MISSIONE 4
ISTRUZIONE
RICERCA

SPOKE 1

FUTURE HPC & BIG DATA

FLAGSHIP 1:
Survey of State-of-the-art Approaches and Gap Analysis of RISC-V Platform Requirement for HPC

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 overviews the state-of-the-art and provides a gap analysis about RISC-V platforms to be deployed and used in HPC systems, according to the main objectives described in the milestone #4, Spoke 1 - Flagship 1.

Chapter 1 briefly introduces the main purposes of HPC systems, their relevance in modern applications, and describes the crucial aspects of RISC-V platforms in developing current and future exascale HPC systems. Moreover, the Chapter highlights the main features and possible challenges of their integration into modern HPC machines and the market.

Chapter 2 overviews the current status of HPC machines and discusses the two principal challenges (*efficiency* and *performance*) of modern and future generations of HPC machines.

Chapter 3 surveys the major efforts and advances when considering RISC-V-based platforms to be adopted in HPC machines. In particular, the analysis focuses on the architectural requirements and needs of RISC-V platforms for mature adoption and deployment in HPC systems. The Chapter also shows the main advantages, issues, and possible challenges in their deployment in HPC systems. This Chapter is divided into three subsections to show the main aspects of RISC-V core/cluster architectures, hardware accelerator trends and support, and the main features of the memory hierarchy and interconnect networks.

Chapter 4 overviews the main solutions for modeling non-functional properties in HPC systems and highlights the main challenges and open questions in the HPC domain, especially focusing on deploying RISC-V platforms. This Chapter comprises four subsections that survey the main challenges regarding reliability, thermal and power supply monitoring, energy efficiency, and performance.

Chapter 5 overviews several challenges and gaps in software stack support in current and future HPC generations, focusing on run-time management and compiler support and software assistance of HPC systems to emerging applications, such as decentralized machine learning workloads.

Finally, Chapter 6 provides concluding remarks and emphasizes the main research opportunities and open questions.

AUTHORS (in alphabetical order):

- Giovanni Agosta, Politecnico di Milano
- Enrico Bini, Università degli studi di Torino
- Robert Birke, Università degli studi di Torino
- Daniele Cattaneo, Politecnico di Milano
- Daniele Cesarini, CINECA
- Iacopo Colonnelli, Università degli studi di Torino
- Stefano Di Carlo, Politecnico di Torino
- William Fornaciari, Politecnico di Milano
- Andrea Galimberti, Politecnico di Milano
- Alberto Garfagnini, Università degli studi di Padova
- Gabriele Magnani, Politecnico di Milano
- Marco Lapegna, Università degli studi di Napoli Federico II
- Doriana Medić, Università degli studi di Torino
- Gabriele Mencagli, Università di Pisa
- Cecilia Metra, Università degli studi di Bologna
- Gianluca Mittone, Università degli studi di Torino
- 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
- Federico Tesser, CINECA
- Federico Terraneo, Politecnico di Milano
- 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

High-Performance Computing (HPC) systems have evolved to be used in domains where physical experimentation is prohibitively impractical (e.g., financial disaster modeling, deep space experimental models) [1], [2], expensive (e.g., weather forecast) [3]–[5], or dangerous (e.g., atomic, biological, and chemical interactions) [6], [7]. Furthermore, boosted by the pervasive globalization and digitalization of modern society, HPC machines are now strategic assets for academia, industry, public institutions, and governments [8] in the support and development of crucial fields (i.e., from scientific leadership, and economic prosperity up to national security) and strategic economic and technological trends, including the deployment and training of massive artificial intelligence (AI) tools, the modeling and simulation of complex algorithms, the use in the industrial domain (e.g., designing and operating digital twins and cloud services).

In general, the application domains of HPC systems are characterized by their huge complexity (i.e., complex algorithms and massive data) and the demand for effective and extensive computational power resources to perform increasingly accurate and complex simulations within acceptable time frames [9]. Modern HPC machines exploit distributed computing strategies (partly inherited from the mainframe times), in combination with smart co-design techniques to effectively integrate and correctly operate hardware platforms and software frameworks, prioritizing the operational throughput and performance of the complete system [10], so aiming to achieve their nominal computational power and acceptable levels of performance efficiently (i.e., in terms of operations per watt) [11].

Furthermore, influenced by the market, the industry, supported by the academia, aggressively progresses on three crucial features of modern and next-generation HPC machines: *1)* the operational performance, *2)* the throughput, and *3)* the energy efficiency. The first two features (*performance* and *throughput*) are directly related to the computing capabilities of the HPC machines. An overview of the deployed HPC machines shows that most modern and advanced systems (from 1997 until now) exploit the effective use and efficient management of distributed commodity clusters with special equipment composed of multi-core processors, special-purpose hardware accelerators (i.e., GPUs, APUs, TPUs, DPUs, or DLPs), specialized interconnect systems, and memory storage [12]. Most HPC designs are based on clusters of high-end cores from several leading companies in the semiconductors and microprocessor domain (e.g., Intel Cascade lake, Intel Skylake, AMD EPYC, AMD Zen-2, AMD Zen-3, NEC Vector Engine, or Fujitsu ARM) to provide petascale and exascale computing capabilities.

Regarding energy and power efficiency, several strategies for HPC machines have been investigated in recent years involving improvements in the integration technologies (e.g., development of low-power cores) and the management of the software and hardware layers for HPC systems. Some efforts, such as the Mont Blanc 2020 project[13]–[15] promoted and evaluated the development and use of low-power ARM-based System-on-Chip (SoC) architectures for the HPC market by resorting to mature low-power ARM ISA architectures and taking advantage of the dynamic and already available development and programming ecosystem. Other initiatives, such as DEEP-ER, DEEP-SEA, and RED-SEA [16] projects, promote research activities focused on specific features of current and next generations of exascale HPC systems. On the one hand, DEEP-ER addressed two significant challenges for exascale HPC machines: the speed bottleneck between I/O bandwidth and core infrastructures and the development of resiliency mechanisms to recover corrupted tasks by hardware faults. In addition, DEEP-SEA focuses on developing efficient software stack (from low-level drivers to resource management and programming abstractions) environments for heterogeneous HPC systems. Moreover, RED-SEA aims to extend and adapt BXI interconnect technologies for the new generations of hybrid and heterogeneous HPC machines.

A considerable amount of collaborative efforts (between industry and academia), national and regional

initiatives (e.g., the European Processor Initiative, or EPI [17]) impulse and promote research activities and innovations towards the democratization, development, adoption, and integration of open-sources architectures (i.e., RISC-V) for new HPC machines, boosting the available access to the resources and the partial availability of framework ecosystems for its development. In the European context, the EPI initiative is one of the cornerstones of the EuroHPC Joint Undertaking, which focuses on pooling the Union's and national resources to deploy and develop the most powerful supercomputers within Europe, advise and suggest the adoption of open-source architectures (with RISC-V as a main core solution) for the next generation of exascale embedded HPC platforms [18]. Similarly, initiatives, such as the TEXTAROSSA Project aims to achieve high performance and high energy efficiency on near-future exascale computing systems, specially targeting the efficiency increase of computation with modern HW and new arithmetics, as well as providing methods and tools for seamless integration of reconfigurable accelerators in heterogeneous HPC multi-node platforms [19].

Another justification for selecting RISC-V architectures is that open-source hardware architectures promote research and advances without economic and licensing restrictions and are less affected by external scenarios. In this case, the global and neutral not-for-profit support for the RISC-V development by the academia, the industry, and a foundation with members worldwide assure technical support for the standardization of practices (i.e., ISA extensions), meanwhile providing freedom for custom deployment scenarios (e.g., targeting performance and throughput goals). Moreover, the organization maintains a high degree of neutrality concerning geopolitical tensions and possible implications in technology concerning other dominant and proprietary ISA technologies (e.g., x86 and ARM).

One of the key aspects of adopting RISC-V architectures lies in the efficiency improvements (i.e., power and energy consumption) of the open-source Instruction Set Architectures (ISAs), which is a major innovation in the field and aims to become a standard and universal ISA. Currently, the RISC-V architecture is extremely popular and has been successfully adopted in mainly three representative fields: small IoT devices, personal mobile devices, and warehouse-level computers [20] by many companies (Microchip [21], Google, Qualcomm, Imagination [22], Intel [23], SiFive [24], Greenwaves [25], among others). Furthermore, new start-ups and competitors (e.g., Tenstorrent [26], Ventana Micro Systems Inc. [27], and Alibaba [28]) are adapting the RISC-V philosophy into the Edge, cloud, and HPC domains. A notable academic effort is *'Monte Cimone'* [8] that paves the way towards RISC-V-based cluster commodities. This cluster is a test bed providing a first attempt to develop and evaluate the operational features of RISC-V-based cluster nodes. Moreover, its deployment also faced challenges in terms of shared and coordinate operations of multi-node and multi-cluster cores, as well as their interconnections. Furthermore, the deployment of this prototype supports the improvement, evaluation, analysis, and verification of storage mechanisms, and power monitoring infrastructures, which are highly required in the HPC domain to face Non-Functional Properties (NFP) issues, such as thermal and power monitoring or reliability features.

In most previous scenarios, the efficient evaluation of NFPs, such as dependability and thermal and power supply monitoring, are commonly neglected, since they do not directly impact some application fields. However, a large set of NFPs must be analyzed and evaluated in the HPC domain to guarantee the correct operation of their elements. Indeed, NFPs represent constraints a system faces to deliver its intended functionality or service (e.g., dependability, reliability, thermal monitoring, power budget, etc.) since the operation of one or more commodity clusters might be affected partially or permanently. The premature adoption of RISC-V platforms for the HPC domain involves several challenges in their structural design, interconnection, support for emerging architectural innovations, and domain-specific adaptations of hardware accelerators. Similarly, open questions arise when considering clever and efficient strategies and methods to analyze and evaluate non-functional properties. Both features (architecture and non-functional properties) must be investigated and solved before RISC-V-based commodity clusters start production and become available in the market.

This document summarizes the state-of-the-art of platform requirements for HPC and analyzes the main

research opportunities for RISC-V-based platforms focusing on the analysis of non-functional properties, including reliability, thermal and power budget monitoring, energy efficiency, performance, temporal analysis, and software stack support.

Chapter 2 describes the current status of machines and systems devoted to High-Performance Computing, particularly focusing on the two main challenges that accademia and industry are facing currently (*efficiency* and *performance*). Then, Chapter 3 overviews the most significant advances in RISC-V-based architectures and their adoption to HPC machines. A second Section overviews and discusses the main features and trends of modern architecture and design approaches of HPC computers, the adoption of hardware accelerators, as well as, their main challenges and research opportunities. Furthermore, a third Section discusses and overviews the main advantages and relevant issues in the memory hierarchy and interconnect technologies for technologies used for the integration of HPC systems. In addition, Chapter 4 introduces the main characteristics and challenges for modeling non-functional properties in HPC machines, mainly focusing on reliability, power consumption, energy efficiency, performance, and temporal properties. Chapter 5 analyzes several challenges and gaps in software stack support for current and future HPC generations, focusing on run-time management and compiler support and software assistance of HPC systems to emerging applications, such as decentralized machine learning. Finally, Chapter 6 provides concluding remarks and highlights the main research opportunities and open questions.

2 CURRENT HPC SCENARIO

It is widely acknowledged that the High Performance Computing (HPC) infrastructures play a strategic role in many research and industry fields, ranging from weather forecasting to material science. In our society, the digitalization has become more and more pervasive and high performance computers are at the base of disruptive trends, like the deployments of artificial intelligence (AI) on large scale (e.g., training and infer large machine learning models, like ChatGPT [29]) and the support to industrial use cases (e.g., to create and maintain digital twins). For this reason, HPC systems represent strategic assets, not only for research institutions and industries, but also for public entities and governments [30].

The key challenge in designing a new generation of HPC systems today and in the near future lies in increasing the energy efficiency to support the growing performance demand of applications in a sustainable power envelop in the era of the end of Dennard Scaling. While integrated circuits technology is still increasing density, power consumption does not scale at the same rate. Power density is today one of the main bottlenecks that can compromise performance when thermal design power limitations are reached.

The biggest source of power consumption in today's HPC system is related to the data movement [31]. Moving data is overtaking computation and is becoming the most dominant cost in terms of energy consumption and price. Future generations of HPC architectures will integrate compute and data on the same package to avoid costly off-chip memory accesses in terms of latency and energy.

The performance race of supercomputers in the last six decades, focused on achieving the highest figures in terms of Floating Point Operations per Seconds (FLOPS), is not more sustainable due to today's technological walls, which limit the peak performance of HPC systems. For this reason, supercomputers are mainly limited by memory performance, that degrades the computing capabilities on memory-intensive applications that represent most of the scientific HPC codes. Disruptive technologies, such as quantum or neuromorfic computing, may have an important impact in some specific application areas, but there is no silver bullet in sight.

To address the efficiency challenges of HPC systems, both academia and industry are actively pursuing architectural innovation and co-design strategies. These efforts aim to develop HPC systems that overcome efficiency limitations by utilizing various forms of specialization and domain-specific adaptation. Rapid evolution of Instruction Set Architectures (ISAs) is necessary to sustain architectural progress and domain adaptation. The emergence of the RISC-V ISA, which is open, extensible, and does not require royalties, has been a significant step towards accelerating innovation in this area. In addition, RISC-V offers an advantage over dominant proprietary ISAs such as x86 and ARM, as it is maintained by a global non-profit foundation with members from around the world. This ensures a high degree of neutrality with respect to geopolitical tensions and technology limitations associated with proprietary ISAs.

At present, the development of high-performance 64bit RISC-V processors and accelerator chips is underway. Various publications, including [32] and [8], have demonstrated promising prototypes and systems, while products have been announced at a rapid pace [33]. It is reasonable to expect that high-performance chips based on RISC-V will be available as production silicon within the next year. However, creating a RISC-V-based HPC system requires more than just high-performance RISC-V chips. Many experts believe that the RISC-V software stack and system platforms are still relatively immature, and will require several more years of development effort before full applications can be run and optimized on a RISC-V-based HPC system.

As we know, complex systems are more prone to failures compared to simple systems. In 2019, the Frontier supercomputer broke the exascale barrier and became the first HPC system to be ranked in the Top500 list

[11]. However, despite this achievement, the Frontier supercomputer is also facing reliability issues [34]. These issues primarily revolve around the system's stability when handling highly demanding workloads, particularly on the GPU accelerators that handle the majority of the system's processing load. With a mean time between failure of only a few hours, rather than days, it is clear that the resiliency of hardware and software components is crucial for providing stable services of an acceptable quality.

It is now widely accepted that technological scaling, as dictated by Moore's law, results in increased power consumption and thermally-bound computing systems. This issue is particularly pressing for supercomputers and data centers, which prioritize aggressive performance, integration density, and sustainable power budgets. To address this problem, modern GPU accelerators and many-core processors in supercomputer systems are equipped with thermal sensors and fine-grain power management support to modulate power consumption based on current operating conditions. However, this flexibility is not fully utilized, as operating systems in HPC installations prioritize maximum power to avoid performance imbalances across processing elements. Furthermore, there are several non-idealities in the thermal characteristics of accelerators and large many-core processors targeting the high-performance computing market, including thermal heterogeneity, thermal capacitance, and thermal noise. These non-idealities worsen the efficiency of built-in reactive controllers and make it challenging to find a stable, yet thermally safe, operating point.

Energy and power consumption are significant concerns for modern supercomputers and are expected to be limiting factors for future installations. The high power density can negatively impact the performance, and the total power consumption requires additional power for cooling. All of these factors affect the total cost of ownership and operational costs, which can limit the budget for increasing supercomputer capacity. Therefore, addressing the energy and power wall is crucial for achieving planned performance growth in next generation of supercomputers. Similarly, the majority of HPC applications only utilize a small percentage of the potential performance available on modern supercomputers. The increasing complexity and dimensionality of emerging heterogeneous supercomputing nodes make performance optimization even more challenging. Optimization involves not only identifying code segments that are bottlenecks, but also determining the underlying causes and restructuring the code to improve performance. While simple timers can be used to identify potential bottlenecks, more sophisticated measurements such as hardware performance counters are needed to characterize the causes of the bottleneck. Diagnosing performance issues therefore requires extensive knowledge of the hardware, compiler, and system software. However, most HPC application developers are domain experts and should not be expected to have a detailed understanding of every system on which they run their code. As a result, minimizing and characterizing performance bottlenecks on today's HPC systems using performance tools is a challenging and time-consuming task for most application developers.

In conclusion, HPC systems play a crucial role in many research and industry fields, including the deployment of artificial intelligence and the support for industrial use cases. However, the main challenge for designing new HPC systems lies in increasing energy efficiency while sustaining the growing demand for performance. The data movement is now the biggest source of power consumption in today's HPC systems. At the same time, the most recent semiconductor technologies used to build HPC devices, as well as their complexity, make it increasingly hard to guarantee the reliability of HPC systems. To overcome these challenges, academia and industry are actively pursuing architectural innovation and co-design strategies. The emergence of the RISC-V ISA offers a significant step towards accelerating innovation in this area. However, creating a RISC-V-based HPC system requires more than just high-performance RISC-V chips, as the software stack and system platforms are still relatively immature. Furthermore, energy and power consumption are significant concerns for modern supercomputers, and addressing the energy and power wall is crucial for achieving planned performance growth.

3 OPEN SOURCE HARDWARE AND DESIGN TECHNIQUES

This Chapter reviews and analyze the main trends in the industry and academy for the design of processor-based systems for the HPC domain. The overview includes the current scenarios of RISC-V architecture design, the current design trends of hardware accelerators, and interconnections systems. Moreover, A set of opportunities are identified to improve the development for current and future generations of HPC machines.

3.1 RISC-V

RISC-V processors have been garnering attention in the high-performance computing industry in recent years. This open, extensible, and royalty-free ISA is gaining traction as a potential alternative to proprietary ISAs (x86 and ARM). RISC-V offers several advantages for HPC systems, including improved power efficiency, scalability, and flexibility.

One of the key features of RISC-V is its modularity, which allows for greater flexibility in design [33]. The ISA is composed of a base set of instructions, with optional extension modules that can be added to customize the processor for specific tasks. This modularity enables system designers to tailor the processor to their specific requirements, rather than being limited by the fixed set of instructions offered by proprietary ISAs. Additionally, RISC-V's open and collaborative nature enables community-driven innovation and development. This allows for a wider range of ideas and perspectives to be brought to the table, potentially leading to faster progress and greater innovation in the HPC ecosystem.

With the continued growth of HPC systems, energy consumption has become a major concern. The RISC-V ISA is designed to be highly configurable, allowing designers to select the specific instructions and extensions that are most relevant to their application or workload. This modularity and extensibility enable RISC-V processors to be highly optimized for specific tasks, resulting in better performance and energy efficiency. Furthermore, RISC-V's modular design allows for the implementation of specialized hardware accelerators, which can extremely increase energy efficiency by offloading compute-intensive tasks from the processor.

While RISC-V shows promise as a potential alternative to proprietary ISAs, there are also challenges to be addressed. One of the main concerns is the lack of a mature software ecosystem. While there are existing software tools and libraries available for RISC-V, they are still in the early stages of development [8]. This means that porting existing HPC applications to RISC-V may require significant effort and resources.

Another challenge is the lack of a standardized memory hierarchy. RISC-V does not specify a standard memory hierarchy, leaving this to be determined by system designers. While this flexibility can be an advantage for embedded systems, it can also make it more challenging to design and optimize memory systems for HPC systems.

Despite these challenges, there is a growing interest in using RISC-V for HPC applications. The potential benefits of RISC-V, such as modularity, energy efficiency, and collaborative development, make it an attractive option for HPC systems. As the software ecosystem and tooling continue to develop, the journey of RISC-V processors to become a part of the HPC ecosystem is only in its initial stages.

3.2 Accelerators

Hardware accelerators, or *Domain-Specific Architectures* (DSAs), are special-purpose hardware structures designed to boost the performance of complex operations in highly demanding tasks. Indeed, the design efforts of the academy and industry promote hardware accelerators as solutions to overcome the challenges in performance, power, and parallelism (i.e., Moore's Law and Dennard scaling) of traditional general-purpose CPU-based systems [32]. Modern hardware accelerators efficiently exploit parallelism and data handling through specialized architectures and memory management strategies to increase performance and throughput (several operations performed per cycle), so impulsing their adoption in modern and future generations of HPC machines (used in around 35.8% of the top 500 HPC systems worldwide [12]). In fact, accelerators, such as GPUs and FPGAs, are essential components in the architecture of modern HPCs due to the cost-effective benefits ratio for complex workloads (in comparison to CPU-only commodity clusters). An analysis reveals that the main accelerator architectures (most abundant in the market and academy) are systolic-arrays cores (e.g., TPUs), vector processors and their variations (e.g., VLIW, GPUs [35]) that provide the best trade-off among costs, performance, and power consumption [36]. Moreover, the same analysis suggests that modern accelerators must include trans-precision capabilities (different operative numeric formats and data sizes), since several applications can tolerate lower levels of precision in their computation. For example, modern machine learning DSAs include support for 8INT, 8FP, 16FP, and 32FP with limited support for 16INT or fixed points formats. Similarly, in this field, no accelerators support 64FP, which can be considered for specific domains, only. However, in other domains, such as HPC, supporting 64FP formats might be mandatory. Interestingly, initial efforts have been performed to explore and integrate emerging numeric formats, such as Posits and logarithmic, into the HPC domain [37]. However, it remains a vastly unexplored field. As a preliminary conclusion, current accelerator design trends focus on parallel architectures and trans-precision features, including extensive explorations in the open-source hardware domain.

In the HPCs, hardware accelerators can be divided into two main classes: *i)* discrete components inside commodity clusters, and *ii)* on-chip co-processors. The former class refers to discrete, fully specialized, massive accelerator devices connected to the host CPUs through intelligent and high-speed bus interfaces; meanwhile, the latter class refers to co-processors and accelerators placed on-chip (inside the die) with the host processors to support the execution of some specific tasks at the host level.

Discrete accelerators, such as those exploiting parallel architectures (GPUs, FPGAs, and TPUs), are particularly useful and play a crucial role in the HPC domain since they can perform many parallel computations simultaneously and are crucial in several fields, including big-data analysis and machine learning. More in detail, GPUs are the leading discrete accelerators employed in most advanced HPC systems, see Table 1. Furthermore, GPUs have a prominent role as accelerators in modern generations of HPC machines, which are no longer limited to graphics applications but extended as general-purpose data-parallel accelerators, and use their software flexibility for fast deployment and adaptation to specific tasks. On the other hand, FPGAs exploit hardware reconfigurability to implement specific structures and process complex tasks. Other accelerators (e.g., TPUs, APUs, DLPs, and DPUs) offer specific features for focused tasks and take advantage of several binding domains, such as deep learning and finance applications.

On the other hand, on-chip accelerators directly interact with one or more processors to increase the performance of specific operations. Industry and academia have made a notable effort to promote accelerator development, taking advantage of open-hardware approaches, such as RISC-V architectures, and developing and supporting hybrid and specific ISA extensions for long-vector and SIMD paradigms (e.g., *RISC-V Vector extension*). In [38], the authors introduced a new vector accelerator targeting inference operations for machine learning by resorting to programmable units allowing SIMD operations. Other variations include Vector and Systolic array architectures (VAS) that exploit cluster cores to process vector-like instructions in spatial structural organizations. Similarly, authors in [39] explored the design and integration of clusters of Digital-Signal-Processing cores (DSPs), exploiting SIMD paradigms to support the processing of appli-

Table 1: Leading discrete accelerators used in HPC machines. Adopted from [12].

Accelerator	In TOP 500 HPC systems	System Share (%)	Rmax (TFlops)	Rpeak (TFlops)	Cores
NVIDIA Tesla V100	68	13.6	226,796	443,631	4,688,680
NVIDIA A100	22	4.4	264,775	401,042	2,606,176
NVIDIA A100 SXM4 40 GB	16	3.2	167,209	229,444	1,693,672
NVIDIA Tesla V100 SXM2	11	2.2	90,370	180,163	2,031,440
NVIDIA A100 80GB	10	2	123,156	163,208	1,044,800
NVIDIA A100 40GB	10	2	62,683	101,052	660,740
AMD Instinct MI250X	9	1.8	1,525,179	2,261,385	11,713,152
NVIDIA Tesla P100	6	1.2	44,731	65,634	905,280
NVIDIA Volta GV100	4	0.8	269,439	362,565	4,408,096
NVIDIA A100 SXM4 80 GB	4	0.8	25,696	27,245	206,112
NVIDIA A100 SXM4 64 GB	2	0.4	178,205	260,734	1,488,672
NVIDIA Tesla K40	2	0.4	7,154	12,264	145,600
NVIDIA Tesla K40m	1	0.2	2,478	4,947	64,384
NVIDIA Tesla K40/Intel Xeon Phi 7120P	1	0.2	3,126	5,610	152,692
NVIDIA Tesla P100 NVLink	1	0.2	8,125	12,127	135,828
NVIDIA H100 80GB PCIe	1	0.2	2,038	5,417	5,920
Preferred Networks MN-Core	1	0.2	2,180	3,348	1,664
Nvidia Volta V100	1	0.2	21,640	29,354	347,776
NVIDIA Tesla K80	1	0.2	2,592	3,799	66,000
Intel Xeon Phi 31S1P	1	0.2	2,071	3,075	174,720
Deep Computing Processor	1	0.2	4,325	6,134	163,840
Intel Xeon Phi 5110P	1	0.2	2,539	3,388	194,616
NVIDIA Tesla K20x	1	0.2	3,188	4,605	72,000
Matrix-2000	1	0.2	61,445	100,679	4,981,760
PEZY-SC3	1	0.2	1,952	2,932	1,151,360

cations for the HPC domain (e.g., AI applications). In [40], the authors proposed FAUST, a Floating-point co-processor based on a parallel deployment of cluster cores able to process individual tasks. Unfortunately, the accelerator requires a considerable percentage of SoC area (around 40%). In [41], the authors proposed a RISC-V compliant modular floating-point unit (deployed in parallel as independent units) that combines SIMD paradigms with trans-precision capabilities obtaining a low power overhead (9% to 11%), but considerable area overheads (around 33%). The unit includes support for special operations (e.g., square root). Other cores, such as the *Kunlun* and *Kunlun II* are based on a coarse-grain re-configurable architecture (XPU), which is a combination of configurable and customized logic (for tensor and vector operations) and sets of cluster *"tiny"* cores for scalar operations exploiting SIMD paradigms. This accelerator provides comparable performance to discrete designs in terms of AI computations and can be adapted in HPC domains [42].

Other modern accelerators architectures include variations to long-vector approaches (VPUs) with ISA extension support (e.g., *RISC-VV long-vector*) to allow the development and adoption of long-vector-based hardware accelerators in the open-hardware community since manufacturers promote similar architectures, e.g., *Intel*, *NEC*, and *Fujitsu*. Most of these accelerators efficiently combine several Scalar Processing Units (SPUs) with a few vector processing cores (VPUs), and other specific purpose accelerators, such as FMA vector cores, to compute several different operands in parallel[43]. In Principle, these architectures follow a VLIW philosophy and are more energy-efficient than classical SIMD. However, these require a high interaction and management from the compiler stack to allow kernel optimization and the efficient distribution of resources per instruction. The *'Vitruvius+'* [44] long-vector accelerator is a VPU RISC-V-compliant co-processor specially designed for the HPC domain, including Open Vector Interface (OVI) support for host units (scalar cores) and offering equivalent capabilities to main competitors in the market, such as the Andes NX27V [45], Alibaba T-Head Xuantie910 [46], and SiFive's X280[47].

GPU architectures, including RISC-V ISA extensions, have also been investigated in the open-hardware community. In [48], the author adapted the Single-Instruction Multiple-Thread (SIMT) philosophy (typical of NVIDIA GPU devices) and extended it into RISC-V-based general-purpose CPUs to provide GPU capabilities and operate several execution threads in parallel. In [49], the authors propose a RISC-V-compliant

GPU architecture based on clusters comprising arrays of functional cores. Each core includes several functional units to process individual threads in a SIMD manner. Moreover, each core includes a *Near-Memory* approach to hide latency.

Data-flow accelerators are based on spatial systolic arrays (e.g., Eyerisss [50], [51], or Softbrain[52]) and are effective for specific application domains, such as Deep learning, and might support highly demanded operations in modern HPC systems. For example, the *Intelligence Processing Cores* (or IPUs) comprise individual cores to process independent parallel program threads in the machine learning application but have demonstrated a promising future in the HPC domain[53].

Other application-specific open-source accelerators include the *Nvidia Deep Learning Accelerator* (NVDLA), a scalable, configurable DSA for machine-learning inference. Its ISA, software stack, and several implementation models are all open and can be adapted to HPC machines [54]. The Versatile Tensor Accelerator (VTA) is a programmable and customizable application-specific (deep learning) open-hardware accelerator using a RISC-like programming abstraction to describe operations at the tensor level. Its architecture is based on systolic arrays implementing the General Matrix Multiply (GEMM) algorithm [55]. Other accelerator variations include Custom Function Units (CFUs) [56] that are tightly coupled into the pipeline of a CPU and add new custom instructions (RISC-V ISA extensions) to complement the CPU's standard functions (such as arithmetic/logic operations). Moreover, the CFU adoption on high-programmable platforms (FPGAs) is supported by a rich ecosystem of inter-operable, app-optimized CFU cores and libraries and the straightforward development of app-optimized SoCs. Other emerging methods for designing hardware accelerators exploit new paradigms and technologies, such as Processing In-Memory (or PIMs).

In addition, several authors have proposed frameworks to face issues related to the linking and interconnection of accelerators to host CPU cores, simplifying their integration into CPUs and SoC designs. Authors in [57] proposed a framework to explore multi-RiSC-V cores and multi-cluster systems (Network-on-Chip, or NoC architectures). Similarly, in [58], the authors proposed a framework for the design exploration of many-core architecture based on RISC-V processors and custom accelerators. The communication structure employs a 2D mesh NoC. Authors in [59] proposed Andromeda, a framework for the design space exploration of cluster and many-core systems using RISC-V cores as main operative units. Authors in [60] proposed a configurable framework and platform for designing and developing heterogeneous systems based on 32 or 64-bit RISC-V cores, targeting the deployment in FPGAs. In [61], the authors proposed the ESP framework to quickly integrate particular purpose co-processors, accelerators, and central CPU units. The Chipyard framework [62] allows the interconnection and use of hardware accelerators for custom SoC design. The framework supports hardware IP units such as the Berkeley Out-of-Order Machine (BOOM)[63], NVDLA[54], the Ariane core[64], the Hwacha Vector-Fetch Architecture [65], DSP modules, domain-specific accelerators, memories modules, and peripherals.

A general overview of all current design trends of efficient DSAs suggests that spatial accelerators based on vector architectures (i.e., GPUs, long-vector), systolic arrays, and PIM cores provide higher throughput and are exceptionally efficient for specific operations and domains. Thus, it is clear that modern and future HPC machines require one or a constellation of more efficient hardware accelerators (on-chip and discrete) to improve performance and reduce power consumption. Similarly, the lack of flexible standards for on-chip connection infrastructures still requires further investigation, mainly when targeting the HPC domain.

Modern DSAs face several challenges grouped into three main aspects: *i)* memory wall, *ii)* reconfigurability, and *iii)* software support [36].

DSAs are well-known for performance improvement. Unfortunately, massive performance requires ingenious methods to feed the cores with enough data to provide acceptable throughput levels. The main challenges resort to memory bottlenecks (memory access latency and bandwidth), which aggravates considering the high parallelism of modern accelerators. Current solutions resort to structural strategies to link functional

cores with memory sources. In [66], the authors exploit memory-rich architectures to allocate as much memory as possible near the computing units. Another strategy (*Near-Memory Computing*) divides the memory resources, and it organizes sets of functional units with memory/registers banks, limiting memory movements (a standard solution in GPUs and vector processors). Emerging approaches, such as PIMs, involve integrating and fusing computational units and memory sources. In this case, new technologies are required for implementation and are still in the research stages. Although the previous solutions contribute to solving memory issues in modern accelerators, the constant need to increase processing capabilities suggests that clever solutions are still required, even more when considering clusters of processing cores, constellations of accelerators, and memories sharing information among the processing units with their implicit performance bottlenecks and coherency issues.

Current and future application scenarios demand improvements in the programmability, productivity, and adaptability of DSAs [36]. In the first case, the efficiency of DSAs can be improved by adapting their internal functions for each workload by using some levels of reconfigurability. Current generations of DSAs use fine-grain reconfigurability based on FPGAs. However, fine-grain levels might not always be required, so emerging approaches, such as coarse-grain architectures, might simplify the DSA configuration by handling arrays of cores and units while preserving the performance. Other options include DSA supporting fast reconfigurable logic as a valuable architectural component to provide custom functions. Similarly, another design approach includes *Self-Similar Accelerator Architectures* that hierarchically integrates accelerators to improve performance and handle the application complexity (i.e., TPUs in modern GPUs and intelligent engines in FPGAs).

One open question arises about the compiler and software stack support for modern and future generations of DSAs. The increasing complexity of applications and the DSAs architecture suggest that new efficient mechanisms for preserving performance can be required (e.g., speculative parallelism or streaming support for memory addressing in hardware), as well as approaches to reduce the programmer's effort to handle parallelism and undesirable effects during the compilation (e.g., transparent parallelizing compilers, efficiency programming ecosystems, and natural programming models in software), which are still missing in the open-source domain for HPC-like accelerators. Other open research areas include supporting unified virtual memory mechanisms and cache memory coherency at deep levels (mainly targeting high-bandwidth accelerators) to boost the performance of on-chip DSAs. In addition, handling and managing HPC infrastructures for several users impose challenges in virtualizing physical DSAs and cores of large commodity clusters comprising several accelerators. Some initial solutions include remote API management [67] (in discrete accelerators) and general scheduler controllers (for on-chip DSAs). All previous design challenges can be directly affected by non-functional properties, such as dependability and thermal and power budget issues.

3.3 Memory and Interconnect

The potentially imminent adoption of RISC-V based SoCs in the HPC sector is leveraged by its open standard and free instruction set architecture [68]–[72], together with the possibility to include application-specific functions in the design, that can be accessed by adding custom instructions in the standard RISC-V instruction set [71].

Therefore, it is expected that RISC-V based SoCs will be increasingly employed for a significant spectrum of possible applications, out of which stand highly autonomous systems (such as unmanned robots and vehicles), that are nowadays receiving huge economical investments [73], [74]. However, such highly autonomous systems interact with human beings, thus the need to guarantee their functional safety and reliability emerges. This need of course translates to all electronics that operates the system. In fact, electronic systems for highly autonomous applications have strong requirements in terms of reliability and functional safety, which have been formalized by several international standards (e.g., the ISO 26262 [75], the ISO/PAS 21448 [76], the

ANSI/RIA R15.06 [77], and other standards that are currently under development).

However, due to technology scaling, high performance SoCs possibly used to implement electronic systems have become more and more vulnerable to faults affecting its operation in the field [78]. Therefore, guaranteeing the required high levels of reliability and functional safety that are mandated for highly autonomous systems represents nowadays a great technological challenge, that has to be faced to enable a smarter world [79]. In particular, faults affecting high performance SoCs used within autonomous systems represent one of the main threats to system's reliable and safe operation in the field [75], [78], thus constituting one of the limits to their evolution to higher autonomy levels.

As for any other high performance SoC, also RISC-V based SoCs make a massive use of cache memories (up to 80% of the chip area), in order to eliminate the memory bottleneck effect, thus enabling significant performance increase [68]–[71]. Therefore, it is expected that among all possible faults, soft errors affecting cache memories will be of major concern for RISC-V based SoCs implemented by scaled technologies [80].

Traditionally, in order to increase the reliability of high performance SoCs in the field, Error Correcting Codes (ECCs) are adopted to protect cache memories against soft-errors [81]–[84]. The adoption of ECCs mandate the addition of proper encoding/decoding blocks to the cache memory array. Due to the limited area of these additional blocks compared to the cache array, the occurrence of permanent faults (i.e., mainly bridging faults) possibly affecting such additional blocks in the field is typically neglected. Therefore, the encoding/decoding blocks are typically not protected against possible faults affecting themselves. While this risk has been considered acceptable so far, this is no longer the case in the perspective of high performance SoCs to be used in highly autonomous systems (e.g., highly autonomous vehicles, robots, etc.), due to their strong requirements in terms of reliability and functional safety [75]. In fact, it can be expected that faults affecting the encoder/decoder blocks of ECCs may result in a mis-correction, even if the original word read from the cache was error-free. In this case, the decoder will produce an incorrect output word, that will be propagated throughout the system, thus compromising the SoC reliability, with a dramatic impact on system's functional safety.

In order to cope with this problem, some solutions have been presented in the literature to prevent the catastrophic consequences of permanent faults affecting ECC's encoding/decoding blocks [85]–[87], that can be adopted for SoCs employed for safety critical components of highly autonomous systems.

In particular, in [85], the Authors propose the adoption of differential EXOR gates to implement the Syndrome Generator inside the Decoder and the Encoder of ECCs based on parity check bits. The solution enables to detect faults affecting the Syndrome Generator and the Encoder, but it does not guarantee the detection of faults affecting all other blocks implementing the Decoder of the ECC (e.g., the Syndrome Decoder, the Corrector, etc.). In [86], [87], the Authors propose the adoption of normal checkers (i.e., of the kind described in [88], [89]) to make the encoding/decoding blocks self-checking with respect to internal faults. More in details, the information bits produced at the output of the Decoder are first re-encoded by using an additional encoding block, in order to regenerate the word read from the cache (received at the input of the Decoder). The regenerated word is subtracted (modulo 2) from the word read from the cache (at the Decoder input) in order to generate an error word. For the case of a fault-free Decoder, the error word resulting from such a subtraction should be equal to the word generated by the Syndrome Decoder block inside the Decoder. Based on this property, the Authors propose to detect the presence of faults affecting the encoding/decoding blocks of the ECCs, by using a two-rail code checker to compare the generated error word with the word at the Syndrome Decoder output [87]. The solution is effective in detecting faults affecting the encoding/decoding blocks of the considered ECCs. However, they imply a significant impact on performance (which may be over 100%, depending on the considered ECC), and they also require a non-negligible cost in terms of area and power overhead.

In order to fill the gap between the practical requirements and the state of the art regarding 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), There are several opportunities to analyze, at the electrical level, the effects of permanent faults (such as resistive bridgings and stuck-ats) possibly affecting the ECCs' encoding/decoding blocks during their operation in the field. In fact, it is also possible to introduce metrics to evaluate the risks of the considered faults' effects on functional safety, thus identifying the most critical faults. The performed analyses and metrics will enable to develop low-cost innovative approaches to detect, during the SoC in-field operation, the occurrence of those faults that can compromise system's functional safety, thus enabling the activation of possible recovery mechanisms to re-establish the SoC correct operation.

In addition to the above memory reliability issues, the interconnections of modern SoCs implemented with aggressively scaled technologies are also becoming increasingly prone to bridging/open and crosstalk faults [90]–[103].

In fact, as technology scales down, the area of wires is continuously shrinking, resulting in higher current densities and a consequently increased likelihood of electromigration phenomena, that may potentially give rise to bridging/open faults [90], [96]–[98]. In addition, inter-wire spacing in bus lines features a sensible decrease as technology scales down: wires are closer to each other, and their aspect ratio is reducing, being the wires relatively higher and thinner than with previous technologies [91]–[95]. Consequently, crosstalk is playing an increasingly dominant role as a bus speed limiting factor [89], [91], [104], [105]. If crosstalk affects bus lines of a synchronous system, incorrect data may be sampled by the flip-flops at the receiver side of the bus and propagated through the system, causing an incorrect SoC behavior [93].

As for bridging/open faults, some models have been presented in the literature to estimate the trend over time of bridging/open faults' resistance increase in wires effected by electromigration [90], [96], with the main goal to estimate the Mean-Time-To-Failure (MTTF) of wires. In addition, some solutions have been proposed to detect bridging/open faults during post-manufacturing testing, or during possible periodic test performed in the field by Logic-Built Self -Test (LBIST) structures, when the block being tested is idle or power-off [99]–[103].

In [96], the authors present a physics-based model to estimate electromigration in ICs' power supply networks. The model accounts for process, voltage, and temperature variations across the die, but may be computationally expensive if used to estimate electromigration in large complex SoCs, like modern RISK-V processors. In [90], the Authors show that the dynamics of stresses and atom fluxes in metal lines due to electromigration is the same as the dynamics of voltages and currents in certain RC circuits. Then, the paper presents a methodology that enables to model the wire resistance increase over time (due to electromigration) by simply simulating (at the electrical level) the corresponding equivalent RC circuit of the considered wire.

In [99], the Authors propose a simple model at the electrical level that enables to estimate the delay increase of combinational circuits affected by bridging/open faults. In [100], [101], the Authors extend the model presented in [li03] to account also for delay uncertainties caused by process parameter variations occurring during fabrication. In [102], [103], the Authors propose a testing methodology that enables to detect bridging/open faults with a reduced number of test patterns. The proposed methodology is based on the propagation of voltage pulses (with a given fixed duration) through the data-paths of the circuit being tested. As shown in the paper, in fault-free data-paths the pulses are propagated without attenuation, while they are attenuated (or not propagated at all) in data-paths affected by bridging/open faults, thus exposing the presence of the faults. Compared to alternative approaches aimed at detecting bridging faults, this approach requires a smaller number of test patters.

A limit of all above approaches for bridging/open faults' detection is that they cannot be performed while the SoC is performing useful operations in the field (i.e., while running some safety critical application of an autonomous systems). They can only be adopted to test the SoC periodically, when it is idle or power-off.

Therefore, these techniques cannot avoid the impact of bridging/open faults on SoCs' reliability, thus they cannot guarantee the levels of reliability required for the SoCs employed for safety critical applications.

In order to fill this gap in the state of the art, additional analyses and evaluations are required on the resistance increase over time of interconnections of modern SoCs (e.g., SoCs based on the RISC-V architecture) under different operating/stress conditions. It is expected that this analysis will enable to identify conditions that can be used/exploited to develop low cost approaches for the early detection of bridging/open faults in the field, before they can affect the correct operation of the SoC.

As for crosstalk faults, some techniques have been proposed in the literature to reduce the crosstalk-induced delay uncertainty in bus lines of high performance microprocessors, with the goal to guarantee signal integrity [89], [91]–[95], [104], [105]. Traditional approaches to reduce the effects of crosstalk are based on the insertion of repeaters, or shielding lines (e.g., those belonging to the power distribution network) between bus wires [104]. However, these techniques typically require non-negligible costs in terms of area overhead.

In [89], [105], the authors propose a low-cost and self-checking monitor that is able to detect late transitions at the outputs of bus lines. A possible limitation of this solution is that their effectiveness may be impacted by process parameter variations occurring during fabrication, as well as by aging phenomena (like Bias Temperature Instability, or BTI) affecting the propagation delay of buffers/repeaters of the bus lines during the SoCs lifetime.

In [91]–[93], the Authors propose approaches based on ECCs to correct incorrect data that, due to crosstalk, may be sampled by the flip-flops (FFs) at the receiver side of the bus lines. They employ the Dual Rail and the Hamming codes, and enable to correct single errors. However, these solutions require the addition of extra bus lines for the check bits, with a consequent non-negligible area increase. Additionally, due to the increased impact of crosstalk in modern SoCs, it is expected that more than single errors can be generated at the receiver side of the bus lines, which may be not correctable by these solutions.

In [94], the authors present a monitor to detect crosstalk faults affecting the interconnects of Field Programmable Gate-Arrays (FPGAs). The proposed detector requires low area overhead and features self-checking ability, but it may be not straightforwardly applicable to bus interconnects different from those of FPGAs (like those of SoCs based on the RISC-V architecture).

Finally, in [95] the authors propose a design strategy to reduce the propagation delay of bus lines at very low costs in terms of power consumption and power-delay product. However, as for traditional approaches for crosstalk reduction, the solution in [95] requires the insertion of additional repeaters in the bus lines, with possible consequent non-negligible area overhead increase. Therefore, as far as we are concerned, the solutions presented in the literature to mitigate the detrimental effects of crosstalk in bus interconnections may not be able to guarantee, for modern SoC implemented in scaled technologies, the required levels of reliability for safety critical applications.

In order to fill the gap in the state of the art, new analyses and evaluations would contribute to understand at the electrical level, the crosstalk effects in scaled interconnections of modern SoCs (e.g., SoCs based on the RISC-V architecture), considering also the impact of aging phenomena (such as Bias Temperature Instability –BTI) affecting the buffers/repeaters of the bus lines during the SoCs lifetime. Based on the results of such an analysis, it is expected that innovative and low-cost approaches for crosstalk detection will be devised.

3.4 Direct cooling techniques

Cooling electronic components using a closed-loop liquid circuit applied directly to or near the surface of the chip is not a new technology. This approach has been used in the past, mainly on mainframe or high-performance computing systems commonly found in supercomputer facilities. In recent years, cost-effective

versions of direct cooling using water have been developed and sold to the personal computer market, catering to customers who seek to maximize performance. Nowadays, a modified version of this technology is available for the commercial server market. Direct cooling provides a more efficient method of transferring heat from hot components to the building's chilled water loop and then outside with minimal additional energy consumption, compared to first transferring heat to the air and then to the building's chilled water system. In addition, in a direct cooling system, the water temperature returning after cooling the IT equipment is much higher than typically found in data centers. This provides more opportunities for heat reuse or the ability to reject the heat to the atmosphere using a dry cooler, which eliminates the need for a cooling tower or chiller plant in most climates. Direct cooling systems typically require an external energy source to circulate the fluid in the loop. The general scheme is illustrated in Figure **??**, where the main components of the loop can be identified.

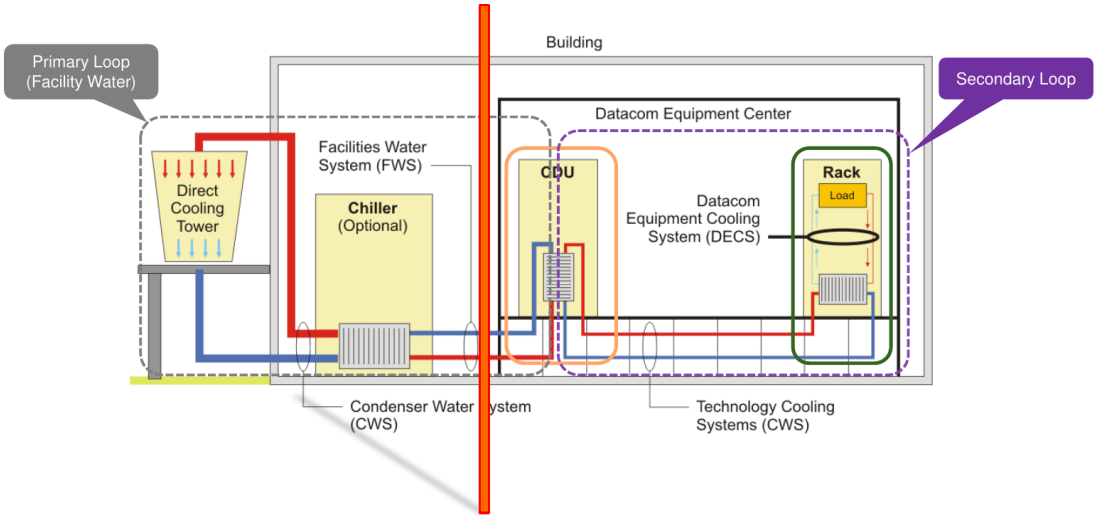
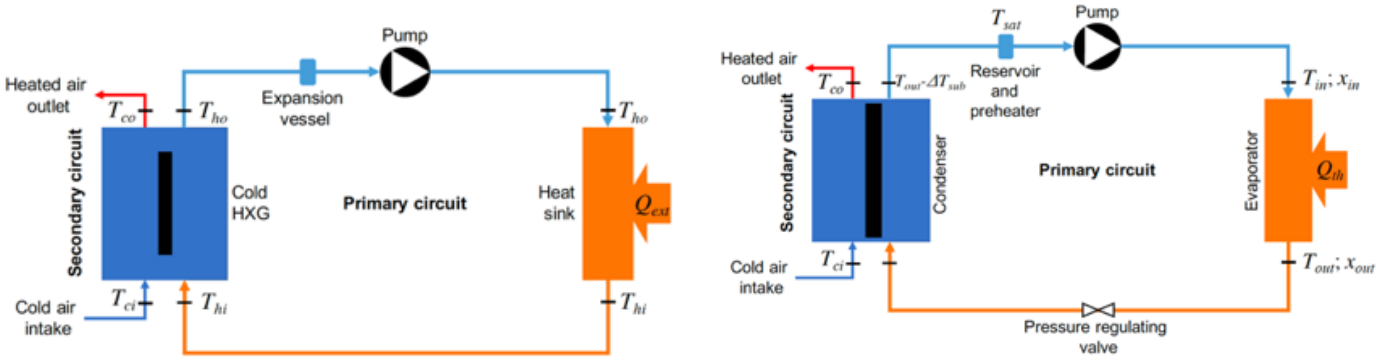


Figure 1: general scheme of direct cooling

The cooling process consists of two coupled loops. The secondary loop typically consists of a liquid coolant, such as water or a specialized fluid, that is pumped through a heat exchanger in close proximity to the CPUs or other heat-generating components. The heat from the CPUs is transferred to the coolant in the secondary loop, which is then carried away to the primary loop for dissipation. The primary loop, on the other hand, typically consists of a separate cooling medium, such as air or water, which is used to remove the heat from the secondary loop. The primary loop may pass through a cooling tower or other heat dissipation device where the heat is transferred to the environment. The two loops couple within a cooling distribution unit (CDU), which is a heat exchanger, designed to transfer heat from the secondary to the primary loop.

There are two types of direct cooling: *i)* single or *ii)* two-phase. Two-phase systems use latent and sensible heat with an evaporator and condenser. In a steady state, sub-cooled fluid flows from the condenser to the heat-sink, and hot fluid or vapor is evacuated in the cold heat exchanger. A Rankine vapor compression cycle can also be used. In direct contact cooling, a collecting tank is needed to ensure the fluid is in a liquid state before it returns to the evaporator. Single-phase cooling systems, known as liquid cooling, are perhaps the simplest and most commonly used configuration. The physical principle of this technology is relatively straightforward and, in its classic configuration, is the best-known technology. The working fluid is heated in the heat sink and flows through the transport lines, ideally adiabatically, 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 an expansion vessel and valves, must be provided to ensure smooth functioning. Figure 2a shows a conceptual diagram of the system.

Two-phase cooling systems are divided into two categories with different physical working principles and



(a) scheme of single phase cooling

(b) scheme of two-phase cooling

thermodynamic characteristics. In pumped two-phase technology, the evaporator operates at a higher temperature than the condenser, and a pump is installed on the liquid line. In vapor compression technology, the evaporator can operate at a lower temperature than the condenser due to the use of the Rankine cycle, and a compressor is used on the vapor line. Although the concept scheme for this technology is similar to single-phase cooling systems, the operating principle is completely different. Here, latent heat is used to extract heat from the hot source. Figure 2b shows the concept scheme of a pumped two-phase cooling system. When a free surface tank is used as a reservoir and saturation conditions are imposed, sub-cooled liquid from the condenser enters it.

Nowadays, there are many examples of systems that use this direct liquid cooling in the HPC market. Here are a few:

- **Atos Bull Sequana XH3000:** a liquid-cooled supercomputer that uses a specialized coolant to directly cool the processor and memory components. The system is designed for HPC workloads and can scale up to hundreds of thousands of cores;
- **Lenovo Neptune:** a liquid-cooled server that is designed to support HPC workloads. It uses a direct-to-chip cooling system that circulates coolant directly through the processor package, allowing for high power densities and more efficient cooling. The system can cool up to two processors per server;
- **Cray Shasta:** a liquid-cooled supercomputer that uses direct-to-chip cooling technology. It features a unique cooling system that circulates coolant directly through the processor package, allowing for higher power densities and more efficient cooling. The system can provide up to 400 kW of cooling per cabinet.
- **CoolIT Systems Rack DCLC:** a liquid-cooled system that is designed for data center and HPC applications. It features a modular design that can be easily integrated into existing data center infrastructure and can provide up to 100 kW of cooling per rack.

All of the above systems are single-phase direct cooling.

In Italy, the Tier0 flagship system Leonardo, developed by CINECA and installed at the Bologna Big Data Technopole, is provided with a complex direct liquid cooling system. Leonardo is based on the Atos XH2000 platform technology. Its computing racks are 95% direct liquid cooled by water at 37°C inlet temperature. More precisely, supercomputer circuits heat the cooling liquid up to 47°C. The cooling liquid is sent out to adiabatic disposers, 2250 kWf dry-coolers, bringing the temperature down to 37°C. The cooling process is based on water evaporation, reaching a Power Usage Effectiveness (PUE) of less than 1.1. The cooling system works in a closed circuit to avoid water waste.

4 NON-FUNCTIONAL PROPERTIES MODELING

The Non-Functional Properties (NFPs) are associated with several constraints a system faces to implement and deliver its intended functionality or service. In HPC machines, the dimensions of the system and their implicit complexity for managing and handling the distributed operation of software applications demand the modeling of several functional and non-functional properties allowing monitoring of the correct operation of the system. Some crucial NFPs for complex software systems include reliability, fault-tolerance, security, availability, efficiency, and scalability.

The research community and the industry have invested considerable efforts in developing and proposing approaches to represent and describe NFPs for HPCs. In most cases, the NFPs are modeled to extract and represent one or more properties resorting to specific abstractions levels (e.g., mainly at high-level software) or through cross-layer strategies (the combination and interaction of several abstractions levels, such as hardware and software) [106]–[108].

This Chapter first overviews the current strategies for reliability evaluation in HPC machines and analyzes several challenges and opportunities to improve the reliability of these systems. Then, Section 4.2 reviews the current thermal and power supply strategies for HPC and possible challenges in modern HPC generations. In addition, Section 4.3 analyses energy efficiency implications on memory management and by exploring approximate computing approaches. Section 4.4 reviews the current trends on performance evaluation and their main challenges. Finally, Section 4.5 analyzes current trends of temporal properties for HPC machines.

4.1 Reliability

Modern HPC machines are progressively scaling according to the required performance of data-massive and operation-intensive applications. In such complex HPC systems, the resilient operation of hardware and software components is crucial to provide services with acceptable quality and accuracy. The increasing size of today's HPC systems (e.g., numbers of processors and hardware accelerators, communication links, and integrated circuit sockets) also increases the probability of system failures. Some studies showed that the system's size influences its failure rate more than the hardware type per commodity cluster. In addition, the high density and dimension of their components and the considerable complexity of HPC machines impulse reliability challenges at local and global levels since software and hardware components' fault rates differ from those of the HPC system during their operative lifetime [109].

Reliability analyses and evaluations into real HPC machines reveals that a considerable percentage of error sources are due to hardware component in HPC systems (from 53% to 64%)[109]. Thus, methods and strategies to model, evaluate, and quantify the system's state and identify possible anomalies are mandatory when adapting recent processor architectures, such as RISC-V-based SoCs. Authors in [110] proposed three main guidelines to improve the reliability and resilience of HPC machines: *i)* identification of the factors affecting the fault rates in HPC systems, *ii)* the need for fault detection and error mitigation mechanisms for HPC systems, and *iii)* the overheating as a critical source affecting the reliability of the HPC machines.

In the first case, the equipment's technology, the system size, and the target applications are essential in selecting effective resilience mechanisms to maintain acceptable fault rate levels in HPCs. Moreover, the high complexity of the system and the possible effects of fault and error propagation demands detection and correction mechanisms with low overhead, suggesting their development and deployment at the hardware level. Finally, overheating is a critical parameter in commodity clusters and a source of unreliable operation. Thus, HPC machines must prevent overheating while preserving the balance between power/energy and performance.

4.1.1 Terminology

Several terms have specific meanings in the context of the reliability and resilience of HPC systems and are used to distinguish from the several sources of possible effects. We briefly summarize the reliability-related concepts to maintain a clear understanding of the terminology used in this document.

In general, the resilience of a system refers to the capability to recover or prevent failures (*"flexibility and survivability in the face of unexpected events"* [70]). In detail, resilience can be divided into two main concepts: *Reliability* and *Availability*. The former refers to the probability of failures appearing in the system and the *"the probability that a system or component will perform its intended function for a prescribed time and under stipulated environmental conditions"* [70]; meanwhile, the latter conveys the probability of a failure affecting the functionality of a system or the running application. In both cases, the source of the failures might be associated with hardware, software, or a combination of both. At the system level (e.g., commodity cluster or HPC machine), a ***failure*** is an event affecting the intended operation of the system, causing the corruption of the results of an HPC application. Moreover, ***errors*** are observable effects corrupting a part of the system's state and mainly arise at the software level (e.g., incorrect data from software operations). The errors can be divided into *corrected* when mitigation mechanisms remove error effects and *uncorrected* when the error remains in the system. In addition, the ***faults*** are the sources of errors at the hardware level. They mainly represent incorrect logic states in the circuits caused by damages originated from manufacturing defects or external sources (e.g., radiation, electromagnetism, or temperature variations) that physically corrupt the hardware components permanently (e.g., accelerated aging, wear-out, electromigration, or Time-Dependent-Dielectric-Breakdown) or transitory (e.g., bit-flips, or Single-Event Upsets) [111]. Another key concept is *safety* that is generally mandatory in high-reliable systems and is defined as *"the ability to avoid damages to people, things and environments"* [70],

4.1.2 Fine-Grain Reliability Issues in Modern Hardware Platforms for HPC

The adoption of RISC-V processors has significantly increased in recent years, mainly due to its open standard and free instruction set architecture [68]–[72]. In addition, application-specific functions can be added to a RISC-V SoC design, and accessed by adding custom instructions to the standard RISC-V instruction set. This is proven to be a driving factor for a broader adoption of RISC-V in the HPC sector [71].

However, RISC-V SoCs for HPC will require their implementation in scaled microelectronic technologies, hence it is expected that they will suffer reliability problems associated with technology miniaturization [68]. This poses significant challenges to designers of RISC-V SoCs that will have to be used in electronic systems for highly autonomous applications, especially those for safety critical applications (e.g., autonomous vehicles, robots, etc.).

In particular, the technology scale continuously reduces power supply voltages (and consequently noise margins) and internal node capacitances, as well as in an increase of operating temperature, that will continue to increase SoCs susceptibility to faults (both transient and permanent) and aging phenomena (such as Bias Temperature Instability, or BTI) [68]–[71], [112]–[115].

As for transient faults (TFs), their likelihood is expected to increase with technology scaling [114], [116], [117]. They are generated by energetic particle hits (in particular, Alpha particles and neutrons) that, affecting nodes of datapaths and memory blocks of SoCs, can give rise to soft errors, with a potential catastrophic impact on the SoC correct operation.

Power supply noise is another phenomenon that is also expected to give rise to considerable signal integrity problems in scaled technologies [115], [118]. In fact, as the integration density increases, the number of devices switching simultaneously increases as well. This causes an increase in the noise affecting the power supply networks (i.e., Simultaneous Switching Noise, SSN), due to both resistive and inductive voltage drops,

with a considerable decrease of noise margins of the whole SoC.

In addition, bridging faults and opens in the interconnects due to electromigration have recently emerged as a major reliability issue for modern complex ICs, like RISC-V based SoCs. This because with technology scaling, the area of wires shrink, with a consequent increase in current densities [90], [96]–[98], and a consequent increase in the likelihood of occurrence of electromigration.

Moreover, modern ICs implemented with aggressively scaled technologies are also becoming increasingly prone to aging mechanisms, such as bias temperature instability (BTI) [112], [119]. BTI increases the absolute value of transistors' threshold voltage over time, thus resulting in the degradation of their driving strength. In data-paths, such a threshold voltage increase may lead to a delayed transition of a signal at the input of an output flip-flop. In case of critical timing paths, this signal may reach its stable final value later than expected and violate the setup time of the output flip-flop, thus resulting in the generation of an error at the flip-flop output, possibly compromising the SoC correct operation. Additionally, it has been shown that BTI also negatively impacts the susceptibility of ICs to soft errors [120]. This because by increasing the absolute value of the transistor threshold voltage over time, BTI also reduces the entity of the restoring current of the transistors driving circuit nodes. As a result, the likelihood of transient fault generation due to particle hits significantly increases over time [112], [120].

Several approaches exist in the literature to design reliable, safe and resilient systems despite the presence of faults and aging phenomena. For instance, reliability can be guaranteed using Error Correcting Codes (ECCs) for memory blocks [121], and Modular Redundancy (MR), or On-Line Testing and Recovery, for logic [122]. In addition, safety can be guaranteed by employing proper monitoring schemes to detect the system's erroneous behavior, and reaction strategies to be activated (e.g., at the system level) after detection, to reduce the risks associated with the detected erroneous behavior [123]. Resiliency can be obtained through the adoption of proper hardening design approaches [rossi09], to reduce the system's likelihood to exhibit an incorrect behavior, once affected by faults.

However, such solutions may require high costs in terms of area overhead (higher than 200% in some cases) and power consumption. Therefore, alternative solutions requiring lower costs have been presented in the literature (e.g., those in [113], [115], [124], [125]).

In [124], the authors propose an on-line testing approach for the control logic of high performance microprocessors. Rather than adding information redundancy (in the form of error detecting codes), the approach uses the information redundancy (referred to as *Function-Inherent Codes*) that the microprocessor control logic typically inherently has, due to its required functionality. This solution allows to achieve on-line testing at very low costs in terms of area and power consumption, and with a lower or similar impact on system performance compared to traditional on-line testing approaches. However, the percentage of possible faults occurring in the field that can be detected by this solution is lower than that achievable by using traditional on-line testing approaches.

In [115], the authors propose a methodology to detect transient faults and power supply noise in General-Purpose Graphics Processing Units (GPGPUs). The methodology is based on temporal redundancy, where instructions are re-executed on idle functional units of the GPGPU. The methodology requires low area and power overhead, as well as low impact on performance. However, its effectiveness mainly relies on the large redundancy/parallelism present on GPGPUs, so that it may be not straightforwardly applicable to other kinds of processors, like RISC-V based SoCs. Similarly, In [125], the authors present an aging sensor to monitor performance degradation at the output of critical datapaths, caused by BTI affecting pMOS transistors (i.e., Negative BTI, or NBTI), which at the time of the publication were the dominant aging phenomena. However, these solutions may be not effective in detecting BTI affecting nMOS transistors (i.e., Positive BTI, PBTI), whose impact on performance degradation became comparable to that of NBTI for modern SoCs implemented by deeply scaled technologies. In [113], the authors present a strategy that enables to reduce the

impact of BTI on the susceptibility to soft errors of standard and low-cost robust latches. This is achieved at a limited increase in terms of area overhead, power consumption, and with no impact on the latch input-output delay. A possible limit of this approach is that it may be not applicable to all kinds of latches and memory cells.

As far as we are concerned, existing approaches in the literature have been demonstrated sufficient to guarantee the required levels of reliability, safety and resiliency of SoCs employed in current, partially autonomous systems. However, these techniques may not be able to guarantee the more stringent requirements in terms of reliability, safety and resiliency required for SoCs for the HPC domain and in highly autonomous systems employed for safety critical applications (such as unmanned robots and vehicles) [114].

4.1.3 Testing HPC systems

The test of HPC machines usually consists of several procedures *(hundreds of tests)* intended to identify possible software errors and hardware faults affecting the system. Moreover, these testing methods also determine the commodity clusters' operative state and support the subsequent correction stages of the system. Several experts agree that modern HPCs need low error rates in their components (e.g., in CPUs and GPUs) and suggest that focused, extensive, and exhaustive testing might be required [126]. However, an exhaustive hardware testing of HPC machines can hardly be performed due to the complexity and the number of components involved in their operation.

Classical testing techniques generally focus on the high-level abstraction of the available HPC system models and mainly target the software stack. Those techniques include *profiling* and *regression testing* (e.g., ReFrame [127]), that have been an essential step of any software development and integration cycle in HPCs. Other approaches include *acceptance tests*, that consist of several stages, as detailed in the following.

In [128], the authors describe an acceptance test comprising two stages: *i)* hardware acceptance testing (HWA) and *ii)* final integration testing (FI). The first stage consists of hardware diagnostics (usually performed by manufacturers of integrator companies) to ensure that each component (e.g., processors, memory, interconnect) meets the required specifications for the HPC system. The second stage (FI) included three elements: functionality testing (FT), performance testing (PT), and stability testing (ST). The FT ensures that the software stack (e.g., *scheduler*, *compilers*, and *libraries*) operate correctly. The PT checks the execution and scale of applications to achieve nominal performance. The ST ensures that the HPC system supports the execution of diverse workloads continuously and for an extended period.

The regression testing is typically customized and ad-hoc, focusing on the correctness and quality operation of the software/application running on top of the HPC system. Moreover, some basic functionality features of several hardware components of the HPC systems can also be checked. Interestingly, this strategy targets the identification of functional and non-functional errors and failures after composition changes (e.g., enhancements, patches, or configuration changes in the system). Unfortunately, the exhaustive evaluation of hardware components is not a priority for the test.

In contrast, hardware acceptance tests evaluate the functional operation of the system's components (e.g., boundaries of new technologies) and are usually performed during the configuration and setup steps of the HPC machine but require information only available to the manufacturer or the system integrator (e.g., Intel, Nvidia, Cray). However, the coverage of hardware acceptance tests is not specified, possibly limiting their effectiveness for all components. It is worth noting that functional hardware tests are not usually deployed during the production stage of commodity clusters.

The other tests (*functionality*, *performance*, and *stability* tests) mainly focus on verifying the status of the software layers on top of the system. Other typical functional software testing approaches (e.g., sanity checks and benchmarks [129]) allow observing some features in the HPC machine, such as the performance or the

variation in precision in the components (e.g., the *Variety* framework [130]). Unfortunately, most approaches directly neglect hardware and component testing.

Profiling and production monitoring mechanisms usually collect information from clusters (e.g., current, energy, power) to evaluate the status (*"health"*) of the components in the HPC system, and might support functional testing goals in the system. The collected monitoring information might serve as preliminary testing and guide the identification of faulty nodes for later check, evaluation, and mitigation goals [131], [132]. However, the measured parameters are collected independently for all components, which might not capture the overall behavior of the HPC system under realistic workloads [133]. The HPC monitoring methods include in-field sensors, such as Built-In Self-Test (BIST) modules in analog and power electronics equipment widely used to measure electric parameters (e.g., power impedance and noise spectrum [20], [134].

4.1.4 Hardening solutions for HPC systems

Fault-tolerance mechanisms for hardening HPC equipment have been exhaustively analyzed, and most current guidelines divide the hardening solutions into two main approaches: 1) *hardware* and 2) *software*. In the first case, the hardware fault-tolerance solutions require special structures to detect and correct errors. In HPCs, this approach mainly targets the interconnect infrastructures and the memories of commodity clusters. In both cases, modern standard error mitigation techniques, such as Error Correcting Codes (ECCs) [135]–[137] are highly employed with minimal hardware and performance overheads. The same approach has been extended to the complete memory hierarchy of cluster processors and hardware accelerators [138]. In addition, ECC-based hardening can be extended to the intra-node interconnect infrastructure. Similarly, hardware redundancy and spare units are hardening alternatives [139], [140]. Other techniques include dynamic voltage and frequency scaling and shutdown mechanisms for CPUs, accelerators, and ASICs, as strategies to prevent and reduce hardware faults arising from overheating issues [110].

Software hardening solutions are highly flexible since they only resort to software-based strategies. These are implemented by modifying the application code or cleverly managing the configuration of the system or the application. The software-based gardening strategies can be classified as *i) Reactive* and *ii) Proactive*. The former technique aims to minimize the application's impact in the presence of faults in one or more components of the HPC system; meanwhile, the latter technique focuses on predicting faults and failures states complemented with corrective actions based on migration techniques [141].

The reactive mechanisms include software checkpoints and restarting mechanisms [142], [143] that rely on statistical properties of failures in the system to correct possible operative errors of the applications [109], [144]. Other software-based hardening techniques include a complete or partial duplication of the application (i.e., in terms of code and execution). The Sentinel [145] approach consists of a compiler-based framework that combines code profiling with fuzzing engines to identify the most vulnerable code blocks per application for later compacted replication. Some duplicating strategies focus on smart and optimized program duplication to exploit idle hardware in commodity clusters. Other strategies combine checkpoints with code duplication to harden applications. In [146], the authors propose *Hauberk* for GPUs, that combines in-field profiling information with software error detectors (for loop and non-loop portions of code) to identify and recover from errors on those variables prone to propagate errors and corrupt the results of a program, meanwhile minimizing its impact in performance. Other fault-tolerance approaches involve the clever division of tasks per application and the management of task scheduling, which is complemented with partial cancellation and restarting of tasks [147], [148]. On the other hand, authors in [149] propose proactive methods for task migration from faulty commodity clusters (or *unhealthy nodes*) into failure-free ones. This approach resorts to virtualization techniques of operating systems, health monitoring mechanisms, and load-based migration strategies. Other hardening mechanisms include algorithmic-based fault tolerance [150], [151], speculative execution, or forward recovery [152].

4.1.5 Opportunities for improving the reliability in HPC systems

Regarding fine-grain technology issues of hardware platforms, there are several challenges and research opportunities to develop innovative monitoring approaches and design techniques for modern SoCs (e.g., SoCs based on the RISC-V architecture), that will enable to achieve higher levels of reliability, safety and resiliency than existing solutions. The effectiveness of the developed approaches for enhanced reliability, safety and resiliency with respect to likely faults and aging phenomena affecting modern SoCs might be verified by means of cross-layer approaches (i.e., combination of electrical level simulations and architectural-level fault injection campaigns).

The current functional testing strategies for HPCs are based on high-level software approaches focused on verifying the software layers and the complete system state. However, hardware testing (focused on the underlying architecture of the commodity clusters, such as processors and hardware accelerators) is barely deployed during the production stages of the HPC by restrictions on their time execution or the availability of effective hardware tests due to the lack of hardware details. Interestingly, both restrictions can be solved in open-hardware environments, such as those based on RISC-V platforms for HPCs, and represent an outstanding opportunity to improve the effectiveness of functional testing mechanisms for HPCs, allowing the merging of performance and functional test goals (typical of HPC system tests) with hardware testing goals. The availability of the hardware architecture in combination with the adaption of functional testing strategies for hardware, such as the Software-Based Self-Test (SBST) [153], might contribute to designing more effective testing routines considering the architectural features of all hardware elements composing the commodity clusters (processors, accelerators, and intra-node interconnect infrastructures).

Other opportunities to improve HPC reliability include exploring and adapting hardware-based hardening solutions for the execution cores (processors and hardware accelerators) by exploring and adapting mitigation strategies, such as flexible Built-In Self-Repair mechanisms, reconfigurable mechanisms, Design Diversity approaches for hardware accelerators, and Error-Correcting structures.

4.2 Thermal & Power Supply

4.2.1 Thermal Management

Reliable and environmental-friendly operation of HPC infrastructure requires proper thermal management at the level of the individual chip for all processors and accelerators. To be able to perform computation, digital integrated circuits draw electrical power and generate heat that has to be dissipated to keep operating temperature sufficiently low for reliable operation.

CMOS integrated circuits are prone to thermal runaway [154] since part of their power consumption, static power, increases as temperature increases, potentially generating a positive feedback loop. High operating temperatures impact system reliability and lead to failures over prolonged time, even in the absence of thermal runaway [155]. Additionally, thermal cycling [156] due to computation-dependent, time-varying power consumption further impact reliability.

In the HPC environment, thermal management directly impacts the computational performance due to the so called Dark Silicon issue [157] as a heat dissipation solution capable of dissipating the maximum system power consumption is often either not technically possible or not economically feasible. To prevent damage to the integrated circuits, the heat dissipation solution is complemented with a run-time thermal control strategy that adapts the operating power consumption - and consequently achievable performance - to the thermal state of the chip. Turbo Boost is a commercial implementation of such a thermal management policy for Intel processors.

Heat dissipation solutions use mechanisms to facilitate heat transfer from the integrated circuits and other

power dissipating components to the outside environment. Air cooling uses heat sinks to increase the heat exchange surface area, and possibly fans to increase air flow. Liquid cooling improves heat dissipation further by replacing air with a suitable coolant liquid. Immersion heating is possible, where the entire computing device is immersed in the cooling fluid, but it is often more practical to apply liquid cooling in a localized way through heat exchangers in thermal contact with the highest power dissipating components. In this case, it is important to validate that sufficient cooling is provided to the rest of the system components. Pumps are required to ensure sufficient coolant liquid flow. Evaporative cooling is the next generation cooling solution utilizing latent heat of vaporization to further improve heat dissipation. These systems require similar components to liquid cooling, but use refrigerant fluids with a suitable boiling point and provide a significant increase in dissipation heat flux compared to conventional alternatives.

Run-time thermal management improves the cooling system efficiency and compensates its limitations utilizing temperature sensors, a control algorithm and actuators to keep temperature under control. Actuators are divided in two categories: those that can increase the cooling system dissipation heat flux, such as increasing fan or pump speeds, and those that reduce the dissipated power, such as DVFS. The former type of actuators are used to improve cooling energy efficiency by reducing the power used by the cooling system and improving PUE. The latter kind of actuators are instead needed despite their impact on computation performance due to the aforementioned Dark Silicon issue. There is a wide corpus of academic research on thermal management policies due to the possibility of obtaining performance gains by maximizing the use of the available cooling system capacity [158]. Additionally, low-overhead thermal control policies is another area which promises further performance gains [159].

Thermal simulation is the main tool used for the design, verification, and validation of the heat dissipation solution [160]. Due to its central role and challenging behavior in the design of cooling systems, several methodologies and tools have been developed by the academic community, so improving simulation performance and accuracy is a crucial subject of on-going research.

4.2.2 Power Supply Regulation

A main limitation to the computational performance of modern RISC-V system-on-chip (SoCs) processors is power consumption [161]–[164]. In particular, battery lifetime and maximum operating temperature (associated to a maximum power consumption) constraint the computational performance of modern SoCs [161], [164]. Consequently, power management strategies, such as *Dynamic Voltage and Frequency Scaling* (DVFS), are usually adopted to increase the performance-per-watt of complex SoCs [165]. These strategies usually employ an integrated Power Control Unit (PCU), that monitors the activity of different power domains and that, based on such activities, determines the optimum voltage for each domain.

Efficient DVFS techniques require that the voltage of each power domain can be controlled individually. This can be achieved by employing different Voltage Regulator (VR) circuits. The possibility to have a large number of power domains within a SoC has been enabled by the availability of VRs implemented on the SoC itself (usually referred to as Fully Integrated Voltage Regulators - FIVRs), rather than on the motherboard outside the die [161]–[164]. FIVRs do not require extra I/O pins on the die (as it is the case for VRs' implemented on the motherboard), and they can be placed close to their respective loads, thus minimizing the power dissipated in interconnections [162]–[164]. Moreover, FIVRs enable faster transitions between different values of supplied voltages (power states), thus enabling to implement more efficient power management strategies.

In modern SoCs, FIVRs are typically implemented by means of a Buck topology DC-DC switch-mode converter [162]–[164], that receives as input a desired reference voltage (encoded as a digital word) from the PCU, and generates an output voltage (i.e., the power supply voltage) that is proportional to the received reference. The main components of a FIRV are the switching devices, a low-pass filter, and a FIVR Control

Module. Except for the low-pass filter, the remaining FIVR components are implemented within the SoC die, thus they suffer from the same reliability problems as the other SoC components, due to technology scaling [162]–[164]. In particular, process parameter variations occurring during fabrication, faults and aging phenomena (such as Bias Temperature Instability - BTI) are some important problems that may cause significant variations of the FIVR output voltage, and of the FIVR response to load or reference voltage changes. Such undesired variations of FIVR output voltage and response time may exceed the maximum limits required for the correct operation of the SoC that the FIVR is powering, with consequent possible catastrophic effects if the SoC is running safety critical applications (e.g., autonomous cars or robots interacting with humans).

In order to prevent such catastrophic consequences, some approaches have been presented in the literature to detect/prevent the occurrence of incorrect power supply voltages (e.g., those described in [75], [162], [163], [166]–[169]).

The solution suggested in the ISO 26262 standard [75] continuously monitors the FIVR output voltage during its operation in the field, and generates an alarm signal in case it differs from the desired reference voltage coming from the PCU. In particular, this solution employs an Analog-to-Digital Converter (ADC) [170] to convert the FIVR output voltage into a digital word, that is then compared to the digital word coming from the PCU (that encodes the desired reference voltage). A limit of this solution is that it may be very expensive in terms of area overhead (higher than the 100% of the FIVR area). In addition, this solution is not able to monitor/detect variations of the response time of the FIVR to load or reference voltage changes.

Instead, in [162], [163] the authors present some Design for Test (DfT) structures that, if included in the SoC, enable to characterize some specific FIVR key performance parameters (e.g., the response time of the FIVR to load or reference voltage changes, the FIVR efficiency, the output voltage ripple, etc.). However, the characterization enabled by the proposed DfT structures cannot be performed for each fabricated chip because of its high cost in terms of time. Therefore, it is performed only for a small set of fabricated SoCs. Moreover, even testing the FIVR performance parameters of all fabricated chips would not guarantee the correct operation of the FIVRs in the field. In fact, the impact of faults and aging mechanisms occurring in the field will not be detected if the FIVRs are only tested after fabrication.

In [166], the authors proposed a detector capable of detecting power supply variations (power supply noise) by revealing the increase in the delay of gates powered by such a monitored supply voltage. These detectors present low-cost in terms of area and power overhead and are effective in detecting power droops. Properly modified versions of such detectors might be employed to detect slow transient responses of the FIVR to load (or reference voltage) variations, or the presence of an excessive ripple on the FIVR output voltage. However, these detectors may be not able to detect the possible occurrence of stable incorrect FIVR output voltages.

In [167], [171], the authors present a self-checking monitor that is able to detect faults affecting the control circuitry and the power switches of DC-AC converters employed in photovoltaic PV systems. This is achieved at low cost by measuring the harmonic content of the current delivered by the converters to the load. Properly modified versions of this monitor might be employed to detect excessive reductions of the FIVR power efficiency, but they may be not effective in detecting, for instance, the inability of the FIVR to keep the output voltage at the desired value.

In [168], [169], the authors present a solution to monitor the power supply voltage by using Tunable Delay Lines (TDLs) that are configured as ring oscillators to generate the clock (CK) signal for a counter. The counter counts the number of CK periods in a given time interval. Since the propagation delay of the delay elements composing the TDL is proportional to the voltage of their power supply, the final count of the counter is proportional to the power supply voltage. A limit of these solutions is that they may be expensive in terms of area overhead. In addition, the propagation delay of the delay elements composing the TDL is susceptible to process parameter variation occurring during fabrication. Therefore, a complex calibration phase would be needed in order to be able to employ these solutions to monitor the FIVR output voltage

during in-field operation.

Finally, accurate analyses at the electrical level of the effects of the most likely faults and BTI phenomena possibly affecting a FIVR in the field is missing in the literature. In addition to such analyses, proper metrics enabling to evaluate the severity of the effects of such faults and aging phenomena are not presented in the literature, and would be needed in order to develop efficient monitoring schemes to detect the occurrence of incorrect FIVR output voltages during its in-field operation.

In order to fill the gap in the state of the art regarding efficient monitoring solutions for power supply voltages powering modern SoCs (e.g. based on the RISK-V architecture), the project partners plan to analyze, at the electrical level, the effects of the most likely faults and BTI phenomena possibly affecting a FIVR during its operation in the field. In this phase, the partners also plans to introduce metrics to evaluate the severity of such effects on the SoC correct operation. The performed analyses and metrics will enable to study the existence of possible correlations between performance parameters of the FIVR and other circuit parameters (e.g., input/output/ currents/voltages of the FIRV), that could enable to develop, in a successive phase, low-cost monitors to detect variations of power supply voltage exceeding the limits required for the correct operation of the SoC that the FIVR is powering

4.3 Energy Efficiency

Global warming caused by greenhouse gas emissions is one of the main concerns for both developed and developing countries. In a fast growing Information and Communication Technology industry, current energy efficiency methodologies are not sufficient for new raising problems such as optimization of complex distributed systems. Existing studies claimed that 78.7 million metric tons of CO_2 are produced by datacenters, which is equal to 2% of global emissions [172]. CDCs in the United States consumed 100 billion kilowatt hours (kWh) in 2015, which is sufficient for powering Washington, DC [173] for a year. The consumption of electricity will reach 150 billion kWh by 2022, that is, increase by 50% [173]. Moreover, energy consumption in CDCs can be increased to 8000 terawatt hours (TWh) in 2030 if controlled mechanisms are not identified [174]. Due to underloading and overloading of resources in infrastructure (cooling, computing, storage, networking, etc.), the energy consumption in Cloud and HPC datacenters is not efficient and the energy is consumed mostly while some of the resources are in idle state, which increases the cost of Cloud and HPC services [173].

Carbon footprints produced by CDCs are the same as that of the aviation industry. Currently, between 80 and 95 million metric tons of CO_2 are produced by datacenters, which corresponds to about 3% of the global emissions, overtaking the aviation industry [172]. Big cloud providers, like Google, Amazon and Microsoft, are working to reach zero-emissions cloud-based services. Renewable energy, heat waste utilization and sustainable cooling mechanisms are the current focus for the so-called green cloud.

Energy consumption is not related to an unique aspect of a datacenter: on average, processors take about 50% - 55% of the available energy, while more than 20% is taken by the storage, and the remaining 30% is consumed by cooling, memory and network. Therefore, underloading and overloading of the resources lead to an energy waste and inefficiency of the data center. A resource in idle state continue to consume a relevant amount of energy. Resource management policies and computing architectures and algorithms are the key element and focus of current research, albeit they remain challenging.

The current research on sustainable Cloud and HPC computing is organized into several categories: application design, sustainability metrics, capacity planning, energy management, virtualization, thermal-aware scheduling, cooling management, renewable energy, waste heat utilization, and approximate computing.

The Cloud and HPC infrastructures need continuous monitoring in order to perform predictions and estimations for the different aspects. Sustainability metrics are becoming essential, but every year researchers try

to define new metrics or use different combination of metrics to reach new sustainability levels [173]. The following are environmental-related metrics:

- **Resource Utilization (RU):** defined as the execution time of workloads with respect to the uptime of the resource
- **Carbon Usage Efficiency (CUE):** the CO_2 emissions over the total energy consumed
- **Energy Reuse Effectiveness (ERE):** i.e. the energy reused and consumed by cooling, lighting and other IT devices over the total energy consumed
- **Green Energy Coefficient (GEC):** the fraction of green energy consumed over the total energy consumed by the datacenter
- **Cooling System Efficency (CSE):** the amount of cooling capacity per unit of energy consumed to maintain the datacenter
- **Energy Consumption (EC):** the amount of electricity consumed by a resource to complete the execution of an application
- **Energy Efficiency (EE):** the number of executed workloads over the total energy consumed to execute the workloads
- **Power Usage Effectiveness (PUE):** the energy consumed by the ICT devices over the total energy consumed by all the Data Center devices including the cooling devices.

Current challenges to improve energy efficiency (EE) and energy consumption (EC) need the interaction of several factors, including the evaluation and monitoring of resources utilization, software design, cooling systems, and temperature monitoring. Different metrics need to be monitored, such as PUE and RU. In fact, EC can be improved by working at three levels:

1. **Software level** (Deep Power Down)
2. **Hardware level** (Processor instruction set)
3. **Intermediate level** (energy-aware provisioning of resources)

4.3.1 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. Another basis for such models lies in the popularity of virtualization/cloud computing services, that allow one to rent computational resources on demand; minimizing the integral of memory capacity "rented" for a computation minimizes monetary cost.

The first to address a similar problem was Chrobak [175], 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 [176] and later, in the more general version where pages have sizes and weights, by Gupta et al. [177]. It turns out [178], [179] 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. [180], [181] showed that the optimal competitive ratio for deterministic online green paging is $O(\log p)$. However, many questions remain open, such as whether we can break this logarithmic barrier by using randomization or by leveraging (machine-learning) predictions about future page requests.

4.3.2 Approximate Computing

Approximate Computing (AxC) proved to be a very promising one [182]. The idea behind AxC is that several applications do not need to be executed on "accurate" and thus "energy-expensive" hardware. AxC aims to reduce the hardware's precision to save energy consumption. Interestingly, the reduced precision leads to applications providing less accurate but still good enough results while reducing by orders of magnitude the required energy [183], [184]. Such applications are intrinsically resilient to noise and errors affecting the computation (i.e., because of the less precise hardware). Indeed, the inherent resiliency property tightly depends on the application domain.

Well-known examples are algorithms dealing with noisy real-world input data (e.g., image processing, sensor data processing, speech recognition, etc.) or with outputs that require human interpretation, such as digital signal processing of images or audio; also data analytics, web search, and wireless communications exhibit an equivalent property [185]–[187]. Other examples are iterative applications that process large amounts of information, sample data, stop the convergence procedure early, or apply heuristics [188]. Most of the proposed techniques try to define new methods to generate alternative versions of specific components (hardware or software) with fewer resources. For example, several proposals of approximate arithmetic operations [189]–[191]. Such variants differ from speculative implementations because they do not focus on generating alternatives but restoring the possible introduced error [192]–[194]. Other techniques generate variants by considering a high-level description of the application or its implementation at a low-level [185]. Moreover, existing approaches target only implementations at a specific level of the computing stack, i.e., software or hardware.

Approximation techniques can be applied to all the computation stack levels. Figure 3 divides approximation techniques into three groups that define their implementation level: Software, architecture, and hardware. As Figure 3 shows, some approximation methods can be implemented on multiple levels. For example, Load value approximation can be implemented by purely software approaches or at memory control units. Figure 3 presents only some of the most used approximation methods and the most discussed in the literature. However, there are uncountable ways of approximating an application, and the very definition of what is to be

considered an approximation or not is debatable [195].

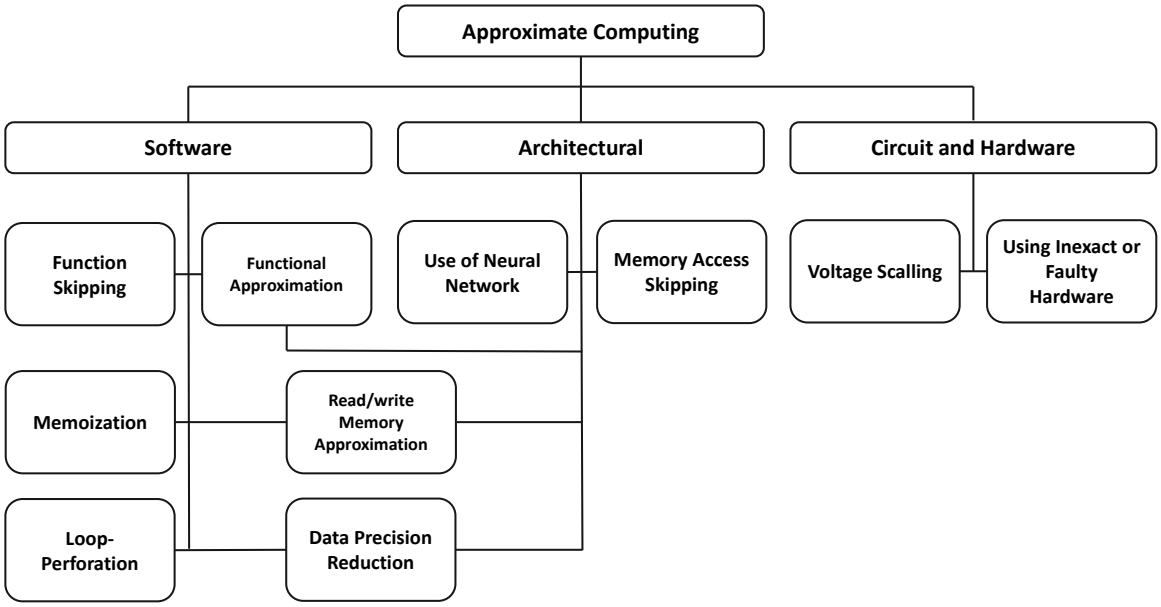


Figure 3: Approximate computing classification (Figure adapted from [195])

The loop-perforation technique is an excellent software approximation example, achieving useful outputs while not executing all the iterations of an iterative code. Similarly, task-skipping involves skipping code blocks during run-time following previously defined conditions. Another approximation technique for software applications is reducing the bit-width used for data representation. This technique mainly impacts the memory footprint of the application. Indeed, data precision reduction can also impact the execution time performance of the software, but that would depend on the hardware implementation of operations in use by the application. Hardware-based approximation techniques usually make use of alternative speculative implementations of arithmetic operators. An example of this approach is the implementation of variable approximation modes on operators [196]. Hardware approximation is also present in the image processing domain in the form of approximate compressors [197]. The techniques presented in Figure 3 are explained in detail and with implementation examples below:

Function Skipping. In a system composed of tasks that complement each other to provide a final result, some of the tasks can be skipped while maintaining a level of accuracy and error resiliency defined by the user [198].

Memoization. Traditionally, memoization consists of saving outputs of functions for given inputs to be reused later. Given that some input data are frequently reused, their calculated outputs can be stored and used without needing to re-execute the function. Memoization can also approximate applications; if similar inputs provide similar outputs for a given function, the already-calculated function output can approximately cover a range of inputs. In [199], the authors propose approximate value reuse for memoization, providing a shallow accuracy loss.

Loop-Perforation. In loop-based algorithms, loop perforation can reduce execution time and energy consumption. An excellent example of this type of application is numerical algorithms. For example, calculating an integral using the trapezoidal method consists of calculating the area of a high number of trapezoids under the curve of a function, providing an approximation of the area beneath it. Reducing the number of calculated trapezoids will make the final value less accurate, but the program will finish earlier and use less energy. The

literature also presents techniques to apply this approximation method on general-use algorithms, filtering out the loops that cannot be approximated and using loop-perforation on those that can [200]. Authors claim that their approach typically delivers performance improvements greater than a factor of two while introducing an output deviation of less than 10%. Loop-perforation is an algorithm-based approximate technique, as it only applies to loop-based code, limiting its applicability. It can be implemented both with software and programmable hardware code. The difference is that, on programmable hardware, a loop might be implemented as many circuits executing in parallel (one being each iteration of the loop) or one circuit re-executed in a timeline. Therefore, the impact of loop perforation on software and hardware implementations can be very different. In software, it will mainly impact the execution time of the application, while in hardware implementation, it could also affect energy and area consumption.

Functional Approximation. Some systems have components that do not need to provide accuracy as much as others. The idea is to take advantage of the fact that even inside an algorithm, some components affect less the final accuracy than others. Those components can be approximated to reduce energy consumption and improve execution time performance [201]. The functional approximation can appear as alternative speculative implementations of arithmetic operators on the architectural level. When applied to software applications, approximate computing usually consists of inexact computations, which provide results with lower accuracy than usual [202]. Most approximate computing techniques for software consist of modifying the algorithm so that it executes approximately, providing a final result more rapidly. One of the problems with a functional approximation is that it introduces errors in the system output that may be too big to be acceptable. For example, the works at the architectural level of approximate operators [196] do not present a significant hardware implementation area reduction compared to a traditional operator. The size of the used area on programmable hardware devices directly impacts system reliability [203]. Therefore, the quality loss (manifested as errors in some operation results) introduced by the approximation would only be acceptable by safety-critical systems if it sharply reduced its area.

Read/Write Memory Approximation. It consists of approximating data loaded from or written into the memory or the read/write operations themselves. This is primarily used on video and image applications, where accuracy and quality can often be relaxed, to reduce memory operations [204], [205]. In [206], the authors propose a technique that uses dynamic bit-width based on the application accuracy requirements, where a control system determines the precision of data accesses and loads. The authors claim it can be implemented in a general-processor architecture without hardware modifications by communicating with off-chip memory via a software-based memory management unit. The approximation can also be applied to the cache memory. If a load data cache is missed, the processor must fetch the data from the following cache level or at the main memory. This can be a very time-consuming task. Load value approximation can be used to estimate an approximate value instead of fetching the real one from memory. In [207], the authors present a technique that uses the GPU texture fetch units to generate approximate values. This approximation causes an error of less than 0.5% in the final image output while reducing the kernel execution time by 12%. In [208], the authors propose an approximation technique for multi-level cell STT-RAM memory technologies by lowering its reliability to a user-defined accuracy loss acceptance. This memory technology has a considerable reliability overhead, which can be reduced. They selectively approximate the storage data of the application and reduce the error-protection hardware minimizing error consumption.

Data precision reduction. Data precision reduction is one of the techniques that can be implemented both at a software and architectural level. Reducing the data precision of an application (i.e., the number of bits used to represent the data) is a straightforward technique to reduce memory footprint. Reducing memory usage also reduces energy consumption at the cost of accuracy (i.e., fewer data has to be transferred from/to the memory). In [209], the authors show that reducing floating point precision on mobile GPUs can reduce energy consumption with image quality degradation. This degradation, however, can be acceptable and even unperceivable to the human eye. Lower memory utilization is suitable for safety-critical systems because it reduces the essential and critical bit count, making them less susceptible to faults. Reducing the bit-width

used for data representation is also a popular approximation method [210]. The way data precision reduction can approximate software and FPGA applications is obvious: It is a matter of code modification. In software, the precision of floating-point units can be easily modified using dedicated libraries or by merely changing the variable type. The same can be done at VHDL/Verilog projects: A design can be adapted to process smaller data vectors. Data precision reduction can improve area and energy costs for hardware projects, but frequently does not present a high-cost reduction on software. For example, fixed-point arithmetic can approximate mathematical functions, such as logarithm, on FPGA implementations providing low area usage [211]. On software, however, it can increase the execution time of the application because all the operations and data handling routines are implemented at the software level.

Use of Neural Networks. A neural network can learn how a standard function implementation behaves with different inputs via machine learning. The neural network can implement approximate functions in a complex system via software-hardware co-design. Traditional approximable codes can be transformed into equivalent neural networks with lower output accuracy but better execution time performance [212].

Memory Access Skipping. Using a combination of the memoization and function skipping techniques, it is likewise possible to skip memory accesses. Uncritical data can be omitted if it will not heavily damage the output accuracy. Approximate neural networks can skip reading entire rows of their weight matrices if those neurons are not critical, reducing energy consumption and memory access and improving performance [213].

Voltage Scaling. The supplied voltage level can be scaled at the circuit level, impacting the computation timing of processing blocks inside the clock period. It affects the final result's accuracy and energy consumption [214]. In [214], the authors propose the voltage over scaling of individual computation blocks, assuring that the accuracy of the results will "gracefully" scale with it. Voltage scaling can be implemented dynamically in hardware. DVFS, for example, is a power management technique used to improve power efficiency, reducing the clock frequency and the supply voltage of the processor [215]. DVFS can cause data cells to be stuck with a specific value because it diminishes the threshold between a logical one and zero. This type of approximation impacts the hardware integrity and the precision of the data. The voltage scaling technique can be applied at the processor architecture level and programmable hardware. At the architectural level, voltage scaling is implemented during the circuit design. Most FPGA manufacturers make the voltage scaling of the device possible through easy-to-use design tools. Even though it will impact the performance of a software application, it is not part of the software approximation group because its implementation has no direct connection with software development.

Using Inexact/Faulty Hardware. Inexact and faulty hardware can be used at the architecture level to provide an approximation. The literature presents a multitude of approximate adders proposals. One approach is to remove the circuit's carry chain to reduce delay and energy consumption. This can be done by altering the subadders of a standard adder cell of n bits [216]. In [217], the authors presented an approximate 2×2 multiplier design that gives correct outputs for 15 of the 16 possible input combinations and uses half of the area of a standard non-approximate multiplier.

As we can see, approximation techniques can be implemented in all the computation stacks and at many abstraction levels. The functional approximation is an example of a technique applied at the software and architectural computation stacks and implemented via software code modification, programmable hardware, and circuit level. Loop perforation can also be achieved via code modification for embedded software and programmable hardware (high-level synthesis, HLS) or directly with HDL project modifications. The way the approximation techniques are technologically implemented also considerably impacts their performance. Approximate computing at the software level is less presented in the literature than at the hardware level. This is probably due to the origins of approximation being on energy consumption reduction and neural network applications.

4.3.3 More Computing Power

Supercomputers in the near future will be different from today's HPC systems, because the process towards more computing power will need a different approach from the traditional one of *"scaling out by just adding more building blocks"*. This new attitude will need to rely on innovation across the software stack, considering three important key challenges: *i)* performance at scale, *ii)* energy efficiency, and *iii)* resilience.

In fact, an exascale supercomputer will not be *"yet another big machine"*. With a cost of hundreds of million euros, power consumption in the order of tens of megawatts and a lifetime that reaches a decade at most, the management of those resources must be wise and at the same time very accurate, especially in these days, when the usage of computational resources is increasingly related to reduced energy consumption, maintaining the same performances and the same execution times (or most).

But even with the highest technological advancements, a post-exascale machine is expected to well exceed the 20-30MW threshold that is the current upper bound of power consumption for exascale computing, and could be even more than 30MW. A machine of this size will not be able to operate at full power consumption, and energy consumption will become a primary concern to keep its environmental footprint and operational costs at acceptable levels, without neglecting its original purpose: that is to provide computational capacity to highly critical applications, to solve extremely resource hungry problems.

Problems that, focusing on the application side, pose big efforts to achieve scalable performance and high system throughput. To make things even more challenging, next-generation HPC applications can no longer be considered as computation or communication bound, thinking of them as monolithic blocks with fixed and precise schemes.

The revolution of big data and machine learning, the emerging of edge Computing and Internet of Things (IoT), with the scale of modern HPC systems and cloud data centers, are rapidly changing the way we solve scientific problems. Novel computational patterns are rapidly evolving, where the solution of a problem may require a workflow of diverse tasks like: performing simulations, data ingestion, data analytics, machine learning, visualization, uncertainty quantification, verification, computational steering, and more. Existing solutions may render the execution of such applications in a large-scale supercomputer either impossible, or extremely suboptimal in terms of time to solution, due to the absence or inefficiencies of appropriate methods to compose, deploy and execute workflows and due to their extreme requirements in I/O resources, which cannot be met by the system capacity without holistic and sophisticated deployments [218].

That's why collaborations, projects and new frameworks and software are emerging every day, to address these issues and challenges. Related to this topic, at CINECA, we are using two software, developed respectively by UniBO and CINECA itself. They are **ExaMon** [219] and **COUNTDOWN** [220].

ExaMon (Exascale Monitoring) is a data collection and analysis platform oriented to the management of big data. Its main prerogatives are: *i)* to manage in a simple way heterogeneous data, both in streaming and batch mode, and *ii)* to allow the access to these data through a common interface. This simplifies the usability of data, supporting applications such as real time anomaly detection, predictive maintenance and efficient resource and energy management, using techniques in the domain of machine learning and artificial intelligence. Given its scalable and distributed nature, ExaMon is readily applicable to HPC systems, especially exascale sized ones, which is also the primary use case it was designed on.

Instead, **COUNTDOWN** is a run-time library for application-agnostic energy saving, focusing MPI communication primitives. In a nutshell, **COUNTDOWN** is a methodology and a tool for identifying and automatically reducing the power consumption of the computing elements during communication and synchronization primitives, filtering out phases which would detriment the time to solution of the application. Moreover, it is also an MPI tracer and a performance profiler used to collect and report information on the application execution. This is done transparently to the user, without touching the application code nor re-

quiring recompilation of the application. Its low overhead has values in the range that we expect for profiling tools with performance counters (like Linux Perf) and multiplexing enabled.

As previously said, in fact, the most important thing to acquire, in future HPC systems, is to let the users run their applications without degrade their time to solution, while reducing the energy consumption. In modern CPUs, architectures are rapidly changing, and to understand if an application is performing efficiently, we have to consider a lot of aspects, like the efficiency of the: superscalarity, out-of-order execution, access to multi (and different) level caches, parallellism, and power consumption. And to tackle all these potentially issues, we need specific tools and software, to get access to the underlying hardware, automatizing routines for HPC users.

4.4 Performance

Performance analysis of High Performance Computing (HPC) applications is a critical aspect to ensure the efficiency of computations and meet project deadlines. Performance models are used for this purpose, which allow evaluating the performance of an application based on its characteristics and hardware properties. Classical performance models include Amdahl's model, which evaluates the performance of an application based on the degree of parallelism used. This model helps to identify the parts of the application that cannot be parallelized and may represent a bottleneck for overall performance. Gustafson's model is another classical performance model, which evaluates the performance of an application based on the size of the problem. In this case, the performance of the application is evaluated based on the time taken to solve a problem of variable size, rather than based on processing speed.

4.4.1 Application Targeted Benchmarks

In addition, there are other models that go beyond the simple concept of parallelism and consider other aspects that have a significant impact on performance, such as memory access. Such models are more suitable and specific for evaluating the performance of an HPC application.

The Roofline Model is a graphical performance model used in HPC to visualize the performance limits of a given hardware system. The model plots the performance achieved by a computation as a function of operational intensity, which is the ratio of arithmetic operations performed to the data movement between the processor and memory. The roofline model is useful for identifying performance bottlenecks and optimization opportunities, such as improving memory access patterns or increasing arithmetic intensity [221].

The Cache-Aware Roofline Model is an extension of the Roofline Model that takes into account the performance impact of caching. It provides a more detailed view of the system's performance limits by including the effect of caching on the memory bandwidth available to the computation. By considering the cache hierarchy, this model enables more precise analysis and optimization of the performance of HPC applications [222].

The Stencil Model is a performance model used specifically for applications that use stencil operations. Stencil operations are a common computational pattern used in scientific computing applications, such as simulations of fluid dynamics, heat transfer, and other physical phenomena. The stencil model characterizes the performance of these applications by analyzing the memory access patterns and computational requirements of stencil operations. This model is useful for identifying optimization opportunities, such as improving the locality of data access or reducing redundant computation[223].

4.4.2 Hardware Targeted Benchmarks

While the previous models are aimed at measuring the performance of an application, it is also important to remember some methodologies aimed at measuring the performance of an HPC computing system through specific benchmarks.

Linpack: This model uses a matrix factorization algorithm to measure the processing speed of the system. Linpack is often used to classify supercomputers based on their performance, using the "teraflops" metric, which represents the number of trillions of floating-point operations the system can perform per second [224].

HPL: The High Performance Computing Linpack Benchmark is a performance test based on Linpack that is used to evaluate the performance of supercomputers. HPL is used as the official metric for ranking supercomputers in the TOP500 list [225].

Stream: This model evaluates system performance in terms of data transfer speed between memory and the processor. Stream is often used to evaluate the performance of the system's memory architecture [226].

NAS Parallel Benchmarks: This model includes a set of intensive computation kernels that are used to evaluate system performance in terms of parallel computing [227].

SPEC: The Standard Performance Evaluation Corporation (SPEC) produces a series of benchmarks that can be used to evaluate system performance in different areas, including HPC. SPEC benchmarks are widely used in the industry to evaluate the performance of hardware systems [228].

4.4.3 Other relevant benchmarks

Other relevant benchmark suites targeting embedded and approximate computing features are also available.

PolyBench is a benchmark suite of 30 numerical computations with static control flow, extracted from operations in various application domains (linear algebra computations, image processing, physics simulation, dynamic programming, statistics, etc.). PolyBench features include:

1. A single file, tunable at compile-time, used for the kernel instrumentation. It performs extra operations such as cache flushing before the kernel execution, and can set real-time scheduling to prevent OS interference.
2. Non-null data initialization, and live-out data dump.
3. Syntactic constructs to prevent any dead code elimination on the kernel.
4. Parametric loop bounds in the kernels, for general-purpose implementation.
5. Clear kernel marking, using pragma-based delimiters.

PolyBench is currently available in C and in Fortran [229].

AxBench is a benchmark suite combined with the NPU compiler (NPiler) intended for use in approximate computing research. It include seven premade benchmarks with the necessary annotations to work with the NPU compilation workflow. The set of benchmarks covers both CPU and GPU applications. AxBench is written in C++ and CUDA, and aims to provide a set of representative applications from various domains to explore different aspects of approximate computing [230].

Finally, the UKMAC consortium collected a wide range of HPC mini-apps developed as part of collaborations with a number of UK based institutions. The GitHub Page also reports several scientific papers showing their value and effective usage for HPC benchmarking [231].

4.5 Temporal Properties

In HPC, the common metric to measure performance is the throughput, expressed in FLOPS (FLoating-Point Operations Per Seconds). In recent years, new application domains requiring an HPC infrastructure to run are emerging. These include, for instance, use cases from automotive, smart city, healthcare, environmental, and infrastructure monitoring. Application requirements in such domains include specific timing constraints, similar to real-time applications running on embedded systems. The use-cases of two past EU projects, MANGO [232] and RECIPE [233], included applications that have low-latency requirements, stricter than a simple *averaged* throughput. Such applications included video-related applications, including medical imaging, network packet management, and weather forecast and environmental monitoring for disaster predictions.

4.5.1 Mechanisms for isolating workloads

The isolation of workloads was considered a crucial method to prevent the propagation of misbehaviors. One key notion was that of *supply bound function* $\mathsf{sbf}(t)$, which models the minimum amount of processing available in any interval of length t [234]–[236].

The notion of supply function was then extended to contexts with parallelism, notably multiprocessors and distributed systems.

On multiprocessors, Bini et al. [237] proposed a family of supply functions each one for every level of possible parallelism. Later, such a general abstraction was simplified into the *generalized multiprocessor resource model* (GMPR) to make it more simply implementable through periodic servers [238].

In the context of critical applications, a lightweight hypervisor was developed for embedded systems in which the guest OS is aware of running in a virtualized environment [239].

4.5.2 Workload isolation in presence of mixed-criticality applications

Ekberg and Yi [240] determined the demand bound function of a set of mixed-criticality real-time tasks to be scheduled by EDF. The main drawback of mixed-criticality is the penalty when scheduling low criticality tasks. For this reason, adaptive schemes were proposed [241], [242] to mitigate this issue.

Mixed-criticality was also extended to the case of parallel machines [243]. For example, MC-fluid [241] is an algorithm that assigns shares of processors to mixed-criticality tasks as function of their criticality too.

Methods to re-distributed unused budgets among tasks of different criticality were also proposed [244].

4.5.3 Probabilistic-WCET Estimations in HPC

Determining the Worst-Case Execution Time (WCET) of the tasks in HPC clusters is extremely difficult due to many sources of temporal non-determinism present in modern hardware and to the high complexity of such systems. A possible solution is to use measurement-based techniques, that infers the WCET from the observation of the execution time rather than performing a static analysis of the software and hardware. The use of probabilistic techniques to obtain the probabilistic-WCET (pWCET) in embedded systems dates back to 2001 [245] and two surveys [246], [247] recap all the works in the last years. A preliminary study on the use of pWCET for HPC has been published in 2020 [248]. How to design the computing platform and the HPC cluster as a whole is an open problem.

4.5.4 Intra-core interferences

Hyper-Threading (HT) in HPC cluster is very often disabled, due to the not simple performance implication. Indeed, not all applications are able to exploit the hyper-threading, and for some it can have negative effects on the performance [249]. In cloud computing, HT is generally beneficial as noted by Zhang et al. [250]. However, the same authors recognized that CPU-intensive and IO-intensive workload may be penalized by HT. Several research works on real-time embedded systems investigated the interference between workload. One example is the probabilistic copulas approach by Bernat et al. [251].

4.5.5 Impact of the Linux kernel on time-critical software

The Linux OS has become very complex during the years, in order to support a large number of hardware devices and to provide several features to the software. Moreover, Linux is a general-purpose OS and, therefore, focused more in providing high throughput rather than low latencies. The PREEMPT_RT kernel patch [252] has been developed to reduce such latencies and its main feature is to allow the preemption inside kernel spinlocks. The `osnoise` tool [253] has been recently developed to measure such latencies and their sources. How the patch and Linux latencies in general impact on HPC time-critical software needs further investigation.

4.5.6 The timing problem of fault tolerance in HPC

In HPC, being resilient to transient and permanent faults is a necessary properties. Very often, such resilience is implemented via software techniques, such as checkpoint/restart [254] or proactive workload migration [255]. These techniques have a strong impact on the timing of application, in both normal execution (e.g., the checkpoint overhead) and the time required to restore the execution after a fault. For real-time workload, such overheads are potentially disruptive of the timing constraint satisfaction and requires novel techniques to handle such faults.

4.5.7 Open challenges

Guaranteeing temporal determinism and bounded execution times in high-performance architectures is a challenging issue at all levels of the computing continuum, from small embedded systems to HPC clusters. Resource management and scheduling algorithms play a key role in the satisfaction of the temporal requirements of the software tasks. We can summarize the open challenges in the following research topics:

- Modeling the intra- and inter-core interferences of the workload so that timing upper-bounds can be provided.
- Verifying the limits of partitioned systems and their positioning with respect to the alternative mixed-criticality theory.
- Understanding the implications of using general-purpose software (such as Linux OS) when running time-critical workload.
- Correctly managing the workload having fault-tolerance mechanisms in place, so that timing requirements are still met.
- Developing and novel WCET estimation methods and improving the existing ones.

5 SOFTWARE SUPPORT

This section provides a state-of-the-art analysis related to software supports for HPC computing applications and hardware platforms. The first part, described in Sect. 5.1, introduces a deep description of techniques to make parallel applications self-adaptive and self-aware, and capable of reacting to dynamic workloads. In the context of Federated Learning systems, a quite popular and hot research topic to train distributed ML/AI models with privacy constraints, a comprehensive analysis of Federated Learning applications are provided in Sect. 5.2, with special emphasis on support tools enabling Federated Learning on highly distributed and parallel environments. Finally, the last part in Sect. 5.3 describes the compiler supports available for precision tuning. The section is a deep dive into techniques able to trade floating-point precision and accuracy with performance for a broad range of scientific HPC workloads.

5.1 Runtime Management

In the mid-00's, Autonomic Computing [256] emerged as a promising paradigm to address the various needs of complex applications and systems to deal with changing execution conditions, workloads and user requirements. The general goal was the one of designing systems with the ability to expose different *self-* properties*, such as self-adaptation, self-healing, self-protecting, self-optimizing and further on. A natural application of this paradigm was related to the control of parallel and distributed applications, through runtime systems capable of adapting several non-functional properties (e.g., performance and energy consumption) of running applications as a response to dynamic and hardly predictable workloads. This can be done by setting up a so-called *MAPE control loop* (Monitor-Analyze-Plan-Execute) consisting of: *i)* proper sensors and monitoring facilities of non-functional properties; *ii)* an adaptation logic (e.g., ML/AI-driven or based on logic formalisms) that derives periodic control actions; *iii)* a set of adaptation knobs, i.e., some actuators that can be used to change the non-functional behavior of the system through reconfigurations. In this section we describe the most relevant techniques to make parallel applications autonomic, and we briefly review some of the research frameworks supporting self-adaptive parallel programming.

Hardware monitoring. Multi-core architectures expose different performance monitoring mechanisms that often rely on hardware counters, such as the number of instructions issued per time unit (IPS) [257]. However, correlating the IPS (or other available counters) to the actual performance exhibited by the application can be hard for end users. Furthermore, the direct use of hardware counters poses portability issues, since not all the performance counters are available in all multi-core platforms. A better solution is to monitor the level provided by the application using high-level metrics closer to the actual *Quality of Service* or *Experience* (QoS/QoE) (e.g., monitoring the service time per request, the end-to-end latency, and so forth). Such high-level monitoring data can be transmitted by the application to a manager component (within the application or as a separate process) by instrumenting the application itself [258]. Although this approach is more intrusive, it can be done by adding few monitoring primitives inside the code. Analogously, this approach can be extended to monitor different non-functional properties such as power consumption, which can be done by using external monitoring devices connected to the monitored platform or, rather, by using hardware counters available in some architectures. This has led to a wide spectrum of monitoring tools [259], [260] that can be used by instrumenting the application source code in a quite user-friendly manner.

Actuators and system knobs. In modern multi-core platforms, different reconfigurability options and techniques (also called *system knobs*) are available. One is *thread placement*, applied through pinning directives that can be used to map threads onto cores in a highly configurable manner. This may have remarkable

effects in the memory hierarchy exploitation [261] and can be used to control power consumption through *thread packing* [262], i.e., by dynamically packing threads onto a smaller set of cores while shutting down unused cores to save energy. Furthermore, it is an effective approach to control the interference of Operating System activities, which is often referred to as *OS noise* in the literature [263]. Another aspect is the way used by a parallel runtime to implement *synchronization primitives* (e.g., mutexes, conditional variables and others), which has been historically named *concurrency control* in the literature [264]. As a matter of fact, switching between busy-waiting primitives, which are highly responsive but power hungry, and passive waiting ones, which in contrast are energy saving but impaired by context switching overheads, plays a remarkable role to trade performance with energy efficiency. To cope with dynamic workloads, *concurrency throttling* [265] can be used to adapt the parallelism degree of the application (i.e., number of actively running threads). Since dynamically restructuring the design of generic applications in terms of threads and their interaction can be a quite hard task, this technique plays nicely together with the Algorithmic Skeletons [266] methodology, which fosters the design of parallel applications as composition of *parallel patterns* whose parametric structure allows the degree of parallelism to be shrinked or enlarged quite easily. Furthermore, modern multi-core CPUs provide *Dynamic Voltage and Frequency Scaling* (DVFS) supports to change the operating frequency (and consequently the voltage) of the CPU during program execution. Although this can be seldomly applied to individual cores (but rather to groups/islands of sibling cores or even entire CPUs), DVFS can be relevant system knob available to parallel runtime environments to meet desired trade-offs between performance and energy consumption [267].

Self-* autonomic parallel runtimes. One successful experience that brings together Autonomic Computing and Algorithmic Skeletons was the so-called Bahavioural Skeleton approach [268], which was one of the main results of the EU funded project GridCOMP. The idea underpinning this approach was to extend the core concept of parallel skeletons (e.g., farm, pipeline, map, divide-and-conquer, stencils) to incorporate a self-optimizing/self-adaptive manager component capable of adapting the behavior of the monitored skeleton implementation to achieve desired non-functional goals from the overall execution (e.g., reaching a desired throughput level, or keeping the end-to-end latency below a threshold). To do that, the manager dynamically adapts the parallelism degree of the parallel program, enforcing the parametric definition of the used parallel patterns. The adaptation logic was implemented through Event-Condition-Action (ECA) rules as in active databases. The Behavioural Skeleton approach was a pioneering research whose most significant fallout was to demonstrate that the exposition of the parallel structure of the program enables programmers to include in the MAPE loop notable rules and efficient control policies with reduced design and engineering effort.

More recently, idea of Behavioural Skeletons has been applied to the FastFlow [269] parallel programming framework. FastFlow provides the programmer with two main abstractions layers. The first one is composed by *high-level parallel patterns*, which essentially represent Algorithmic Skeletons that can be combined and nested to build the logic of a complex parallel program. The second level is related to *parallel building blocks*, which represent lower-level components that can be used to build a directed graph of concurrent/parallel entities executed by independent threads cooperating by pointer-sharing through lock-free Single-Producer Single-Consumer queues. FastFlow provides several kinds of autonomic mechanisms with the potential of restructuring the parallel program implementation at runtime in order to fulfill a desired QoS/QoE. In terms of concurrency control, FastFlow can dynamically switch between busy-waiting synchronization (i.e., in accessing shared queues) and more conservative suspension-based primitives. Furthermore, it provides a user-friendly approach to control thread pinning of the application on the underlying cores of the machine. In addition, FastFlow threads can be freezed and activated programmatically by sending special messages through lock-free queues. This allows self-adaptive mechanisms able to control the parallelism degree of the computation, to balance CPU utilization with performance and energy consumption.

The idea of dynamically reconfiguring parallel programs has been generalized by the Nornir C++ framework [260] , which aims at providing an effective support to develop adaptive parallel programs on shared-

memory systems. Nornir is capable of managing structured (based on skeletons such as FastFlow parallel programs) or even unstructured parallel programs (e.g., written using other paradigms such as task-based parallelism). It allows the developer to easily integrate a manager component in the application, to setup an adaptation logic, and to steer the target non-functional parameters by using a set of default knobs such as thread pinning and DVFS when available and supported by the multi-core platform.

In this landscape, autonomic parallel computing is still an evolving research field. The need of dynamically reconfiguring parallel applications in an effective manner is a challenging problem exacerbates in case of long-running applications, such as the ones in the stream processing domain. As described in a recent survey [270], self-adaptation strategies and techniques should address important features that are not adequately supported by existing systems and tools. One direction is to support self-adaptation in scenarios where the application has been decomposed in more than one parallel pattern or skeleton, possibly distributed in more machines. Decentralizing the adaptation logic and the MAPE loop appears a strong requirement in this context. Furthermore, structural changes of the applications, which might change the adopted composition of patterns dynamically, have been rarely studied in the literature. Such advanced reconfigurations might produce significant improvements although possibly impaired by the need of a more complex set of runtime mechanisms to enable a smooth switching from a skeleton to another one at runtime. Last but not least, self-adaptation supports should also address the transparent offloading to accelerators of different kinds (e.g., GPUs and FPGAs), which can be profitable used to achieve better balancing between performance and energy consumption in a broad space of platforms.

5.2 Support for Decentralized Machine Learning

Decentralized Machine Learning (DML) enables collaborative machine learning without centralized input data. Federated Learning (FL) and Edge Inference (EI) are the most prominent examples of DML.

5.2.1 Federated Learning

Federated Learning has been proposed by McMahan et al. [271] as a way to develop better ML systems without compromising the privacy of final users and the legitimate interests of private companies. Initially deployed by Google for predicting the text input on mobile devices, FL has now been adopted by many other industries such as mechanical engineering and health-care [272].

FL is a learning paradigm where multiple parties (*clients*) collaborate in solving a learning task using their private data. Importantly, each client's local data is not exchanged or transferred to any participant. In its most common configuration, instead of moving the data, clients collaborate by exchanging their local models. The *aggregator* collects the local models and aggregates them to produce a global model. The global model is then sent back to the clients, who use it to update their local models. Then, using their private data further update the local model. This process is repeated until the global model converges to a satisfactory solution or a maximum number of rounds is reached.

There are two main federated learning settings: cross-device and cross-silo [273]. In cross-device FL, the parties can be edge devices (e.g., smart devices and laptops), and they can be numerous (order of thousands or even millions). Parties are considered not reliable and with limited computational power. In the cross-silo FL setting, the involved parties are organizations; the number of parties is limited, usually in the $[2, 100]$ range, and rich in resources ranging from server rooms to HPC centers [274]. Given the nature of the parties, it can also be assumed that communication and computation are no real bottlenecks.

Since its introduction, FL has mainly exploited the inner workings of neural networks and other gradient descent-based algorithms by exchanging the model weights or the gradients computed during learning. While this approach has been very successful, it rules out applying FL in contexts where other models would be

preferred, either because they are more interpretable or because they are known to work better. For instance, in the case of medical studies, data often comes in tabular form; examples are not numerous and distributed among medical centers that need to respect hard privacy constraints. Also, medical doctors often require to be able to interpret the inferred models. In these situations, decision trees or rule-based systems are often preferred to neural networks; however, they cannot be readily applied without collecting the data in one single place (e.g., [275]), which makes the whole process hard or impossible to implement due to the privacy constraints mentioned above. Several industrial sectors can easily benefit from FL techniques, inter alia, automotive, healthcare, and finance.

In the automotive industry, essential facets of vehicle and driver behavior can be inferred from user data transmitted by a fleet of vehicles. However, the quantity of data produced by modern vehicles is substantial (up to hundreds of gigabytes per day [276]), and it is not feasible to collect and store it in a centralized data lake. Furthermore, user data is private, which raises privacy concerns about transferring and storing such data on a central server.

Health systems have been collecting data about patients for decades by many healthcare providers. However, the data is often stored in silos, and health institutions are very cautious in sharing these pieces of information, even with other health institutions. In many cases, a single patient is treated by multiple providers, and the data collected by each provider is insufficient to provide a complete picture of the patient's health. It is then desirable to share the data across different providers to improve the quality of the treatment. FL can solve this problem by allowing different providers to collaborate in training a model that can be used to predict a patient's health.

In many other cases, FL techniques have already proven effective. For instance, it has been used to identify the electricity profile of residential households [277] or for fault diagnosis in mechanical engineering [278].

Different FL frameworks already exist on the market. They are generally agnostic concerning the underlying ML framework used to implement the trainable models, even if, to our knowledge, almost all of them target DNNs, and support the master-worker federation schema only. An exception to the DNN-only rule is the federated AdaBoost algorithm [279], which has been experimented with in a simulated distributed setting. Also, there is still a widespread lack of attention concerning the performance of the distributed infrastructure and communication involved in the Federated process. In this section, we describe the main FL frameworks representing the currently available research-oriented frameworks with their different aims and characteristics.

OpenFL [280] is an open source FL framework developed by Intel®. It is the freely available version of IntelFL, which, in comparison, offers more functionalities, security features, and user support. OpenFL aims for the so-called *cross-silo* scenario, in which the federated training is carried out by a small number of entities that possess a large amount of data and computational power. More specifically, it assumes that the computational infrastructure of these edge nodes is reliable and powerful; thus, this scenario does not deal specifically with issues such as huge scalability and unreliable devices. In this scenario, OpenFL is designed to be secure and privacy-preserving via cryptographed communications and the Intel SGX enclave technology, which should make the federated process cryptographed end-to-end. From a software engineering perspective, OpenFL essentially adopts a master-worker approach, in which the central server acts as the aggregator for the various clients (collaborators); the communication between these two entities is handled by gRPC, which makes this software inherently synchronous. The primary use cases OpenFL aims for are collaborations between hospitals, banks, or insurance companies, in which each entity holds a good quantity of data and has the resources to carry out the FL process.

FLOWER [281], on the other hand, is developed by the University of Cambridge and targets the *cross-device* scenario, in which the FL process is carried out by a large number of small and unreliable devices. This is also the original scenario for which Google first introduced FL: to train a model capable of predicting

the words typed on the smartphone keyboard. It is easy to see that carrying on a FL process on thousands or millions of mobile devices has inherently different challenges than the cross-silo scenario. First, devices are unreliable, not always connected to the network, and provided with low computational power; this sums up to the fact that it is not possible to use all devices simultaneously for the FL process. To deal with that, FLOWER still implements a master-worker approach based on gRPC communications, but focuses more on having light clients, keeping the communication at a minimum, and allowing to sample just a subset of devices at each federated round. This way, FLOWER has been proven to scale up to thousands of devices with little overhead.

Another fundamental open-source FL framework to mention is TensorFlow Federated (TFF). This framework gained popularity mainly due to its intrinsic affinity with TensorFlow and its use in a paper from Sun and many Google researchers, including McMahan, investigating backdoor attacks in the FL setting [282]. The possibility to simulate backdoor attacks directly with the features made available by TFF is indeed one of the distinctive features of this framework, making it particularly suitable for FL research oriented towards security and privacy, also due to its interoperability with the TensorFlow Privacy library, which implements many Differential Privacy algorithms.

Moving on to the non-open-source software, IBM Federated Learning [283] is a proprietary FL framework from IBM capable of supporting four different ML libraries: TensorFlow, PyTorch, SciKit-Learn, and XGBoost. This framework is the only one reviewed here to support the federation of non-deep ML models. However, since the framework code is unavailable, it is unclear how it handles these federated models not based on gradient descent. Some recent work aims to understand how this is done [279], but the proprietary nature of this software makes it impossible to investigate this question further.

Another closed-source FL framework developed by a hi-tech giant is NVIDIA Clara. Strictly speaking, this is not exclusively an FL framework but a platform containing various libraries and frameworks for applying AI to the medical domain. However, its FL features have recently been successfully applied [284] to a mobile healthcare use case. NVIDIA Clara is an example of a domain-specific FL framework. The software design takes advantage of a series of choices, like the focus on healthcare, the optimization for NVIDIA accelerators, and the interconnected software environment, to find its strength not in generality, but specificity.

Swarm Learning [285] is an FL framework developed by HP with particular stress on a peer-to-peer structure of the federation. This decentralized structure and the Blockchain authenticated exchanges between peers assure high-security standards and fault tolerance capabilities. Swarm Learning is the only FL framework in this brief review that is not based on the Master-Worker approach.

Lastly, workflow managers have been used to implement FL architectures. [274] uses StreamFlow [286] to run an FL pipeline described using the Common Workflow Language. The system tested on a federation involving two HPC centers demonstrates that hybrid workflows are ready to provide adequate performance in the FL field, enabling cross-HPC FL as a pluggable step in modern large-scale simulation pipelines.

5.2.2 Edge Inference

Inference in Neural Networks is much more lightweight than training. Relative performance varies significantly between network architectures (e.g., convolutional networks tend to pay a slightly higher cost during training) and between system architectures (e.g., executing the operations on GPU may change the relative cost of training and inference). Roughly speaking, one may expect the inference cost to be about 1/3 of the cost needed for performing a single training step of the same model on similarly sized data [287] due to additional loss and backward pass requires in the training phase. For this, the inference is generally computed on a single (possibly SIMD/GPU/TPU accelerated) processing element. However, use cases consisting of a network using ML both in the edge devices, co-located with data sources, and in several relay nodes that belong to the data and control plane of the network also emerged. These use cases, termed here *Edge Inference* (EI),

aim to distribute and adapt the inference load and complexity to the different device capabilities. In the data plane, relay nodes can be used to further process, filter, or coalesce stream items, whereas in the control plane they can be used to tackle reliability issues, such as shaky network conditions, churn of peers, and device battery conditions.

5.2.3 RISC-V Support

Most of the above frameworks rely on the Python programming language to offer a familiar environment to data scientists and delegate the heavy computations to heavily specialised libraries, e.g., Pytorch [288], TensorFlow [289], Scikit-learn [290] which rely on C/C++-based libraries to optimize the compute performance. Consequently, these libraries need to be adapted to the different underlying hardware platforms to express their full potential. Only recently, such libraries have been ported to the RISC-V ISA. Most notably [291] provides the first publicly available RISC-V port of PyTorch able to run on commercially available RISC-V silicon. Looking into the future, **bsc-onednn** rewrites performance critical functions of the OneDNN library (used among others by PyTorch) to leverage the standardized, but not yet available, RISC-V extended vector instructions. Differently, [292] proposes to build an accelerator based on RISC-V cores for (deep) ML tasks.

5.3 Compiler Support for Mixed Precision Computing

Many scientific applications can benefit in terms of performance and energy efficiency from reduced precision calculations [293], [294]. However, the problem of finding the precision mix that satisfies the accuracy requirements while providing the maximum performance is not trivial. As such, automated end-to-end solutions that can perform this process are necessary.

Precimonious [210] is a precision tuning tool that works with C/C++ source code and outputs the suggested type changes in a json file. It uses delta-debugging [295] search algorithm to find a precision mix that has better performance while maintaining enough accuracy. Precimonious uses dynamic analysis to verify that the precision mix satisfies the requirements, which depends on having a representative dataset. Precimonious only supports IEEE-754 floating-point types, which limits its use.

CRAFT [296], [297] is a source-to-source precision tuning tool that works with C/C++ code. It uses binary search to determine the precision required at the given program level. It goes through the modules, functions, basic blocks, and individual instructions in a breadth-first search fashion to refine the precision mix. CRAFT uses dynamic analysis to verify that the precision mix satisfies the requirements, which depends on having a representative dataset. The tool can potentially work with OpenMP. CRAFT only supports IEEE-754 floating-point types, which limits its use.

FloatSmith [298], [299] is a source-to-source precision tuning tool that is based on CRAFT [296] and that works with C/C++ code. FloatSmith integrates ADAPT [300] to narrow the search space for CRAFT using static analysis. It uses CRAFT to further optimize the precision mix using different search strategies: combinational, compositional, delta-debugging, hierarchical, hierarchical-compositional, and Genetic Search Algorithm. FloatSmith uses dynamic analysis to verify that the precision mix satisfies the requirements, which depends on having a representative dataset. The paper reports a successful test with OpenMP version of LULESH benchmark [301]. FloatSmith does not support fixed-point types, which limits its use.

GeCoS + ID.Fix [302] is a source-to-source precision tuning tool that works with C/C++ code and targets generic hardware platforms. It uses static analysis technique called value range propagation to infer the value range of dependent variables based on user-annotated variables. However, it mostly focuses on floating point to fixed point conversion to minimise the number of bits used during computation. Additionally, it does not consider the possibility of a mixed precision output, with floating point and fixed point data types coexisting in the same program.

Daisy [303] is a precision tuning tool that targets generic platforms, supports fixed-point types, and provides formal guarantees on the result precision. It uses a combination of mixed-precision tuning with delta-debugging algorithm and rewriting with a genetic algorithm to reduce the roundoff error. Daisy uses a static error analysis with interval arithmetic and SMT [304], and a static heuristic performance cost function. Unfortunately, Daisy requires the program to be written in a Scala-based domain-specific language, and only supports optimization of arithmetic kernels without conditionals or loops, which makes it unsuitable for optimizing programs that use OpenMP.

TAFFO [305] is a precision tuning tool based on LLVM [306] for optimizing C/C++ programs. This paper introduces in TAFFO support for inter-procedural precision tuning of the programs parallelized with OpenMP [307]. TAFFO is a precision tuning tool with user-defined scope based on variable annotations. It performs static code analysis using user-provided range values to infer the algorithm properties and the affected variables and statically validates the effect of the precision tuning step on the target values. It also provides formal guarantees about error magnitude for programs without unbounded loops and gives an estimate when unbounded loops are present. It controls the overhead introduced by the type casting operators [308]. TAFFO is built as an LLVM pass and uses LLVM-IR as its input and output, so it can support a wide variety of programming languages, although it is mainly targeted at programs written in C/C++. It supports optimization using IEEE-754 [309] floating-point, and dynamic fixed-point types with a focus on general-purpose computing platforms.

For the more detailed overview of the field we refer the reader to the recent surveys. Cherubin and Agosta [310] surveys the software tools used at the different stages of precision tuning. Stanley-Marbell et al. [294] introduces unified terminology for quality versus resource usage tradeoffs. It also surveys the field categorizing both software and hardware approaches used on the different levels of the computing stack.

6 CONCLUDING REMARKS

This document overviews the current state-of-the-art about HPC systems and highlights the major challenges and also opportunities to be faced in the design, integration, operation and monitoring of modern and future generation of HPC machines. The manuscript focuses on the issues of non-functional properties, such as power budget, thermal efficiency, and reliability, to address possible solutions, opportunities and research challenges in the HPC domain with special emphasis on current solutions and alternative approaches, such as the RISC-V processor architecture.

In general, modern and future generations of HPC machines face several design and operative challenges, including critical performance and energy efficiency targets. Moreover, other concerns, such as the dependability features of the hardware components (e.g., premature aging caused by hardware faults due to transistor technology integration), or system scheduling management (involving issues of thermal and power budgets) must be considered as integral part in the design and operation of HPC systems. Interestingly, in all steps of design and operation, the management of non-functional properties plays an important role in the final operation of the system.

In fact, the current HPC scenario needs improvements in several areas. In particular, we analyzed and discussed the two major design challenges of modern HPCs: *i)* Performance and *ii)* Energy efficiency. Moreover, we highlight how RISC-V-based platforms can be used as feasible alternative due to their improved instruction set architecture, design flexibility and industrial support.

Modern RISC-V architectures, hardware accelerators, memory, and interconnect infrastructures involve several major challenges in terms of design, integration and standardization for commodity clusters for HPC machines. Interestingly, design factors, such as the ISA, architecture, and memory hierarchy might contribute to solve performance challenges, such as memory performance bottleneck *(memory-wall)* in all the HPC infrastructure. In addition, approaches of optimized operation and resource management are crucial for the cluster's integration.

In literature and industry, several trends and challenges in terms of hardware architecture for accelerators are highlighted. In fact, the memory bottleneck still plays a major concern in the design of new accelerators with near- and in-memory computing strategies to optimize the performance of current and future accelerators generations. The exploration of numeric formats in the HPC domain is still vastly unexplored. Finally, due to the complexity of the current hardware accelerators, flexible, programmable, and effective programming environments are demanded to handle the internal hardware complexity but allow the adaptation in performance to different workloads or tasks. Moreover, hardware topology play an important role in the design of modern hardware accelerators for specific and general workloads (importance of spatial, e.g., systolic arrays, SIMD, and long vector-processor). In other structures, such as memory and interconnect infrastructures, reliability challenges caused by technology scaling integration involve the analysis and the proposal of new techniques and mechanisms.

In particular, HPC systems face several non-functional properties that affect their design, and operation. In fact, we identified several challenges and research opportunities for current and future generations of HPC systems in terms of reliability, thermal and power efficiency, performance and temporal properties.

In terms of reliability, we identified the need of complementary mechanisms to address the pseudo-exhaustive testing of the internal components in commodity clusters, during setup and production stages (e.g., based on effective and compacted software approaches), as well as their main implementation challenges. Interestingly, the adoption of RISC-V architectures might contribute and provides a promising opportunity to develop and evaluated effective solutions in this area. Similarly, in the hardware side, the insertion of hardening

and error-correcting strategies are now standard for many components in the system (e.g., ECC codes in memories and interconnect/communication systems). However, there are still opportunities to improve the resilience of other hardware components by resorting to clever and smart in-field hardware structures to identify and correct faults and failures arising in HPC systems. Potential solutions might resort to complementary combinations of Built-in self-repair, selective hardening mechanisms or the exploration of design diversity solutions.

Furthermore, in terms of performance, the trends suggest that it is of paramount importance selecting a representative set of benchmarks to properly tune the design methodologies for both the hardware platforms and the resource management layer of the software stack. In fact, it is clear that the variability of the real workloads and the heterogeneity of the emerging platforms and of the operating conditions, make it impossible to identify a vanilla solution to support design and evaluation of the proposed solutions.

Regarding efficiency, collaborations, projects and new software are now everyday subject material, to address (and possibly tackle) the hot topic of reducing power and energy consumption, still keeping the same (or even reducing the) time-to-solution of HPC applications, taking into account the new architectural and computational trends. Several metrics need to be taken into account concurrently, considering not only the performance but also encompassing non functional aspects such as the sustainability of the computing and the effectiveness in the use of the electrical power.

The execution over HPC clusters challenges the establishment of temporal properties. Architectural features such as speculative execution, simultaneous multithreading, and different level of caches make hard the estimation of execution times. Among current techniques to mitigate these variabilities, it is worth recalling: Linux `SCHED_DEADLINE` scheduling class which provides strong isolation, mixed-criticality constraint and probabilistic WCET which allows weakening temporal requirements.

Finally, we identified the importance and main benefits of software mechanisms to support the run-time operation and the power management of HPC infrastructures, as well as their interaction with HPC workloads. Indeed, we analyzed the current mechanisms and solutions to trace and set performance and power consumption features at software level (e.g., performance monitors, DVFS, self-adaptive/self-optimizing managers). Similarly, new challenges and opportunities suggest that software and programming support for specific workloads (e.g., machine learning) might demand clever solutions for scheduling and resource management. In fact, the need of flexible and mature tools and frameworks to allow the precision tuning according to the workloads involve the proposal of innovative solutions in the domain, including the management of hardware settings, such as in host and hardware accelerators.

References

[1] S. Panitkin, "Look to the clouds and beyond," *Nature Physics*, vol. 11, no. 5, pp. 373–374, 2015.

[2] G. Poghosyan and S. others, "Architecture, implementation and parallelization of the software to search for periodic gravitational wave signals," *Computer Physics Communications*, vol. 188, pp. 167–176, 2015, ISSN: 0010-4655.

[3] J. Michalakes, "Hpc for weather forecasting," in *Parallel Algorithms in Computational Science and Engineering*, A. Grama and A. H. Sameh, Eds. Cham: Springer International Publishing, 2020, pp. 297–323, ISBN: 978-3-030-43736-7.

[4] D. K. Panda *et al.*, "The mvapich project: Transforming research into high-performance mpi library for hpc community," *Journal of Computational Science*, vol. 52, p. 101 208, 2021, Case Studies in Translational Computer Science, ISSN: 1877-7503.

[5] T. Nakaegawa, "High-performance computing in meteorology under a context of an era of graphical processing units," *Computers*, vol. 11, no. 7, 2022, ISSN: 2073-431X.

[6] K. Sanbonmatsu and C.-S. Tung, "High performance computing in biology: Multimillion atom simulations of nanoscale systems," *Journal of Structural Biology*, vol. 157, no. 3, pp. 470–480, 2007, Advances in Molecular Dynamics Simulations, ISSN: 1047-8477.

[7] B. Schmidt and A. Hildebrandt, "Next-generation sequencing: Big data meets high performance computing," *Drug Discovery Today*, vol. 22, no. 4, pp. 712–717, 2017, ISSN: 1359-6446.

[8] A. Bartolini *et al.*, "Monte cimone: Paving the road for the first generation of risc-v high-performance computers," in *2022 IEEE 35th International System-on-Chip Conference (SOCC)*, 2022, pp. 1–6. DOI: 10.1109/SOCC56010.2022.9908096.

[9] S. Usman *et al.*, "Big data and hpc convergence for smart infrastructures: A review and proposed architecture," in *Smart Infrastructure and Applications: Foundations for Smarter Cities and Societies*, R. Mehmood, S. See, I. Katib, and I. Chlamtac, Eds. Cham: Springer International Publishing, 2020, pp. 561–586, ISBN: 978-3-030-13705-2.

[10] P. Ranganathan, *Plenary keynote - make computing count: Some grand opportunities for testing*, IEEE International Test Conference (ITC 2022), Jun. 2022.

[11] Top500, *The topgreen500*, https://top500.org/lists/green500/2022/11/, Feb. 2023.

[12] Top500, *The top500 project*, https://www.top500.org/lists/top500/2022/11/highs/, Feb. 2023.

[13] A. Armejach *et al.*, "Mont-blanc 2020: Towards scalable and power efficient european hpc processors," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021, pp. 136–141. DOI: 10.23919/DATE51398.2021.9474093.

[14] A. Nocua *et al.*, "Mont-blanc 2020: Simulation efforts towards exascale high performance computing : Embedded tutorial on "ddecs-2020"," in *2020 23rd International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*, 2020, pp. 1–2. DOI: 10.1109/DDECS50862.2020.9095675.

[15] A. Armejach *et al.*, "Using arm's scalable vector extension on stencil codes," *The Journal of Supercomputing*, vol. 76, pp. 2039–2062, 2020.

[16] A. Biagioni *et al.*, "Red-sea: Network solution for exascale architectures," in *2022 25th Euromicro Conference on Digital System Design (DSD)*, 2022, pp. 712–719. DOI: 10.1109/DSD57027.2022.00100.

[17] E. Commission, *European processor initiative*, https://www.european-processor- initiative.eu/, Feb. 2023.

[18] M. Kovač *et al.*, "How europe is preparing its core solution for exascale machines and a global, sovereign, advanced computing platform," *Mathematical and Computational Applications*, vol. 25, no. 3, 2020, ISSN: 2297-8747.

[19] G. Agosta *et al.*, "Towards extreme scale technologies and accelerators for eurohpc hw/sw supercomputing applications for exascale: The textarossa approach," *Microprocessors and Microsystems*, vol. 95, p. 104 679, 2022, ISSN: 0141-9331. DOI: https://doi.org/10.1016/j.micpro.2022.

104679. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0141933122002095.

[20] T.-C. Lu *et al.*, "A cost-effective on-chip power impedance measurement (pim) system in 7nm finfet for hpc applications," in *2021 Symposium on VLSI Circuits*, 2021, pp. 1–2.

[21] Microchip, *Mi-v product*, https://www.microchip.com/en-us/products/fpgas-and-plds/ fpga-and-soc-design-tools/ mi-v, Feb. 2023.

[22] Imagination, *Risc-v imagination cores*, https://www.imaginationtech.com/products/cpu/, Feb. 2023.

[23] Intel, *Risc-v intel core products*, https://www.intel.com/content/www/us/en/products/details/fpga/nios-processor /v.html, Feb. 2023.

[24] SiFive, *Risc-v sifive cores*, https://www.sifive.com/risc-v-core-ip, Feb. 2023.

[25] G. Technologies, *Risc-v gap8*, https://greenwaves-technologies.com/gap8_mcu_ai/, Feb. 2023.

[26] tenstorrent, *Risc-v cores*, https://tenstorrent.com/risc-v/, Feb. 2023.

[27] V. Microsystems, *Ventana risc-v cores*, https://www.ventanamicro.com/technology/, Feb. 2023.

[28] AlibabaCloud, *Alibaba risc-v platform*, https://www.alibabacloud.com/es/press-room/ alibaba-unveils-risc-v-chip-development-platform, Feb. 2023.

[29] K. Roose, "The brilliance and weirdness of chatgpt," *The New York Times*, 2022.

[30] M. Malms, M. Ostasz, M. Gilliot, *et al.*, "Etp4hpc's strategic research agenda for high-performance computing in europe 4," Ph.D. dissertation, European Technology Platform for High-Performance Computing (ETP4HPC), 2020.

[31] P. Kogge and J. Shalf, "Exascale computing trends: Adjusting to the "new normal"' for computer architecture," *Computing in Science & Engineering*, vol. 15, no. 6, pp. 16–26, 2013. DOI: 10.1109/MCSE.2013.95.

[32] J. L. Hennessy and D. A. Patterson, "A new golden age for computer architecture," *Commun. ACM*, vol. 62, no. 2, pp. 48–60, Jan. 2019, ISSN: 0001-0782. DOI: 10.1145/3282307. [Online]. Available: https://doi.org/10.1145/3282307.

[33] A. Dörflinger, M. Albers, B. Kleinbeck, *et al.*, "A comparative survey of open-source application-class risc-v processor implementations," in *Proceedings of the 18th ACM International Conference on Computing Frontiers*, ser. CF '21, Virtual Event, Italy: Association for Computing Machinery, 2021, pp. 12–20, ISBN: 9781450384049. DOI: 10.1145/3457388.3458657. [Online]. Available: https://doi.org/10.1145/3457388.3458657.

[34] D. Black, *Frontier testing and tuning problems downplayed by oak ridge*, https://inside-hpc.com/2022/ 10/frontier-testing-and-tuning-problems-downplayed-by-oak-ridge/, Oct. 2023.

[35] 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.

[36] B. Peccerillo *et al.*, "A survey on hardware accelerators: Taxonomy, trends, challenges, and perspectives," *J. Syst. Archit.*, vol. 129, no. C, Aug. 2022, ISSN: 1383-7621.

[37] A. Poulos *et al.*, "Posits and the state of numerical representations in the age of exascale and edge computing," *Software: Practice and Experience*, vol. 52, no. 2, pp. 619–635, 2022.

[38] I. A. Assir, M. E. Iskandarani, H. R. A. Sandid, and M. A. Saghir, "Arrow: A risc-v vector accelerator for machine learning inference," *arXiv preprint arXiv:2107.07169*, 2021.

[39] Y. Wang *et al.*, "Advancing dsp into hpc, ai, and beyond: Challenges, mechanisms, and future directions.," *CCF Transactions on High Performance Computing*, vol. 3, no. 1, pp. 114–125, 2021, ISSN: 2524-4930.

[40] M. Kovač, L. Dragić, B. Malnar, *et al.*, "Faust: Design and implementation of a pipelined risc-v vector floating-point unit," *Microprocessors and Microsystems*, vol. 97, p. 104 762, 2023, ISSN: 0141-9331.

[41] S. Mach *et al.*, "A 0.80pj/flop, 1.24tflop/sw 8-to-64 bit transprecision floating-point unit for a 64 bit risc-v processor in 22nm fd-soi," in *2019 IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC)*, 2019, pp. 95–98. DOI: 10.1109/VLSI-SoC.2019.8920307.

[42] J. Ouyang *et al.*, "Baidu kunlun an ai processor for diversified workloads," in *2020 IEEE Hot Chips 32 Symposium (HCS)*, 2020, pp. 1–18.

[43] C. Gómez, F. Mantovani, *et al.*, "Hpcg on long-vector architectures: Evaluation and optimization on nec sx-aurora and risc-v," *Future Generation Computer Systems*, vol. 143, pp. 152–162, 2023, ISSN: 0167-739X.

[44] F. Minervini *et al.*, "Vitruvius+: An area-efficient risc-v decoupled vector coprocessor for high performance computing applications," *ACM Trans. Archit. Code Optim.*, Dec. 2022, Just Accepted, ISSN: 1544-3566. DOI: 10.1145/3575861. [Online]. Available: https://doi.org/10.1145/3575861.

[45] Andes-Technology, *Riscv nx27v andes processor*, http://www.andestech.com/en/products-solutions/andescore-processors/riscv-nx27v/, Feb. 2023.

[46] T-head, *T-head c910*, https://www.t-head.cn/product/c910?lang=en, Feb. 2023.

[47] SiFive, *Sifive intelligence x280*, https://www.sifive.com/cores/intelligence-x280, Feb. 2023.

[48] C. Collange, "Simty: Generalized simt execution on risc-v," in *CARRV 2017-1st Workshop on Computer Architecture Research with RISC-V*, vol. 6, 2017, p. 6.

[49] T. Blaise and H. Kim, "Implementing hardware extensions for multicore risc-v gpus," 2022.

[50] Y.-H. Chen *et al.*, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 367–379. DOI: 10.1109/ISCA.2016.40.

[51] V. Sze *et al.*, "Hardware for machine learning: Challenges and opportunities," in *2017 IEEE Custom Integrated Circuits Conference (CICC)*, 2017, pp. 1–8. DOI: 10.1109/CICC.2017.7993626.

[52] T. Nowatzki *et al.*, "Stream-dataflow acceleration," *SIGARCH Comput. Archit. News*, vol. 45, no. 2, pp. 416–429, Jun. 2017, ISSN: 0163-5964.

[53] T. Louw and S. McIntosh-Smith, "Using the graphcore ipu for traditional hpc applications," in *3rd Workshop on Accelerated Machine Learning (AccML)*, 2021.

[54] N. Corporation, *Nvidia deep learning accelerator*, http://nvdla.org/, Feb. 2023.

[55] T. Moreau *et al.*, "A hardware–software blueprint for flexible deep learning specialization," *IEEE Micro*, vol. 39, no. 5, pp. 8–16, 2019. DOI: 10.1109/MM.2019.2928962.

[56] Google, *Composable custom function unit specification*, https://cfu.readthedocs.io/en/ latest/, Feb. 2023.

[57] M. A. Elmohr *et al.*, "Rvnoc: A framework for generating risc-v noc-based mpsoc," in *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, 2018, pp. 617–621.

[58] S. Savas *et al.*, "A framework to generate domain-specific manycore architectures from dataflow programs," *Microprocessors and Microsystems*, vol. 72, p. 102 908, 2020, ISSN: 0141-9331.

[59] F. Merchant *et al.*, "Andromeda: An fpga based risc-v mpsoc exploration framework," in *2021 34th International Conference on VLSI Design and 2021 20th International Conference on Embedded Systems (VLSID)*, IEEE Computer Society, Feb. 2021, pp. 270–275.

[60] A. Kamaleldin and D. Göhringer, "Design for agility: A modular reconfigurable platform for heterogeneous many-core architectures," in *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*, 2021, pp. 265–266. DOI: 10.1109/FPL53798.2021.00050.

[61] D. Giri *et al.*, "Accelerator integration for open-source soc design," *IEEE Micro*, vol. 41, no. 4, pp. 8–14, 2021. DOI: 10.1109/MM.2021.3073893.

[62] A. Amid *et al.*, "Chipyard: Integrated design, simulation, and implementation framework for custom socs," *IEEE Micro*, vol. 40, no. 4, pp. 10–21, 2020. DOI: 10.1109/MM.2020.2996616.

[63] K. Asanovic *et al.*, "The berkeley out-of-order machine (boom): An industry-competitive, synthesizable, parameterized risc-v processor," University of California at Berkeley, Berkeley, United States, Tech. Rep., 2015.

[64] J. Balkind *et al.*, "Openpiton+ ariane: The first open-source, smp linux-booting risc-v system scaling from one to many cores," in *Workshop on Computer Architecture Research with RISC-V (CARRV)*, 2019, pp. 1–6.

[65] Y. Lee *et al.*, "The hwacha vector-fetch architecture manual, version 3.8. 1," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2015-262*, 2015.

[66] Y. Ro *et al.*, "Evaluating the impact of optical interconnects on a multi-chip machine-learning architecture," *Electronics*, vol. 7, no. 8, 2018, ISSN: 2079-9292.

[67] P. Shamis *et al.*, "Ucx: An open source framework for hpc network apis and beyond," in *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*, 2015, pp. 40–43. DOI: 10.1109/HOTI.2015.13.

[68] N. Lodéa *et al.*, "Early soft error reliability analysis on risc-v," *IEEE Latin America Transactions*, vol. 20, no. 9, pp. 2139–2145, 2022.

[69] O. V. Mamoutova *et al.*, "On design of cache with efficient soft error protection," in *2017 IEEE 37th International Conference on Electronics and Nanotechnology (ELNANO)*, 2017, pp. 57–60.

[70] J. Gava *et al.*, "Soft error assessment of cnn inference models running on a risc-v processor," in *2022 29th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2022, pp. 1–4.

[71] R.-V. Foundation, *Risc-v processor cores*, https://riscv.org/risc-v-cores/, Feb. 2023.

[72] A. Waterman *et al.*, *The risc-v instruction set manual, volume i: Userlevel isa, version 2.1*. UCB/EECS-2016-118, UC Berkeley, Tech. Rep. 2016.

[73] V. Sze *et al.*, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017. DOI: 10.1109/JPROC.2017.2761740.

[74] A. Ruospo *et al.*, "On-line testing for autonomous systems driven by risc-v processor design verification," in *2019 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2019, pp. 1–6. DOI: 10.1109/DFT.2019.8875345.

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

[76] *Https://www.iso.org/standard/70939.html*, https://www.iso.org/standard/70939.html.

[77] ANSI/RIA-standards, *Ansi/ria r15.06-2012 industrial robots and robot systems – safety requirements*, https://webstore.ansi.org/Standards/RIA/ANSIRIAR15062012, 2012.

[78] C. Metra *et al.*, "Function-inherent code checking: A new low cost on-line testing approach for high performance microprocessor control logic," in *2008 13th European Test Symposium*, 2008, pp. 171–176.

[79] A. Bosio *et al.*, "A reliability analysis of a deep neural network," in *2019 IEEE Latin American Test Symp. (LATS)*, 2019.

[80] A. Azizimazreah *et al.*, "Tolerating soft errors in deep learning accelerators with reliable on-chip memory designs," in *2018 IEEE International Conference on Networking, Architecture and Storage (NAS)*, 2018, pp. 1–10.

[81] 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.

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

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

[84] 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.

[85] F. Aymen *et al.*, "A new efficient self-checking hsiao sec-ded memory error correcting code," in *ICM 2011 Proceeding*, 2011, pp. 1–5. DOI: 10.1109/ICM.2011.6177346.

[86] I. Boyarinov, "Self-checking circuits and decoding algorithms for binary hamming and bch codes and reed-solomon codes over gf(2m)," *Problems of Information Transmission*, vol. 44, no. 2, pp. 99–111, 2008.

[87] G. R. Redinbo, "Fault-tolerant decoders for cyclic error-correcting codes," *IEEE Transactions on Computers*, vol. C-36, no. 1, pp. 47–63, 1987.

[88] M. Ashjaee and S. M. S.M. Reddy, "On totally self-checkingcheckers for separable codes," *IEEE Transactions on Computers*, vol. C-26, pp. 737–744, 1977.

[89] C. Metra *et al.*, "Self-checking detection and diagnosis of transient, delay, and crosstalk faults affecting bus lines," *IEEE Transactions on Computers*, vol. 49, no. 6, pp. 560–574, 2000.

[90] F. N. Najm, "Equivalent circuits for electromigration," *Microelectronics Reliability*, vol. 123, p. 114 200, 2021, ISSN: 0026-2714.

[91] D. Rossi *et al.*, "Coding scheme for low energy consumption fault-tolerant bus," in *Proceedings of the Eighth IEEE International On-Line Testing Workshop (IOLTW 2002)*, 2002, pp. 8–12.

[92] D. Rossi *et al.*, "Power consumption of fault tolerant codes: The active elements," in *9th IEEE On-Line Testing Symposium, 2003. IOLTS 2003.*, 2003, pp. 61–67.

[93] D. Rossi *et al.*, "Error correcting codes for crosstalk effect minimization [system buses]," in *Proceedings 18th IEEE Symposium on Defect and Fault Tolerance in VLSI Systems*, 2003, pp. 257–264.

[94] M. Omana *et al.*, "Low-cost and highly reliable detector for transient and crosstalk faults affecting fpga interconnects," in *Proceedings of the Design Automation& Test in Europe Conference*, vol. 1, 2006, pp. 1–6.

[95] M. Omaña, S. Govindaraj, and C. Metra, "Low-cost strategy for bus propagation delay reduction," *Journal of Electronic Testing*, vol. 35, pp. 253–260, 2019.

[96] S. Chatterjee *et al.*, "Power grid electromigration checking using physics-based models," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 7, pp. 1317–1330, 2018.

[97] D.-A. Li *et al.*, "A method for improving power grid resilience to electromigration-caused via failures," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 1, pp. 118–130, 2015.

[98] J. Warnock, "Circuit design challenges at the 14nm technology node," in *2011 48th ACM /EDAC /IEEE Design Automation Conference (DAC)*, 2011, pp. 464–467.

[99] Z. Li *et al.*, "A circuit level fault model for resistive opens and bridges," in *Proceedings. 21st VLSI Test Symposium, 2003.*, 2003, pp. 379–384.

[100] U. Ingelsson *et al.*, "Process variation-aware test for resistive bridges," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 8, pp. 1269–1274, 2009.

[101] H. Kim *et al.*, "Process variation-aware bridge fault analysis," in *2016 International SoC Design Conference (ISOCC)*, 2016, pp. 147–148.

[102] M. Favalli and C. Metra, "Pulse propagation for the detection of small delay defects," in *2007 Design, Automation& Test in Europe Conference& Exhibition*, 2007, pp. 1–6.

[103] M. Favalli and C. Metra, "Testing resistive opens and bridging faults through pulse propagation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 6, pp. 915–925, 2009.

[104] S. Muddu *et al.*, "Repeater and interconnect strategies for high-performance physical designs," in *Proceedings. XI Brazilian Symposium on Integrated Circuit Design (Cat. No.98- EX216)*, 1998, pp. 226–231.

[105] C. Metra *et al.*, "On-line detection of logic errors due to crosstalk, delay, and transient faults," in *Proceedings International Test Conference 1998 (IEEE Cat. No.98CH36270)*, 1998, pp. 524–533.

[106] C.-H. Ho *et al.*, "Mechanisms and evaluation of cross-layer fault-tolerance for supercomputing," in *2012 41st International Conference on Parallel Processing*, 2012, pp. 510–519.

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

[108] 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.

[109] B. Schroeder and G. A. Gibson, "A large-scale study of failures in high-performance computing systems," *IEEE Transactions on Dependable and Secure Computing*, vol. 7, no. 4, pp. 337–350, 2010.

[110] P. Radojkovic *et al.*, "Towards resilient eu hpc systems: A blueprint," *European HPC resilience initiative*, 2020.

[111] A. Avizienis *et al.*, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, 2004. DOI: 10.1109/TDSC.2004.2.

[112] M. Omaña *et al.*, "Impact of aging phenomena on latches' robustness," *IEEE Transactions on Nanotechnology*, vol. 15, no. 2, pp. 129–136, 2016.

[113] M. OmaÑa *et al.*, "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.

[114] J. E. R. Condia *et al.*, "Microarchitectural reliability evaluation of a block scheduling controller in gpus," in *2022 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2022, pp. 26–31.

[115] R. K. K *et al.*, "Refu: Redundant execution with idle functional units, fault tolerant gpgpu architecture," in *2022 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2022, pp. 394–397.

[116] M. Omaña *et al.*, "High-performance robust latches," *IEEE Transactions on Computers*, vol. 59, no. 11, pp. 1455–1465, 2010.

[117] D. Rossi *et al.*, "Multiple transient faults in logic: An issue for next generation ics?" In *20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'05)*, 2005, pp. 352–360.

[118] D. Rossi *et al.*, "Analysis of the impact of bus implemented edcs on on-chip ssn," in *Proceedings of the Design Automation& Test in Europe Conference*, vol. 1, 2006, 6 pp.-.

[119] J. Keane *et al.*, "An on-chip nbti sensor for measuring pmos threshold voltage degradation," in *Proceedings of the 2007 international symposium on Low power electronics and design (ISLPED '07)*, 2007, pp. 189–194.

[120] D. Rossi *et al.*, "Impact of bias temperature instability on soft error susceptibility," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 4, pp. 743–751, 2015.

[121] M. Y. 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.

[122] C. Metra *et al.*, "Hardware reconfiguration scheme for high availability systems," in *Proceedings. 10th IEEE International On-Line Testing Symposium*, 2004, pp. 161–166.

[123] F. Fernandes dos Santos *et al.*, "Evaluation and mitigation of soft-errors in neural network-based object detection in three gpu architectures," in *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, 2017, pp. 169–176.

[124] C. Metra *et al.*, "Function-inherent code checking: A new low cost on-line testing approach for high performance microprocessor control logic," in *2008 13th European Test Symposium*, 2008, pp. 171–176.

[125] M. Omaña *et al.*, "Low cost nbti degradation detection and masking approaches," *IEEE Transactions on Computers*, vol. 62, no. 3, pp. 496–509, 2013.

[126] N. DeBardeleben *et al.*, "Gpu behavior on a large hpc cluster," in *Euro-Par 2013: Parallel Processing Workshops*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 680–689.

[127] V. Karakasis *et al.*, "Enabling continuous testing of hpc systems using reframe," in *Tools and Techniques for High Performance Computing*, Cham: Springer International Publishing, 2020, pp. 49–68, ISBN: 978-3-030-44728-1.

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

[129] P. Luszczek *et al.*, "Introduction to the hpc challenge benchmark suite," Apr. 2005.

[130] I. Laguna, "Varity: Quantifying floating-point variations in hpc systems through randomized testing," in *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2020, pp. 622–633.

[131] S. Agarwala *et al.*, "System-level resource monitoring in high-performance computing environments," *Journal of Grid Computing*, vol. 1, pp. 273–289, 2003.

[132] H. Sharifi, O. Aaziz, and J. Cook, "Monitoring hpc applications in the production environment," in *Proceedings of the 2nd Workshop on Parallel Programming for Analytics Applications*, ser. PPAA 2015, San Francisco, CA, USA, 2015, pp. 39–47, ISBN: 9781450334051.

[133] V. G. V. Larrea *et al.*, "Hpc system testing: Procedures, acceptance, regression testing, and automation," in *The International Conference for High Performance Computing, Networking, Storage, and Analysis, SC'19*, 2019.

[134] J. Chuang and H. Krishnaswamy, "19.4 a 0.0049mm2 2.3ghz sub-sampling ring-oscillator pll with time-based loop filter achieving -236.2db jitter-fom," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, 2017, pp. 328–329.

[135] A. Dörflinger *et al.*, "Ecc memory for fault tolerant risc-v processors," in *Architecture of Computing Systems–ARCS 2020: 33rd International Conference, Aachen, Germany, May 25–28, 2020, Proceedings 33*, Springer, 2020, pp. 44–55.

[136] D. A. G. Oliveira *et al.*, "Gpgpus ecc efficiency and efficacy," in *2014 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2014, pp. 209–215.

[137] D. Tiwari *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.

[138] C. Lunardi *et al.*, "On the efficacy of ecc and the benefits of finfet transistor layout for gpu reliability," *IEEE Transactions on Nuclear Science*, vol. 65, no. 8, pp. 1843–1850, 2018.

[139] J. E. R. Condia *et al.*, "Dyre: A dynamic reconfigurable solution to increase gpgpu's reliability," *The Journal of Supercomputing*, vol. 77, pp. 11 625–11 642, 2021.

[140] J. E. R. Condia *et al.*, "A dynamic hardware redundancy mechanism for the in-field fault detection in cores of gpgpus," in *2020 23rd International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*, 2020, pp. 1–6.

[141] I. P. Egwutuoha *et al.*, "A fault tolerance framework for high performance computing in cloud," in *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, 2012, pp. 709–710. DOI: 10.1109/CCGrid.2012.80.

[142] B. Nicolae and F. Cappello, "Blobcr: Efficient checkpoint-restart for hpc applications on iaas clouds using virtual disk image snapshots," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '11, 2011, ISBN: 9781450307710.

[143] I. P. Egwutuoha *et al.*, "A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems," *The Journal of Supercomputing*, vol. 65, no. 3, pp. 1302–1326, 2013. DOI: 10.1109/TDSC.2004.2.

[144] N. El-Sayed and B. Schroeder, "Understanding practical tradeoffs in hpc checkpoint-scheduling policies," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 2, pp. 336–350, 2018.

[145] Y. Huang *et al.*, "Hardening selective protection across multiple program inputs for hpc applications," in *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '22, 2022, pp. 437–438, ISBN: 9781450392044.

[146] K. S. Yim *et al.*, "Hauberk: Lightweight silent data corruption error detector for gpgpu," in *2011 IEEE International Parallel& Distributed Processing Symposium*, 2011, pp. 287–300.

[147] J. M. Wozniak *et al.*, "Mpi jobs within mpi jobs: A practical way of enabling task-level fault-tolerance in hpc workflows," *Future Generation Computer Systems*, vol. 101, pp. 576–589, 2019, ISSN: 0167-739X.

[148] N. Losada *et al.*, "Fault tolerance of mpi applications in exascale systems: The ulfm solution," *Future Generation Computer Systems*, vol. 106, pp. 467–481, 2020, ISSN: 0167-739X.

[149] A. B. Nagarajan *et al.*, "Proactive fault tolerance for hpc with xen virtualization," in *Proceedings of the 21st Annual International Conference on Supercomputing*, ser. ICS '07, 2007, pp. 23–32, ISBN: 9781595937681.

[150] Y. Zhu *et al.*, "Ft-pblas: Pblas-based fault-tolerant linear algebra computation on high-performance computing systems," *IEEE Access*, vol. 8, pp. 42 674–42 688, 2020.

[151] G. Pawelczak *et al.*, "Application-based fault tolerance techniques for fully protecting sparse matrix solvers," in *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, 2017, pp. 733–740.

[152] F. Cappello, "Fault tolerance in petascale/ exascale systems: Current knowledge, challenges and research opportunities," *The International Journal of High Performance Computing Applications*, vol. 23, no. 3, pp. 212–226, 2009.

[153] M. Psarakis *et al.*, "Microprocessor software-based self-testing," *IEEE Design& Test of Computers*, vol. 27, no. 3, pp. 4–19, 2010.

[154] A. Vassighi and M. Sachdev, "Thermal runaway in integrated circuits," *IEEE Transactions on Device and Materials Reliability*, vol. 6, no. 2, pp. 300–305, 2006. DOI: 10.1109/TDMR.2006.876577.

[155] I. Hill, P. Chanawala, R. Singh, S. A. Sheikholeslam, and A. Ivanov, "Cmos reliability from past to future: A survey of requirements, trends, and prediction methods," *IEEE Transactions on Device and Materials Reliability*, vol. 22, no. 1, pp. 1–18, 2022. DOI: 10.1109/TDMR.2021.3131345.

[156] R. Ghaffarian, "Accelerated thermal cycling and failure mechanisms for bga and csp assemblies," *Journal of Electronic Packaging, Transactions of the ASME*, vol. 122, no. 4, pp. 335–340, 2000. DOI: 10.1115/1.1289627.

[157] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ser. ISCA '11, San Jose, California, USA: Association for Computing Machinery, 2011, pp. 365–376, ISBN: 9781450304726. DOI: 10.1145/2000064.2000108. [Online]. Available: https://doi.org/10.1145/2000064.2000108.

[158] J. Kong, S. W. Chung, and K. Skadron, "Recent thermal management techniques for microprocessors," *ACM Comput. Surv.*, vol. 44, no. 3, Jun. 2012, ISSN: 0360-0300. DOI: 10.1145/2187671.2187675. [Online]. Available: https://doi.org/10.1145/2187671.2187675.

[159] A. Leva, F. Terraneo, I. Giacomello, and W. Fornaciari, "Event-based power/performance-aware thermal management for high-density microprocessors," *IEEE Transactions on Control Systems Technology*, vol. 26, no. 2, pp. 535–550, 2018. DOI: 10.1109/TCST.2017.2675841.

[160] F. Terraneo, A. Leva, W. Fornaciari, M. Zapater, and D. Atienza, "3d-ice 3.0: Efficient nonlinear mpsoc thermal simulation with pluggable heat sink models," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 4, pp. 1062–1075, 2022. DOI: 10.1109/TCAD.2021.3074613.

[161] B. Keller, M. Cochet, B. Zimmer, *et al.*, "A risc-v processor soc with integrated power management at submicrosecond timescales in 28 nm fd-soi," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 7, pp. 1863–1875, 2017. DOI: 10.1109/JSSC.2017.2690859.

[162] 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.

[163] 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.

[164] 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.

[165] 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.

[166] C. Metra, L. Schiano, and M. Favalli, "Concurrent detection of power supply noise," *IEEE Transactions on Reliability*, vol. 52, no. 4, pp. 469–475, 2003. DOI: 10.1109/TR.2003.821937.

[167] M. Omaña, A. Fiore, and C. Metra, "Inverters' self-checking monitors for reliable photovoltaic systems," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2016, pp. 672–677.

[168] S. Kim, Y.-C. Shih, K. Mazumdar, *et al.*, "Enabling wide autonomous dvfs in a 22nm graphics execution core using a digitally controlled hybrid ldo/switched-capacitor vr with fast droop mitigation," *IEEE Journal of Solid-State Circuits*, vol. 51, no. 1, pp. 18–30, 2016.

[169] K. Bowman, S. Raina, T. Bridges, *et al.*, "A 16nm auto-calibrating dynamically adaptive clock distribution for maximizing supply-voltage-droop tolerance across a wide operating range," in *2015 IEEE International Solid-State Circuits Conference - (ISSCC) Digest of Technical Papers*, 2015, pp. 1–3. DOI: 10.1109/ISSCC.2015.7062971.

[170] Y. Lin, C. Liu, G. Huang, Y. Shyu, Y. Lui, and S. Chang, "A 9-bit 150-ms/s subrange adc based on sar architecture in 90-nm cmos," *IEEE Circuits and Systems I*, vol. 1, pp. 570–581, 2013.

[171] M. Omaña, A. Fiore, M. Mongitore, and C. Metra, "Fault-tolerant inverters for reliable photovoltaic systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 1, pp. 20–28, 2019. DOI: 10.1109/TVLSI.2018.2874709.

[172] E. Masanet *et al.*, "Recalibrating global data center energy-use estimates," *Science*, vol. 367, p. 984, 2020. DOI: 10.1126/science.aba3758.

[173] S. S. Gill and B. Rajkumar, "A taxonomy and future directions for sustainable cloud computing: 360 degree view," *ACM Computing Surveys*, vol. 51, p. 104, 2018. DOI: 10.1145/3241038.

[174] A. Andrae and T. Edler, "On global electricity usage of communication technology: Trends to 2030," *Challenges*, vol. 6, p. 117, 2015. DOI: 10.3390/challe6010117.

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

[176] 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.

[177] 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.

[178] 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.

[179] 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.

[180] 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.

[181] 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.

[182] W. Liu, F. Lombardi, and M. Shulte, "A retrospective and prospective view of approximate computing [point of view," *Proceedings of the IEEE*, vol. 108, no. 3, pp. 394–399, 2020.

[183] S. Barone, M. Traiola, M. Barbareschi, and A. Bosio, "Multi-objective application-driven approximate design method," *IEEE Access*, vol. 9, pp. 86 975–86 993, 2021. DOI: 10.1109/ACCESS.2021.3087858.

[184] M. Barbareschi, S. Barone, A. Bosio, J. Han, and M. Traiola, "A genetic-algorithm-based approach to the design of dct hardware accelerators," vol. 18, no. 3, Jan. 2022, ISSN: 1550-4832. DOI: 10.1145/3501772. [Online]. Available: https://doi.org/10.1145/3501772.

[185] A. Sampson, A. Baixo, B. Ransford, *et al.*, "Accept: A programmer-guided compiler framework for practical approximate computing," *University of Washington Technical Report UW-CSE-15-01*, vol. 1, no. 2, 2015.

[186] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *Test Symposium (ETS), 2013 18th IEEE European*, IEEE, 2013, pp. 1–6.

[187] X. Wu, X. Zhu, G.-Q. Wu, and W. Ding, "Data mining with big data," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 26, no. 1, pp. 97–107, 2014.

[188] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Neural acceleration for general-purpose approximate programs," *Commun. ACM*, vol. 58, no. 1, pp. 105–115, Dec. 2014, ISSN: 0001-0782. DOI: 10.1145/2589750. [Online]. Available: https://doi.org/10.1145/2589750.

[189] V. Gupta, D. Mohapatra, S. P. Park, A. Raghunathan, and K. Roy, "Impact: Imprecise adders for low-power approximate computing," in *Proceedings of the 17th IEEE/ACM international symposium on Low-power electronics and design*, IEEE Press, 2011, pp. 409–414.

[190] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, "Low-power digital signal processing using approximate adders," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 32, no. 1, pp. 124–137, 2013.

[191] J. Liang, J. Han, and F. Lombardi, "New metrics for the reliability of approximate and probabilistic adders," *Computers, IEEE Transactions on*, vol. 62, no. 9, pp. 1760–1771, 2013.

[192] A. Cilardo, "A new speculative addition architecture suitable for two's complement operations," in *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, Apr. 2009, pp. 664–669. DOI: 10.1109/DATE.2009.5090749.

[193] A. Cilardo, "Modular inversion based on digit-level speculative addition," *Electronics Letters*, vol. 49, no. 25, pp. 1609–1610, Dec. 2013, ISSN: 0013-5194. DOI: 10.1049/el.2013.3467.

[194] A. Cilardo, "Variable-latency signed addition on fpgas," in *Field Programmable Logic and Applications (FPL), 2015 25th International Conference on*, Sep. 2015, pp. 1–6. DOI: 10.1109/FPL.2015.7293957.

[195] G. Rodrigues, F. Lima Kastensmidt, and A. Bosio, "Survey on approximate computing and its intrinsic fault tolerance," *Electronics*, vol. 9, no. 4, p. 557, 2020.

[196] M. Shafique, W. Ahmad, R. Hafiz, and J. Henkel, "A low latency generic accuracy configurable adder," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, IEEE, 2015, pp. 1–6.

[197] A. Momeni, J. Han, P. Montuschi, and F. Lombardi, "Design and analysis of approximate compressors for multiplication," *IEEE Transactions on Computers*, vol. 64, no. 4, pp. 984–994, 2014.

[198] I. Goiri, R. Bianchini, S. Nagarakatte, and T. D. Nguyen, "Approxhadoop: Bringing approximations to mapreduce frameworks," in *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2015, pp. 383–397.

[199] G. Keramidas, C. Kokkala, and I. Stamoulis, "Clumsy value cache: An approximate memoization technique for mobile gpu fragment shaders," in *Workshop on approximate computing (WAPCO'15)*, 2015, p. 2.

[200] S. Sidiroglou-Douskos, S. Misailovic, H. Hoffmann, and M. Rinard, "Managing performance vs. accuracy trade-offs with loop perforation," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, 2011, pp. 124–134.

[201] V. Vassiliadis *et al.*, "A programming model and runtime system for significance- aware energy-efficient computing," *ACM SIGPLAN Notices*, vol. 50, no. 8, pp. 275–276, 2015.

[202] S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Approximate computing and the quest for computing efficiency," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, IEEE, 2015, pp. 1–6.

[203] M. Wirthlin, "High-reliability fpga-based systems: Space, high-energy physics, and beyond," *Proceedings of the IEEE*, vol. 103, no. 3, pp. 379–389, 2015.

[204] A. Ranjan, S. Venkataramani, X. Fong, K. Roy, and A. Raghunathan, "Approximate storage for energy efficient spintronic memories," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, IEEE, 2015, pp. 1–6.

[205] Y. Fang, H. Li, and X. Li, "Softpcm: Enhancing energy efficiency and lifetime of phase change memory in video applications via approximate write," in *2012 IEEE 21st Asian Test Symposium*, IEEE, 2012, pp. 131–136.

[206] Y. Tian, Q. Zhang, T. Wang, F. Yuan, and Q. Xu, "Approxma: Approximate memory access for dynamic precision scaling," in *Proceedings of the 25th edition on Great Lakes Symposium on VLSI*, 2015, pp. 337–342.

[207] M. Sutherland, J. San Miguel, and N. E. Jerger, "Texture cache approximation on gpus," in *Workshop on approximate computing across the stack*, 2015, p. 2.

[208] F. Sampaio, M. Shafique, B. Zatt, S. Bampi, and J. Henkel, "Approximation-aware multi-level cells stt-ram cache architecture," in *2015 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*, IEEE, 2015, pp. 79–88.

[209] C.-C. Hsiao, S.-L. Chu, and C.-Y. Chen, "Energy-aware hybrid precision selection framework for mobile gpus," *Computers & Graphics*, vol. 37, no. 5, pp. 431–444, 2013.

[210] C. Rubio-González, C. Nguyen, H. D. Nguyen, *et al.*, "Precimonious: Tuning assistant for floating-point precision," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '13, Denver, Colorado, Nov. 2013, 27:1–27:12, ISBN: 978-1-4503-2378-9. DOI: 10.1145/2503210.2503296.

[211] J. Pandey, A. Karmakar, C. Shekhar, and S. Gurunarayanan, "An fpga-based fixed-point architecture for binary logarithmic computation," in *2013 IEEE Second International Conference on Image Information Processing (ICIIP-2013)*, IEEE, 2013, pp. 383–388.

[212] R. St. Amant, A. Yazdanbakhsh, J. Park, *et al.*, "General-purpose code acceleration with limited-precision analog computation," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 3, pp. 505–516, 2014.

[213] Q. Zhang, T. Wang, Y. Tian, F. Yuan, and Q. Xu, "Approxann: An approximate computing framework for artificial neural network," in *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2015, pp. 701–706.

[214] V. K. Chippa, D. Mohapatra, K. Roy, S. T. Chakradhar, and A. Raghunathan, "Scalable effort hardware design," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 9, pp. 2004–2016, 2014.

[215] E. Le Sueur and G. Heiser, "Dynamic voltage and frequency scaling: The laws of diminishing returns," in *Proceedings of the 2010 international conference on Power aware computing and systems*, 2010, pp. 1–8.

[216] A. B. Kahng and S. Kang, "Accuracy-configurable adder for approximate arithmetic designs," in *Proceedings of the 49th Annual Design Automation Conference*, 2012, pp. 820–825.

[217] P. Kulkarni, P. Gupta, and M. Ercegovac, "Trading accuracy for power with an underdesigned multiplier architecture," in *2011 24th Internatioal Conference on VLSI Design*, IEEE, 2011, pp. 346–351.

[218] "Regale project." (2021), [Online]. Available: https://regale-project.eu/.

[219] "Examon." (), [Online]. Available: https://github.com/EEESlab/examon.

[220] "Countdown." (), [Online]. Available: https://github.com/EEESlab/countdown.

[221] S. Williams, A. Waterman, and D. Patterson, "Roofline: An insightful visual performance model for multicore architectures," *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, 2009.

[222] H. Lim and R. Vuduc, "Cache-aware roofline model: Upgrading performance measurement for multicore systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ACM, 2016, pp. 153–165.

[223] P. Basu, A. Mandal, M. S. Muller, and F. Schlimbach, "Modeling stencil computations on modern hpc architectures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 10, pp. 1988–1998, 2013.

[224] J. Dongarra, J. Bunch, C. Moler, and G. Stewart, "Linpack users' guide," *Society for Industrial and Applied Mathematics*, 1986.

[225] P. Luszczek, J. J. Dongarra, D. Koester, and R. Rabenseifner, "High-performance linpack benchmark: Past, present, and future," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '16, Salt Lake City, Utah: ACM, 2016, pp. 1–12, ISBN: 978-1-4503-4092-2. DOI: 10.1145/3014904.3015013. [Online]. Available: https://doi.org/10.1145/3014904.3015013.

[226] J. Bai, X. Liu, X. Wang, X. Li, and X. Li, "A study on the stream benchmark performance on a power system," *ACM Trans. Embed. Comput. Syst.*, vol. 18, no. 2s, pp. 1–22, Mar. 2019, ISSN: 1539-9087. DOI: 10.1145/3331467. [Online]. Available: https://doi.org/10.1145/3331467.

[227] D. H. Bailey, J. Barton, D. S. Browning, *et al.*, "The NAS parallel benchmarks," NASA Ames Research Center, Tech. Rep. NASA/TP-2000-209478, 2000.

[228] S. P. E. Corporation, "Spec cpu benchmarks," Standard Performance Evaluation Corporation, Technical Report, 2017. [Online]. Available: https://www.spec.org/cpu2017/.

[229] Ohio-State-University, *Polybench wiki*, Accessed on 2023-03-15. [Online]. Available: https://sourceforge.net/p/polybench/wiki/Home/.

[230] A. Yazdanbakhsh, D. Mahajan, H. Esmaeilzadeh, and P. Lotfi-Kamran, "Axbench: A multiplatform benchmark suite for approximate computing," *IEEE Design& Test*, vol. 34, no. 2, pp. 60–68, 2017. DOI: 10.1109/MDAT.2016.2630270.

[231] U. M.-A. Consortium, *Collection of hpc mini-apps representing the collaborative work of uk institutions*, Accessed on 2023-03-15. [Online]. Available: https:%20//%20uk-mac.%20github.io/%20papers.html.

[232] J. Flich, G. Agosta, P. Ampletzer, *et al.*, "Mango: Exploring manycore architectures for next- generation hpc systems," in *2017 Euromicro Conference on Digital System Design (DSD)*, 2017, pp. 478–485. DOI: 10.1109/DSD.2017.51.

[233] W. Fornaciari, G. Agosta, D. Atienza, *et al.*, "Reliable power and time-constraints-aware predictive management of heterogeneous exascale systems," in *Proceedings of the 18th International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*, ser. SAMOS '18, Pythagorion, Greece: Association for Computing Machinery, 2018, pp. 187–194, ISBN: 978 145 036 49 42. DOI: 10.1145/3229631.3239368. [Online]. Available: https://doi.org/10.1145/3229631.3239368.

[234] X. Feng and A. K. Mok, "A model of hierarchical real-time virtual resources," in *23rd IEEE Real-Time Systems Symposium, 2002. RTSS 2002.*, IEEE, 2002, pp. 26–35.

[235] G. Lipari and E. Bini, "Resource partitioning among real-time applications," in *15th Euromicro Conference on Real-Time Systems, 2003. Proceedings.*, IEEE, 2003, pp. 151–158.

[236] I. Shin and I. Lee, "Periodic resource model for compositional real-time guarantees," in *RTSS 2003. 24th IEEE Real-Time Systems Symposium, 2003*, IEEE, 2003, pp. 2–13.

[237] E. Bini, M. Bertogna, and S. Baruah, "Virtual multiprocessor platforms: Specification and use," in *2009 30th IEEE Real-Time Systems Symposium*, IEEE, 2009, pp. 437–446.

[238] A. Burmyakov, E. Bini, and E. Tovar, "Compositional multiprocessor scheduling: The gmpr interface," *Real-Time Systems*, vol. 50, pp. 342–376, 2014.

[239] S. Groesbrink, L. Almeida, M. De Sousa, and S. M. Petters, "Towards certifiable adaptive reservations for hypervisor-based virtualization," in *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, IEEE, 2014, pp. 13–24.

[240] P. Ekberg and W. Yi, "Bounding and shaping the demand of generalized mixed-criticality sporadic task systems," *Real-time systems*, vol. 50, pp. 48–86, 2014.

[241] J. Lee, H. S. Chwa, L. T. Phan, I. Shin, and I. Lee, "Mc-adapt: Adaptive task dropping in mixed-criticality scheduling," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 16, no. 5s, pp. 1–21, 2017.

[242] A. Papadopoulos, E. Bini, S. Baruah, and A. Burns, "Adaptmc: A control-theoretic approach for achieving resilience in mixed-criticality systems," in *Proceeding ECRTS Conference*, LIPICS, 2018, p. 14.

[243] S. Baruah, B. Chattopadhyay, H. Li, and I. Shin, "Mixed-criticality scheduling on multiprocessors," *Real-Time Systems*, vol. 50, no. 1, pp. 142–177, 2014.

[244] X. Gu and A. Easwaran, "Dynamic budget management and budget reclamation for mixed-criticality systems," *Real-Time Systems*, vol. 55, pp. 552–597, 2019.

[245] S. Edgar and A. Burns, "Statistical analysis of wcet for scheduling," in *Proceedings 22nd IEEE Real-Time Systems Symposium (RTSS 2001) (Cat. No.01PR1420)*, 2001, pp. 215–224. DOI: 10.1109/REAL.2001.990614.

[246] F. J. Cazorla, L. Kosmidis, E. Mezzetti, C. Hernandez, J. Abella, and T. Vardanega, "Probabilistic worst-case timing analysis: Taxonomy and comprehensive survey," *ACM Comput. Surv.*, vol. 52, no. 1, Feb. 2019, ISSN: 0360-0300. DOI: 10.1145/3301283. [Online]. Available: https://doi.org/10.1145/3301283.

[247] R. I. Davis and L. Cucu-Grosjean, "A survey of probabilistic schedulability analysis techniques for real-time systems," *Leibniz Transactions on Embedded Systems*, vol. 6, no. 1, 04:1–04:53, May 2019. DOI: 10.4230/LITES-v006-i001-a004. [Online]. Available: https://ojs.dagstuhl.de/index.php/lites/article/view/LITES-v006-i001-a004.

[248] F. Reghenzani, G. Massari, and W. Fornaciari, "Timing predictability in high-performance computing with probabilistic real-time," *IEEE Access*, vol. 8, pp. 208 566–208 582, 2020. DOI: 10.1109/ACCESS.2020.3038559.

[249] R. A. Tau Leng, J. Hsieh, V. Mashayekhi, and R. Rooholamini, "An empirical study of hyper-threading in high performance computing clusters," *Linux HPC Revolution*, vol. 45, 2002.

[250] X. Zhang, A. Li, B. Zhang, W. Liu, X. Zhao, and Z. Li, "The cost performance of hyper-threading technology in the cloud computing systems," in *Advances in Swarm Intelligence: 7th International Conference, ICSI 2016, Bali, Indonesia, June 25-30, 2016, Proceedings, Part II 7*, Springer, 2016, pp. 581–589.

[251] G. Bernat, M. Newby, and A. Burns, "Probabilistic timing analysis: An approach using copulas," *Journal of Embedded Computing*, vol. 1, no. 2, pp. 179–194, 2005.

[252] F. Reghenzani, G. Massari, and W. Fornaciari, "The real-time linux kernel: A survey on preempt_rt," *ACM Comput. Surv.*, vol. 52, no. 1, Feb. 2019, ISSN: 0360-0300. DOI: 10.1145/3297714. [Online]. Available: https://doi.org/10.1145/3297714.

[253] D. B. de Oliveira, D. Casini, and T. Cucinotta, "Operating system noise in the linux kernel," *IEEE Transactions on Computers*, vol. 72, no. 1, pp. 196–207, 2023. DOI: 10.1109/TC.2022.3187351.

[254] I. P. Egwutuoha, D. Levy, B. Selic, and S. Chen, "A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems," *The Journal of Supercomputing*, vol. 65, no. 3, pp. 1302–1326, Sep. 2013, ISSN: 1573-0484. DOI: 10.1007/s11227-013-0884-0. [Online]. Available: https://doi.org/10.1007/s11227-013-0884-0.

[255] 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*, ser. EuroMPI 2016, Edinburgh, United Kingdom: Association for Computing Machinery, 2016, pp. 64–73, ISBN: 9781450342346. DOI: 10.1145/2966884.2966903. [Online]. Available: https://doi.org/10.1145/2966884.2966903.

[256] M. C. Huebscher and J. A. McCann, "A survey of autonomic computing—degrees, models, and applications," *ACM Comput. Surv.*, vol. 40, no. 3, Aug. 2008, ISSN: 0360-0300. DOI: 10.1145/1380584.1380585. [Online]. Available: https://doi.org/10.1145/1380584.1380585.

[257] V. M. Weaver and S. A. McKee, "Can hardware performance counters be trusted?" In *2008 IEEE International Symposium on Workload Characterization*, 2008, pp. 141–150. DOI: 10.1109/IISWC.2008.4636099.

[258] H. Hoffmann, M. Maggio, M. D. Santambrogio, A. Leva, and A. Agarwal, "A generalized software framework for accurate and efficient management of performance goals," in *2013 Proceedings of the International Conference on Embedded Software (EMSOFT)*, 2013, pp. 1–10. DOI: 10.1109/EMSOFT.2013.6658597.

[259] M. Maggio, H. Hoffmann, M. D. Santambrogio, A. Agarwal, and A. Leva, "Controlling software applications via resource allocation within the heartbeats framework," in *49th IEEE Conference on Decision and Control (CDC)*, 2010, pp. 3736–3741. DOI: 10.1109/CDC.2010.5717893.

[260] D. De Sensi, T. De Matteis, and M. Danelutto, "Simplifying self-adaptive and power-aware computing with nornir," *Future Generation Computer Systems*, vol. 87, pp. 136–151, 2018, ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2018.05.012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X17326699.

[261] M. Popov, A. Jimborean, and D. Black-Schaffer, "Efficient thread/page/parallelism autotuning for numa systems," in *Proceedings of the ACM International Conference on Supercomputing*, ser. ICS '19, Phoenix, Arizona: Association for Computing Machinery, 2019, pp. 342–353, ISBN: 978 1450 360 791. DOI: 10.1145/3330345.3330376. [Online]. Available: https://doi.org/10.1145/3330345.3330376.

[262] J. Park, S. Park, M. Han, and W. Baek, "Poster: The performance impact of thread packing on synchronization-intensive applications," in *2019 28th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2019, pp. 461–462. DOI: 10.1109/PACT.2019.00045.

[263] D. B. de Oliveira, D. Casini, and T. Cucinotta, "Operating system noise in the linux kernel," *IEEE Transactions on Computers*, vol. 72, no. 1, pp. 196–207, 2023. DOI: 10.1109/TC.2022.3187351.

[264] A. Thomasian, "Concurrency control: Methods, performance, and analysis," *ACM Comput. Surv.*, vol. 30, no. 1, pp. 70–119, Mar. 1998, ISSN: 0360-0300. DOI: 10.1145/274440.274443. [Online]. Available: https://doi.org/10.1145/274440.274443.

[265] J. Schwarzrock, M. G. Jordan, G. Korol, *et al.*, "Dynamic concurrency throttling on numa systems and data migration impacts," *Des. Autom. Embedded Syst.*, vol. 25, no. 2, pp. 135–160, Jun. 2021, ISSN: 0929-5585. DOI: 10.1007/s10617-020-09243-5. [Online]. Available: https://doi.org/10.1007/s10617-020-09243-5.

[266] 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.

[267] S. Kumar, A. Gupta, V. Kumar, and S. Bhalachandra, "Cuttlefish: Library for achieving energy efficiency in multicore parallel programs," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '21, St. Louis, Missouri: Association for Computing Machinery, 2021, ISBN: 9781450384421. DOI: 10.1145/3458817.3476163. [Online]. Available: https://doi.org/10.1145/3458817.3476163.

[268] M. Aldinucci, S. Campa, M. Danelutto, *et al.*, "Behavioural skeletons in gcm: Autonomic management of grid components," in *16th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP 2008)*, 2008, pp. 54–63. DOI: 10.1109/PDP.2008.46.

[269] 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.

[270] A. Vogel, D. Griebler, M. Danelutto, and L. G. Fernandes, "Self-adaptation on parallel stream processing: A systematic review," *Concurrency and Computation: Practice and Experience*, vol. 34, no. 6, e6759, 2022. DOI: https://doi.org/10.1002/cpe.6759. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.6759. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.6759.

[271] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. of the 20th Intl. Conference on Artificial Intelligence and Statistics AISTATS*, A. Singh and X. (Zhu, Eds., ser. Proceedings of Machine Learning Research, vol. 54, PMLR, 2017, pp. 1273–1282.

[272] L. Li, Y. Fan, M. Tse, and K.-Y. Lin, "A review of applications in federated learning," *Computers & Industrial Engineering*, vol. 149, p. 106 854, 2020, ISSN: 0360-8352. DOI: https://doi.org/10.1016/j.cie.2020.106854.

[273] P. Kairouz, H. B. McMahan, B. Avent, *et al.*, "Advances and open problems in federated learning," *Found. Trends Mach. Learn.*, vol. 14, no. 1-2, pp. 1–210, 2021. DOI: 10.1561/2200000083. [Online]. Available: https://doi.org/10.1561/2200000083.

[274] I. Colonnelli, B. Casella, G. Mittone, *et al.*, "Federated learning meets hpc and cloud," in *Proceedings of astrophysics and space science*, 2022.

[275] F. D'Ascenzo, O. De Filippo, G. Gallone, *et al.*, "Machine learning-based prediction of adverse events following an acute coronary syndrome (PRAISE): A modelling study of pooled datasets," *The Lancet*, vol. 397, no. 10270, pp. 199–207, 2021, ISSN: 0140-6736. DOI: 10.1016/S0140-6736(20)32519-8.

[276] M. Johanson, S. Belenki, J. Jalminger, M. Fant, and M. Gjertz, "Big automotive data: Leveraging large volumes of data for knowledge-driven product development," in *2014 IEEE Intl. Conference on Big Data (Big Data)*, 2014, pp. 736–741. DOI: 10.1109/BigData.2014.7004298.

[277] Y. Wang, I. L. Bennani, X. Liu, M. Sun, and Y. Zhou, "Electricity consumer characteristics identification: A federated learning approach," *IEEE Transactions on Smart Grid*, vol. 12, no. 4, pp. 3637–3647, 2021. DOI: 10.1109/TSG.2021.3066577.

[278] Z. Zhang, C. Guan, H. Chen, X. Yang, W. Gong, and A. Yang, "Adaptive privacy-preserving federated learning for fault diagnosis in internet of ships," *IEEE Internet of Things Journal*, vol. 9, no. 9, pp. 6844–6854, 2022. DOI: 10.1109/JIOT.2021.3115817.

[279] M. Polato, R. Esposito, and M. Aldinucci, "Boosting the federation: Cross-silo federated learning without gradient descent," 2022.

[280] G. A. Reina, A. Gruzdev, P. Foley, *et al.*, "Openfl: An open-source framework for federated learning," *arXiv preprint arXiv:2105.06413*, 2021.

[281] D. J. Beutel, T. Topal, A. Mathur, *et al.*, "Flower: A friendly federated learning research framework," *arXiv preprint arXiv:2007.14390*, 2020.

[282] Z. Sun, P. Kairouz, A. T. Suresh, and H. B. McMahan, "Can you really backdoor federated learning?" *arXiv preprint arXiv:1911.07963*, 2019.

[283] H. Ludwig, N. Baracaldo, G. Thomas, *et al.*, "Ibm federated learning: An enterprise framework white paper v0. 1," *arXiv preprint arXiv:2007.10987*, 2020.

[284] J. Kang, Z. Xiong, D. Niyato, Y. Zou, Y. Zhang, and M. Guizani, "Reliable federated learning for mobile networks," *IEEE Wireless Communications*, vol. 27, no. 2, pp. 72–80, 2020.

[285] S. Warnat-Herresthal, H. Schultze, K. L. Shastry, *et al.*, "Swarm learning for decentralized and confidential clinical machine learning," *Nature*, vol. 594, no. 7862, pp. 265–270, 2021.

[286] I. Colonnelli, B. Cantalupo, I. Merelli, and M. Aldinucci, "StreamFlow: Cross-breeding cloud with HPC," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 4, pp. 1723–1737, 2021. DOI: 10.1109/TETC.2020.3019202.

[287] R. Adolf, S. Rama, B. Reagen, G.-y. Wei, and D. Brooks, "Fathom: Reference workloads for modern deep learning methods," in *2016 IEEE Intl. Symposium on Workload Characterization (IISWC)*, 2016.

[288] A. Paszke, S. Gross, F. Massa, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., 2019, pp. 8024–8035.

[289] Martín Abadi, Ashish Agarwal, Paul Barham, *et al.*, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: https://www.tensorflow.org/.

[290] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[291] G. Mittone, N. Tonci, R. Birke, *et al.*, *Experimenting with emerging arm and risc-v systems for decentralised machine learning*, 2023. DOI: 10.48550/ARXIV.2302.07946. [Online]. Available: https://arxiv.org/abs/2302.07946.

[292] O. Rausch, T. Ben-Nun, N. Dryden, A. Ivanov, S. Li, and T. Hoefler, "DaCeML: A data-centric optimization framework for machine learning," in *Proceedings of the 36th ACM International Conference on Supercomputing*, ser. ICS '22, 2022.

[293] M. Baboulin *et al.*, "Accelerating scientific computations with mixed precision algorithms," *Computer Physics Communications*, vol. 180, pp. 2526–2533, 12 Dec. 2009, ISSN: 00104655. DOI: 10.1016/j.cpc.2008.11.005. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0010465508003846.

[294] P. Stanley-Marbell *et al.*, "Exploiting errors for efficiency: A survey from circuits to applications," *ACM Computing Surveys (CSUR)*, vol. 53, no. 3, pp. 1–39, 2020.

[295] A. Zeller and R. Hildebrandt, "Simplifying and isolating failure-inducing input," *IEEE Transactions on Software Engineering*, vol. 28, no. 2, pp. 183–200, Feb. 2002, ISSN: 0098-5589. DOI: 10.1109/32.988498.

[296] M. O. Lam *et al.*, "Automatically adapting programs for mixed-precision floating-point computation," in *Proc. 27th Int'l ACM Conf. on Supercomputing*, ser. ICS '13, Eugene, Oregon, USA, 2013, pp. 369–378, ISBN: 978-1-4503-2130-3. DOI: 10.1145/2464996.2465018.

[297] M. O. Lam and J. K. Hollingsworth, "Fine-grained floating-point precision analysis," *The International Journal of High Performance Computing Applications*, vol. 32, no. 2, pp. 231–245, 2018. DOI: 10.1177/1094342016652462.

[298] M. Lam, T. Vanderbruggen, H. Menon, and M. Schordan, "Tool integration for source-level mixed precision," *2019 IEEE/ACM 3rd International Workshop on Software Correctness for HPC Applications (Correctness)*, pp. 27–35, 2019.

[299] K. Parasyris *et al.*, "Hpc-mixpbench: An hpc benchmark suite for mixed-precision analysis," *2020 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 25–36, Oct. 2020. DOI: 10.1109/IISWC50251.2020.00012. [Online]. Available: https://ieeexplore.ieee.org/document/9251260/.

[300] H. Menon, M. O. Lam, D. Osei-Kuffuor, *et al.*, "Adapt: Algorithmic differentiation applied to floating-point precision tuning," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, ser. SC '18, Dallas, Texas: IEEE Press, 2018. DOI: 10.1109/SC.2018.00051. [Online]. Available: https://doi.org/10.1109/SC.2018.00051.

[301] I. Karlin, J. Keasler, and J. R. Neely, "Lulesh 2.0 updates and changes," Jul. 2013. DOI: 10.2172/1090032. [Online]. Available: https://www.osti.gov/biblio/1090032.

[302] S. Cherubin, G. Agosta, I. Lasri, E. Rohou, and O. Sentieys, "Implications of Reduced-Precision Computations in HPC: Performance, Energy and Error," in *International Conference on Parallel Computing (ParCo)*, Bologna, Italy, Sep. 2017, ISBN: 978-1-61499-842-6.

[303] E. Darulova, E. Horn, and S. Sharma, "Sound mixed-precision optimization with rewriting," in *Proceedings of the 9th ACM/IEEE International Conference on Cyber-Physical Systems*, ser. ICCPS '18, Porto, Portugal, 2018, pp. 208–219, ISBN: 978-1-5386-5301-2. DOI: 10.1109/ICCPS.2018.00028.

[304] E. Darulova and V. Kuncak, "Sound compilation of reals," 1, vol. 49, New York, NY, USA: Association for Computing Machinery, Jan. 2014, pp. 235–248. DOI: 10.1145/2578855.2535874. [Online]. Available: https://doi.org/10.1145/2578855.2535874.

[305] 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.

[306] C. Lattner and V. Adve, "LLVM: A compilation framework for lifelong program analysis & transformation," in *Proc. Int'l Symp. on Code Generation and Optimization*, Palo Alto, California, 2004, ISBN: 0-7695-2102-9.

[307] L. Dagum and R. Menon, "Openmp: An industry standard api for shared-memory programming," *Computational Science & Engineering, IEEE*, vol. 5, no. 1, pp. 46–55, 1998.

[308] D. Cattaneo, M. Chiari, S. Cherubin, and G. Agosta, "Feedback-driven performance and precision tuning for automatic fixed point exploitation," in *International Conference on Parallel Computing*, ser. ParCo, Prague, Czech Republic, Sep. 2019.

[309] "Ieee standard for floating-point arithmetic," *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pp. 1–84, 2019. DOI: 10.1109/IEEESTD.2019.8766229.

[310] S. Cherubin and G. Agosta, "Tools for reduced precision computation: A survey," *ACM Computing Surveys*, vol. 53, no. 2, Apr. 2020, ISSN: 0360-0300. DOI: 10.1145/3381039.