frontiers | Frontiers in Physics

TYPE Review
PUBLISHED 28 April 2025
DOI 10.3389/fphy.2025.1542474

OPEN ACCESS

EDITED BY
Andreas Redelbach,
Frankfurt Institute for Advanced Studies, Germany

REVIEWED BY
Dongwook Lee,
University of California, Santa Cruz, United States
Jürgen Knödlseder,
UMR5277 Institut de Recherche en Astrophysique et Planétologie (IRAP), France

*CORRESPONDENCE
Estela Suarez,
e.suarez@fz-juelich.de

RECEIVED 09 December 2024
ACCEPTED 19 February 2025
PUBLISHED 28 April 2025

CITATION
Suarez E, Amaya J, Frank M, Freyermuth O, Girone M, Kostrzewa B and Pfalzner S (2025) Energy efficiency trends in HPC: what high-energy and astrophysicists need to know.
Front. Phys. 13:1542474.
doi: 10.3389/fphy.2025.1542474

Energy efficiency trends in HPC: what high-energy and astrophysicists need to know

Estela Suarez[1,2,3]*, Jorge Amaya[4], Martin Frank[5], Oliver Freyermuth[6], Maria Girone[7], Bartosz Kostrzewa[8] and Susanne Pfalzner[1]

[1]Jülich Supercomputing Centre, Forschungszentrum Juelich GmbH, Jülich, Germany, [2]SiPEARL, Maisons-Laffitte, France, [3]Institute of Computer Science, Rheinische Friedrich-Wilhelms-Universität Bonn, Bonn, Germany, [4]European Space Agency, Space Weather Office, Space Safety Programme, ESA/ESOC, Darmstadt, Germany, [5]Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany, [6]Physikalisches Institut, Rheinische Friedrich-Wilhelms-Universität Bonn, Bonn, Germany, [7]CERN Openlab, CERN, Geneva, Switzerland, [8]Helmholtz-Institut für Strahlen- und Kernphysik (Theorie), Rheinische Friedrich-Wilhelms-Universität Bonn, Bonn, Germany

The growing energy demands of High Performance Computing (HPC) systems have made energy efficiency a critical concern for system developers and operators. However, HPC users are generally less aware of how these energy concerns influence the design, deployment, and operation of supercomputers even though they experience the consequences. This paper examines the implications of HPC's energy consumption, providing an overview of current trends aimed at improving energy efficiency. We describe how hardware innovations such as energy-efficient processors, novel system architectures, power management techniques, and advanced scheduling policies do have a direct impact on how applications need to be programmed and executed on HPC systems. For application developers, understanding how these new systems work and how to analyse and report the performances of their own software is critical in the dialog with HPC system designers and administrators. The paper aims to raise awareness about energy efficiency among users, particularly in the high energy physics and astrophysics domains, offering practical advice on how to analyse and optimise applications to reduce their energy consumption without compromising on performance.

KEYWORDS

high performance computing, HPC, energy efficiency, monitoring, programming, application optimisation

1 Introduction

Computational methods are considered the third pillar of science, together with theory and experiments [1, 2]. As science advances, problem complexity grows and the volumes of data necessary to extract scientific conclusions as well, increasing the demand for computing and data management capabilities. In consequence, nowadays all scientific fields, and especially Astrophysics and High Energy Physics (HEP), heavily rely on the use of supercomputers. High Performance Computing (HPC) infrastructures are absolutely necessary to run computer simulations that test theories explaining the

fundamental nature of matter and the forces that rule its behavior – all the way from femtometer scales up to the size of the Universe –, as well as to analyse the data generated by physical experiments and observations. This makes HPC infrastructures as important for the physical sciences as experimental instrumentation, such as telescopes, satellites, or particle accelerators. While relatively few of the latter kind of instruments are deployed worldwide, the number of HPC systems and their sizes are steadily growing, driven by the increasing computational demands from all research and engineering fields, further enhanced by the relatively recent exponential growth in training Artificial Intelligence (AI) models [3]. Therefore, even if all science infrastructures have a high energy consumption and in consequence a large carbon footprint, increasing energy efficiency in HPC bears the highest potential to reduce the environmental impact of science overall. Therefore, many scientific communities have started to report on energy consumption in scientific work, and some journals start demanding this information.

Considering one individual run of an application on an HPC system, if its energy consumption diminishes, also its carbon or environmental footprint become smaller. However, it shall be mentioned that the gain is often overcompensated by either executing more runs of the same application, running more applications, or even installing larger HPC systems for the same costs, so that the overall environmental impact is not reduced. This is generally referred to as the *Jevons paradox* [4] or *rebound effect*, which can only be solved by decision makers (i.e., funding agencies or governments) setting up strict upper limits on the overall consumption. Therefore, we focus in this paper strictly on energy efficiency and refrain from discussing carbon or environmental impact. However, this caveat does neither diminish the importance of striving for the maximum energy efficiency in HPC, nor the need of HPC users to be aware of and contribute to these efforts. Because while the gain in energy efficiency of individual jobs do not necessarily materialise in overall producing less CO_2, it does lead to a higher *scientific throughput*: more scientific results are produced using the HPC system for the same amount of time and energy.

HPC providers and operators are applying a variety of measures to maximise the energy efficiency of HPC infrastructures [212]. Energy savings can be achieved by selecting energy efficient hardware in the first place, by operating it with system software that ensure maximum utilisation of resources, and by running optimised applications. While HPC sites can directly impact the former aspects, the latter is in the hands of application developers alone. Strategies such as reporting the energy consumption per job back to users, or awarding more compute time to applications that are more energy efficient, are envisioned. This makes it therefore very important for *domain scientists* (i.e., researchers in specific scientific areas, such as biology, material science, astrophysics, high energy physics, etc.) to be aware of the main factors contributing to the energy consumption of their codes. Furthermore, it is important that HPC users understand the impact that some of the energy-saving measures applied by hardware developers and system operators have on the way supercomputers are exposed to the end users.

Astrophysics and HEP are amongst the strongest consumers of compute time (core and node-hours) on public HPC systems (see Figure 1 and, e.g., [5–8]), and therefore also in terms of energy. It is therefore especially important that application developers and users in these areas know about the techniques and strategies that they can apply to minimise the energy footprint of their HPC workloads. This paper gives an overview of what is done in HPC at the hardware and software level, with specific recommendations for application developers and users. It aims at raising their awareness, motivating them to work on improving not only the performance of their applications but also the scientific output per Watt they are able to produce.

Contributions of this paper are:

- Summarise the energy-saving techniques applied in HPC, describing them from the user perspective and highlighting their impact on the user-space. Section 2 also reports general recommendations for users and application developers, which apply in principle to any scientific domain.
- Review the historical evolution and specific concerns of applications in computational astrophysics, particularly focusing in the areas of planetary and solar astrophysics, space weather, and space physics. Section 3 gives specific advice for the astrophysics users and application developers, which go beyond the general comments stated in Section 2.
- Review the historical evolution and specific concerns of computational high-energy physics, particularly in the subareas of experimental collider physics (Section 4) and lattice quantum field theory (Section 5). These two sections focus on aspects particularly relevant for high-energy physicists.
- Summarise the main observations or recommendations to application developers and users to facilitate readers identifying the take-away messages.

2 Energy efficiency trends in high performance computing

Energy efficiency is a major factor in the design and operation of today's HPC systems. Hosting sites build up data centres – the building and physical infrastructure in which the HPC systems are physically located – so that the energy used for cooling and operation of a supercomputer is minimised, and the unavoidable waste heat can be reused. Because water can absorb more heat than air, water cooling systems are more efficient than air cooling ones. Compute nodes are integrated in blades and chassis that include Direct Liquid Cooling (DLC) pipes, in direct contact with the hottest electronic components on the motherboard (Central Processing Units (CPUs), Graphics Processing Units (GPUs), and memory). Water circulates from one blade to the next, absorbing the heat in the process, and creating an internal water loop that reaches all elements within one computer rack. The entry temperature of water in this internal loop is in the range of 35-45 °C, gaining ~5°C during its pass through the computer. A heat exchanger transfers the gained heat from the internal water loop into a secondary one, which runs across the whole data centre. At most HPC sites, particularly in central and northern Europe where the outside air temperature is all year round well below the aforementioned values, the secondary water loop simply runs out of the building and is cooled down via dry

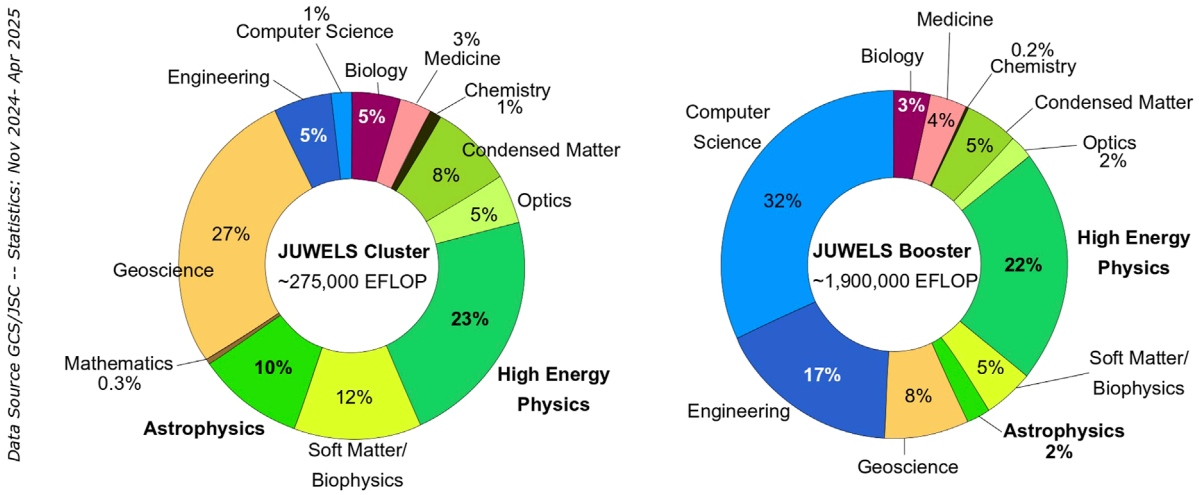

FIGURE 1
Statistics of allocated compute time per scientific domain in the period November 2024 to April 2025, on the JUWELS supercomputer [15]. Left shows the CPU module, aka *JUWELS Cluster*. Right shows the GPU module, aka *JUWELS Booster*. Astrophysics and HEP (the latter being to a large extent LQCD applications) account together for between 1/4 and 1/3 of the available compute time. Data Source: GCS/JSC. Data available in [210].

coolers (basically ventilators in contact with outside air). A further optimisation is to utilise the warm water from the secondary loop to heat neighboring buildings, e.g., offices or living spaces, saving money and energy on other heating infrastructures, and therefore maximising energy efficiency.

While HPC centres are already applying energy efficiency measures voluntarily, policies mandating efficient use of computational resources are being put into place. For example, the European Union regulates data centres through its Energy Efficiency Directive [9], last updated in 2023. The directive contains energy efficiency targets, energy savings obligations, requirements for the establishment of energy management systems and reporting duties specifically for data centres.

2.1 System hardware

Thanks to miniaturisation in microelectronics, historically more and more transistors have been included in the same area of silicon. Smaller structures also mean that electrical signals travel shorter distances across the chip, which implies lower energy consumption. This phenomenon has allowed processors in the last decades to become more computationally powerful and energy efficient from generation to generation. The exponential increase of computing power over time has been formulated in *Moore's law* [10]. Until around 2004 miniaturisation was accompanied by a steady increase of operational frequency, following the *Dennard scaling*: faster CPU clocks mean faster operations and therefore more computing performance. But miniaturisation and high frequency have as consequence higher power density (more power per area) due to leakage currents, and more generated heat, to a point where electronics would get damaged. The end of Dennard scaling marked the end of single-core CPUs. After that moment in time more compute power could only be added by increasing the number of cores in the CPU, and parallelising tasks across them (see Figure 2).

When endless miniaturisation of CPUs is impossible and operational frequency cannot grow any further, *specialisation* might at least bring some added performance. While CPUs are *general purpose* processors aiming at solving all possible tasks, compute *accelerators* are devices designed to solve *a limited class* of tasks in the most efficient way possible. Some accelerators used in HPC are GPUs, many-core processors, Field Programmable Gate Arrays (FPGAs), and AI-accelerators. While CPUs are still needed to perform some tasks, one can obtain the computational results faster and consuming less energy by employing accelerators to execute exactly the operations at which they excel.

The principle behind *heterogeneous architectures* is to combine CPUs with accelerators (most frequently GPUs) to build supercomputers and deliver the highest performance in an energy efficient way. The high-level architecture differences between CPUs and GPUs are schematically described in Figure 3. GPUs are equipped with a very large number of execution units for different types of arithmetic operations. Lacking advanced CPU features such as out-of-order execution, GPUs have higher latency for each individual operation, but they enable very high throughput through massive parallelism. Because their operating clock frequency is lower, the overall power consumption per operation (Watt per Floating Point Operation (FLOP)) is lower (better) on GPUs than on CPUs.

Typically, one, two or more GPUs are connected to a CPU via a network interface called Peripheral Component Interconnect Express (PCIe). Historically, the CPU had to actively participate in inter-device and inter-network data transfers, resulting in a communications bottleneck across PCIe as its bandwidth is one or more orders of magnitude lower than CPU-to-memory or GPU-to-memory links. Inter-GPU network links such as NVlink and advanced communication features such as GPU-aware Message Passing Interface (MPI) have strongly improved the situation. The current trend goes even further (see Figure 2), integrating CPU and GPU chips in the same package, or even as chiplets on the

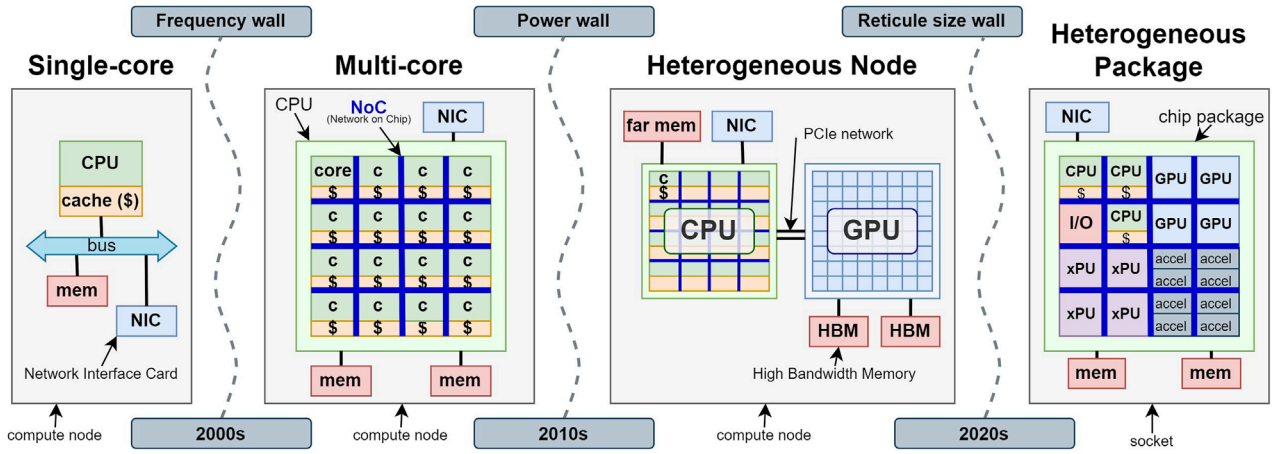

FIGURE 2
Evolution of processor architectures over time: from single-core CPUs, to many-core CPUs, to heterogeneous compute nodes with CPU and GPUs. The most recent trend goes towards specialised chip designs that are internally heterogeneous, combining different processing technologies in the form of chiplets.

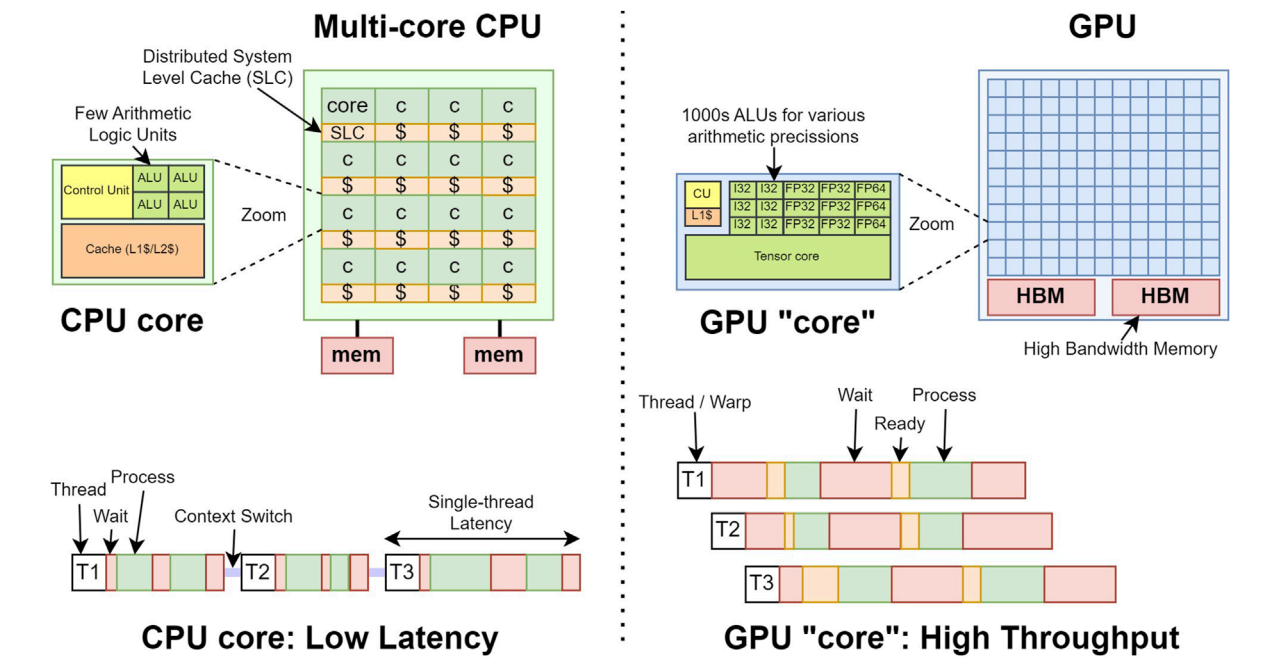

FIGURE 3
Schematic description of the architectures of a CPU (left) and a GPU (right). CPUs typically contain a limited amount of large, complex, highly capable processing cores, while GPUs contain thousands of simple arithmetic execution units (increasingly for low-precision operations). CPUs are therefore good for latency limited applications, while GPUs serve best highly parallel workloads.

same substrate [11, 12], improving CPU-GPU communication and enabling cache coherency between the two.

It is worth noting that at the same time as heterogeneity is increasing at the package level, the trend at the system level is towards *resource disaggregation* or *modularity*: clusters with different node configurations (CPU-only, CPU + GPU, accelerator-only, quantum processors, etc.) are connected to each other via a common high-speed network. The resource scheduler allows users to simultaneously allocate resources on different partitions. In cases where the MPI library has been adapted, an application can even communicate across module boundaries via MPI. This system design, sometimes called *Modular Supercomputing Architecture (MSA)* [13, 14], is used in several systems around the world (e.g., [15–19]). All these systems couple CPU-only modules with GPU-accelerated modules or partitions. The concept itself allows for further heterogeneity, using different interconnect technologies,

			Examples
	Applications		Astrophysics, High Energy Physics, LQCD, Climate & Meteorology, CFD, Protein Dynamics,...
Programming Environm.	Programming	High-Level Portability	Kokkos, SYCL, RAJA, ALPAKA, DaCE, Mamba, PyTorch, TensorFlow...
		Languages	C/C++, Fortran, Python
		Parallel Programming	OpenMP, MPI, PGAS, OpenACC, OpenCL, CUDA, HIP
		Libraries	Math (BLAS, LAPACK...), I/O (HDF5, NetCDF, SionLib...), checkpointing (SCR, FTI)
	Tools	Compilers	icc, gcc, llvm,...
		Debuggers	TotalView, Allinea DDT, PGI, GNU GDB,...
		Profilers & Tracers	Scalasca, Vampir, VTune, Extrae, TAU
System Software	System Management	Resource Management / Job Scheduling	Slurm, Flux, Torque/Maui, IBM LSF, PBS pro, Containers (Kubernetes, Docker, Apptainer...)
		File System	Lustre, NFS, Storage Scale, BeeGFS
		Cluster Management	monitoring (llview, ClusterCockpit, DCDB...), energy (DVFS, power capping...), software installation (EasyBuild, Spack...)
	Operating System		Linux OS (RedHat, CentOS,...)
	Hardware		processors (CPU, GPU, accelerator), memory (DDR, HBM, NVM), storage (HDD, SDD, tape), network (InifiniBand, Ethernet, Slingshot), Infrastructure

FIGURE 4
High-level view of the software stack running on HPC systems. The product names given as example are merely illustrative, with no intention of giving a comprehensive list of all possible solutions.

acceleration devices, and even connecting disruptive modules such as quantum computers to *classic* HPC modules. For users, this modularity requires thinking about their application in a coarse-granular way.[1] If the code simulates multi-physics or multi-scale problems, which are often tackled by coupling different physical models, it is worth considering whether they could benefit from running on different hardware, i.e., across modules. If the code is monolithic (e.g., a dense matrix-matrix operation), it is better to run the whole application on a single module.

Observation 1: *HPC systems are becoming more heterogeneous, combining CPUs with GPUs and other accelerators. Monolithic system architectures combine these devices within massively heterogeneous nodes, but keeping all nodes equal, which requires a fine-granular partition of application codes. Modular architectures create partitions or modules each with a different node configuration, which requires a coarse-granular partition of application codes, suitable for, e.g., multiphysics or multi-scale applications.*

2.2 System software

A deep software stack is installed on HPC systems to operate them and provide all functionalities needed by the users (see Figure 4). Energy efficiency must be addressed at every layer of the stack, ensuring that all hardware resources are employed as efficiently as possible.

From the administrator perspective, several approaches can be used to optimise power consumption in HPC environments. Over-provisioned systems, which have peak power consumption exceeding infrastructure limits, can benefit from dynamic power capping. HPE PowerSched [20], for example, assigns dynamic power limits to individual jobs to maximise system utilisation. The framework EAR [21], among other things, adapts frequency to runtime application characteristics and a pre-defined energy policy. Idle power consumption is also a significant factor, especially in modern CPUs with varying idle power states [22–24].

Modern CPUs and GPUs offer software interfaces to control power consumption through Dynamic Voltage and Frequency Scaling (DVFS) [25]. While higher processor frequencies generally improve performance, they can negatively impact energy efficiency. Memory-bound workloads can operate more efficiently at lower frequencies, as their time-to-solution depends more on the speed at which data is retrieved from memory, than the speed at which the processor computes operations, so that their performance does not suffer as much when the processor clock is slowed down [26]. Compute-bound workloads, however, may benefit from higher frequencies, as shorter runtimes can compensate for increased power consumption. Some HPC sites offer *energy efficiency knobs*, which users can employ to define per-job system configurations with lower operational frequency [27]. Often, incentives are given in terms of higher queue priority or additional compute time to users that run their jobs at lower energy consumption [28, 29]. But even if users do not employ such knobs, both the hardware itself and the system operators might change the operation mode of individual compute nodes by applying power control techniques (e.g., DVFS or power capping) without prior notification, which make it challenging for users to reproduce the same performance numbers across equivalent job runs executed at different points in time.

HPC systems run comprehensive monitoring tools to optimise energy efficiency. This involves collecting metrics from various

1 If the code is partitioned between two modules, the *cut* should be done so that most communication happens within the modules, and not in between.

components, including compute nodes, infrastructure components, and batch jobs. Data collection is typically handled by node collectors that gather metrics and send them to a central communication component, such as a message broker. The monitoring system must integrate with the batch job scheduler to track resource usage and performance. At many sites, web interfaces provide access to monitoring data, with different levels of access for various user groups [30, 31]. In this way, users can read from their job reports how much energy they consumed, as well as the amount of resources that they employ. This helps identifying potential improvements for more efficient energy and hardware use. Goal in this optimisation is maximising the scientific throughput.

Most jobs use far less resources than available in the node [32, 33], and minimising idle resources is one of the main objectives of modern scheduling software. HPC sites are considering co-scheduling jobs within the same node, especially on HPC systems with *fat nodes*, i.e., those containing large numbers of cores and accelerators. Research trends in resource management go also towards dynamic scheduling mechanisms able to react at runtime to variations on both application requirements or overall system workload, or even further, to changes on the electricity network. HPC users should be therefore prepared for an HPC environment in which scheduling decisions might not only depend on their own choices at job submission, but also on energy optimisation strategies applied by the HPC centre. Adapting application codes so that they become *malleable*, i.e., able to grow or shrink its hardware needs at runtime, shall bring them the advantage of higher queue priorities [34]. Furthermore, compute time allocations might start soon to be measured in terms of energy consumed, and not merely core-hours, making it crucial for users to optimise their workloads for maximum efficiency.

Observation 2: *The rising energy demands and cost of energy leads vendors and HPC operators to apply numerous techniques to minimise the energy consumption and maximise scientific throughput of HPC systems. Some of them are exposed to the users, but not all. Users should be aware that the operational mode of an HPC system is not under their control and operational frequencies of processors are now more volatile. For benchmarking runs in which performance reproducibility is crucial, users are advised to consult job reports and contact the support teams from HPC sites to ensure their jobs always run under the same conditions. Furthermore, the allocation of compute resources per job might soon become much more dynamic and be measured in terms of energy consumed than of core-hours. Users are therefore advised to use monitoring data to determine the energy consumption of their jobs and identify the most energy-efficient run configurations before starting production jobs.*

2.3 Programming models

The programming environment in HPC abstracts the hardware complexity from the user through general Application Programming Interfaces (APIs), helping existing code to run on rapidly evolving compute hardware. The following subsections describe state of the art, challenges and recommendations at each of the layers in the programming environment.

2.3.1 Programming languages

Many, if not a majority of codes running on HPC systems, have been developed over many years with their basic architectures designed decades ago relying on relatively low-level approaches (mostly Fortran or C) conceived to run on CPUs (e.g., [35–39]). The reasons for these codes to still exist, despite newly available hardware and software solutions, is evident: their goal is to produce data of scientific interest, in a very short time, for a limited number of users. This leads in general to codes that are very difficult to use, not well documented, and not widely distributed. Even codes used to publish results in high impact journals are still considered complex to use and unavailable to the public [40]. In particular, users at the end of the analysis chain employ tools as black boxes to extract scientific insight of their interest. These tools are developed within smaller groups of developers who are often inexperienced with the programming concepts, data structures and performance tuning. For such legacy codes, it cannot be assumed that more performance can be gained automatically on modern heterogeneous HPC systems without significant intervention of researchers.

With the growth in popularity of data analytics and Machine Learning (ML), there has been a slow shift in the use of programming languages, moving from the traditional Fortran and C/C++, to high-productivity alternatives like Matlab, Python or Julia [41], which offer a simple syntax, a large number of easily accessible libraries, and fast application prototyping. This has lowered the entrance barrier to new computer programmers, at the expense of computing efficiency.

Still, codes optimised for HPC, taking advantage of the latest hardware technology, need to be written in traditional languages that are closer to the hardware itself. Unfortunately, new generations of scientists rarely learn these (see also Section 2.5.1). Students in many scientific domains lack formal training in software engineering in general, and in HPC in particular. Also education in advanced C++ and modern Fortran is extremely rare. Without such prior knowledge, potential new contributors to state-of-the-art software frameworks face a very steep learning curve, often incompatible with their limited time budgets as they begin to establish themselves as domain experts in their academic career paths.

2.3.2 Hardware heterogeneity and portability

The increasing heterogeneity of HPC systems described in Section 2.1 and the consequent system complexity make software development more challenging. Transitioning application workflows to energy-efficient platforms, such as GPUs or FPGAs, involves more than rewriting codes. It requires fundamental redesigns of algorithms to accommodate the distinct memory hierarchies and parallel processing capabilities of these architectures. Each kind of compute device requires different optimisation approaches, e.g., while spatial and temporal locality play a role in the optimisation of access patterns and data layouts on both CPUs and GPUs, good performance on GPUs can only be reached if threads within a thread team access data in a coalesced fashion and if thread divergence is avoided. Although these efforts

promise substantial energy savings, the initial development is effort-intensive, demanding both time and specialised expertise.

New hardware also comes with new programming models and libraries that need to be adopted into the software ecosystem. Compilers need to include in their standards additional ways to translate codes into machine binaries and for a new hardware component it can take years before full adoption by the end users. Vendor-specific programming models promote vendor lock-in (e.g., to get optimal performance, NVIDIA GPUs had to be programmed using CUDA [42], and AMD GPUs with HIP [43]), which can backfire on application developers as it happened with the Intel Xeon Phi many-core processor launched in 2010: while many research groups saw this as an opportunity to accelerate their codes without having to deal with complex parallelisation using CUDA on GPUs, the processor ended up being discontinued in 2020 [44] and developments that took years to implement became suddenly dependent on an obsolete technology. Scientists learned that it is important to write codes so that they are portable to different architectures.

Observation 3: *Modern compilers are a great source of information to identify low hanging fruit opportunities for improvements in performance. They often provide flags to detect misplaced data accesses, to unroll nested loops, or to replace known inefficient instructions. However, it is our experience that recent compilers remain very unstable, and it is preferable to use reliable older versions of the compilers and miss out on the opportunity to test the newest features in the most recent version. It takes from 2 to 5 years for known compiler versions to be fully tested by developers worldwide.*

As hardware becomes more heterogeneous, performance-portability across platforms is harder to attain. Using libraries, especially I/O and numerical linear algebra (e.g., the BLAS APIs), does help because processor vendors provide their own hardware-optimised implementations, reaching highest efficiency. The performance, energy efficiency, and the level of code portability that HPC users can achieve depend on the APIs and programming models they choose. Generally, low-level programming models give more control to the users, at the price of lower portability. High-level abstractions (e.g., Python [45], OpenMP [46], SYCL [47], Kokkos [48], ALPAKA [49], RAJA [50], Mamba [51], Julia [41], DaCe [52], and many others not named here), on the other hand, are portable and achieve good performances [41, 53–55] by relying on backend optimisations to generate executable codes adapted to different hardware devices. However, for these languages to deliver performance in an energy-efficient manner, it is important to call their optimised packages, such as numerical libraries and backends, typically written in C. Otherwise applications written in Python can be very power-hungry [56].

Observation 4: *To cope with increasing hardware heterogeneity, application development teams are advised to modularise their codes, decoupling the optimisations for different hardware platforms from the scientific core of the application, e.g., by calling appropriate numerical libraries. Failure to do this can lead to much lower energy and resource usage efficiency. Teams lacking specialised personnel with the necessary expertise and effort to optimise code for different hardware devices are advised to employ portable programming models and frameworks that isolate their scientific implementation from the hardware-specific features, which shall be addressed by optimised libraries and backends.*

2.3.3 Domain specific languages

Domain Specific Languages (DSLs) separate user-facing interfaces and performance-critical code altogether. The prime example are ML frameworks such as PyTorch and TensorFlow. Examples in high-energy physics are, in increasing order of abstraction, Grid [57, 58], Chroma [59] and ROOT [60] as well as Lyncs [61], PyQUDA [62], and Grid Python Toolkit (GPT) [63], where the latter three provide Python interfaces to the highly optimised algorithms implemented in QUDA [64–66] and Grid, respectively. In the domain of space physics, problems that can be approximated by the large-scale approach, where a plasma can be considered as a fluid, can use one of the existing frameworks (e.g., [67–69]) that decouple numerics, parallelisation, and mathematical equations from each other. This approach allows scientists to focus on the equations to be solved, leaving the software heavy-lifting to the underlying code programmed by specialists.

It is to be seen in what way tools from generative AI will impact code generation for HPC systems. While using Large Language Models (LLMs)/chatbots or GitHub Copilot to generate code for routine tasks is by now common, using generative AI also shows promise for HPC code generation [70, 71].

2.3.4 Parallelisation over multiple compute nodes

Any code or application developer that wants to take advantage of modern HPC systems shall start by making sure that: (i) their code runs as optimal as possible in a single node, (ii) their code is well structured and documented, and (iii) it presents an easy to use interface that does not require to recompile the source code for different use cases. Among the applications using the biggest supercomputers in the world it is not uncommon to miss points (ii) and (iii).

While the high-level programming models described in Section 2.3.2 allow for portability between individual nodes with different hardware configurations, parallel applications need to run across many nodes. The *de facto* standard multi-node parallelism framework in HPC is MPI [72]. It requires explicit transfer of messages between nodes, but is in principle agnostic to the type of node underneath, as long as an MPI library is installed and supported. While this is true in most cases, various implementations of the MPI standard exist, and performance differences are often observed between implementations running on the same hardware. Therefore, users are advised to test their applications against different MPI implementations, verify correctness, and identify the most efficient one in each system.

An alternative to message-passing is given by the Partitioned Global Address Space (PGAS) paradigm in which a global memory space is logically partitioned over many processes and where the various portions of this shared space have an affinity for particular processes. In contrast to message-passing, individual processes may access remote memory directly with the necessary communication taking place either explicitly or implicitly as part of the semantics. Examples of such PGAS systems are Coarray Fortran [73], Unified Parallel C [74] or Global Arrays [75].

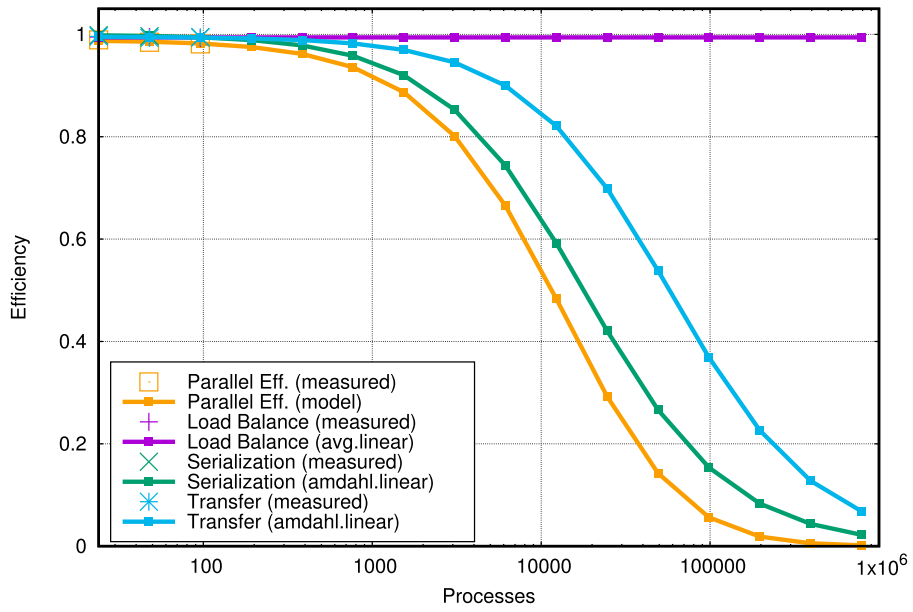

FIGURE 5
Measured and extrapolated parallel efficiency of a kinetic space plasma code using DIMEMAS [85], reproduced from Figure 26.a (page 63) in [87]. The parallel efficiency, representing the time spent on computation (useful work), is composed of the load balance, the serialisation, and the transfer efficiencies, as defined in Section 4.1 of [87] and the DIMEMAS [85] documentation. In this example three measuring points at 1, 24, and 96 processes, were used to extrapolate the parallel efficiency of the code up to one million processes.

For problems subject to load-imbalance, a proliferation of small subtasks, formulations on irregular grids or implementations, which aim to make use of highly heterogeneous computational resources, Asynchronous Many-Task (AMT) systems may provide a more appropriate programming model with the potential of increasing scalability and resource utilisation. Examples of such AMT systems are Uintah [76], Charm++ [77], ParSEC [78], Legion [79] and HPX [80, 81]. A more complete comparison can be found in reference [82], but in general it can be said that these solutions are either compilers and run-time systems, separate libraries and run-time systems, or systems in which the run-time is linked directly into the application. Generally, AMT systems provide some form of dependency resolution mechanism and a data-flow model based on a Directed Acyclic Graph (DAG) together with some form of Asynchronous PGAS (APGAS) model for representing distributed data. It is also possible to benefit from the performance-portability of Kokkos and the task-based parallelism of an AMT system like HPX to move away from the more traditional fork-join model with good results [83, 84] by making use of *futures* to launch and synchronise Kokkos kernels.

We have observed that computer applications developed to run in parallel architectures with a small number (< 1000) of cores sometimes struggle with the porting to larger systems, and in particular to Exascale supercomputers. In Figure 5 we present the measured and extrapolated efficiency of a parallel kinetic plasma physics code. Based on three measurement points of the code efficiency, the tool DIMEMAS [85] can predict the performance of the code on very large systems using Amdahl's law [86]. Such analysis is an essential step to discover what are the scalability limitations of any software. As the number of processes increase serial sections of the code and parallel transfers become dominant.

Observation 5: *The secret to a good parallel application ready for the next generation of supercomputers rely on an early and thorough analysis of single node peak performance, and parallel strategies that reduce data transfers to the minimum. Once this is done, the developer should test as early as possible the scalability of their parallel strategy using performance extrapolation tools. A parallel algorithm is difficult to change when a code is mature with multiple years of development. To develop and test a scalable parallel algorithm it is possible today to use tools like DIMEMAS [85] that project the potential scalability of a code on any existing or idealised supercomputer using performance traces from experiments run in a few hundred cores.*

2.4 Data management

When discussing energy efficient trends in HPC, it is important to include data management and storage. While storage systems may not directly consume as much electricity as the processors during peak loads, they can significantly impact overall efficiency. Moreover, storage systems can become persistent power consumers, with data often requiring preservation for decades, leading to high integrated energy costs over time. Fields such as HEP and radio astronomy exemplify these challenges. Constrained by budget and technological limitations, they cannot store all the raw data generated. Instead, they rely on real-time processing and filtering to reduce storage needs and concentrate on valuable insights. However, this approach introduces its own energy-intensive demands. Advancements in ML algorithms offer the

potential to enhance data selection accuracy and speed, but their training and optimisation processes come with additional storage and computational requirements, further complicating the energy equation.

High performance data management is an increasingly important consideration as the demand for HPC grows across scientific and industrial domains. One major source of energy inefficiency is the movement of data within computing systems. This issue does not stem primarily from the storage or file systems themselves, which are relatively energy-efficient, but from the impact on processing units. HPC processors, designed for high throughput, are power-hungry even when idle, consuming almost as much energy as during full operation. These processors often remain underutilised, idling as they wait for data transfers or caching to complete. This mismatch between data availability and processing demands highlights the need for improved data transfer and caching infrastructure. By minimising idle time and ensuring that processors operate at high utilisation rates, significant energy savings can be achieved. Users are therefore advised to consider overlapping communication with computation, to keep the processors busy while they wait for new data. It is also important that they employ optimised I/O libraries (e.g., HDF5 [88], SIONlib [89]), which ensure that data is aggregated in a reduced amount of files to avoid saturating the file system. When data transfers between nodes are impossible to avoid, creating a regular pattern for the data transfer is a key factor. Reducing the number of communication steps, and taking advantage of algorithm optimisations by the parallel libraries can increase the scalability of a code.

Observation 6: *Data movement is an important contributor to energy consumption, due to idling processors while they wait for data to be loaded from either disk, memory, or cache. Application developers are advised to employ optimised I/O libraries and overlap computation with communication when possible, to minimise waiting times and idling resources.*

Storing data, whether for active use or long-term archival, also contributes substantially to energy consumption. Active storage media must consume power continuously to keep data accessible and to serve it efficiently across distributed systems. This often necessitates maintaining multiple data copies to ensure reliability and availability. Even archival storage, which might seem like a low-energy alternative, requires sophisticated infrastructure to prevent data loss or corruption, such as climate-controlled facilities. Promising advancements in archival technologies aim to create dense, stable media that can preserve data for centuries under normal conditions without requiring power [90]. These innovations could drastically reduce the integrated energy footprint of long-term data storage.

Another critical challenge lies in managing the massive volumes of data generated by instruments and experiments in fields such as physics and astronomy. Projects like the Large Hadron Collider (LHC) and the Square Kilometer Array (SKA) produce data at such scale that it becomes infeasible to store and load all of it. Instead, much of this data must be processed *on the fly*. However, streaming data directly into HPC systems is not a traditional use case and poses significant logistical and technical challenges. HPC workflows typically rely on pre-scheduled tasks, but the irregular operation of data-generating instruments—affected by maintenance schedules, weather, and other factors—complicates integration. The synchronisation of real-time and scheduled workflows demands innovative solutions, including new scheduling mechanisms and potentially dedicated HPC systems optimised for streaming workflows. Nevertheless, gaps in instrument operation could present opportunities for resource sharing, further enhancing efficiency.

The demand for energy-efficient solutions grows even greater in all scientific areas, as they increasingly rely on AI and ML techniques.[2] AI and ML require large datasets for training models, which in turn demand significant computational power. This adds another layer of complexity to the energy equation, underscoring the importance of developing efficient algorithms and hardware optimised for AI/ML workloads. Innovations in this area can reduce the energy costs of model training and inference, making AI-driven data processing more efficient.

Efficiency also means two things: ensuring that scientific results are reliable, and avoiding unnecessary repetition of the same work over and over again. This is why open data and the Findable, Accessible, Interoperable, Reusable (FAIR) principles [91] start to play an increasing role. Open data is the essential first step to make simulations reproducible and therefore trustworthy, by providing access to usable software and to raw data from the numerical experiments.

Observation 7: *Every user should strive to make their simulation results FAIR [90], which includes making them reusable. This way, other researchers can avoid running the same simulation again, use the results as input for their own work, and re-analyse the data from a different research perspective. For simulations producing only small statistical samples, reusable data can be accumulated to increase the sample size by pooling the results from different researchers.*

The challenges of energy-efficient data management demand a holistic approach that integrates improvements across data movement, storage, and processing. Advancements in transfer and caching infrastructure, archival solutions, and HPC systems designed for real-time workflows are critical.

2.5 Applications and benchmarks

Computational sciences create digital representations of scientific problems and in doing so are constantly confronted by a limitation on the available human and computational resources: (i) the available budget for a project, the number of people assigned to the development of the code, the availability of the relevant experts, and (ii) the available memory, the available disk space, and the available compute time. These constraints are different for different research institutions, and while it is still possible (and elegant) to produce groundbreaking research results on a single desktop computer, exploiting the biggest supercomputers can lead to high impact results [92–97]. This is the motivation for most computational scientists to approach HPC.

2 For instance, when analysing the user statistics of the JUWELS Booster (right side of Figure 1), we found out that 47% of the total compute time is used by AI-related jobs.

We give in this section the general recommendations for any developer or user of applications in HPC. The following three sections go into more detail on how scientists specifically in the areas of astrophysics (Section 3), HEP (Section 4), and Lattice Field Theory (LFT) (Section 5) can address energy efficiency and performance.

2.5.1 Skills

Up to about the year 2000, in most scientific fields primarily the same researcher developed the code, ran the simulations, and analysed and visualised the results. These researchers knew every line of their codes and understood the numerical challenges. The severe limitations in computing time forced the researchers of this era to spend much effort optimising every aspect of the code for efficiency.

Nowadays, scientists starting to use HPC typically take their first steps with legacy code inherited from their predecessors, or with large analysis frameworks used as a black box. Their tasks often incorporate writing small extensions to study a new aspect of a scientific problem, but there is rarely an immediate motivation or an offer for the necessary education to get accustomed to the existing code base. Both time and software engineering training are lacking to perform tasks such as porting code to new environments, refactoring, or performance engineering. Furthermore, they implicitly assume that the used code has already been optimised for efficiency, but this assumption is often wrong because the code optimisation, if done at all, happened on an entirely different computer infrastructure. As a result, the code produced may not fit today's compute hardware, or may perform significantly worse than it could, due to wrong data layouts, inadequate programming models, or outdated libraries and environments. The long term success of a scientific code rely purely on the motivation, training, and acquired expertise of a few experts associated to the institution developing the software, but too few permanent software developer positions are available.

Observation 8: *Application developers benefit from working hand in hand with experts in HPC software development, usually employed at HPC centres. There are multiple means for developers to have access to this expertise (e.g., training courses and direct mail contact with support teams), learning in a very short time how to extract the maximum potential of their codes. However, the best approach for continuous code improvement and adaptation to emerging technology is to include in the development team Research Software Engineer (RSE) experts with knowledge on the physical domain of interest and on the HPC ecosystem. It is expected that the research teams using the latest HPC systems to their full potential have in their ranks expert RSEs, who are commonly postdoctoral researchers with additional specialised training in HPC and advanced programming.*

In particular, there is a growing risk of losing, over time, the expertise required by research institutions to produce high performance codes, as new generations of researchers are not trained in traditional scientific computing languages (e.g., C/C++ or Fortran). Today, the few research engineers trained with detailed knowledge of traditional programming languages, have a clear advantage in the job market not only in the scientific domain but also in any industry requiring coding (see, for example, the banking and the video game industries).

Application developers can find it difficult to keep up with the very rapid evolution of hardware, and only well-funded research groups with the necessary HPC expertise and specialised support can take advantage of emerging new technologies. Given the growing use of HPC in science and industry, building the knowledge of the next generations of computational and computer scientists through dedicated courses at universities and training programs becomes crucial. These courses should cover all aspects of HPC, including low-level programming languages, performance analysis and engineering, parallelisation and scaling, code optimisation and tuning, and data management.

Observation 9: *Students and early career researchers in computational sciences should seek and receive training on numerical algorithms, data access patterns, and the frameworks used in their community, as well as on general HPC topics such as performance analysis, optimisation, and software engineering. Specific education on energy-efficient simulation techniques is required, including concepts such as the energy footprint of different hardware devices, software tools, and programming models. Similarly, beyond familiarity with a high-level language such as Python or R for data analysis, students should know that some Python packages are more energy efficient than others, and establish a good foundation in modern C++, which is a prerequisite to contribute to the continued development of efficient application software.*

2.5.2 Performance analysis

Generally speaking making an application code more energy efficient is equivalent to ensuring that it reaches the maximum performance on a given compute device, which itself is running at the most energy efficient operational configuration. Or said otherwise, the application should not waste the capabilities of the hardware on which it runs. In practice, it means: (i) understanding which is the maximum theoretical performance that the target computer is capable of, (ii) measuring the performance that the own code currently attains on this computer, and (iii) knowing how to close the gap between (i) and (ii). This can be studied in first approximation with tools like the *roofline* model [98], which relates the performance (in FLOP/s) with the arithmetic intensity (in FLOP/Byte, aka operational intensity) of an application with the peak performance of a given computing device, classifying applications into memory and compute bound. Numerous profilers and performance analysis tools integrate the roofline model and provide further bottleneck analysis mechanisms for parallel applications (e.g., Vampir [99], Paraver [100], Scalasca [101], Vtune [102]).

Using these tools, developers have to invest on the analysis of the performances of their code prior to seeking the best technological solutions to remove bottlenecks. Common blocking points include inadequate parallel communications, uneven data transfers, blocking I/O tasks, slow data ingestion, irregular memory access, idling computing resources, and unrealised compiler optimisations. Addressing all these requires access to tutorials, literature, and tools that simplify the analysis of computer software ported to HPC systems is needed [103].

Observation 10: *The first step to improve the energy efficiency of an application is reducing its time-to-solution by investing effort on performance engineering. Performance analysis tools and profilers help identifying communication and computation bottlenecks and sources*

of performance loss. Small changes in bottlenecks can rapidly provide great gains in performance. Typical code optimisations include early detection of memory leaks, good alignment of data in memory to take advantage of cache access, use of existing compiler optimisations taking advantage of hardware characteristics, and smart use of multi-threading in all cores in a node.

2.5.3 Artificial intelligence and low precision arithmetics

GPUs have been successfully adopted in HPC, but even more so in the AI market. Model training requires very large amounts of computation, which can be done very efficiently with GPUs. Unlike most HPC applications, AI training makes heavy use of reduced precision arithmetic (e.g., [104]). The large size of the AI market motivates GPU vendors to devote more silicon area to these arithmetic units at the expense of double precision support. Applications using low-precision arithmetic can achieve higher performance with less power consumption. HPC application developers are therefore strongly encouraged to consider whether some of the operations in their codes could be performed in single (or lower) precision. However, it remains to be seen how well the diverse portfolio of HPC applications can be successfully ported, taking into account the resulting losses in numerical stability and reproducibility. Intense algorithmic research is being done to enable more HPC applications the transition from double precision, or 64-bit floating point performance (FP64) to reduced-precision arithmetic (e.g., FP32, BF16). If the AI market continues to grow at its current rate, we can expect to see new accelerators entering the market, particularly for inference operations (e.g., TPU [105], Graphcore [106], Groq [107], etc.), which are likely to support mostly low precision arithmetic, but potentially also new programming models. This again, might require additional adaptations of HPC application codes.

Observation 11: *Opportunities to work with reduced arithmetic precision should be exploited, as this translates into a higher performance and energy efficiency of the application when running on the same hardware.*

2.5.4 Benchmarks

Application-based benchmarks are included in the procurement and acceptance of HPC systems to represent the main consumers of computing time [108]. This allows for choosing a system design and a selection of hardware and software components appropriate for the site's application portfolio, which is critical to maximising energy efficiency in real-world operations. The main difficulty is to accurately predict how the application portfolio at the time of acquisition will evolve over the operational lifetime of the supercomputer. Application developers can contribute to this effort by creating *mini-apps* and application use cases representative of their operational workloads, making their source code and input data publicly available, and keeping them up-to-date with the latest code releases. Benchmarking campaigns are also important for the application developers themselves, as they serve to analyse how changes in a given code (or environment) affect performance. Equally important is applying professional software development strategies, with curated code repositories, version control, Continuous Integration and Continuous Deployment (CI/CD) pipelines, automated testing, code documentation, and regular software releases.

Observation 12: *Benchmarking studies should be performed before running production jobs to identify the most energy-efficient job and system configuration for the given application, looking for the lowest possible operational frequency of the hardware without strongly increasing the time-to-solution (e.g., in memory bound applications).*

3 Computational astrophysics

Studying the movement of galaxies at the cosmological scale or the motion of a single electron around a magnetic field line, the domain of astrophysics covers a very wide range of scales in time and space. While the equations that govern all the dynamics of the universe are well known, it is still impossible to reproduce in a computer all the different processes at once. Scientists need to make assumptions, neglect terms in equations, reduce the complexity of the environment under study, eliminate non-linearities, and translate mathematical equations into numerical algorithms solved on discrete machines working with ones and zeros. Each one of these simplifications introduce an uncertainty to the final result. The role of the application developers in computational astrophysics is to minimise such uncertainties and to describe as accurately as possible the real universe into a matrix of discrete values.

Many computer models in the domain of astrophysics have emerged from the work done by scientists over multiple years, even decades. In a traditional academic life-cycle new numerical techniques are typically introduced by senior researchers, which are then extended and refined by their research teams (postdocs, PhDs, master students). Very rarely an academic software is developed by a professional software engineer. Over the past two decades the hardware used to execute such software has become increasingly more complex (see Section 2). Memory strategies, data traffic, data structures, I/O, accelerators, scalability, interconnections, are some of the terms that have come to the attention of research software developers. A new type of expert is emerging from this environment: the RSE, who is both an expert in physics and computer sciences.

Making simulation results reusable for others is an essential contribution to making astrophysics simulations more energy efficient. In observational astronomy, a large proportion of the data are published. There, the Flexible Image Transport System (FITS) format [109] is a widely accepted standard for data publication. It is used for the transport, analysis, and archival storage of scientific data sets. By contrast, open access is much less common in astrophysics simulations, especially if it does not only concern sharing code but also the resulting data. Here, general agreements on used formats and metadata standards are still lacking. A few large collaborations set good examples of sharing the results data, i.e., the Illustris project [110], the TNG project [111], the Eagle project [112], and the Horizon simulations [113]. In the context of astronomy and astrophysics, reproduction usually requires re-evaluation of large datasets and the discussion about FAIR research data management has just started for astrophysics simulations.

It is our goal in this section to present the trends in energy efficient HPC to research leaders and research engineers working on astrophysics codes.

3.1 Stellar and planetary physics

Codes in stellar and planetary physics were initially developed to describe specific physical process, for which a distinct numerical method was used. For example, the dynamics of particles were calculated using N-body methods and hydrodynamic flows employed numerical fluid dynamics methods. Over the decades, these codes have developed and nowadays they usually include multiple physical processes simultaneously. Often, they also allow one to choose which numerical scheme to employ. For example, the newest version of the GADGET code (GADGET 4) [114] also contains a simple model for radiative cooling and star formation, a high dynamic range power spectrum estimator, and an initial conditions generator based on second-order Lagrangian perturbation theory. Besides, it gives several choices of gravity solvers and smooth particle hydrodynamic options.

Other approaches are frameworks that allow combining different codes to provide for various physical processes. One example is the Astronomical Multipurpose Software Environment (AMUSE) [115], which allows to combine multiple solvers to build new applications, used to study gradually more complex situations. This procedure enables the growth of multi-physics and multi-scale application software hierarchically. Generally, the complexity of the software continues to increase.

Most simulation runs are a single realisation of an ensemble of possible states of a system. Thus, it is necessary to perform an entire suite of simulations to increase the statistical significance of the inferred result. However, multi-physics codes, especially, often require such extensive computational resources that only a single simulation run can be performed. The statistical significance of a single realisation of a statistical approach is rather limited. Examples of such simulations that went to the limits at the then available computational resources are the Millenium simulation [116] or the simulation of clustered star formation [117]. In principle, this problem should have solved itself with computational performance increasing by a factor of several thousands to several ten thousands during the last 20 years. However, this is often not the case. Rather than increasing statistical significance, researchers often prefer to use the increasing resources to boost resolution or include additional physics.

Due to the modular nature of many physics aspects included in a single code, testing the code to its entire extent becomes increasingly challenging. The researcher can evaluate only individual components against a set code test. However, few to no tests exist of how the different parts work together.

Making use of GPU-based HPC systems can increase energy efficiency in astrophysics considerably. For example, the GPU-based version of Nbody6 is approximately 10× faster than its CPU-based version [118]. The degree of transition to GPU-ready codes is mixed in the star and planet formation field. Popular codes for simulating like FLASH [119], GADGET4 [114] or REBOUND [120] were not intended for GPU-based machines. The user community has become aware, and parts of the codes are gradually changing the algorithms to use GPU-based structures. For N-body codes, the situation is better. Codes like Nbody6GPU++ already saw the advantages of using GPUs about 15 years ago. Other GPU-ready N-body codes are, for example, BONSAI [121], GENGA [122] and PETAR [123]. In summary, the star and planet formation community is currently in the transition phase to GPU-ready codes.

Making open access of results and data, and ideally fulfilling the FAIR principles [91] would make simulations also more energy efficient. However, the community is only starting to evaluate how to adopt FAIR principles when dealing with simulation results and data management. Codes are increasingly shared on GitHub or the Astrophysics Source Code Library [124]. However, the shared code versions often differ from those used to obtain the results reported in publications, because the codes are proprietary or the team simply lacks the resources to properly document new code parts. Benchmark tests are performed for the publicly available code version but are less often performed for the codes with non-public add-ons. Thus, the user is sometimes unaware that these add-ons are less efficient than the main code.

In the past, many codes with similar applications existed in parallel, and there was a constant stream of newly developed codes. While new codes are still being created from scratch, this happens to a lesser degree due to the growing code complexity. Nowadays, much of the development effort goes into adapting legacy codes written in Fortran, C or C++ to new computing structures and extending them by adding new modules often developed in Python. Frequently, simulation codes are written by students who are unaware of the energy efficiency issue or lack the knowledge of more energy-efficient programming languages.

Especially for simulations reaching the limits of computing, replacing part of the code with AI-generated information is implemented in some codes. So far, this transformation code approach has only been adopted by a relatively small part of the community. While promising to make the codes more energy-efficient, one loses some of the causal information. Furthermore, the compute and energy cost of training the AI-models should not be neglected.

Observation 13: *When an application framework allows switching on and off different physical processes, it should be ensured that only those necessary to answer the investigated scientific question are activated. Additionally, during the planning of the simulation approach, one should put emphasis on allocating the minimum necessary amount of computational resources (nodes), while maximising the use of those that are reserved. Users should check after completion whether the code performance stayed high throughout the runs, and learn how to minimise idling resources.*

3.2 Space weather and space physics

Scientific codes in the area of space physics show very similar characteristics as those described in the previous section. However, the domain of space weather is one step closer to real-time operations and require high availability of computer resources, and high reliability on the outcomes of the computations. To forecast the impact of the solar activity on the Earth, several advanced computer models benefit from HPC systems. Multiple

FIGURE 6
Number of publications in scientific journals over the past 20 years, containing the keyword *space weather*, which include in the text references to *supercomputer* or *high performance computing*. Data gathered using the OpenAlex database [130]. The median number of these publications over the past 20 years is seven.

models exist still in their research phase while a few have already transitioned to an operational phase and are used to inform the public and industry. The process of transforming academic codes into consolidated operational software is called Research-to-Operations (R2O). Historically, the domain of space weather has been a continuous user of HPC resources. Figure 6 shows that over the past 20 years the number of publications in the domain of space weather that makes explicit use of HPC systems has been constant. In general the domain of space physics (including solar physics, space weather, heliospheric physics, planetology, and plasma physics) is known to exploit the latest computing technology, including new generations of processors, accelerators, memory, storage, and network interconnection [125–129].

The physics of the space environment cover a large range of scales in time and space. It is common practice to study the phenomena observed at each characteristic scale by independent models. This translates in general into computer models that calculate numerical approximations, covering different segments of the full space environment. To capture as much detail as possible, these numerical models can take advantage of very large computer resources to extend their applicability to a large range of characteristic scales. Figure 7 shows a very small selection of individual computer models studying different segments of the space weather domain that connects the solar activity with its effects on Earth. While most of the physics in these codes are very similar, the numerical methods and characteristic scales processed by each code are different.

In the domain of space physics and space weather the three most common approximations to model the corresponding physics are: (i) the particle kinetic scale [131–134], (ii) the fluid continuous scale [68, 135, 136], and (iii) the hybrid scale, where particles and fluids coexist [137–140]. The equations solved by these numerical approximations can be very diverse, and different computer hardware might be better adapted to each of them. It is reasonable to solve the dynamics of billions of particles in a group of accelerators, while solving the memory intensive resolution of a continuum using a network of general purpose CPUs [141]. To make a selection, a careful knowledge of the algorithms, the hardware, and how to connect the two of them is necessary.

HPC is used in space applications to produce data. With high value data, scientific discoveries can be published, and mitigating actions can take place to reduce the impacts of the space environment on our technology. As the HPC systems become larger, models grow to use them, generating much more data per run. This makes it more challenging to process and translate the outputs of these models into information that can be used in the daily operations of the end-users of space weather forecasts. High precision numerical models producing data at a high cadence can lead to gigabyte or even terabyte of data in a single execution. Therefore, there is a growing need to process and visualise data in the same location where it is produced, *on the fly*, at the same time as the code is executed. In modern HPC systems GPU and visualisation nodes are also available to the end user. This allows to perform data gathering, analysis, processing, and visualisation in the same system where the execution of the code took place. This also requires the end user to provide all the data pipelines necessary to extract valuable data from the raw outputs, which will then be discarded by the end of the execution.

We encourage application developers to also evaluate the potential use of high-order numerical methods. It is a common misconception to think that high-order numerical methods are more resource intensive. It is clear that if a problem uses the same time and space discretisation, high-order methods will take much longer to converge. However, this is an inadequate way to measure performance [142]. To obtain the same error low-order methods require much finer discretisation and longer convergence times. The use of high-order methods is then critical for applications that require fine precision in their resolution, as, for example, in the case of sharp shocks, energy dissipation, or multi-scale effects. In an optimal code, the order of the methods shall be adaptive to optimise the use of resources.

Application developers should also consider implementing high-order methods in cases where performance analysis show that the software is memory bound. This approach may help the software

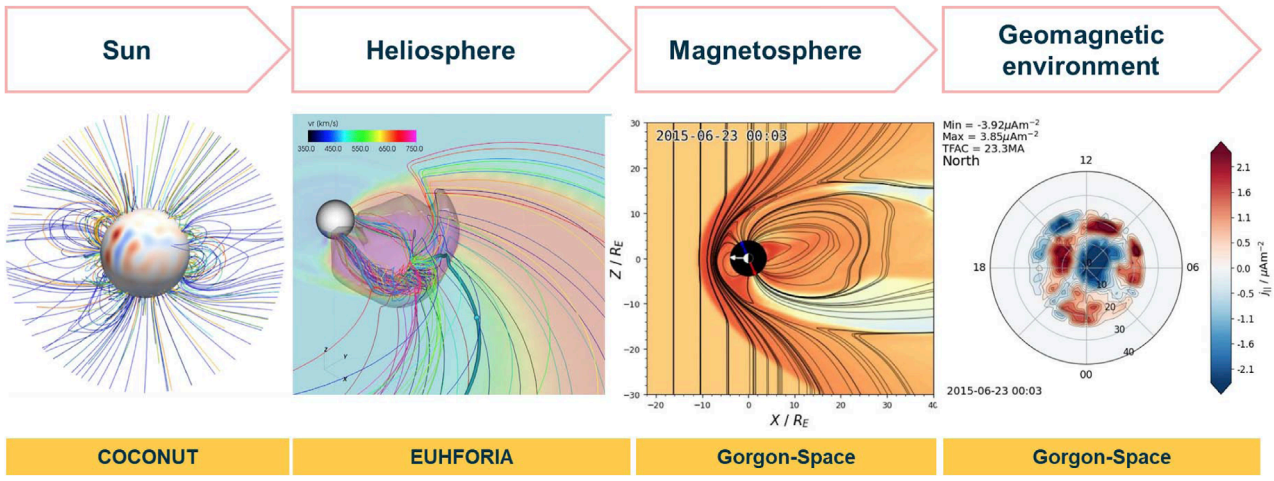

FIGURE 7
Three computer models used in the space weather domain to connect solar energetic events on the Sun with its impacts on Earth. COCONUT: modelling the state of the lower solar corona, image reproduced from [211]. EUHFORIA: transports solar plasma towards the planets of the Solar System, image courtesy of KU Leuven. Gorgon-Space:calculates the geomagnetic environment around our planet, images courtesy of Imperial College London..

to switch the balance towards higher computing intensity. This is of course dependent on the specific numerical method used, as memory stencils might also increase to accommodate the larger data size required by these methods. The application developer should keep in mind how the different cache and memory levels are used in the computationally intensive segments of their software [143–145]. Reducing the data movement among the different memory levels, and performing vector operations, can further help transforming memory bound into compute bound codes.

4 Data processing in experimental high-energy physics

Energy efficiency has become a pressing challenge in HEP computing as the field grapples with the ever-growing demand to manage and process massive datasets. With experiments at the LHC already having generated over an exabyte of data, and the forthcoming High Luminosity LHC (HL-LHC) expected to add another exabyte per experiment annually, the infrastructure is facing unprecedented challenges. This rapid expansion underscores the need for innovative approaches to improve energy efficiency across data processing, storage, and distribution.

HEP computing relies on highly complex workflows, particularly for reconstructing and simulating particle collisions. Many of these workflows have been developed over decades, with vast, intricate code-bases optimised for traditional computing architectures. However, this reliance on legacy poses significant challenges for modernisation.

Distributed computing, exemplified by the Worldwide LHC Computing Grid (WLCG), has been instrumental in enabling HEP's global research efforts. This extensive network connects hundreds of computing centres worldwide and has successfully supported the immense computational needs of the field for years. However, adapting such a vast and distributed infrastructure to modern, energy-efficient architectures presents another layer of complexity. The extensive volume of legacy code, coupled with the need for algorithmic and structural redesigns, complicates the process of integrating new hardware technologies.

Recognising the urgency of these challenges, the HEP community has launched robust research and development initiatives to enhance energy efficiency and modernise its computational ecosystem. These efforts include optimising workflows, reducing unnecessary data movement, and developing software tailored to more efficient hardware platforms. CERN Openlab [146], a collaborative partnership between HEP scientists and industry leaders, has helped drive this progress. Over two decades, it has guided the transition to modern computing paradigms, from x86 commodity clusters to accelerated hardware architectures like GPUs. More recently, it has facilitated explorations of tools such as SYCL [47], Intel OneAPI [147], and other portability libraries, which facilitate the migration of legacy applications to heterogeneous systems. These innovations have not only improved efficiency but have also increased the adaptability and maintainability of HEP software, ensuring its compatibility with future hardware advancements.

Experimental HEP computing is increasingly challenged by the need to manage and process vast and ever-growing datasets while improving energy efficiency. As the field pushes the limits of data-intensive science with experiments at the LHC and prepares for the High Luminosity upgrade, innovative solutions are needed to address the very large energy demands.

Observation 14: *The field of HEP faces a significant challenge in achieving energy efficiency due to its reliance on legacy code and traditional architectures. The transition to modern, energy-efficient platforms such as GPUs and FPGAs demands not only rewriting code but also fundamentally redesigning algorithms to align with the parallel processing and memory hierarchies of these platforms. Adopting*

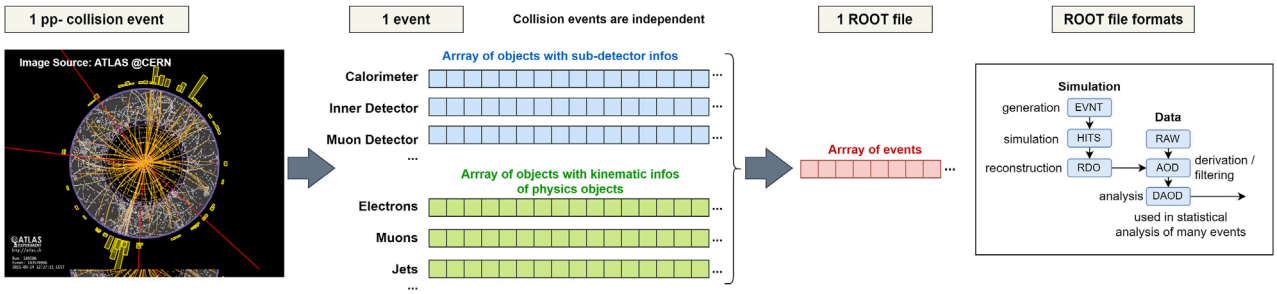

FIGURE 8
Stages in the analysis in High Energy Physics for the example of the ATLAS experiment. Both simulation and real data processing produce different levels of data files, which are stored in the common ROOT [60] file format. The DAOD files containing structured object data are commonly used as input for end user analysis [151].

programming models such as SYCL [47], OneAPI [147], and other portability libraries increases the adaptability and maintainability of HEP software, facilitating compatibility with novel energy efficient hardware technologies and meeting future computational demands.

4.1 Data analysis in collider physics

The computing model in HEP involves several different approaches on how code is developed and how resources are used. The core part of pre-processing all the data collected at the experiments and generating Monte Carlo simulation data is performed using the main analysis frameworks of the experiments, usually maintained by experienced developers as performance is critical. Programming languages such as C++ remain prevalent and in general changes undergo a review procedure before being put into production. Examples for these core components are Athena [148] for the ATLAS experiment, the CMS offline software (CMS-SW [149]) for CMS, or the Belle II software basf2 [150]. At this point of the analysis chain, raw data is converted into highly structured object data, and finally a reduced set can be derived as filtered structured object data, see Figure 8.

These tasks are executed on the WLCG, which leverages both dedicated resources and an increasing amount of resources that can be used opportunistically. Dedicated workload managers operated by the experiments such as PanDA [152], the CMS workload manager [153] or DIRAC [154] are used on top of the resource's own workload managers such as Slurm [155] or HTCondor [156]. The experiment-specific workload managers wrap the actual computing payload within so-called pilots, which monitor the job execution and report back on resource usage and potential problems to the experiment-specific workload managers and monitoring systems as illustrated in Figure 9. The highly different error conditions of such a large, distributed systems require a significant amount of manpower to hunt down any issues and to intervene if a drop in efficiency is detected. Efforts to automate the most common cases are continuously undertaken [157].

The physicists performing actual data analysis are commonly leveraging such pre-processed data and pre-generated Monte Carlo samples in their analysis tasks. They may be using the core frameworks to further filter the data or develop extensions as part of their work, and can also execute these on the WLCG. However, they usually do not work with the raw data directly, but with data filtering and histogramming tools operating on these pre-filtered data samples stored in a columnar data format as shown in Figure 10. Both the centrally organised *production usage* and the end user analysis are subject to growing computing demands, due to an actual increase in the amount and complexity of the data, and to more complex algorithms employing ML techniques or interactive data filtering with a short feedback loop.

Most production workflows churn through data event by event, producing filtered and highly structured data and/or statistically summing up the individual results. By design, there is no interference between the individual events during analysis (assuming effects such as pile-up have been accounted for). For that reason, the workflows could adapt to dynamic availability of resources if eviction of the jobs without loss of computation time was possible. To accommodate this, frameworks need to be extended to checkpoint either regularly or on demand, as only the current event number and other status data (such as random seeds) and the intermediate output need to be stored. This would allow to suspend the execution without loss of computation time or migrate the computation to a location with underutilised resources, ideally close-by. In the current computing model, though, the processes are executed within a pilot environment that monitors the execution, reports back resource usage, and sends regular heartbeats to the central experiment workload management and monitoring systems. These, in turn, disallow the underlying schedulers of the compute site to checkpoint and restore running jobs or migrate them to another node as the meta-schedulers of the experiments would declare such a job as failed. It should be noted that this can also cause problems with DVFS due to the expected wallclock time being calculated from a previously reported benchmarking value. Removing this limitation would increase the overall efficiency due to better usage of existing resources, for example, by backfilling empty slots opportunistically, without blocking main users of these resources by enabling eviction without loss of computation time.

It has become common in HEP to wrap both the production workflows and the end user analysis in containers with a well-defined environment and software packages maintained by the corresponding communities. While this certainly helps with analysis reproducibility and eases the time spent until a new student can

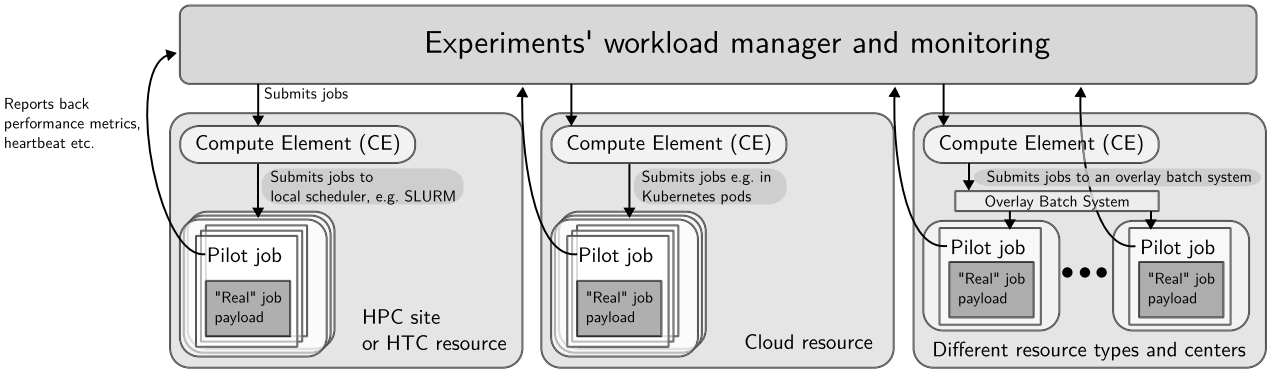

FIGURE 9
Simplified and generalised schematic of job execution on the WLCG. Experiments operate their own workload managers, which submit pilot jobs to compute elements at various resource centres. These in turn submit jobs to local schedulers such as Slurm or HTCondor, cloud resources, or even overlay batch systems grouping various resources into another batch system. The pilots launch the actual payload job and report performance metrics and heartbeats back to the experiments' central monitoring infrastructure. Note that most workflows in HEP rather fit a HTC model as analyses are event-based, i. e., focus is put on the overall throughput aiming to maximise the usage of available resources. Individual compute jobs are commonly limited to single nodes and can use dedicated HTC resources, which might be built from very heterogeneous systems operated by members of the community. The overall model of operation bundles together such dedicated resources, HPC resources and cloud resources into one overlay batch system.

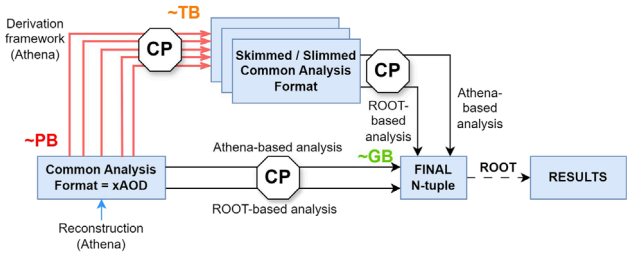

FIGURE 10
Data reduction for end user analysis for the example of the ATLAS experiment. AOD files with structured object data after reconstruction are still sizable and derived data sets are produced for end users. *CP* relates to common calibration and object selections, and in the final stage, columnar n-tuple files are produced for consumption by end user analysis tools [151].

execute code for the first time, it is often also used as a reason not to port the code to a newer base operating system or new releases of the base libraries used within the community. However, porting code is preferred when possible, as it can have positive effects on analysis correctness, performance, and energy efficiency.

An ongoing trend especially in end-user analysis is a large push for interactive data analysis, i.e., fast access to columnar data formats and dynamic filtering for statistical analysis. This requires not only high bandwidth to the actual storage and access to compute resources, which are either dedicated to the use case or can be provisioned on-demand, but also a change to the underlying storage formats for increased performance when changing access patterns quickly between different analyses. RNTuple, which is effectively developed to become the successor of the classic TTree columnar data storage format for arbitrary C++ types and collections, has proven to surpass the existing format in terms of performance and space efficiency. It even outperforms industry standards such as HDF5 and Apache Parquet in access patterns common to HEP analyses [158]. However, adoption of new formats proves to be a slow process due to the plethora of different analysis frameworks built on top of the common ROOT framework [60] and reliance of existing code on specific behaviour of TTree, as backwards compatibility had to be broken.

Some of the user and production workflows in HEP rely on large local scratch space. This is the consequence of modular tooling relying, e.g., on exchange data formats between event generators and analysis tools, such as the verbose *Les Houches* event file format [159]. In terms of performance and efficiency, it would be worthwhile to investigate a more direct way to connect event generators and analysis tools to each other, avoiding on-disk data exchange formats. But this is hindered by large code bases and barriers between different developer subcommunities and programming languages.

It must be noted that most workflows operate in a data streaming model, i.e., data is read and written out event-by-event with the exception of smaller statistical summary output such as histograms. While there is a requirement for high data throughput especially for the input files, there is no actual requirement of POSIX semantics such as locking, directory structures or parallel exclusive file access. Data is accessed through the common base framework ROOT, which can deal well with streaming mode or data access through caches [160]. Still, the operational model usually implies data replication to the site on which workflows are executed. Some of the used tools, especially in end user analysis, also expect POSIX semantics to be available. A stronger decoupling would allow to switch to other storage systems such as object stores as already operated in few large HEP resource centres [161], which would help to reduce the complexity of the necessary storage systems and hence also improve energy efficiency.

Observation 15: *For production usage, the experiment's workload managers should be improved to grant checkpointability and eviction*

functionality, as well as to better handle different operational configurations of the compute resources (e.g., DVFS).

5 Lattice field theory and lattice quantum chromodynamics

Over the past two decades the Standard Model (SM) of particle physics has been largely confirmed as a correct effective description of the interactions of all known fundamental particles. The discovery of the Higgs boson [162, 163] can be argued to mark the transition from an era of experimentally mapping the SM to the so-called *precision frontier* [164] era. In the search for physics beyond the SM, high-energy experiments at the LHC are complemented by B-factories such as LHCb [165] and Belle-2 [166] and by high-intensity experiments in Mainz [167], at Jefferson Lab [168] and at Fermilab [169]. The latter also hosts the Muon $g-2$ [170] experiment aiming at a substantial improvement of the experimental determination of the anomalous magnetic moment of the muon, $a_\mu = (g-2)_\mu$, a key quantity for which the present experimental value disagrees with the data-driven theoretical determination based on dispersion relations [171].

Many of these experimental efforts aim to identify very small deviations from the SM at high energy scales through imprints at much lower energy scales. Their expected smallness necessitates experimental and theoretical precision at the sub-percent level and below and the study of phenomena at low energies in turn implies that hadronic and non-perturbative effects play a significant role. As a result, the achievable theoretical precision depends crucially on non-perturbative calculations using LFT methods, chief of which is LQCD. Simulations in LFT in general and LQCD in particular consume substantial fractions of the available compute budgets on the largest supercomputers and thus contribute significantly to the overall energy consumed by HPC systems (see Figure 1).

The past 20 years have also witnessed first results of LQCD simulations at physical light quark masses [172–175], removing one of the largest sources of systematic uncertainty. In order to achieve the required sub-percent precision with reliable uncertainties, however, such calculations must fully account for strong and electromagnetic isospin breaking effects and be performed in large physical volumes, at fine lattice resolutions and employing very high statistics. Based on current state-of-the-art algorithms, one can easily estimate [176] that this necessitates several Exaflop-years of computing time, showing not only that machines beyond the current generation of Exascale supercomputers are required, but that LFT practitioners must take special care to make efficient use of these resources if the overall energy efficiency of HPC is to be improved. This implies dedicated performance-engineering efforts to ensure that all employed algorithms are implemented in such a way as to achieve performance as close to optimal as possible, subject to machine limitations and the arithmetic intensity of the algorithms in question.

Members of the LFT community have been trail-blazers in the HPC field, having contributed to the development of various architectures over the past 40 years [177–182] as well as the corresponding algorithms to make use of these machines. LFT practitioners were also amongst the earliest adopters of GPU acceleration [183, 184] for key kernels and today, most state-of-the-art calculations would not be possible without GPU offloading. Despite the relatively high level of expertise, the maintenance, improvement and adaptation of the many software frameworks used for LFT calculations in lock-step with current hardware trends has become a substantial RSE challenge also for this community.

Software frameworks in LFT span many hundreds of thousands of lines of code and have accumulated substantial technical debt through their evolution. At the same time many of the algorithms used for LFT calculations have become relatively complex and the hardware landscape has been subject to significant technological diversification and an increased pace of change compared to even just a decade ago (see Section 2.1). As a consequence, LFT software frameworks now have to target multiple memory and execution spaces, multiple communication APIs, and various programming models in order to ensure efficiency and performance-portability.

While historically large parts of these frameworks were written by doctoral students or early-career postdoctoral researchers addressing particular research questions, it is very likely that current challenges can only be met by more dedicated long-term investment into software engineering efforts. This is demonstrated most prominently by the Grid [57, 58] and QUDA [64–66] libraries, which use abstraction through C++ features and, in the case of QUDA, autotuning of kernel launch parameters and communication policies, to provide portable and highly optimised implementations of efficient data structures, computational kernels and algorithms for LFT.

More generally, performance-portability approaches such as Kokkos and SYCL have been explored [185, 186] by the LQCD community and could serve as a relatively easy-to-learn basis for the design of *mostly* performance-portable frameworks to be linked against libraries like Grid or QUDA for particularly demanding kernels or algorithms. Programmer-productivity layers such as Grid Python Toolkit (GPT) [63], Lyncs [61] or PyQUDA [62], which provide Python interfaces to Grid and QUDA, respectively, can instead be used by less experienced students or for exploratory work.

Observation 16: *Optimised performance-portable libraries such as Grid and QUDA give access to highly efficient implementations of various kernels and algorithms and should be used, if possible, to benefit from the many years of development and performance-engineering invested in them. They can be combined with programmer-productivity interfaces like Grid Python Toolkit, Lyncs or PyQUDA to make Grid and QUDA accessible to those without the necessary background to use them directly.*

A calculation in LFT usually proceeds through two stages requiring HPC resources. Generally, the first stage consists in the generation of sets of ensembles of representative field configurations sampled from a probability distribution given by the action of the underlying theory using Markov Chain Monte Carlo (MCMC) methods, usually variants of the Hamiltonian/Hybrid Monte Carlo (HMC) algorithm [187]. By their nature, the samples in a Markov chain have a sequential dependency and the ensemble generation stage in LFT is thus a capability-class computational problem run at the edge of the strong scaling window in order to minimise the real

time required for generating independent samples. Energy efficiency improvements at this stage thus correspond to either significantly improving or even replacing the HMC as a sampling algorithm or improving the efficiency of the various kernels and iterative algorithms that are used in such a simulation.

At the second stage the path integral for observables of interest is evaluated using the representative field configurations generated in the first stage. Most observables in LFT are n-point correlation functions, usually in the time-momentum representation, between operators carrying different quantum numbers as relevant for the problem in question. These give access to the masses and other properties of hadrons as well as matrix elements of operators of interest between these hadronic states. The evaluation of these correlation functions involves the computation of fermion propagators and their combination with projection tensors through so-called *Wick contractions* as well as the injection of momentum through Fourier transforms. These calculations straddle the boundary between capability and capacity problem classes and are generally run on as few compute nodes as possible based on their memory footprint.

Generally speaking both stages are dominated by the repeated solution of linear systems of the type $Dx = b$, where D is a very large, very regular and very sparse and potentially very poorly-conditioned complex-valued matrix called the *Dirac* operator. In state-of-the-art calculations with $128^3 \cdot 256$ lattice points, the vector x represents close to 10^{10} unknowns with near-term future calculations set to increase this by at least one order of magnitude. Depending on the type of fermion discretisation, the application of D can have an arithmetic intensity as low as 0.3 in double precision and is thus strongly memory-bandwidth-bound.

The development of HPC systems since the beginning of the millennium has been marked by a substantial increase in the available peak FP64, a commensurate but slower increase in peak memory bandwidth, and a very slow increase in peak interconnect bandwidth [188, 189]. This is illustrated in Figure 11 which shows per-node ratios of these three metrics for a selection of representative systems employed for LFT calculations (data from [176]). The first two iterations of the BlueGene line of supercomputers, for example, had memory and interconnect bandwidths that were well-balanced for sparse stencil type problems, such that LQCD kernels could achieve in excess of 50% of the peak FP64 rate even when running on many nodes. By contrast, systems which rely on GPUs for their raw computational power can only exploit this potential for problems with high arithmetic intensity such as dense matrix multiplication.

Even more challenging is the fact that interconnect bandwidth has not increased to the same extent as memory bandwidth and floating point performance as it is easily the most significant bottleneck to scalability on current machines. This is somewhat offset by the fact that in order to efficiently use the GPUs on these systems, per-node computational volumes need to be rather large, resulting in surface-to-volume ratios that allow at least for some overlap of communication and computation. Even so, these imbalances negatively affect efficiency such that, on a machine like JUWELS Booster [15], highly optimised kernels only achieve around 10% of the peak FP64 rate on a single node, dropping quickly as the problem is strong-scaled. This is shown in Figure 12 for a double-precision Wilson-clover twisted mass Dirac operator using different lattice volumes $V = 2 \cdot L^4$ as implemented in the QUDA [64–66] library. As a result, even though full system peak performance has increased by about a factor of 100 between BlueGene/L and JUWELS Booster, for example, the effective improvement for LFT calculations has been quite a bit lower.

The moderate growth in the computational power effectively available to LFT calculations has been accompanied by the development of highly algorithmically efficient solvers based on the adaptive multigrid (MG) preconditioning of flexible algorithms such as Flexible Generalised Minimal Residual (FGMRES) or Generalised Conjugate Residual (GCR), at least for a subset of lattice formulations employing Wilson and Wilson-clover [190, 191] or twisted mass [192] fermions. By design, MG algorithms exhibit a low degree of data parallelism at the coarsest level of the aggregation hierarchy. As a result, the efficient implementation of these algorithms on GPUs is only possible through fine-grained optimisations [66] of the underlying computational kernels. Once optimised in this way, the coarsest-grid kernels become dominated by a combination of communication API and network latency, as a result of which the fastest implementations have moved away from MPI and instead make use of proprietary solutions such as NVSHMEM for significantly improved strong-scaling.

The efficiency of MG algorithms is demonstrated in Figure 13, which shows the time required to invert the Wilson-clover twisted mass Dirac operator on a $64^3 \cdot 128$ lattice on 8 JUWELS Booster nodes in the calculation of the fermion derivative in the HMC as a function of the quark mass. The comparison is between two solvers implemented in the QUDA library [64–66], the fastest available mixed-precision Conjugate Gradient (CG) performing a single inversion and MG-preconditioned GCR doing two inversions due to the nature of the problem in question. Even though the MG solver is running with too small a local computational volume, it outperforms API by around a factor of 100 at the physical light quark mass.

Because it is unlikely that the computational imbalance of HPC systems will improve, it is important for the LFT community to invest in a broad spectrum of algorithmic research, even premising that the situation will worsen further. This implies that the target should be to increase arithmetic intensity and reduce interconnect requirements whenever possible. Beyond making use of mixed and adaptive fixed-bit-width [193] representations, arithmetic intensity can also be increased by modifying solvers to operate on *multiple right-hand-sides* where possible [176, 194], also enabling the usage of vendor library routines for efficient batched matrix-multiplication in certain cases. Techniques for communication-avoidance can be applied both at the level of solvers [195, 196] as well as in sampling algorithms, for instance through domain-decomposition [197].

At the level of sampling algorithms, the issue of critical slowing down [198] requires further urgent research to reduce the cost of generating independent samples at fine lattice resolutions. Promising directions of study include flow-based sampling [199], approaches to Fourier acceleration [200–202] and parallel tempering [203]. As a complement to improved sampling, variance-reduction techniques for particularly noisy observables, most

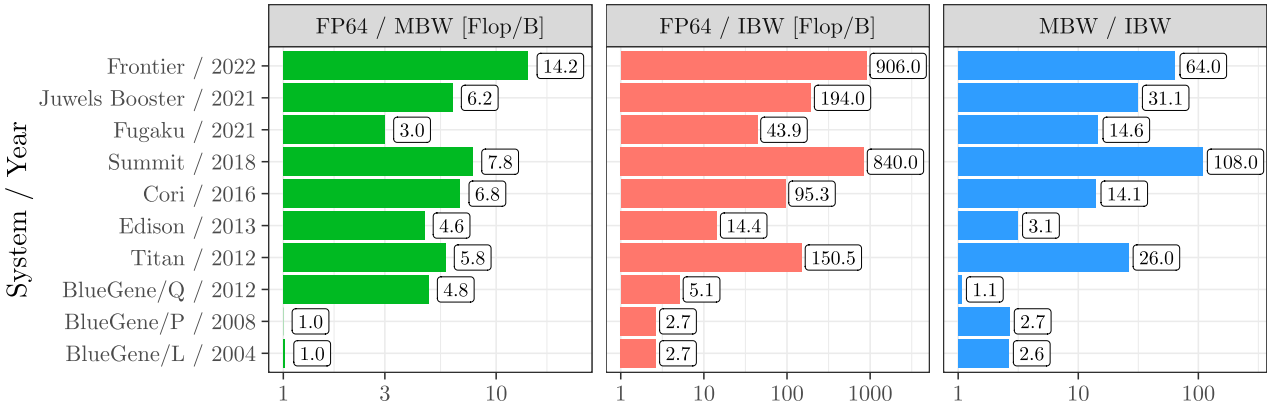

FIGURE 11
Per-node ratios of peak FP64, peak memory bandwidth (MBW) and peak bidirectional interconnect bandwidth (IBW) for representative HPC systems used for calculations in lattice field theory. Data from [176], and available in [210].

notably those with quark-line disconnected contributions, can also be effective for improving the overall energy efficiency of calculations in LFT. A recent example is multigrid multilevel Monte Carlo [204, 205].

In the calculation of correlation functions, if the number of particles in the considered state is larger than two, if the number of required momentum combinations is large, or if the number of insertion points, n, grows beyond three, overall computational cost may be dominated by tensor contractions rather than linear system solves. Sub-expression elimination and the caching of common intermediate results can thus lead to significant efficiency improvements compared to the naïve sequential calculation of a large collection of such correlation functions. It has been shown [206] that very good use can be made of accelerators once memory management [207], as well as dependency management and scheduling [208] have been addressed, and once the many small-matrix-multiplications are streamed in batches [209] to profit from GPU stream parallelism.

In conclusion it can be said that in LFT in general and LQCD in particular energy efficiency is almost synonymous with performance optimisation and algorithmic research. In light of current hardware trends, this implies on the one hand that performance-portability and dedicated performance-engineering are central pillars that allow to maximise machine utilisation and therefore energy efficiency. On the other hand, the problem of critical slowing down and the expected further growth of the imbalance between computational power and memory, as well as interconnect bandwidth, necessitate a broad spectrum of algorithmic study aimed at improved sampling algorithms, variance reduction techniques, and communication-avoiding linear solvers.

Observation 17: *Continued algorithmic research is essential to tackle variance reduction for noisy observables as well as the issue of critical slowing down through improved or accelerated sampling. In addition, the growing imbalance between computational power and memory as well as interconnect bandwidth must be addressed through communication-avoidance approaches.*

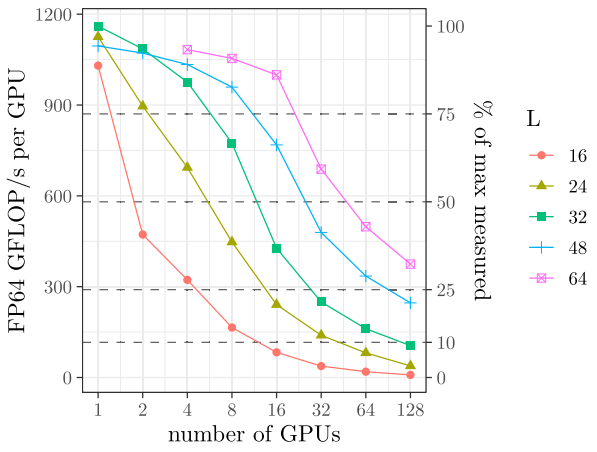

FIGURE 12
Strong scaling study of the double-precision Wilson-clover twisted mass Dirac operator as implemented in the QUDA [64–66] library for different lattice volumes $V = 2 \cdot L^4$ on JUWELS Booster [15]. The points indicate the performance per GPU in GFLOP/s (higher is better) for the different cases with lines added to guide the eye. The right-hand axis indicates the performance relative to the fastest case in percent. The $L = 64$ problem size does not fit on less than 4 GPUs. NVIDIA specifies a peak FP64 rate of 9.7 TFLOP/s for the A100 GPU. Data available in [210].

6 Conclusion

HPC systems are amongst the scientific instruments with the highest energy consumption. To improve their energy efficiency, a combined effort from technology providers, system operators, software developers, and users is required. In particular, application developers and users need to be conscious of energy efficiency from two perspectives: *(i)* they need to know that energy optimisation techniques applied at hardware and system management levels do have an impact on how they experience HPC systems, and *(ii)* they need to learn how to optimise their codes and execution scripts to minimise the energy consumed by their own jobs. This is especially important in the fields of astrophysics, high-energy physics, and

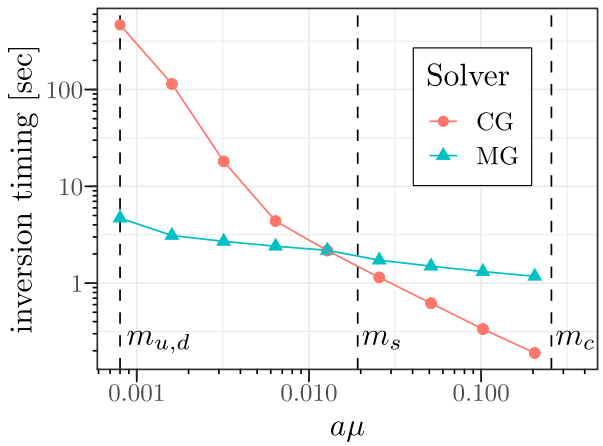

FIGURE 13
Comparison of time-to-solution (lower is better) for the inversion of the Wilson-clover twisted mass Dirac operator in the calculation of the fermion derivative on a $64^3 \cdot 128$ lattice on 8 JUWELS Booster [15] nodes as a function of the quark mass, μ using the fastest mixed-precision conjugate gradient algorithm available performing a single solve (red), and multigrid-preconditioned GCR performing two solves and including unavoidable overheads (blue). The dashed vertical lines indicate the physical bare average up/down (light), strange and charm quark masses in lattice units. Both algorithms are implemented in the QUDA library [64–66]. Data available in [210].

lattice field theory, which consume large volumes of compute hours on HPC resources.

From the system design and operations, the growing hardware heterogeneity (CPUs, GPUs and other accelerators) observed in the recent years is a direct consequence of trying to improve performance per Watt on HPC systems. Heterogeneous computers are more difficult to program, which hinders application portability across systems. High-level programming models, frameworks, and domain-specific languages can help to alleviate the burden, but users should enploy the correct backends and libraries on each platform. Data management is also a main concern for energy efficiency and HPC users should strive to keep their data as local as possible, hide communication behind computation tasks, and employ I/O optimised libraries to optimally use the file system. HPC users must also be aware that they may easily experience performance variability even on the same HPC machine across equal runs, since system management automatically adjusts voltage and frequency of compute devices during operations, and may even reallocate jobs to maximise system throughput for a given energy. Therefore, understanding performance metrics requires consultation of monitoring information and direct contact with HPC user support teams.

What is solely in the hands of the application developers and users is making the application codes themselves more energy efficient, and running them in optimal configurations. Based on our experience, we assume in this paper that nowadays most application developers start their work building on an existing code, which is already parallelised.[3] Their task is then to either add some new aspect to the code, making the simulation more realistic, or improving the performance and scaling of the existing implementation. This means first analysing the code with performance analysis tools, detecting inefficiencies, and making the necessary modifications to maximise the single-node performance (considering also using reduced or mixed arithmetic precision), to then improve its scaling when running in parallel on a growing number of nodes. Once this is done, it should be explored whether the application is tolerant to reductions in the operational frequency of the hardware. Furthermore, application configuration and job settings should always be verified before starting long production runs. The most efficient configuration is likely to be different between HPC systems with diverse architectures, and one cannot rely on pre-tuned configurations. Energy is also saved by ensuring long-term sustainability and availability of any code development, since this avoids iterating past efforts and errors over and over again. Professional software development strategies, including version control, automated testing, CI/CD pipelines, and applying FAIR principles for code and data management are therefore crucial.

It must be stressed that the most important asset in any field is its people. The long-term future of HPC depends on the ability of the community to attract, train and keep the engagement of talents from the younger generations. Therefore, it must be invested in training future HPC experts and providing them with the necessary support and a tolerant and open environment that enable minority groups to enter the field and motivate all to stay. Training courses focused on algorithms, performance analysis and optimisations, data access patterns and frameworks must exist for each user community. It is also crucial to establish career paths for HPC experts, in particular for research software engineers (RSE), who are necessary to optimise code, both from the energy and performance perspective. Unfortunately, the traditional view in the domain sciences determines the academic merits of its members based only on their publication record in the highest impact journals from their fields. This must change, so that application software releases and publications about code improvements are equally valued, recognising their crucial contributions to the advancement of science.

Author contributions

ES: Conceptualization, Writing – original draft, Writing – review and editing. JA: Writing – original draft, Writing – review and editing. MF: Writing – original draft, Writing – review and editing. OF: Writing – original draft, Writing – review and editing. MG: Writing – original draft, Writing – review and editing. BK: Writing – original draft, Writing – review and editing. SP: Writing – original draft, Writing – review and editing.

Funding

The author(s) declare that financial support was received for the research and/or publication of this article. This work was funded in part by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) as part of the CRC 1639/1 NuMeriQS – 511713970. SP acknowledges funding by the DFG as part of the

3 If the start point would be a serial code that the developer is parallelising, it is important to conduct regression tests to guarantee consistency between the results of serial and parallel versions.

PUNCH4NFDI project (project number 460248186) and by the project "NRW-Cluster for data intensive radio astronomy: Big Bang to Big Data (B3D)" funded through the programme "Profilbildung 2020", an initiative of the Ministry of Culture and Science of the State of North Rhine-Westphalia.

Acknowledgments

The authors thank Florian Janetzko (JSC) for providing access to the user statistics represented in Figure 1.

Conflict of interest

Author ES was employed by the company SiPEARL.

The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

The author(s) declared that they were an editorial board member of Frontiers, at the time of submission. This had no impact on the peer review process and the final decision.

Generative AI statement

The author(s) declare that Generative AI was used in the creation of this manuscript. The tool deepl was used in just a few paragraphs, exclusively to perform grammar corrections. No content has been generated with any GenerativeAI tools.

Publisher's note

References

1. Skuse B. The third pillar. *Phys World* (2019) 32:40–3. doi:10.1088/2058-7058/32/3/33

2. Weinzierl T. *The pillars of science*. Cham: Springer International Publishing (2021). p. 3–9. doi:10.1007/978-3-030-76194-3_1

3. Sevilla J, Heim L, Ho A, Besiroglu T, Hobbhahn M, Villalobos P. Compute trends across three eras of machine learning. In: *2022 international joint conference on neural networks (IJCNN)* (2022). p. 1–8. doi:10.1109/IJCNN55064.2022.9891914

4. Jevons W. *The coal question: an enquiry concerning the progress of the nation, and the probable exhaustion of our coal-mines. In: making of the modern world. Part*, 2. Macmillan (1865).

5. CSCS. Annual report 2023 from the Swiss national supercomputing centre (2023). Available online at: https://report2023.cscs.ch/facts-and-figures (Accessed January 23, 2025).

6. EuroHPC Joint Undertaking (2022). *EuroHPC JU 2022 consolidated annual activity report*. Available online at: https://eurohpc-ju.europa.eu/document/download/c7a5fc77-4236-41d8-979d-56af07607c25_en?filename=Annex%20to%20Decision%2012.2023-EuroHPC%20JU%20Consolidated%20Annual%20Activity%20Report%202022_0.pdf (Accessed 2025 January 23)

7. HLRS. HLRS annual report 2023 (2023). Available online at: https://www.hlrs.de/about/profile/annual-report (Accessed January 23, 2025).

8. NERSC. National energy research scientific computing CenterTable of contents, 2023 annual report (2023). Available online at: https://www.nersc.gov/assets/Annual-Reports/2023-NERSC-Annual-Report-compressed.pdf (Accessed January 23, 2025).

9. European Commission. EU energy efficiency directive (2024). Available online at: https://energy.ec.europa.eu/topics/energy-efficiency/energy-efficiency-targets-directive-and-rules/energy-efficiency-directive_en (Accessed October 14, 2024).

10. Moore GE. Cramming more components onto integrated circuits. In: *IEEE solid-state circuits society newsletter [reprinted from electronics (1965)]* (2006). p. 33–5DOI. doi:10.1109/N-SSC.2006.4785860

11. AMD. *AMD CDNA 3 architecture* (2024). Available online at: https://www.amd.com/content/dam/amd/en/documents/instinct-tech-docs/white-papers/amd-cdna-3-white-paper.pdf (Accessed 2024 12 August).

12. NVIDIA (2024). Grace-Hopper architecture. Available online at: https://resources.nvidia.com/en-us-grace-cpu/nvidia-grace-hopper (Accessed 2024 12 August)

13. Suarez E, Eicker N, Lippert T. Modular supercomputing architecture: from idea to production; 3rd. *CRC Press* (2019) 3(chap. 9):223–51. doi:10.1201/9781351036863

14. Suarez E, Eicker N, Moschny T, Pickartz S, Clauss C, Plugaru V, et al. (2022). Modular supercomputing architecture. doi:10.5281/zenodo.6508394

15. Alvarez D. JUWELS cluster and booster: exascale pathfinder with modular supercomputing architecture at juelich supercomputing centre. *J large-scale Res Facil* (2021) 7:A183. doi:10.17815/jlsrf-7-183

16. CINECA (2024). Leonardo: pre-exascale supercomputer. Available online at: https://leonardo-supercomputer.cineca.eu/ (Accessed 2024 12 August)

17. Jülich Supercomputing Centre (2024). JUPITER: the arrival of exascale in Europe. Available online at: https://www.fz-juelich.de/en/ias/jsc/jupiter (Accessed 2024 12 August)

18. LuxProvide (2024). MeluXina. Available online at: https://www.luxprovide.lu/meluxina/ (Accessed 2024 12 August)

19. The University of Tokyo (2024). Wisteria/BDEC. Available online at: https://www.cc.u-tokyo.ac.jp/en/supercomputer/wisteria/service/ (Accessed 2024 12 August)

20. Simmendinger C, Marquardt M, Mäder J, Schneider R. Powersched - managing power consumption in overprovisioned systems. In: *IEEE international conference on cluster computing workshops (CLUSTER workshops)* (2024). doi:10.1109/CLUSTERWorkshops61563.2024.00012

21. Corbalán J, Brochard L (2018). Ear: energy management framework for supercomputers. Link

22. Ilsche T, Schrader S, Schöne R. Optimizing idle power of hpc systems: practical insights and methods. In: *2024 IEEE international conference on cluster computing workshops (CLUSTER workshops)* (2024). p. 19–25. doi:10.1109/CLUSTERWorkshops61563.2024.00014

23. Sundriyal V, Sosonkina M. Reducing idle power consumption in high performance systems. In: *2017 international conference on computational science and computational intelligence (CSCI)* (2017). p. 1629–32. doi:10.1109/CSCI.2017.283

24. Tröpgen H, Schöne R, Ilsche T, Hackenberg D. 16 years of spec power: an analysis of x86 energy efficiency trends. In: *2024 IEEE international Conference on cluster computing workshops (CLUSTER workshops) (IEEE)* (2024). p. 76–80. doi:10.1109/clusterworkshops61563.2024.00020

25. Bhalachandra S, Porterfield A, Prins JF. Using dynamic duty cycle modulation to improve energy efficiency in high performance computing. In: *Proceedings of the 2015 IEEE international parallel and distributed processing symposium workshop (USA: IEEE computer society)*, 15. IPDPSW (2015). p. 911–8. doi:10.1109/IPDPSW.2015.144

26. Bhalachandra S, Porterfield A, Olivier SL, Prins JF, Fowler RJ. Improving energy efficiency in memory-constrained applications using core-specific power control. In: *Proceedings of the 5th international workshop on energy efficient supercomputing*. Association for Computing Machinery (2017). E2SC'17. doi:10.1145/3149412.3149418

27. Kodama Y, Odajima T, Arima E, Sato M. Evaluation of power management control on the supercomputer fugaku. In: *2020 IEEE international conference on cluster computing (CLUSTER)* (2020). p. 484–93. doi:10.1109/CLUSTER49012.2020.00069

28. Auweter A, Bode A, Brehm M, Brochard L, Hammer N, Huber H, et al. A case study of energy aware scheduling on supermuc. In: JM Kunkel, T Ludwig, HW Meuer, editors. *Supercomputing*. Springer International Publishing (2014). p. 394–409. doi:10.1007/978-3-319-07518-1_25

29. Solórzano ALV, Sato K, Yamamoto K, Shoji F, Brandt JM, Schwaller B, et al. Toward sustainable HPC: in-production deployment of incentive-based power efficiency mechanism on the fugaku supercomputer. In: *2024 SC24: international conference for high performance computing, networking, storage and analysis SC*. IEEE Computer Society (2024). p. 342–57. doi:10.1109/SC41406.2024.00030

30. Jülich Supercomputing Centre. LLview job monitoring (2024). Available online at: https://www.fz-juelich.de/en/ias/jsc/services/user-support/software-tools/llview (Accessed October 20, 2024).

31. The ClusterCockpit Project. ClusterCockpit generic datastructure specification (2024). Available online at: https://github.com/ClusterCockpit/cc-specifications/tree/master/datastructures (Accessed October 25, 2024).

32. Li J, Michelogiannakis G, Cook B, Cooray D, Chen Y. Analyzing resource utilization in an HPC system: a case study of NERSC's perlmutter. In: *International conference on high performance computing (ISC)* (2023). p. 297–316. doi:10.1007/978-3-031-32041-5_16

33. Maloney S, Suarez E, Eicker N, Guimarães F, Frings W. Analyzing hpc monitoring data with a view towards efficient resource utilization. In: *2024 IEEE 36th international symposium on computer architecture and high performance computing (SBAC-PAD)* (2024). p. 170–81. doi:10.1109/SBAC-PAD63648.2024.00023

34. Tarraf A, Schreiber M, Cascajo A, Besnard J-B, Vef M-A, Huber D, et al. Malleability in modern hpc systems: current experiences, challenges, and future opportunities. *IEEE Trans Parallel Distrib Syst* (2024) 35:1551–64. doi:10.1109/TPDS.2024.3406764

35. Ishiyama T. Supercomputer simulations of structure formation in the universe. *Proc Int Astron Union* (2016) 12:10–6. doi:10.1017/s174392131700045x

36. Nordlund Å, Ramsey JP, Popovas A, Küffmeier M. dispatch: a numerical simulation framework for the exa-scale era – i. fundamentals. *Mon Not R Astron Soc* (2018) 477:624–38. doi:10.1093/mnras/sty599

37. Tóth G, van der Holst B, Sokolov IV, De Zeeuw DL, Gombosi TI, Fang F, et al. Adaptive numerical algorithms in space weather modeling. *J Comput Phys* (2012) 231:870–903. doi:10.1016/j.jcp.2011.02.006

38. Verbeke C, Baratashvili T, Poedts S. ICARUS, a new inner heliospheric model with a flexible grid. *Astron Astrophys* (2022) 662:A50. doi:10.1051/0004-6361/202141981

39. Xia C, Teunissen J, Mellah IE, Chané E, Keppens R. MPI-AMRVAC 2.0 for solar and astrophysical applications. *Astrophys J Suppl Ser* (2018) 234:30. doi:10.3847/1538-4365/aaa6c8

40. Hotta H, Kusano K. Solar differential rotation reproduced with high-resolution simulation. *Nat Astron* (2021) 5:1100–2. doi:10.1038/s41550-021-01459-0

41. Teichgräber JMR. Julia: a competitive high-level choice for performance portability in hpc? In: *MPCDF seminar (performance) portable programming of HPC applications*. Link (2022).

42. NVIDIA. CUDA toolkit (2024). Available online at: https://developer.nvidia.com/cuda-toolkit (Accessed October 10, 2024).

43. AMD. HIP (2024). Available online at: https://github.com/ROCm/HIP (Accessed October 12, 2024).

44. Cutress I, Shilov A (2019). The larrabee chapter closes: intel's final Xeon Phi processors now in EOL. Available online at: https://www.anandtech.com/show/14305/intel-xeon-phi-knights-mill-now-eol (Accessed 2024 12 August)

45. Python Software Foundation (2025). Python. Available online at: https://www.python.org (Accessed 2025 02 August)

46. OpenMP Architecture Review Board. *OpenMP* (2024). Available online at: https://www.openmp.org/(Accessed October 15, 2024).

47. Khronos. SYCL (2024). Available online at: https://www.khronos.org/sycl/(Accessed October 10, 2024).

48. Trott CR, Lebrun-Grandié D, Arndt D, Ciesko J, Dang V, Ellingwood N, et al. Kokkos 3: programming model extensions for the exascale era. *IEEE Trans Parallel Distributed Syst* (2022) 33:805–17. doi:10.1109/TPDS.2021.3097283

49. Zenker E, Worpitz B, Widera R, Huebl A, Juckeland G, Knüpfer A, et al. Alpaka – an abstraction library for parallel kernel acceleration. In: *IEEE international parallel and distributed processing symposium workshops (IPDPSW)* (2016). doi:10.1109/IPDPSW.2016.50

50. Beckingsale DA, Burmark J, Hornung R, Jones H, Killian W, Kunen AJ, et al. (2019). Raja: portable performance for large-scale scientific applications. In *2019 IEEE/ACM international workshop on performance, portability and productivity in HPC (P3HPC)*. 71–81. doi:10.1109/P3HPC49587.2019.00012

51. Dykes T, Foyer C, Richardson H, Svedin M, Podobas A, Jansson N, et al. Mamba: portable array-based abstractions for heterogeneous high-performance systems. In: *2021 international workshop on performance, portability and productivity in HPC (P3HPC)* (2021). p. 10–21. doi:10.1109/P3HPC54578.2021.00005

52. Ben-Nun T, de Fine Licht J, Ziogas AN, Schneider T, Hoefler T. Stateful dataflow multigraphs: a data-centric model for performance portability on heterogeneous architectures. In: *Proceedings of the international conference for high performance computing, networking, storage and analysis*. New York, NY, USA: Association for Computing Machinery (2019). SC '19. doi:10.1145/3295500.3356173

53. Godoy WF, Valero-Lara P, Dettling TE, Trefftz C, Jorquera I, Sheehy T, et al. Evaluating performance and portability of high-level programming models: julia, python/numba, and kokkos on exascale nodes. In: *2023 IEEE international parallel and distributed processing symposium workshops (IPDPSW)* (2023). p. 373–82. doi:10.1109/IPDPSW59300.2023.00068

54. Hunold S, Steiner S. Benchmarking julia's communication performance: is julia hpc ready or full hpc? In: *IEEE/ACM performance modeling, benchmarking and simulation of high performance computer systems (PMBS)* (2020). doi:10.1109/PMBS51919.2020.00008

55. Vay J-L, Huebl A, Almgren A, Amorim LD, Bell J, Fedeli L, et al. (2021). Modeling of a chain of three plasma accelerator stages with the warpx electromagnetic pic code on gpus. *Phys Plasmas* 28. doi:10.1063/5.0028512

56. Pereira R, Couto M, Ribeiro F, Rua R, Cunha J, Fernandes Ja. P, et al. Energy efficiency across programming languages: how do energy, time, and memory relate? In: *Proceedings of the 10th ACM SIGPLAN international conference on software language engineering*. New York, NY, USA: Association for Computing Machinery (2017). p. 256–67. SLE 2017. doi:10.1145/3136014.3136031

57. Boyle PA, Cossu G, Yamaguchi A, Portelli A. Grid: a next generation data parallel C++ QCD library. *PoS LATTICE2015* (2016) 023. doi:10.22323/1.251.0023

58. Yamaguchi A, Boyle P, Cossu G, Filaci G, Lehner C, Portelli A. Grid: OneCode and FourAPIs. *PoS LATTICE2021* (2022) 035. doi:10.22323/1.396.0035

59. Edwards RG, Joo B. The Chroma software system for lattice QCD. *Nucl Phys B Proc Suppl* (2005) 140:832–4. doi:10.1016/j.nuclphysbps.2004.11.254

60. Brun R, Rademakers F, Canal P, Naumann A, Couet O, Moneta L, et al. (2020). root-project/root: v6.18/02. doi:10.5281/zenodo.3895860

61. Bacchio S, Finkenrath J, Stylianou C. Lyncs-API: a Python API for lattice QCD applications. *PoS LATTICE2021* (2022) 542:542. doi:10.22323/1.396.0542

62. Jiang X, Shi C, Chen Y, Gong M, Yang Y-B. Use quda for lattice qcd calculation with python. *arXiv:2411* (2024):08461. doi:10.48550/arXiv.2411.08461

63. Lehner C, et al. (2024). Grid Python toolkit (GPT). Available online at: https://github.com/lehner/gpt (Accessed 2024 12 March)

64. Babich R, Clark MA, Joo B, Shi G, Brower RC, Gottlieb S (2011). Scaling lattice QCD beyond 100 GPUs. In *SC11 international conference for high performance computing, networking, storage and analysis seattle*, *Washington, November 12-18, 2011*. doi:10.1145/2063384.2063478

65. Clark MA, Babich R, Barros K, Brower RC, Rebbi C. Solving Lattice QCD systems of equations using mixed precision solvers on GPUs. *Comput Phys Commun* (2010) 181:1517–28. doi:10.1016/j.cpc.2010.05.002

66. Clark MA, Joó B, Strelchenko A, Cheng M, Gambhir A, Brower RC (2016). Accelerating lattice QCD multigrid on GPUs using fine-grained parallelization. In *SC '16: proceedings of the international conference for high performance computing, networking, storage and analysis*. 795–806. doi:10.1109/SC.2016.67

67. Gombosi TI, Chen Y, Huang Z, Manchester WB, Sokolov I, Toth G, et al. Adaptive global magnetohydrodynamic simulations. In: *Space and astrophysical plasma simulation*. Cham: Springer International Publishing (2023). p. 211–25. doi:10.1007/978-3-031-11870-8_7

68. Keppens R, Teunissen J, Xia C, Porth O. MPI-AMRVAC: a parallel, grid-adaptive PDE toolkit. *Comput Math Appl* (2021) 81:316–33. doi:10.1016/j.camwa.2020.03.023

69. Lani A, Villedie N, Bensassi K, Koloszar L, Vymazal M, Yalim SM, et al. COOLFluiD: an open computational platform for multi-physics simulation and research. In: *21st AIAA computational fluid dynamics conference*. Reston, Virginia: American Institute of Aeronautics and Astronautics (2013). doi:10.2514/6.2013-2589

70. Godoy W, Valero-Lara P, Teranishi K, Balaprakash P, Vetter J. Evaluation of openai codex for hpc parallel programming models kernel generation. In: *Proceedings of the 52nd International Conference on Parallel Processing Workshops* 23. New York, NY, USA: Association for Computing Machinery, ICPP Workshops ' (2023). p. 136–44. doi:10.1145/3605731.3605886

71. Nichols D, Davis JH, Xie Z, Rajaram A, Bhatele A *Can large language models write parallel code?*. New York, NY, USA: Association for Computing Machinery (2024) 281–94. doi:10.1145/3625549.3658689

72. Forum MPI. Message passing interface documentation (2024). Available online at: https://www.mpi-forum.org/docs/(Accessed November 30, 2024).

73. Numrich RW, Reid J. Co-array fortran for parallel programming. *SIGPLAN Fortran Forum* (1998) 17:1–31. doi:10.1145/289918.289920

74. Carlson WW, Draper JM, Culler DE, Yelick KA, Brooks ED, Warren KH. Introduction to upc and language specification. *CorpusID* (2000):59868665.

75. Nieplocha J, Palmer B, Tipparaju V, Krishnan M, Trease H, Aprà E. Advances, applications and performance of the global arrays shared memory programming toolkit. *Int J High Perform Comput Appl* (2006) 20:203–31. doi:10.1177/1094342006064503

76. Germain JDd. S, McCorquodale J, Parker SG, Johnson CR. Uintah: a massively parallel problem solving environment. In: *Proceedings the ninth international symposium on high-performance distributed computing*. IEEE (2000). p. 33–41. doi:10.1109/HPDC.2000.868632

77. Kale LV, Krishnan S. Charm++: a portable concurrent object oriented system based on c++. *SIGPLAN Not.* (1993) 28:91–108. doi:10.1145/167962.165874

78. Bosilca G, Bouteiller A, Danalis A, Faverge M, Hérault T, Dongarra JJ. Parsec: exploiting heterogeneity to enhance scalability. *Comput Sci and Eng* (2013) 15:36–45. doi:10.1109/MCSE.2013.98

79. Bauer M, Treichler S, Slaughter E, Aiken A. Legion: expressing locality and independence with logical regions. In: *SC'12: proceedings of the international conference on high performance computing, networking, storage and analysis*. IEEE (2012). p. 1–11. doi:10.1109/SC.2012.71

80. Diehl P, Brandt SR, Kaiser H. Shared memory parallelism in modern c++ and hpx. *SN Computer Sci* (2024) 5:459. doi:10.1007/s42979-024-02769-6

81. Kaiser H, Heller T, Adelstein-Lelbach B, Serio A, Fey D. Hpx: a task based programming model in a global address space. In: *Proceedings of the 8th international conference on partitioned global address space programming models* (2014). p. 1–11. doi:10.1145/2676870.267688

82. Thoman P, Dichev K, Heller T, Iakymchuk R, Aguilar X, Hasanov K, et al. A taxonomy of task-based parallel programming technologies for high-performance computing. *The J Supercomputing* (2018) 74:1422–34. doi:10.1007/s11227-018-2238-4

83. Daiß G, Simberg M, Reverdell A, Biddiscombe J, Pollinger T, Kaiser H, et al. Beyond fork-join: Integration of performance portable kokkos kernels with hpx. In: *2021 IEEE international parallel and distributed processing symposium workshops (IPDPSW)* (2021). p. 377–86. doi:10.1109/IPDPSW52791.2021.00066

84. Daiß G, Singanaboina SY, Diehl P, Kaiser H, Pflüger D. From merging frameworks to merging stars: experiences using hpx, kokkos and simd types. In: *2022 IEEE/ACM 7th international workshop on extreme scale programming models and middleware (ESPM2)* (2022). p. 10–9. doi:10.1109/ESPM256814.2022.00007

85. BSC. Dimemas: predict parallel performance using a single CPU machine (2019). Available online at: https://tools.bsc.es/dimemas (Accessed 2024 12 August).

86. Ene D, Anireh VI (2022). Performance evaluation of parallel algorithms. *Int J Computer Sci Eng* 9, 10–4. doi:10.14445/23488387/ijcse-v9i6p102

87. Corbalan J, Smolenko A, Amico MD, Llort G, Mercadal E, Gimeenez J, et al. Deliverable D2.3: Benchmarking, evaluation 1237 and prediction report (2019). Available online at: https://deep-projects.eu/wp-content/uploads/2023/09/DEEP-EST_D23_Benchmarking_evaluation_and_prediction_report_v10.pdf (Accessed January 27, 2025).

88. The HDF Group (2025). Hierarchical data format, version 5

89. Juelich (2024). SIONLib. Available online at: https://apps.fz-juelich.de/jsc/sionlib/docu/index.html. (Accessed 2024 12 January)

90. CERN openlab (2024). CERN openlab project with Cerabyte. Available online at: https://openlab.cern/cerabyte (Accessed 2024 12 January)

91. Wilkinson MD, Dumontier M, Aalbersberg IJ, Appleton G, Axton M, Baak A, et al. The FAIR guiding principles for scientific data management and stewardship. *Scientific Data* (2016) 3:160018. doi:10.1038/sdata.2016.18

92. Kawazura Y, Kimura SS. Inertial range of magnetorotational turbulence. *Sci Adv* (2024) 10:eadp4965. doi:10.1126/sciadv.adp4965

93. McAlpine S, Helly JC, Schaller M, Sawala T, Lavaux G, Jasche J, et al. SIBELIUS-DARK: a galaxy catalogue of the local volume from a constrained realization simulation. *Mon Not R Astron Soc* (2022) 512:5823–47. doi:10.1093/mnras/stac295

94. Riley P, Caplan RM, Downs C, Linker JA, Lionello R. Comparing and contrasting the properties of the inner heliosphere for the three most recent solar minima. *J Geophys Res Space Phys* (2022) 127:e2022JA030261. doi:10.1029/2022JA030261

95. Schaye J, Kugel R, Schaller M, Helly JC, Braspenning J, Elbers W, et al. The FLAMINGO project: cosmological hydrodynamical simulations for large-scale structure and galaxy cluster surveys. *Mon Not R Astron Soc* (2023) 526:4978–5020. doi:10.1093/mnras/stad2419

96. Vasil GM, Lecoanet D, Augustson K, Burns KJ, Oishi JS, Brown BP, et al. The solar dynamo begins near the surface. *Nature* (2024) 629:769–72. doi:10.1038/s41586-024-07315-1

97. Warnecke J, Korpi-Lagg MJ, Gent FA, Rheinhardt M. Numerical evidence for a small-scale dynamo approaching solar magnetic Prandtl numbers. *Nat Astron* (2023) 7:662–8. doi:10.1038/s41550-023-01975-1

98. Williams S, Waterman A, Patterson D. Roofline: an insightful visual performance model for multicore architectures. *Commun ACM* (2009) 52:65–76. doi:10.1145/1498765.1498785

99. GWT-TUD GmbH (2024). Vampir - performance optimization. Available online at: https://vampir.eu/. (Accessed 2024 12 August)

100. BSC. Paraver: a flexible performance analysis tool (2024). Available online at: https://tools.bsc.es/paraver (Accessed 2024 12 August)

101. Geimer M, Wolf F, Wylie BJN, Ábrahám E, Becker D, Mohr B. The scalasca performance toolset architecture. *Concurrency Comput Pract Experience* (2010) 22:702–19. doi:10.1002/cpe.1556

102. Intel (2024). VTune profiler: performance analysis for applications and systems. Available online at: https://www.intel.com/content/www/us/en/developer/tools/oneapi/vtune-profiler.html (Accessed 2024 12 August)

103. Kreuzer A, Kreutz J, Steinbusch B, Huda ZU, Llort G, Corbalan J, et al. *Best practices guide* (forschungszentrum jülich GmbH zentralbibliothek, verlag jülich). *Schriften des Forschungszentrums Jülich IAS Ser* (2021) 48:187–232. FZJ Record: 905853.

104. Dörrich M, Fan M, Kist AM. Impact of mixed precision techniques on training and inference efficiency of deep neural networks. *IEEE Access* (2023) 11:57627–34. doi:10.1109/ACCESS.2023.3284388

105. Google. TPU (cloud) (2024). Available online at: https://cloud.google.com/tpu/docs (Accessed October 15, 2024).

106. Graphcore. Graphcore (2024). Available online at: https://www.graphcore.ai/(Accessed October 15, 2024).

107. Groq Inc. Groq (2024). Available online at: https://groq.com/ (ccessed October 15, 2024).

108. Herten A, Achilles S, Alvarez D, Badwaik J, Behle E, Bode M, et al. Application-driven exascale: the jupiter benchmark suite. In: *Proceedings of the international conference for high performance computing, networking, storage, and analysis*. IEEE Press (2024). SC '24. doi:10.1109/SC41406.2024.00038

109. Wells DC, Greisen EW, Harten RH. FITS - a flexible image transport system. *A&AS SAO/NASA Astrophysics Data Syst* (1981).

110. The Illustris Collaboration (2024). The Illustris project. Available online at: www.illustris-project.org (Accessed 2024 12 August)

111. The TNG Collaboration (2024). The Illustris TNG project. Available online at: www.tng-project.org (Accessed 2024 12 August)

112. The Virgo Collaboration. The Eagle project (2024). Available online at: https://icc.dur.ac.uk/Eagle/ (Accessed January August, 2024).

113. The Horizon Collaboration (2024). The Horizon simulation: modelling galaxy formation in a cosmic framework. Available online at: www.horizon-simulation.org (Accessed 2024 12 August)

114. Springel V, Pakmor R, Zier O, Reinecke M. Simulating cosmic structure formation with the GADGET-4 code. *MNRAS* (2021) 506:2871–949. doi:10.1093/mnras/stab1855

115. Portegies Zwart S, McMillan S. *Astrophysical recipes; the art of AMUSE*. IOP Publishing (2018). doi:10.1088/978-0-7503-1320-9

116. Springel V, White SDM, Jenkins A, Frenk CS, Yoshida N, Gao L, et al. Simulations of the formation, evolution and clustering of galaxies and quasars. *Nature* (2005) 435:629–36. doi:10.1038/nature03597

117. Bate MR, Bonnell IA, Bromm V. The formation of a star cluster: predicting the properties of stars and brown dwarfs. *MNRAS* (2003) 339:577–99. doi:10.1046/j.1365-8711.2003.06210.x

118. Nitadori K, Aarseth SJ. Accelerating NBODY6 with graphics processing units. *MNRAS* (2012) 424:545–52. doi:10.1111/j.1365-2966.2012.21227.x

119. Fryxell B, Olson K, Ricker P, Timmes FX, Zingale M, Lamb DQ, et al. FLASH: an adaptive mesh hydrodynamics code for modeling astrophysical thermonuclear flashes. *ApJS* (2000) 131:273–334. doi:10.1086/317361

120. Rein H, Liu SF. REBOUND: an open-source multi-purpose N-body code for collisional dynamics. *Astron Astrophys* (2012) 537:A128. *astro-ph* 537. doi:10.1051/0004-6361/201118085

121. Bedorf J, Portegies Zwart S. Bonsai-SPH: a GPU accelerated astrophysical smoothed particle hydrodynamics code. *SciPost Astron* (2020) 1:001. doi:10.21468/SciPostAstro.1.1.001

122. Grimm SL, Stadel JG. The GENGA code: gravitational encounters in N-body simulations with GPU acceleration. *Astrophysical J* (2014) 796:23. doi:10.1088/0004-637X/796/1/23

123. Wang L, Iwasawa M, Nitadori K, Makino J. PETAR: a high-performance N-body code for modelling massive collisional stellar systems. *MNRAS* (2020) 497:536–55. doi:10.1093/mnras/staa1915

124. ASCL at Michigan Technological University. Astrophysics source code library (2025). Available online at: https://ascl.net (Accessed January 28, 2025).

125. Deluzet F, Fubiani G, Garrigues L, Guillet C, Narski J (2023). Efficient parallelization for 3d-3v sparse grid particle-in-cell: shared memory architectures. *J Comput Phys* 480, 112022. doi:10.1016/j.jcp.2023.112022

126. Ren J, Luo J, Peng I, Wu K, Li D. *Optimizing large-scale plasma simulations on persistent memory-based heterogeneous memory with effective data placement across memory hierarchy*, 21. ICS ' (2021). p. 203–14. New York, NY: Proceedings of the ACM International Conference on Supercomputing ACM. doi:10.1145/3447818.3460356

127. Romein JW. The tensor-core correlator. *Astron and Astrophysics* (2021) 656:A52. doi:10.1051/0004-6361/202141896

128. Ruzicka J, Asch C, Meneses E, Rampp M, Laure E (2024). A study of performance portability in plasma physics simulations. doi:10.48550/ARXIV.2411.05009

129. Zhang K, Gladman BJ (2022). Glisse: a gpu-optimized planetary system integrator with application to orbital stability calculations. *New Astron* 90, 101659. doi:10.1016/j.newast.2021.101659

130. OpenAlex. The open catalog to the global research system (2025). Available online at: https://openalex.org/(Accessed January 28, 2025).

131. Bacchini F. RelSIM: a relativistic semi-implicit method for particle-in-cell simulations. *Astrophys J Suppl Ser* (2023) 268:60. doi:10.3847/1538-4365/acefba

132. Chen G, Chacón L, Barnes DC. An efficient mixed-precision, hybrid CPU–GPU implementation of a nonlinearly implicit one-dimensional particle-in-cell algorithm. *J Comput Phys* (2012) 231:5374–88. doi:10.1016/j.jcp.2012.04.040

133. Chen Y, Tóth G. Gauss's law satisfying Energy-Conserving Semi-Implicit Particle-in-Cell method. *J Comput Phys* (2019) 386:632–52. doi:10.1016/j.jcp.2019.02.032

134. Lapenta G. Exactly energy conserving semi-implicit particle in cell formulation. *J Comput Phys* (2017) 334:349–66. doi:10.1016/j.jcp.2017.01.002

135. Eggington JWB, Mejnertsen L, Desai RT, Eastwood JP, Chittenden JP. Forging links in earth's plasma environment. *Astron Geophys* (2018) 59(6):6.26–8. doi:10.1093/astrogeo/aty275

136. Hinterreiter J, Magdalenic J, Temmer M, Verbeke C, Jebaraj IC, Samara E, et al. Assessing the performance of EUHFORIA modeling the background solar wind. *Sol Phys* (2019) 294:170. doi:10.1007/s11207-019-1558-8

137. Le A, Daughton W, Karimabadi H, Egedal J. Hybrid simulations of magnetic reconnection with kinetic ions and fluid electron pressure anisotropy. *Phys Plasmas* (2016) 23:032114. doi:10.1063/1.4943893

138. Lottermoser R-F, Scholer M, Matthews AP. Ion kinetic effects in magnetic reconnection: hybrid simulations. *J Geophys Res* (1998) 103:4547–59. doi:10.1029/97JA01872

139. Shi F, Lin Y, Wang X, Wang B, Nishimura Y. 3-D global hybrid simulations of magnetospheric response to foreshock processes. *Earth Planets Space* (2021) 73:138. doi:10.1186/s40623-021-01469-2

140. Swift DW. Use of a hybrid code for global-scale plasma simulation. *J Comput Phys* (1996) 126:109–21. doi:10.1006/jcph.1996.0124

141. Kreuzer A, Eicker N, Amaya J, Suarez E. Application performance on a cluster-booster system. In: *2018 IEEE international parallel and distributed processing symposium workshops (IPDPSW)* (2018). p. 69–78. doi:10.1109/IPDPSW.2018.00019

142. Wang ZJ, Fidkowski K, Abgrall R, Bassi F, Caraeni D, Cary A, et al. High-order CFD methods: current status and perspective. *Int J Numer Methods Fluids* (2013) 72:811–45. doi:10.1002/fld.3767

143. Guillet C. *Sparse approach to accelerate Particle-In-Cell method in 3D*. Theses: Université Paul Sabatier - Toulouse III (2023).

144. Williams JJ, Tskhakaya D, Costea S, Peng IB, Garcia-Gasulla M, Markidis S. *Leveraging HPC profiling and tracing tools to understand the performance of particle-in-cell Monte Carlo simulations*. Springer Nature Switzerland (2024). p. 123–34. doi:10.1007/978-3-031-50684-0_10

145. Yotov K, Roeder T, Pingali K, Gunnels J, Gustavson F (2007). An experimental comparison of cache-oblivious and cache-conscious programs. In *Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures (ACM)*, SPAA07, 93–104. doi:10.1145/1248377.1248394

146. CERN. CERN openlab (2024). Available online at: https://openlab.cern (Accessed November 27, 2024).

147. Intel. OneAPI (2024). Available online at: https://www.intel.com/content/www/us/en/developer/tools/oneapi/overview.htm (Accessed November 27, 2024).

148. ATLAS Collaboration (2019). Athena. doi:10.5281/zenodo.2641997

149. CMS Collaboration. *CMS physics: technical design report volume 1: detector performance and software*. Geneva: CERN: CERN CDS Record (2006). Technical design report. CMS.922757

150. Kuhr T, Pulvermacher C, Ritter M, Hauth T, Braun N. The belle II core software: belle II framework software group. *Comput Softw Big Sci* (2018) 3:1. doi:10.1007/s41781-018-0017-9

151. Elmsheuser J, Anastopoulos C, Boyd J, Catmore J, Gray H, Krasznahorkay A, et al. Evolution of the atlas analysis 1382 model for run-3 and prospects for hl-lhc. *EPJ Web of Conferences* (2020) 245:06014. doi:10.1051/epjconf/202024506014

152. Maeno T, Alekseev A, Barreiro Megino FH, De K, Guan W, Karavakis E, et al. Panda: production and distributed analysis system. *Comput Softw Big Sci* (2024) 8:4. doi:10.1007/s41781-024-00114-3

153. Cinquilli M, Evans D, Foulkes S, Hufnagel D, Mascheroni M, Norman M, et al. The CMS workload management system. *J Phys Conf Ser* (2012) 396:032113. doi:10.1088/1742-6596/396/3/032113

154. Casajus A, Graciani R, Paterson S, Tsaregorodtsev A, on behalf ofthe Lhcb Dirac Team). Dirac pilot framework and the Dirac workload management system. *J Phys Conf Ser* (2010) 219:062049. doi:10.1088/1742-6596/219/6/062049

155. SchedMD. Slurm scheduler (2024). Available online at: https://slurm.schedmd.com/(Accessed September 12, 2024).

156. HTCondor Software Suite (2024). HTcondor collaboration. Available online at: https://htcondor.org/ (Accessed 2024 12 August)

157. Di Girolamo A, Legger F, Paparrigopoulos P, Schovancová J, Beermann T, Boehler M, et al. Preparing distributed computing operations for the hl-lhc era with operational intelligence. *Front Big Data* (2022) 4:753409. doi:10.3389/fdata.2021.753409

158. Lopez-Gomez J, Blomer J. RNTuple performance: status and outlook. *J Phys Conf Ser* (2023) 2438:012118. doi:10.1088/1742-6596/2438/1/012118

159. Alwall J, Ballestrero A, Bartalini P, Belov S, Boos E, Buckley A, et al. A standard format for les houches event files. *Computer Phys Commun* (2007) 176:300–4. doi:10.1016/j.cpc.2006.11.010

160. Boccali T. Computing models in high energy physics. *Rev Phys* (2019) 4:100034. doi:10.1016/j.revip.2019.100034

161. Ellis K, Brew C, Patargias G, Adye T, Appleyard R, Dewhurst A, et al. Xrootd and object store: a new paradigm. *EPJ Web of Conferences* (2020) 245:04006. doi:10.1051/epjconf/202024504006

162. Aad G, Abajyan T, Abbott B, Abdallah J, Abdel Khalek S, Abdelalim A, et al. Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC. *Phys Lett B* (2012) 716:1–29. doi:10.1016/j.physletb.2012.08.020

163. Chatrchyan S, Khachatryan V, Sirunyan A, Tumasyan A, Adam W, Aguilo E, et al. Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC. *Phys Lett B* (2012) 716:30–61. doi:10.1016/j.physletb.2012.08.021

164. Heinrich G. Collider physics at the precision frontier. *Phys Rept* (2021) 922:1–69. doi:10.1016/j.physrep.2021.03.006

165. Lhcb Collaboration LE. *Future physics potential of LHCb*. CERN, Geneva (2022). CERN Record.Tech. rep. 2806113

166. Bertacchi V. Recent results from belle ii. *Nucl Part Phys Proc* (2023) 324-329:107–12. doi:10.1016/j.nuclphysbps.2023.01.022

167. Schlimme S, Aulenbacher K, Baunack S, Berger N, Denig A, Doria L, et al. The MESA physics program. *EPJ Web Conf* (2024) 303:06002. doi:10.1051/epjconf/202430306002

168. Arrington J, Benesch J, Camsonne A, Caylor J, Chen JP, Covrig Dusa S, et al. The solenoidal large intensity device (SoLID) for JLab 12 GeV. *J Phys G* (2023) 50:110501. doi:10.1088/1361-6471/acda21

169. Shiltsev V. Fermilab proton accelerator complex status and improvement plans. *Mod Phys Lett A* (2017) 32:1730012. arXiv:1705.03075. doi:10.1142/s0217732317300129

170. Keshavarzi A. The muon $g-2$ experiment at fermilab. *EPJ Web Conf* (2019) 212:05003. doi:10.1051/epjconf/201921205003

171. Keshavarzi A, Khaw KS, Yoshioka T. Muon $g-2$: a review. *Nucl Phys* (2022) B:115675. doi:10.1016/j.nuclphysb.2022.115675

172. Abdel-Rehim A, Alexandrou C, Constantinou M, Dimopoulos P, Frezzotti R, Hadjiyiannakou K, et al. Nucleon and pion structure with lattice QCD simulations at physical value of the pion mass. *Phys Rev D* (2015) 92:114513. doi:10.1103/PhysRevD.92.114513

173. Abdel-Rehim A, Alexandrou C, Burger F, Constantinou M, Dimopoulos P, Frezzotti R, et al. First physics results at the physical pion mass from $N_f = 2$ Wilson twisted mass fermions at maximal twist. *Phys Rev D* (2017) 95:094515. doi:10.1103/PhysRevD.95.094515

174. Bazavov A, Bernard C, DeTar C, Foley J, Freeman W, Gottlieb S, et al. Leptonic decay-constant ratio $f_K+/f_\Pi+$ from lattice QCD with physical light quarks. *Phys Rev Lett* (2013) 110:172003. doi:10.1103/PhysRevLett.110.172003

175. Durr S, Fodor Z, Frison J, Hoelbling C, Hoffmann R, Katz SD, et al. Ab-initio determination of light hadron masses. *Science* (2008) 322:1224–7. doi:10.1126/science.1163233

176. Boyle P, Bollweg D, Brower R, Christ N, DeTar C, Edwards R, et al. (2022). Lattice qcd and the computational frontier arXiv:2204.00039. doi:10.48550/arXiv.2204.00039

177. Bodin F, Boucaud P, Cabibbo N, Di Carlo F, De Pietri R, Di Renzo F, et al. apeNEXT: a Multi-TFlops computer for elementary particle physics. *Adv Parallel Comput* (2004) 13:355–62. doi:10.1016/S0927-5452(04)80047-9

178. Cabibbo N. APE: a high performance processor for lattice qcd. In: *Symposium on old and new problems in fundamental physics, held in honor of G.C. Wick*. inspireHEP (1984). p. 137–44.

179. Chen D, Christ N, Cristian C, Dong Z, Gara A, Garg K, et al. QCDOC: a 10-teraflops scale computer for lattice QCD. *Nucl Phys B Proc Suppl* (2001) 94:825–32. doi:10.1016/S0920-5632(01)01014-3

180. Haring R, Ohmacht M, Fox T, Gschwind M, Satterfield D, Sugavanam K, et al. The ibm blue gene/q compute chip. *Ieee Micro* (2011) 32:48–60. doi:10.1109/MM.2011.108

181. Mawhinney RD. The 1 Teraflops QCDSP computer. *Parallel Comput* (1999) 25:1281–96. doi:10.1016/S0167-8191(99)00051-4

182. Pleiter D, Maurer T. Qpace – a qcd parallel computer based on cell processors. In: *Proceedings of the XXVII international symposium on lattice field theory — PoS(LAT2009)*. Sissa Medialab (2010). LAT2009, 001. doi:10.22323/1.091.0001

183. Barros K, Babich R, Brower R, Clark MA, Rebbi C. Blasting through lattice calculations using CUDA. *PoS LATTICE2008* (2008) 045. doi:10.22323/1.066.0045

184. Egri GI, Fodor Z, Hoelbling C, Katz SD, Nogradi D, Szabo KK. Lattice QCD as a video game. *Comput Phys Commun* (2007) 177:631–9. doi:10.1016/j.cpc.2007.06.005

185. Joó B, Kurth T, Clark MA, Kim J, Trott CR, Ibanez D, et al. Performance portability of a wilson dslash stencil operator mini-app using kokkos and sycl. In: *2019 IEEE/ACM international workshop on performance, portability and productivity in HPC (P3HPC)*. IEEE (2019). p. 14–25. doi:10.1109/P3HPC49587.2019.00007

186. Schlepphorst S, Krieg S (2023). *Benchmarking a portable lattice quantum chromodynamics kernel written in kokkos and mpi. In proceedings of the SC '23 workshops of the international conference on high performance computing, network, storage, and analysis* (New York, NY, USA: Association for Computing Machinery, SC-W' 23, 1027–37. doi:10.1145/3624062.3624179

187. Duane S, Kennedy AD, Pendleton BJ, Roweth D. Hybrid Monte Carlo. *Phys Lett B* (1987) 195:216–22. doi:10.1016/0370-2693(87)91197-X

188. Khan A, Sim H, Vazhkudai SS, Butt AR, Kim Y. *An analysis of system balance and architectural trends based on top500 supercomputers*, 21. HPCAsia ' (2021). p. 11–22. doi:10.1145/3432261.3432263The International Conference on High Performance Computing in Asia-Pacific Region (Association for Computing Machinery

189. McCalpin J (2022). Memory Bandwidth and system balance in HPC systems. Available online at: https://sites.utexas.edu/jdm4372/2016/11/22/sc16-invited-talk-memory-bandwidth-and-system-balance-in-hpc-systems/ (Accessed 2024 12 August)

190. Brannick J, Brower RC, Clark MA, Osborn JC, Rebbi C. Adaptive multigrid algorithm for lattice QCD. *Phys Rev Lett* (2008) 100:041601. doi:10.1103/PhysRevLett.100.041601

191. Frommer A, Kahl K, Krieg S, Leder B, Rottmann M. Adaptive aggregation-based domain decomposition multigrid for the lattice wilson–Dirac operator. *SIAM J Sci Comput* (2014) 36:A1581–608. doi:10.1137/130919507

192. Alexandrou C, Bacchio S, Finkenrath J, Frommer A, Kahl K, Rottmann M. Adaptive aggregation-based domain decomposition multigrid for twisted mass fermions. *Phys Rev D* (2016) 94:114509. doi:10.1103/PhysRevD.94.114509

193. Clark MA, Howarth D, Tu J, Wagner M, Weinberg E. Maximizing the Bang per bit. *PoS LATTICE2022* (2023) 338:338. doi:10.22323/1.430.0338

194. Boyle PA (2024). Multiple right hand side multigrid for domain wall fermions with a multigrid preconditioned block conjugate gradient algorithm arXiv:2409.03904. doi:10.48550/arXiv.2409.03904

195. Espinoza-Valverde J, Frommer A, Ramirez-Hidalgo G, Rottmann M. Coarsest-level improvements in multigrid for lattice qcd on large-scale computers. *Computer Phys Commun* (2023) 292:108869. doi:10.1016/j.cpc.2023.108869

196. Lehner C, Wettig T. Gauge-equivariant neural networks as preconditioners in lattice qcd. *Phys Rev D* (2023) 108:034503. doi:10.1103/PhysRevD.108.034503

197. Boyle PA, Bollweg D, Kelly C, Yamaguchi A. Algorithms for domain wall Fermions. *PoS LATTICE2021* (2022) 470:470. doi:10.22323/1.396.0470

198. Schaefer S, Sommer R, Virotta F. Investigating the critical slowing down of QCD simulations. *PoS LAT2009* (2009) 032. doi:10.22323/1.091.0032

199 Kanwar G. Flow-based sampling for lattice field theories. In: *40th international symposium on lattice field theory* (2024) arXiv:2401.01297. doi:10.48550/arXiv.2401.01297

200. Abbott R, Albergo MS, Boyda D, Cranmer K, Hackett DC, Kanwar G, et al. Gauge-equivariant flow models for sampling in lattice field theories with pseudofermions. *Phys Rev D* (2022) 106:074506. doi:10.1103/PhysRevD.106.074506

201. Finkenrath J (2022). Tackling critical slowing down using global correction steps with equivariant flows: the case of the Schwinger model arXiv:2201.02216. doi:10.48550/arXiv.2201.02216

202. Jung C, Christ NH. Riemannian manifold HMC with fermions. *PoS LATTICE2023* (2024) 009. doi:10.22323/1.453.0009

203. Bonanno C, Clemente G, D'Elia M, Maio L, Parente L. Full QCD with milder topological freezing. *JHEP 08* (2024) 236. doi:10.1007/JHEP08(2024)236

204. Frommer A, Khalil MN, Ramirez-Hidalgo G. A multilevel approach to variance reduction in the stochastic estimation of the trace of a matrix. *SIAM J Scientific Comput* (2022) 44:A2536–56. doi:10.1137/21M1441894

205. Whyte T, Stathopoulos A, Romero E, Orginos K. Optimizing shift selection in multilevel Monte Carlo for disconnected diagrams in lattice QCD. *Comput Phys Commun* (2024) 294:108928. doi:10.1016/j.cpc.2023.108928

206. Chen J, Edwards RG, Mao W. Graph contractions for calculating correlation functions in lattice QCD. In: *Platform for advanced scientific computing* (2023). doi:10.1145/3592979.3593409

207. Wang Q, Peng Z, Ren B, Chen J, Edwards RG. Memhc: an optimized gpu memory management framework for accelerating many-body correlation. *ACM Trans Archit Code Optim* (2022) 19:1–26. doi:10.1145/3506705

208. Wang Q, Ren B, Chen J, Edwards RG. Micco: an enhanced multi-gpu scheduling framework for many-body correlation functions. In: *2022 IEEE international parallel and distributed processing symposium (IPDPS)* (2022). p. 135–45. doi:10.1109/IPDPS53621.2022.00022

209. Abdelfattah A, Haidar A, Tomov S, Dongarra J. Performance, design, and autotuning of batched GEMM for GPUs. In: JM Kunkel, P Balaji, J Dongarra, editors. *High performance computing*. Springer International Publishing (2016). p. 21–38. doi:10.1007/978-3-319-41321-1_2

210. Janetzko F, Kostrzewa B, Suarez E (2025). Data, build and plot scripts accompanying "energy efficiency trends in hpc: what high-energy and astrophycisists need to know". doi:10.5281/zenodo.14790357

211. Błażej K, Michaela B, Barbara P, Tinatin B, Fan Z, Andrea L, et al. COCONUT, a Novel fast-converging MHD model for solar corona simulations. III. Impact of the preprocessing of the magnetic map on the modeling of the solar cycle activity and comparison with observations. *The Astrophysical Journal* (2023) 942. doi:10.3847/1538-4357/aca483

212. Estela S, Hendryk B, Norbert E, Eitzinger J, Salem E, Thomas F, et al. Energy-aware operation of HPC systems in Germany. *Front High Perfor Comput* (2025) 3. doi:10.3389/fhpcp.2025.1520207