MISSIONE 4
ISTRUZIONE
RICERCA

SPOKE 1

FUTURE HPC & BIG DATA

FLAGSHIP 1: Selection of Candidate Prototypes for Power Management and Energy/Reliability Monitoring Platforms

Finanziato dall'Unione europea
NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani
PIANO NAZIONALE DI RIPRESA E RESILIENZA
ICSC
Centro Nazionale di Ricerca in HPC, Big Data and Quantum Computing

EXECUTIVE SUMMARY

This document introduces, analyzes, and describes the candidate prototypes (e.g., computational models, frameworks, programming environments, tools, test beds, and software artifacts) to be used as instruments for the evaluation and validation of methods, techniques, or strategies for power management and energy/reliability monitoring, according to the main objectives described in the milestone #5, Spoke 1 - Flagship 1.

Chapter 1 briefly describes the main objectives of the document and main targets in the selection of candidate prototypes for the evaluation of two crucial non-functional properties in HPC systems (*Energy* and *Reliability*).

Chapter 2 analyses and describes the main candidate solutions for evaluating power management features in HPCs. In general, a considerable number of frameworks, platforms, and prototypes are evaluated, focusing mainly on the most advanced features and critical components. Particular attention has been paid to the analysis and automatic generator of power monitors for RTL components (e.g., hardware accelerators), the contribution to the energy related to the memory subsystems, and the power monitoring in the large when considering the entire data center. Moreover, it also included the description of a possible solution for cooling exploiting a 2-phase liquid technology.

Chapter 3 analyses the possible candidates for evaluating reliability in several structures of commodity clusters for HPCs and their and management platforms. This chapter is divided into five main sections. The first section identifies and analyses possible tools, frameworks, and models to perform reliability evaluations. Then, the second section addresses the candidates for developing and validating hardening approaches. Furthermore, the third section targets identifying system-level online monitoring prototype platforms and frameworks. Then, the fourth section targets the identification of frameworks for the performance specifications in HPC, targeting the detection of possible anomalies. Finally, the fifth section addresses the identification of frameworks, environments, models, and simulators for system-level fault tolerance and real-time applications.

Chapter 4 overviews and analyses the main tools, software artifacts, libraries, frameworks, and environments that can be used for performance evaluation in HPC systems.

Finally, Chapter 5 provides concluding remarks and emphasizes the main research opportunities and open questions to be developed through the identified candidate prototypes.

AUTHORS (in alphabetical order):

- Giovanni Agosta, Politecnico di Milano
- Enrico Bini, Università degli studi di Torino
- Daniele Cattaneo, Politecnico di Milano
- Daniele Cesarini, CINECA
- Federico Tesser, CINECA
- William Fornaciari, Politecnico di Milano
- Andrea Galimberti, Politecnico di Milano
- Alberto Garfagnini, Università degli studi di Padova
- Marco Lapegna, Università degli studi di Napoli Federico II
- Gabriele Magnani, Politecnico di Milano
- Gabriele Mencagli, Università di Pisa
- Cecilia Metra, Università degli studi di Bologna
- Martin Eugenio Omana, Università degli studi di Bologna
- Filippo Palombi, ENEA
- Federico Reghenzani, Politecnico di Milano
- Josie E. Rodriguez Condia, Politecnico di Torino
- Michele Scquizzato, Università degli studi di Padova
- Matteo Sonza Reorda, Politecnico di Torino
- Davide Zoni, Politecnico di Milano

LIST OF ABBREVIATIONS:

AI: Artificial Intelligence
APU: Accelerated Processing Unit
ASIC: Application Specific Integrated Circuit
CPU: Central Processing Unit
DLP: Deep Learning Processors
DPU: Deep Learning Processor Unit
DSA: Domain-Specific Architecture
DVFS: Dynamic Voltage and Frequency Scaling
ECC: Error-Correcting Codes
Exascale: refers to computing systems capable of calculating at least 10^{18} IEEE 754 Double Precision (64-bit) operations per second.
FIFO: First-In First-Out
FLOPS: Floating-Point Operations Per Second
FMA: Fused Multiply-Add unit
FPGA: Field-Programmable Gate-Array
GPGPU: General-Purpose Graphics Processing Unit
GPU: Graphics Processing Unit
HPC: High-Performance Computing
IoT: Internet of Things
IPU: Intelligence Processing Unit
ISA: Instruction set Architecture
LRU: Least Recently Used
NBTI: Negative-Bias Temperature Instability
NFP: Non-functional properties
NoC: Network in Chip
OVI: Open Vector Interface
PBTI: Positive-Bias Temperature Instability
PIM: Processing In-Memory
PUE: Power Usage Effectiveness
SIMD: Single-Instruction Multiple-Data
SIMT: Single-Instruction Multiple-Thread
SoC: System on Chip
SPU: Scalar Processing Unit
TCU: Tensor Core Unit
VLIW: Very Long Instruction Word
VPU: Vector Processing Cores/Units

Contents

1 INTRODUCTION

Representative prototypes, models, and frameworks are essential tools in engineering and science for the development, evaluation, validation, and test of ideas, concepts, and processes [1], [2]. Moreover, these tools provide support and feedback when analysing new solutions, methods, approaches, schemes, or architectures. In fact, prototypes reduce the gap between first stages of an idea and the realistic implementations by supporting analyses of a system under clear evaluation targets (e.g., evaluate the performance of a multi-processor under specific operational conditions).

In general, most prototypes provide a set of controlled conditions to evaluate the technical feasibility of theories in environments and scenarios that are similar to the real operation of a system. The quality and functionality of a prototype is given by its fidelity, so high-fidelity models and prototypes include most of the expected functionalities and details of a real system. Similarly, low-fidelity prototypes include a reduced number of features but still allow the evaluation and analysis under limited scenarios. In some cases, one single high-fidelity prototype involves all possible operative scenarios and can be exploited for evaluation and analysis. On the other hand, one or more high/medium-fidelity and complementary prototypes (i.e., with different abstraction levels) might contribute to evaluate different features and provide consistency, high accuracy, or more extended analyses than one single prototype. Similarly, the use of prototypes can be used to anticipate behaviors in a system and also identify unexpected constraints (e.g., physical, technical, or financial) before reaching production phases. Thus, the identification and selection of feasible prototypes and models is a crucial step for the evaluation of several features also in the HPC domain. In fact, one or more prototypes are vital tools to evaluate solutions and management mechanisms against non-functional properties in HPC machines, such as the power management and the monitoring of energy or reliability features.

This delivery report analyses and selects a set of prototype candidates, frameworks, tools, and artifacts for the evaluation of three main non-functional properties affecting modern and future generations of HPC systems: *i) Power*, *ii) Reliability*, and *iii) Performance*.

Chapter 2 describes and analyses the most feasible prototype candidates, methods, frameworks, and tools for the evaluation of power features in HPC machines. Furthermore, this chapter analyze the different management platforms, tools, methods, and artifacts for power handling. First, this chapter introduces the most feasible prototype candidates for the generation of on-line power estimators, to be possibly employed in conjunction with approximate computing coding, to better balance accuracy and energy requirements. Then, a set of prototypes are analyzed for the monitoring of power and energy in memories and HPC infrastructure. Finally, we describe a realistic experimental setup by resorting to a proof-of-concept for HPC system's cooling.

Chapter 3 mainly focuses on the identification, report, and analysis of a set of candidate prototypes, models, methods, frameworks, and strategies for the evaluation of reliability at different levels. First, the chapter identifies the possible prototypes to allow the evaluation of reliability features in all abstraction levels (from the transistor and micro-architectural levels up to application and system levels of operation). In this case, the feasible prototypes are selected to allow the analysis and support the identification of possible error effects caused by propagated faults across the different components of a HPC system. At the same time, we identify the most feasible evaluation strategies and highlight methods and frameworks to support the reliability assessment. Then, A set of tools and frameworks are identified to support the analysis and evaluation of hardening, online monitoring, and fault management mechanisms. Finally, this chapter identifies frameworks and artifact to support the evaluation of anomaly detection resorting to automata-based languages, as well as, a set of feasible techniques for the evaluation of system-level fault tolerance mechanisms under real-time constraints.

Chapter 4 introduces and highlights the main prototype candidates for the evaluation of performance features in HPC machines. In particular, we analyze prototypes for monitoring in terms of kernel-based job execution, cloud infrastructure, parallel workloads and shared resources, such as memories. Moreover, the COUNTDOWN library is introduced as clever mechanism to optimize energy consumption, while monitoring distributed workloads.

Finally, Chapter 5 provides some concluding remarks.

2 POWER EVALUATION AND MANAGEMENT PLATFORMS

Energy, power and thermal monitoring and management are becoming crucial aspects to be considered when designing a modern HPC infrastructure. Such features are highly crross-related and the contributions of several components of the computing platform need to be taken into account to provide an holistic view of the scenario to be managed. This section focuses on some of the most important elements responsible of the dissipation of power and on the strategies to mitigate the possible rising of the operating temperature by using innovative cooling solutions. We present strategies that are state of the art and sometime not yet fully moved into the level of maturity enabling the technology transfer at this initial stage of the project.
Since we are considering heterogeneous computing for high demanding applications, the contribution to the power of hardware accelerators, implemented on FPGAs, cannot longer be neglected. This is the goal of presented methodology in Section 2.1 for the automatic generation of all-digital power monitors implemented in hardware. The energy-related aspects concerning the memory, given its specific peculiarities, has been addressed in a different dedicated space in Section 2.3. Moving up the granularity at which the system in considered, Section 2.2 turns the discussion into the view of the power and energy monitoring that is familiar to the engineers responsible of the management of a supercomputing center. The path from the generation of power to the transfer of the heat outside of the computing platform, culminates in the analysis of the existing cooling techniques for high-power systems, with some insights on the most advanced and still experimental solutions exploiting a 2-phase transition cooling (Section 2.4).

2.1 Semi automatic generation of on-line power estimators implemented in hardware

The design of all-digital run-time power monitoring infrastructure represents a critical innovation that allow us to precisely monitor the power consumption of any hardware accelerator within large FPGAs for HPC with the possibility of reconfiguring the power monitoring at any change in the set of hardware accelerators implemented into the FPGA. Notably, the run-time power monitoring infrastructure enables traditional power- and energy-aware control policies on large FPGAs also allowing to monitor the power consumption of each portion of the FPGA according to the application and system-level requirements.

Figure 1 illustrates the proposed toolchain to generate a hardware-level resource-constrained power monitor for generic computing platforms in an automatic way. Starting from the hardware description of the computing platform (`RTL_Source`), its corresponding set of design constraints (`Design Constraints`), a Testbench (`Testbench`), the user-defined constraints (`User-defined Constraints`), and the technology library files (`Tech-lib`), the flow outputs an augmented RTL netlist containing the original RTL design coupled with the run-time power monitoring infrastructure (`Augmented RTL-netlist`). We note that the design constraints are expressed in terms of timing and physical requirements e.g., respectively, operating frequency and pinout, for the implemented computing platform. On the contrary, the user-defined constraints allow the user to specify the number of allowed resources to implement the power monitoring infrastructure.

The design and implementation of the power monitor is organized in four different steps: (i) Simulation, (ii) Data extraction, (iii) Model Identification and (iv) Power Monitor Implementation.

The simulation step simulates the design with the Testbench and the provided constraints and generates two files containing power values (SAIF – Switching Activity Interchange Format) and switching activity information (VCD – Value Change Dump). The data extraction step parses the SAIF and VCD files preparing and filtering the data for the step (iii).

The Model Identification step identifies a power model that ensures the smallest accuracy error within the resource budget, by leveraging: (i) the switching activity of the signals, (ii) the power consumption of the computing platform and (iii) the user-imposed resource constraint. Finally, the Power Monitor Implementation stage augments the RTL description of the computing platform with a power monitoring infrastructure which implements the identified power model.

Power model identification The power model identification step produces a linear model starting from three inputs: (i) the power consumption traces, (ii) the corresponding switching activity of the target computing platform, (iii) the user-defined resource constraints. In particular, the power model employs the switching activity of a selected subset of input and output wires of the design modules to derive the power estimates. To describe the power model, we adopted the formulation proposed in [3]. Its formal definition is:

$$P\Omega \sum_{i} c_i \times s_{i,t}^{SVC} \sum_{j} c_j \times s_{j,t}^{HWC} \tag{1}$$

At time t, the power estimate $P\Omega$ is computed as the weighted sum of the switching activity of a carefully identified subset of signals, i.e., $s_{i,t}$ and $s_{j,t}$, where the values of the coefficients c_i and c_j are tuned by the identification algorithm. To improve the quality of the information associated with the switching activity used to calculate the power estimates, the power model formulation includes the switching activity of each signal either in the form of either the Hamming Weight (HWC) or the Single Variation Count (SVC). For each clock cycle, the HWC counts the number of bits of the signals that changed with respect to the previous sampled value. In contrast, SVC measures the switching activity, as 0 or 1, if the current signal value is changed or not with respect to the previously sampled value. Even if the HWC and SVC statistics of the same physical signal can be hardly employed in the same power model due to correlation issues, the availability of multiple models to measure the switching activity can improve the final accuracy of the power model.

Power modeling The power model identification algorithm employs a recursive approach to implement the top-down hierarchical visit of the target design. The algorithm takes five inputs: (i) the top module of the design, (ii) the user-defined constraints, where each of them is specified as a fraction of the same resource type used by the target design also including an upper bound that specifies the maximum acceptable accuracy error for the identified model, (iii) profiled information from the Power Monitor RTL description module and finally, the (iv) the switching activity and the (v) power traces of the target computing platform. The mathematical formulation of the identified power model represents the output of the algorithm. The power model is a list of triples, where each triple is defined as the name of a signal of the design to monitor, the estimated coefficient, and the employed switching activity that is selected as either the Single Variation Count (SVC) or the Hamming Weight Count (HWC).
In the following of the project, a proper template for the generation of the implementation of the power models will be developed, in order to make possible the automatic generation of all-digital power monitors.

Power control exploiting approximate computing The generated power monitors can be employed as the sensing components of a control loop, where the actuators are represented by the choice of the different implementations of the kernels generated by a mixed precision compiler tool. Based on the analysis of the state of the art provided in the first report " Survey of State-of-the-art Approaches and Gap Analysis of RISC-V Platform Requirement for HPC", we plan to employ the TAFFO [4] plug-in set for LLVM for the precision tuning task. Note that the same type of knob, i.e. an implementation of the application exploiting a different data representation, can be used as part of other more general control strategies, considering for instance the entire set of applications composing the workload and their possible level of criticality related to the accuracy of the results.

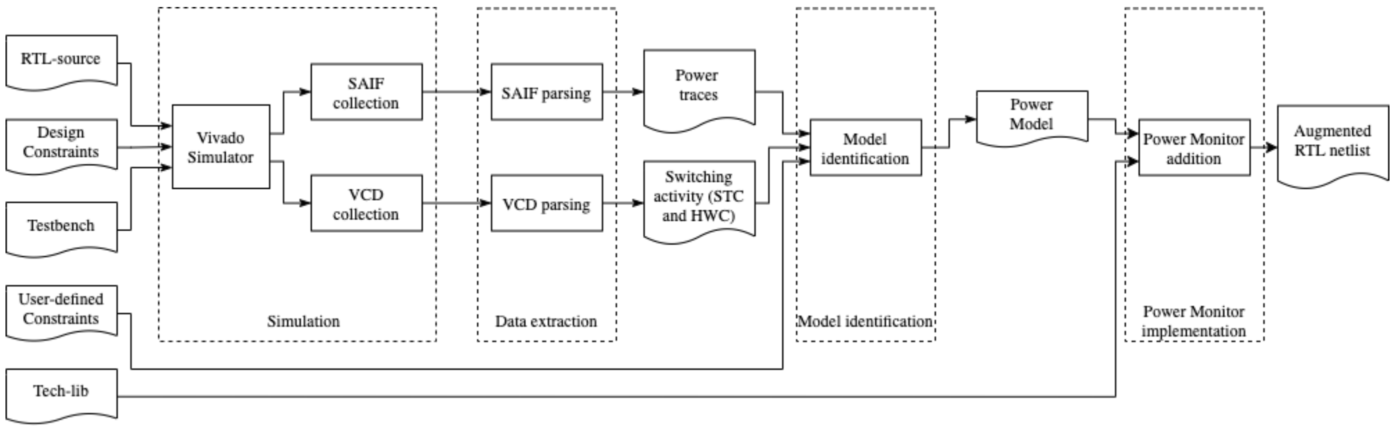


Figure 1: High level view of the flow.

2.2 Power and energy monitoring of HPC systems and supercomputing facility

High-performance computing systems racing towards exascale are facing significant challenges that are limiting their efficiency. One of the most significant challenges is power and energy consumption, which is being fueled by the end of Dennard's scaling and is starting to impact the peak performance and cost-effectiveness of supercomputers. Additionally, the reliability and security of both hardware and software components of computing systems present novel challenges to managing the system at large. This poses a daunting task for system administrators and users to optimize supercomputer and job performance, power consumption, identify anomalous behaviors, faulty situations, and ensure optimal system operation.

To address these challenges, data center automation aims to combine control theory, artificial intelligence, and big data technologies towards automating the data center management process. To pave the way towards data center automation, several steps must be undertaken. The first step is to implement a monitoring framework with a high level of detail and granularity capable of characterizing the target system. This system-level data collection infrastructure must be scalable, capable of handling a large amount of information (big-data oriented), and suitable for connection with information extraction level. With the collected data, it is possible to create a virtual model that behaves similarly to its physical counterpart and can be used for automated processes and predictive maintenance. By infusing reasoning capability via machine learning techniques in AI, it is possible to automatically detect faults or anomalous conditions disrupting the normal behavior of the supercomputer. Moreover, AI approaches can also be used to improve the general system management, such as improving the scheduling and resource allocation policies based on the predicted evolution of the system. The monitoring infrastructure and added AI can be hosted on the supercomputer itself, creating a self-monitoring and -adapting system. The second step is to obtain a job-level monitoring framework and run-time suitable for intercepting the application characteristics and leveraging them to reduce energy consumption.

What we aim is to realize a digital twin of HPC system and the facility in order to digital represent a system and its performance, facilitating the depiction of the system's structure and dynamics, forecasting its progression, and enhancing its functioning, and administration. Crucial to achieving this objective is the close interconnection between the foundational monitoring infrastructure and data management middleware, as well as the advanced AI-powered data analysis instruments. As far as we know, there are only a limited number of research studies that employ this vertical method in data center and supercomputing facilities.

Over the past few years, we have developed Examon, a monitoring framework designed to achieve our objectives. As represented in Figure 2, this solution is comprehensive, adaptable, and capable of handling GBs of telemetry data per day from the entire datacentre. Additionally, it seamlessly integrates with machine learning and artificial intelligence techniques and tools. To create a complete digital twin of the datacentre, we need to link the monitoring information obtained from both the facility and HPC systems. By doing so, we can interpolate and present the data to system administrators and users, providing a comprehensive digital representation of the datacentre and its operations.

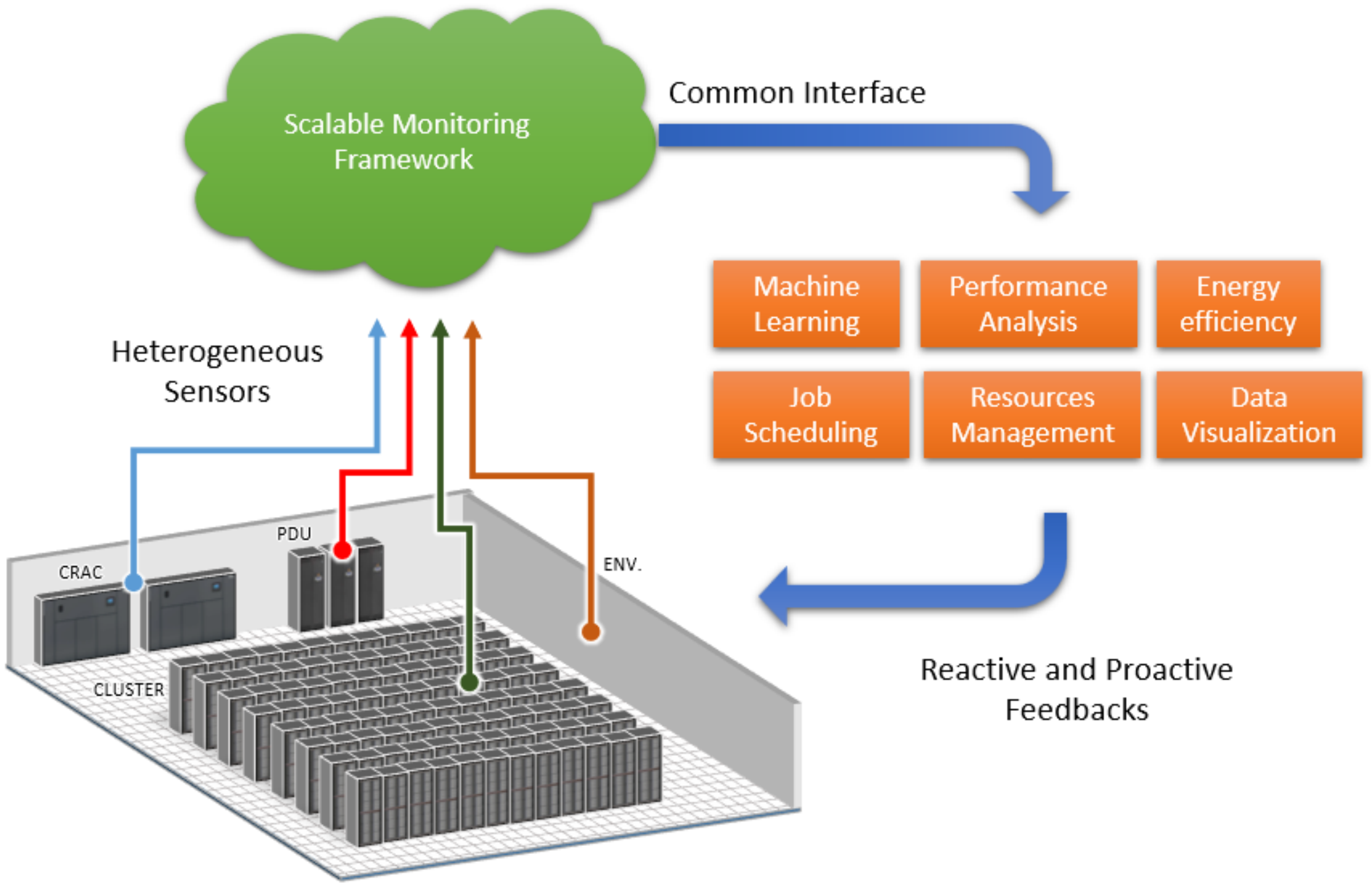


Figure 2: General scheme of a Examon framework

Our approach involves extending Examon to gather all sensors data from both the facility and supercomputers. In addition, we will integrate user-level tools, such as COUNTDOWN library (Section 4.3), with Examon to monitor application performance and energy efficiency. By linking this information with data from the facility, we can inform users about the environmental impact of their supercomputer usage. Furthermore, this approach enables us to identify and address facility and application inefficiencies, ultimately improving overall performance and reducing energy waste. In particular, Examon will be ported on Leonardo supercomputer, the 4th fastest supercomputer in the world and recently installed in the CINECA's datacentre at the Big Data Technopole in Bologna, Italy.

2.3 Energy consumption in memory management

In general computing systems, *memory* is considered to the second-largest power consumer after the processors, responsible for up to 40% of total system's power consumption.

Researchers proposed different management techniques to optimize the power consumption of this component. In this document we will focus our attention on the optimization at the level of the *paging* algorithms in a system with two levels of memory (e.g., main memory–disks).

Green paging is a fundamental variant of the classic paging problem in which we allow memory capacity to vary over time under the control of the paging algorithm, between a maximum of k and a minimum of k/p

pages. Accessing a page in memory takes one unit of time, while a page fault takes $s \gg 1$ units. The goal is to minimize, rather than the total time (equivalently, number of faults) taken to service a sequence of page requests, the integral of memory capacity over that time—a quantity we call *memory impact*. The main basis for this model lies in the increasing importance of energy consumption for both mobile and supercomputing platforms: modern hardware can dynamically turn off portions of the memory, both at the main memory and processor cache layers, so that instantaneous power consumption is proportional to the amount of active memory—and total energy consumption is proportional to its integral over time. It is crucial to observe that minimizing power by minimizing active memory does not necessarily minimize *energy*, i.e. the integral of power over time, since less memory may yield disproportionately longer executions. Also, note that below a certain capacity, other costs may become dominant; hence our choice of a minimum capacity below which no substantial savings can be realized.

The first to address a similar problem was Chrobak [5], allowing the paging algorithm to determine both the capacity and the contents of the memory on any given request, with the goal of minimizing a linear combination of the total number of faults and the average capacity over all requests. This problem has been investigated by López-Ortiz and Salinger [6] and later, in the more general version where pages have sizes and weights, by Gupta et al. [7]. It turns out [8], [9] that one can effectively decouple page replacement from memory allocation: even if the latter is chosen adversarially, a number of well-known paging algorithms like LRU or FIFO sport $O(1)$ competitive ratios (i.e., they perform within a constant factor of the optimal offline algorithm) with $O(1)$ resource augmentation (as in classic paging). Thus, green paging is essentially a problem of memory allocation: once memory is allocated, one can simply use LRU for page replacement, as it will incur a cost within a constant factor of what is achievable with (half) that memory capacity.

Agrawal et al. [10], [11] showed that the optimal competitive ratio for deterministic online green paging is $O(\log p)$. However, many questions remain open. The following are perhaps the two most relevant, for both theory and practice.

- Can we break the logarithmic barrier by using randomization? We want to design randomized green paging algorithms with better guarantees than those achievable by deterministic algorithms [11], or provide a proof that this is not possible (i.e., that randomization does not help in green paging).
- Can we break the logarithmic barrier by leveraging (machine-learning) predictions about future page requests? We want to design green paging algorithms that are able to take advantage of page request patterns of the current workload to achieve better performance. We aim for algorithms that have near-optimal performance when these predictions are accurate, but recover the prediction-less worst-case behavior when the predictions have large errors. Besides theoretical results, we also aim at experimental results on real-world workloads.

2.4 A proof of concept of two-phase cooling on a realistic experimental setup

Cooling electronic components via a closed-loop liquid circuit applied directly to, or near the surface of the chip, is not a new technology. This approach was used in the past and is now almost exclusively used on mainframes or HPC systems commonly found in supercomputer facilities. In recent years, economical versions of water-based direct cooling have been developed and sold to PC customers who wish to maximize computer performance. Nowadays, an improved version of the technology enabling secondary modifications is available on the commercial server market.

2.4.1 Direct cooling

Direct cooling provides a more efficient method of transferring heat from hot electronic components to the building's chilled water loop and then to the outside with little additional energy, compared to first transferring heat to the air and then to the building's chilled water system. In a direct cooling system, the temperature of the water returning after cooling IT equipment is much higher than typically found in data centers. This provides more opportunities for heat reuse or the ability to reject heat to the atmosphere via a dry cooler, thereby eliminating the need for a cooling tower or chiller plant in most climates.

Usually, direct cooling systems require an external source of energy to move the fluid across the loop. In Fig. 3, we show the general scheme of direct cooling with the main components of the loop highlighted.

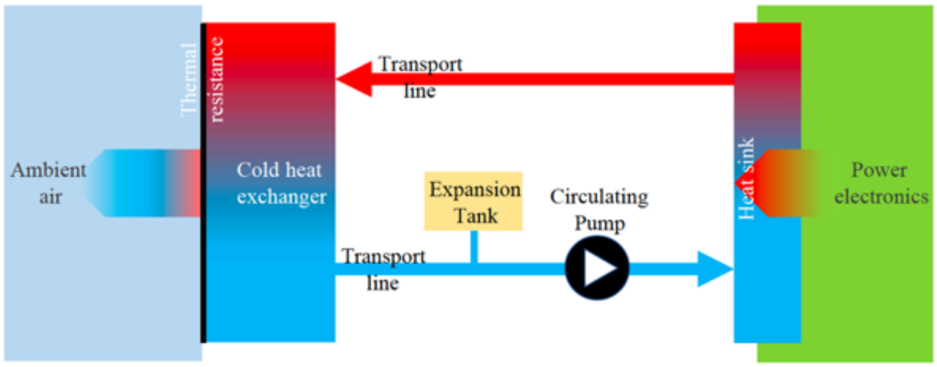


Figure 3: General scheme of a direct liquid cooling system

The circulating pump, or compressor in the case of vapor compression technologies, must be designed to counterbalance all pressure drops in the loop and maintain the necessary mass flow rate to evacuate the thermal load from the heatsink, where the thermal load is transferred to the coolant. Inside the cold heat exchanger, heat is definitively dissipated to the external environment. An expansion tank helps prevent fluid expansion, potential water hammers, and other instabilities that could cause mechanical damage to the components of the loop.

2.4.2 Single- vs Two-phase Direct Cooling

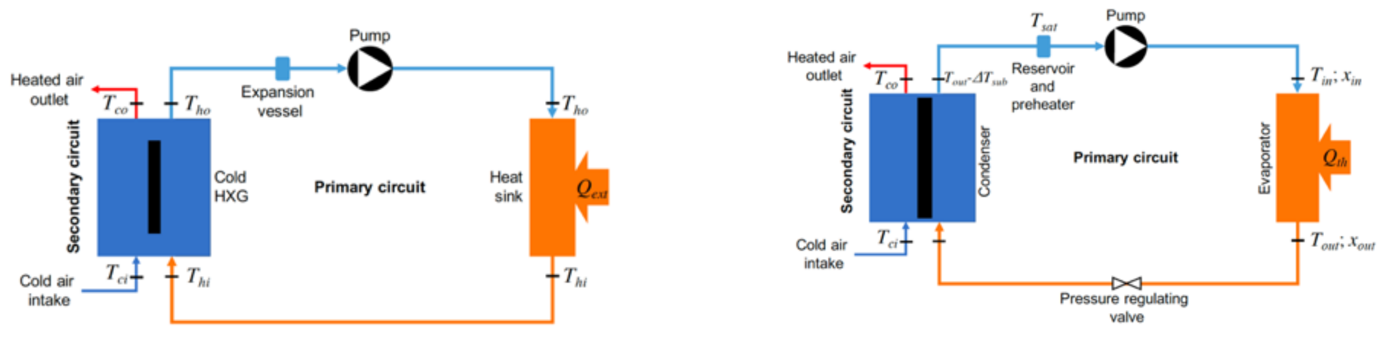
Active direct cooling can be single-phase or two-phase. Although the heat transfer mechanisms differ significantly between the two cases, the constituent components play the same role. In a two-phase system, both latent and sensible heat are used: the heatsink works like an evaporator, while the cold source acts as a condenser (with sub-cooling). In the diagram presented in Fig. 3, and for standard two-phase pumped circuits in steady-state conditions, the sub-cooled fluid flowing from the cold heat exchanger/condenser flows to the heatsink. The hot fluid or vapor then reaches the cold heat exchanger or condenser, where the heat is definitively expelled to the external environment. The sub-cooled liquid returns to the evaporator through the return transport/liquid line, where the loop starts again. An expansion tank is also used to impose and regulate saturation conditions, but this point will be detailed below. In other cases, the pump is replaced by a compressor, the tank is removed, and an expansion valve is placed on the liquid line, resulting in a Rankine vapor compression cycle. In situations where direct contact technology is utilized, such as jet-impingement/spray-cooling, the diagram shown in Fig. 3 remains applicable, but a collection tank must be installed to ensure the fluid is in a liquid state before returning to the evaporator.

Single-phase cooling systems, also known as *liquid cooling*, are perhaps the simplest and most commonly used configurations. The physical principle of this technology is relatively simple and, in its classical configuration, it is the most well-known technology. The working fluid is heated in the heatsink and flows through transport lines, which are ideally adiabatic, to the cold heat exchanger. A pump is used to move the fluid into the loop, and it must be designed to ensure an adequate mass flow rate and the necessary pressure head to counterbalance all the pressure drops in the circuit. Some accessories, such as expansion vessels and valves, must be provided to ensure smooth functioning. In Fig. 4a, a schematic diagram is shown.

Active two-phase cooling systems can be divided into two categories based on their physical working principles and thermodynamic features. *Pumped two-phase technology* operates with the evaporator at a higher temperature than the condenser, and a pump is installed on the liquid line. On the other hand, *vapor compression technology*, which uses the Rankine cycle, allows the evaporator to operate at a lower temperature than the condenser, and a compressor is used on the vapor line. While the conceptual design of these cooling systems is similar to that of single-phase cooling systems, their operating principle is entirely different. Latent heat is used to extract heat from the hot source. Fig. 4b shows the conceptual design of a pumped two-phase cooling system, where a free surface tank serves as a reservoir, and saturation conditions are enforced. The subcooled liquid from the condenser enters the tank.

In recent years, Direct Liquid Cooling (DLC) has been subject to several tests. A comprehensive demonstration of the test setup was conducted at ORNL [12] to determine the thermal efficiency of DLC and its potential impact on the overall energy usage of data centers. Fig. 5 displays the location of the system components and their interactions in that specific testbed layout. The integrated pump and cold-plate assemblies absorb heat from the CPU, while the memory DIMMs are cooled by transferring heat to a manifold (in contact with a heat transfer tape) carrying cooling water. The cooling water supply and return paths are provided by a set of flexible tubes for each server. The heat collected is transferred to the facility cooling support using a Cooling Distribution Unit (CDU) using the tube set. While the direct cooling system does not provide cooling for all the server components, some electronic components still need to be air-cooled using fans within the server. However, the airflow requirement is reduced, resulting in a reduction in the number and speed of fans and significant energy savings for the same processing load. This reduction in server fan energy accounts for most of the reduction in server energy use. The tests performed at ORNL used a two-step approach: (1) measure the percentage of power supplied to the servers that are captured as heat by the prototype cooling system, and (2) use these percentages to estimate and compare the energy consumption of a server equipped with a stock cooling configuration (air-cooled) to a prototype cooling system (direct-water-cooled) using models.At the end of the project we shall have a proof of concept of two-phase cool- ing on a real HPC system with quantitative estimates of the energy savings produced by the novel technology

The parameters changed during the tests were the supply water temperature, the supply water flow rate (which affects the return water temperature), and server power. The lowest supply water temperature tested for this demonstration was 15°C. While lower supply water temperatures may yield improved results, data



(a) single-phase (b) two-phase

Figure 4: scheme of single- vs. two-phase cooling

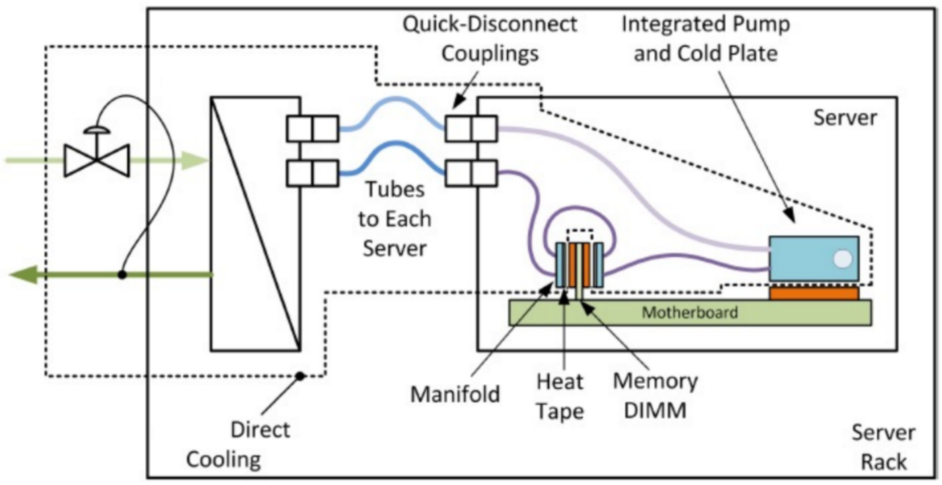


Figure 5: Schematic of Direct Liquid Cooling System

center operators may be concerned with managing condensation, which is a valid concern and should be investigated via installation of a direct cooling system. One key advantage of direct cooling is that adequate cooling is also achievable with much higher (e.g., 45°C) supply water temperatures. In such cases, the directly cooled components remain well below the critical temperature specified by the component or server manufacturer. This demonstration evaluated the thermal performance over a range of supply water temperatures to investigate the potential for energy savings in cooling infrastructure using alternative cooling methods, such as dry coolers or cooling towers. The temperature range used was 15°C–45°C. Additionally, three computing power levels were tested: idle (120 watts per node), middle (270 watts per node), and full (430 watts per node), with a water supply flow rate of 4.9 gallons per minute (gpm). In Fig. 6, we show the results obtained for the maximum facility-side flow rate (approximately 4.93 gpm) provided by the demonstration setup.

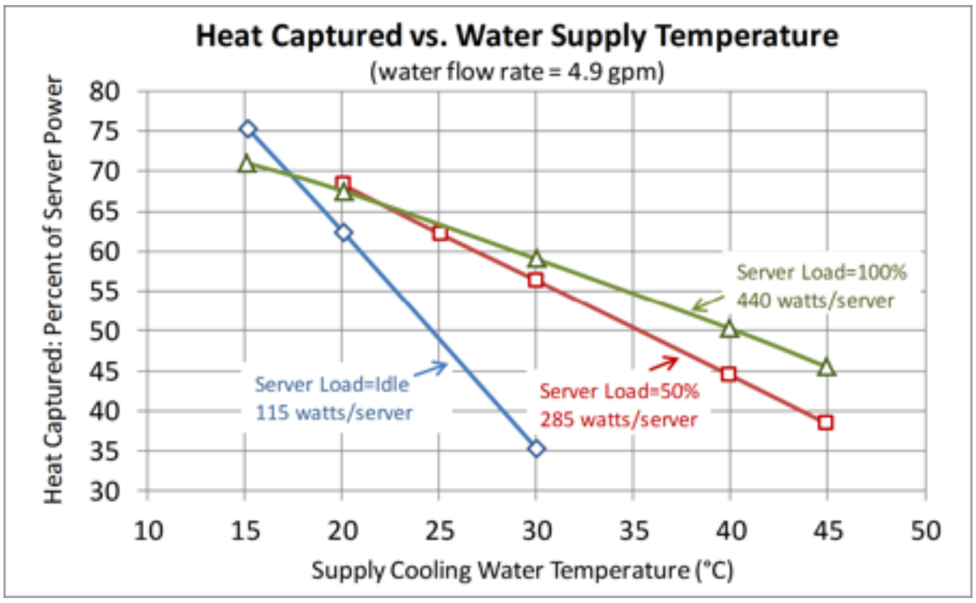


Figure 6: Maximum Heat Capture Vs. Water Supply Temperature For Three Sever Power Levels

Based on the test results, the following observations could be made:

- the proportion of captured heat increases as the supply water temperature decreases, for all power levels;

- at higher supply water temperatures, the proportion of captured heat becomes more distinct among the three power levels. However, at lower temperatures, the difference is less noticeable;

These findings suggest that the server power level should be considered when determining the optimal supply water temperature to minimize overall operating costs. For instance, if the server power level is low and the supply water temperature is high (e.g., 30°C), the amount of captured heat is lower compared to higher server power levels. While having the flexibility to choose from a wide range of supply water temperatures and capture a large fraction of the server power would be advantageous, the data indicate that such freedom may not be feasible.

2.4.3 Two-phase Cooling Proof of Concept

We now present our proposal for the two-phase cooling of a HPC system. Direct Two-Phase Cooling (DTPC) is a highly effective method for removing thermal loads, which can improve the overall efficiency of the cooling system. Previous research has mainly focused on the performance, heat transfer efficiency, and heat recovery requirements of two-phase pumped cooling systems. In particular, the cooling of chips and data servers, which have high heat flux densities and small heat transfer surfaces, has received significant attention. The two-phase technology has been proven to be effective in maintaining a constant chip temperature while saving pumping power and reducing coolant mass flow rate, as demonstrated in a study by Thome (2010). For military aircraft applications, two-phase technologies are designed to handle high thermal power loads of hundreds kilowatts, with peaks of thousands kilowatts (Homitz, 2010). The American company ACT is developing two-phase cooling systems for both civil and military applications, which are expected to reduce the overall system mass by reducing the mass flow rate and pumping power (ACT, 2019).

Although the basic concept of two-phase cooling systems is similar to that of single-phase systems, the operating principle is entirely different. In two-phase systems, latent heat is used to extract heat from the hot source. To demonstrate the potential energy savings of two-phase cooling technology compared to traditional Direct Liquid Cooling (DLC), we shall design and realize an experimental test setup to replace the conventional air cooling system used in some high-heat generating components (e.g., CPUs, GPUs, or FPGAs, either discrete or integrated) with cooling provided by a liquid in single-phase or vapor in two-phase transition. In our setup we shall use two computing nodes to compare the energy-saving performance of direct cooling technology with two heat removal solutions (single/two-phase). We shall make estimates of the energy savings potential by analyzing two scenarios: (1) a computing node retrofitted with single-phase DLC, and (2) a computing node retrofitted with two-phase DLC. Since measuring energy savings directly is generally difficult, we shall compare the energy use for the single-phase DLC and two-phase DLC. In our experimental test setup, we shall vary several parameters, including the supply water temperature, supply water flow rate, and the computing power of the node. We shall analyze the performance associated with higher return water temperatures to investigate the possibilities of heat reuse. We shall vary the computing power of the node to investigate the effects of different levels of computing node utilization. At the end of the project we shall have a proof of concept of two-phase cooling on a real HPC system with quantitative estimates of the energy savings produced by the novel technology

3 RELIABILITY EVALUATION AND MANAGEMENT PLATFORMS

This Chapter analyses and presents a set of prototype candidates for the reliability evaluation.

The reliability evaluation is focused on mainly three abstraction levels: *i)* Fine-grain technology level, *ii)* Coarse-grain architectural level and *iii)* System level. Fine-grain analyses and evaluations are focused on the implementation technology of the devices (e.g., FinFet transistor technology). In addition, Coarse-grain evaluation focused on the component, units and structures inside the devices and their reliability features. Finally, system level evaluations are focused on the software solutions and tools for the system's evaluation.

In every abstraction levels, we discuss and analyze several representative physical and computational models and prototypes, as well as framework environments to allow the development, evaluation, and validation of hardening techniques. Similarly, we analyze device-level monitoring mechanisms for the system level reliability evaluation. Finally, we discuss mechanisms and models to allow the system-level fault tolerance in real workload operation.

The following subsections introduce and highlights the main prototypes according to each abstraction level and evaluation target.

3.1 Reliability Evaluation

One of the crucial steps of the reliability assessment is the evaluation of the reliable state of a system. For this purpose, a set of techniques targets the evaluation, characterization, and identification of vulnerable hardware and software structures prone to be affected by failures and corrupt system operations. In fact, the reliability evaluation of a system aims to characterize the incidence and impact of faults, errors, and failures in their hardware and software components.

Figure 7 depicts the propagation of a physical defect as faults, errors, and failures in modern devices. As observed, different abstraction layers (i.e., physical, hardware, software, and system) interact in the operation of the system as well as with the propagation of faults. In addition, a group of faults and errors might be masked by the implicit structure of the hardware, their software development and deployment, or by a combination of both effects.

To analyze the reliability of a system and identify hardware structures and software vulnerabilities prone to propagate faults and errors, the reliability evaluation resorts to fault and error models that represent the impact of physical defects on hardware and effects at the software level. In general, fault models for reliability evaluation can be classified by their effects on the system. The most common fault models are briefly introduced as follows:

- *Permanent faults*: they refer to hardware malfunctions that persist indefinitely after the occurrence (e.g., short or open circuits). These faults represent physical defects in a system caused during the production phases of a system (e.g., electromigration causing an open or bridge effect) or its in-field operation (i.e., premature aging or wear-out impacting the operation of a transistor);

- *Delay faults*: they refer to hardware malfunctions that affect the timing execution of a given system's operation (e.g., a logic delay corrupting an operation during a clock period). These faults depend on the initial conditions of the system to arise and propagate their effects to the system or a running application. These faults represent similar physical defects in a system to those that cause permanent faults. These faults might represent the early stages of permanent faults.

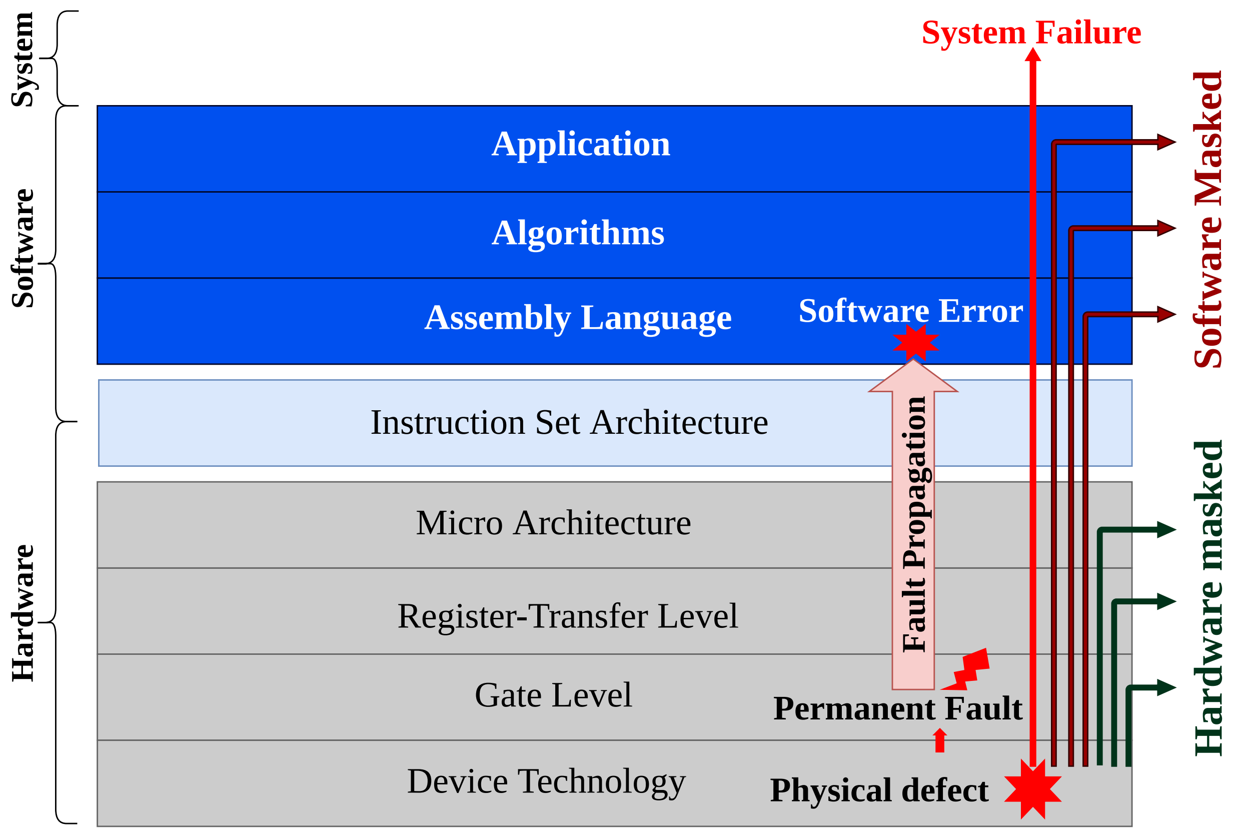


Figure 7: A general scheme of the propagation of physical defects across the different abstraction layers in modern devices. Adapted from [13].

- *Intermittent faults*: they occur sporadically and under specific initial conditions of the system. This temporary situation occurs sporadically in a system, and it is caused by defects in the hardware or bugs in the software or a combination of both. In fact, this fault category is hard to evaluate and predict when a system increases its complexity and size;
- *Transient faults*: they refer to mostly external events caused by external sources that impact the system's hardware and corrupt its operation (e.g., electromagnetic interference or radiation effects corrupting a memory cell). This fault category mainly represents the temporal effect of corrupting the data of a system's operation (i.e., bit-flip in memory), and it usually occurs during the in-field operation of the system.

Regarding permanent faults, *stuck-at* and *bridging* faults are two typical models currently used to assess the reliability of designs in safety-critical and mission-critical applications. Similarly, *Path-delay* and *Transition Path-Delay* fault evaluations are used to evaluate delay effects on the hardware infrastructure of modern devices [14]. The evaluation of transient fault effects (i.e., resorting to fault models, such as Single-Event Upset *'SEU'* and Single-Event Transient or *'SET'*), is of high interest in modern generations of complex and massive systems (e.g., HPC machines) since they occur during the in-field operations and used to affect the application's data then causing corruptions in the application [15]–[17].

All the previous fault models are commonly evaluated to assess the reliability of systems in the safety-critical and mission-critical domains (e.g., automotive) [18] since reliability is a significant concern. Furthermore, equivalent analyses are also applied to the hardware infrastructure of complex systems, such as current and future generations of HPC machines [19]–[21]. It is worth noting that modern transistor technologies and emerging implementation approaches might demand the development of more efficient and accurate fault models for the reliability analysis, see subsection 3.1.1.

Unfortunately, fault models (especially the fine-grain ones) are hard to apply to large-size designs (e.g., hardware accelerators and multi-cluster processors, typical of the HPC domain). Thus, higher-level abstractions, such as error models, are commonly used to represent the main features of faults at higher levels (e.g., instruction, software, application, or system levels). These error models provide the mean characteristics of the fault effects to allow the acceptable reliability evaluation of a system considering feasible evaluation times[13], [22], [23].

In both cases (fine-grain and high-level), an adequate reliability evaluation requires the definition of primary targets of analyses that imply the selection of a given abstraction level of evaluation, as well as the fault/error models that also involve the use of appropriate prototypes, artifacts, frameworks, tools, and models, i.e., the reliability evaluation of a complete application might involve high-level error models on architectural models, or real system prototypes, instead of fully low-level micro-architectural (e.g., RT-level) prototypes and fine-grain fault models due to the costly computational power and unfeasible simulation times for a complete application. However, hybrid, cross-layer, and co-simulation approaches can be exploited to provide accuracy with feasible simulation times.

The following subsections describe the possible candidate prototypes and their required frameworks to evaluate reliability at different abstraction levels. The first subsection describes and introduces the main prototypes and frameworks for the fine-grain technology-level reliability evaluation and the importance of their fault modeling. Then, the second subsection analyses and describes the possible prototype candidates and feasible methods for the reliability evaluation at the coarse-grain (high) level of abstraction.

3.1.1 FinFET Aging and Fault Modeling

The proposed research activity will mainly address the gap in the state of the art consisting in the lack of accurate analysis and modelling approaches to evaluate the effects of latent faults and aging phenomena affecting simultaneously FinFETs transistors of data-paths of modern SoCs (e.g., based on the RISC-V architecture) during their operation in the field. Based on the results achieved by the performed analysis, possible low-cost monitors to detect the presence of latent faults during SoC operation in the field might be then derived.

In order to achieve the goal of this activity, we plan to perform a preliminary research phase to evaluate, at the electrical level, the effects of likely FinFETs faults occurring individually (i.e., not combined with aging phenomena). The goal of this preliminary evaluation is to identify the subset of FinFET faults that may be not detected during manufacturing testing, thus becoming “latent” faults that could combine with aging phenomena during the SoC operation in the field.

We plan to perform such a preliminary analysis by means of **HSPICE**, considering elementary gates (*NOT*, *NAND* and *NOR*) implemented by a **7nm FinFET technology** available in [24], and considering two possible FinFET implementations (with *six* and with *eight* fins). Based on the obtained results, we may also consider FinFETs with a different number of fins (generally indicated by n).

We plan to emulate the most likely FinFET faults, that is stuck-open and stuck-on faults affecting one or multiple (up to all) fins of FinFETs, as described in [25]. In particular, FinFETs affected by stuck-opens on k fins (with $k \leq n$) will be simulated as if they were FinFETs with (n-k) fins [25].

Instead, as shown in [25], FinFETs affected by stuck-ons on k fins (with $k \leq n$) behave as if a wire connecting the source and drain of the fins were present, regardless of the gate voltage. Therefore, this kind of faults make the affected fin(s) always conductive. Similarly to [25], we plan to emulate FinFETs affected by stuck-ons on k (out of n) fins as FinFETs with the gate terminals of the affected k fins connected to Vdd (for n-type FinFETs), or ground (for p-type FinFETs).

In a following phase of this research activity, we plan to evaluate, at the electrical level, the combined effects of *"latent"* faults identified by the analyses described above and aging affecting FinFETs transistors. Initially, we plan to analyze the combined effects on the propagation delay of the elementary gates (NOT, NAND and NOR) considered in the previous part of our analysis. Then, we plan to consider also more complex and realistic case study circuits, that can be representative examples of data-paths present within modern SoCs. For instance, one or more benchmark circuits belonging to the **ISCAS 85 benchmark set** [26] could be considered as case study in this phase of research.

Similarly to the analyses described above, we plan to perform this analysis by means of **HSPICE**, considering FinFETs implemented by a 7nm technology [24], with six and eight fins. For fault simulations we will consider the most likely FinFET faults (i.e., stuck-opens and stuck-ons) affecting one or multiple fins of FinFETs, emulated as as described in [25].

We will consider Bias Temperature Instability (BTI) as aging mechanism, since it is recognized as one of the main aging mechanisms for scaled FinFET technologies [27]. In particular, we plan to evaluate the impact of BTI on the conductance of FinFET transistors by performing electrical level simulations with the **MOSRA tool** from **HSPICE** [28], considering a range of operating time between 0 and 10 years, with the aging model's technological parameters in [27], [29].

We expect that the results of the performed analyses will enable to understand the interaction of faults and aging phenomena simultaneously affecting FinFETs, and their combined effects on the propagation delay of the considered case study circuits. This will enable to derive a mathematical model for such a combined effect on the propagation delay of data-paths of modern SoCs.

Finally, based on the achieved results, we expect to derive, in a following phase of research, low-cost innovative approaches to detect, during the SoC in-field operation, the presence of FinFET *"latent"* faults. Such detection approaches will enable the activation of possible recovery mechanisms, to avoid that FinFET *"latent"* faults can then combine with aging mechanisms, thus possible compromising the system's functional safety.

3.1.2 High-level (Coarse-grain) Reliability Evaluation

At medium and high-level abstractions (i.e., gate-, RT, architectural, and system levels), ideal prototypes and artifacts for the reliability evaluation must include the main structural features of the hardware architecture to support the characterization and also allow the identification of vulnerable structures that can later be targeted for mitigation or hardening purposes.

Table 1 describes the possible candidates of prototypes according to the abstraction level, type of unit, and primary features for the evaluation. We identified 12 different prototypes and models that can be used to evaluate reliability main focused on two main features: *i)* scope and structural main targets of the evaluation, and *ii)* the relevance of the prototypes in the HPC domain. In the first case, we select prototypes that allow the reliability evaluation at different levels, from individual co-processor units that can be used as in-chip modules in processors or hardware accelerators up to complete processors and hardware accelerators organized as systems for the evaluation of complete applications. Then, the second feature targets the selection of these representative prototypes, models, and artifacts that can be used to evaluate the reliability and are relevant for the HPC domain, such as processors in cluster organizations and hardware accelerators.

The candidate prototypes provide specific and individual features for the evaluation. In case of coarse-grain prototypes (those at system level), such as microarchitectural models of RISC-V processors (**IBEX**[1], **Hero**[2],

[1]`https://github.com/lowRISC/ibex`
[2]`https://pulp-platform.org/hero.html`

Table 1: Selected candidate's prototypes for the reliability evaluation.

Abstraction level	**Type**	**Prototype/model name**	**Implementation language**	**Main features for reliability evaluation**
Architectural	Host single-core processor	RISC-V SPIKE	C/C++	Structural organization of host processors
RT-level	Host single-core processor	RISC-V IBEX	Verilog	Micro-structural organization of host processors
RT-level	Host single-core processor	RISC-V RI5CY	Verilog	Micro-structural organization of host processors
RT-level / Architectural / System	Host cluster processor	RISC-V Hero	SystemVerilog / Verilog	Structural and micro-structural organization of host processors
Architectural	Hardware accelerator	GPGPU-SIM	C/C++	Structural organization of GPU accelerators
RT-level	Hardware accelerator	FlexGripPlus	VHDL/Verilog	Micro-structural organization of GPU accelerators
RT-level	Hardware coprocessor	Fused SFU	VHDL/verilog	Micro-structural organization of special purpose accelerators
RT-Level	Hardware coprocessor	Modular SFU	VHDL/verilog	Micro-structural organization of special purpose accelerators
RT-level	Hardware accelerator	Open TCU	VHDL/verilog	Micro-structural organization of GPU/AI accelerators
RT-level / Architectural / System	Hardware accelerator	NVDLA	Verilog/SystemC	Structural organization of AI accelerators

or **RI5CY**[3]), hardware accelerators (such as **FlexGripPlus** [4] or **NVDLA** [5]) provide the complete system description and the main representive architectural structures to allow the evaluation of several reliability features, including fault effects during the in-field operation and their impacts on representative applications. In addition, analysis at unit level (e.g., resorting to co-processor and individual units), such as on Special Function Units(**Fused**[6] and **Modular**[7]), which are representative arithmetic units involved in several applications and are present as in-chip accelerators in processors and hardware accelerator, including GPUs, are representative study case for the analyses and evaluation on specific workloads or specific instructions. Similarly, other hardware accelerators, such as **TCUs**[8][9] represent opportunities to analyze and evaluate the reliability at unit levels for later integration into cross-layer environments for complete system reliability evaluations, e.g., in RISC-V-based environments. Similarly, unit accelerators, such as **TCUs** and **SFUs**, allow the design exploration, organization, and integration of special-purpose cores for RISC-V processors, as well as the evaluation of their principal reliability advantages and vulnerabilities.

3.1.3 Methods and frameworks for reliability evaluation

Usually, the industry and the research communities evaluate the reliability of digital devices by resorting to Fault Injection (FI) techniques, which can be classified into four primary groups: (1) ***Hardware-based* FI**, that induces physical distortion on a real device by exposure to external effects (i.e., by modifying parameters, such as voltage, or temperature, or by direct interaction with radiation). Hardware evaluation might resort to beam experiments, allowing the effective evaluation of transient fault effects on parallel applications [30], [31]. Unfortunately, these analyses can hardly identify with enough accuracy the individual vulnerable structures, or correlate application's errors with design structures in a processor or hardware accelerator. (2) ***Emulation-based* FI** resorts to hardware prototyping platforms (e.g., FPGAs) to implement and deploy a design for fault effects evaluation. However, this approach demands custom fault injector frameworks[32]

[3]`https://github.com/embecosm/ri5cy`
[4]`https://github.com/Jerc007/Open-GPGPU-FlexGrip-`
[5]`http://nvdla.org/`
[6]`https://opencores.org/projects/special_functions_unit`
[7]`https://opencores.org/projects/special_function_unit_ppa`
[8]`https://opencores.org/projects/open_tcu`
[9]`https://github.com/TheColombianTeam/tensor_core`

and design models, which are not commonly available for commercial products [33], [34]. (3) ***Software-implemented*** **FI** suitably modifies the application's code to represent hardware faults as instruction errors (i.e., source code mutations representing faults) and allows the reliability characterization of applications. This approach allows the fault propagation at device speed but is mainly restricted to corruptions on user's visible resources (i.e., register files or memory locations). Thus, their analyses are limited to a few structures in processor-based designs [22], [35], [36]. (4) ***Simulation-based*** **FI** is the most versatile and accurate approach for reliability evaluation by taking advantage of the extended information and details of available fine/coarse-grain abstractions of a design (i.e., from gate/RT-level implementations up to architectural and functional *'high-level'* specifications)[37]. A low-level micro-architectural (fine-grain) FI of a design provides an accurate evaluation but demands a considerable amount of computing power and extensive simulation times often prevent their massive use for complex designs, such as multi-core cluster platform or parallel hardware accelerators, running realistic workloads [13]. In contrast, resilience evaluation on functional and architectural simulators (i.e., coarse-grain) is flexible and can allow the evaluation of hardening strategies (on the structure of a design or at software levels) and the execution of complete workloads in reasonable execution times and acceptable accuracy.

Other approaches target the evaluation resorting to combinations of two or more FI levels (i.e., using cross-layer resilience evaluation), which is feasible for complex and large designs, such as those typically in the HPC domain. In fact, several HPC applications, such as CNN training, and CNN inference can be benefited by resorting to clever cross-layer mechanisms, considering the efficient and accurate error propagation among the different operative layer of the underlying hardware and the running software, see Figure 7. Reliability evaluations for such complex workloads might demand custom frameworks and the definition of accurate metrics. Once a FI approach is selected, the evaluation of resilience in a device comprises several simulation campaigns to independently evaluate the effect of faults, errors or failures in the system under evaluation and allow the identification of the possible vulnerable hardware structures, or characterize the behavior of applications and structures. In general, one fault or error is placed and evaluated per simulation. Then, similar procedures are repeated for other fault targets.

Depending on the complexity, evaluating fault and error models might resort to focused fault injection campaigns. Unfortunately, as already mentioned, the increasing complexity of modern systems (i.e., SoCs with multi-core and multi-cluster processors and parallel accelerators) requires a clever combination of different techniques to provide acceptable reliability evaluation in affordable execution times. Thus, currently, we plan to use smart approaches of multi-level, cross-layer, and co-simulation schemes of logic simulation to evaluate the reliability. Additionally, alternative activities might be focused in the exploration, adoption, and use of HPC machines for speed up the analysis and evaluation of reliability features of medium and fine-grain digital designs, such as processors and GPU cores for HPCs. In fact, the distributed execution paradigm of HPCs can benefit the complex, data intensive, and computationally demanding operation of fault simulation campaigns, which are commonly used in the reliability evaluation of early and medium stage of fine-grain and architectural digital designs. Thus, improvements in terms of performance, throughput, and efficiency are desirable in the evaluation of large designs. Similarly, evaluation environments and framework for design exploration and co-simulation can be adapted to exploit distributed HPC paradigms.

3.2 Hardening

3.2.1 Memory Hardening

The proposed research activities will aim at developing memory elements (e.g., Flip-Flops, or FFs, latches and memory cells) presenting a high robustness against Transient Faults (TFs) and low costs in terms of area, power consumption and delay. The availability of such low-cost robust memory elements would enable to improve the robustness of data-paths and memory blocks of modern SoCs (e.g. based on the RISC-V architecture) with respect to soft errors (SEs), at affordable extra costs in terms of area and power consumption.

In order to achieve this goal, we plan to start developing low-cost robust memory elements based on our previous approaches in [38], [39]. Such approaches are based on the idea to duplicate (or triplicate) the latch internal node(s), and to adopt an output C-element [39] to make the output of the memory element change its logic value accordingly to the value assumed by the majority of internal nodes.

We expect that, based on the previous approaches in [38], [39], we will develop new memory elements that, compared to existing alternative solutions, will feature a similar robustness against TFs affecting single internal nodes (i.e., TFs affecting single nodes will not result in the generation of SEs at the outputs of the memory elements), but will require lower costs in terms of propagation delay, power consumption and area occupation. Then, we plan to develop also more robust new memory elements, that will not give rise to the generation of SEs at their outputs, even in the presence of TFs affecting two and/or three internal nodes simultaneously. We expect that the new, more robust, memory elements will present only a limited cost increase with respect to the previously developed robust memory elements (that are able to tolerate only TFs on single internal nodes).

We will verify the operation, the robustness and the costs of all proposed solutions by means of **HSPICE** electrical level simulations, considering a **16nm CMOS technology**, with a power supply voltage Vdd=0.8V. The correct behavior of the proposed solutions will be also verified by means *Monte Carlo* electrical level simulations, considering statistical variations of power supply voltage and of relevant technological parameters (e.g., transistor threshold voltage, oxide thickness, and electron/hole mobility).

The robustness of the proposed solutions against TFs will be assessed by evaluating their Soft Error Rate (SER), that is the typical metric used to assess the robustness of memory elements [38], [40]. In order to estimate the SER of the proposed solutions, we will need to determine, for each one of their internal nodes: *i)* the time interval within a clock period (i.e., window-of-vulnerability) during which a TF hitting the node can propagate till the output of the memory element and give rise to a SE; *ii)* the amount of charge that has to be collected by the hit node to produce a voltage glitch whose amplitude exceeds the logic threshold of the fan-out gate (i.e., the critical charge). We will derive the amount of charge deposited on circuit nodes affected by TFs by emulating the current glitch generated by the hitting particle, as described in [38].

In addition, we also plan to evaluate the impact of bias temperature instability (BTI) degradation on the robustness of the developed memory elements over time. In fact, as shown in [40], [41], BTI may significantly increase the SER of memory elements during circuit lifetime. For this evaluation, we plan to emulate the impact of BTI on the conductance of transistors by performing electrical level simulations with the **MOSRA tool** from **HSPICE** [28], considering a range of operating times between 0 and 10 years.

Based on the obtained results , we may need to develop strategies to reduce the impact of BTI on the SER of the developed robust memory elements. Initial strategies could be based on the solutions that we have presented in [41]. For instance, we plan to modify the internal structure of the memory elements to reduce the stress time (i.e., the degradation) of transistors driving the most susceptible nodes of the memory elements. Such an approach will enable to reduce the BTI due degradation of such transistors during circuit lifetime, and consequently the impact of BTI on the total SER of the proposed memory elements.

3.2.2 Software-based Hardening

An alternative method to protect and avoid anomalies in the application's execution due to faults in the hardware is based on software hardening approaches that are based on adapting the application's code with hardening mechanisms (e.g., redundancy or design diversity) with the goal of reducing or tolerating the effects of hardware faults [42]. This approach consist of modifying the code of an application to include one or more fault-tolerance mechanisms to allow the detection and possible correction of corrupted operations due to the incidence of hardware faults or software errors.

Most software hardening solutions are based on some level of redundancy. Indeed, software hardening through redundancy can be deployed in two forms: *i)* as timing redundancy and *ii)* as spatial redundancy. The former approach considers the replication and execution of two or more times of the same software code at different times, while the latter considers using redundant hardware to process the same program's code. In both cases (timing and spatial), the main objective is to reduce the presence of anomaly effects during the program's operation by resorting to several key software attributes, such as modularity, system closure, atomicity of actions, decision verification, and exception handling) [43].

Timing redundancy aims to execute the complete program or the most vulnerable portions several times, mostly on the same device and hardware structures. In contrast, spatial redundancy targets the execution of the same program's code in different available devices or hardware structures, so limiting the effect of hardware faults affecting some system structures. This second approach is feasible in systems with some degrees of parallelism through scheduling management [44]. Optimized versions of both approaches are based on selective hardening, so limiting the blocks of software's code requiring redundancy. In these cases, external analyses support the identification of the most sensitive and vulnerable code blocks.

Another software hardening approach is based on software design diversity. In this case, the diversity relies on the *"independent"* generation of *"different"* application's (or portions of) code implementations. In fact, design diversity has been effectively used to protect systems from failures[45]. In this case, the code (or portions) are carefully written to avoid the use of the same devices or hardware components, so preventing possible anomalies by errors in the system structures (e.g., replacing code blocks using arithmetic operations and arithmetic cores by algorithms that avoid the cores but achieve equivalent results). Thus, in some cases, alternative algorithms or mechanisms are required to adapt the application's code. Unfortunately, these solutions can be hardly applied to centralized systems resorting to the same operative system, compilers, and available hardware.

Other software-based hardening alternatives rely on analyzing the application and its internal algorithms to develop algorithm-specific methods (e.g., Algorithm-Based Fault Tolerance, or ABFT [46]). In these cases, the algorithm is adapted to add error detection and mitigation features by some levels of redundancy or by modifying the structure of the algorithms (e.g., check-sums and redundant operations). Interestingly, several solutions have been applied into highly parallel systems and applications, such as FFT and Matrix multiplications by their low overheads and costs for adapting and implementing [46] [47] [48] [49] [50].

To develop and evaluate software-based hardening solutions, a combination of programming environments and the description of the target processor-based platform are required. The programming environment implies the access to compilation frameworks to develop custom versions to automatically handle the software hardening. Similarly, the description of the target device (e.g., structural or micro-architectural) is needed for two main purposes: *1)* evaluation and validation of the proposed solutions and *2)* development and implementation of extended features of software-based hardening (e.g., hybrid mechanisms in the device). In the first case, the description of the target device is directly used to evaluate (through fault simulation experiments) the effectiveness of the proposed software-based solution. In the second case, the description can be used to extend the software hardening features at hardware level(e.g., software instructions activating hardware redundancy units in a processor). Interestingly, software-based hardening solutions for processors and hardware accelerators (e.g., GPUs and TPUs) can be developed by resorting to structural and micro-architectural prototypes or a clever combination of both (i.e., by means of cross-layer evaluations). Table 1 already lists the main candidate prototypes for the development and analysis of software-based hardening. Unfortunately, the associated frameworks (e.g., compilers and programming environments of the processor-based system) require customization to automate the proposed solutions and directly depend on a target device or systems (e.g., RISC-V-based processor core or hardware accelerator).

3.3 Device-level on-line reliability monitoring and fault management

3.3.1 Memory and Power Management Monitoring

A goal of the proposed research activity is to fill the gap in the state of the art due to the lack of efficient solutions to prevent the catastrophic consequences of permanent faults affecting ECC's encoding/decoding blocks of modern SoCs (e.g., based on the RISC-V architecture).

In order to achieve this goal, we initially plan to analyze, at the electrical level, the effects of the most likely permanent faults possibly affecting ECCs' encoding/decoding blocks during their operation in the field.

For our analyses, we plan to consider, as representative case studies, the encoding/decoding blocks of two widely adopted ECCs for caches: *1)* the Single Error Correcting – Double Error Detecting **(SEC-DED) Hsiao ECC** [51]; *2)* the Double Error Correcting **(DEC) Orthogonal Latin Square (OLS) ECC** [52]–[54].

We plan to consider a realistic case of words composed by 16 information bits (d0...d15) and 6 (16) check bits for the case of the **SEC-DED Hsiao (DEC OLS)** ECC. We will implement the encoding/decoding blocks of both ECCs at the electrical level, considering a **16nm standard CMOS technology**, with power supply voltage Vdd=0.8V, and encoding/decoding blocks' typical implementations present in the literature. In particular, for the **SEC-DED Hsiao ECC**, we plan to consider the implementation in [51], [55], while for the **DEC OLS ECC**, we plan to consider the implementation in [54].

Then, by means of **HSPICE**, we will perform fault simulation at the electrical level, in order to evaluate the effects of the most likely faults affecting the encoding/decoding blocks of the two considered ECCs. In particular, we will evaluate the effects of all possible resistive bridging (BFs) with values of connecting resistance (RB) in the interval $[0\Omega, 100k\Omega]$, and stuck-at faults (SAs). We will consider the realistic assumption of faults occurring one at a time. In this phase of the research, we also plan to introduce proper metrics to evaluate the risks of the considered faults' effects on functional safety, thus enabling to identify the most critical faults.

Based on the achieved fault simulations' results and on the developed metrics, in a following research phase we plan to develop low-cost approaches to detect, during the SoC in-field operation, the occurrence of those faults that can compromise system's functional safety. The proposed detection approaches will enable the activation of possible recovery mechanisms to re-establish the SoC correct operation.

Another goal of the research activity of UniBo is to fill the gap in the state of the art concerning the lack of efficient solutions to monitor the power supply of modern SoCs (e.g., based on the RISC-V architecture).

In order to achieve this goal, we plan to analyze, at the electrical level, the effects of the most likely permanent faults and aging phenomena possibly affecting *Fully Integrated Voltage Regulators* (FIVRs), that are typically adopted in modern SoCs to generate the required power supply voltages [56]–[58].

For our analyses, we plan to consider, as case study, the **FIVR** implementation in [59], that is representative of FIVRs that are currently used in multi-core high performance SoCs. Such a FIVR will be implemented considering a **22nm CMOS technology** (the same technology considered in [59]), with an input voltage Vin = 1.8V. In addition, we will emulate the load by means of a resistor with a variable resistance, in order to be able to simulate different values for the current absorbed by the load.

Then, by means of **HSPICE**, we will perform a fault simulation at the electrical level, in order to evaluate the effects of the most likely faults affecting the FIVR. In particular, we will evaluate the effects of all possible Transistor Stuck-Opens (SOPs), Transistor Stuck-ONs (SONs), and BFs with values of connecting resistance in the interval $[0\Omega, 100k\Omega]$. Again, we will consider the realistic assumption of faults occurring one at a time. We will also evaluate the impact of the Bias Temperature Instability (BTI) aging mechanism, which is recognized as the main aging mechanism for modern ICs [29]. In particular, we plan to evaluate the impact of BTI on the conductance of the transistors composing the FIVR by performing electrical level simulations with the **MOSRA tool** from **HSPICE** [28], considering a range of operating times between 0 and 10 years.

Also for this case we then plan to introduce metrics to evaluate the severity of the effects of the considered FIVR faults and BTI on the SoC correct operation.

Based on the results achieved by such analyses and metrics we will identify the most critical faults from the functional safety point of view. In a following research phase, we plan to develop low-cost monitors to detect variations in the FIVR generated power supply exceeding the limits required for the correct operation of the SoC in the field.

3.3.2 Self-Test Libraries

One strategy to provide on-line reliability monitoring for the hardware structures in complex systems (i.e., parallel devices and processor-based systems) is based on the clever and smart development of special software libraries to monitor the operation of the underlying hardware and identify possible error effects.. Indeed, the Software-Based Self-Test (SBST) strategy is a non-intrusive and flexible approach to provide functional hardware testing capabilities in processor-based systems [60]. This strategy has been successfully adapted to safety-critical domains, such as the automotive field, as a vital and complementary strategy to monitor and evaluate the effects of faults (i.e., permanent faults) arising in hardware components of a system [61].

The SBST strategy consist on developing compacted and efficient special programs (TPs) considering the underlying structures, the hardware architecture of a device, and their hardware-software interface for a later building of software Self-Test Libraries (STLs). These STLs are deployed at-speed on a target system and are able to verify and identify a considerable amount of hardware faults corrupting its operation. The STLs are composed of routines developed using purely machine instructions [62]–[77], high-level languages [78], or a clever combination of both[79]. Each TP consists of software routines providing patterns (*exciting patterns*) able to activate a hardware fault and then propagate their effects, so aiming to verify and identify hardware faults and potential effects on the internal structures of processor cores, as well as, their peripherals and associated hardware accelerators.

In particular, three approaches are mainly used for the development of TPs: *i) Automatic*, *ii) Deterministic*, and *iii) Custom*. The automatic approach resorts to special algorithms to analyze the structure of a target hardware unit and identifies the possible patterns activating most faults. Then, the patterns are converted into equivalent instructions to build routines and programs [80], [81]. A variation of the method includes using random and pseudo-random machine instructions to generate as many patterns as possible to detect faults in hardware. Both techniques can be complemented by software compacting algorithms to reduce the size and improve execution performance while providing acceptable fault coverage [62]. The deterministic approaches exploit well-known algorithms to generate test programs and address specific structures in the cores (e.g., controllers, schedulers, or the register file). In contrast, custom approaches require details of the underlying hardware architecture to determine the most efficient combination of instructions for the functional test (e.g., the arithmetic and logic unit). The SBST-based TPs might include several test approach combinations. Moreover, SBST-based test programs are closer to hardware than other functional test strategies (e.g., acceptance or stability tests in the HPC domain [82]). In fact, their adoption into HPC systems represents an opportunity to provide additional mechanisms of on-line monitoring, in particular in the context of RISC-V-based platforms due to the availability of their structural and architectural details. Hence, these details can be exploited in the development of effective STLs.

Interestingly, the adoption and evaluation of SBST-based solutions require the development and adaptation of several prototypes, frameworks and tools to support the development and evaluation of TPs. Figure 8 depicts the classification of the different frameworks, environments and tools that we consider can be used to design and evaluate STL solutions. It must be noted that the development of effective solutions require low-level descriptions of a system (e.g., gate-, RT-, and architectural levels of processor cores), see Table 1. Thus, also the associated and selected tools are focused on these levels of description.

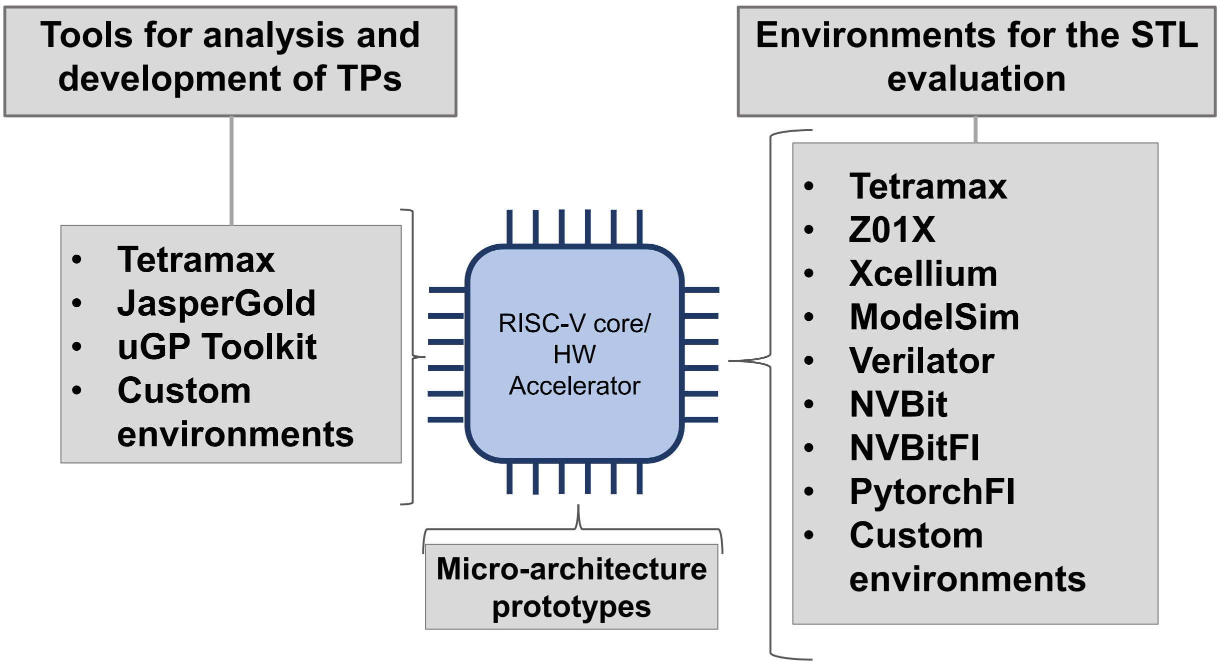

Tools for analysis and development of TPs
Tetramax
JasperGold
uGP Toolkit
Custom environments
RISC-V core/ HW Accelerator
Micro-architecture prototypes
Environments for the STL evaluation
Tetramax
Z01X
Xcellium
ModelSim
Verilator
NVBit
NVBitFI
PytorchFI
Custom environments

Figure 8: A general scheme of the selected tools and environments for the development and evaluation of on-line reliability monitoring solutions based on STLs.

Among the environments used in the analysis and development of TPs, the Automatic Test Pattern Generators or ATPGs tools (e.g., **Tetramax** [83], [84]) and evolutionary algorithm-based tools, such as **µGP** [85] support the analysis of the internal structures of a device and identify the feasible test patterns to be used in the description of the TPs. In particular, evolutionary algorithms can outperform experts and conventional heuristics in providing effective test patterns that can later be translated into equivalent assembly-based programs.

In addition, a set of frameworks, tools, and development environments allow the evaluation, validation and verification of the STL's operation. These tools are used to perform logic simulation of a system at micro-architecture level. Some of these environments (e.g., ***Xcelium***, ***ModelSim***, and ***Verilator***[10]) support the evaluation of STLs (through logic fault injection campaigns) at different abstraction levels, i.e., from fine-grain low-level microarchitectural implementations (e.g., RT- and gate-level in VHDL and Verilog) up to architectural abstraction of the system (e.g., SystemC and SytemVerilog). Other environments (e.g., ***ZOIX***[86], and ***Tetramax***) are specialized on evaluating and analysing the micro-architectural impact of faults on the structures under evaluation and can be adopted to develop elaborated and extended evaluation environments (e.g., representing conditions of in-field operation) [13], [22], [23].

Additionally, other models, tools, and environments (e.g., ***GPGPU-SIM***[11], ***NVbit***[12], ***NVBitFI***[13], ***NVBit-PERFI***[14], and ***PytorchFI***) support the evaluation and validation of the proposed STL solutions at higher abstraction levels (i.e., architectural, system, and application levels). Similarly, the architectural and system models/prototypes can be used to perform design exploration and evaluate feasible trade-off of a proposed

[10]`https://www.veripool.org/verilator/`
[11]`https://github.com/gpgpu-sim`
[12]`https://github.com/NVlabs/NVBit`
[13]`https://github.com/NVlabs/nvbitfi`
[14]`https://github.com/divadnauj-GB/nvbitPERfi`

STL solution. In some cases, the proposed STLs can be evaluated on real platforms under the supervision and support of these environments (e.g., GPU's STLs can be evaluated in real GPU platforms by resorting to *NVBit* or *NVBitFI* environments).

3.4 Languages (automata-based) for Specifying Performance Indices for High-Performance Computing Systems and for Anomaly Detection.

An automata-based performance index is a performance index that is specified using an automata, and that can compute values according to the rules that label the automaton nodes and edges. Automata-based methods have been implemented in both academic and industrial automated-verification tools (e.g., COSPAN, SPIN, ForSpec, and NuSMV). These methods can be used to specify performance indices for high-performance computing systems by allowing researchers to specify performance indices as automata (or using a language that generates automata, like LTL or PLTL) in a way that is both precise and concise.

Automata-based methods can either run using pre-recorded logs, or can run to monitor a process in real-time. The former is easier ti reproduce, as the log is already sequentialized, but it may be unfeasible if the event log of the monitored processes is too large, which is not unusual for High-Performance Computing Systems. The latter case thus is more general, as indices can be computed in real-time with a low memory and computational footprint.

Implementing real-time automata-based monitoring systems remains challenging, as several constraints (efficiency, low resource consumption, ease in property specification) have to be met. Complex properties can be exploited to define monitoring rules that control an HPC system and detects anomalies.

3.5 System-level Fault tolerance for Real-time Applications

Resilience to faults is traditionally implemented with hardware solutions, such as the ones described in previous Section 3.2. Another solution is to implement the different aspects of the fault tolerance at the software-level and, in particular, at the operating system level. These techniques are under the umbrella term *Software-Implemented Hardware Fault Tolerance (SIHFT)* [87]:

- *Fault detection* techniques, such as range checks of the input/output values, plausibility checks of the output values, error detection codes, an external monitoring device, control flow monitoring, and watchdogs. The goal of the detection mechanisms is to identify, as soon as possible, the occurrence of a fault by looking at the temporal and logical properties of the running software tasks.
- *Fault recovery* techniques, which are categorized in *space redundancy* and *time redundancy*. The former includes task replicas and standby tasks strategies: each task runs replicated multiple times (possibly on different processing units); then, a voting procedure decides which is the correct result among the task replicas. Fault detection techniques can also be used to determine which one is the correct output. The standby strategy consists in performing the full execution of the replicated tasks only if it is necessary. The *time redundancy* category, instead, includes approaches that react to a fault detected by the fault detection. In this case, the most common approaches are: re-execution (sometimes called *retry mechanism*), checkpoint/restore (sometimes called checkpoint/restart), recovery blocks, code correcting codes, and forward and backward error recovery.
- *Fault prediction* techniques, which can be further divided in:
  - *Prognostic techniques*: An algorithm analyzes the history and the current values of on-board sensors and the operating system metrics in order to recognize, and anticipate, a potential component

fault in the future. Anticipating the fault allows the system to reconfigure itself to prevent the fault to scale up to the application failure.

– *Predictive techniques*: Previous experience of faults in the cluster is the input provided to the fault prediction algorithm, that tries to determine the current aging of the system and triggers the tuning of certain system-level parameter to keep the reliability in acceptable value ranges.

Regarding fault detection, recent tools implemented into compilers, especially LLVM, can be used to automatically implement SIHFT techniques transparently to the developer [88]. These tools are still experimental and may require extensions and further research, especially with the integration with HPC libraries. Fault recovery is often implemented with checkpoint/restore, such as approaches based on CRIU[15], and other framework. On the application side, some libraries, such as Open MPI, support checkpoint/restore via transparent methods, such as Berkeley Lab Checkpoint/Restart (BCLR) [89], or more recently, based on User-Level Failure Mitigation (ULFM) [90]. Novel techniques for fault prediction can be employed at system-level in HPC, in particular the prognostic techniques, that can exploit the information provided by the on-board sensors. This can also be applied to heterogeneous hardware, such as GPU or FPGA.

When predictive fault tolerance is available, whether prognostic or based on historical data, the system must take actions to anticipate the fault and save the work performed by the workload. This includes the task migration, which can occur inside the same node (switching active processing element), inter-core, when a core is suspected to fail in a short time, or inter-node, when the whole machine (or one of its critical component) does not reach the required online reliability value. This migration needs to have the support from the application, similarly to checkpoint/restore or a transparent approach must be used, such as the MIG framework [91]. Orchestrating the workload migration is a complex task usually delegated to resource managers, such as BarbequeRTRM [92], which are able to take smart decision with as minimal as possible impact on the application performance.

Implementing SIHFT and workload migration (at any level) present numerous challenges when the applications must satisfy real-time constraints. Time-critical applications need to satisfy the time constraint even in a case of fault, otherwise the recovery from a fault may produce no beneficial effects. Scheduling policies must be aware of the failure requirements and the presence of SIHFT mechanisms. Recently, novel models that integrate failure requirements and real-time requirements have been developed [93], [94]. Further developing these models and implement them in the HPC context is a key enabler to allow real-time and fault tolerant applications on HPC clusters. Another important issue is to determine the WCET via proper tools. Indeed SIHFT approaches often require to perform re-execution or running in replica-mode multiple tasks. Therefore, a tight WCET estimation is very important, to avoid duplicating, or even more, the over-approximations of existing tools. In this context, we can exploit the probabilistic estimations, such as the `chronovise` tool, to obtain a tight estimation [95]. A list of existing works and tools is available in a recent survey [96].

[15]`https://criu.org/`

4 PERFORMANCE MONITORING PLATFORMS

Monitoring the application performance is the key enabler of resource management. Effective monitoring must be taken at different abstraction levels.

- At OS level, the operating system must be aware that the same time allocated to a task may not mean an equal progress in the execution as processor frequency, speculative execution, caches and other HW accelerations may result in non-deterministic improvements. Section 4.1 illustrates these aspects in greater details.
- At runtime level, the progress of parallel applications can be monitored through the number of inter-thread messages waiting to be served as explained in Section 4.2. Also, in the context of saving energy, low-power modes can be entered whenever the sleep times are enough to compensate the overheads (Section 4.3).
- Finally, at VM level, monitoring can be effectively used to detect anomalies in the resources used by the tenants. Section 4.4 illustrates a monitoring infrastructure of VMs.

4.1 Kernel-based Job Execution Monitoring

Simultaneous Multi-Threading (SMT) was introduced to increase the computing capacity of processors. In SMT, a number of *Hardware Threads (HTs)* may execute in parallel[16]. From the first released processor to the latest ones, SMT has kept the following common traits:

- a **dedicated** set of registers, including the Instruction Pointer, per HT, and
- a **shared** set of processing units.

Over the years, architectures then differed in the type and the number of shared processing units. Intel's Core 'i' Series (i7, i9, ...) normally have two HTs per physical core. In AMD's Bulldozer, every HT has a dedicated ALU while they share FPUs. In IBM's Power10, the available HTs are 8 per core, but the Matrix-Multiply Assist (MMA) accelerators are shared by two HTs, whereas the Quad-Precision floating-point (QP) and Decimal floating-point (DF) unit is shared by 4 HTs[17]. Interestingly, IBM's PowerXX architectures offer the possibility to program pre-fetch units [97]. By doing so, the speed of HTs can be modified at software/OS level.

The intimate bond between HTs sharing computing or storage elements was recognized as a source of side-channel attacks [98]–[100]. In fact, the code executed by one HT leaves a trace and malicious code executing on the sibling HT may exploit such information. RIDL [101], Spectre [102], Meltdown [103] are just a few examples of attacks discovered so far that exploit shared elements among HTs.

In the Linux kernel, from version 5.14 released August 2021, the mitigation of these vulnerability is made through the *core scheduling*. In Linux's core scheduling, it is possible to specify which tasks are allowed to execute on HTs belonging to the same physical core (possibly because they trust each other), and which ones are not[18]. This mechanism is effective in confining the execution of tasks of the same group to HTs of the same core. However, it necessarily reduces the utilization of the available processing capacity.

[16]The term "Hardware Thread" is used in this section as generic synonym of different vendor-dependent branded names: "hyper-threads" in Intel's Hyper-Threading Technology, Virtual Processing Element (VPE) in MIPS64-based processors, "ways" in 2-way or 4-way SMT provided by AMD's Bulldozer or Zen processors, and more.

[17]IBM Power E1080: Technical Overview and Introduction. `https://www.ibm.com/products/power-e1080`

[18]`https://www.kernel.org/doc/html/latest/admin-guide/hw-vuln/core-scheduling.html`

The presence of SMT, speculative execution and different levels caches does improve benchmark-based performance. However, how can we predict the performance losses or gains due to current and future architectural accelerations? How do OS mechanisms such as the core scheduling affect performance? It is necessary to develop a low level monitor of execution and then to correlate the measured execution times to hardware characteristics. Possible implementation mechanisms include:

- the Linux kernel `ftrace` tracing infrastructure[19],
- the Runtime Verification[20], available in the Linux kernel from version 6.0 to build automata that monitor condition of interest,
- the evaluation of losses in the performance due to the kernel patches that fix vulnerabilities due to SMT and other speculative execution accelerations.

4.2 Monitoring Structured Parallel Programs

The *Structured Parallel Programming* (SPP) methodology is a parallel programming approach fostering the composition and nesting of few parallel components to express arbitrarily complex parallel computations. In its first application, with so-called *Algorithmic Skeletons* libraries and frameworks [104], [105], such components are in the form of programming skeletons, i.e., classes of object-oriented languages or constructs in ad-hoc parallelism coordination languages that express recurrent parallelism models. Examples of skeletons are map, reduce, stencils, divide-and-conquer, pipelining, functional replication and others. A common aspect of all the major skeleton-based languages and libraries is the rigid exploitation of few skeletons (and their composition) to express complex applications by hiding low-level implementation details to the application developer (usually an expert of the application domain more than an expert of parallel programming). More recently, the SPP concepts have been extended and revisited with the broader notion of *parallel patterns* [106], where patterns are high-level recipes that can be implemented by the users (i.e., application developers) through proper programming constructs/mechanisms provided by the parallel programming framework, so alleviating the limits in terms of expressiveness of traditional skeleton-based systems at the expense of lowering the abstraction level and increasing the programming effort.

The FastFlow programming environment. FastFlow [107] is a parallel programming framework developed in C++ that merges both visions of SPP: it provides two main abstraction layers as shown in Fig. 9:

- high-level algorithmic skeletons, where the application developer can instantiate complex predefined patterns by providing few business logic functions having the right expected signature;
- an intermediate layer called *building blocks*, which can be used by more expert programmers to develop new parallel run-time systems for specific purposes (e.g., the WindFlow parallel library for Data Stream Processing [108]) as well complex parallel patterns not directly expressible through the direct use of the built-in skeletons available in the library.

Building blocks are of special importance for FastFlow. They allow the definition of a large set of data-flow graphs of concurrent/parallel entities exchanging pointers in shared-memory architectures or communicating through distributed channels when the graph is partitioned among more computing nodes. In doing so, FastFlow fosters a *single-source programming model* where the same program can be transparently executed on a single multi-core architecture or even in more nodes requiring minor modifications to the source code and to the configuration setup of the application. Fig. 9 shows a graphical representation of the FastFlow ecosystem.

[19]`https://www.kernel.org/doc/html/v5.0/trace/ftrace.html`
[20]`https://docs.kernel.org/trace/rv/index.html`

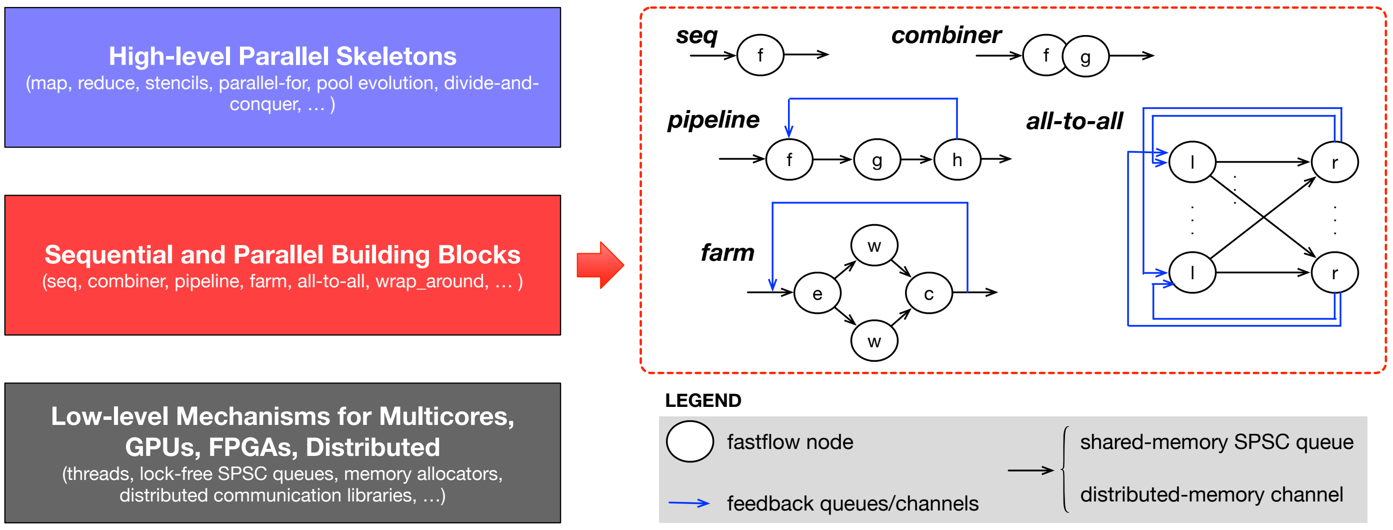

High-level Parallel Skeletons
(map, reduce, stencils, parallel-for, pool evolution, divide-and-conquer, …)
Sequential and Parallel Building Blocks
(seq, combiner, pipeline, farm, all-to-all, wrap_around, …)
Low-level Mechanisms for Multicores, GPUs, FPGAs, Distributed
(threads, lock-free SPSC queues, memory allocators, distributed communication libraries, …)
seq
combiner
pipeline
all-to-all
farm
f
g
h
e
w
c
l
r
LEGEND
fastflow node
feedback queues/channels
shared-memory SPSC queue
distributed-memory channel

Figure 9: Layered architecture of `FastFlow`.

Performance monitoring in FastFlow. Several performance parameters can be inspected and monitored at runtime during the application execution. These parameters are of special importance to assess the quality of the parallel execution on a given platform, and to discover bottlenecks in the concurrency graph of building blocks. Furthermore, the continuous monitoring of performance parameters can be used to drive an actionable logic that can dynamically reconfigure the behavior of a FastFlow application in terms of different actions, such as:

- *thread pinning*: FastFlow application starts with a default pinning that might be disabled at compile time using a proper compilation macro. If enabled, FastFlow pins threads onto cores in a straightforward manner, by numbering threads with progressive indexes and assigning them to virtual cores in a linear manner. In NUMA machines, different strategies can be useful to improve performance (i.e., to minimize memory access time), and this can be controlled by inspecting monitoring information available at runtime;
- *concurrency throttling*: FastFlow skeletons, and building blocks too, can be started with an oversized number of threads that express the highest possible parallelism degree on a given machine. FastFlow is capable of starting/deactivating threads by sending special messages through the same queues/channel used to forward information. This can be used to dynamically control how many threads the application is currently using, in order to control resource utilization and energy consumption in different platforms (e.g., by turning off some threads pinned on a set of cores, the user can turn off the socket/island where those cores are placed in the system to save energy);
- *concurrency control*: FastFlow comes with different synchronization primitives to exchange data between threads. Default mechanisms are based on busy-waiting lock-free queues, which are highly responsive but consume many CPU cycles through spin loops on shared flags. This can impair energy consumption on some devices, and can be detrimental to performance where more threads are placed onto virtual cores of the same physical cores, so sharing CPU resources. FastFlow also provides more resource conservative synchronization mechanisms (based on short process suspension phases according to exponential backoff strategies), which are less eager and can suspend threads in case of long waiting conditions. The runtime system can dynamically change the used implementation of synchronization mechanisms (globally from the application-wise perspective at the moment) in order to trade off performance with resource consumption (e.g., CPU cycles and, so, energy consumption).

All these actions can be driven, as said, by the continuous monitoring of runtime information that can be:

- the actual *working time* spent by a FastFlow node (i.e., the basic sequential block of FastFlow, which is responsible to run a sequential processing logi over each input from its input channels to produce outputs transmitted onto its output channels). It is the total time spent by the node in running user-defined functions within the node, so excluding the time spent in run-time system activities (e.g., for popping new inputs and for delivering outputs to other nodes);
- the total number of *processed inputs* by the FastFlow node from the beginning of the execution. Analogously, it is possible to track the number of processed outputs and so counting the so-called *selectivity* of the node, i.e., the number of outputs produced per input;
- the number of *failed pop/push operations* from/to the input/output queues of a node. We recall that a push fails if the destination queue is full, while a pop fails just in case the input queue is empty. Such an information can be useful to understand whether a node is a bottleneck or not. For example, a node spending much time in popping from the input queue is a node whose arrival rate of inputs is on average smaller than its service rate, so having a low utilization.

Such parameters can be retrieved at least at the end of the execution, if the skeleton/graph of nodes is executed synchronously with respect to the main thread. In doing so, monitoring information can be retrieved for profiling purposes, e.g., for discovering bottlenecks offline. However, FastFlow programs can be run asynchronously, and the main thread can start a loop where periodically such a monitoring information can be accessed and inspected at runtime in order to report statistics to the user or, if possible, to change the behavior of the FastFlow program through reconfiguration activities such as the ones mentioned before.

Energy monitoring with the Mammut library. The native performance monitoring support provided in FastFlow can be further extended by relying on external libraries such as Mammut [109]. Mammut is a C++ library providing a user-friendly API to monitor parallel applications. Most of the activities performed with Mammut require superuser privileges to be effectively done. The library has a module-based architecture providing different features. The *topology* module can be used to retrieve a high-level description of the underlying multi-core platform in terms of virtual cores, physical cores, cache hierarchy, and grouping of cores in terms of sockets (like `lstopo`). The API of the module allows the user to easily turn on/off specific cores or sockets, or to manage idle states (C-states). Unfortunately, the supported CPUs are quite old Intel models (up to the Haswell family of processors), and how to extend the support to newer models, as well as to support AMD and ARM multicores, is worth of investigation. The *energy* module can be used to measure the energy consumed by the hardware for running a specific piece of code. This can be done in a quite customizable manner, by inspecting the energy consumed by specific hardware components (CPUs, memory, and so forth). The *frequency* module can be used to easily play with governors, in order to control the operating frequency of the CPUs. Finally, the *taskmanager* module can be used to easily change the mapping of threads onto cores of the machine, by providing a high-level API for doing that. The library is a remarkable tool to enhance the monitoring support of different parallel programming frameworks, not only FastFlow but others for shared-memory systems such as OpenMP or Intel oneAPI programs. However, the support of new CPUs of different vendors is still lacking and should be integrated in the future.

4.3 COUNTDOWN library

Power and energy consumption is nowadays key challenges in supercomputing field. Large-scale HPC applications waste a significant amount of power in communication and synchronization idle times. However, due to the time scale at which communication happens, transitioning in low power states during idle times may introduce overheads due to switching.

In fact, a typical HPC application is composed of several processes which exchange messages through a high-bandwidth and low-latency network, among different nodes of a cluster. And these processes access the network sub-system through a software interface that abstracts the network level. The Message-Passing Interface (MPI) is a software interface for communication, that allows processes to exchange explicit messages by abstracting the network level.

When the scale of the application increases, the time spent in the MPI library becomes non negligible, impacting the overall power consumption. By default, when MPI processes are hanging in a synchronization primitive, the MPI libraries use a busy-waiting mechanism which, along with IO/memory accesses, makes up the bulk of the workload during MPI primitives. But these idle-waiting mechanisms are not used in practice to avoid performance penalties caused by the transition times into and out of low-power states. As a matter of fact, there is no known low-overhead and reliable mechanism, for reducing energy consumption selectively, during MPI communication slack.

Regarding the most common approaches on energy reduction for real applications, low power design strategies trade performance for power consumption, by mean of low power modes of operation obtained by Dynamic and Voltage Frequency Scaling (DVFS) (P-States), clock gating or throttling states (T-states), and idle states which switch off unused resources (C-states). These power states transitions are controlled by hardware, OS and user-space policies.

While OS policies try to maximize the usage of the computing resources, increasing the processor's speed (P-state) proportionally to its utilization, two main families of power control policies are emerging in scientific computing. The first is based on the assumption that the performance penalty can be tolerated to reduce the overall energy consumption[110][111]. The second focuses on the idea that it is possible to slow down a processor, only when it does not execute critical tasks, saving energy without penalizing application performance[112][113].

However, there are some drawbacks:

- time to solution (TTS) is the one correlated to the first approach; limiting the overall supercomputer throughput and capacity, of course penalizes the time required to conclude a specific job;
- prediction of critical tasks is instead associated with the second approach. When we have a lot of mispredictions in getting the path of a specific application, we can observe severe performance losses.

These are some reasons why we propose to adopt COUNTDOWN [114], [115] as a runtime library. Also, it is supported by a methodology and analysis tool for identifying and automatically reducing the power consumption of the computing elements, during MPI communication and synchronization. COUNTDOWN saves energy without imposing significant time-to-completion increase, by lowering CPUs power consumption only during idle times for which power state transition overhead are negligible. This is done transparently to the user, without requiring labor-intensive and error-prone application code modifications, nor requiring recompilation of the application.

COUNTDOWN library is able to automatically track at fine granularity MPI and application phases, injecting power management calls whenever it is convenient to do so. COUNTDOWN can be dynamically linked with the application at loading time: it can intercept dynamic linking to the MPI library, instrumenting all the application calls to MPI functions before the execution workflow jumps to the library.

The runtime also provides a static version of the library which can be connected with the application at linking time. COUNTDOWN supports C/C++ and Fortran HPC applications, and most of the open-source and commercial MPI libraries.

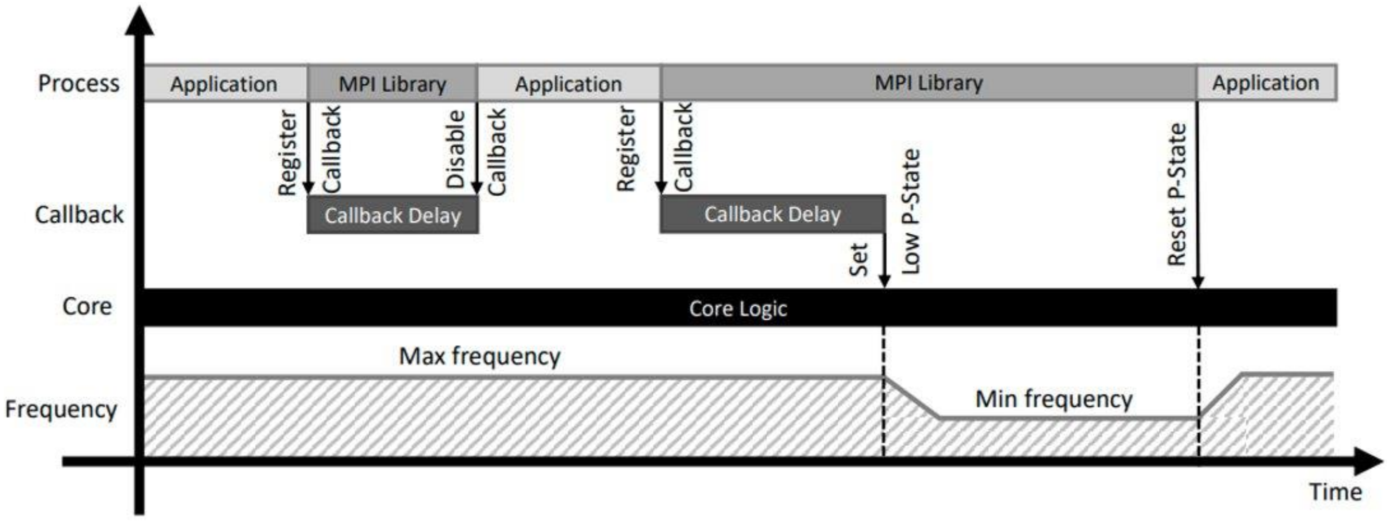
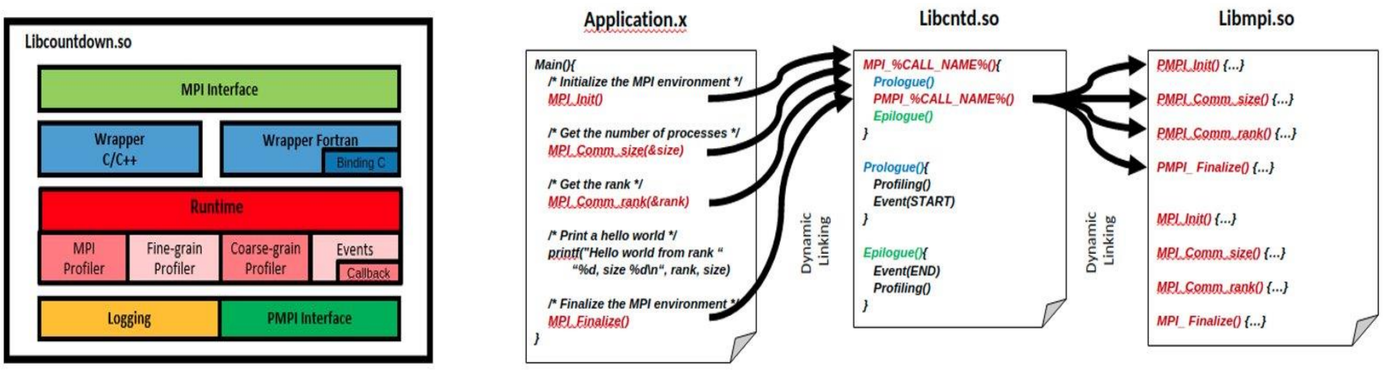


Figure 10: COUNTDOWN is a tool to identify and automatically reduce the power consumption of the CPU cores, during communication phases of an MPI-based application, and it is built upon a pure reactive mechanism (based on a timer), without using a learning mechanism. This image is from [114].



```
Main(){
   /* Initialize the MPI environment */
   MPI_Init()

   /* Get the number of processes */
   MPI_Comm_size(&size)

   /* Get the rank */
   MPI_Comm_rank(&rank)

   /* Print a hello world */
   printf("Hello world from rank "
        "%d, size %d\n", rank, size)

   /* Finalize the MPI environment */
   MPI_Finalize()
}
```

```
MPI_%CALL_NAME%(){
   Prologue()
   PMPI_%CALL_NAME%()
   Epilogue()
}

Prologue(){
   Profiling()
   Event(START)
}

Epilogue(){
   Event(END)
   Profiling()
}
```

```
PMPI_Init() {...}

PMPI_Comm_size() {...}

PMPI_Comm_rank() {...}

PMPI_ Finalize() {...}


MPI_Init() {...}

MPI_Comm_size() {...}

MPI_Comm_rank() {...}

MPI_ Finalize() {...}
```

Figure 11: COUNTDOWN does not require modification of source code nor rebuilding process, just the usage (and the exportation) of the environmental variable LD_PRELOAD=path_to_libcntd.so. This image is from [114].

Also, COUNTDOWN can extract workload traces of real HPC applications, using a user-defined time-based approach.

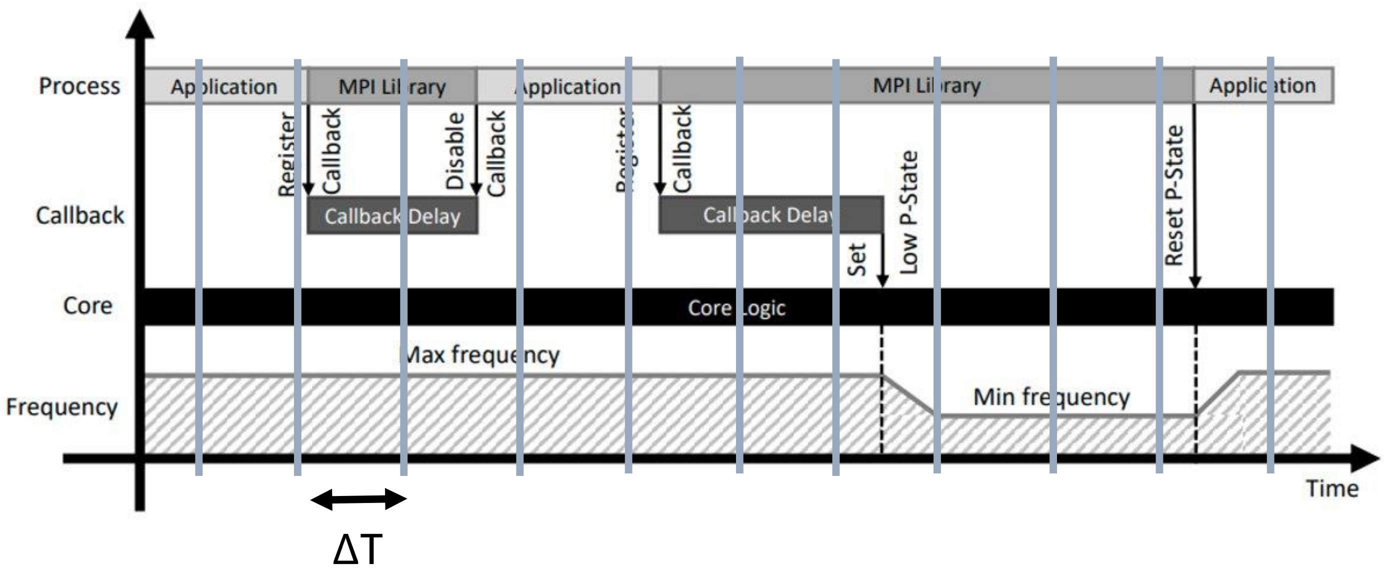


Figure 12: The user can choose the desired ΔT at which he wants to sample his own application, exporting the variable CNTD_SAMPLING_TIME.

This feature is quite important with today's new architectural and micro-architectural features, like superscalarity, out-of-order (OoO) execution, multi-level caches, complex instructions, multi-threading, manycore, multi-socket with DVFS, non-uniform memory access, and so on.

Performance's events are monitored using counters which are present, on a CPU, in a fixed number. If there are more events than counters to be measured, time multiplexing gives to each event a chance to access the monitoring hardware. COUNTDOWN implements this approach (using under the hood Linux Perf), to provide a final estimate of the desired hardware events, following this formula:

$$final\_count \;\; raw\_count \times \frac{time\_enabled}{time\_running}$$

where *time_enabled* is the time event active, and *time_running* is the time event CPU.

Events are usually divided into two macro areas:

- on-core: microarchitecture events at the core level (cycles, instructions retired, ...);
- off-core: microarchitecture events outside of the cores (memory read/write, ...).

This subdivision can lead us to the definition of a model, to provide informations related to the performances of a specific application: the Roofline Model[116], which instead of trying to predict performances, uses a bound and bottleneck approach.

Future extensions of COUNTDOWN will take into consideration this performance model, to give to the user an estimation of how well performed his/her application. Of course, these future works shall evaluate peak performances and bandwidth of the underlying system, taking into account multiple architectures and micro-architectures, and so to be able to count the right and specific on-core and off-core events. This task can be challenging, because not always the specific reference manuals report the hexadecimal codes of the desired events, and therefore other ways must be pursued to find them, and to be able to measure them correctly.

4.4 Monitoring VMs for anomaly detection

The goal of this prototype is to develop new tools to support cybersecurity in cloud platforms. In particular, we aim at exploiting cloud telemetry together with control plane information from the cloud network to monitor the status of tenants' virtual machines. The idea is to leverage safe and privacy-friendly metrics from the cloud telemetry to monitor resource usage and activity of the tenants' virtual machines, identifying situations in which workloads show signs of compromise to alert both the cloud operator and tenants about the anomalies. In this context, this prototype will answer the following questions:

- What are the most effective approaches for collecting the metrics needed to identify compromised machines in cloud environments?
- How should these metrics be used to learn the tenants' fingerprints and how to learn these fingerprints effectively and timely?

Cloud telemetry solutions provide large volumes of metrics about the resources used by tenants. These metrics are typically used for monitoring resource utilization and/or billing. As such, performing detection and identification of compromised virtual machines using cloud telemetry could represent a powerful tool for the host and tenants while introducing limited overhead to the infrastructure. The envisioned architecture for the prototype is composed of three key blocks: the `Telemetry`, the `Monitor`, and the `Controller` modules.

The tenants' virtual machines are monitored by the `Telemetry` module, which collects metrics about tenants' VMs and information about the virtual networks. These metrics are saved for posterior analysis and used to infer the status of tenants' infrastructure. This prototype will consider as candidate the OpenStack telemetry modules. OpenStack is an open-source cloud computing platform that enables the control of large sets of computing, storage, and networking resources. OpenStack offers flexible monitoring capabilities by means of its Ceilometer module.[21] Hundreds of metrics are available,[22] from parameters of virtual machines to usage of virtual network interfaces.

Ceilometer is a natural candidate given its wide range of metrics and seamless integration into the platform. However, it is unclear whether these metrics are sufficient for identifying anomalies in virtualized environments. Indeed, legitimate and malicious workloads do present several characteristics in common, and distinguishing between them without directly observing tenants' systems and traffic is a challenging task. Moreover, Ceilometer adds considerable overhead to the platform when performing fine-grained monitoring of resources (e.g., at a millisecond granularity). These trade-offs are indeed a key research problem that needs to be answered in the first development phase of this prototype.

The `Monitor`'s main goal is to build signatures for anomalous workloads based on information from the `Telemetry`. These fingerprints will come from controlled environments (sandboxes), in which virtual machines with known vulnerabilities and (controlled) malicious software will be used to collect information about well-known attacks. The main activity in this second phase will be automating data collection methodology to allow feasible and timely constructions of attack fingerprints. This is a complex task that requires combining both (i) frameworks for cybersecurity testing (e.g., Metasploit) and (ii) learning approaches to identify key features from the telemetry that characterize attacks, thus building models able to identify attacks in real/noisy environments.

Finally, the goal of the `Controller` is to provide a feedback loop that acts when anomalies are identified. We believe this module can be implemented with existing cloud technologies, and indeed the prototype will directly borrow from OpenStack architecture. For example, OpenStack already includes an `autoscaler` that allows for the allocation/removal of resources for the provision of sandboxes needed for building signatures, providing elasticity to the service. Equally, OpenStack's `networking manager` relies on Network Function

[21]https://github.com/openstack/ceilometer

[22]The list of all monitored features is available at https://docs.openstack.org/ceilometer/rocky/admin/telemetry-measurements.html

Virtualization (NFV) and Service Function Chaining (SFC) to provide network isolation as well as to perform on-the-fly changes in the network flows (e.g., diverging traffic once VMs are identified as anomalous). Moreover, OpenStack security features can readily be used to set up rules to act when services are identified as anomalous by the `Monitor`. All these functions will be evaluated and supported by the planned prototype.

5 CONCLUDING REMARKS

This document analyses and introduces several feasible candidate prototypes to allow the evaluation, the proposal of methods and strategies, and the validation of solutions for two crucial non-functional properties: 1) *Energy* and 2) *Reliability* in components, units, modules, and systems employed in current and future generations of HPC machines, especially focusing on RISC-V-based platforms and associated architectures. The candidate prototypes include computational models, frameworks, development environments, software artifacts, test beds, and proof of concepts that provide many equivalent functional and operational features of real HPC systems under controlled environments for experimentation and development.

In general, we identify three major stages for the evaluation of the non-functional properties: *i)* Power evaluation platforms, *ii)* Reliability evaluation platforms and models, and *iii)* performance monitoring infrastructures and artifacts.

In the first case, Chapter 2 introduced and analyzed the feasible platforms and prototypes for the proposal of solutions and the evaluation of power management in HPC systems. In particular, we analyze possible steps for the development of mechanisms for power management, as well as methods for their evaluation considering the different components of a generic heterogeneous HPC platform. Then, the global power monitoring at coarse grain and the problem of the cooling are addressed. In the latter case, it is presented a valuable research path that will be followed during the project regarding the exploitation of a 2-phase cooling technology, that will be prototyped within the project timeframe.

Then, Chapter 3 analyzed the feasible physical and computational models for the analysis and evaluation of the reliability of HPC systems based on fault simulation strategies. In this case, the increasing complexity of systems involved in HPC machines and the propagation of physical defects as faults, errors, and failures require a smart and clever combination of different abstraction levels (e.g., cross-layer evaluation and state-of-the-art co-simulation schemes). Thus, according to the abstraction level of each analysis (e.g., technology, architecture, or application levels), the selection and use of specific prototypes, models, frameworks, and artifacts are required. For each case, the setting of parameters in the prototype and its associated environments pursue the standard operational features of the systems related to HPC machines. Interestingly, the reliability evaluation and the proposal of hardening solutions can reuse some of the selected prototypes and their associated development environments with minimal changes according to their evaluation objectives.

Finally, we identify that a key element for controlling the processing capacity allocated to the many workloads is the monitoring of resource usage. Such monitoring can be made at different levels. In Section 4, we have been illustrating how we propose to monitor applications and resource usage at the OS level, at the run-time level, and at the VM level.

It is worth noting that most of our identified candidate prototypes are open-source and can be adapted according to the evaluation targets. Furthermore, we also consider several commercial-grade tools and frameworks to simplify the development and evaluation steps of methods, strategies, and solutions.

References

[1] J. Arnowitz, M. Arent, and N. Berger, *Effective prototyping for software makers*. Elsevier, 2010.

[2] M. Carr and J. Verner, "Prototyping and software development approaches," *Department of Information Systems, City University of Hong Kong, Hong Kong*, pp. 319–338, 1997.

[3] D. Zoni, L. Cremona, A. Cilardo, M. Gagliardi, and W. Fornaciari, "Powertap: All-digital power meter modeling for run-time power monitoring," *Microprocessors and Microsystems*, vol. 63, pp. 128–139, 2018, ISSN: 0141-9331. DOI: `https://doi.org/10.1016/j.micpro.2018.07.007`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0141933118302308`.

[4] S. Cherubin, D. Cattaneo, M. Chiari, A. Di Bello, and G. Agosta, "TAFFO: Tuning assistant for floating to fixed point optimization," *IEEE Embedded Systems Letters*, 2019, ISSN: 1943-0663. DOI: `10.1109/LES.2019.2913774`.

[5] M. Chrobak, "SIGACT news online algorithms column 17," *SIGACT News*, vol. 41, no. 4, pp. 114–121, 2010, ISSN: 0163-5700.

[6] A. López-Ortiz and A. Salinger, "Minimizing cache usage in paging," in *Proceedings of the 10th Workshop on Approximation and Online Algorithms (WAOA)*, 2012, pp. 145–158.

[7] A. Gupta, R. Krishnaswamy, A. Kumar, and D. Panigrahi, "Elastic caching," in *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2019, pp. 143–156.

[8] M. A. Bender, R. Ebrahimi, J. T. Fineman, G. Ghasemiesfeh, R. Johnson, and S. McCauley, "Cache-adaptive algorithms," in *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2014, pp. 958–971.

[9] E. Peserico, "Paging with dynamic memory capacity," in *Proceedings of the 36th International Symposium on Theoretical Aspects of Computer Science (STACS)*, 2019, 56:1–56:18.

[10] K. Agrawal, M. A. Bender, R. Das, W. Kuszmaul, E. Peserico, and M. Scquizzato, "Green paging and parallel paging," in *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2020, pp. 493–495.

[11] K. Agrawal, M. A. Bender, R. Das, W. Kuszmaul, E. Peserico, and M. Scquizzato, "Tight bounds for parallel paging and green paging," in *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2021, pp. 3022–3041.

[12] H. Coles, "Direct liquid cooling for electronic equipment.," *Lawrence Berkeley National Laboratory. Retrieved from https:\\escholarship.org/uc/item/*$0g50294h, LBNL-6641E$, 2014.

[13] J. E. R. Condia *et al.*, "A multi-level approach to evaluate the impact of gpu permanent faults on cnn's reliability," in *2022 IEEE International Test Conference (ITC)*, 2022, pp. 278–287. DOI: `10.1109/ITC50671.2022.00036`.

[14] IEEE, "The international roadmap for devices and systems: 2022," in *Institute of Electrical and Electronics Engineers (IEEE)*, 2022.

[15] N. DeBardeleben, S. Blanchard, D. Kaeli, and P. Rech, "Field, experimental, and analytical data on large-scale hpc systems and evaluation of the implications for exascale system design," in *2015 IEEE 33rd VLSI Test Symposium (VTS)*, 2015, pp. 1–2. DOI: `10.1109/VTS.2015.7116295`.

[16] D. Tiwari, S. Gupta, J. Rogers, *et al.*, "Understanding gpu errors on large-scale hpc systems and the implications for system design and operation," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, 2015, pp. 331–342. DOI: `10.1109/HPCA.2015.7056044`.

[17] D. A. G. D. Oliveira, L. L. Pilla, M. Hanzich, *et al.*, "Radiation-induced error criticality in modern hpc parallel accelerators," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2017, pp. 577–588. DOI: `10.1109/HPCA.2017.41`.

[18] *Iso 26262-1:2011(en) road vehicles — functional safety — part 10: Guideline on iso 26262, international standardization organization*, 2011.

[19] J. Sun *et al.*, "Pinpointing crash-consistency bugs in the hpc i/o stack: A cross-layer approach," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '21, 2021, ISBN: 9781450384421.

[20] P. Kousha *et al.*, *Cross-layer visualization and profiling of network and i/o communication for hpc clusters*, 2021.

[21] W. Li, C. Nigh, D. Duvalsaint, S. Mitra, and R. Blanton, "Pepr: Pseudo-exhaustive physically-aware region testing," in *2022 IEEE International Test Conference (ITC)*, 2022, pp. 314–323. DOI: `10.1109/ITC50671.2022.00083`.

[22] F. F. d. Santos *et al.*, "Revealing gpus vulnerabilities by combining register-transfer and software-level fault injection," in *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2021, pp. 292–304. DOI: `10.1109/DSN48987.2021.00042`.

[23] J. E. R. Condia *et al.*, "Combining architectural simulation and software fault injection for a fast and accurate cnns reliability evaluation on gpus," in *2021 IEEE 39th VLSI Test Symposium (VTS)*, 2021, pp. 1–7. DOI: `10.1109/VTS50974.2021.9441044`.

[24] *Predictive technology model (ptm)*, http://ptm.asu.edu, Feb. 2022.

[25] Y. Liu and Q. Xu, "On modeling faults in finfet logic circuits," in *2012 IEEE International Test Conference*, 2012, pp. 1–9. DOI: `10.1109/TEST.2012.6401565`.

[26] *Http://web.eecs.umich.edu/ jhayes/iscas.restore/benchmark.html*, http://ptm.asu.edu.

[27] K. Choi, H. C. Sagong, W. Kang, *et al.*, "Enhanced reliability of 7nm process technology featuring euv," in *2019 Symposium on VLSI Technology*, 2019, T16–T17. DOI: `10.23919/VLSIT.2019.8776580`.

[28] B. Tudor, J. Wang, W. Liu, and H. Elhak, "Mos device aging analysis with hspice and customsim," *Synopsys*, 2011.

[29] J. Keane, T.-H. Kim, and C. H. Kim, "An on-chip nbti sensor for measuring pmos threshold voltage degradation," *IEEE Trans. on Very Large Scale Integration Systems*, vol. 18, no. 6, pp. 947–956, 2009.

[30] D. A. G. Goncalves de Oliveira *et al.*, "Evaluation and mitigation of radiation-induced soft errors in graphics processing units," *IEEE Transactions on Computers*, vol. 65, no. 3, pp. 791–804, 2016.

[31] K. Ito *et al.*, "Analyzing due errors on gpus with neutron irradiation test and fault injection to control flow," *IEEE Transactions on Nuclear Science*, vol. 68, no. 8, pp. 1668–1674, 2021. DOI: `10.1109/TNS.2021.3098845`.

[32] F. Benevenuti *et al.*, "Investigating the reliability impacts of neutron-induced soft errors in aerial image classification cnns implemented in a softcore sram-based fpga gpu," *Microelectronics Reliability*, vol. 138, p. 114 738, 2022, 33rd European Symposium on Reliability of Electron Devices, Failure Physics and Analysis, ISSN: 0026-2714. DOI: `https://doi.org/10.1016/j.microrel.2022.114738`.

[33] J. E. R. Condia *et al.*, "Flexgripplus: An improved gpgpu model to support reliability analysis," *Microelectronics Reliability*, vol. 109, p. 113 660, 2020, ISSN: 0026-2714.

[34] M. Al Kadi, B. Janssen, and M. Huebner, "Fgpu: An simt-architecture for fpgas," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '16, Monterey, California, USA, 2016, pp. 254–263, ISBN: 9781450338561.

[35] T. Tsai *et al.*, "Nvbitfi: Dynamic fault injection for gpus," in *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2021, pp. 284–291.

[36] S. K. S. Hari *et al.*, "Sassifi: An architecture-level fault injection tool for gpu application resilience evaluation," in *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2017, pp. 249–258.

[37] S. K. Bukasa *et al.*, "When fault injection collides with hardware complexity," in *Foundations and Practice of Security*, Cham: Springer International Publishing, 2019, pp. 243–256.

[38] M. Omana, D. Rossi, and C. Metra, "Latch susceptibility to transient faults and new hardening approach," *IEEE Transactions on Computers*, vol. 56, no. 9, pp. 1255–1268, 2007. DOI: `10.1109/TC.2007.1070`.

[39] M. Omaña, D. Rossi, and C. Metra, “High-performance robust latches,” *IEEE Transactions on Computers*, vol. 59, no. 11, pp. 1455–1465, 2010. DOI: `10.1109/TC.2010.24`.

[40] M. Omaña, D. Rossi, T. Edara, and C. Metra, “Impact of aging phenomena on latches’ robustness,” *IEEE Transactions on Nanotechnology*, vol. 15, no. 2, pp. 129–136, 2016. DOI: `10.1109/TNANO.2015.2494612`.

[41] M. OmaÑa, T. Edara, and C. Metra, “Low-cost strategy to mitigate the impact of aging on latches’ robustness,” *IEEE Transactions on Emerging Topics in Computing*, vol. 6, no. 4, pp. 488–497, 2018. DOI: `10.1109/TETC.2016.2586380`.

[42] O. Goloubeva, M. Rebaudengo, M. S. Reorda, and M. Violante, *Software-implemented hardware fault tolerance*. Springer Science & Business Media, 2006.

[43] A. Avizienis and J.-C. Laprie, “Dependable computing: From concepts to design diversity,” *Proceedings of the IEEE*, vol. 74, no. 5, pp. 629–638, 1986. DOI: `10.1109/PROC.1986.13527`.

[44] N. Wu, D. Zuo, and Z. Zhang, “Dynamic fault-tolerant workflow scheduling with hybrid spatial-temporal re-execution in clouds,” *Information*, vol. 10, no. 5, 2019, ISSN: 2078-2489. DOI: `10.3390/info10050169`. [Online]. Available: `https://www.mdpi.com/2078-2489/10/5/169`.

[45] S. Mitra, N. Saxena, and E. McCluskey, “A design diversity metric and analysis of redundant systems,” *IEEE Transactions on Computers*, vol. 51, no. 5, pp. 498–510, 2002. DOI: `10.1109/TC.2002.1004589`.

[46] M. D. Lerner, “Algorithm based fault tolerance in massively parallel systems,” 1988.

[47] J. Rexford and N. Jha, “Algorithm-based fault tolerance for floating-point operations in massively parallel systems,” in *1992 IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. 2, 1992, 649–652 vol.2. DOI: `10.1109/ISCAS.1992.230168`.

[48] L. L. Pilla *et al.*, “Software-based hardening strategies for neutron sensitive fft algorithms on gpus,” *IEEE Transactions on Nuclear Science*, vol. 61, no. 4, pp. 1874–1880, 2014. DOI: `10.1109/TNS.2014.2301768`.

[49] V. L. Fèvre *et al.*, “A comparison of several fault-tolerance methods for the detection and correction of floating-point errors in matrix-matrix multiplication,” in *Euro-Par 2020: Parallel Processing Workshops*, Springer International Publishing, 2021, pp. 303–315, ISBN: 978-3-030-71593-9.

[50] C. Braun, S. Halder, and H. J. Wunderlich, “A-abft: Autonomous algorithm-based fault tolerance for matrix multiplications on graphics processing units,” in *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2014, pp. 443–454. DOI: `10.1109/DSN.2014.48`.

[51] M. Hsiao, “A class of optimal minimum odd-weight-column sec-ded codes,” *IBM Journal of Research and Development*, vol. 14, no. 4, pp. 395–401, 1970.

[52] D. Bossen, “B-adjacent error correction,” *IBM Journal of Research and Development*, vol. 14, no. 4, pp. 402–408, 1970.

[53] M. Hsiao, D. Bossen, and R. T. Chien, “Orthogonal latin square codes,” *IBM Journal of Research and Development*, vol. 14, no. 4, 1970.

[54] P. Reviriego, S. Pontarelli, A. Sánchez-Macián, and J. A. Maestro, “A method to extend orthogonal latin square codes,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 7, pp. 1635–1639, 2014. DOI: `10.1109/TVLSI.2013.2275036`.

[55] P. Novac O.and Otto, V. K. Stefan, and M. Vladutiu, “Implementation of a sec-ded code with fpga xilinx circuits to the cache level of a memory hierarchy,” *”, J. of Computer Science and Control Systems*, vol. 1, pp. 62–65, 2008.

[56] S. Eyerman and L. Eeckhout, “Fine-grained dvfs using on-chip regulators,” *ACM Transactions on Architecture and Code Optimization*, vol. 8, no. 1, pp. 1–24, 2011.

[57] B. Bowhill, B. Stackhouse, N. Nassif, *et al.*, “The xeon® processor e5-2600 v3: A 22 nm 18-core product family,” *IEEE Journal of Solid-State Circuits*, vol. 51, no. 1, pp. 92–104, 2016. DOI: `10.1109/JSSC.2015.2472598`.

[58] N. Sturcken, M. Petracca, S. Warren, *et al.*, “A switched-inductor integrated voltage regulator with nonlinear feedback and network-on-chip load in 45 nm soi,” *IEEE Journal of Solid-State Circuits*, vol. 47, no. 8, pp. 1935–1945, 2012. DOI: 10.1109/JSSC.2012.2196316.

[59] E. A. Burton, G. Schrom, F. Paillet, *et al.*, “Fivr — fully integrated voltage regulators on 4th generation intel® core™ socs,” in *2014 IEEE Applied Power Electronics Conference and Exposition - APEC 2014*, 2014, pp. 432–439. DOI: 10.1109/APEC.2014.6803344.

[60] L. Chen and S. Dey, “Software-based self-testing methodology for processor cores,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 3, pp. 369–380, 2001.

[61] P. Bernardi *et al.*, “Development flow for on-line core self-test of automotive microcontrollers,” *IEEE Transactions on Computers*, vol. 65, no. 3, pp. 744–754, 2016.

[62] A. Riefert *et al.*, “A flexible framework for the automatic generation of sbst programs,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 10, pp. 3055–3066, 2016. DOI: 10.1109/TVLSI.2016.2538800.

[63] A. Paschalis and D. Gizopoulos, “Effective software-based self-test strategies for on-line periodic testing of embedded processors,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 1, pp. 88–99, 2005. DOI: 10.1109/TCAD.2004.839486.

[64] P. Bernardi *et al.*, “On-line software-based self-test of the address calculation unit in risc processors,” in *2012 17th IEEE European Test Symposium (ETS)*, 2012, pp. 1–6. DOI: 10.1109/ETS.2012.6233004.

[65] D. Sabena *et al.*, “A new sbst algorithm for testing the register file of vliw processors,” in *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2012, pp. 412–417. DOI: 10.1109/DATE.2012.6176506.

[66] D. Sabena and other, “On the automatic generation of optimized software-based self-test programs for vliw processors,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 4, pp. 813–823, 2014. DOI: 10.1109/TVLSI.2013.2252636.

[67] T. Faller *et al.*, “Towards sat-based sbst generation for risc-v cores,” in *2021 IEEE 22nd Latin American Test Symposium (LATS)*, 2021, pp. 1–2. DOI: 10.1109/LATS53581.2021.9651819.

[68] P. Bernardi *et al.*, “Software-based self-test techniques of computational modules in dual issue embedded processors,” in *2015 20th IEEE European Test Symposium (ETS)*, 2015, pp. 1–2. DOI: 10.1109/ETS.2015.7138730.

[69] J. Hudec, “An efficient adaptive method of software-based self test generation for risc processors,” in *2015 4th Eastern European Regional Conference on the Engineering of Computer Based Systems*, 2015, pp. 119–121. DOI: 10.1109/ECBS-EERC.2015.26.

[70] J.-D. Guerrero-Balaguera *et al.*, “On the functional test of special function units in gpus,” in *2021 24th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*, 2021, pp. 81–86. DOI: 10.1109/DDECS52668.2021.9417025.

[71] J. E. Rodriguez Condia *et al.*, “Using stls for effective in-field test of gpus,” *IEEE Design & Test*, vol. 40, no. 2, pp. 109–117, 2023. DOI: 10.1109/MDAT.2022.3188573.

[72] J. E. R. Condia and M. Sonza Reorda, “Testing permanent faults in pipeline registers of gpgpus: A multi-kernel approach,” in *2019 IEEE 25th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2019, pp. 97–102. DOI: 10.1109/IOLTS.2019.8854463.

[73] J. E. R. Condia and M. S. Reorda, “On the testing of special memories in gpgpus,” in *2020 IEEE 26th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2020, pp. 1–6. DOI: 10.1109/IOLTS50870.2020.9159711.

[74] J. E. R. Condia and M. S. Reorda, “Testing the divergence stack memory on gpgpus: A modular in-field test strategy,” in *2020 IFIP/IEEE 28th International Conference on Very Large Scale Integration (VLSI-SOC)*, 2020, pp. 153–158. DOI: 10.1109/VLSI-SOC46417.2020.9344088.

[75] B. Du *et al.*, "About the functional test of the gpgpu scheduler," in *2018 IEEE 24th International Symposium on On-Line Testing And Robust System Design (IOLTS)*, 2018, pp. 85–90. DOI: `10.1109/IOLTS.2018.8474174`.

[76] S. d. Di Carlo *et al.*, "On the in-field test of the gpgpu scheduler memory," in *2019 IEEE 22nd International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*, 2019, pp. 1–6. DOI: `10.1109/DDECS.2019.8724672`.

[77] S. Di Carlo *et al.*, "An on-line testing technique for the scheduler memory of a gpgpu," *IEEE Access*, vol. 8, pp. 16 893–16 912, 2020. DOI: `10.1109/ACCESS.2020.2968139`.

[78] J.-D. Guerrero-Balaguera *et al.*, "A new method to generate software test libraries for in-field gpu testing resorting to high-level languages," in *2022 IEEE 40th VLSI Test Symposium (VTS)*, 2022, pp. 1–7. DOI: `10.1109/VTS52500.2021.9794225`.

[79] J.-D. Guerrero-Balaguera *et al.*, "Stls for gpus: Using high-level language approaches," *IEEE Design & Test*, pp. 1–1, 2023. DOI: `10.1109/MDAT.2023.3267601`.

[80] N. I. Deligiannis *et al.*, "Automating the Generation of Programs Maximizing the Sustained Switching Activity in Microprocessor units via Evolutionary Techniques," *Microprocessors and Microsystems*, vol. 98, 2023.

[81] N. I. Deligiannis *et al.*, "Automating the Generation of Programs Maximizing the Repeatable Constant Switching Activity in Microprocessor Units via MaxSAT," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023.

[82] V. G. V. Larrea *et al.*, "Towards acceptance testing at the exascale frontier," in *Proceedings of the Cray User Group 2020 conference*, 2020.

[83] C. Hay, "Multicore and distributed processing with tetramax® atpg," 2009.

[84] M. Vannan, "Test coverage analysis of partial scan socs using atpg and fault simulation of functional vectors using tetramax: A case study,"

[85] E. Sanchez, M. Schillaci, and G. Squillero, *Evolutionary Optimization: the μGP toolkit*. Springer Science & Business Media, 2011.

[86] Synopsys, *Z01x functional safety assurance product*, [online] Available: https://www.synopsys.com/verification/ simulation/z01x-functional-safety.html, Feb. 2023.

[87] O. Goloubeva, M. Rebaudengo, M. S. Reorda, and M. Violante, *Software-implemented hardware fault tolerance*. Springer Science & Business Media, 2006.

[88] M. Bohman, B. James, M. J. Wirthlin, H. Quinn, and J. Goeders, "Microcontroller compiler-assisted software fault tolerance," *IEEE Transactions on Nuclear Science*, vol. 66, no. 1, pp. 223–232, 2019. DOI: `10.1109/TNS.2018.2886094`.

[89] P. H. Hargrove and J. C. Duell, "Berkeley lab checkpoint/restart (blcr) for linux clusters," in *Journal of Physics: Conference Series*, IOP Publishing, vol. 46, 2006, p. 494.

[90] W. Bland, A. Bouteiller, T. Herault, J. Hursey, G. Bosilca, and J. J. Dongarra, "An evaluation of user-level failure mitigation support in mpi.," *EuroMPI*, vol. 12, pp. 193–203, 2012.

[91] F. Reghenzani, G. Pozzi, G. Massari, S. Libutti, and W. Fornaciari, "The mig framework: Enabling transparent process migration in open mpi," in *Proceedings of the 23rd European MPI Users' Group Meeting*, 2016, pp. 64–73.

[92] P. Bellasi, G. Massari, and W. Fornaciari, "Effective runtime resource management using linux control groups with the barbequertrm framework," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 14, no. 2, pp. 1–17, 2015.

[93] F. Reghenzani and W. Fornaciari, "Mixed-criticality with integer multiple wcets and dropping relations: New scheduling challenges," in *Proceedings of the 28th Asia and South Pacific Design Automation Conference*, ser. ASPDAC '23, Tokyo, Japan: Association for Computing Machinery, 2023, pp. 320–325, ISBN: 9781450397834. DOI: `10.1145/3566097.3567851`. [Online]. Available: `https://doi.org/10.1145/3566097.3567851`.

[94] F. Reghenzani, Z. Guo, L. Santinelli, and W. Fornaciari, "A mixed-criticality approach to fault tolerance: Integrating schedulability and failure requirements," in *2022 IEEE 28th Real-Time and Embed-*

ded Technology and Applications Symposium (RTAS), 2022, pp. 27–39. DOI: 10.1109/RTAS54340.2022.00011.

[95] F. Reghenzani, G. Massari, W. Fornaciari, *et al.*, "Chronovise: Measurement-based probabilistic timing analysis framework," *Journal of Open Source Software*, vol. 3, pp. 711–713, 2018.

[96] F. Reghenzani, Z. Guo, and W. Fornaciari, "Software fault tolerance in real-time systems: Identifying the future research questions," *ACM Comput. Surv.*, Mar. 2023, Just Accepted, ISSN: 0360-0300. DOI: 10.1145/3589950. [Online]. Available: https://doi.org/10.1145/3589950.

[97] A. Morari, C. Boneti, F. J. Cazorla, *et al.*, "Smt malleability in ibm power5 and power6 processors," *IEEE Transactions on Computers*, vol. 62, no. 4, pp. 813–826, 2012.

[98] D. M. Tullsen, S. J. Eggers, J. S. Emer, H. M. Levy, J. L. Lo, and R. L. Stamm, "Exploiting choice: Instruction fetch and issue on an implementable simultaneous multithreading processor," in *Proceedings of the 23rd annual international symposium on Computer architecture*, 1996, pp. 191–202.

[99] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, "Last-level cache side-channel attacks are practical," in *2015 IEEE symposium on security and privacy*, IEEE, 2015, pp. 605–622.

[100] D. Evtyushkin, D. Ponomarev, and N. Abu-Ghazaleh, "Jump over aslr: Attacking branch predictors to bypass aslr," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, IEEE, 2016, pp. 1–13.

[101] S. Van Schaik, A. Milburn, S. Österlund, *et al.*, "Ridl: Rogue in-flight data load," in *2019 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2019, pp. 88–105.

[102] P. Kocher, J. Horn, A. Fogh, *et al.*, "Spectre attacks: Exploiting speculative execution," *Communications of the ACM*, vol. 63, no. 7, pp. 93–101, 2020.

[103] M. Lipp, M. Schwarz, D. Gruss, *et al.*, "Meltdown: Reading kernel memory from user space," *Communications of the ACM*, vol. 63, no. 6, pp. 46–56, 2020.

[104] M. Danelutto, G. Mencagli, M. Torquati, H. González–Vélez, and P. Kilpatrick, "Algorithmic skeletons and parallel design patterns in mainstream parallel programming," *International Journal of Parallel Programming*, vol. 49, no. 2, pp. 177–198, 2021. DOI: 10.1007/s10766-020-00684-w. [Online]. Available: https://doi.org/10.1007/s10766-020-00684-w.

[105] H. González-Vélez and M. Leyton, "A survey of algorithmic skeleton frameworks: High-level structured parallel programming enablers," *Softw. Pract. Exper.*, vol. 40, no. 12, pp. 1135–1160, Nov. 2010, ISSN: 0038-0644.

[106] M. Danelutto, G. Mencagli, M. Torquati, H. González-Vélez, and P. Kilpatrick, "Algorithmic skeletons and parallel design patterns in mainstream parallel programming," *Int. J. Parallel Program.*, vol. 49, no. 2, pp. 177–198, 2021. DOI: 10.1007/s10766-020-00684-w. [Online]. Available: https://doi.org/10.1007/s10766-020-00684-w.

[107] M. Aldinucci, M. Danelutto, P. Kilpatrick, and M. Torquati, "Fastflow: High-level and efficient streaming on multicore," in *Programming multi-core and many-core computing systems*. John Wiley & Sons, Ltd, 2017, ch. 13, pp. 261–280, ISBN: 9781119332015. DOI: https://doi.org/10.1002/9781119332015.ch13. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781119332015.ch13. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119332015.ch13.

[108] G. Mencagli, M. Torquati, A. Cardaci, A. Fais, L. Rinaldi, and M. Danelutto, "Windflow: High-speed continuous stream processing with parallel building blocks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 11, pp. 2748–2763, 2021. DOI: 10.1109/TPDS.2021.3073970.

[109] D. D. Sensi, M. Torquati, and M. Danelutto, "Mammut: High-level management of system knobs and sensors," *SoftwareX*, vol. 6, pp. 150–154, 2017, ISSN: 2352-7110. DOI: https://doi.org/10.1016/j.softx.2017.06.005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352711017300225.

[110] F. Fraternali, A. Bartolini, C. Cavazzoni, G. Tecchiolli, and L. Benini, "Quantifying the impact of variability on the energy efficiency for a next-generation ultra-green supercomputer," *2014 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 2014.

[111] F. Fraternali, A. Bartolini, C. Cavazzoni, and L. Benini, “Quantifying the impact of variability and heterogeneity on the energy efficiency for a next-generation ultra-green supercomputer,” *IEEE Transactions on Parallel and Distributed Systems*, 2018.

[112] B. Rountree, D. K. Lownenthal, B. R. de Supinski, M. Schulz, V. W. Freeh, and T. Bletsch, “Adagio: Making dvs practical for complex hpc applications,” *Proceedings of the 23rd International Conference on Supercomputing*, 2009.

[113] D. Li, B. R. de Supinski, M. Schulz, K. Cameron, and D. S. Nikolopoulos, “Hybrid mpi/ openmp power-aware computing,” *2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS)*, 2010.

[114] D. Cesarini, A. Bartolini, P. Bonfa, C. Cavazzoni, and L. Benini, “Countdown: A run-time library for performance-neutral energy saving in mpi applications,” *IEEE TRANSACTIONS ON COMPUTERS*, 2021.

[115] D. Cesarini, A. Bartolini, A. Borghesi, C. Cavazzoni, M. Luisier, and L. Benini, “Countdown slack: A run-time library to reduce energy footprint in large-scale mpi applications,” *IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS*, 2020.

[116] S. Williams, A. Waterman, and D. Patterson, “Roofline: An insightful visual performance model for multicore architectures,” *Communications of the ACM*, 2009.