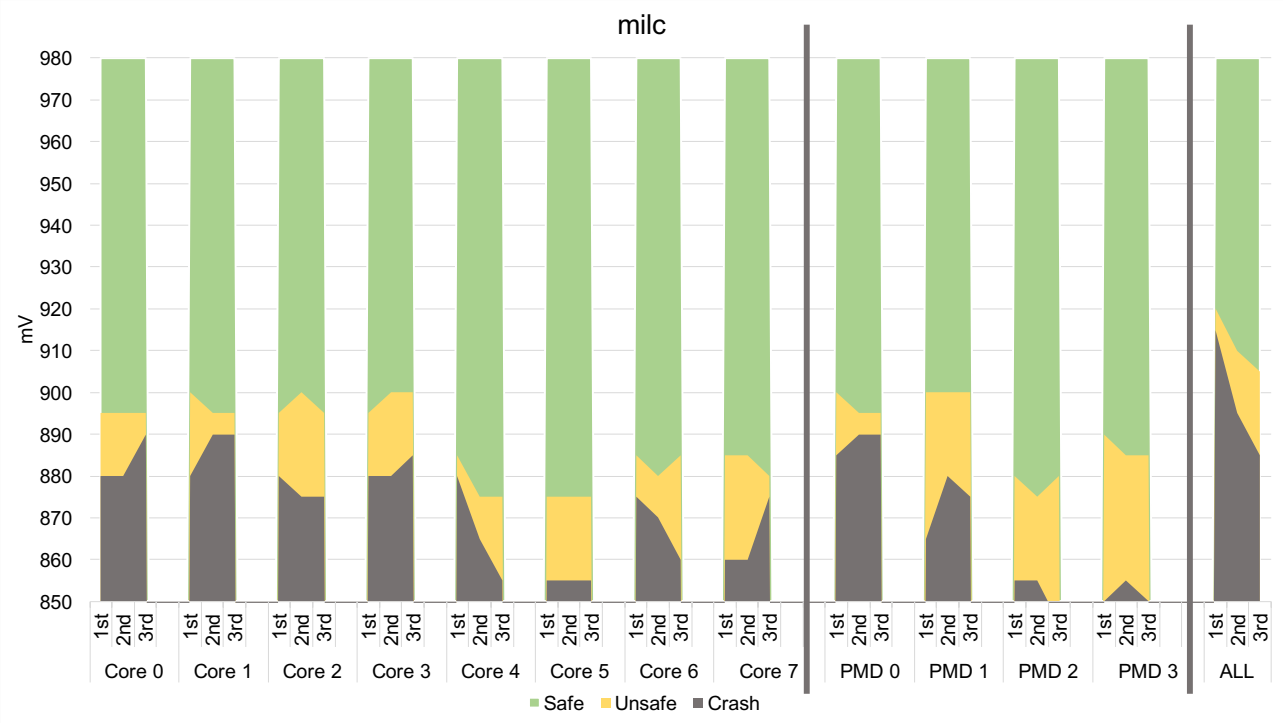

Figure 13: milc benchmark.

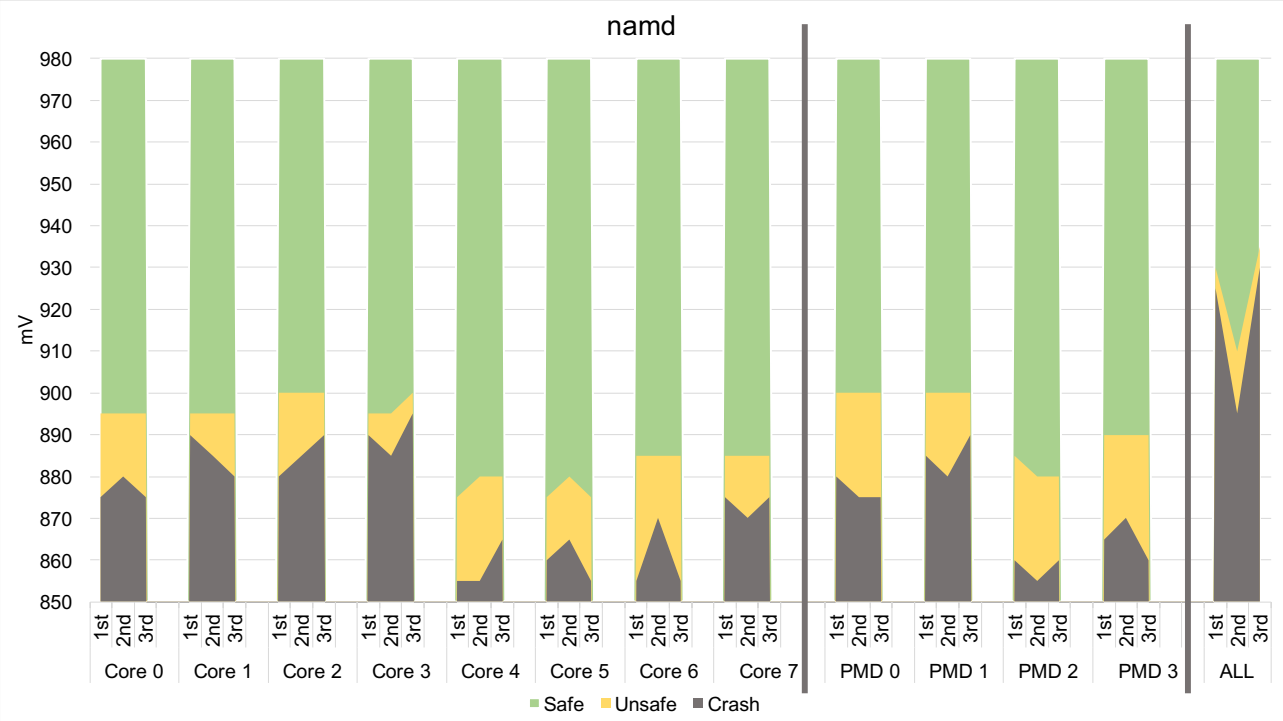

Figure 14: namd benchmark.

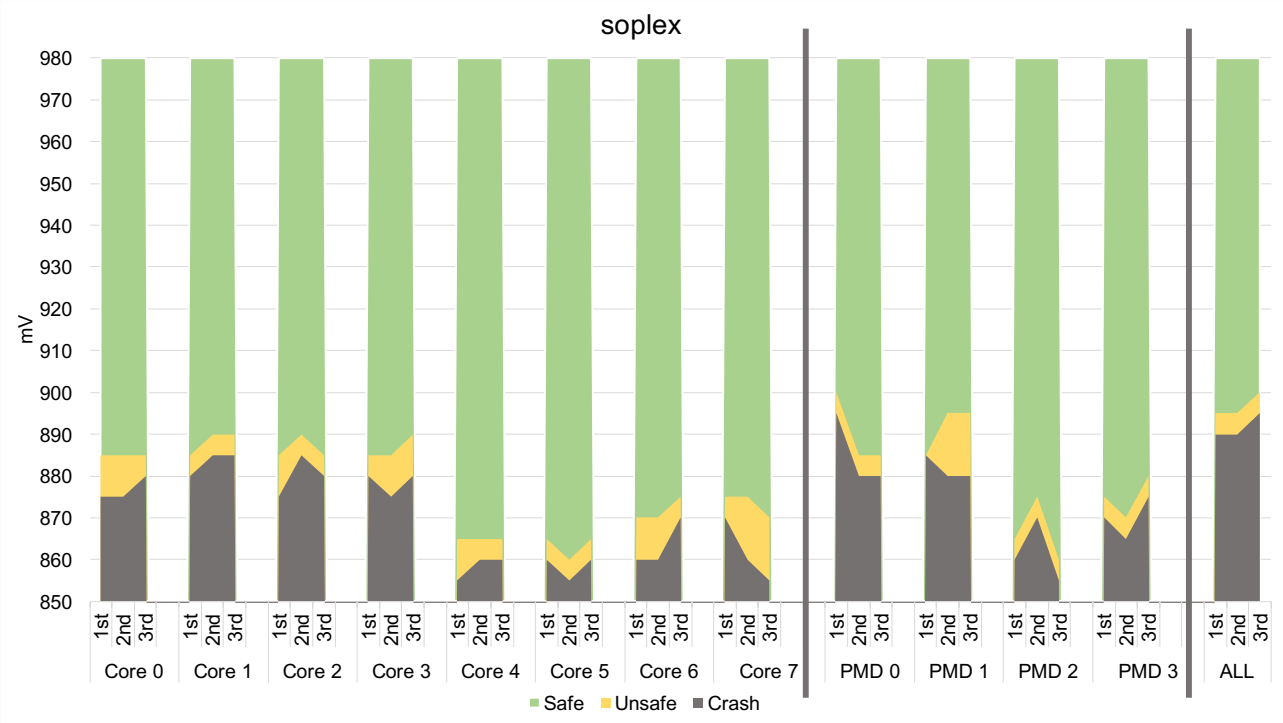

Figure 15: soplex benchmark.

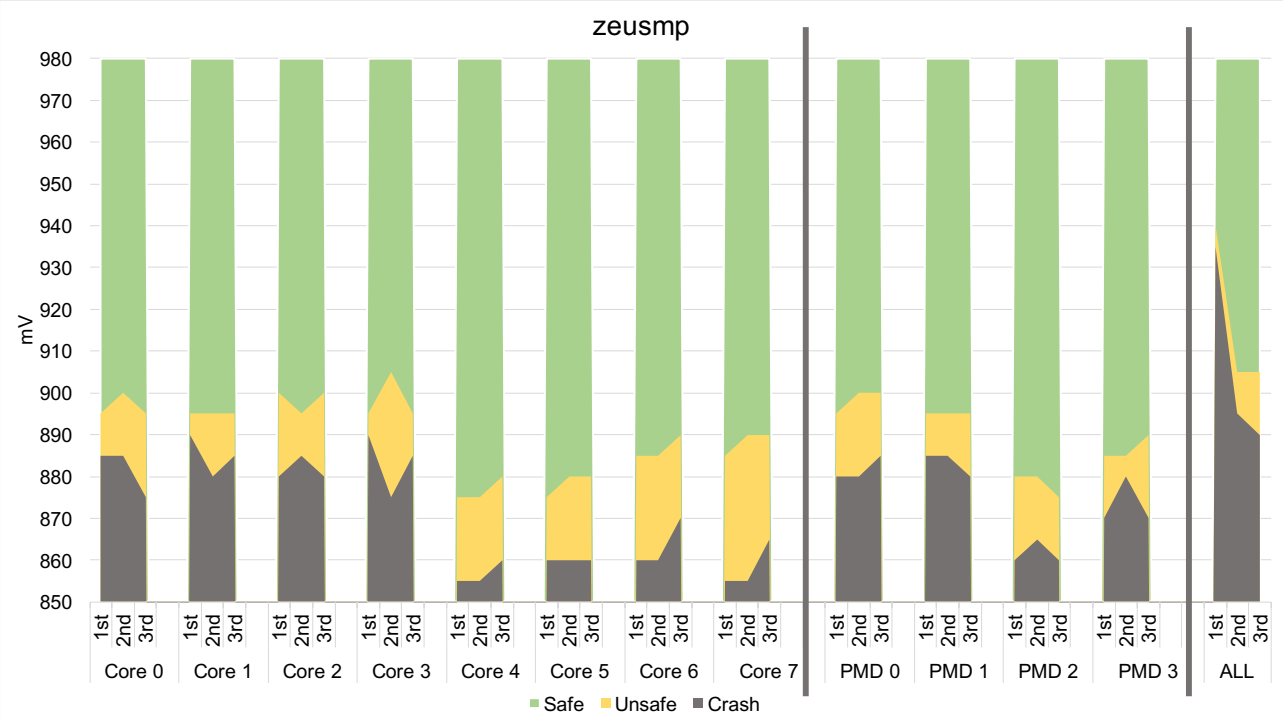

Figure 16: zeusmp benchmark.

Moreover, for the same characterization runs we used the severity function presented in subsection 2.2.2.3 to present the scaling of the effects and their severity in the reduced voltage margins. Figure 17, Figure 19 and Figure 21 represent the case of running the benchmark in individual cores. Figure 18, Figure 20, and

Figure 22 present the case that we run the benchmark in the four PMDs, and finally Figure 43 illustrates the case that the benchmarks run simultaneously in all the cores. In this representation, colour brightness is proportional to severity, with light colour translated into low severity and dark colour translated into high severity. The worst-case severity in our function equals to 19, which corresponds to a combination of CEs, UEs and SC (1 + 2 + 16). While reducing the voltage margins, we observe that the instability increases (the colour becomes darker), until the darkest colour (crash point), which indicates that the system cannot operate in such voltage margins.

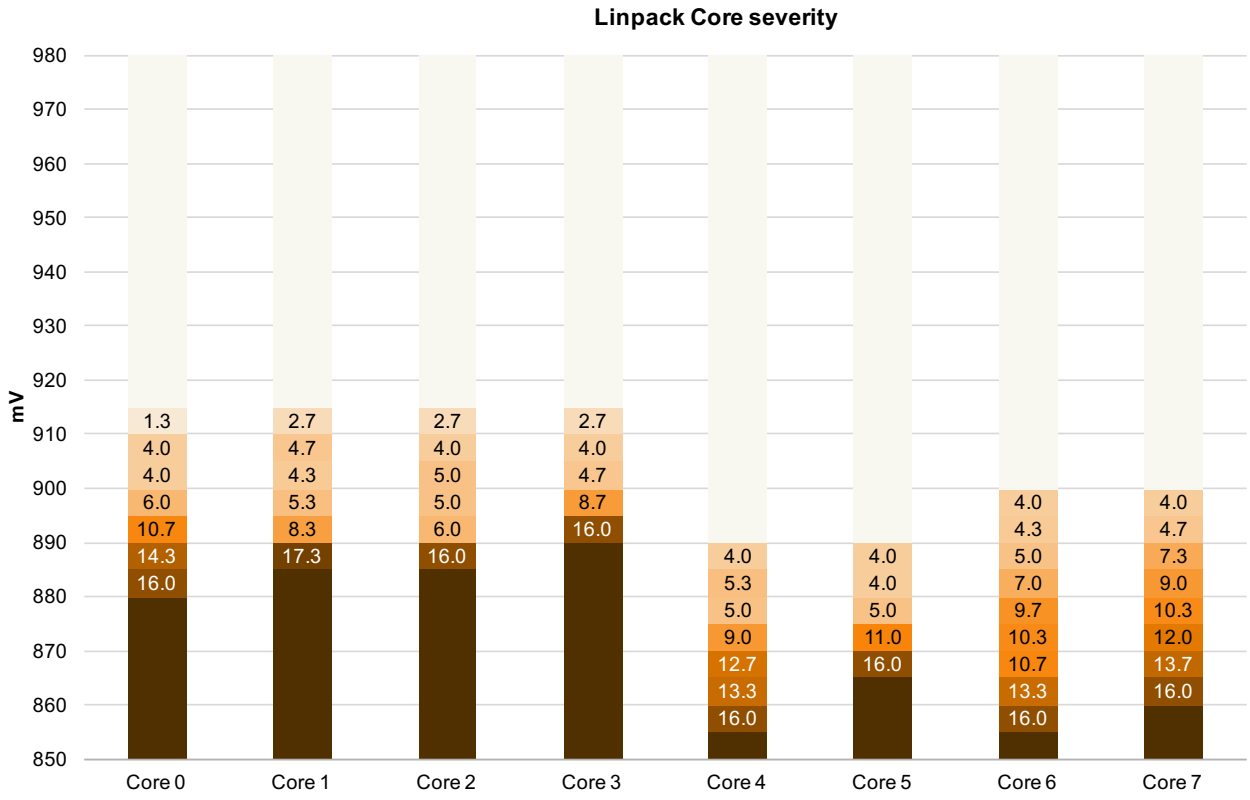

Figure 17: Linpack benchmark – Core severity scaling.

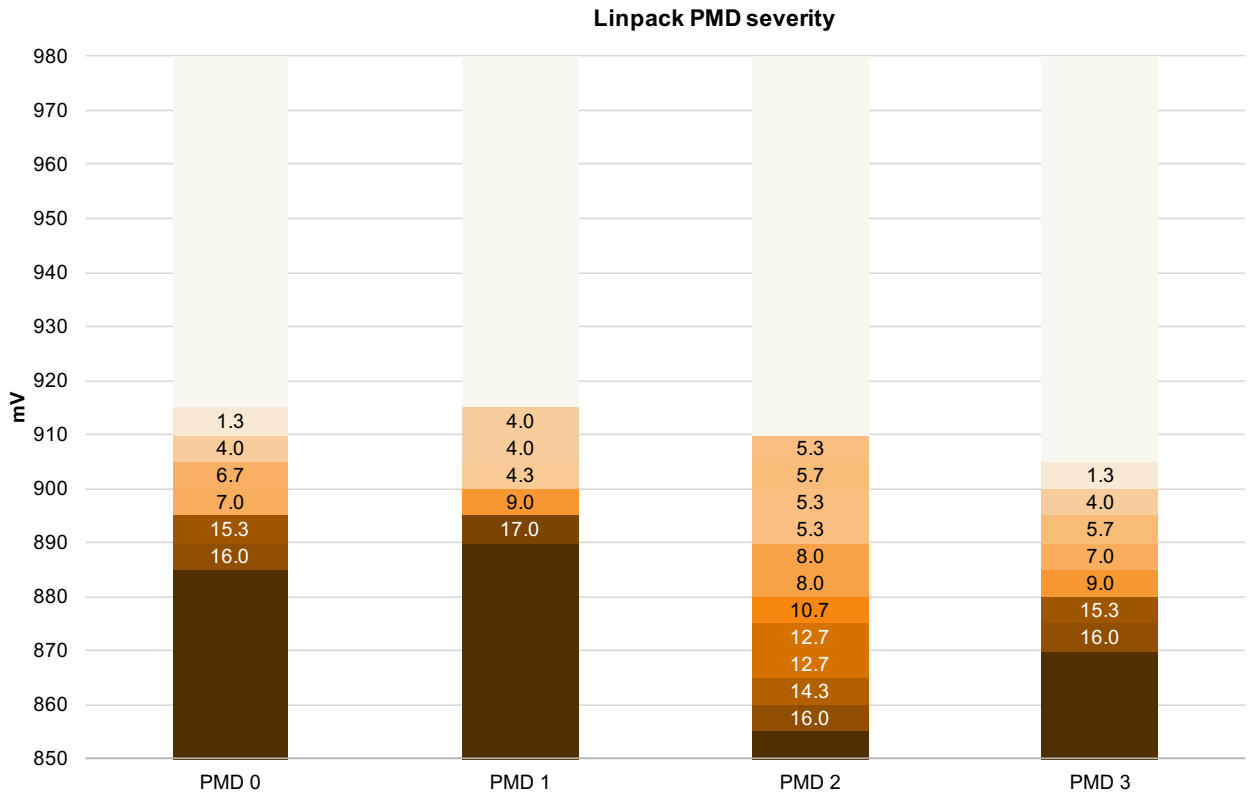

Figure 18: Linpack benchmark – PMD severity scaling.

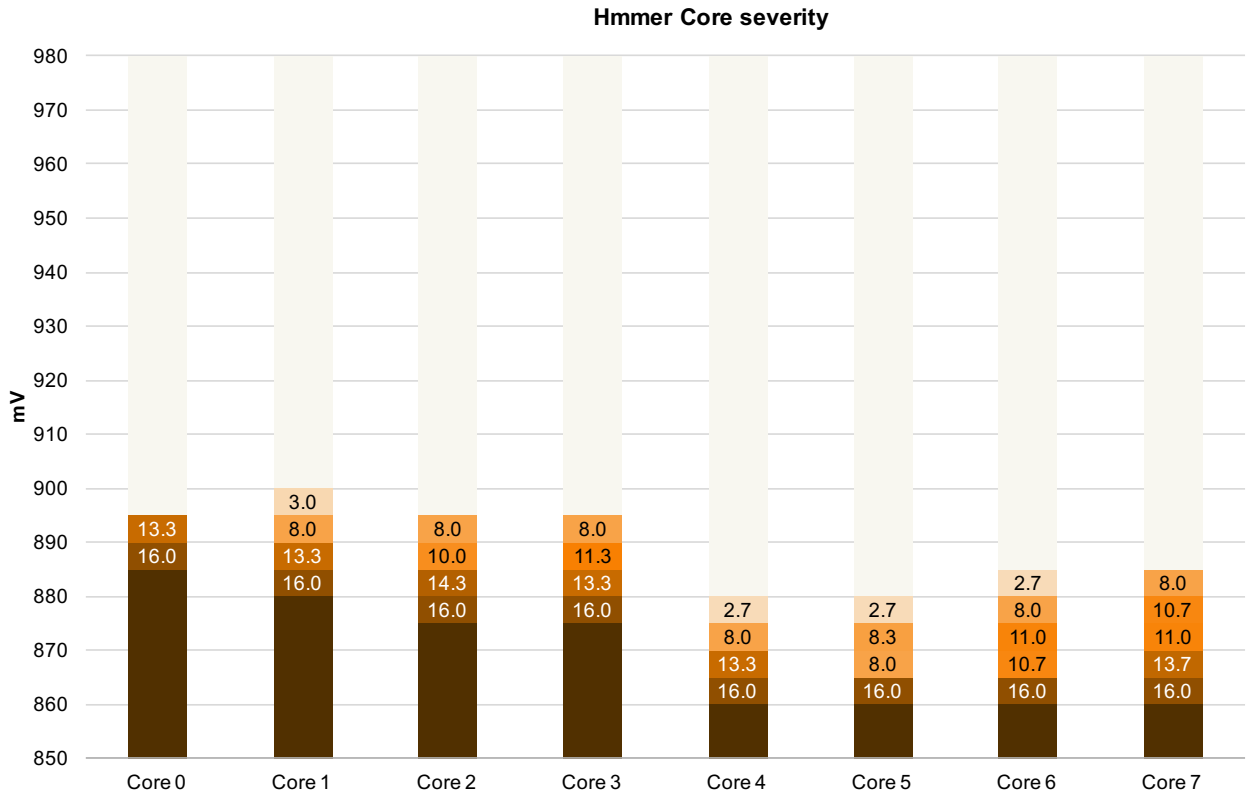

Figure 19: hmmer benchmark – Core severity scaling.

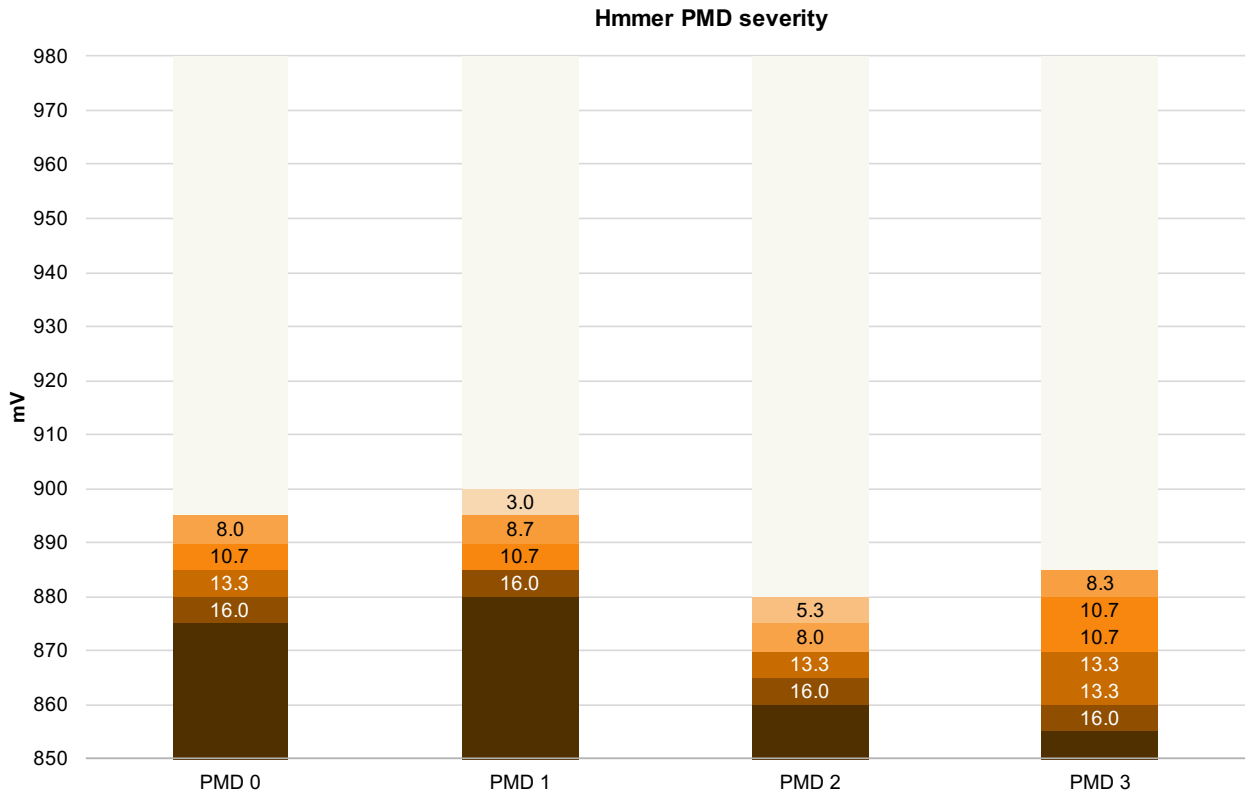

Figure 20: hmmer benchmark – PMD severity scaling.

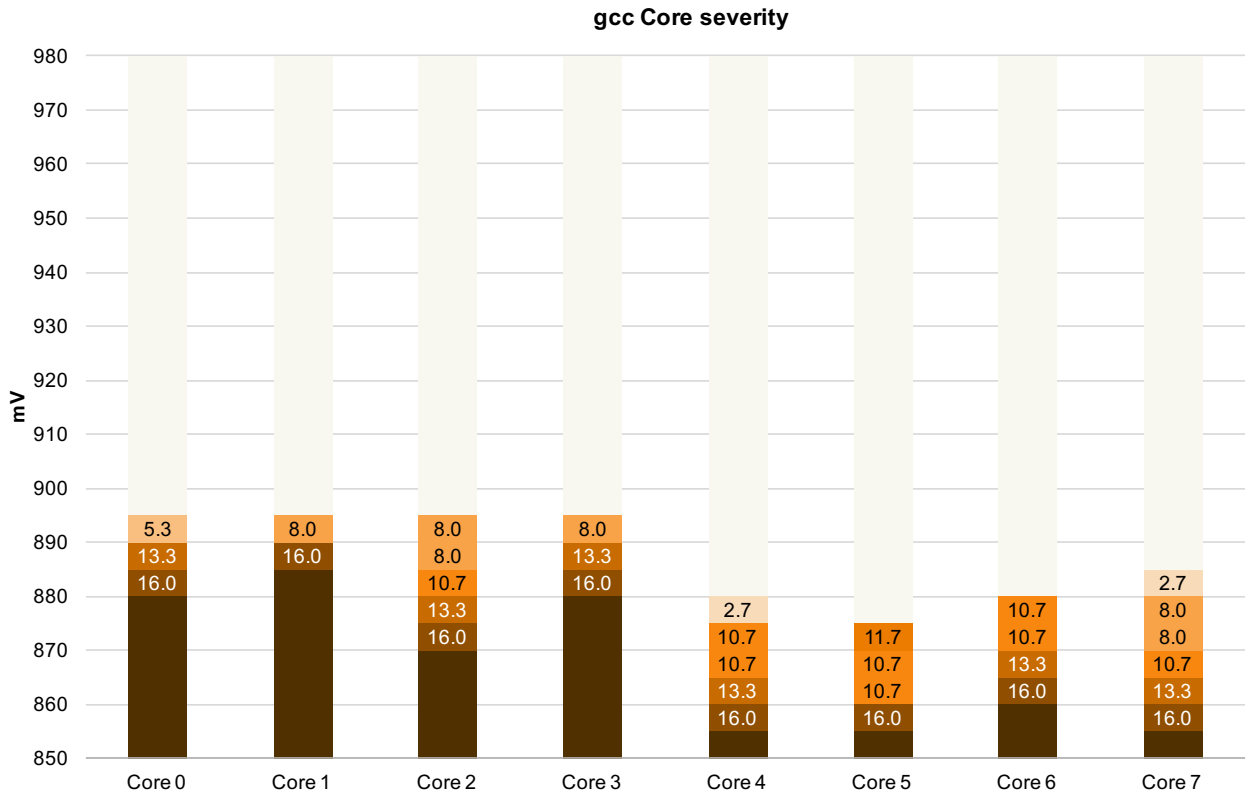

Figure 21: Gcc benchmark – Core severity scaling.

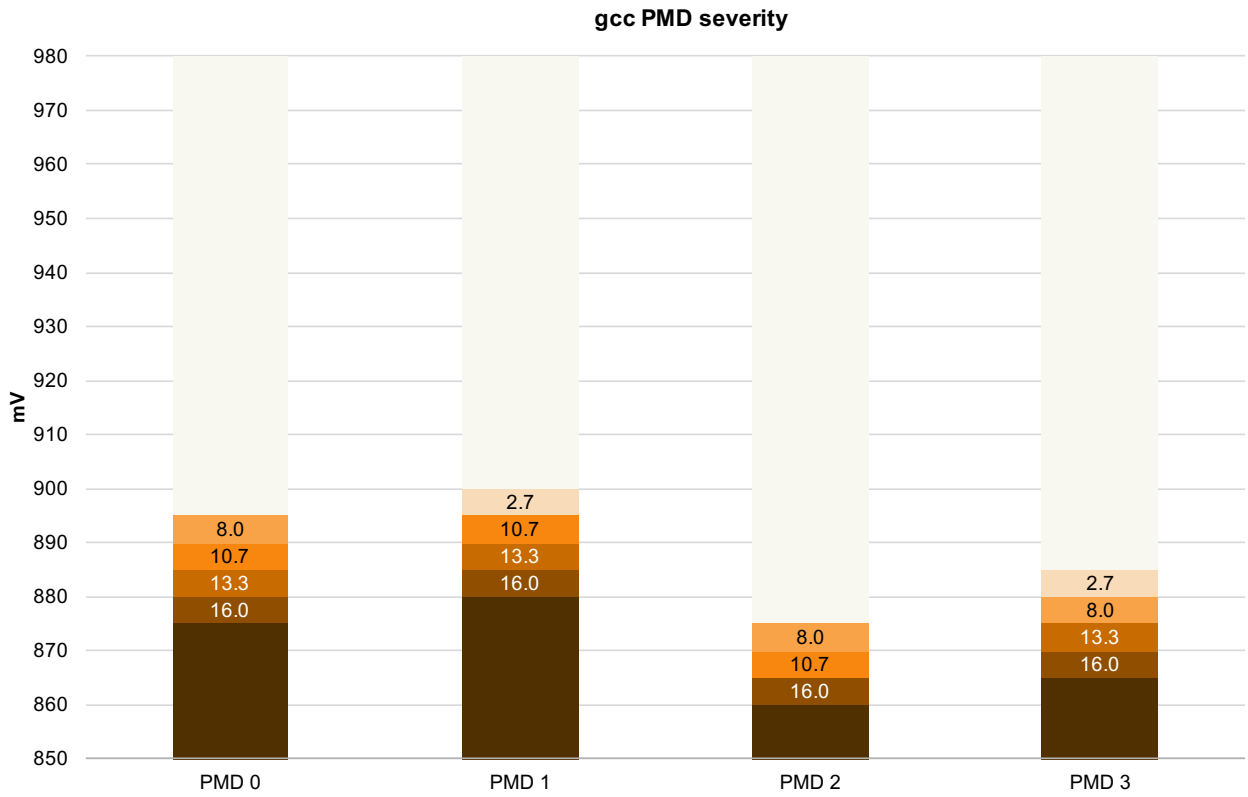

Figure 22: Gcc benchmark – PMD severity scaling.

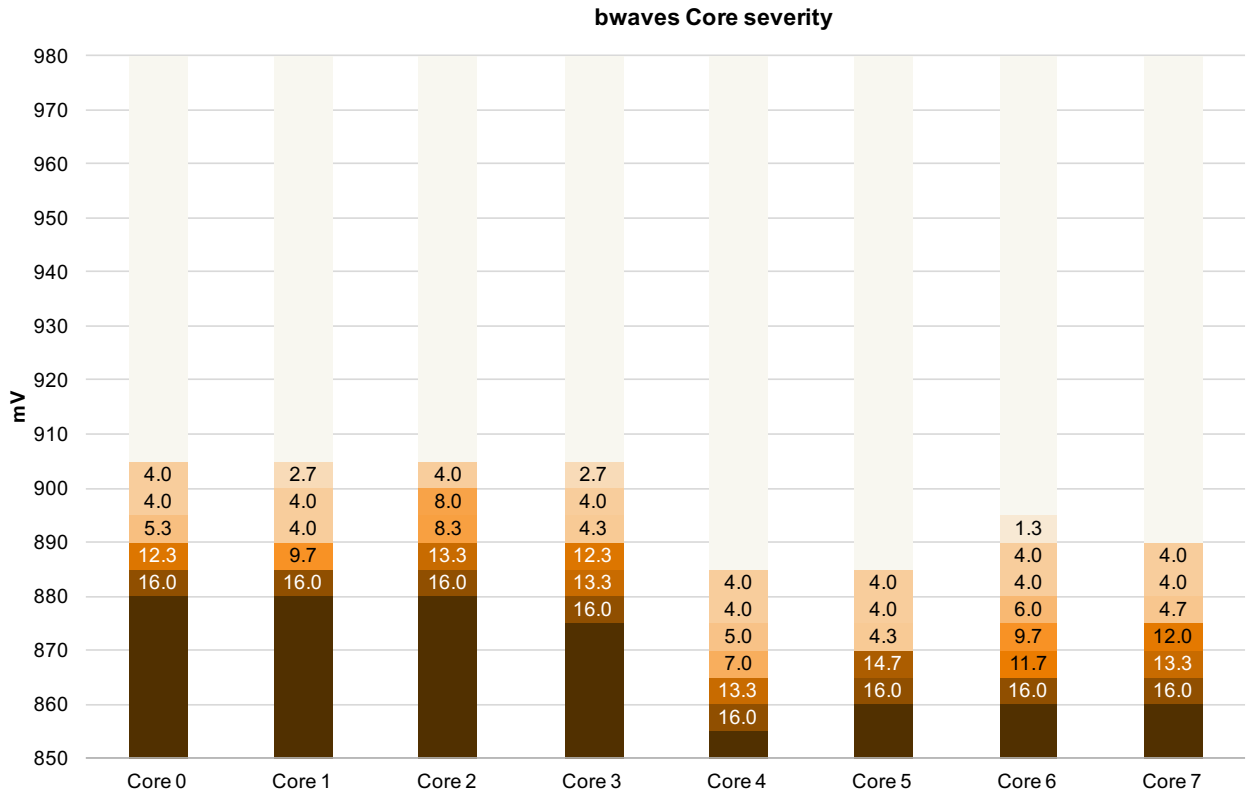

Figure 23: bwaves benchmark – Core severity scaling.

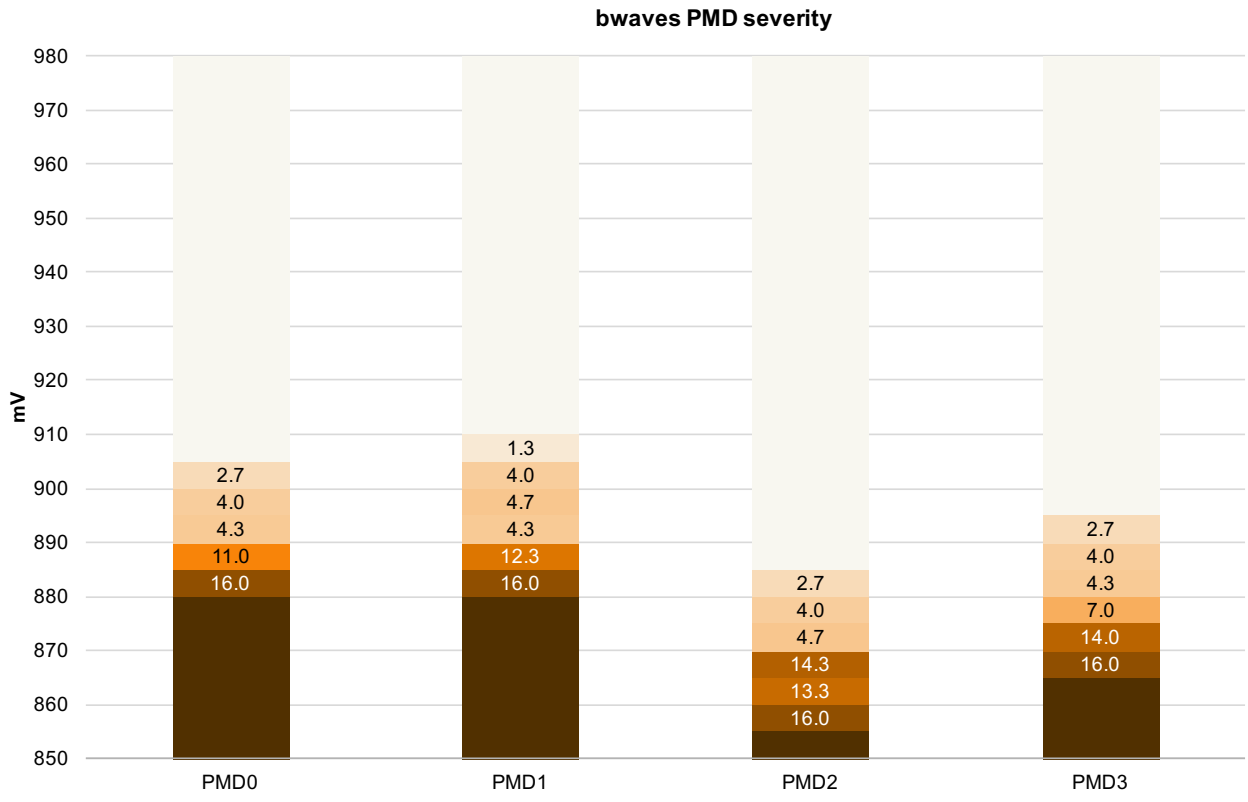

Figure 24: bwaves benchmark – PMD severity scaling.

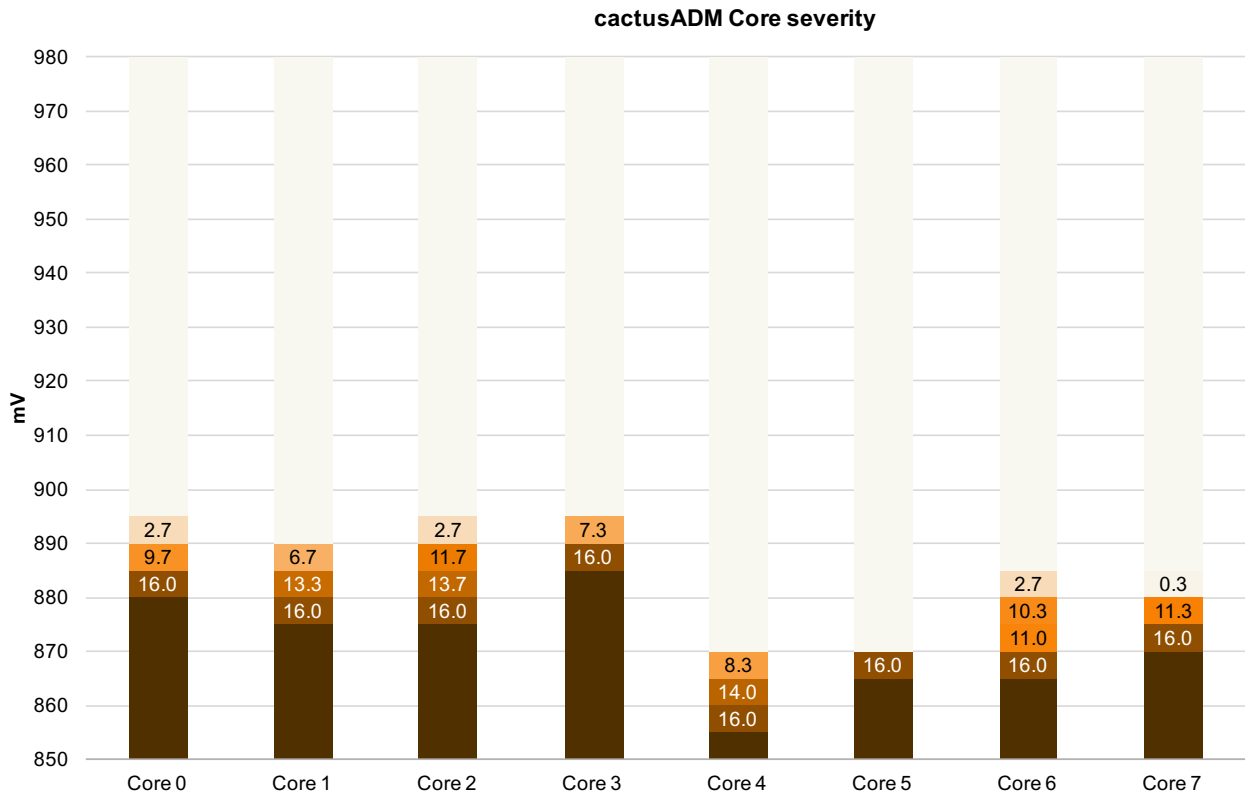

Figure 25: cactusADM benchmark – Core severity scaling.

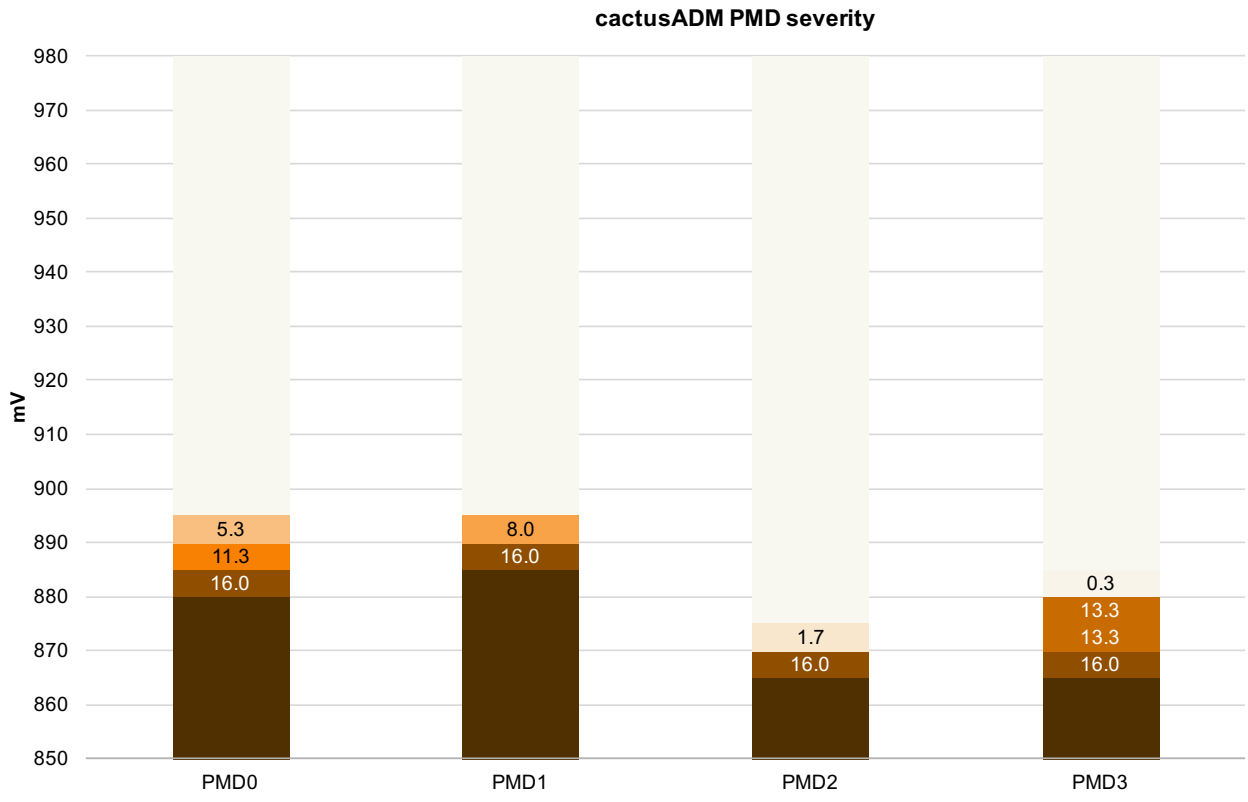

Figure 26: cactusADM benchmark – PMD severity scaling.

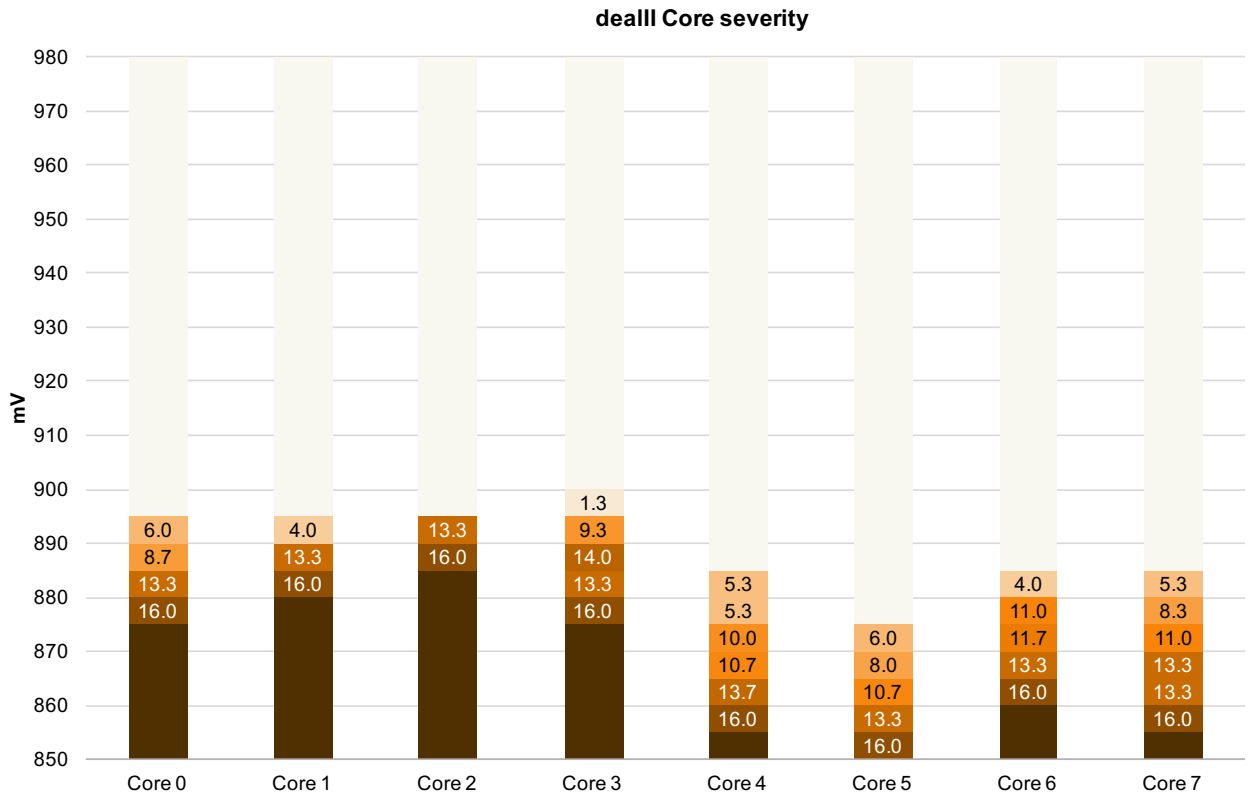

Figure 27: dealII benchmark – Core severity scaling.

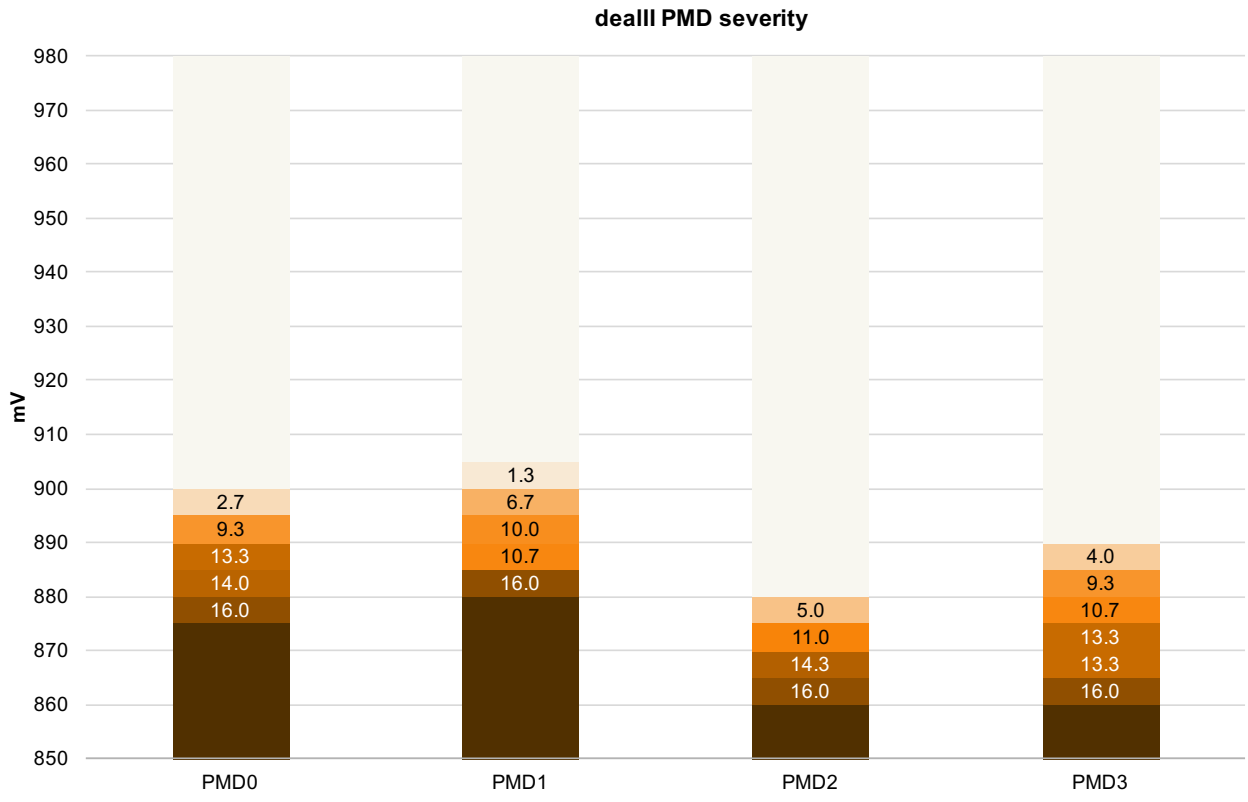

Figure 28: dealII benchmark – PMD severity scaling.

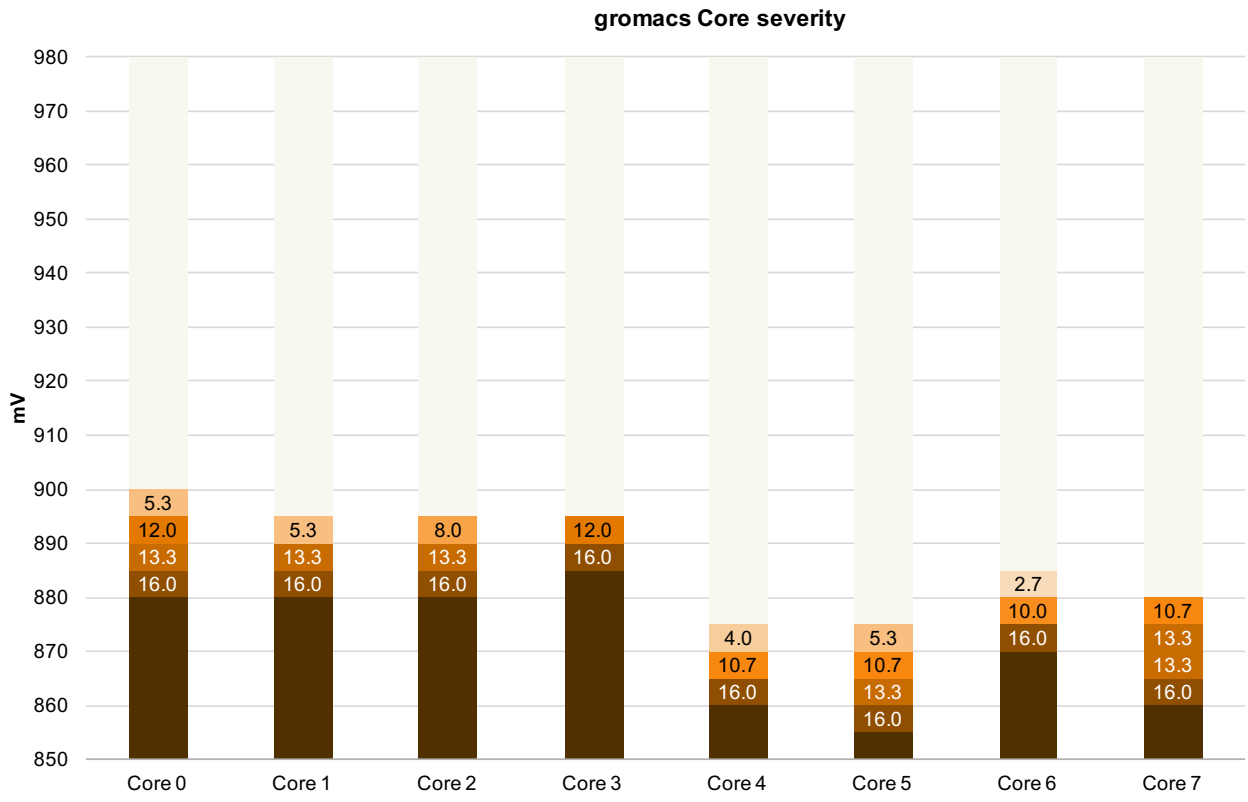

Figure 29: gromacs benchmark – Core severity scaling.

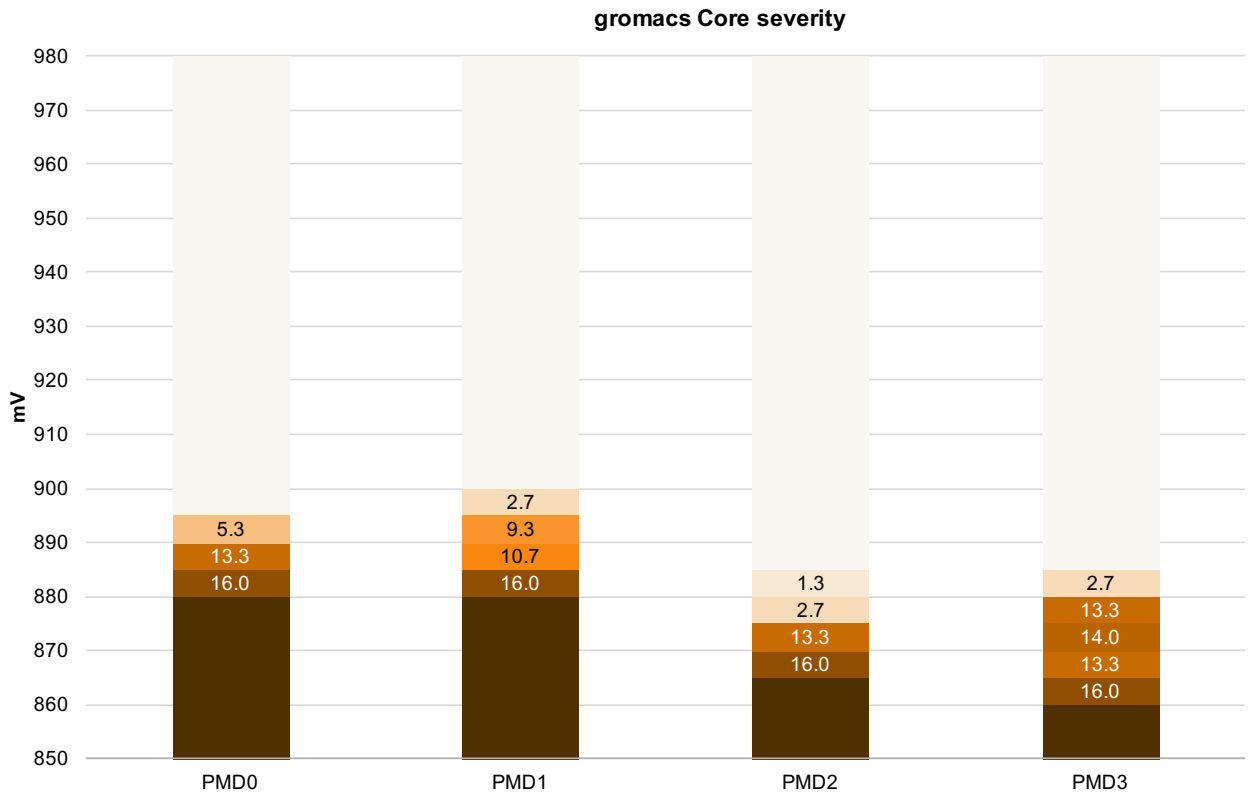

Figure 30: gromacs benchmark – PMD severity scaling.

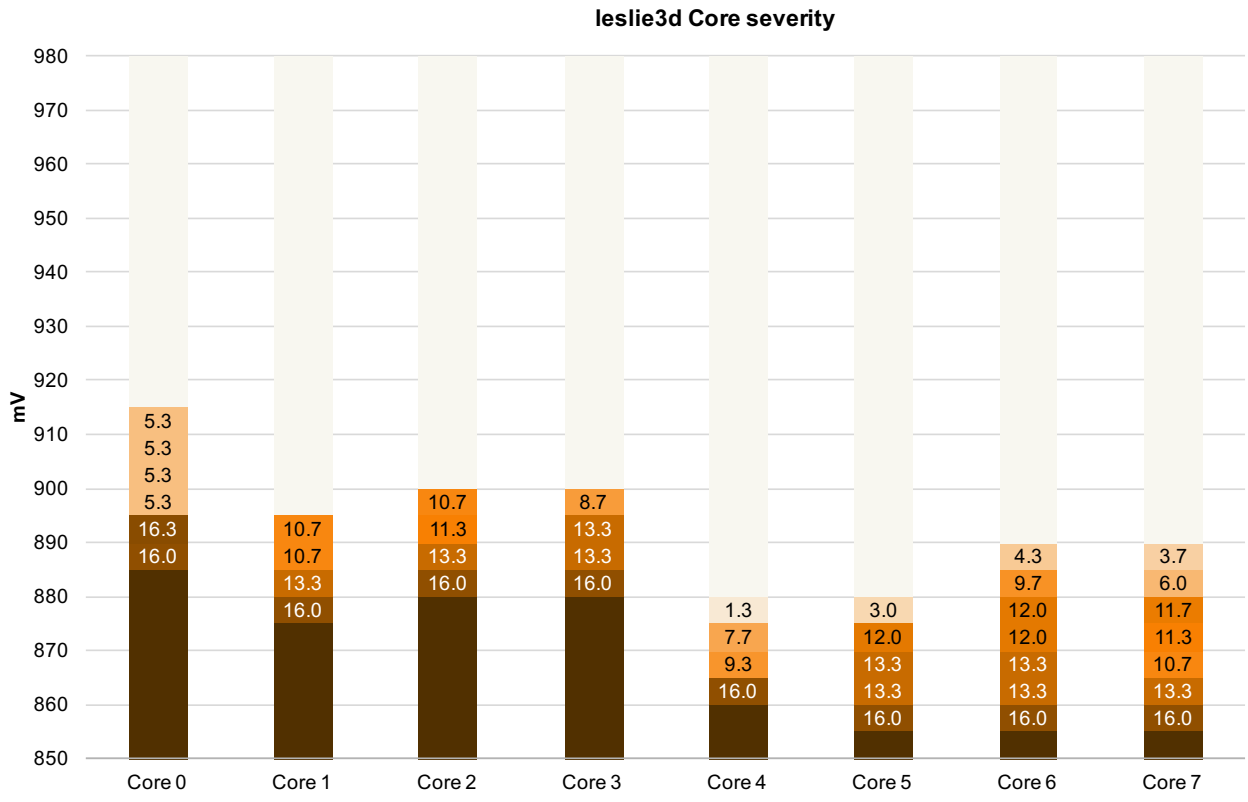

Figure 31: leslie3d benchmark – Core severity scaling.

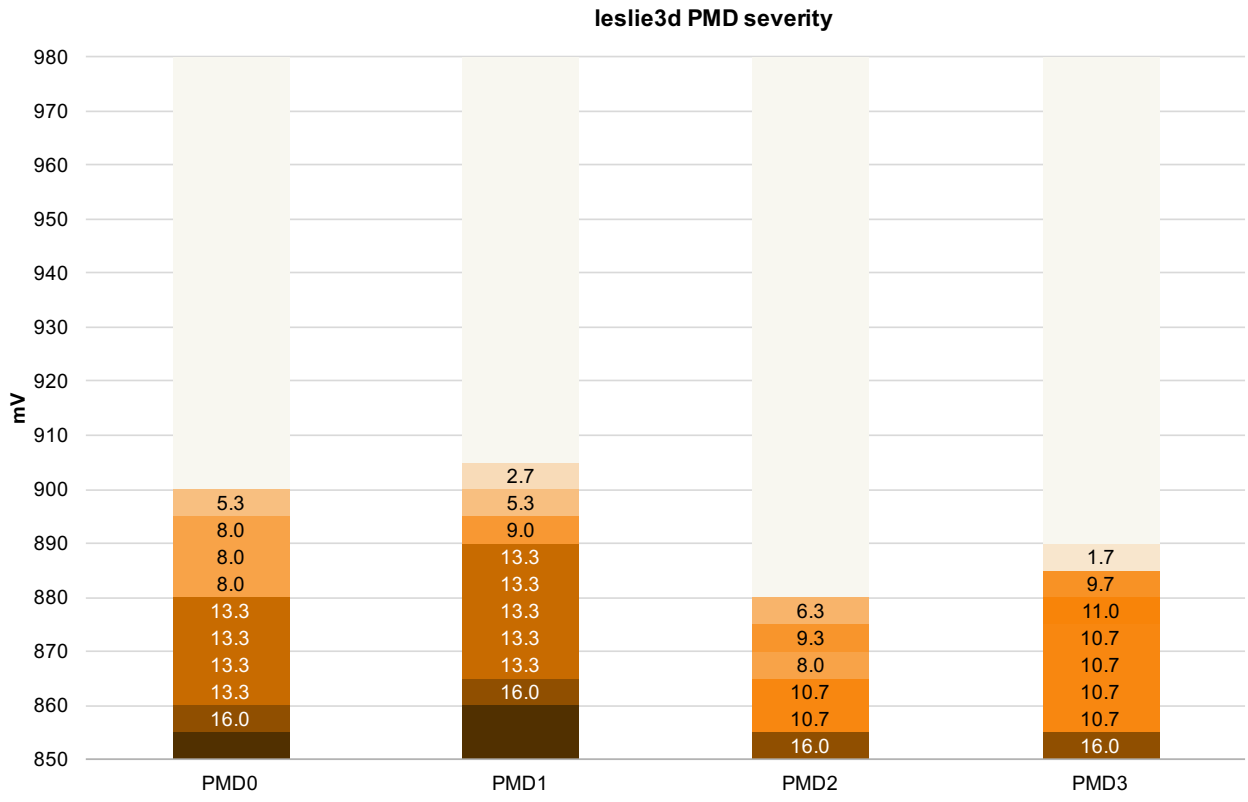

Figure 32: leslie3d benchmark – PMD severity scaling.

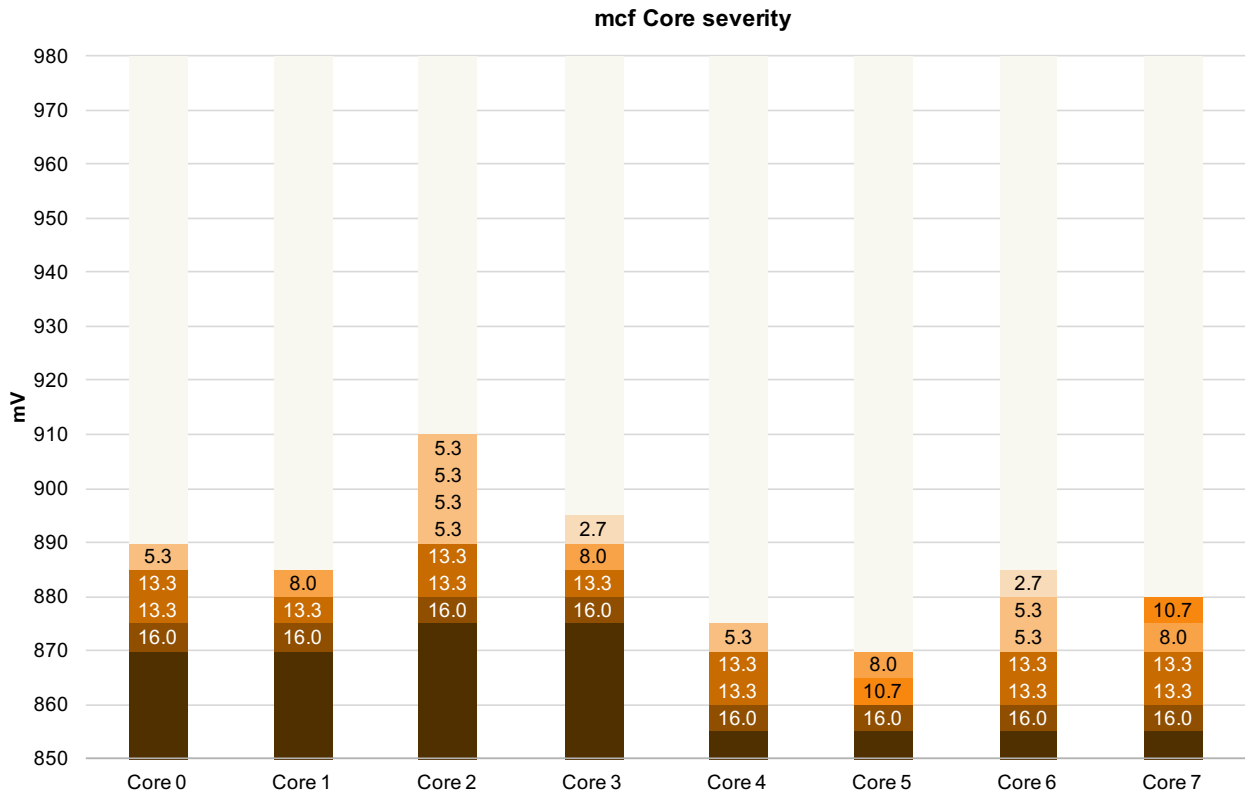

Figure 33: mcf benchmark – Core severity scaling.

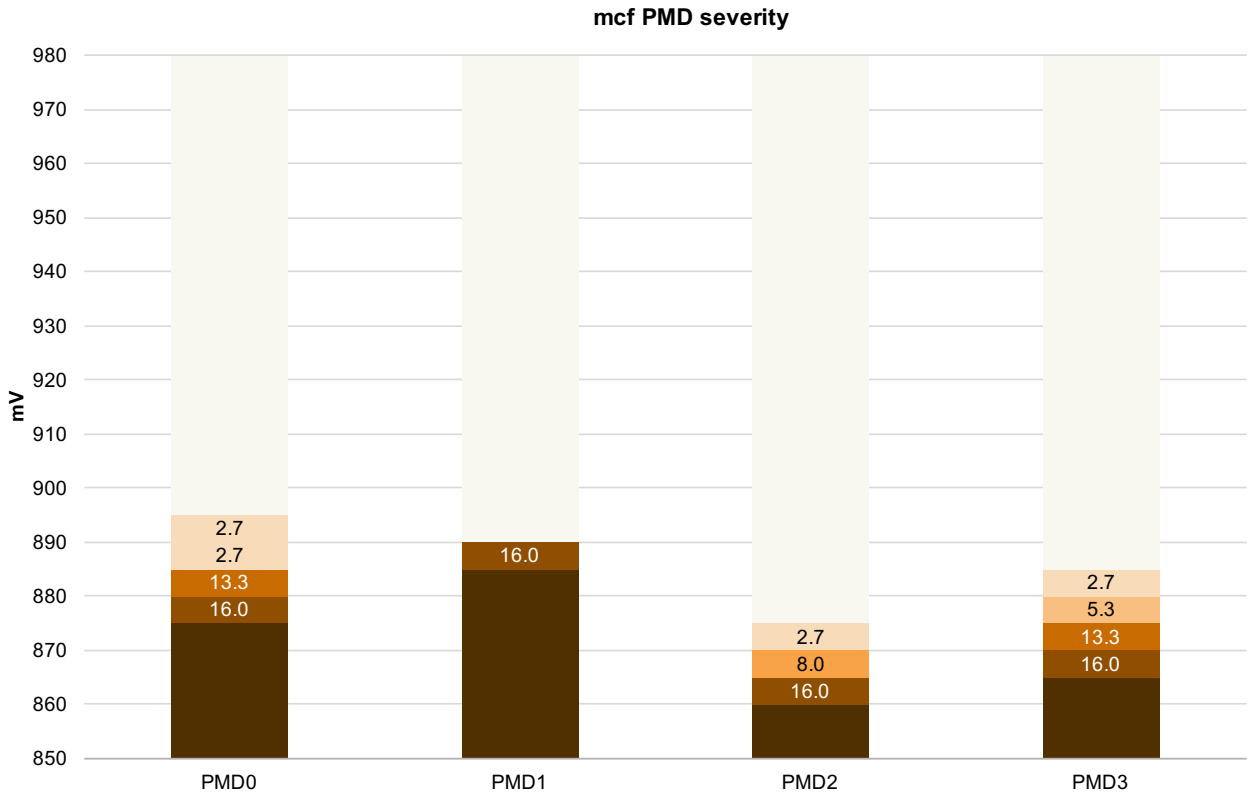

Figure 34: mcf benchmark – PMD severity scaling.

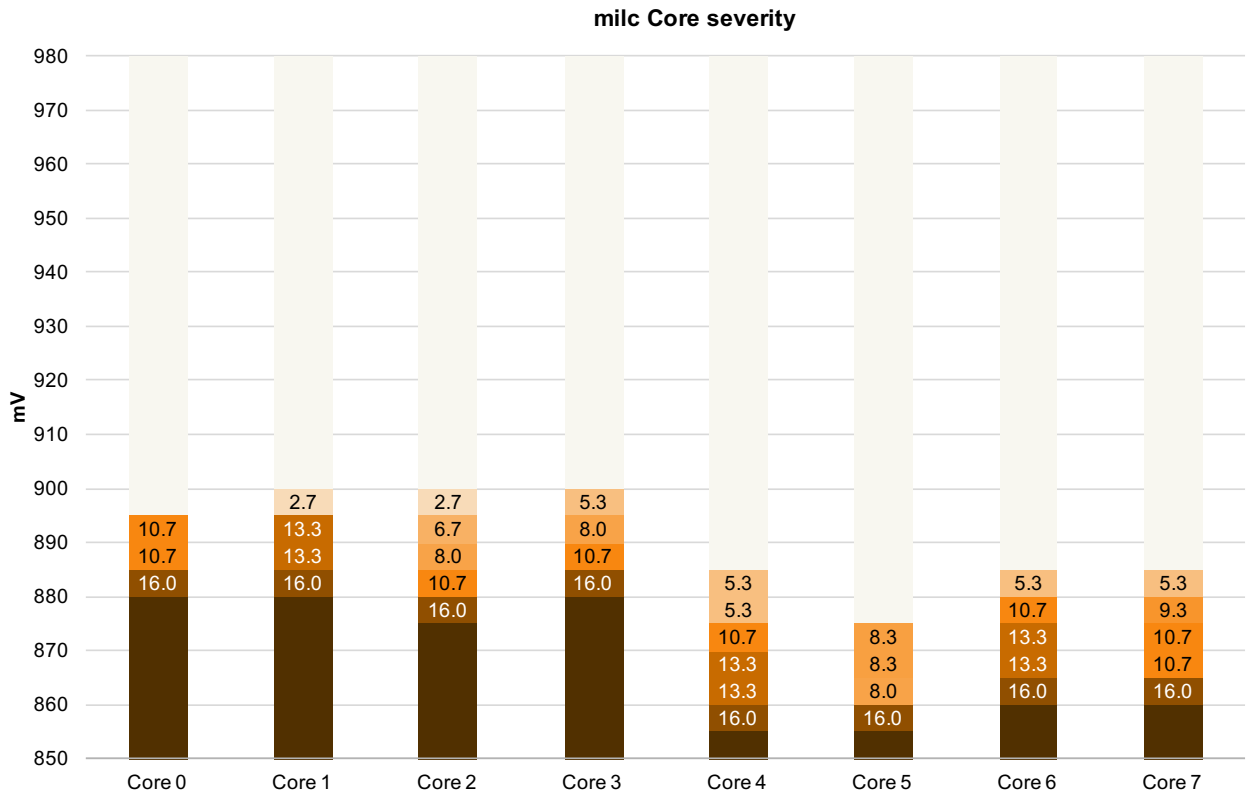

Figure 35: milc benchmark – Core severity scaling.

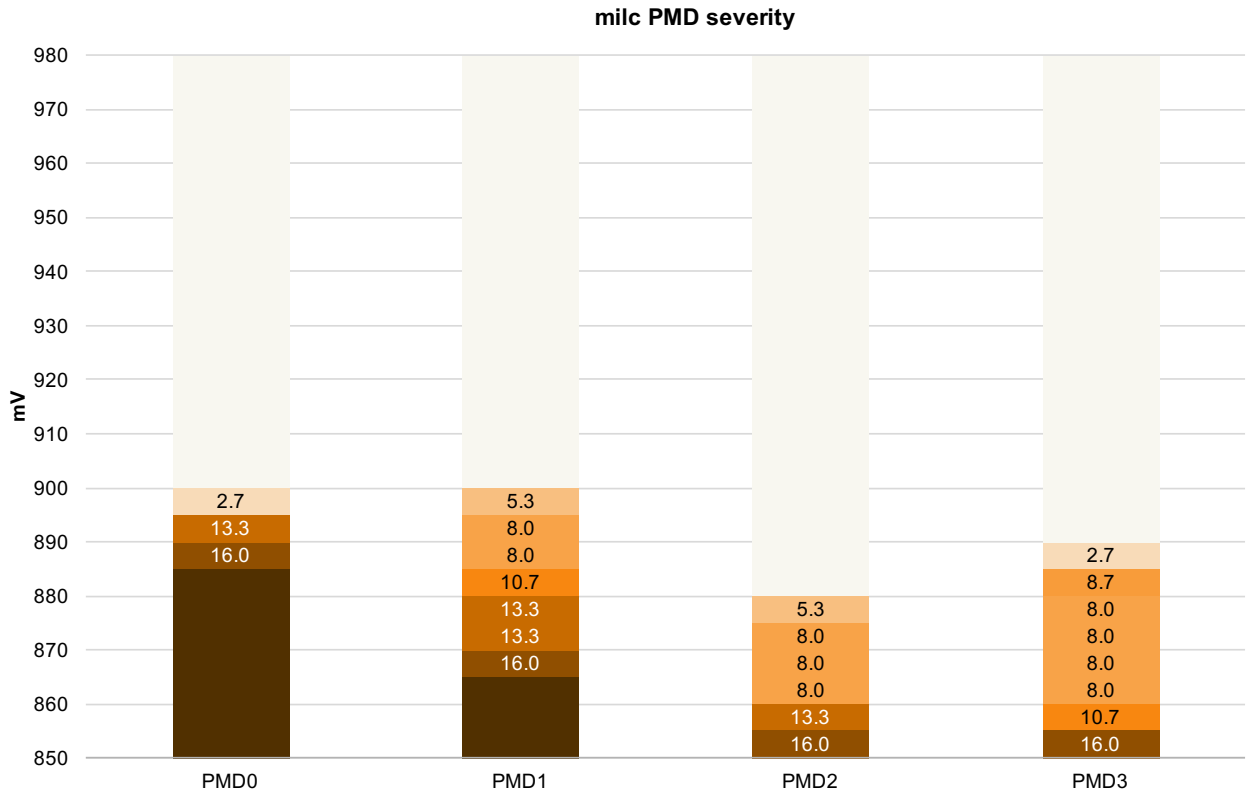

Figure 36: milc benchmark – PMD severity scaling.

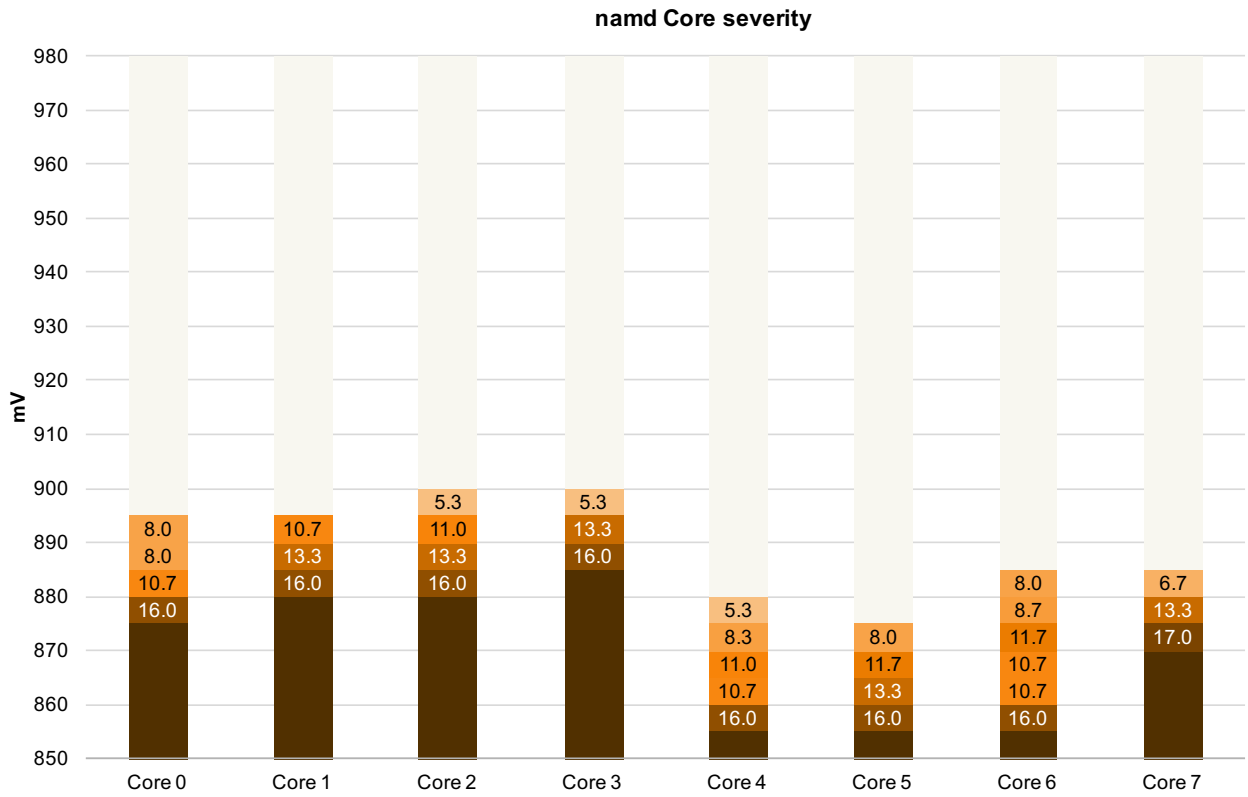

Figure 37: namd benchmark – Core severity scaling.

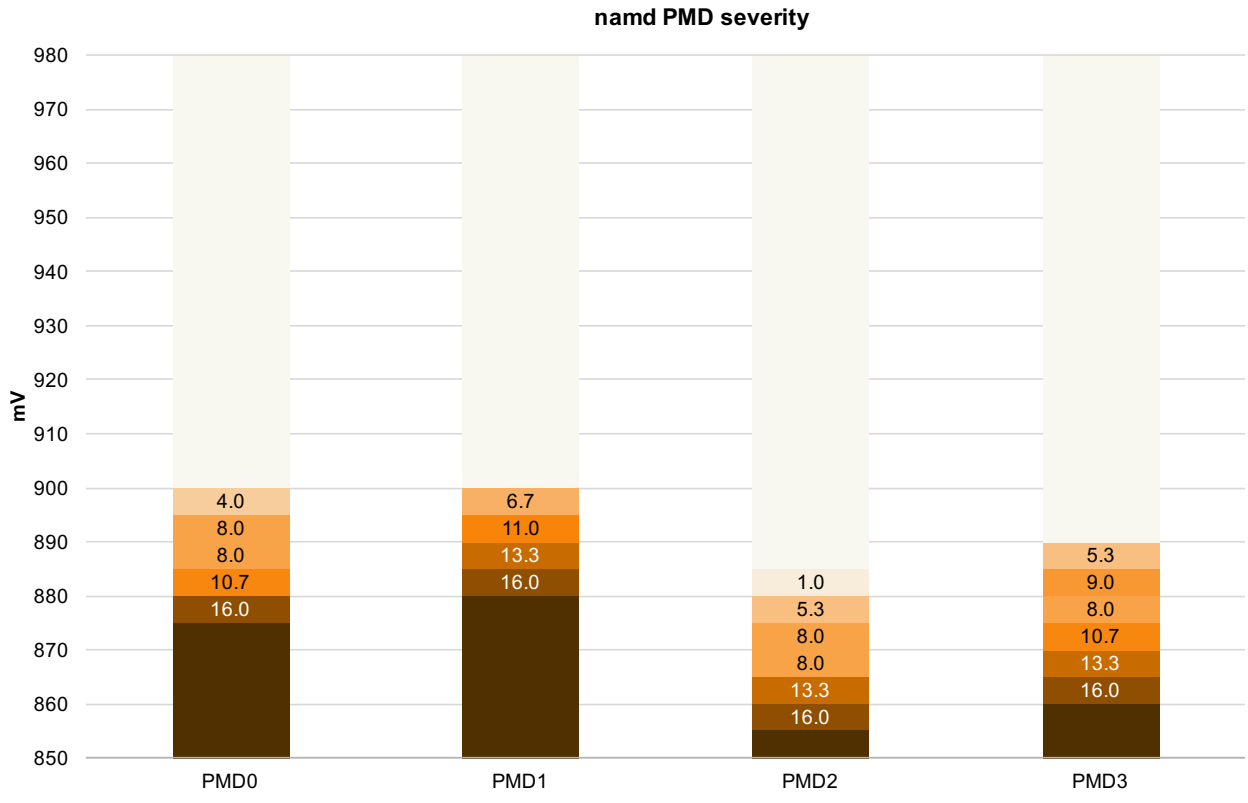

Figure 38: namd benchmark – PMD severity scaling.

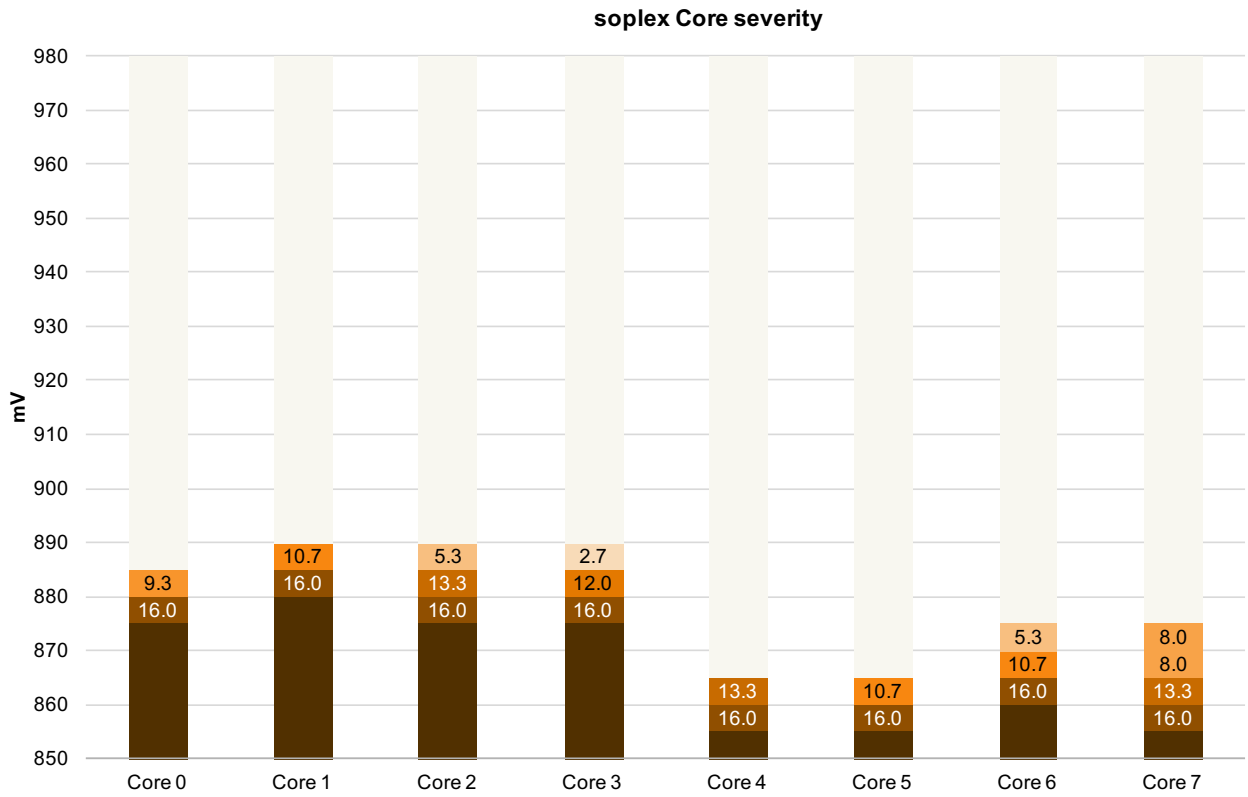

Figure 39: soplex benchmark – Core severity scaling.

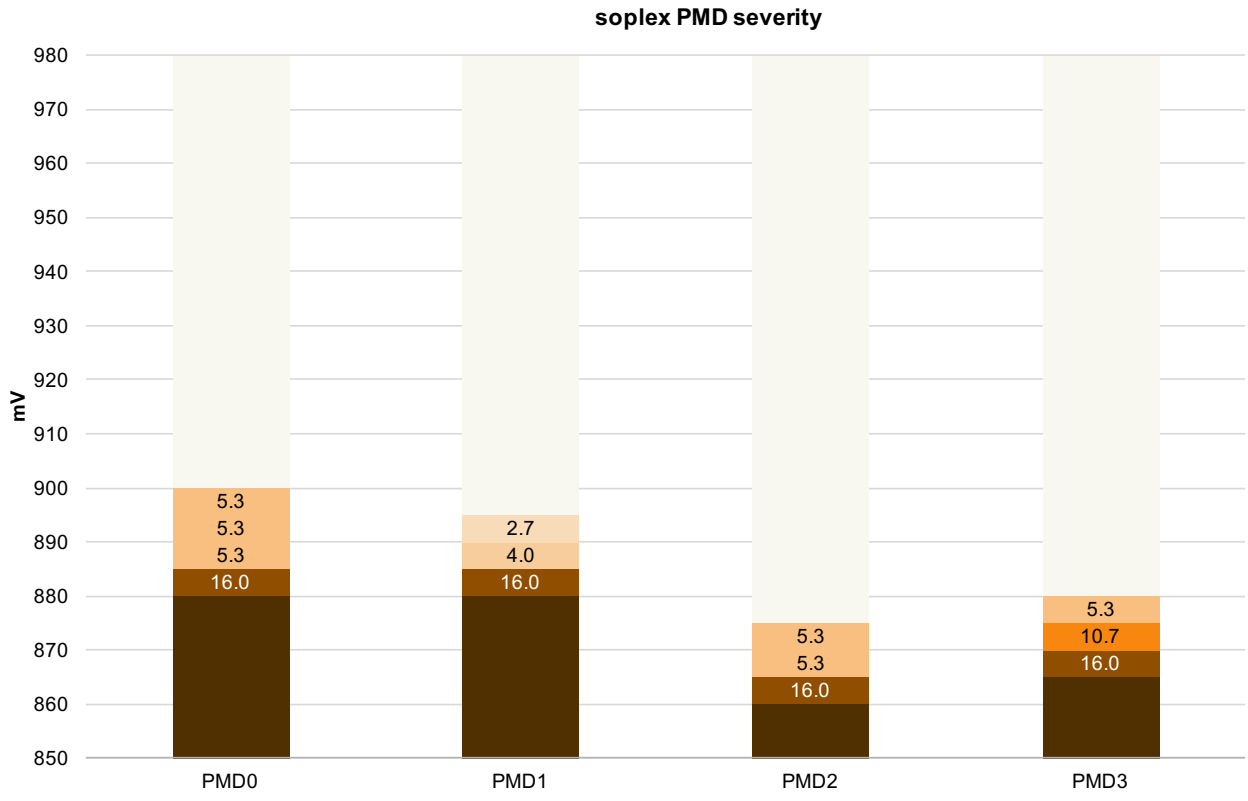

Figure 40: soplex benchmark – PMD severity scaling.

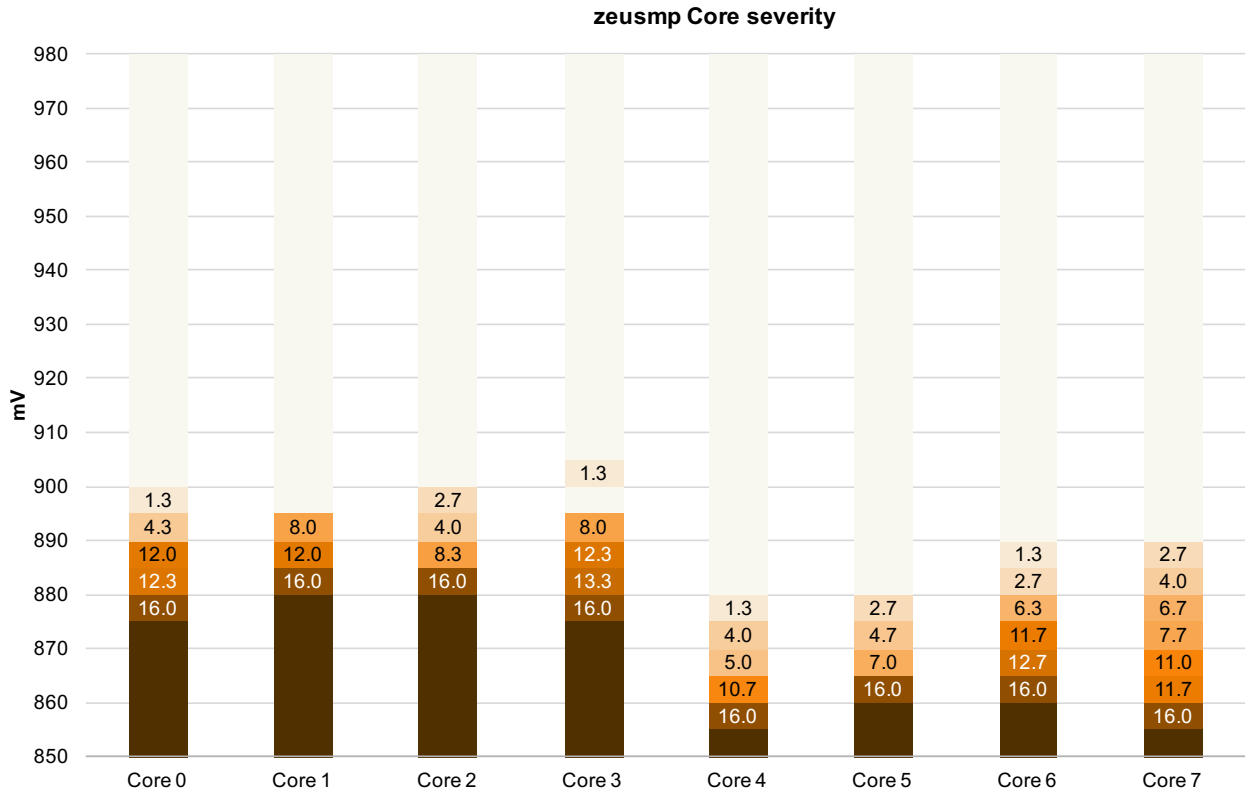

Figure 41: zeusmp benchmark – Core severity scaling.

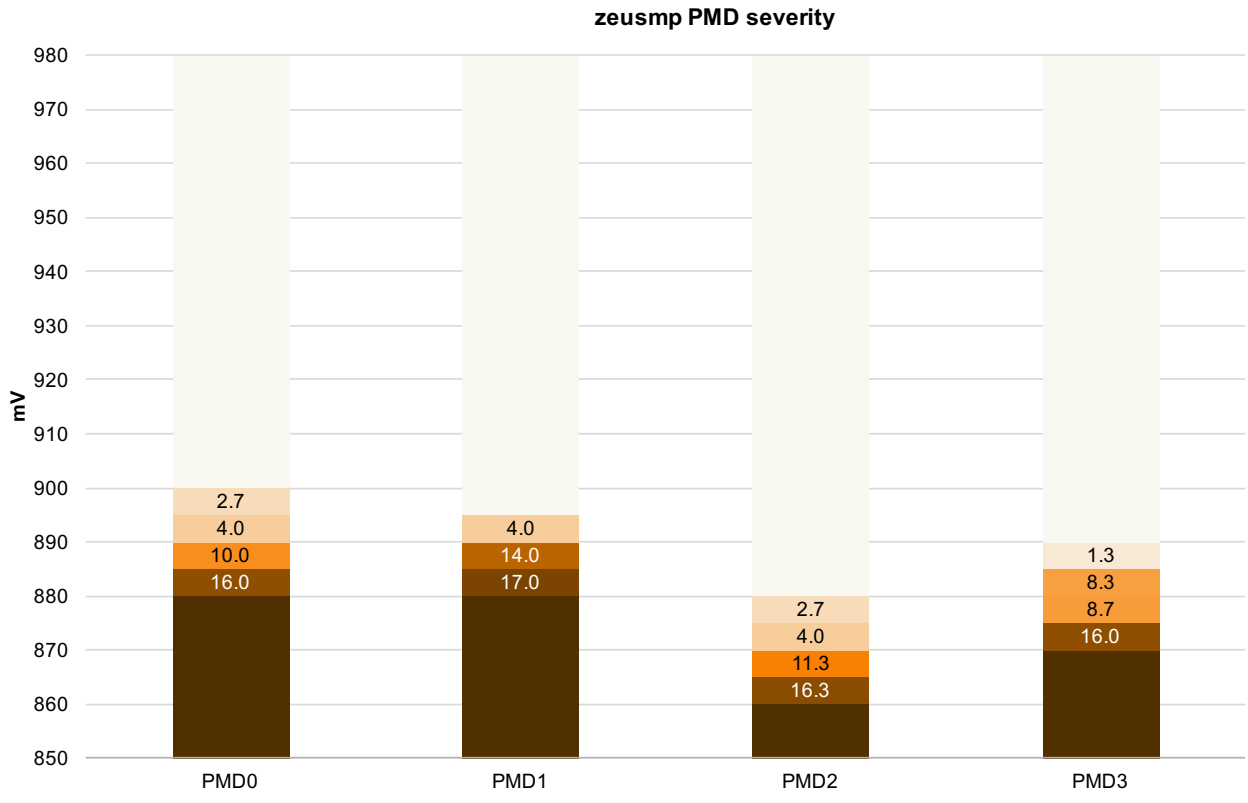

Figure 42: zeusmp benchmark – PMD severity scaling.

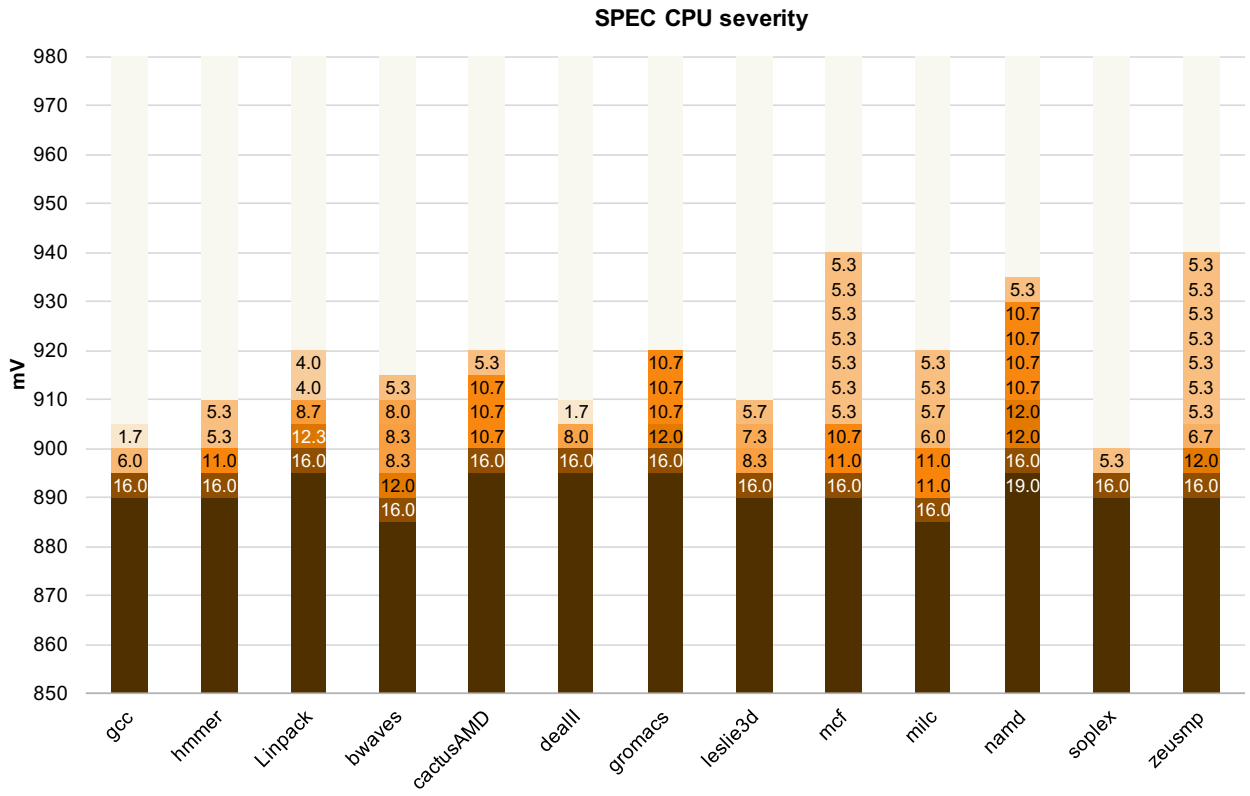

Figure 43: CPU severity scaling.

The unsafe region can consist of a combination of SDCs, CEs, UEs, PCs and/or SCs. However, concerning the caches we also collect all the exposed corrected and uncorrected errors, provided by the syslog of Linux kernel. Figure 44 presents the distribution of errors concerning L1 and L2 caches for 2 benchmarks: the linpack and zeusmp. As shown in this figure, the majority of errors is in L1 cache for both of the benchmarks.

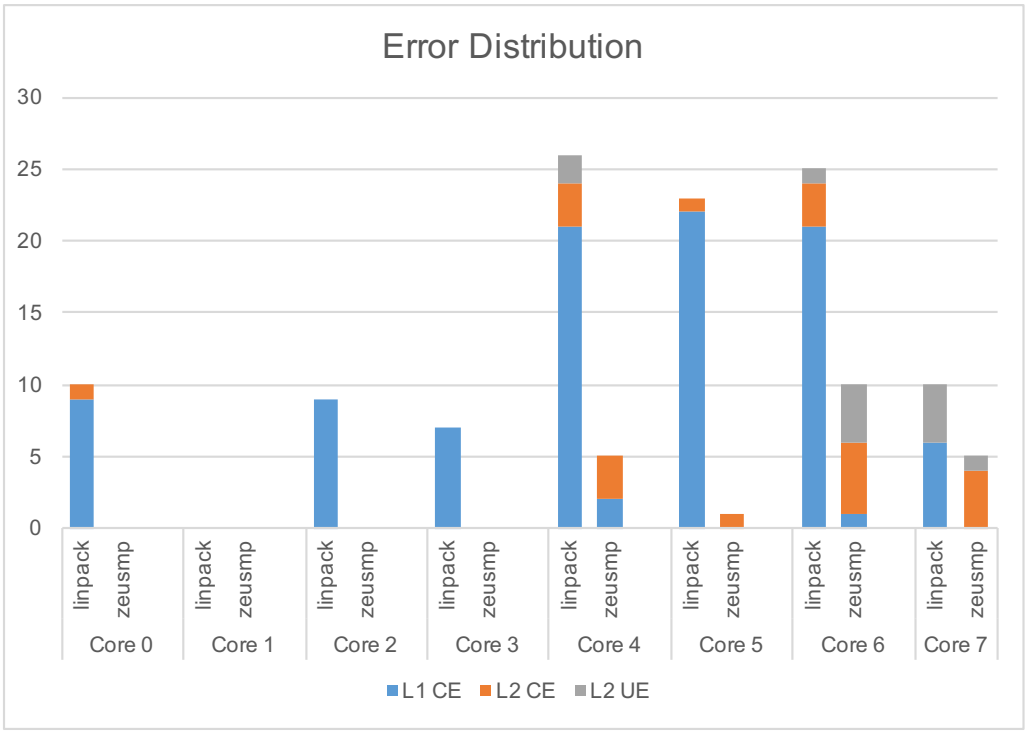

Figure 44: Distribution of Errors for Linpack and zeusmp benchmarks. This figure presents the population of errors of each benchmark. The benchmarks have different execution time.

Figure 45 summarizes the entire image concerning the minimum and maximum observed Vmin for all the benchmarks that were used in this study for different configurations (when the benchmarks run individually in different cores, different PMDs or simultaneously in all cores). From this figure, we can observe the variation of the minimum and maximum Vmin among different configurations. Again, it is remarkable that Core 4 and Core 5 are the most robust among the cores (865mV-900mV), while the configuration of running the benchmarks simultaneously in all the cores of the CPU seems to be the most sensitive (900mV-940mV). In Figure 46, we translate the results of the minimum and maximum Vmin of Figure 45 into percentage of power savings compared to the operation in nominal voltage. The gain in power savings can range from 8% (CPU) to 22.1% (Core 4 and Core 5).

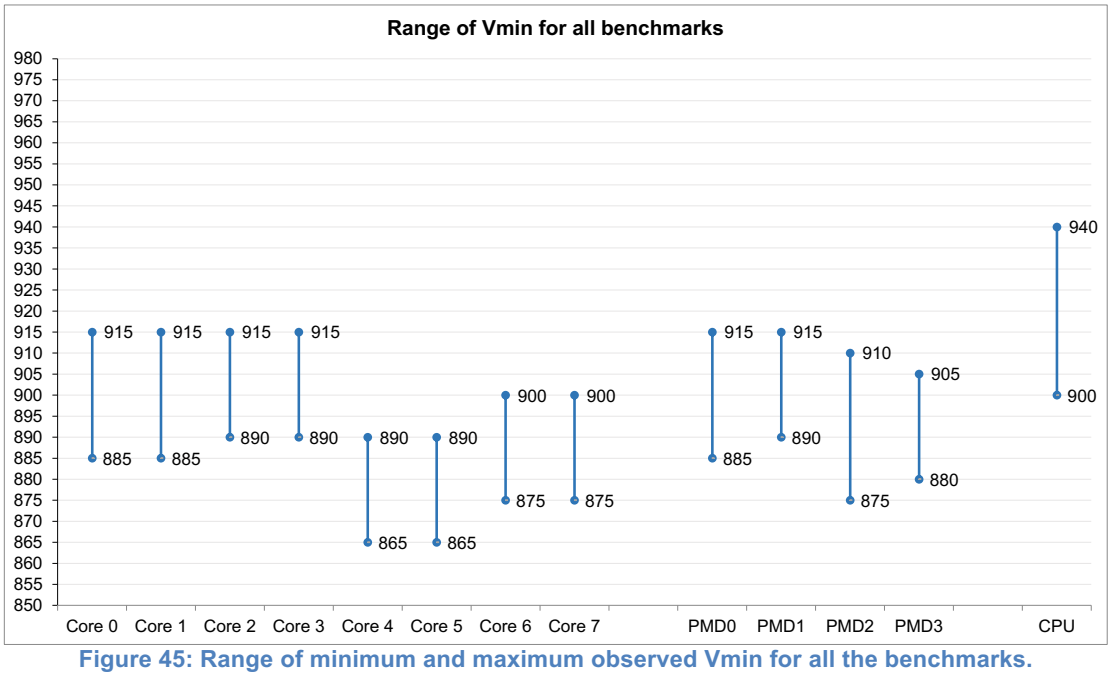

Figure 45: Range of minimum and maximum observed Vmin for all the benchmarks.

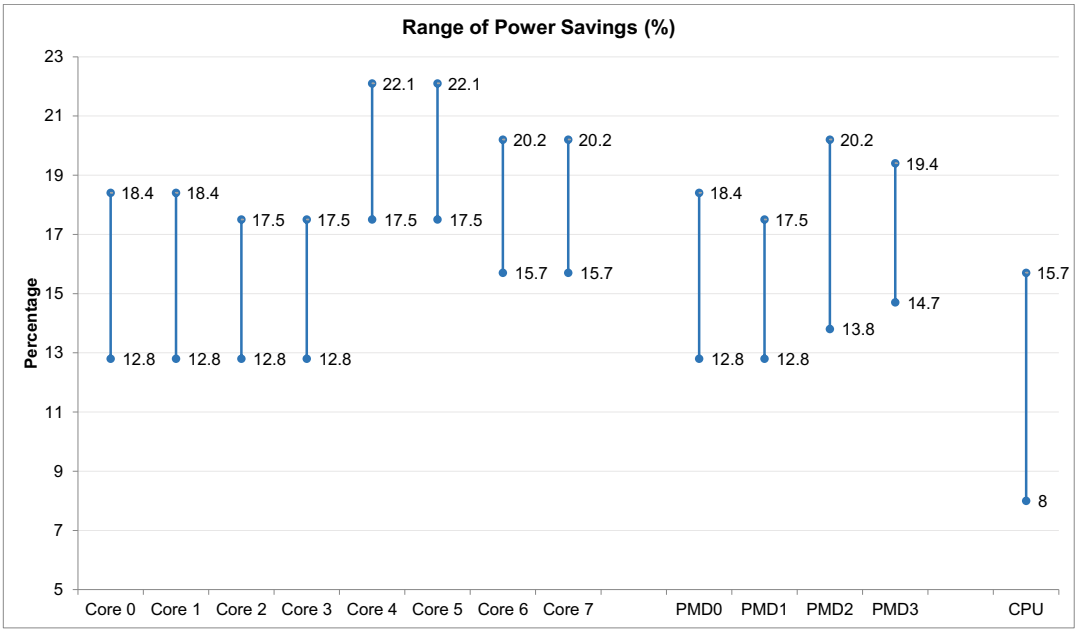

Figure 46: Range of minimum and maximum power savings (%) for all the benchmarks

2.3.2. Results Provided by TSS Chip at 2.4GHz

This section provides information about the 2 benchmarks run on TSS chip. In the figures below (Figure 47, Figure 48) we can observe that although the behaviour of these benchmarks is the same, the unsafe region has shifted by approximately 15 mV higher than the corresponding results of TTT chip.

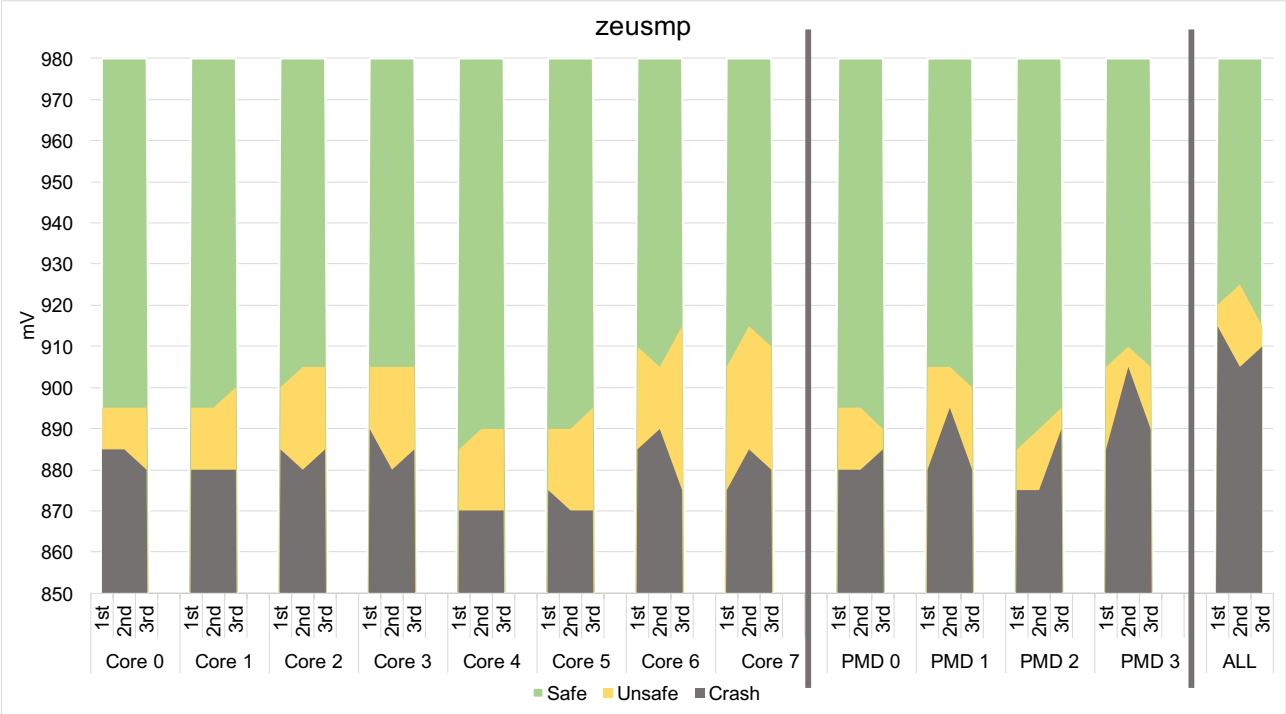

Figure 47: zeusmp benchmark run in TSS chip.

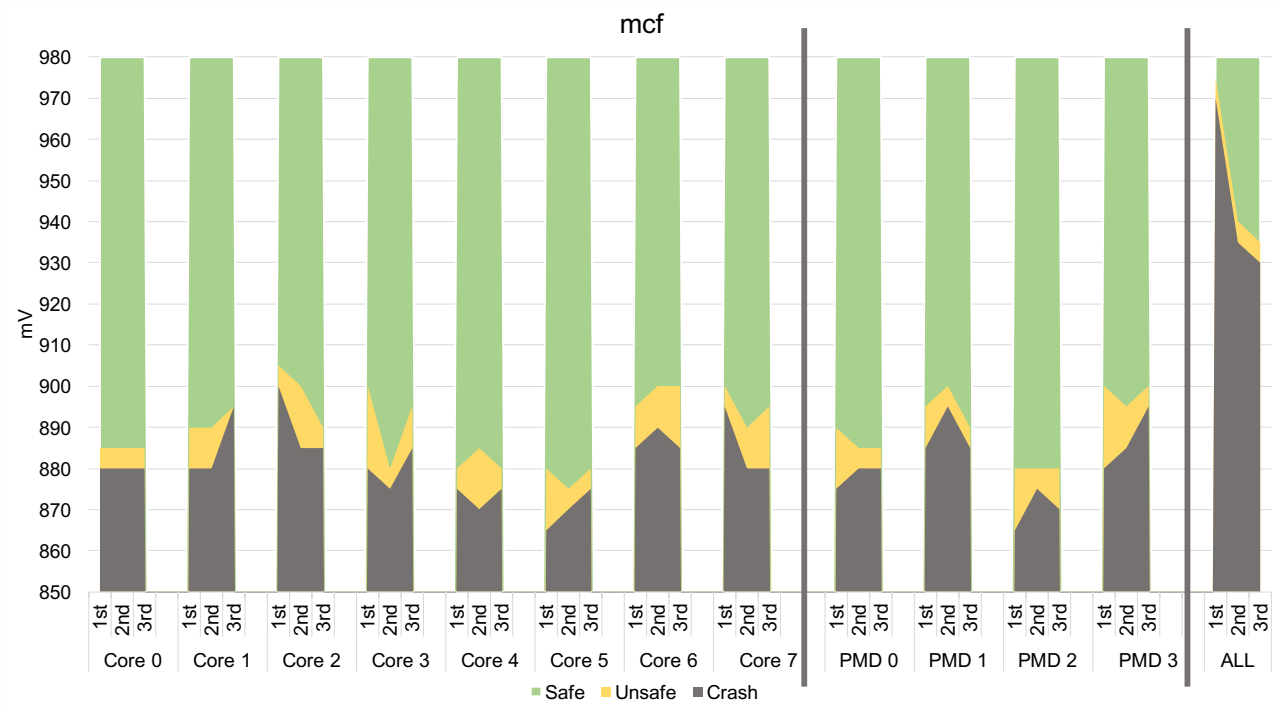

Figure 48: mcf benchmark run in TSS chip.

Below the severity scaling graphs are shown for the same benchmarks.

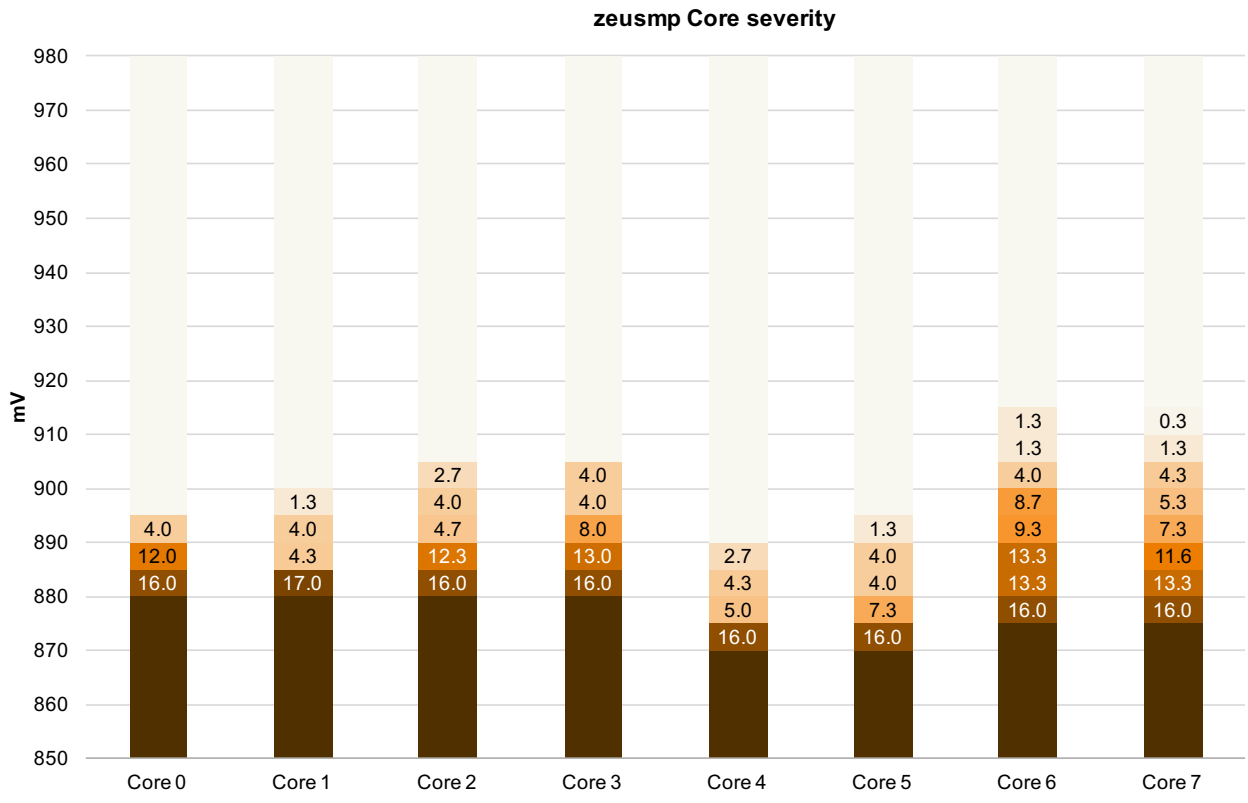

Figure 49: zeusmp benchmark – Core severity scaling.

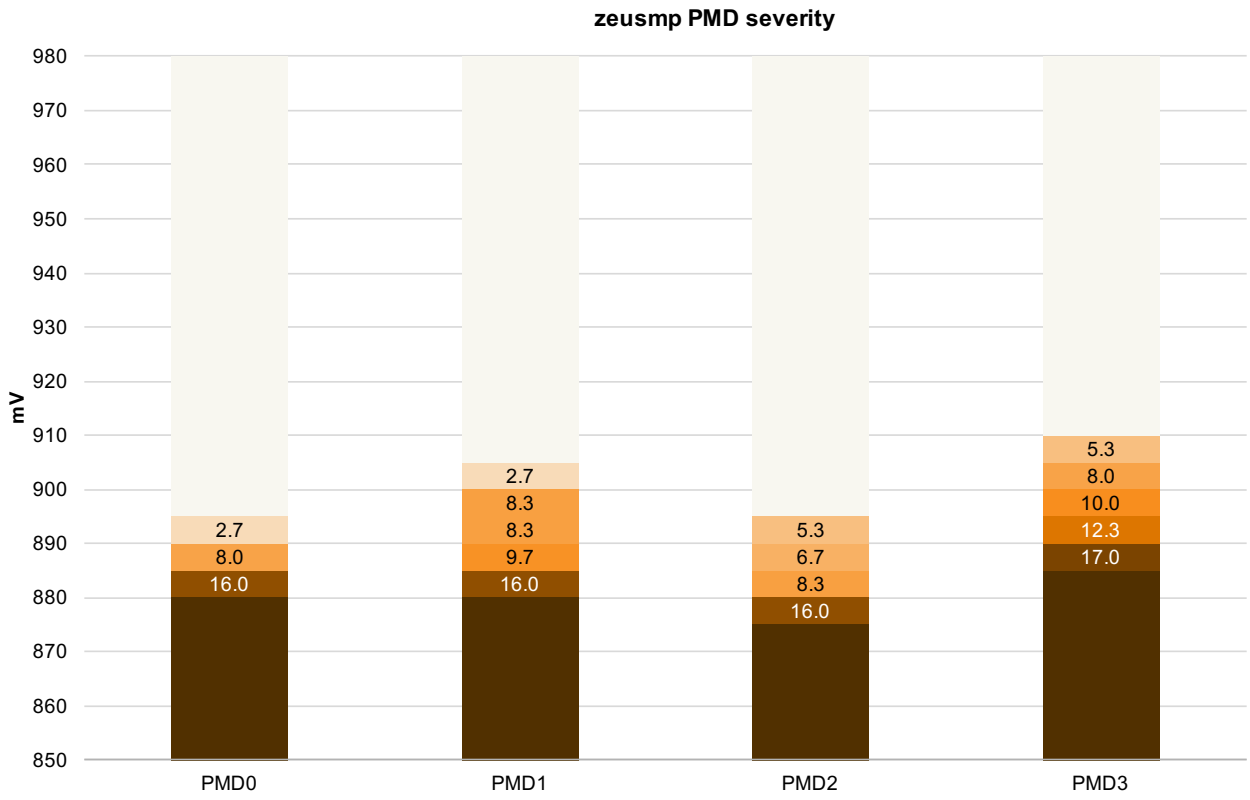

Figure 50: zeusmp benchmark – PMD severity scaling.

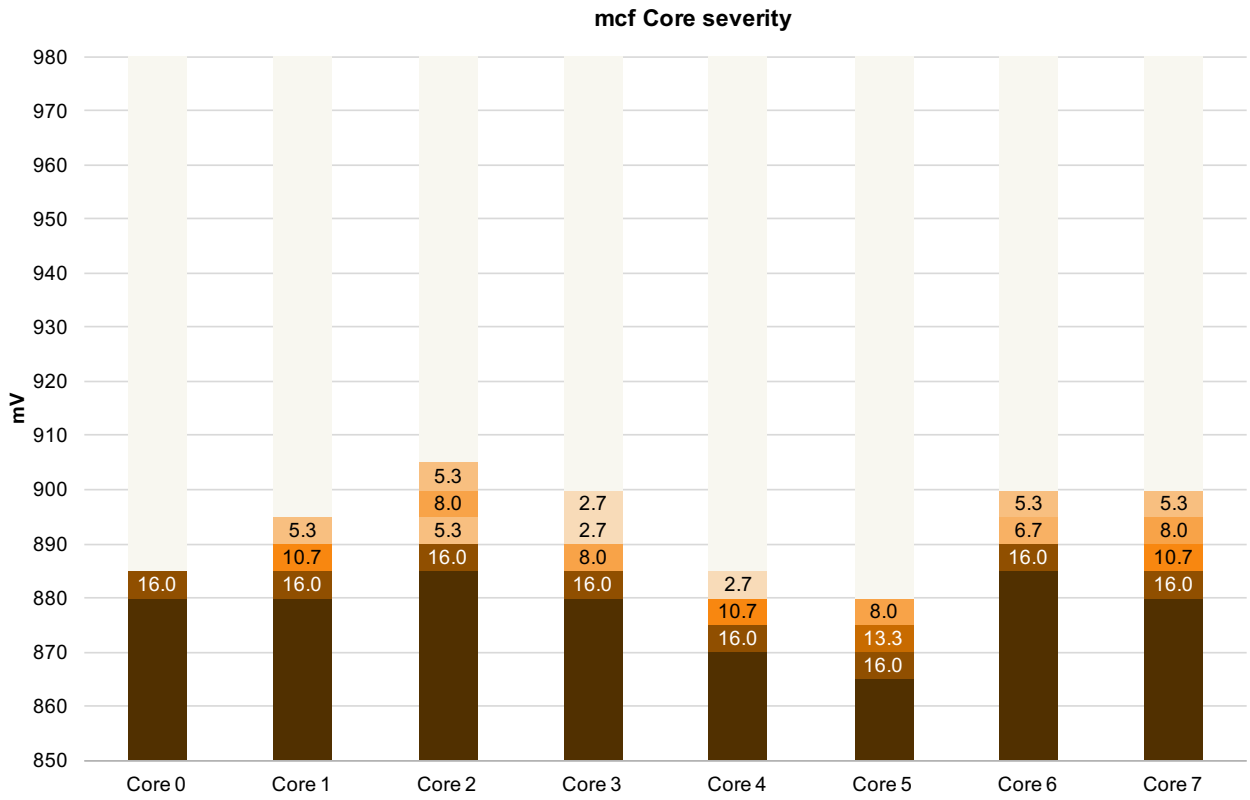

Figure 51: mcf benchmark – Core severity scaling

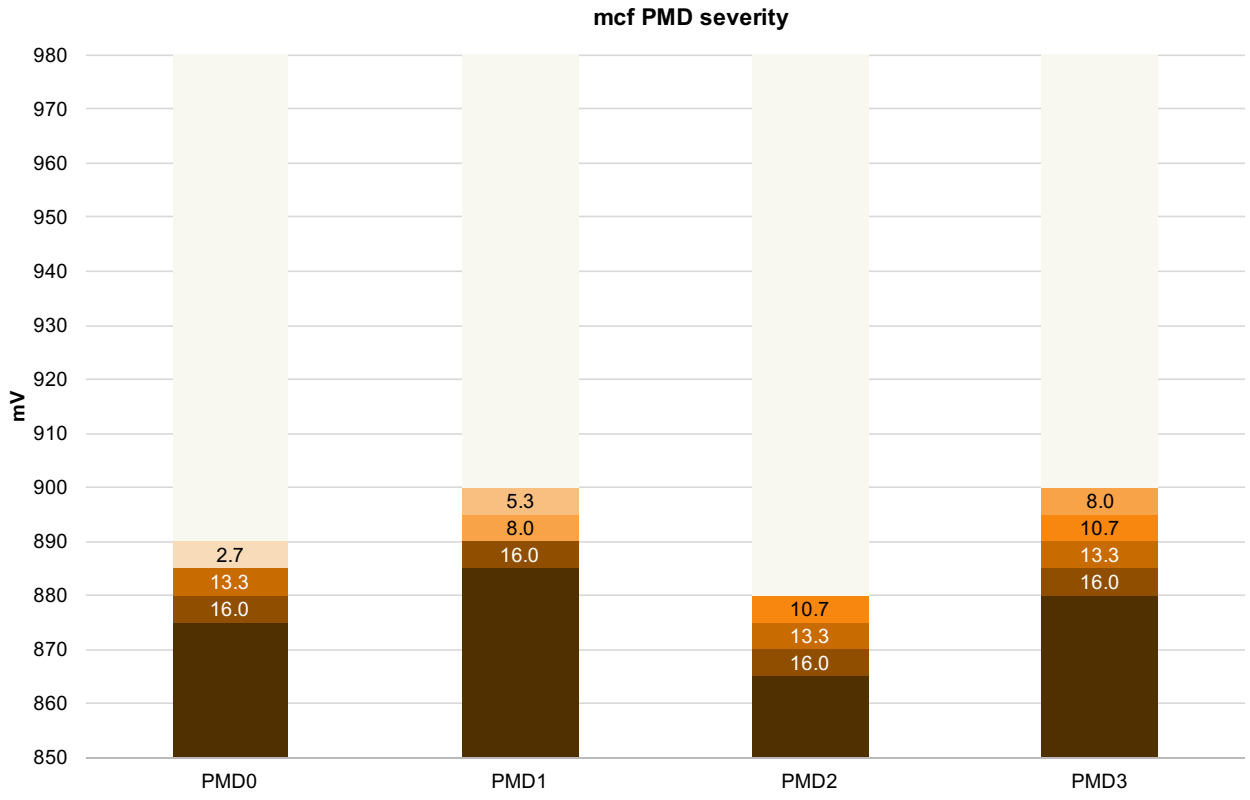

Figure 52: mcf benchmark – PMD severity scaling.

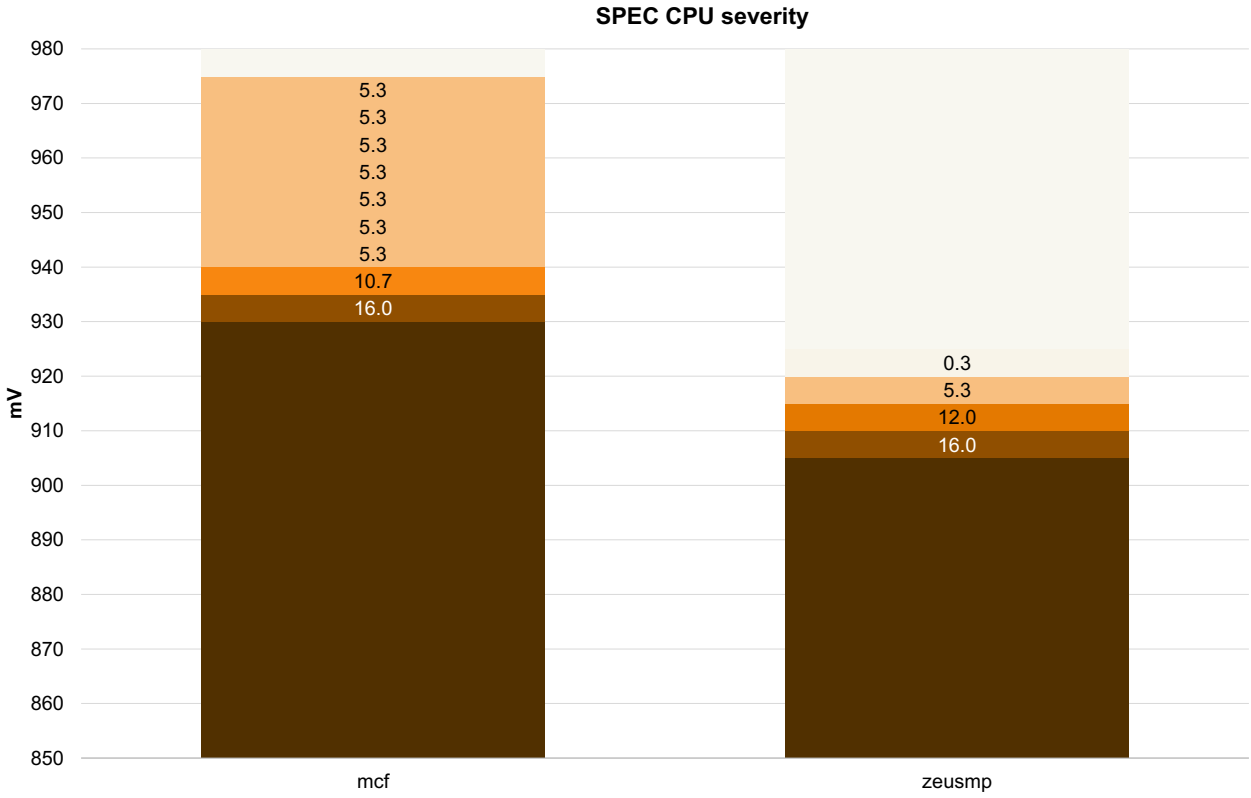

Figure 53: CPU severity scaling.

2.4. L1 Data Cache Self-Test

We developed a memory self-test for L1 Data cache for X-Gene 2. The self-test has the property of creating instruction and data traces that can completely fill the L1 Data cache and iteratively stress it. Then, the test itself can recognize if there is a faulty bit-cell in L1 Data cache due to undervolting conditions.

To ensure that our program fills the whole cache array, we leveraged the native performance counters provided by X-Gene 2. We instrumented our self-test to measure the L1 Data cache hits and misses to understand if the test performs the appropriate reads and writes according to the cache configuration. This can be done only by using the Performance Monitoring Unit's (PMU) registers, and not the perf tool. The reason is that the perf tool adds some extra operations to monitor the behavior of the program, and thus, given that we need as much targeted information as possible, we read directly the PMU registers. Also, to avoid any unwanted accesses to the memory stack (because the C compiler adds this overhead), our self-test is a mix of C and assembly instructions, in order to use general purpose registers instead of variables, and thus, undesirable memory accesses that could harness the data traces of our test.

Although it is very difficult to develop a self-test in a real hardware that ensures 100% bit-cell coverage of L1 Data Cache, due to the limited observability, the complex design and several microarchitectural events that can occur during the microprocessor's and operating system's operation, the information provided by the performance counters indicates that our self-test has almost 100% coverage. More specifically, if we take into account that the X-Gene's 2 L1 Data cache is 32KB, with 512 blocks of 64 bytes each, our self-test performs 513 misses (instead of 512, which would be the ideal) and approximately 4100 cache hits (the ideal would be 4096 total hits). These minor differences are caused by noise from the full system and performance microarchitectural components (e.g. branch predictor, coherence protocols, etc.). After exhaustive testing, we have concluded that these differences do not affect the coverage of our self-test, especially when referring to multiple iterations.

The self-test was developed accordingly to cover the write-through/no write-allocate policy of L1 Data cache of the X-Gene 2. This practically means that when we initially perform single store operations (store misses), the blocks are allocated in L2 cache instead of L1. To ensure that our data blocks are valid in L1 Data cache, we perform a series of load operations to the same data values in order to allocate L1 blocks. A second pass is also required to ensure that the fetched blocks will not replace our data, due to pseudo-LRU replacement scheme. With these 2 initial passes, we guarantee that our data reside in L1 Data cache and we can start performing iteratively reads to the contents of L1 Data cache, while reducing the voltage gradually by 5mV per 30 seconds.

After ensuring the correctness and the coverage of the self-test, we perform an extensive undervolting campaign by running this test. Our first observations are that we neither lead to any cache error (note that the L1 Data has parity protection), nor any data corruption.

Self-test programs for the other cache blocks will be developed and applied to study the behavior of L1I, L2 and L3 caches when the corresponding voltage domain is undervolted.

3. Characterization of Dynamic Memories

Apart from the cores and the on-chip caches, Dynamic Random-Access Memory (DRAM) is an important subsystem of computing systems that has recently attracted a great amount of interest as a target for power and performance optimizations. The density of DRAM devices which has been increasing steadily the past few decades, coupled with our increased needs for main memory capacity makes DRAM a significant contributor in the total power budget of contemporary computing systems [25]. Moreover, further scaling of DRAM cells is accompanied by reduction of cell reliability and increased cell leakage leading to high refresh rates, which in turn inflates power consumption [26] [27] [28] [29] [30] [31]. As a result, any power optimizations regarding DRAM devices has a significant impact [32].

DRAM power could be reduced through lowering the supplied voltage, however this could compromise consistency of data stored in DRAM. Alternatively, DRAM refresh rate could be relaxed to save power. This rate determines how often each cell is refreshed, which is required to sustain data stored in DRAM due to its dynamic nature. Refresh operation consumes power that is expected to be up to 50% of the total power consumption of the whole DRAM power [33].

The bottom-line is that both reduction of the input voltage and relaxing of the refresh rate may help reduce power but it could also negatively affect the DRAM reliability and compromise the consistency of data stored in DRAM.

In the first year of the project we have started investigated how applications with different data and access patterns affect the number of errors for DRAM operating at the lowered input voltage and relaxed refresh rates. We discovered that the number of DRAM faults vary significantly and it is highly depended on the distribution of the stored data as well as on the memory access patterns of the applications. Our initial characterization revealed that the real benchmarks produce more errors than the data pattern micro-benchmarks considered in the previous research studies to characterize DRAM errors.

One major finding of our research is that memory access patterns have a dominating impact, especially for parallel applications, on the manifested errors in a DRAM operated at a scaled-down supply voltage or at a relaxed refresh rate. To investigate in more depth, the DRAM properties we have setup an experimentation setup by considering classical micro-benchmarks and test patterns used in DRAM testing but also benchmarks with different access patterns.

3.1. Main Memory – Dynamic Random Access Memory (DRAM)

Before moving into the description of our setup and results, we briefly provide some basic information about the DRAM organization and functionality. A modern main memory DRAM system is organized hierarchically into channels, ranks, banks, rows and columns, as shown in Figure 54. Each channel has each own independent lines for sending commands and address, and for transceiving data. It consists of one or more Dual In-line Memory Modules (DIMMs). Each DIMM usually has two ranks that each one consists of DRAM devices that are grouped together and all the devices of a rank are working in parallel to serve the request.

Within each chip, there is logic that controls the operation of the device and a number of banks, in typical configuration there are 8 banks. A bank is a two-dimensional array of DRAM cells, the storing elements of DRAM. Banks offer parallelism as each bank can work independent of the others except for the moment that they output data out of the chip. Each bank also contains a row of sense amplifiers that are working as a buffer for a row of cells. A DRAM cell consists of a capacitor and an access transistor.

Due to the dynamic nature, the charge of the capacitor of the DRAM cell leaks gradually and may lead to a complete loss of the stored information. The time interval during which the information stored in a DRAM cell can be retrieved correctly is called the retention time of the cell. To maintain the integrity of the data stored in the cell, the charge on each capacitor must be periodically restored.

Modern memory chips employ the so-called auto-refresh operation, where the memory controller periodically issues a refresh command, during which the DRAM device refreshes one or more rows per bank depending on the DRAM size. During Auto-Refresh (AF), a DRAM bank cannot serve any request and all rows in a rank remain closed. The charge of the cells is also refreshed implicitly when there is an access to the row, the

refresh-by-access allows even further relaxing of the refresh rate. Later in this work, we will briefly characterize and utilize this property.

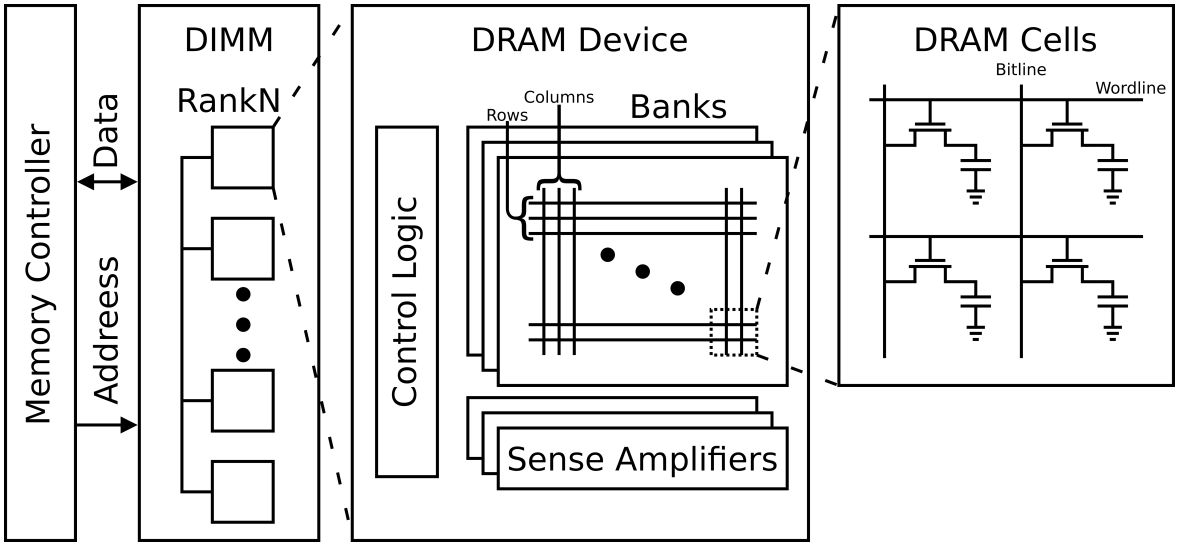

Figure 54: DRAM system organization.

3.2. Literature Review

Liu et al. [34] and Venkatesan et al. [35] have exploited the spatial characteristics of non-uniform retention of DRAM cells to relax DRAM refresh rates. These methods aim at enabling error free DRAM storage by grouping rows into different bins and applying a high refresh rate only for rows belonging to the lower retention bin. However, such multirate-refresh techniques require intrusive hardware modifications, which hinder their use on real systems. More importantly, the error-free storage that these approaches aim at is impossible to achieve in practice, since the retention time of some cells can change over time [36]. Therefore, methods that relax the refresh rates of DRAM should be aware that errors are unavoidable and must be mitigated.

Qureshi et al. [36] show that it may be possible to deal with errors in DRAM with error correcting schemes, however such schemes may incur significant power consumption, which may negate most of the gains from using relaxed refresh rates. A more viable alternative would be to allow errors to be masked by exploiting the inherent error resilient properties of many real-world applications. Recent studies have revealed numerous error-resilience properties in applications, such as probabilistic input data processing, iterative execution patterns that resolve or mitigate errors, algorithmic smoothing of the impact of errors, or user tolerance to small deviations in output quality as Chippa et al. are analyzing in [37]. These error resilience properties provide opportunities to relax the DRAM refresh cycles without concern about the resulting failures on at least some application data.

Venkatesan et al. [35], Liu et al. [38] and Isen et al. [39] are some out of the few that are attempting to exploit application error-resilience to relax DRAM refresh rates. Invariably, these studies perform allocation of data based on their criticality to two different memory domains; one being refreshed at the conventional worst-case rate and the other at a relaxed one. Such methods, hereto referred to as criticality aware data allocation (CADA) have exhibited that application resilience may help to relax refresh rates but are still limited in three key aspects: i) they fail to identify and exploit the inherent ability of applications to refresh their own data by accessing it in memory, a property that we call Refresh-by-Access, ii) they have not employed systematic ways to modulate the classification of data into criticality domains, and iii) they were implemented and evaluated with simulation, thus ignoring the implications of the full system stack, in particular the time-dependencies that exist between data accesses and cell retention. All such limitations indicate missing opportunities for aggressive refresh relaxation and intelligent handling of the potential errors.

Furthermore, while the used DRAM simulators, such as DRAMsim2 [40], are widely acceptable, when concerned about fault-injectors, such as GemFI [41], none of them is taking into account the temporal characteristics of each DRAM access since they are focused only on the spatial variation of the retention time of each cell. Relaxing the refresh rate based on the cell retention time may help to limit the refresh overheads without increasing the failure probability however its evaluation is not complete without considering the fact that during each access the cells is also refreshed. Ignoring this inherent property may lead to incorrect conclusions concerning the incurred faults, which may constrain the amount of the refresh relaxation that can be achieved.

DRAM technologies also achieved their limits in scaling down to smaller cell sizes [42] [43] [44] [45]. The scaling results in a decrease of cell capacitance which, in turn, affects the main memory parameters, such as retention time, write recovery time and the probability of sense amplifier operation failures or a charge loss caused by the leakage current. As follows, scaling of DRAM technology also has negative consequences to reliability.

Thus, characterization of DRAM errors could provide essential hints for software and hardware design which enables mitigation of DRAM faults. This characterization is critical for both emerging and existing technologies where DRAM could be operated under marginal conditions (the relaxed refresh rate or the lowered input voltage)

3.3. Experimental setup

Our experimental setup is based on the two X-Gene2 validation boards that have been delivered to QUB by APM. In D4.1 APM provided a brief description of the X-Gene2 firmware and specification of i2c interfaces to control the main operational parameters of various components inside the development boards, such as supplied chip and DRAM voltages. We note that over the past few months during the first year we worked closely with APM for identifying required changes at the firmware and updating it for providing the needed control over several additional parameters for the DRAM characterization.

APM released the Linux kernel (the default kernel version) which reports all DRAM and cache errors registered by ECC in *syslog*. The kernel receives information about ECC errors from the firmware, and all errors are registered asynchronously. The kernel reports information about source of an error, including DRAM, cache, MCU, bank, rank and so on. It also provides information about the type of error: correctable (single-bit error which could be corrected by ECC) and uncorrectable (double-bit errors which could be only registered by ECC).

Figure 55 demonstrates our experimental framework used to characterize the DRAMs. This framework contains a monitoring tool (which registers ECC errors reported by the kernel and saves all information about errors to a database) and scripts which control the DRAM voltage and refresh rate and run a set of benchmarks. Currently, in our framework, we use 2 micro-benchmarks containing data patterns used in the previous research studies [46] to identify DRAM data consistency errors caused by an inflated DRAM refresh rate: *CBoard-pattern,* CheckerBoard pattern, and random-pattern benchmarks. In the CheckerBoard pattern, consecutive bits alternate between 0 and 1, which may negatively affect retention time behaviour since bitline coupling noise increases with the voltage difference between the coupled bitlines, and storing alternating values in consecutive bitlines maximizes the voltage difference between each bitline and its immediate neighbors [46]. In the random pattern, memory is filled with randomly generated data which makes it possible to explore the effect of various patterns on consistency of data stored in DRAM.

To compare the number of DRAM errors reported by ECC for the micro-benchmarks with the number of errors registered for real benchmarks, we also run Needleman-Wunsch and Kmeans from the Rodinia benchmarks suite [47]. We have limited the data size used by each benchmark 8.64 Gb. We run each benchmark for at least 8 hours and with different number of threads to evaluate the effect of memory access pattern on the total number of DRAM errors reported by ECC. If no error had been reported by ECC during the execution of a benchmark, we were running this benchmark up to 72 hours to register more errors and identify the average number of DRAM faults occurred during the execution. We modified each benchmark to run the computing phase in an infinite loop, while initialization phase is executed only once. Given the fact that the initialization phase in each benchmark is up to several seconds and the computation phase runs for

hours, we expect that the overwhelming majority of reported DRAM errors are related with computing phases. Such modification allows us to estimate the probability to get an error in DRAM operating at the marginal voltage levels during the computational phases. In our experiments, we target particularly the computational phase for each benchmark since it could contain various parallel access patterns apart of the initialization phase which is almost always sequential. In the case of the micro-benchmarks we enclosed the reading phase in an infinite loop where the stored DRAM data are validated.

During the first stage of our research, we stressed the DRAMs operated at the marginal voltage level and the maximum refresh rate. Our experimental APM X-Gene2 boards allow users to control voltage level and the DRAM refresh rate per MCU device through i2c interfaces. APM reports that the refresh period could be changed in the range from 7.8 *usec* to 327.6 *usec*, however we found that in practice the maximum refresh rate which could be set by the user is limited by 280.7 *usec*. Due to limitations of the on-board memory controller, it is also not possible to turn-off the refresh completely and potentially investigate the operation at even more extended margins.

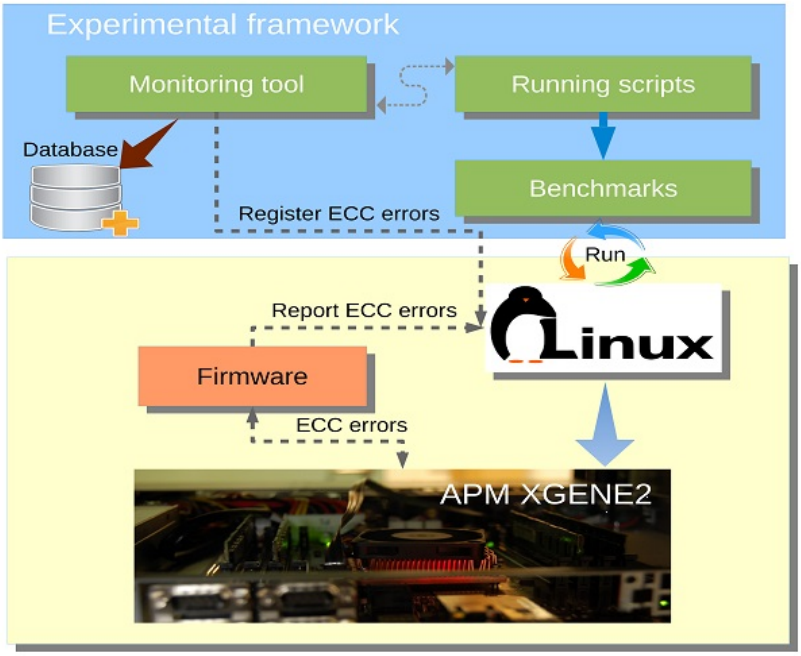

Figure 55: Experimental framework.

3.4. Experiments

3.4.1. The DRAM marginal voltage

By running a set of two benchmarks (*Kmeans* and *Needleman-Wunsch*) and varying the DRAM power rail voltage, we identified the voltage which doesn't lead to a fatal crash (doesn't hang the system) of a board. Our experiments show that the marginal voltage depends on the board and the DIMMs. In particular, we observed that in the first board the voltage could be scaled down to **1.412 V** from the nominal 1.5V and for the second board the voltage can be scaled to **1.423 V** before observing a fatal crash. We note that if we set any DRAM voltage below the marginal voltage for a board then the board crashes immediately in most cases, rarely it crashes in a few seconds, even if there are no running benchmarks.

We also reported that during our experiments with DRAM operating higher than the marginal voltage the boards have never crashed as expected.

3.4.2. Experiments at a scaled voltage

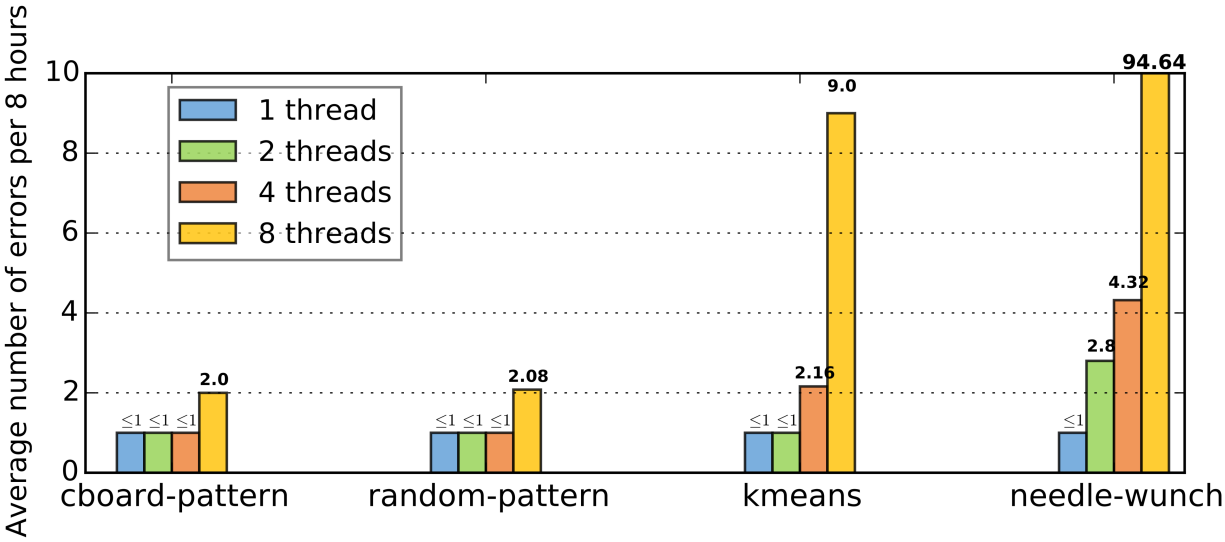

Figure 56: The average number of errors reported by ECC for DRAM operating at the marginal voltage level (1.412 V).

In our first experiment, we collected the number of errors reported by ECC for DRAM operating at the marginal voltage level (1.412 V).

Figure 56 demonstrates the average number of DRAM errors registered with ECC for 8 hours. We see that the number of DRAM errors reported by ECC for 8 hours is less than or equal to 1 on average for all benchmarks, if 1 thread is used. However, this number grows with the number of parallel threads for all benchmarks, which is our first essential finding. More importantly, we see that the micro-benchmarks widely used to characterize DRAM faults produce less number of errors than the real benchmarks from the Rodinia suite when running in parallel. We attribute those results at the dominating effect of memory access patterns on consistency of data in DRAM operating at the marginal voltage in comparison with the effect of data patterns. Thus, the considered earlier micro-benchmarks characterizing DRAM errors should be revised taking into account memory access patterns.

Note that in our experiments, we use ECC-SECDED based Hamming code which supports single-bit error correction and double-bit error detection. When the kernel handles an ECC error, it reports the type of error: single-bit error and double-bit error. We note that single-bit errors are corrected by hardware without any consequences, while double-bit errors are only detected. Thus, only uncorrectable errors could have negative consequences for running applications in terms of reliability, however in our experiments these errors haven't led to a crash of the system or the running benchmarks.

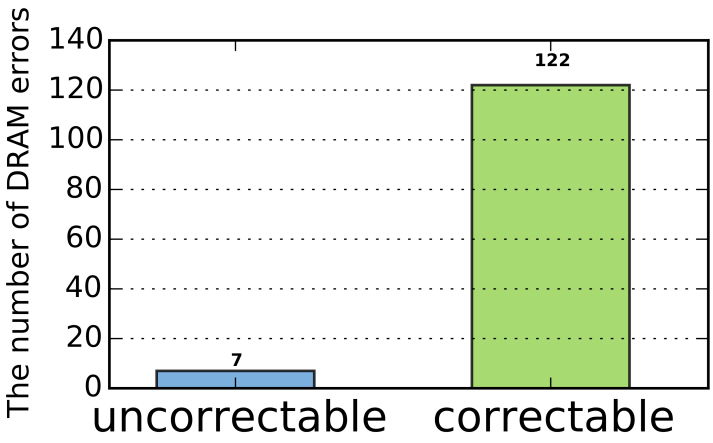

Figure 57: The ratio between correctable and uncorrectable errors for DRAM operating at a scaled voltage.

Figure 57 demonstrates the ratio between correctable errors and uncorrectable errors in this experiment for all benchmarks. In total, we were running the benchmarks for more than **238** hours. Through accumulation of all errors registered in this experiment, we estimated the probability to observe an uncorrectable error for a running application per hour, which is **0.02941**. Nonetheless, note that the failure probability is application specific and varies greatly since all uncorrectable errors were registered during the execution of an 8-thread version of the Needleman-Wunsch benchmark. If the probability to observe an uncorrectable error is zero for a specific application or if the application is resilient to double-bit errors then the DRAM could be operating at the marginal voltage without compromising the output quality of the application, which in turn could help to save 13 % of the DRAM power budget (per DIMM).

3.4.3. Experiments with the refresh rate

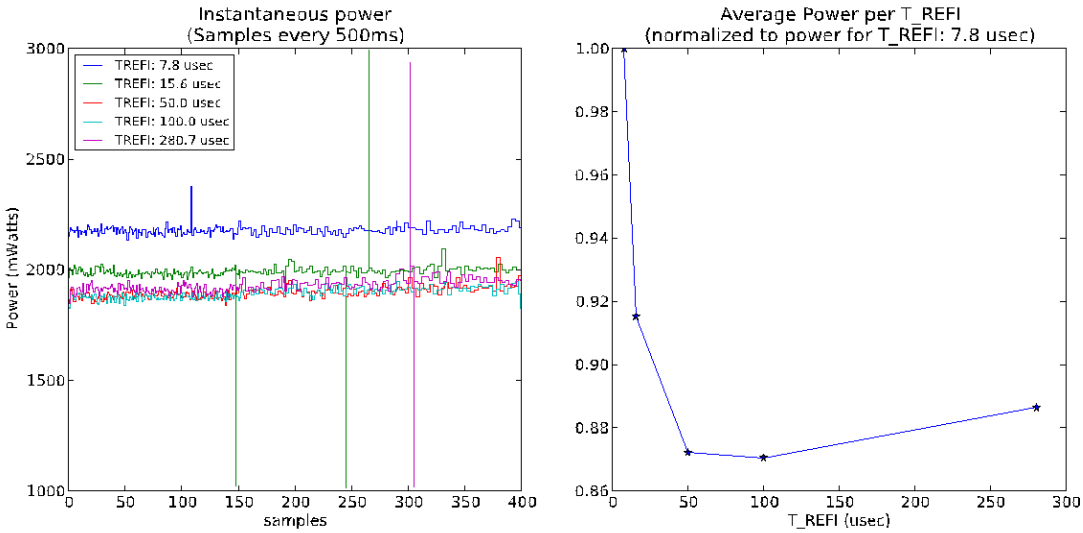

Figure 58: Left: Instantaneous power consumption of Xgene2 board when idle for refresh periods from 7.8 usec (nominal) up to 280.7usec (upper limit in our board). Right: Average power consumption over measuring period for the same refresh period values.

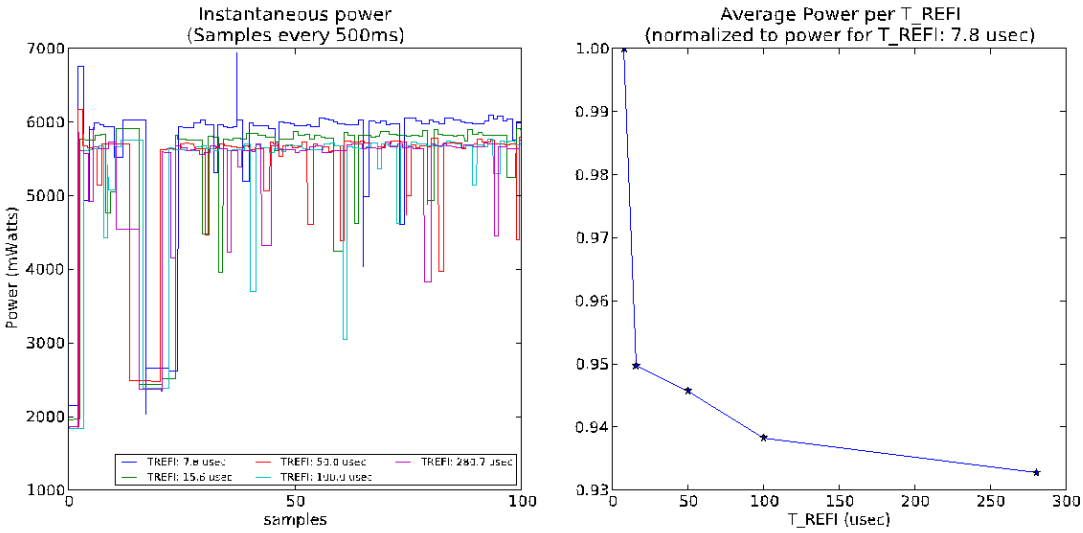

Figure 59: Same experiment as Figure 57 when running the STREAM benchmark.

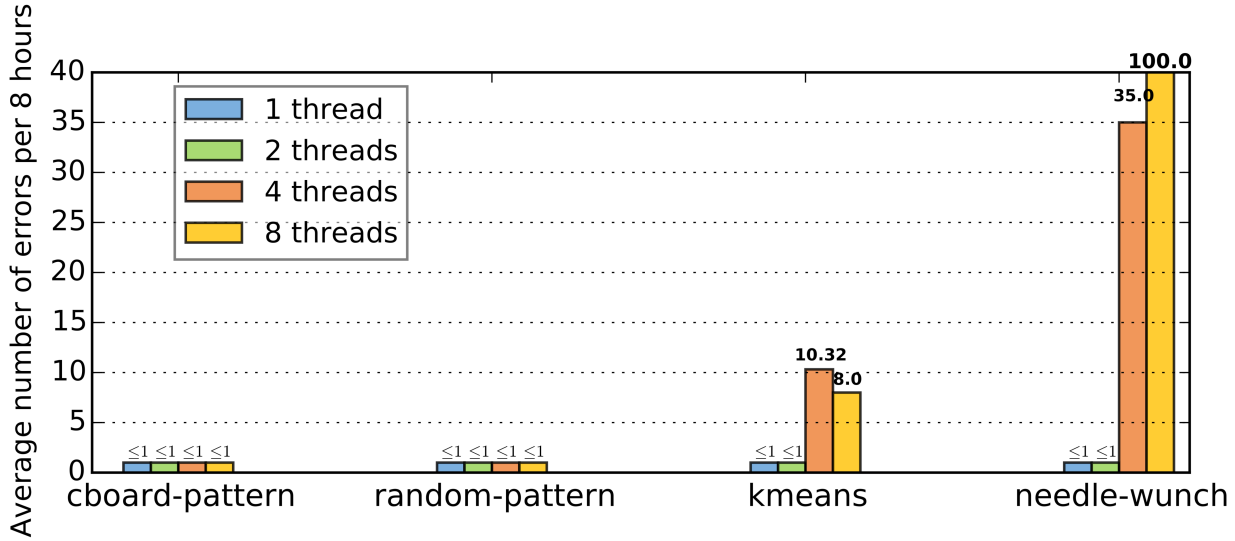

Figure 60: The average number of errors reported by ECC for DRAM operating with the relaxed refresh period (280.7 usec).

To illustrate the potential power gain from relaxing the refresh rate, Figure 58 demonstrates how DRAM power consumption changes with the refresh rate on APM Xgene2, when the board is idle. On the left figure, we depict the instantaneous power consumption taken every 500 msec. The right figure shows the average reduction of power for each setting of the refresh period. We see that we can save up to 13% of power consumption when the machine is idle.

Figure 59 presents the same results when running the STREAM benchmark while measuring. We can see similar trends as before, however a smaller margin for power gains, since in this case the refresh power is a smaller fraction of the total power consumption.

We use a scheme of the experimental setup introduced in the previous section, including the same set of benchmarks, for our experiments with the refresh rate. However, in this part of our research we varied only the refresh rate while DRAM was operating at the nominal voltage. Figure 60 demonstrates the average number of errors reported by ECC per 8 hours for DRAM operating with the maximum available refresh period (280.7 usec). Similar to our experiments with the DRAM voltage, we see that the number of DRAM errors grows up with the number of threads and we clearly observe that the real benchmarks produce more DRAM errors than the micro-benchmarks. Thus, the results of our experiments with the refresh rate also suggest that memory access patterns have dominating impact on consistency of data in DRAM operating with the relaxed refresh period. As follows a future research on characterization of DRAM errors should investigate memory access patterns and its effect on consistency of accessed data stored in DRAM which is operating at the marginal voltage level or with the relaxed refresh period.

During our experiments, we registered only 2 uncorrectable errors among 143 correctable errors. Notably, these two errors were also reported by ECC during the execution of the Needleman-Wunsch benchmark with 8 threads. Due to the fact that correctable errors handled by ECC are transparent to a running application, only uncorrectable errors could lead to a failure of the application and thus have a great importance. The results of this research imply that specific memory access and data patterns could trigger uncorrectable errors, which will be investigated in more detail in the second year.

3.5. Limitations

As we mentioned according to the board manual the refresh period could be changed in the range from 7.8 *usec* to 327.6 *usec*, however we found that in practice the maximum refresh rate which could be set by the user is limited by 280.7 *usec*. Due to limitations of the on-board memory controller it is not also possible to turn-off the refresh completely and potentially investigates the operation at even more extended margins.

Thus, our experiments on refresh characterization are constrained between a refresh rate of 64ms (the nominal) and 280.7 *usec* (which in any case is still 30x less than the nominal one).

Note that in many cases during operation under relaxed refresh and lower supply voltages the system crashes due to errors affecting the kernel data, thus having to restart the experiments.

Currently, one of the boards does not report ECC errors under relaxed refresh rates or scaled supply voltage, even though both boards have the same kernel version and firmware installed, thus we used mainly one of the boards for characterizing the number of correctable and un correctable errors in the DRAMs.

In our experiments, we cannot control the physical location of the data accessed by each benchmark. Thus, the average number of DRAM errors could vary from run to run due to unique placement of data for each benchmark run. However, due to a relatively big size of used memory (8.64 Gb) and duration of each run (at least 8 hours), we suggest that the majority of the effects taking place in memory allocated for data are stable and thus the average number of DRAM faults will stay almost the same between different runs.

Hamming code used by ECC could not identify and fix errors in DRAM if 3 or more bits are corrupted. Thus, all results in this research should be interpreted taking into account this limitation.

3.6. Addressing the Limitations

To address the range of the refresh rate and the potential system crashes we have also developed another setup on a dual-socket commodity server. Each socket hosts an Intel Xeon E5-2650 [48] (Sandy Bridge) processor featuring an integrated memory controller (iMC) to control the DRAM device attached to the socket. Using the iMC register interface, we can enable and disable the refresh operations for each iMC separately, so we opt out for a software refresh mechanism that we devise and implement in order to allow setting the refresh rate at any arbitrary length by enabling and disabling the refresh periodically for emulating the hardware refresh.

Following the fact that each iMC controls a single memory domain attached to the CPU socket, memory domains can be aligned to sockets as shown on Figure 61. As such, DRAMs attached to different CPU sockets may have different refresh periods, implementing memory domains with a variable degree of reliability. In our dual-socket system, the first memory domain is deemed reliable with the nominal refresh applied, while the second one is the variably-reliable one with a configurable refresh.

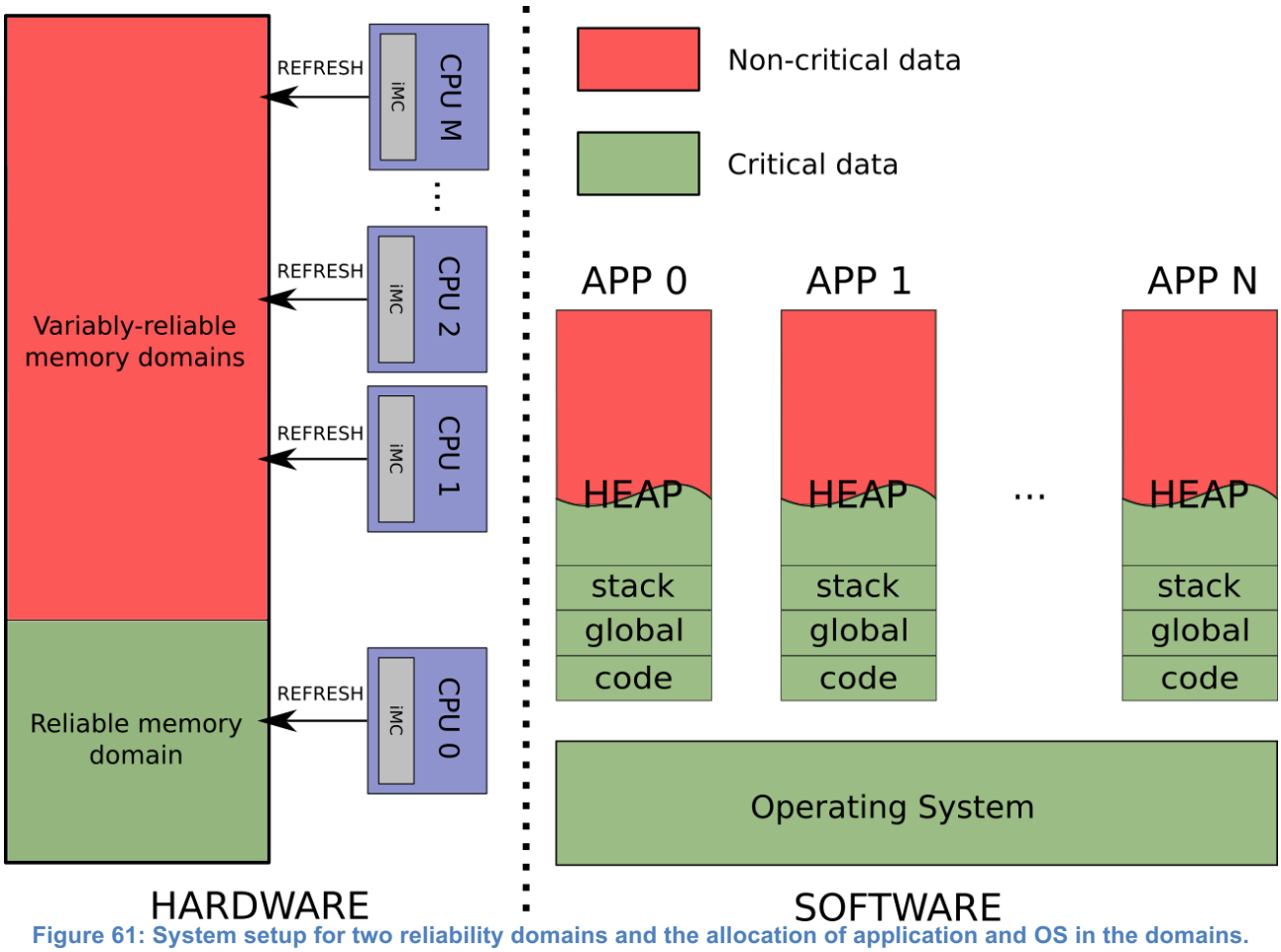

Figure 61: System setup for two reliability domains and the allocation of application and OS in the domains.

In many cases during operation under relaxed refresh and lower supply voltages, the system could crash due to errors. To minimize those, we have instrumented the software platform architecture for enabling variably-reliable memory operation. This includes modifications to the Linux operating system to ensure crash-free system operation and provide a system API to control refresh and data allocation on variably-reliable memory. The first memory domain is the reliable domain for allocating critical data, both of the OS and applications, whereas the second one is the variably-reliable one, reserved for allocating non-critical data of applications.

One can find in the literature a significant body of work that proposes methods for dividing the data of an application in critical and non-critical and storing the latter in unreliable memory in order to save power. We name these methods, Criticality-Aware Data Allocation (*CADA)* policies. On top of CADA, we propose to take into account the effects of implicit refreshes of the memory by means of reading or writing to it through a scheme that we call Data-Access Aware Refresh (DARE). The goal of our scheduling methodology is to minimize the lifetime of data in variably-reliable memory before they are consumed. Tasks that read from data from the variably-reliable memory domain will be executed first, in order to consume those data as soon as possible. Tasks that write to variably-reliable memory will be executed last to reduce the time, between when they were produced until the time they will be consumed (by another part of the algorithm).

Figure 62 shows the results of applying DARE scheduling during the execution of the Sobel filter. On the left-hand side graph, we show the number of errors occurred during the execution of Sobel under default (CADA only) and DARE enabled scheduling, depending on the amount of data, as a percentage of the total Sobel data, stored in variably-reliable memory. On the right-hand graph, we show the effect of DARE scheduling in terms of the quality of output expressed as PSNR of the produced image. DARE scheduling can reduce the number of errors up to an order of magnitude which translates to 3dB better PSNR in the quality of the picture. Apart from improved quality this can translate in increased power benefits. For example, if the user requires a quality of 37dB we need to store up to 25% of the total Sobel data to variably-reliable memory

under default scheduling. With DARE scheduling we can achieve this PSNR storing 2x more data to the variably-reliable memory, increasing its utilizations and thus the potential power benefits.

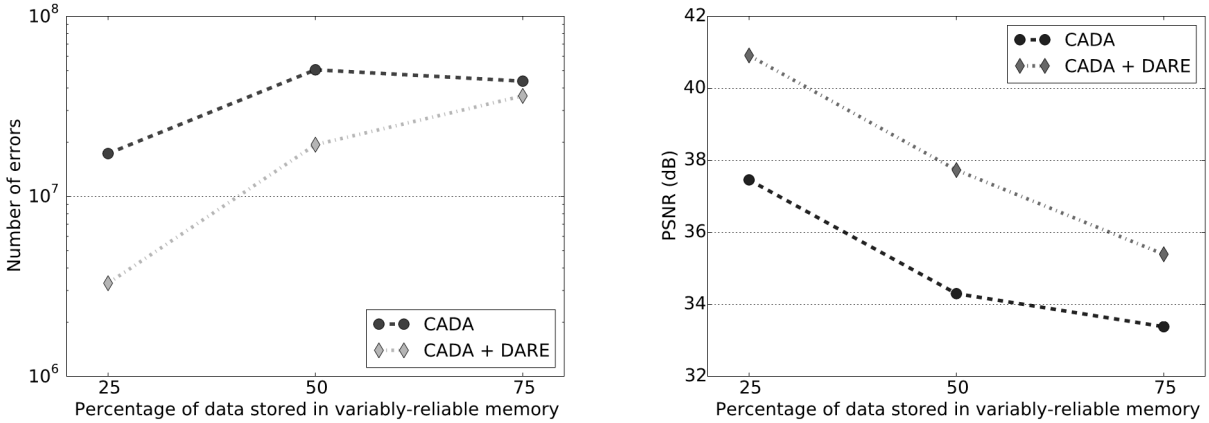

Figure 62: Benefits of DARE scheduling. Left: number of errors with amount of data stored in variably-reliable memory. Right: Quality achieved with amount of data stored in variably-reliable memory.

3.7. Simulator for characterizing access patterns

As previously shown, one significant attribute of DRAM is the temporal characteristics of each DRAM access and the refresh-by-access. Depending on the frequency of access for all the non-critical data and their access patterns, we can further decrease the refresh rate of the DRAM.

Our experiments have indicated that various popular applications have such access pattern and thus refresh their data frequently, a property that can be utilized for further reducing and even turning off the refresh during execution, complementing the existing schemes.
In order to characterize the applications and identify their access pattern, we have developed a framework in which we can characterize the frequency of accesses in the data-structures of interest, the access pattern and the absolute timings between accesses. Furthermore, knowing the timings of each access, we can fault-inject emulating the realistic errors that are injected into the applications while the system is running on relaxed refresh rate.

In this framework, we are using a trace-based technique to speed up the timing extraction that is based on running natively on top of dynamic binary instrumentation tools, specifically using Intel® Pin [49]. Our framework works in distinct phases to extract different measurements resulting in having a timing memory trace. For each instruction that has access in the memory, we instrument a jump to a tool function as shown in Figure 63. We are keeping track of the timings of the instrumentation functions and those of the accesses, in a sequence of runs, starting with running the benchmark natively, continuing with very light instrumentation and finally running with all instrumentation enabled. In each step, we are measuring the execution time, the overheads that are introduced by our framework and the metrics in interest for that step.

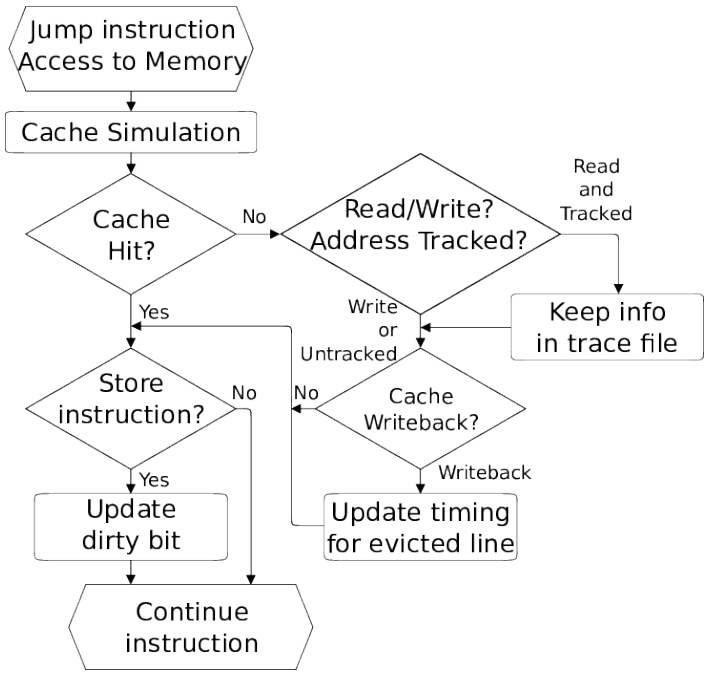

Figure 63: Data flow for the instrumentation called for each instruction that accesses the memory.

Figure 64 shows the measured metrics from an artificial benchmark that we have developed for validation of the framework. Specifically, on the left figure we can see a graph across the execution time of the benchmark, the number of last level cache misses and the average access interval on the dashed line. On the right figure, we see the distribution of accesses based on the duration of the last accessed time for each line.

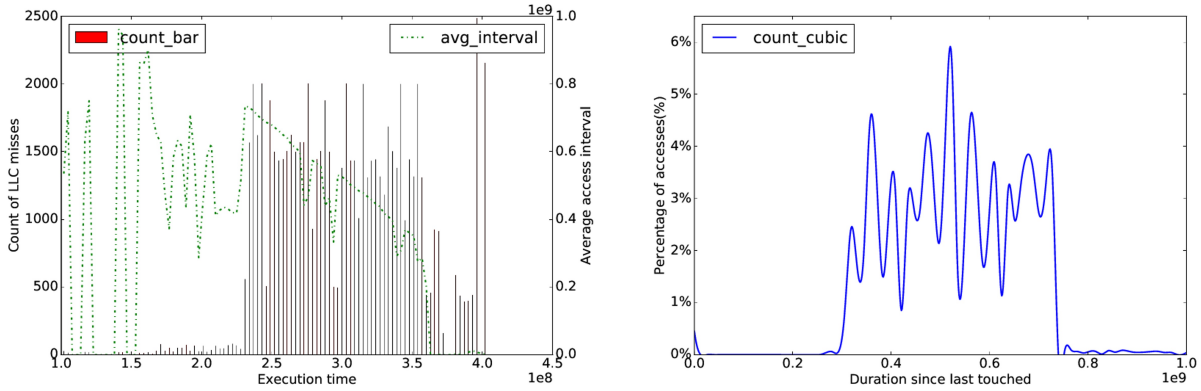

Figure 64: The metrics measured by the simulator. Left: the number of LLC misses and the average access interval during the execution of a benchmark. Right: shows the distribution of the accesses based on the duration that the data remained untouched.

Furthermore, this solution allows us to fault-inject applications based on the timing trace by having functional simulation of the application, also running on top of Intel® Pin. We are using the information extracted from the real system such as, the timing trace of each access (taking into account that each access refreshes the memory data line), the cumulative distribution function of the bit-errors for different refresh rates and the probability for errors while having increased traffic to the memory, to accurately generate a trace that consists of the errors that should be injected on the functional simulator. In this way, we can evaluate the behaviour of the application with errors (crashing, running correctly or with wrong output) and quantify the degraded quality of the output.

4. Conclusions and Future Research

We developed a versatile framework for system-level voltage and frequency scaling characterization built on top of the ARMv8-compliant APM's X-Gene 2 micro-server family. The framework is fully automated and reports information supported by the hardware itself, such as cache ECC errors, as well as SDCs, system or process crashes, and hangs. We present the challenges for the development of such a framework, and describe how we overcame these challenges, by using and combining several hardware, software and system engineering techniques. We also proposed a new metric for both aggregating the results produced by multiple runs due to non-deterministic executions, and quantifying the microprocessor's ability to operate beyond nominal conditions. Finally, we presented some experimental results, which demonstrate potential uses of the characterization framework to identify the limits of operation and support system-level design decisions for improved energy efficiency.

The next steps on T3.3 regarding cache memories include:

- Expansion of the existing result set with more benchmarks and runs.
- Experiments on L3/SOC voltage domain.
- Experiments using TFF and TSS chips.
- Investigation on the attributes that affect system margins.
- Development of viruses for the remaining memory hierarchy.

Moreover, in this research, we investigated how applications with different data and accesses patterns affect the number of errors for DRAM operating at the marginal input voltage and relaxed refresh rates. We discovered that the number of DRAM faults could vary significantly and it is related with a running application. Our research revealed that the real benchmarks from the Rodinia suite produce more errors than the data pattern micro-benchmarks considered in the previous research studies to characterize DRAM errors. However, our major finding in this research is that memory access patterns have a dominating impact on consistency of accessed data for DRAM operating at the lowered input voltage or with the relaxed refresh rate.

In future research, we will further investigate the effect of memory access patterns on consistency of accessed data in DRAM in more detail. We will try to identify specific memory access scenarios which trigger DRAM faults. We will revise the micro-benchmarks characterizing DRAM errors taking into account the memory access patterns. We will try to extend the simulator to work with a range of different metrics that could be useful for understanding the behaviour of the applications and for online estimation of increased probability for errors, without having to tolerate the heavy overhead of instrumentation currently implemented.

5. References

[1] C. Wilkerson, H. Gao, A. R. Alameldeen, Z. Chishti, M. Khellah, and S.-L. Lu, “Trading off cache capacity for reliability to enable low voltage operation”, in International Symposium on Computer Architecture (ISCA), 2008, pages 203–214.

[2] Z. Chishti, A. R. Alameldeen, C. Wilkerson, W. Wu, and S.-L. Lu, “Improving cache lifetime reliability at ultra-low voltages”, in International Symposium on Microarchitecture (MICRO), 2009, pages 89–99.

[3] A. Bacha, and R. Teodorescu, “Dynamic reduction of voltage margins by leveraging on-chip ECC in Itanium II processors”, in International Symposium on Computer Architecture (ISCA), 2013, pages 297–307.

[4] A. Bacha, and R. Teodorescu, “Using ECC feedback to guide voltage speculation in low-voltage processors”, in International Symposium on Microarchitecture (MICRO), 2014, pages 306–318.

[5] A. Bacha, and R. Teodorescu, “Authenticache: Harnessing cache ECC for system authentication”, in International Symposium on Microarchitecture (MICRO), 2015, pages 128–140.

[6] H. Duwe, X. Jian, D. Petrisko, and R. Kumar, “Rescuing uncorrectable fault patterns in on-chip memories through error pattern transformation”, in International Symposium on Computer Architecture (ISCA), 2016, pages 634–644.

[7] M. S. Gupta, V. J. Reddi, G. Holloway, G.-Y. Wai, and D. M. Brooks, “An event-guided approach to reducing voltage noise in processors”, in Design, Automation & Test in Europe Conference (DATE), 2009, pages 160-165.

[8] V. J. Reddi, M. S. Gupta, G. H. Holloway, G.-Y. Wei, M. D. Smith, and D. M. Brooks, “Voltage emergency prediction: Using signatures to reduce operating margins”, in International Conference on High-Performance Computer Architecture (HPCA), 2009, pages 18–29.

[9] J. Leng, A. Buyuktosunoglu, R. Bertran, P. Bose, V. J. Reddi, “Safe limits on voltage reduction efficiency in GPUs: a direct measurement approach”, in International Symposium on Microarchitecture (MICRO), 2015, pages 294–307.

[10] C. R. Lefurgy, A. J. Drake, M. S. Floyd, M. S. Allen-Ware, B. Brock, J.A. Tierno, J. B. Carter, “Active management of timing guardband to save energy in POWER7”, in International Symposium on Microarchitecture (MICRO), 2009, pages 1–11.

[11] V. J. Reddi, S. Kanev, W. Kim, S. Campanoni, M. D. Smith, G.-Y. Wei, and D. Brooks, “Voltage smoothing: Characterizing and mitigating voltage noise in production processors via software-guided thread scheduling”, in International Symposium on Microarchitecture (MICRO), 2010, pages 77–88.

[12] M. S. Gupta, K. K. Rangan, M. D. Smith, G.-Y. Wei, and D. Brooks, “Towards a software approach to mitigate voltage emergencies”, in International Symposium on Low Power Electronics and Design (ISPLED), 2007, pages 123-128.

[13] R. Joseph, D. Brooks, and M. Martonosi, “Control techniques to eliminate voltage emergencies in high performance processors”, in International Conference on High-Performance Computer Architecture (HPCA), 2003, pages 79–90.

[14] T. N. Miller, R. Thomas, X. Pan, and R. Teodorescu, “VRSync: Characterizing and eliminating synchronization-induced voltage emergencies in many-core processors”, in International Symposium on Computer Architecture (ISCA), 2012, pages 249–260.

[15] M. D. Powel, and T. N. Vijaykumar, “Pipeline muffling and a priori current ramping: architectural techniques to reduce high-frequency inductive noise”, in International Symposium on Low Power Electronics and Design (ISPLED), 2003, pages 223-228.

[16] M. S. Gupta, K. K. Rangan, M. D. Smith, G.-Y. Wei, and D. Brooks, “DeCoR: A Delayed Commit and Rollback mechanism for handling inductive noise in processors”, in International Conference on High-Performance Computer Architecture (HPCA), 2008, pages 381–392.

[17] M. Ketkar, and E. Chiprout, “A microarchitecture-based framework for pre- and post-silicon power delivery analysis”, in International Symposium on Microarchitecture (MICRO), 2009, pages 179–188.

[18] Y. Kim, and L. K. John, “Automated di/dt stressmark generation for microprocessor power delivery networks”, in International Symposium on Low Power Electronics and Design (ISPLED), 2011, pages 253-258.

[19] Y. Kim, L. K. John, S. Pant, S. Manne, M. Schulte, W. L. Bircher, and M. S. S. Govindan, “AUDIT: Stress Testing the Automatic Way”, in International Symposium on Microarchitecture (MICRO), 2012, pages 212–223.

[20] Y. Zu, C. R. Lefurgy, J. Leng, M. Halpern, M. S. Floyd, and V. J. Reddi, "Adaptive guardband scheduling

to improve system-level efficiency of the POWER7+", in International Symposium on Microarchitecture (MICRO), 2015, pages 308-321.
[21] The Raspberry Pi Documentation, https://www.raspberrypi.org/documentation
[22] The Linux Kernel Documentation (Parent Directory), https://www.kernel.org/doc/Documentation.
[23] J. J. Dongarra, P. Luszczek, and A. Petitet, "The LINPACK benchmark: past, present and future", Concurrency and computation: practice and experience. 15(9), 2003, pp. 803-820.
[24] J. L. Henning, "SPEC CPU2006 benchmark descriptions," in SIGARCH Computer Architecture News, Volume 34, Issue 4, September 2006, pages 1-17.
[25] C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Kistler, and T.W. Keller, "Energy management for commercial servers", in Computer Journal, Volume 36, Issue 12, December 2003, pages 39-48.
[26] G. Atwood, "Current and emerging memory technology landscape", 2011. URL http://www.flashmemorysummit.com/English/Collaterals/Proceedings/2011/20110811_S303_Atwood.pdf , flash Memory Summit.
[27] U. Kang, H. Soo Yu, C. Park, H. Zheng, J. Halbert, K. Bains, S. Jang, and J.S. Cho, "Co-architecting controllers and dram to enhance dram process scaling".
[28] S. Hong, "Memory technology trend and future challenges", in International Electron Devices Meeting (IEDM), 2010; 12.4.1–12.4.4, doi:10.1109/IEDM.2010.5703348.
[29] K. Kim, "Future memory technology: challenges and opportunities", in International Symposium on VLSI Technology, Systems and Applications (VLSI-TSA), 2008; 5–9, doi:10.1109/VTSA.2008.4530774.
[30] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable dram alternative", in SIGARCH Computer Architecture News, Volume 37, Issue 3, June 2009, pages 2-13.
[31] J. A. Mandelman, R.H. Dennard, G.B. Bronner, J.K. DeBrosse, R. Divakaruni, Y. Li, and C.J. Radens, "Challenges and future directions for the scaling of dynamic random-access memory (dram)", in IBM Journal of Research and Development, Volume 46, Issue 2-3, March 2002, pages 187-212.
[32] O. Mutlu, and L. Subramanian, "Research problems and opportunities in memory systems", in International Journal in Supercomputing Frontiers and Innovations, Volume 1, Issue 3, October 2014, pages 19-55.
[33] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu, "RAIDR: Retention-aware intelligent dram refresh", in 39th International Symposium on Computer Architecture (ISCA), 2012, Volume 40, Issue 3, June 2012, pages 1-12.
[34] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu. RAIDR: Retention-aware intelligent dram refresh. In Proceedings of the 39th Annual International Symposium on Computer Architecture, ISCA '12, pages 1–12, Washington, 2012.
[35] R.K. Venkatesan, S. Herr, and E. Rotenberg. "Retention-Aware Placement in DRAM (RAPID): Software Methods for Quasi-Non-Volatile DRAM", in The Twelfth International Symposium on High-Performance Computer Architecture, 2006., pages 157–167. IEEE, 2006.
[36] M. K. Qureshi, D. H. Kim, S. Khan, P. J. Nair, and O. Mutlu. "Avatar: A variable-retention-time (vrt) aware refresh for dram systems", in Proceedings of the 2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN '15, pages 427–437, Washington, 2015.
[37] V. K. Chippa, Srimat T. Chakradhar, K. Roy, and A. Raghunathan. "Analysis and characterization of inherent application resilience for approximate computing", in Proceedings of the 50th Annual Design Automation Conference, DAC '13, pages 113:1–113:9, New York, NY, USA, 2013.
[38] S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn. "Flikker", in ACM SIGPLAN Notices, 46(3):213, Mar 2011.
[39] C. Isen and L. John, "ESKIMO - energy savings using semantic knowledge of inconsequential memory occupancy for DRAM subsystem," 2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), New York, NY, 2009, pp. 337-346.
[40] P. Rosenfeld, E. Cooper-Balis, and B. Jacob. "Dramsim2: A cycle accurate memory system simulator", IEEE Computer Architecture Letters, 10(1):16–19, Jan 2011.
[41] K. Parasyris, G. Tziantzoulis, C. D. Antonopoulos, and N. Bellas. "Gemfi: A fault injection tool for studying the behavior of applications on unreliable substrates", in 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, pages 622–629, June 2014.
[42] G. S. Sandhu, "Emerging memories technology landscape," 2013 13th Non-Volatile Memory Technology Symposium (NVMTS), Minneapolis, MN, 2013, pp. 1-5.
[43] S. Hong, "Memory technology trend and future challenges," 2010 International Electron Devices Meeting, San Francisco, CA, 2010, pp. 12.4.1-12.4.4.
[44] K. Kim, "Future memory technology: challenges and opportunities," 2008 International Symposium on

VLSI Technology, Systems and Applications (VLSI-TSA), Hsinchu, 2008, pp. 5-9.
[45] O. Mutlu and L. Subramanian. “Research Problems and Opportunities in Memory Systems”, Supercomput. Front. Innov.: Int. J. 1, 3 (October 2014), 19-55.
[46] J. Liu, B. Jaiyen, Y. Kim, C. Wilkerson, and O. Mutlu, “An experimental study of data retention behavior in modern dram devices: Implications for retention time profiling mechanisms”, in SIGARCH Computer Architecture News, Volume 41, Issue 3, June 2013, pages 60-71.
[47] S. Che, M. Boyer, J. Meng, D. Tarjan, J.W. Sheaffer, S.H. Lee, K. Skadron, “Rodinia: A benchmark suite for heterogeneous computing”, in Proceedings of the 2009 IEEE International Symposium on Workload Characterization (IISWC), 2009, pages 44-54.
[48] Intel. “Intel Xeon processor E5-1600/2400/2600/4600 (E5-product family) product families datasheet”, volume two, 2012.
[49] C. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood. “Pin: building customized program analysis tools with dynamic instrumentation”, SIGPLAN Not. 40, 6 (June 2005), 190-200.

[END OF DOCUMENT]

