Increasing the Lifetime of HPC Machines: Issues, Implications, and Open Challenges

Robin Boëzennec, Fernando Fernandes dos Santos, Brice Goglin, Angeliki Kritikakou, Guillaume Pallez, Erven Rohou, Olivier Sentieys, Marcello Traiola

Increasing the Lifetime of HPC Machines: Issues, Implications, and Open Challenges

Robin Boezennec, Fernando Fernandes dos Santos, Brice Goglin, Angeliki Kritikakou, Guillaume Pallez, Erven Rohou, Olivier Sentieys and Marcello Traiola

***Abstract*—Extending the lifetime of High-Performance Computing (HPC) machines is becoming an important concern for a variety of reasons. These include the environmental and human costs associated with chip manufacturing, the rising demands by AI workloads, the soaring prices of accelerator chips, political blocks, and delays in the delivery of next-generation supercomputers. As a community, we must reconsider the traditional HPC paradigm and explore new strategies for making existing HPC infrastructure viable for longer periods. In this work, we highlight the current barriers in prolonging HPC machines lifespan and discuss key technical and operational challenges towards this goal.**

***Index Terms*—Aging, HPC lifetime, maintainability, hardware faults, software compatibility**

I. Introduction

The traditional paradigm in HPC relies on regular deployment of new systems every few years, each generation offering increased performance, efficiency, and capabilities. However, this paradigm is nowadays reaching its limits. The most important reasons stems from the environmental and human costs associated with producing the hardware that powers HPC systems, such as the raw materials and the manufacturing costs, or even geopolitical costs. At the same time, the rapid growth of artificial intelligence has led to increased demand for hardware accelerators, such as Graphics Processing Unit (GPU) and Tensor Processing Unit (TPU) platforms, pushing prices high and causing global supply shortages. Delays in the delivery of next-generation supercomputers only worsen the problem, leaving users with the only available option of relying on their current systems for extended periods. As a result, it becomes a matter of practical importance, long term vision, and ethical responsibility to increase the efforts for extending the lifetime of existing HPC systems. This implies using the initial system resources, even when they start exhibiting non-nominal behavior, instead of replacing them immediately. Additionally, proactive investments in design improvements and usage practices play a key role in delaying resource degradation. To achieve this goal, it is imperative for the HPC community to rethink how to design, maintain, and evolve HPC systems, not only focusing on peak performance, but also **considering longevity as an equally important objective**.

System longevity is threatened by both hardware and software degradation over time, which affects the system nominal behavior. On the hardware side, components such as memory modules, processors, and interconnects degrade due to physical phenomena, such as electromigration [1], [2], resulting in slower transistors and faults on the chip. With the ongoing technology reduction, hardware wear-out occurs significantly earlier, reducing the lifetime of the circuits [3]. Software degradation manifests as the inability of outdated operating systems, libraries, and middleware to support evolving workloads, security standards, and hardware interfaces. With the software complexity increase of modern systems, software obsolescence is becoming increasingly challenging to manage over time. The effects of hardware wear-out and software obsolescence lead to HPC system aging, reducing the overall efficiency and usability of HPC systems, posing significant challenges for their long-term maintainability.

The goal of this paper is to lay the groundwork for future research on HPC system aging towards designing more maintainable HPC infrastructures. To achieve this, we discuss the barriers and the scientific and technical challenges, as well as the implications across the system stack, when the operational lifetime of an HPC systems is extended beyond its originally intended lifetime. More precisely, we initially identify and analyze the hardware-related and software-related issues that arise when the lifetime of the HPC infrastructure is extended, considering the evolving stakeholder requirements and concerns. This is achieved through a high-level meta-classification of non-nominal behavior in HPC systems describing the diverse manifestations of system degradation. Section III distinguishes between non-nominal functional behavior, parametric deviations, and complete unavailability on both hardware and software fronts, while considering stakeholder expectations and system-level trade-offs to support better diagnosis, management, and mitigation of HPC system aging. Addressing the issues that emerge when extending the lifespan of HPC machines presents several open challenges. In Section IV, we provide an overview of potential families of solutions to the hardware and software issues with the goal of identifying research directions and open challenges necessary to make these solutions both practical and applicable in real-world scenarios. Tackling these challenges requires a coordinated effort across the entire HPC stack and a **collaborative, community-wide initiative to redefine the HPC stack with longevity as one of the primary design goals.**

Note that, this work focuses on research challenges on increasing longevity of HPC systems by dealing with aging occurring due to the prolonged use of hardware and software components. Furthermore, it is important to clarify that **this**

All authors are with Inria. RB, FDS, AK, GP, ER, OS, MT are with Univ Rennes, Inria, CNRS, IRISA. BG is with Inria, LaBRI, Univ. Bordeaux.
E-mail: {first.last}@inria.fr
Authors are sorted by alphabetical order.

work does not aim to provide a survey of HPC reliability techniques, fault tolerance methods, or reliability assessment approaches. Existing literature covers the aspects on how to deal with faults and failures in supercomputing systems (whether they are "hard" [4], [5], [6], [7], [8] or "silent" [9], [10], [11], [12]), and their usage due to the extremely large number of resources used in HPC infrastructures [13]. Fault tolerance also includes mitigating the impact of faults through redundancy [14]. This has been used in high performance and scalable networks [15] or storage [16] for a long time. It may be extended to other hardware components through disaggregated architectures [17] in the future, but it is outside the scope of this work where we rather study the challenges of implementing more general and long-term solutions for extending the lifetime of HPC machines.

II. Motivation and current trends

In recent years, the HPC community has begun to recognize the importance of increasing the lifetime of HPC machines as a key strategy for improving sustainability, reducing costs, and minimizing environmental impact. As awareness of the ecological and economic implications of frequent hardware turnover grows, both researchers and industry stakeholders are exploring ways to extend the usable life of HPC systems. This section presents an overview of current trends in this direction, including recent research efforts and practical actions that have been taken to support this goal. Recent work provides a comprehensive overview of sustainability challenges and opportunities in HPC [18], with a particular emphasis on reducing environmental impact. The study highlights how current HPC systems face growing energy demands and frequent hardware turnover, both of which contribute significantly to their carbon footprint. The work advocates a holistic approach that aligns hardware design, software efficiency, and operational strategies to enhance long-term system viability. A key contribution of this research is its emphasis on collaborative efforts across academia, industry, and government to standardize sustainability metrics and promote best practices. The study highlights the importance of hardware longevity as a cornerstone of sustainable HPC, arguing that longer-lasting infrastructure not only reduces e-waste and manufacturing impact but also complements energy efficiency initiatives. While quantitative data on carbon intensities across different geographical regions in Europe is provided, along with useful directions for the community, there is no comprehensive classification or detailed analysis of the diverse manifestations of system degradation and the open challenges faced in implementing related solutions. The absence of a thorough analysis makes it difficult to establish a clear research path forward, as critical insights into the complexities of these issues are lacking.

Efforts to extend the lifespan of HPC machines are increasingly emerging across both industry and research. Institutions are increasingly opting to extend the operational life of HPC systems through targeted hardware upgrades. For instance, the Texas Advanced Computing Center (TACC) extended the lifespan of its Stampede2 supercomputer by replacing 448 older Intel Knights Landing nodes with 224 newer Dell PowerEdge R650 servers equipped with Intel Xeon Platinum 8380 processors [19]. Collaborations with hardware vendors have proven effective in prolonging system support. Oak Ridge National Laboratory's Summit supercomputer, initially slated for decommissioning in early 2024, received an additional year of support through negotiations with IBM and NVIDIA [20]. This extension allowed Summit to continue contributing to over 100 research projects. Repurposing older HPC hardware for specialized tasks is another approach to extending utility. Argonne National Laboratory plans to transform the retiring ThetaGPU hardware into a new system named Sophia, dedicated to specific computational tasks. This strategy not only maximizes the value extracted from existing hardware but also supports sustainability goals by reducing electronic waste [21]. By adopting circular economy principles, companies like Google implement practices such as maintenance, refurbishment, and redistribution of hardware components. By optimizing the end-of-life process based on Total Cost of Ownership (TCO) rather than traditional accounting life, Google has achieved significant cost savings and reduced environmental impact [22]. Ensuring software can operate across various hardware platforms is crucial for longevity. The Lawrence Livermore National Laboratory's "HPC Center of the Future" initiative advocates for open standards to seamlessly integrate hardware and software from multiple vendors, emphasizing a flexible base software environment and infrastructure-as-a-service models [23]. Such approaches facilitate seamless transitions between hardware generations, thereby extending the functional lifespan of software applications. Ongoing optimization of software components contributes to system longevity. TACC's Stampede2, for example, continues to enhance key open-source technologies like MVAPICH MPI libraries and the Lustre parallel file system. These improvements ensure that the software remains efficient and compatible with evolving hardware, thereby extending its useful life. Implementing systems that allow for dynamic allocation of resources can extend hardware utility. Google's use of the Borg architecture enables flexible resource management across its data centers, allowing for efficient utilization and prolonging hardware relevance [22]. Incorporating AI and data analysis capabilities into existing HPC systems can rejuvenate their utility. Argonne's Theta supercomputer, for instance, was augmented with NVIDIA GPUs to support COVID-19 research, thereby extending its relevance and application scope [21].

Motivated by these reasons and current trends in the HPC community aiming to increase the lifetime of HPC machines, we propose a study of the key issues related to extending HPC machine lifetimes through a meta-classification framework, and highlight the main technical and operational challenges for the community to collectively address them.

III. HPC non-nominal behavior meta-classification

The objective of this section is to raise awareness about the issues stemming from extending the lifespan of HPC systems. Extending the lifespan of HPC systems is related to *increasing*

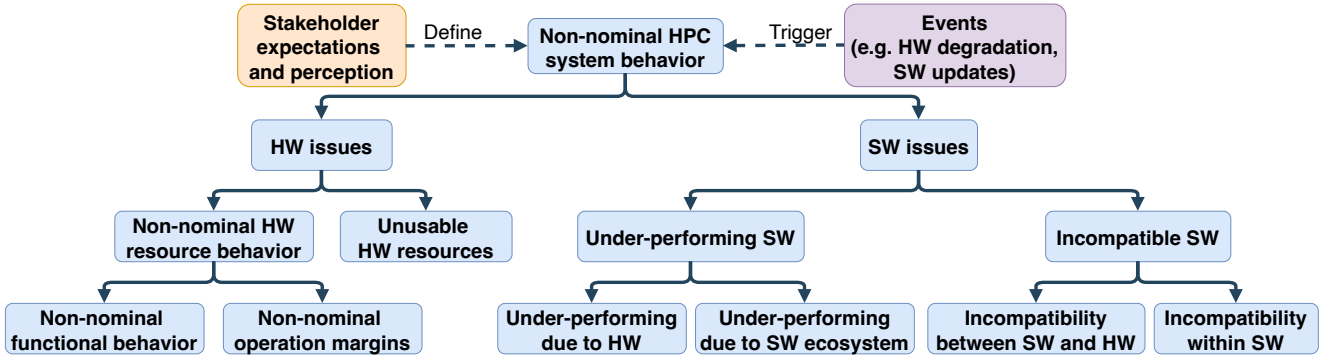


Fig. 1: Proposed non-nominal behavior meta-classification. The stakeholders will have expectations and perceptions defining what is nominal/non-nominal. The non-nominal behaviors will be triggered by external factors such as hardware aging, software updates, etc.

the efforts to still use the machine original resources, even when they start exhibiting non-nominal behavior, instead of replacing them, or even investing proactively in design efforts and usage practices with the goal of delaying the resources degradation. We introduce and discuss the phenomena that can trigger the non-nominal behavior of an HPC system. To achieve this, we propose a high-level meta-classification of the possible reasons leading to non-nominal behavior in HPC systems, depicted in Figure 1. A non-nominal system behavior is typically triggered by some *events* impacting or interacting with the system. Examples can be the hardware wear-out (i.e., transistors degradation) and the software evolution and maintenance (e.g., updates during the system lifetime). The non-nominal system behavior is caused by issues coming from *hardware (HW)* or *software (SW)*, which are described in details in the following paragraphs, along with representative examples. As hardware, we refer to *the collection of physical elements of a computing system that are involved in the execution, processing, storage, and communication of digital information*. Unlike software, which consists of instructions and data, hardware refers to the mechanical, magnetic, electronic, and electrical components that support and perform computational tasks. As an example, we consider CPUs and GPUs as hardware, and the code that runs on them (e.g., firmwares, drivers, OS, kernels, applications) as software.

Note that, the graph is intentionally high-level, to include different potential scenarios and examples. The meta-classification itself is parametric as the definition of *non-nominal behavior* depends on the expectation and perception of the system's stakeholders (e.g., users, system administrators, system owners, investors, etc.). For example, characterizing a hardware resource as a resource that behaves in a *non-nominal* way or as an *unusable* resource partially depends on the stakeholder's expectation, e.g., using an GPU working in a lower frequency can be an acceptable or not depending on the timing requirements of the stakeholder. In this way, the meta-classification can be adapted to more scenarios and examples becomes general and flexible.

A. Hardware issues

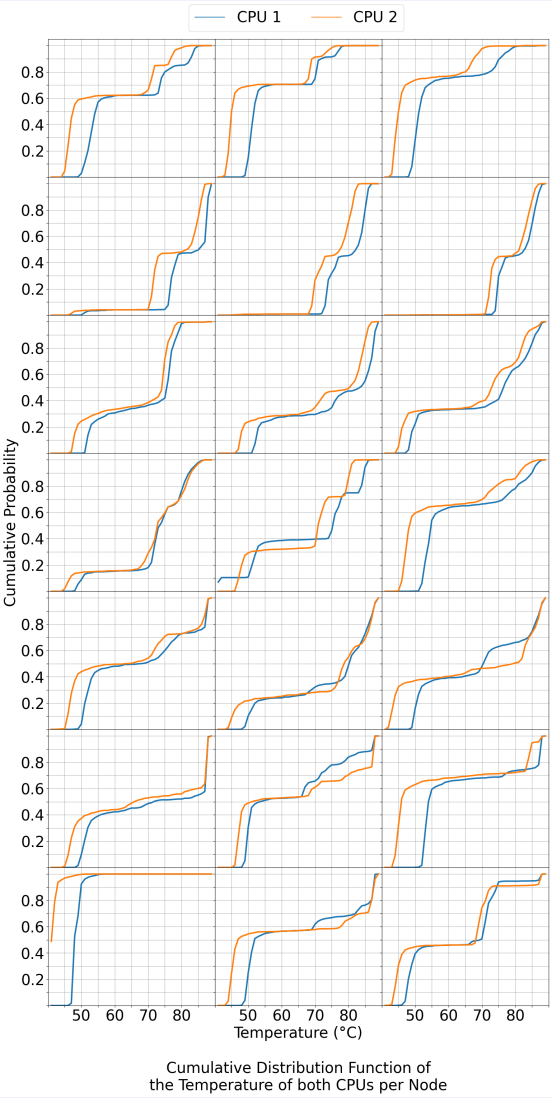
Current and upcoming generations of hardware will inevitably have faults and defects as they execute workloads, encountering stress, and thus, aging. These challenges are further exacerbated by the increasing complexity and scale of HPC systems, making them more susceptible to failure over time. To provide insight on the relevant hardware issues, we first present the primary sources that can jeopardize the proper functioning of hardware, and then, we discuss the possible effects on HPC systems.

1) Sources: Hardware can be affected by faults originating from several sources, including environmental perturbations, process, temperature, and voltage variations, material defects, and hardware aging [24], [25], [26], [27], [1], [28]. With the continuous miniaturization of transistors, these fault sources are becoming more frequent. For instance, it is reasonable to anticipate an increase in fault occurrences as smaller devices are more susceptible to faults over time [29], [26], [30], [31], [32]. Recent studies [33], [34] show that the vulnerability of modern digital devices, particularly those built on sub-7 nm processes, will be affected by phenomena such as Electromigration and Time-Dependent-Dielectric-Breakdown—both major contributors to hardware faults and accelerated aging [35], [2]. The International Roadmap for Devices and Systems notes that device lifetime diminishes by half with each new manufacturing generation [35], and when combined with harsh operating conditions, including high temperatures and radiation-induced latch-ups, the risk of permanent faults increases significantly [36], [37], [25], [38], [39]. Furthermore, hardware faults are becoming increasingly sensitive to the workload [40], [41], [42], [43]. For instance, different cores are subjected to different amounts of stress due to varying workloads, leading to an aging imbalance among cores [44].

Example III-A.1: Heterogeneous cooling

Cooling in data centers is heterogeneous. Whether achieved by fans or liquid cooling, nodes closer to the cooling source have a higher cooling parameter than those farther away. We illustrate this by measuring using

ipmitool on a simple small cluster (22 nodes but one was never turned on, each with 2 AMD Epyc 9224 CPUs) the distribution of temperature of the different CPUs during a 10 day period (the measurement were taken every 10 minutes).



Each node shows a different trend depending on the workload (for instance, the node in the bottom left graph was never used during this period; 45-50°C corresponds to idle mode). If we focus on the highest temperature mode, we observe not only a difference in maximum temperature for each node, but also a difference of several degrees between the two CPUs within each node, highlighting the impact of heterogeneous cooling. This heterogeneity directly impacts the aging of these nodes [45], [26], [46], thus leading to heterogeneity in the lifespan of nodes. We discuss in Challenge IV-B.1 how one can use aging models to study the effect of node permutations – i.e. unplugging and re-plugging compute nodes in different locations of the data-center – to impact aging, and the challenges that this solution poses.

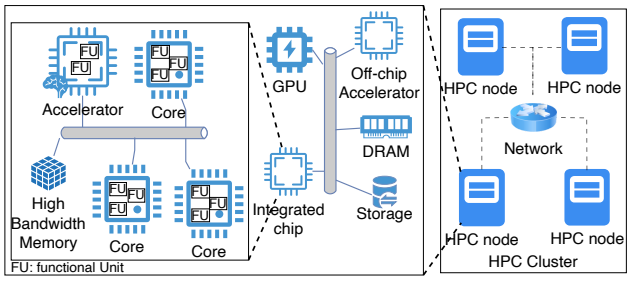


Fig. 2: An abstract representation of an HPC system includes nodes with sub-components like integrated chips, GPUs, DRAM, and storage. Integrated chips can be divided into processing cores, HBM, and accelerators. This helps fault isolation, e.g., a chip is still usable if 3 processing cores are usable and 1 core is unusable due to a permanent fault.

2) Effects: Once a hardware *fault* interacts with the active computation, the fault can manifest as an *error*, corrupting data or computation flow. When such errors propagate to the system level, they result in *failures*, leading to a reliability violation, i.e., incorrect functional behavior of the system [47], [48]. Failures are usually grouped into ones causing immediate crashes or exceptions (i.e., function interruptions) and ones leading to incorrect computations or data loss, which may not be noticed immediately. The latter class of failures is often referred to as *silent data corruption (SDC)*.

To understand the impact of faults and defects at the system level, it is important to classify how they affect different system components. To illustrate the need for such a classification, consider a typical HPC system as shown in Figure 2. Faults and defects can impact various system parts, potentially rendering them unusable. However, if only a single part (e.g., one core) is compromised, the remaining hardware resources will still function. Thus, the system can still be considered usable, even if it does not operate as originally designed. This principle applies to all system levels, including functional units, cores, chips, and nodes. Therefore, we classify the hardware issues, when the HPC system lifetime is extended, into three main categories: • *Non-nominal Functional Behavior* (Subsec. III-A3); • *Non-nominal Operation Margins* (Subsec. III-A4); • *Unusable Hardware Resources* (Subsec. III-A5).

3) Non-nominal Functional Behavior: This class includes hardware resources that are operational, but are not performing computations as originally designed. Such issues can happen at various levels within the system, including the core level (e.g., a functional unit producing incorrect computations) [49], memory levels (e.g., bits that are permanently stuck at value 0 or 1) [7], interconnections on the chip (e.g., defects in the lines) [50], or accelerators with permanent faults (e.g., incorrect thread scheduling) [8]. In this category, we include the situations where the cause of non-nominal functional behavior cannot be addressed (i.e., some permanent defect prevent the transistors from functioning properly). Approaches to circumvent the issue and still provide at least a partial functionality are discussed in Section IV.

Example III-A.2: Silent File Loss Due to CPU Defect

A concrete instance of non-nominal functional behavior has been reported by Meta in large-scale production environments [51]. They showcase a decompression job running on a Spark-based pipeline sometimes silently drops input files. The root cause was a faulty CPU core: when computing file sizes using math.pow() and casting the result to an integer, the operation occasionally returned zero instead of the correct positive size. This defect triggered silent data corruption by causing valid files to be dropped during decompression, without raising any hardware errors (ECC, machine check, logs). The bug only appeared on a specific core, and only for certain exponent values. From the HPC system's point of view, everything looked *healthy*, yet data was silently lost.

4) Non-nominal Operation Margins: A hardware component can remain functional but exhibit parametric-induced faults, i.e., faults originating from deviations in hardware parameters [48]. We refer to this category as system operating under non-nominal operation margins. An example is Inter-cell Parametric Disturbance in DRAM memory, where the capacitance and isolation margins of DRAM cells drop below safe limits. This occurs when repeatedly activating one row exceeds these margins, causing charge leakage and bit flips in neighboring cells. [52]. Another example is hardware defects, such as short circuits or soldering issues, which can cause the operating temperature to rise abnormally [48]. An additional example is the shift in the *threshold voltage* of transistors due to aging, where the delay of individual transistors is proportional to this threshold voltage shift [53], [54]. Hence, transistor aging leads to an increased probability of *timing violation*, i.e., the duration of the computation exceeds the clock period, leading to faults. As we will discuss in Section IV, techniques to reduce the effects of those defects exist, but they often have an impact on the resource performance to ensure correct execution, e.g., by reducing the clock frequency.

Example III-A.3: Rowhammer: Parametric-Induced Faults in Scaled DRAM

An example of a hardware component operating under non-nominal margins is the Rowhammer effect in DRAM [52]. DRAM cells store data as charges in tiny capacitors, and under nominal conditions, each cell retains its charge long enough for reliable operation, with minimal interference between neighbors. However, as process nodes shrink, cell capacitance decreases and parasitic coupling increases, reducing safe operating margins. When a row is repeatedly accessed ("hammered"), these parametric deviations can cause charge leakage in adjacent rows, producing unintended bit flips. Importantly, this is a parametric-induced fault: the cells are not defective, but the altered physical parameters lead to failures outside nominal operation, leading to reliability and security problems [55].

5) Unusable HW Resources: The resource has reached its end of life and is no longer usable. It may either be completely unresponsive or considered no longer suitable for the system. At this point, as it will be discussed in Section IV, the resource must be excluded from usage, emulated with other components, or replaced.

Example III-A.4: Broken hardware resources

Hardware components, such as memory modules or CPUs, may be considered to reach end-of-life by some stakeholders when certain parts fail or become unsuitable for operation. For example, if some memory modules are defective, the entire memory may be considered unusable, or part of it may still be utilized, if the memory system supports this. Similarly, in cases such as an HPC server with a malfunctioning CPU core, where other components remain functional, the administrators can still use the remaining components or emulate the faulty ones to ensure continued operation, preventing the system from failing to boot due to the faulty component.

B. Software issues

When extending a system's lifetime, stakeholders decide whether to upgrade the system software stack or not. Both choices have distinct advantages and drawbacks. Upgrading offers access to newer environments and tools, but it may result in compatibility issues, such as missing drivers or limited support for older hardware. Conversely, maintaining the existing configuration ensures functionality without modification, but it may lack modern support and libraries required by modern applications. Note that, in some cases, the upgrades are not optional. For example, security concerns may force to upgrade software as vulnerabilities appear.

Attempting to run either recent software on older hardware or older software on newer hardware can lead to under-performance or incompatibility issues. *Under-performance* happens when the application operates but in a diminished capacity—such as reduced speed or accuracy. Applications may adjust their Quality of Service (QoS) based on hardware performance. For instance, a machine learning model can use a smaller float-point precision to keep the processing time to acceptable levels. Contrarily, *incompatibility* happens when the application fails to start or crashes soon after launching. Similar to hardware problems previously discussed, the distinction between under-performing and incompatible software can be subjective, depending on stakeholder expectations. What one considers slow might be acceptable to another, but if it crosses the threshold of usability, it becomes unacceptable. The following paragraphs discuss these cases in details, along with representative examples.

1) Sources: Software issues may appear over time, because of the numerous components of the HPC stack, whose compatibility is not always well maintained in the long-term, especially when a partial system update takes place. Most HPC software depends on several libraries whose compatibility is

defined by *Application Programming* and *Binary Interfaces* (API and ABI). Good software development practices are required to maintain backward compatibility (e.g., not removing functions, or changing their prototype or behavior). However, long-term maintenance may impose a high workload on developers. Functions deprecated for a long time are usually removed after some years, breaking other software parts that were not updated. Such issues can appear because libraries have dependency chains, newer hardware support, or for security reasons. Furthermore, compilers and programming languages may also cause issues because languages and compilers evolve. An old compiler may not support the recent revision of C++ used in a newer application version, or new compilers can be so pedantic that they may not successfully compile an old program.

2) Effects: Due to the aforementioned reasons, there are software compatibility problems between applications, libraries, and compilers. Therefore, we classify the issues related to the software, when the lifetime of HPC systems is extended, as: • *Under-performing SW due to HW* (Subsec. III-B3); • *Under-performing SW due to Surrounding SW Ecosystem* (Subsec. III-B4); • *Incompatibility between SW and HW* (Subsec. III-B5); • *Incompatibility within SW* (Subsec. III-B6).

3) Under-performing SW due to Hardware: Modern computing environments often face challenges due to the complex interplay between hardware configurations and software behavior. This phenomenon gets exacerbated when specific hardware resources become partially unavailable, causing execution to slow down. This may occur due to reduced resource availability or the need to emulate missing features on alternative components.

Example III-B.1: Hardware/software interactions

If a CPU experiences partial failure, requiring some cores to be turned off, this can lead to an asymmetric compute node with an unusual number of cores, which may cause algorithms to misbehave during workload distribution. Failure of other computing elements, such as GPUs or TPUs, also has a severe impact: even though applications might be designed to function without them, their performance will degrade significantly. Similarly, if only some chips of the memory are available due to hardware failure, this can lead to memory allocation issues. On a finer scale, software relying on advanced CPU instructions (e.g., Intel Advanced Matrix Extensions) might fall back to scalar implementations if those features become unavailable, but at a cost.

4) Under-performing SW due to Surrounding SW Ecosystem: Some software may execute sub-optimally because it cannot benefit from features usually available in some libraries. The reasons for the unavailability of a library are diverse: it is simply not installed, the license expired and is too expensive to consider renewal, or the vendor no longer supports it. In either case, developers must rely on less-performing substitutes, such as manually programming a missing feature.

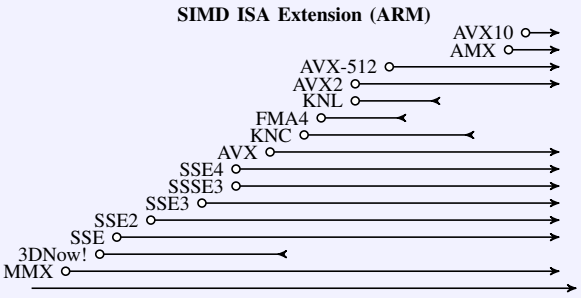
Example III-B.2: Missing libraries and drivers

Most HPC runtimes rely on hwloc [56] for detecting the topology of the compute nodes, but they may fall to manual detection when it is unavailable. The result will be less precise and less portable, possibly resulting in inadequate process placement and decreased overall performance. On the commercial software side, NVIDIA regularly phases out support for older GPU architectures after some product generations [57].

5) Incompatibility between SW and HW: Hardware characteristics are usually specified in technical manuals, and software is free to take this information for granted. Some software is likely to fail to run if the aging hardware does not anymore match the platform anticipated at design time.

Example III-B.3: Broken assumptions about HW

Software drivers may hard-) – (1-1996+1,-.2) node[label = below:1] ;



Developers gain better performance by using recent extensions, but this comes at the cost of compatibility: code that relies on newer instructions may not run on older machines. This issue is especially problematic with proprietary software that users cannot recompile. If we are targeting older machines, how can we manage this process automatically to prevent users from being discouraged?

6) Incompatibility within SW: Some software may fail to run because it becomes incompatible with other software components. Our modern HPC systems consist of many layers inter-operating through APIs. Typical stacks range from low-level drivers through operating systems and runtimes to compilers and languages. Upgrading a single layer incurs the risk of breaking the compatibility of its neighbor layers.

Upgrades of some layers may be almost mandatory, as in the case of cybersecurity. Vulnerabilities require updating the software to patch the hole. However, some developers may decide to only backport some security fixes to some recent release series, which may not be compatible with our old application or may not support some old hardware. Users are then left with a dilemma between the security of the new release and the usability of the old one.

Example III-B.4: Old software

Attempts at recompiling a recent application on an old setup may prove difficult because languages and standards evolve. New software will likely rely on new dialects (e.g., ISO C++20 constructs), which are thus unknown to old compilers. Newer compilers may be difficult or impossible to install if the operating system is too old. Similarly, when some libraries have not been updated for years, they may miss some features used by recent applications. For instance an old hwloc installation of a recent HPC cluster would be unable to report information about heterogeneous hardware, rendering some hybrid programming models unusable if they use hwloc for getting heterogeneity information.

C. Challenges of Classifying Hardware and Software Issues

Accurately determining whether a failure is due to a hardware or software issue is challenging. Misclassifications during debugging can result in incorrect mitigation actions, wasting time and resources. The task becomes even trickier when the hardware starts operating outside the nominal operation margins. Estimating performance degradation caused by changes in operational margins or system aging depends on numerous variables, and using different estimation methods can yield substantially different results. In some cases, this may result in estimations that differ by an order of magnitude [58], [59], [29], [32]. Note that, the software can also under-perform because of older hardware or other software issues, adding another layer of complexity to fault location and isolation.

Ultimately, HPC stakeholders are responsible for defining the boundaries between usable and unusable hardware and software. These boundaries can be ambiguous and should be established based on performance preferences and software maintainability. The final determination by HPC managers will define whether the systems are classified as *a refurbished HPC* or deemed *entirely unusable*. The software stack then must accurately identify which resources remain usable and how to inform it to users, potentially via the resource manager software. The following section will address the main challenges in solving the hardware and software problems we discussed.

Challenge III-C.1: Finding the source of the problem

Meta reports that debugging CPU SDCs can require months of engineering effort [51], [60], while Alibaba Cloud experienced a CPU SDC issue that required several weeks to resolve [10], and Google has reported suffering bugs in production due to a very rare failure miss-classified as SDC [61].
Classifying whether the source of misbehavior is from hardware or software presents significant challenges. Hardware faults and defects can be intertwined with software bugs, incompatible software, and underperforming software, making the task quite complex.
Finally, after identifying and isolating the fault, defect, or bug, the next challenge is determining whether the hardware and software are usable or unusable.

IV. Towards Extended Lifetime: Challenges to Practical Solutions

So far, we discussed non-nominal behavior that we expect to observe as we start increasing the lifetime of HPC machines and their resources. We have focused on non-nominal behaviors that have a concrete impact on how we use HPC machines. In the near future, the HPC community will need to identify solutions for each of these behaviors to prolong the lifespan of HPC machines. This section discusses some of these solutions. **Our goal is not to find a practical solution for each of the behaviors we discussed**. We believe this is too early at this stage and could only be covered by a community-wide research program. This position paper wants to give directions toward possible solutions along with some examples and, more importantly, highlight the research challenges that must be solved through a community-wide effort before similar solutions can be used. With this in mind, we point to possible directions to address given unusual behaviors and discuss the challenges of implementing such solutions while considering increasing the lifespan of HPCs. Note that a given class of solutions may address different non-nominal behaviors and different classes of solutions may contribute to the same non-nominal behavior. We identify and discuss three main categories of possible responses to non-nominal behavior:

1) Increase/Replace/Rethink Hardware Resources;
2) Dynamic Adaptation of Resources;
3) Increase Software Compatibility and Lifetime.

A. Increase/Replace/Rethink Hardware Resources

One of the most straightforward reactions to *non-nominal HW resource behavior*, *unusable HW resources*, and *underperforming SW due to HW*, is increasing the available resources or replacing components with newer and more capable hardware parts. This approach may include adding memory modules, upgrading accelerators, GPUs, or CPUs, or adding nodes to the system. For both solutions, increase and replace, identical hardware parts may not be available anymore due to manufacturers' end-of-life policies. Enforced hardware upgrades may require software upgrades, hence they are rarely plug-and-play solutions.

Although increasing and replacing hardware parts works to some extent for some parts of HPC machines, this is a reactive action. That is, after the problem occurs, a solution is employed. **In order to extend the lifetime of future HPC systems, it is essential to adopt proactive approaches during the early hardware design phases.** More proactive fault tolerance and mitigation strategies are not new to other research communities. Fault tolerance and mitigation are well-established domains, particularly among researchers and engineers in safety- and mission-critical applications, such as space exploration, avionics, and the automobile industry. Reliability is a fundamental requirement in these sectors, driving the development of various robust methods to address non-nominal

behaviors, such as multi-bit error detection and correction [62], aging sensors [63], and silicon lifecycle management [64], [65]. *While reliability constraints differ for HPC machines, the HPC community can and should draw insights from the test and reliability sectors to rethink hardware design.* Note that this initiative would demand a collaborative effort, as the ultimate goal would be to maintain performance at acceptable levels while integrating aging and fault tolerance into the design constraints of hardware architectures.

1) Challenges and Implications:: Even if identical or similar resources are available for replacement, aging-related challenges will still exist. The physical location of components within a system can influence their degradation rate, as some may operate under higher thermal stress or power variations, accelerating the aging process. This heterogeneity complicates system reconfiguration and load balancing, particularly when integrating newer components. Integrating new resources also raises questions about connectivity, spatial placement, and thermal management. *Should newer components be placed where older ones failed, especially in areas prone to faults? Is it feasible to connect them to the existing hardware without introducing bottlenecks?* These decisions impact performance, resilience, and maintainability.

Replacing old hardware with newer alternatives increases system heterogeneity and complicates software and hardware management. A critical issue is how the Resource and Job Management Software (RJMS) handles such environments. While some heterogeneity is common in HPC clusters, users generally expect homogeneous environments to avoid performance imbalances in parallel jobs. Existing RJMS frameworks must be extended to support finer-grained heterogeneity management. Unlike cloud environments, where heterogeneity is abstracted through virtualization layers, HPC workloads are typically communication-intensive and tightly coupled. Simplified classifications (e.g., "fast" vs. "slow" nodes) may be more effective for balancing usability and complexity [66]. Schedulers also need adaptation to reason about heterogeneous performance characteristics, such as determining which jobs should use newer hardware and based on what criteria [67].

Low-level software, including compilers and runtime libraries, must adapt to exploit new hardware capabilities while maintaining compatibility with legacy components. Compilation strategies need target-specific optimizations, which become more challenging to manage in heterogeneous environments. Inconsistent performance or feature availability across nodes can further challenge application developers and reduce overall system efficiency.

Another key challenge is how we can proactively mitigate the impact of aging hardware on HPC. Fault tolerance and mitigation are highly effective when implemented during the hardware design. However, the primary challenge of employing such approaches in HPC systems is *scalability*. Indeed, hardware changes may incur significant costs in terms of physical area and increased power consumption. While additional power and area are justified for safety- or mission-critical applications due to their nature, HPC requires more cost-effective solutions. *Therefore, the main challenge of applying proactive fault detection and correction in HPCs lies in modifying the hardware design to enable these processes without incurring excessive overhead, i.e. rethinking HPC hardware.*

Challenge IV-A.1: Increasing or replacing is never a plug-and-play situation

Increasing or replacing hardware resources will increase system complexity. Identical parts can become obsolete, and different hardware parts must be integrated into the system, creating heterogeneity. This makes scheduling, load balancing, and predicting performance more difficult. Resource and job management software must adjust to these environments, as traditional HPC users prefer uniform systems to minimize job imbalances. Compilers and low-level runtime systems must manage diverse capabilities and ensure compatibility, requiring new abstractions and adaptive mechanisms. Additionally, components may age differently depending on location. Integrating newer parts raises challenges related to placement, connectivity, and thermal constraints, particularly in systems not designed for upgrades.

To highlight the importance of rethinking HPC hardware, consider the current trend in chip design, which focuses on maximizing resource density within a chip. For example, NVIDIA Grace Hopper includes multiple HBM modules, many ARM cores, and a GPU all integrated into the chip, and consumes up to 1000W [68]. While high-density chips yield significant short-term performance gains, they may accelerate the aging process due to thermal dissipation in the long term.

Challenge IV-A.2: Proactive measures requires rethinking HPC hardware design

Integrating proactive fault detection and correction at scale in HPC systems faces cost, power, and area constraints. High-density chip designs—like NVIDIA Grace Hopper—optimize short-term performance but, in the long term, may worsen thermal stress and accelerate aging, demanding new hardware design trade-offs.

2) Recommendations for Future Research:: Future work should address several open challenges to enable more effective hardware replacement and/or resource increase in HPCs:

- **Resource management:** New scheduling models must be designed to accommodate the hardware heterogeneity that arises when resources are added or replaced. They should also incorporate behavior to cope with aging, depending of what are the platform's operator objectives. However, scheduling models have to remain simple enough for practical use in HPC environments. This includes developing abstractions that reflect essential performance characteristics without overwhelming users or system designers.
- **Hardware architecture:** The hardware architecture for HPCs has to be rethought for a better understanding of how seemingly identical hardware components degrade. For instance, in safety-critical applications, methods such as silicon lifecycle management and aging sensors are used to

extract in-field data about the health of the hardware over time. Similar approaches enable monitoring the health of the HPC hardware, allowing better (re)placement strategies. The collected data can also address aging and fault tolerance during initial design phases of future hardware architectures.

- **Compilers and low-level software:** The research should focus on adaptive compilation techniques and runtime mechanisms that dynamically adjust to varying hardware configurations while ensuring stability and correctness across execution environments while limiting code size explosion.

B. Dynamic adaptation of resources

Dynamic adaptation of resources is another possible direction to handle *non-nominal HW resource behaviors* and *unusable HW resources*. Two primary adaptation strategies are employed to address system degradation.

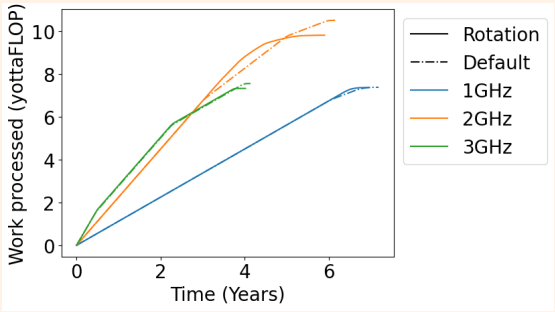
(i) *Reducing Active Resource Footprint*: This strategy involves excluding or deactivating faulty components (i.e. components whose results cannot be trusted) to minimize the system's resource usage. Measures include not scheduling unusable nodes, deactivating malfunctioning CPU cores or GPUs, or restricting use of faulty memory regions. These adaptations ensure system operation despite hardware degradation, particularly valuable in long-running HPC environments where full replacements are impractical or costly.

(ii) *Modifying Resource Usage*: this approach consists in adapting resource utilization rather than replacing or disabling components. As a reactive approach, in cases of partial hardware failure such as shifted voltage threshold due to aging, the system adjusts its usage patterns to accommodate reduced performance without deactivating the component entirely. For example, a standard approach to fix timing-related errors is to extend clock periods a with a safety margin adjusted based on hardware aging estimates [32]. This adjustment reduces operating frequency and, consequently, performance. More proactively, throughout the system lifetime workloads can be balanced to prevent future issues by distributing tasks across different resources [69]. For example, in a GPU, instead of overloading specialized hardware units (such as a matrix multiplication unit), the workload can be distributed among both the standard GPU floating-point units and the specialized ones. Implementing telemetry during hardware design, i.e., enabling runtime chip monitoring and data collection, helps effectively guide these adjustments [64]. Other examples focus on proactively optimizing supply voltage, operating clock frequency, and dynamic cooling throughout the system's lifetime to slow aging effects and enhance energy efficiency [70]. Additionally, applying power capping, i.e., limiting power usage, can improve energy consumption, hardware temperature, and performance with minimal performance degradation [71].

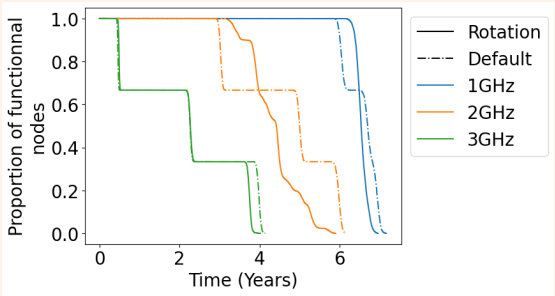
Combining reactive and proactive approaches can help the system remain operational by adapting resource usage or reducing its footprint.

Challenge IV-B.1: Aging-aware Management

In example III-A.1 we discussed the impact of heterogeneous cooling on the lifespan of compute nodes. To study a solution such as "Would relocating compute node frequently allow for a better usage of the machine?", we need to develop accurate models. For instance, using aging model such as the ones developed by Boezennec et al. [46] that compute the deterioration of each node (typically, a CPU or GPU) until they stop working, we can evaluate the strategy from different perspective.
For instance, we simulated a machine where all nodes run at a single constant speed, where there are three levels of cooling heterogeneity and compared two strategies: one that does not rotate nodes, and one that rotates nodes every year. We study the volume of work (measured in cumulative work done by the machine until no more nodes are functional):



The first observation is that in general rotation are detrimental to the total number of Flop obtained from the machine, however it allows to have more Flop earlier, because when we do not rotate, some node die earlier:



For details about the methodology of this specific example, see [72]. Using these models, a system administrator may decide the right platform strategy.

1) Challenges and Implications: The complexity of reducing active resource footprint varies by type and extent, with traditional RJMS systems excelling at handling entire node failures [73], [74], though performance impacts can occur for jobs using large allocations. More fine-grained reductions, such as disabling part of a CPU or limiting GPU memory, *will necessitate hardware and firmware support*. For instance, booting a system with a partially functional CPU requires the firmware to correctly report and support such configurations. While x86 servers offer some flexibility through dynamic

ACPI table generation, specialized architectures like GPUs and TPUs may lack this capability, underscoring the need for hardware design modifications to accommodate partial functionality. On the software side, adapting to reduced or partially functional hardware *demands changes across the system stack*. Firmwares and operating systems must be able to expose these hardware details. Then RJMS must recognize dynamic resource topologies to make informed scheduling decisions, abandoning assumptions of fixed hardware capabilities. Compilers and low-level system software must support dynamic reconfiguration and scaling, as homogeneous resource assumptions can lead to performance issues or failures when core counts, memory sizes, or GPU capabilities vary across nodes.

Challenge IV-B.2: Dynamic adaptation requires HPC systems rethinking

The challenges of handling partial failures or performance degradation in high-performance computing (HPC) systems require significant **rethinking of both hardware and software architectures**. Monitoring and interpreting issues at the component level, such as failed vector units, demands fine-grained observability and fallback mechanisms that may be limited or dependent on specific architectures. Similarly, partial hardware use, e.g. deactivating faulty CPU cores, often lacks sufficient support in specialized architectures, necessitating a redesign of HPC hardware. On the software side, managing uncertainty due to hardware variability adds complexity. RJMS systems need lightweight models to adapt to dynamic resource topologies while maintaining performance. Compilers and runtimes must also support real-time adjustments to non-nominal behavior. Traditional HPC workloads, which rely on homogeneous and predictable environments, may require new abstractions or refactoring to handle such conditions effectively.

One key challenge in modifying resource usage is *ensuring degraded components behave within acceptable bounds*. Traditional HPC scheduling assumes stable resource performance, yet aging systems exhibit increasing variability. This highlights the limitations of current RJMS, primarily addressing user uncertainty [75], [76], [77] rather than resource uncertainty. To support degraded components, systems must include monitoring mechanisms to assess resource behavior and communicate operational parameters to middleware and applications. A proactive solution would require hardware design changes aiming at monitoring performance degradation and detecting sub-component failures. Aging sensors included in the design phases would allow identification of the defective components. Modifying resource usage will directly impact also Software. Instruction set architectures offering some backward compatibility (e.g., x86 AVX-512 to AVX2 fallback) can already offer some partial solutions. However, in more flexible ISAs like ARM SVE, the implications could be more challenging, for example when a malfunctioning functional unit impacts the supported vector length. At the compilation and runtime levels, modifying hardware usage will require dynamic execution strategy reconfiguration. This includes instruction set fallback mechanisms, thread parallelism tuning, or workload redistribution across nodes with uneven performance. Without such flexibility, applications may underperform or fail due to unexpected hardware behavior.

In conclusion, managing reduced or degraded hardware resources demands a holistic approach, integrating robust monitoring, proactive solutions at the hardware level, and dynamic software adaptations. This integrated strategy ensures system reliability and efficiency in diverse computing environments.

2) Recommendations for Future Research: Implementing, as a resilience strategy, the reduction of active resource footprint and the modification of resource usage calls for coordinated improvements across the system stack:

- **Monitoring and reliability modeling:** Develop hardware-level mechanisms to proactively report and quantify functional degradation, such as variable instruction throughput or downclocked frequencies. Additionally, investigate firmware and hardware approaches for dynamic enumeration and reporting of partially functional components across architectures.
- **Resource Scheduling Models**: Design Real-Time RJMS-aware abstractions that dynamically detect and adapt to performance uncertainty in real time while minimizing complexity and overhead. This includes advancing heterogeneity-aware RJMS design to automatically detect and manage fine-grained system heterogeneity, balancing usability with system complexity.
- **Compiler and Runtime Mechanisms**: Enhance compiler and runtime support by developing adaptive strategies for rescaling jobs or redistributing workloads based on hardware degradation status and on aging models, to decelerate the system wear-out. Support for fallback implementations across instruction set versions or vector lengths, as well as runtime mechanisms to adjust parallelism and memory usage, will be key to adapt to performance uncertainty. Hence, research should focus on automatically detecting scenarios where these mechanisms are necessary and implementing these adjustments without manual intervention.

C. Increase Software Compatibility and Lifetime

A suitable direction to address *incompatible SW* and *underperforming SW due to the SW ecosystem* is spending effort in proactively increasing Software compatibility and lifetime. The HPC software stack is huge, ranging from several compilers for many different languages to multiple parallel runtimes, communication libraries, resource management systems, and many different applications. Over the years, maintaining a working stack with all these components has been difficult. As explained in Section III-B, mixing old and new software often brings issues because one has requirements that the other does not meet yet or anymore. Well-established standards have good evolution practices that ease end users' life. For instance, the upcoming Message Passing Interface (MPI) 5.0 standard will define a common Application Binary Interface (ABI) [78] between different implementations so that users can easily

switch from one to the other without recompiling. Things are more complicated when it comes to interactions between different components, for instance, an OpenMP runtime, the MPI launcher, and the resource manager. The PMIx standard [79] defined ways for them to exchange information about their needs and use, so that, for instance, they do not try to use the same resources. However, very few HPC libraries are actually able to interoperate this way, hence many issues may arise from interactions between components of the stack.

1) Challenges and Implications:: Computer users are used to the fact that upgrading "underlying layers" of their systems, such as hardware or operating systems, does not break applications. It feels natural that buying a new PC or applying OS updates adds new functionality but does not remove any. Open-source software usually maintains backward compatibility for a long time. For instance, drivers are not removed from Linux or NetBSD unless nobody uses them anymore. Some Linux distributions (Fedora 41 or Debian 12 at the time of writing) support down to Pentium 4 (shipped more than two decades ago). However, proprietary or hardware-specific software may deprecate legacy hardware support after only a few years, limiting their suitability in long-term deployments. This may especially happen for GPU drivers [57] or performance optimization tools. Moreover, they usually only release security fixes in newer releases instead of backporting them to stable releases, hence enforcing unnecessary upgrades that may break compatibility with other components.

Challenge IV-C.1: Workforce is needed to improve the long-term stability of software interfaces and features

There is a strong need to avoid removing support for hardware or features too early. This will lengthen the life of software by ensuring other components will still be able to use them in the future without requiring significant changes. This naturally pushes toward the user of open-source software, where community-driven development eases long-term maintenance. However, long-term maintenance of old software interfaces requires significant workforce which is rarely available to open-source projects.

The need to improve software's long-term stability applies to libraries, low-level drivers, and even firmwares. Firmwares are often closed source on current HPC hardware, preventing community-driven development and long-term support. Initiatives such as OpenBIOS are a good way to improve the situation by providing open-source support down to firmware.

A solution for long-term software usability is *reverse* binary compatibility, e.g., the ability to run modern software on older equipment. Apple and Microsoft developed complex systems based on dynamic binary rewriting to guarantee compatibility of existing software with newer systems (respectively Rosetta2 and Prism [80]). Conversely, Itanium targeted the high-end market with a new ISA and turned out to be a commercial failure. Another approach consists in compiling a portable intermediate bytecode. Java is notably the most popular, famous for the "Write once, run anywhere" slogan. The Common Language Interface (aka CLI) [81] popularized by Microsoft under the name .NET for a family of languages is another approach to deploying bytecodes. Nuzman et al. [82] exploited the CLI format to convey vectorization information to a JIT compiler, making efficient exploitation of SIMD instructions at little cost. Finally, native executables may be augmented with a compiler intermediate representation (resulting in so-called *fat binaries*). These programs are directly run-able, but a JIT compiler may decide to re-optimize the code before or while the code runs, taking into account the precise details of the processor [83]. fittChooser [84] also relies on embedded IR to defer optimization of a program's most processor-intensive functions until execution time. It begins by profiling the application to determine the performance characteristics that are in effect for the present execution, then generates a set of candidate variations and dynamically links them in succession to empirically measure which of them performs best. However, the cost of redeveloping existing HPC codes for new languages is high, especially when some libraries are still written in Fortran.

Besides the usual API/ABI compatibility between layers, there is also a need to better expose what they support and have a degraded mode that does not require many underlying features. This need goes from the (possibly partially broken) hardware to firmwares, operating systems, libraries, and applications. One should not assume anything about the underlying layer capabilities since those may vary with the age and state of hardware and software. If a software driver cannot turn off some broken hardware, some operating systems, such as Linux, can hide some resources or parts of them (poisoned pages, cache partitioning, offlining cores, etc.).

Challenge IV-C.2: Improve capability queries between software layers

Software should be made aware at runtime of what is supported (or not) by the underlying components, including features and resources. Heterogeneity is supported in current HPC runtimes but there are so many possible configurations that the software sometimes has to be explicitly tuned for some specific platforms. Approaches based on querying the existing hardware and measuring its performance before deciding where to allocate/execute [85] are much more portable. This will require extensive APIs and abstractions to describe the available features, which may be difficult to define in a future-proof manner.

2) Recommendations for Future Research: We believe that the work done by LLNL on the challenges of software stewardship [86] is probably highly complete. They have seen firsthand the problem of vendors unwilling to extend their software stack's life when the Aurora supercomputer was delayed. This has led to the creation of the High-Performance Software Foundation [87]. To extend the lifetime of HPC software, we recommend considering the following suggestions for future research and investments:

- **Open-source software:** Investment in open-source software

engineering effort to lengthen its life beyond the vendor's economic considerations.

- **New best practices and standards:** Developing novel best practices, nomenclatures, and standards in software design emphasizing the long-term stability of interfaces.
- **Software degraded mode:** Including degraded mode in the design of applications to encourage the use of *simpler* machines. For instance, it is important to make sure that applications can run even in the absence of dedicated hardware (GPU, TPU) or state-of-the-art library by providing a default (degraded) mode or being able to turn off some functionalities.

V. Conclusion

We have identified the main issues affecting current and future HPC machines, along with the challenges of extending their lifespan. As a community, we must move beyond isolated hardware and software fixes by embracing a comprehensive, cross-layer, co-design philosophy that encourages synergy-driven innovation. HPC software must evolve to be more adaptive and capable of operating efficiently, even with partially failing hardware components. Simultaneously, HPC hardware design must be re-conceptualized with reliability and long-term deployment as primary design objectives. Future research should prioritize developing resilient architectures, fault-aware software stacks, and adaptive mechanisms to effectively manage non-nominal behaviors that inevitably arise as HPC machines age.

VI. Acknowledgement

Parts of this work have been supported by the Inria Exploratory Project REPAS and by the Exa-DoST project (PEPR NumPEx), reference ANR-22-EXNU-0004.

Some experiments presented in this paper were carried out using the PlaFRIM experimental testbed, supported by Inria, CNRS (LaBRI and IMB), Université de Bordeaux, Bordeaux INP and Conseil Régional d'Aquitaine (see https://www.plafrim.fr/).

References

[1] J. Srinivasan, S. Adve, P. Bose, and J. Rivers, "The impact of technology scaling on lifetime reliability," in *International Conference on Dependable Systems and Networks, 2004*. Firenze, Italy: IEEE, 2004, pp. 177–186.

[2] W. Li and C. M. Tan, "Black's equation for today's ulsi interconnect electromigration reliability — a revisit," in *2011 IEEE International Conference of Electron Devices and Solid-State Circuits*. Tianjin, China: IEEE, 2011, pp. 1–2.

[3] S. J. Babu, F. Hu, L. Zhu, S. Singhal, and X. Guo, "Extending Silicon Lifetime: A Review of Design Techniques for Reliable Integrated Circuits," Mar. 2025, arXiv:2503.21165. [Online]. Available: http://arxiv.org/abs/2503.21165

[4] J. Dongarra, T. Herault, and Y. Robert, *Fault Tolerance Techniques for High-Performance Computing*. Cham: Springer International Publishing, 2015, pp. 3–85. [Online]. Available: https://doi.org/10.1007/978-3-319-20943-2_1

[5] G. Aupy, Y. Robert, F. Vivien, and D. Zaidouni, "Checkpointing algorithms and fault prediction," *Journal of Parallel and Distributed Computing*, vol. 74, no. 2, pp. 2048–2064, 2014.

[6] J. Meza, Q. Wu, S. Kumar, and O. Mutlu, "Revisiting memory errors in large-scale production data centers: Analysis and modeling of new trends from the field," in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. Rio de Janeiro, Brazil: IEEE, 2015, pp. 415–426.

[7] V. Sridharan, J. Stearley, N. DeBardeleben, S. Blanchard, and S. Gurumurthi, "Feng shui of supercomputer memory: positional effects in DRAM and SRAM faults," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '13. New York, NY, USA: Association for Computing Machinery, 2013. [Online]. Available: https://doi.org/10.1145/2503210.2503257

[8] J. D. Guerrero Balaguera, J. E. Rodriguez Condia, F. Fernandes Dos Santos, M. Sonza Reorda, and P. Rech, "Understanding the effects of permanent faults in GPU's parallelism management and control units," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3581784.3607086

[9] G. Aupy, A. Benoit, T. Hérault, Y. Robert, F. Vivien, and D. Zaidouni, "On the combination of silent error detection and checkpointing," in *2013 IEEE 19th Pacific Rim International Symposium on Dependable Computing*. Vancouver, BC, Canada: IEEE, 2013, pp. 11–20.

[10] S. Wang, G. Zhang, J. Wei, Y. Wang, J. Wu, and Q. Luo, "Understanding Silent Data Corruption in Processors for Mitigating its Effects," *ACM Trans. Archit. Code Optim.*, vol. 21, no. 4, pp. 84:1–84:27, nov 2024. [Online]. Available: https://dl.acm.org/doi/10.1145/3690825

[11] P. H. Hochschild, P. J. Turner, J. C. Mogul, R. K. Govindaraju, P. Ranganathan, D. E. Culler, and A. Vahdat, "Cores that don't count," in *Proc. 18th Workshop on Hot Topics in Operating Systems (HotOS 2021)*. New York, NY, USA: ACM, 2021, pp. 1 – 8.

[12] H. D. Dixit, L. Boyle, G. Vunnam, S. Pendharkar, M. Beadon, and S. Sankar, "Detecting silent data corruptions in the wild," 2022. [Online]. Available: https://arxiv.org/abs/2203.08989

[13] S. Gupta, T. Patel, C. Engelmann, and D. Tiwari, "Failures in large scale systems: long-term measurement, analysis, and implications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. New York, NY, USA: ACM, 2017, pp. 1–12.

[14] E. Yılmaz, "Redundancy and Reliability for an HPC Data Centre," Zenodo, Tech. Rep., Jul. 2012. [Online]. Available: https://zenodo.org/records/810592

[15] S. Jha, V. Formicola, C. D. Martino, M. Dalton, W. T. Kramer, Z. Kalbarczyk, and R. K. Iyer, "Resiliency of HPC Interconnects: A Case Study of Interconnect Failures and Recovery in Blue Waters," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 6, pp. 915–930, 2018.

[16] J. Lüttgau, M. Kuhn, K. Duwe, Y. Alforov, E. Betke, J. Kunkel, and T. Ludwig, "Survey of storage systems for high-performance computing," *Supercomputing Frontiers and Innovations*, vol. 5, no. 1, 2018.

[17] G. Michelogiannakis, B. Klenk, B. Cook, M. Y. Teh, M. Glick, L. Dennison, K. Bergman, and J. Shalf, "A Case For Intra-rack Resource Disaggregation in HPC," *ACM Trans. Archit. Code Optim.*, vol. 19, no. 2, Mar. 2022. [Online]. Available: https://doi.org/10.1145/3514245

[18] M. Chadha, E. Arima, A. Raoofy, M. Gerndt, and M. Schulz, "Sustainability in HPC: Vision and opportunities," in *Proceedings of the SC'23 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis*. New York, NY, USA: ACM, 2023, pp. 1876–1880.

[19] F. Singer, "Nsf extends lifespan of tacc's stampede2 supercomputer through june 2023," https://tacc.utexas.edu/news/latest-news/2022/02/21/nsf-extends-lifespan-taccs-stampede2-supercomputer-through-june-2023/, 2022, accessed: 2025-03-31.

[20] C. Trueman, "Summit supercomputer has life extended by 12 months," https://www.datacenterdynamics.com/en/news/summit-supercomputer-has-life-extended-by-12-months/, 2024, accessed: 2025-03-31.

[21] J. Collins, "Theta supercomputer set to retire: A look back at its impact on science at argonne and beyond," `https://www.anl.gov/article/theta-supercomputer-set-to-retire-a-look-back-at-its-impact-on-science-at-argonne-and-beyond`, 2023, accessed: 2025-03-31.

[22] S. Rana and K. Brandt, "Circular economy at work in google data centers," *Google & the Ellen MacArthur Foundation*, vol. 1, pp. 1

– 11, 2016. [Online]. Available: https://static.googleusercontent.com/media/www.google.com/en//green/pdf/data-center-case-study.pdf

[23] T. Gamblin, B. de Supinski, B. Van Essen, A. Bertsch, T. D'Hooge, M. Leininger, J. Hill, B. Behlendorf, and T. Quinn, "Hpc center of the future: R&d acquisition intent," Lawrence Livermore National Laboratory (LLNL), Livermore, CA (United States), Tech. Rep., 2024.

[24] C. Constantinescu, "Intermittent faults and effects on reliability of integrated circuits," in *2008 Annual Reliability and Maintainability Symposium*. Las Vegas, NV, USA: IEEE, 2008, pp. 370–374.

[25] ——, "Trends and challenges in VLSI circuit reliability," *IEEE Micro*, vol. 23, no. 4, pp. 14–19, 2003.

[26] J. Srinivasan, S. Adve, P. Bose, and J. Rivers, "Lifetime reliability: toward an architectural solution," *IEEE Micro*, vol. 25, no. 3, pp. 70–80, 2005.

[27] D. Tiwari, S. Gupta, J. Rogers, D. Maxwell, P. Rech, S. Vazhkudai, D. Oliveira, D. Londo, N. DeBardeleben, P. Navaux, L. Carro, and A. Bland, "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)*. Burlingame, CA, USA: IEEE, 2015, pp. 331–342.

[28] C. Constantinescu, "Impact of deep submicron technology on dependability of vlsi circuits," in *Proceedings International Conference on Dependable Systems and Networks*. Washington, D.C., USA: IEEE, 2002, pp. 205–209.

[29] H. Amrouch, V. M. van Santen, T. Ebi, V. Wenzel, and J. Henkel, "Towards interdependencies of aging mechanisms," in *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, IEEE. San Jose, CA, USA: IEEE, 2014, pp. 478–485.

[30] F. Oboril and M. B. Tahoori, "Extratime: Modeling and analysis of wearout due to transistor aging at microarchitecture-level," in *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012)*, IEEE. Boston, MA, USA: IEEE, 2012, pp. 1–12.

[31] K.-C. Wu, M.-C. Lee, D. Marculescu, and S.-C. Chang, "Mitigating lifetime underestimation: A system-level approach considering temperature variations and correlations between failure mechanisms," in *2012 Design, Automation and Test in Europe Conference and Exhibition (DATE)*. Dresden, Germany: IEEE, 2012, pp. 1269–1274.

[32] A. Masrur, P. Kindt, M. Becker, S. Chakraborty, V. Kleeberger, M. Barke, and U. Schlichtmann, "Schedulability analysis for processors with aging-aware autonomic frequency scaling," in *2012 IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*. Seoul, South Korea: IEEE, 2012, pp. 11–20.

[33] S. Hamdioui, D. Gizopoulos, G. Guido, M. Nicolaidis, A. Grasset, and P. Bonnot, "Reliability challenges of real-time systems in forthcoming technology nodes," in *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. Grenoble, France: IEEE, 2013, pp. 129–134.

[34] A. J. Strojwas, K. Doong, and D. Ciplickas, "Yield and reliability challenges at 7nm and below," in *2019 Electron Devices Technology and Manufacturing Conference (EDTM)*. Singapore: IEEE, 2019, pp. 179–181.

[35] IEEE, "The international roadmap for devices and systems: 2022," in *Institute of Electrical and Electronics Engineers (IEEE)*. New Jersey, United States: IEEE, 2022, pp. 1–86.

[36] J.-G. Ahn, R. Nathanael, I.-R. Chen, P.-C. Yeh, and J. Chang, "Product lifetime estimation in 7nm with large data of failure rate and Si-based thermal coupling model," in *2021 IEEE International Reliability Physics Symposium (IRPS)*. virtual: IEEE, 2021, pp. 1–6.

[37] J.-W. Han, M. Meyyappan, and J. Kim, "Single event hard error due to terrestrial radiation," in *2021 IEEE International Reliability Physics Symposium (IRPS)*. Monterey, CA, USA: IEEE, 2021, pp. 1–6.

[38] S. Pae, J. Maiz, C. Prasad, and B. Woolery, "Effect of BTI degradation on transistor variability in advanced semiconductor technologies," *IEEE Transactions on Device and Materials Reliability*, vol. 8, no. 3, pp. 519–525, 2008.

[39] C. Prasad, L. Jiang, D. Singh, M. Agostinelli, C. Auth, P. Bai, T. Eiles, J. Hicks, C. H. Jan, K. Mistry, S. Natarajan, B. Niu, P. Packan, D. Pantuso, I. Post, S. Ramey, A. Schmitz, B. Sell, S. Suthram, J. Thomas, C. Tsai, and P. Vandervoorn, "Self-heat reliability considerations on intel's 22nm tri-gate technology," in *2013 IEEE International Reliability Physics Symposium (IRPS)*. Monterey, CA, USA: IEEE, 2013, pp. 5D.1.1–5D.1.5.

[40] S. Hamdioui, D. Gizopoulos, G. Guido, M. Nicolaidis, A. Grasset, and P. Bonnot, "Reliability challenges of real-time systems in forthcoming technology nodes," in *Design, Automation Test in Europe Conf. Exhibition (DATE)*. Grenoble, France: IEEE, March 2013, pp. 129–134.

[41] D. Kraak, M. Taouil, S. Hamdioui, P. Weckx, F. Catthoor, A. Chatterjee, A. Singh, H. Wunderlich, and N. Karimi, "Device aging: A reliability and security concern," in *European Test Symp. (ETS)*. Bremen, Germany: IEEE, May 2018, pp. 1–10.

[42] F. Gabbay and A. Mendelson, "Asymmetric aging effect on modern microprocessors," *Microelectronics Reliability*, vol. 119, p. 114090, 2021.

[43] Y.-G. Chen, I.-C. Lin, and J.-T. Ke, "ROAD: Improving reliability of multi-core system via asymmetric aging," in *Int. Conf. Computer-Aided Design (ICCAD)*. California, USA: IEEE, 2019, pp. 1–8.

[44] S. Rehman, M. Shafique, and J. Henkel, *Reliable Software for Unreliable Hardware: A Cross Layer Perspective*. New York, USA: Springer Publishing, 2016.

[45] S. Zafar, A. Kumar, E. Gusev, and E. Cartier, "Threshold voltage instabilities in high-/spl kappa/ gate dielectric stacks," *IEEE Transactions on Device and Materials Reliability*, vol. 5, no. 1, pp. 45–64, 2005.

[46] R. Boëzennec, F. Dufossé, G. Pallez, and A. Tremodeux, "Improving supercomputer usage with aging awareness," 2025.

[47] A. Avizienis, J.-C. Laprie, B. Randell *et al.*, "Fundamental concepts of dependability," *Technical Report Series-University of Newcastle upon Tyne Computing Science*, vol. 1, pp. 1–21, 2001.

[48] D. Rodopoulos, G. Psychou, M. M. Sabry, F. Catthoor, A. Papanikolaou, D. Soudris, T. G. Noll, and D. Atienza, "Classification framework for analysis and modeling of physically induced reliability violations," *ACM Comput. Surv.*, vol. 47, no. 3, Feb. 2015. [Online]. Available: https://doi.org/10.1145/2678276

[49] G. Papadimitriou and D. Gizopoulos, "Silent data corruptions: Microarchitectural perspectives," *IEEE Transactions on Computers*, vol. 72, no. 11, pp. 3072–3085, 2023.

[50] S. Murali, T. Theocharides, N. Vijaykrishnan, M. Irwin, L. Benini, and G. De Micheli, "Analysis of error recovery schemes for networks on chips," *IEEE Design & Test of Computers*, vol. 22, no. 5, pp. 434–442, 2005.

[51] H. D. Dixit, S. Pendharkar, M. Beadon, C. Mason, T. Chakravarthy, B. Muthiah, and S. Sankar, "Silent data corruptions at scale," 2021. [Online]. Available: https://arxiv.org/abs/2102.11245

[52] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of dram disturbance errors," in *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, 2014, pp. 361–372.

[53] H. Amrouch, B. Khaleghi, A. Gerstlauer, and J. Henkel, "Reliability-aware design to suppress aging," in *Proceedings of the 53rd Annual Design Automation Conference*. New York, NY, USA: ACM, 2016, pp. 1–6.

[54] V. M. van Santen, H. Amrouch, N. Parihar, S. Mahapatra, and J. Henkel, "Aging-aware voltage scaling," in *2016 Design, Automation and Test in Europe Conference and Exhibition (DATE)*. Dresden, Germany: IEEE, 2016, pp. 576–581.

[55] O. Mutlu and J. S. Kim, "Rowhammer: A retrospective," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 8, pp. 1555–1571, 2020.

[56] F. Broquedis, J. Clet-Ortega, S. anie Moreaud, N. Furmento, B. Goglin, G. Mercier, and S. T. an d Raymond Namyst, "hwloc: a Generic Framework for Managing Hardware Affinities in HPC Applications," in *Proceedings of the 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP2010)*. Pisa, Italia: IEEE Computer Society Press, Feb. 2010, pp. 180–186. [Online]. Available: http://hal.inria.fr/inria-00429889

[57] NVIDIA Corporation, *CUDA Toolkit Release Notes: Deprecated Architectures*, NVIDIA, 2024, accessed: 2025-04-07. [Online]. Available: https://docs.nvidia.com/cuda/cuda-toolkit-release-notes/index.html#deprecated-architectures

[58] J. B. Velamala, K. B. Sutaria, T. Sato, and Y. Cao, "Aging statistics based on trapping/detrapping: Silicon evidence, modeling and long-term prediction," in *2012 IEEE International Reliability Physics Symposium (IRPS)*. Anaheim, CA, USA: IEEE, 2012, pp. 2F.2.1–2F.2.5.

[59] W. Wang, V. Reddy, A. T. Krishnan, R. Vattikonda, S. Krishnan, and Y. Cao, "Compact modeling and simulation of circuit reliability for 65-nm CMOS technology," *IEEE Transactions on Device and Materials Reliability*, vol. 7, no. 4, pp. 509–517, 2007.

[60] H. D. Dixit, S. Pendharkar, M. Beadon, C. Mason, T. Chakravarthy, B. Muthiah, and S. Sankar, "Silent Data Corruptions at Scale," *ArXiv*, vol. 1, pp. 1–8, feb 2021, arXiv:2102.11245. [Online]. Available: http://arxiv.org/abs/2102.11245

[61] D. F. Bacon, "Detection and prevention of silent data corruption in an exabyte-scale database system," in *The 18th IEEE Workshop on Silicon*

Errors in Logic–System Effects. New York, U.S.A.: IEEE, 2022, pp. 1–5.

[62] C. Wilkerson, A. R. Alameldeen, Z. Chishti, W. Wu, D. Somasekhar, and S.-l. Lu, "Reducing cache power with low-cost, multi-bit error-correcting codes," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ser. ISCA '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 83–93. [Online]. Available: https://doi.org/10.1145/1815961.1815973

[63] M. Omaña, D. Rossi, N. Bosio, and C. Metra, "Low cost nbti degradation detection and masking approaches," *IEEE Transactions on Computers*, vol. 62, no. 3, pp. 496–509, 2013.

[64] M. Tahoori, S. M. Ghasemi, and Y. Zorian, "Silicon Lifecycle Management (SLM): Requirements, Trends, and Opportunities," *IEEE Design & Test*, vol. 42, no. 1, pp. 28–38, Feb. 2025. [Online]. Available: https://ieeexplore.ieee.org/document/10662890

[65] Synopsys, "Silicon lifecycle management: Actionable silicon insights through intelligent measurement and analysis," Synopsys, Tech. Rep., 2023. [Online]. Available: https://www.synopsys.com/content/dam/synopsys/solutions/slm/whitepapers/slm-whitepaper.pdf

[66] S. Zrigui, R. Y. de Camargo, A. Legrand, and D. Trystram, "Improving the performance of batch schedulers using online job runtime classification," *Journal of Parallel and Distributed Computing*, vol. 164, pp. 83–95, 2022.

[67] C. Zimmer, D. Maxwell, S. McNally, S. Atchley, and S. S. Vazhkudai, "GPU age-aware scheduling to improve the reliability of leadership jobs on titan," in *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. Dallas, TX, USA: IEEE, 2018, pp. 83–93.

[68] NVIDIA Corporation, "NVIDIA Grace Hopper Superchip Architecture Whitepaper," NVIDIA Corporation, Tech. Rep., 2023. [Online]. Available: https://resources.nvidia.com/en-us-grace-cpu/nvidia-grace-hopper

[69] S. Es'haghi and M. Eshghi, "Aging-aware scheduling and binding in high-level synthesis considering workload effects," *Microelectronics Reliability*, vol. 106, p. 113549, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S002627141930705X

[70] E. Mintarno, J. Skaf, R. Zheng, J. B. Velamala, Y. Cao, S. Boyd, R. W. Dutton, and S. Mitra, "Self-tuning for maximized lifetime energy-efficiency in the presence of circuit aging," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 5, pp. 760–773, 2011.

[71] D. Zhao, S. Samsi, J. McDonald, B. Li, D. Bestor, M. Jones, D. Tiwari, and V. Gadepally, "Sustainable supercomputing for ai: Gpu power capping at hpc scale," in *Proceedings of the 2023 ACM Symposium on Cloud Computing*, ser. SoCC '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 588–596. [Online]. Available: https://doi.org/10.1145/3620678.3624793

[72] R. Boezennec. (2025) Scripts for "increasing the lifetime of hpc machines: Issues, implications and open challenges". https://gitlab.inria.fr/aging-hpc-pub/script-of-the-paper-increasing-the-lifetime-of-hpc-machines-issues-implications-and-open-challenges.

[73] G. Bosilca, A. Bouteiller, A. Guermouche, T. Herault, Y. Robert, P. Sens, and J. Dongarra, "Failure detection and propagation in hpc systems," in *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. Salt Lake City: IEEE Press, 2016, pp. 312–322.

[74] D. Zhong, A. Bouteiller, X. Luo, and G. Bosilca, "Runtime level failure detection and propagation in hpc systems," in *Proceedings of the 26th European MPI Users' Group Meeting*, ser. EuroMPI '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3343211.3343225

[75] F. Boito, G. Pallez, and L. Teylo, "The role of storage target allocation in applications' i/o performance with beegfs," in *2022 IEEE International Conference on Cluster Computing (CLUSTER)*, IEEE. Heidelberg, Germany: IEEE, 2022, pp. 267–277.

[76] M. Skutella and M. Uetz, "Stochastic machine scheduling with precedence constraints," *SIAM Journal on Computing*, vol. 34, no. 4, pp. 788–802, 2005.

[77] M. Mastrolilli, N. Mutsanas, and O. Svensson, "Approximating single machine scheduling with scenarios," in *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*. Berlin, Heidelberg: Springer, 2008, pp. 153–164.

[78] J. Hammond, L. Dalcin, E. Schnetter, M. PéRache, J.-B. Besnard, J. Brown, G. B. Gadeschi, S. Byrne, J. Schuchart, and H. Zhou, "Mpi application binary interface standardization," in *Proceedings of the 30th European MPI Users' Group Meeting*, ser. EuroMPI '23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3615318.3615319

[79] R. H. Castain, D. Solt, J. Hursey, and A. Bouteiller, "Pmix: process management for exascale environments," in *Proceedings of the 24th European MPI Users' Group Meeting*, ser. EuroMPI '17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3127024.3127027

[80] B. Wilson, "What is microsoft's prism? explaining the emulation engine for windows on arm and why it's compared to apple's rosetta 2," https://www.windowscentral.com/software-apps/what-is-microsoft-prism, 2024, accessed: 2025-04-11.

[81] *Common Language Infrastructure (CLI) Partitions I to IV*, 4th ed., ECMA International, Rue du Rhône 114, 1204 Geneva, Switzerland, Jun. 2006.

[82] D. Nuzman, S. Dyshel, E. Rohou, I. Rosen, K. Williams, D. Yuste, A. Cohen, and A. Zaks, "Vapor SIMD – auto-vectorize once, run everywhere," in *Proceedings of the International Symposium on Code Generation and Optimization (CGO)*. Chamonix, France: IEEE, Apr. 2011, pp. 151–160.

[83] D. Nuzman, R. Eres, S. Dyshel, M. Zalmanovici, and J. Castanos, "JIT technology with C/C++: Feedback-directed dynamic recompilation for statically compiled languages," *ACM Trans. Archit. Code Optim.*, vol. 10, no. 4, Dec. 2013. [Online]. Available: https://doi.org/10.1145/2541228.2555315

[84] A. A. AP, C. Le Bon, B. Hawkins, and E. Rohou, "fittChooser: A dynamic feedback based fittest optimization chooser," in *2018 International Conference on High Performance Computing & Simulation (HPCS)*. Orléans, France: IEEE, 2018, pp. 98–105.

[85] C. Augonnet, S. Thibault, R. Namyst, and P.-A. Wacrenier, "StarPU: a unified platform for task scheduling on heterogeneous multicore architectures," *Concurrency and Computation: Practice and Experience*, vol. 23, no. 2, pp. 187–198, 2011. [Online]. Available: https://inria.hal.science/inria-00550877

[86] J. Hittinger, G. Abdulla, D. Ahn, C. Follett, J. Foraker, S. Futral, T. Gamblin, M. Gamboa, M. Goldman, K. Halliday *et al.*, "Llnl response to the doe ascr rfi," stewardship of software for scientific and high-performance computing"," Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), Tech. Rep., 2021.

[87] High Performance Software Foundation. (2025) High performance software foundation. HPSF. [Online]. Available: https://hpsf.io/