
































Application-Driven Exascale: The JUPITER Benchmark Suite

Andreas Herten , Sebastian Achilles , Damian Alvarez , Jayesh Badwaik , Eric Behle , Mathis Bode ,
Thomas Breuer , Daniel Caviedes-Voullième , Mehdi Cherti , Adel Dabah , Salem El Sayed ,
Wolfgang Frings , Ana Gonzalez-Nicolas , Eric B. Gregory , Kaveh Haghighi Mood , Thorsten Hater ,
Jenia Jitsev , Chelsea Maria John , Jan H. Meinke , Catrin I. Meyer , Pavel Mezentsev , Jan-Oliver Mirus ,
Stepan Nassyr , Carolin Penke , Manoel Römmer , Ujjwal Sinha , Benedikt von St. Vieth , Olaf Stein ,
Estela Suarez , Dennis Willsch , Ilya Zhukov 

Jülich Supercomputing Centre
Forschungszentrum Jülich
Jülich, Germany

Abstract—Benchmarks are essential in the design of modern HPC installations, as they define key aspects of system components. Beyond synthetic workloads, it is crucial to include real applications that represent user requirements into benchmark suites, to guarantee high usability and widespread adoption of a new system. Given the significant investments in leadership-class supercomputers of the exascale era, this is even more important and necessitates alignment with a vision of Open Science and reproducibility. In this work, we present the JUPITER Benchmark Suite, which incorporates 16 applications from various domains. It was designed for and used in the procurement of JUPITER, the first European exascale supercomputer. We identify requirements and challenges and outline the project and software infrastructure setup. We provide descriptions and scalability studies of selected applications and a set of key takeaways. The JUPITER Benchmark Suite is released as open source software with this work at github.com/FZJ-JSC/jubench.

Index Terms—Benchmark, Procurement, Exascale, System Design, System Architecture, GPU, Accelerator

I. INTRODUCTION

The field of High Performance Computing (HPC) is governed by the interplay of capability and demand driving each other forward. During the design and purchase phase of supercomputer procurements for public research, the capability of a machine is usually assessed not only by theoretical, system-inherent numbers, but also by effective numbers relating to actual workloads. These workloads are traditionally benchmark programs that test specific aspects of the system design — like the floating-point throughput, memory bandwidth, or internode latency. While these *synthetic benchmarks* are well-suited for the assessment of distinct features, for a more integrated and realistic perspective, they should be complemented by *application benchmarks*. Application benchmarks use state-of-the-art scientific applications to assess the performance of integrated designs. Complex application profiles utilize various types of hardware resources dynamically during the benchmark’s runtime, showcasing real-world strengths and limitations of the system.

This paper introduces the *JUPITER Benchmark Suite*, a comprehensive collection of 23 benchmark programs meticulously documented and designed to support the procurement of JUPITER, Europe’s first exascale supercomputer. On top of 7 synthetic benchmarks, 16 application benchmarks were developed in close collaboration with domain scientists to ensure relevance and rigor. Additionally, this paper offers valuable insights into the state-of-the-practice of exascale procurement, shedding light on the challenges and methodologies involved.

Preparations for the procurement of JUPITER were launched in early 2022 and finally came to fruition with the awarding of the contract in October 2023. JUPITER is funded by the EuroHPC Joint Undertaking (50 %), Germany’s Federal Ministry for Education and Research (25 %), and the Ministry of Culture and Science of the State of North Rhine-Westphalia of Germany (25 %), and is hosted at Jülich Supercomputing Centre (JSC) of Forschungszentrum Jülich. As part of the procurement, the benchmark suite was developed to motivate the system design and evaluate the proposals committed for the Request for Proposals. The suite focuses on application benchmarks to ensure high practical usability of the system. This work presents the *JUPITER Benchmark Suite* in detail, highlighting design choices and project setup, describing the benchmark workloads, and releasing them as open source software. The suite includes 23 benchmarks across different domains, each with unique characteristics such as compute-intensive, memory-intensive, and I/O-intensive workloads. The applications are grouped into three categories: Base, representing a mixed base workload for the system, High-Scaling, highlighting scalability to the full exascale system, and synthetic, determining various key hardware design features. The benchmark suite represents a first step towards *Continuous Benchmarking* to detect system anomalies during the production phase of JUPITER.

The main contributions of this paper are:

- An in-depth description of the use of benchmarks in HPC procurement, including relevant background information.

- The description of a novel methodology to assess exascale system designs in the form of *High-Scaling benchmarks*.
- The development of suitable benchmark workloads based on a representative set of scientific problems, applications, and synthetic codes.
- Scalability results on the preparation system for all application benchmarks of the suite.
- Insights and best practices learned from application scaling and the procurement process in general.
- Release of the full JUPITER Benchmark Suite as open source software.

After providing background information (section II) and details on the used infrastructure (section III), the suite’s individual benchmarks are presented in section IV. Key takeaways are provided in section V, followed by a conclusion and outlook in section VI.

II. BACKGROUND

A. Requirements

The main requirements of the benchmark suite stem from the need to represent existing and upcoming user communities in the system design process. This ensures a fitting design and fosters adoption of the system by users. The suite must cover the wide user portfolio of the HPC center, containing typical applications from various domains utilizing the current HPC infrastructure, and also represent expected future workloads. Moreover, it is essential to ensure diversity in terms of methods, programming languages, and execution models, since such diversity is an inherent characteristic of the application portfolio for upcoming large-scale systems.

The context of a procurement poses high requirements for *replicability*, *reproducibility*, and *reusability* [1]. Replicability, i.e., the seamless execution on the same hardware by the developer, is an elementary requirement to guarantee robustness. Beyond that, reproducibility describes the seamless execution on different hardware by someone else, making it a key requirement in the context of the procurement since both the site and the system provider must be able to run the suite to obtain the same results. Ensuring reusability, i.e., designing the framework for easy adaptation for a variety of tasks in the future, is essential to justify the substantial investment involved in the creation of the suite.

Vendors participating in the procurement process must also invest considerable time in system-specific adjustments while meeting the procurement’s requirements and complying to its rules, usually within short time scales. Therefore, it is in everyone’s best interest to have clear, well-defined guidelines and, ideally, to leverage an existing benchmark suite to build on established expertise.

B. JUPITER Procurement Scheme

The procurement for the JUPITER system uses a Total-Cost-of-Ownership-based (TCO) value-for-money approach, in which the number of executed reference workloads over the lifespan of the system determines the value. Given the size of state-of-the-art supercomputers as well as their corresponding

power consumption, costs for electricity and cooling are a substantial part of the overall project budget. Using a mixture of application benchmarks as well as synthetic benchmarks, operational costs are computed in a well-defined procedure. While synthetic benchmarks allow for assessing key performance characteristics, they do not allow a realistic assessment of resource consumption during the lifetime of the system for TCO. Therefore, a greater emphasis is placed on application performance rather than on synthetic tests.

Given the targeted system performance of 1 EFLOP/s with 64 bit precision, an additional novel benchmark type focusing on the scale of the system is introduced — *High-Scaling benchmarks*. A subset of applications able to fully utilize JUPITER is identified and use cases were defined to make them part of this novel category. In the course of the paper, we will refer to either *High-Scaling benchmarks* or *Base benchmarks*, to differentiate between both.

The JUPITER system is envisioned to consist of two connected sub-systems, *modules* in the Modular Supercomputing Architecture (MSA) concept [2]. JUPITER Booster is the exascale-level module utilizing massive parallelism through GPUs for maximum performance with high energy efficiency (FLOP/J). JUPITER Cluster is a smaller general-purpose module employing state-of-the-art CPUs for applications with lower inherent parallelism and stronger memory demands. Both compute modules are procured jointly, together with a third module made of high-bandwidth flash storage. The benchmark suite has dedicated benchmarks for all modules, partly even benchmarks spanning Cluster and Booster, dubbed *MSA benchmarks*. Execution targets of the benchmarks are listed in the last columns of Table II.

During the procurement, the results of the execution of the benchmark suite for a given system proposal are weighted and combined to compute a value-for-money metric. The outcomes are compared and incorporated with other aspects into the final assessment of the system proposals.

C. Implementation for JUPITER

The previous two sections outline general requirements for the JUPITER Benchmark Suite. Their assessment and implementation is laid out in the following and are visually sketched in Fig. 1.

Based on an analysis of current and previous compute-time allocations on predecessor systems, the suite covers a variety of scientific areas and includes applications from the domains of Weather and Climate, Neuroscience, Quantum Physics, Material Design, Biology, and others. Beyond that, diversity in workloads is realized: Artificial-Intelligence-based (AI) methods as well as classical simulations, codes based on C/C++ and Fortran, OpenACC and CUDA. Various application profiles are included, such as memory-bound or sparse computations. Future trends of workloads, e.g., the uptake of machine learning algorithms, are inferred from general trends in research communities and from recent changes in allocations on the predecessor system.

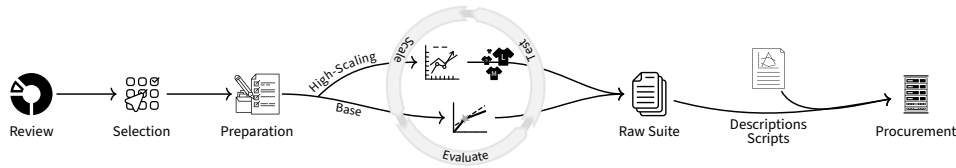


Fig. 1. Major steps in the creation of the JUPITER Benchmark Suite. Based on current workloads, a set of applications is selected. Benchmarks are prepared and then optimized in a feedback loop. Finally, descriptions are revised, and the suite is packaged for procurement.

The requirement of replicability is met by streamlining and automating benchmark execution employing the JUBE framework [3]. Reproducibility is ensured by extensive documentation of all components and the verification of computational results. Thorough testing ensures stable execution in different environments, e.g., with a varying number of compute devices, and lowers risks of commitments by vendors. Reusability is accomplished by a modular design, effective project management workflows, clear licensing, and open source publication. To ensure a high quality of the suite, the benchmarks are standardized through a well-defined setup, with consistent directory structures, uniform descriptions, and similar JUBE configurations. The infrastructure aspects are discussed further in section III).

For each of the Base benchmarks, i.e., the sixteen benchmarks used for the TCO/value-for-money calculation, a Figure-of-Merit (FOM) is identified and normalized to a time-metric. In most cases, the FOM is the runtime of either the full application or a part of it. In case the application focuses on rates, the time-metric is achieved by pre-defining the number of iterations and multiplying with the rate. Each benchmark is executed on a certain number of nodes of the preparation system (see subsection III-A) in order to create a reference execution time. The number of nodes is usually selected to be 8, but deviations are possible due to workload-inherent aspects. This is a trade-off between resource economy and agility in benchmark creation (fewer nodes are more productive), and robustness towards anticipated generational leaps in the hardware of the envisioned system (fewer nodes incorporate latent danger to severely change application profiles when the requested runtime can be achieved with one node). The time-metric, determined on the reference number of nodes, is the value to be improved upon and committed to by proposals of system designs. The number of nodes used to surpass the time-metric can be freely specified by the proposal, but is typically smaller than the reference number of nodes.

Five applications used in the Base benchmarks are capable of scaling to the full scale of the preparation system. They form the additional *High-Scaling* category and define a set of benchmarks that aim to compare the proposed system designs with the preparation system for large-scale executions. By requirement, the future system is known to achieve 1 EFLOP/s High-Performance Linpack (HPL) performance, implying a theoretical peak performance larger than that. In JUPITER’s procurement, runtimes must be committed for the High-Scaling benchmarks using a 1 EFLOP/s(th) (1 EFLOP/s the-

oretical peak performance) sub-partition. For each high-scaling application, a workload is defined to fill a 50 PFLOP/s(th) sub-partition of the preparation system (about 640 nodes) and a $20\times$ larger sub-partition of the future system ($20 \times 50 \text{ PFLOP/s} = 1 \text{ EFLOP/s}$).¹ The final assessment is based on the ratio of the runtime value committed for the future 1 EFLOP/s(th) sub-partition and the reference value. When the benchmark’s specific workload configuration fills up a large portion of the GPU memory on the preparation system, there is a danger that on a proposed system the scaled-up version could become limited by available memory and not showcase an accelerator’s full compute capability. This is due to the trend of growing imbalance between the advancement of compute power and memory. To give more flexibility in system design, up to four reference variants of the respective workload are prepared, taking up 25 % (tiny, T), 50 % (small, S), 75 % (medium, M), and 100 % (large, L) of the available GPU memory on the preparation system (40 GB), respectively. The system proposal may choose the variant that best exploits the available memory on the proposed accelerator after scale-up.

The seven synthetic benchmarks are selected to test individual features of the hardware components, such as compute performance, memory bandwidth, I/O throughput, and network design. Each benchmark has an individual FOM with unique rules and possibly sub-benchmarks, evaluated distinctly.

D. Related Work

Benchmarks have always played an important role in HPC. One objective is the assessment and comparison of technical solutions, whether at the scale of world-leading cluster systems [4], [5] or on a smaller node-level scale [6]–[8]. Benchmarks can be specific to one application topic [9]–[11] or cover multiple areas in the form of suites [12]–[15].

Foundational work focused on identifying common patterns across various applications, with the objective of classifying them based on similar computational and communication characteristics [16]. These patterns, referred to as *dwarfs*, are defined at a high level of abstraction and are intended to capture the most relevant workloads in high-performance computing. Table I categorizes the benchmarks of the JUPITER Benchmark Suite according to these dwarfs, and also gives the predominant scientific domain a benchmark represents.

¹Some benchmarks have algorithmic limitations, like requirement of powers-of-two in node counts. In this case, the smaller, closest compatible number of nodes is taken (for example, 512 nodes).

TABLE I
RELATION OF BENCHMARKS OF THE JUPITER BENCHMARK SUITE TO
DOMAINS[§] AND *Berkeley Dwarfs* [16]; OTHER USE-CASES OF THE
APPLICATIONS MIGHT HAVE OTHER PROFILES.

Benchmark	Domain	Dense LA	Sparse LA	Spectral	Particle	Structured Grid	Unstructured Grid	Monte Carlo
Amber*	MD							
Arbor	Neurosci.							
Chroma-QCD	QCD							
GROMACS	MD							
ICON	Climate							
JUQCS	QC							
nekRS	CFD							
ParFlow*	Earth Sys.							
PICongPU	Plasma							
Quantum Espresso	Materials Sci.							
SOMA*	Polymer Sys.							
MMoCLIP	AI (MM)							
Megatron-LM	AI (LLM)							
ResNet*	AI (Vision)							
DynQCD	QCD							
NASUA	Biology							
Graph500	Graph							
HPCG	CG							
HPL	LA							
IOR	Filesys.							
LinkTest	Network							
OSU	Network							
STREAM	Memory							

* The benchmarks were prepared for the procurement, but not actually used.

§ The following abbreviations are used: *MD* - Molecular Dynamics; *QCD* - Quantum Chromo Dynamics; *QC* - Quantum Computing; *CFD* - Computational Fluid Dynamics; *MM* - Multi-Modal; *LLM* - Large Language Model; *LA* - Linear Algebra; *P2P* - Point-to-Point; *DMA* - Direct Memory Access.

While dwarfs can be viewed as a set of blueprints for application-inspired synthetic benchmarks, their simplicity by design prevents them to fully capture all dynamics of real applications. To address this issue in supercomputer co-design, more complex and versatile computational motifs, termed *octopodes*, are proposed by Matsuoka, Domke, Wahib, *et al.* [17].

SPEC [18], [19] is one of the most extensive, well-known benchmarking initiatives aimed at commercial users. The benchmark suite *SPECaccel2023* uses the offloading APIs OpenACC and OpenMP to measure performance with computationally intensive parallel applications, following the principle “same source code for all”. Benchmarks for the use case of system design and exascale procurement [20] have specific requirements (see subsection II-A) and are typically not made publicly available due to concerns regarding sensitive information and elaborate implementation. The PRACE Unified European Applications Benchmark Suite [21] represents a step towards a culture of open sharing, but its future support is uncertain. Other notable efforts include the CORAL-2 benchmarks [22] used for procurement of the three exascale systems in the US, and the recently developed NERSC-10 Benchmark

Suite [23] used in an ongoing procurement. HPC centers can benefit from each other’s experience, driven by a spirit of Open Science, reproducibility, and sustainable software development [1]. The integration of DevOps principles, such as Continuous Benchmarking, is gaining popularity to support these aims [24], [25].

III. BENCHMARK INFRASTRUCTURE

A. Preparation Systems

The JUPITER Benchmark Suite was prepared on JUWELS, in particular JUWELS Booster, a top 20 system [26] hosted at JSC [27]. JUWELS Booster was installed in 2020 and provides a performance of 73 PFLOP/s(th) peak and 44 PFLOP/s for the HPL. The system is connected to the JUST 5 storage system [28]. JUWELS Booster provides 936 GPU nodes integrated into 39 Eviden BullSequana XH2000 racks, with 2 racks (48 nodes) building a *cell* in the DragonFly+ topology of the high-speed interconnect. Each node has 4 NVIDIA A100 GPUs and 4 NVIDIA Mellanox InfiniBand HDR200 adapters, with one adapter available per GPU. 2 AMD EPYC Rome 7402 CPUs (2×24 cores) are connected to 512 GB DDR4 memory.

Preparations for the High-Scaling benchmarks utilized a 50 PFLOP/s(th) sub-partition of the JUWELS Booster.

JUWELS provides general software dependencies through EasyBuild [29]. Reproducibility is achieved by either using upstream installation recipes, *easyconfigs*, or upstreaming custom recipes.

B. JUBE

Every benchmark is implemented in the JUBE [3] workflow environment to facilitate productive development and reproducibility. In benchmark-specific definition files, *JUBE scripts*, parameters and execution steps (compilation, computation, data processing, verification) are defined. These are then interpreted by the JUBE runtime, resolving dependencies and eventually submitting jobs for execution to the batch system. By inheriting from system-specific definition files, *platform.xml*, batch submission templates are populated and independence of the underlying system is achieved. The various sub-benchmarks and variants are implemented by *tags*, which select different versions of parameter definitions. After execution, the benchmark results are presented by JUBE in a concise tabular form, including the FOM.

Within the JUPITER procurement, the JUBE scripts are part of the documentation. They exactly define execution parameters and instructions with descriptive annotations. A textual documentation is provided as part of the benchmark-accompanying description.

C. Descriptions

Beyond the execution reference through JUBE scripts, each benchmark is accompanied by an extensive description. All descriptions are normalized, using identical structure with similar language. Example parts are information about the source and the compilation, execution parameters and rules,

TABLE II
OVERVIEW OF COMPONENTS OF THE JUPITER BENCHMARK SUITE. SOME DEFINING DETAILS ARE GIVEN. IF USED, SIGNIFICANT LIBRARIES ARE SHOWN (ALL BENCHMARKS USE MPI FOR DISTRIBUTION). FOR **HIGH-SCALE** BENCHMARKS, THE SUPER-SCRIPT INDICATES THE AVAILABLE MEMORY VARIANTS (TINY, SMALL, MEDIUM, LARGE). FOR **MODULE/DEVICE**, THE FOLLOWING ABBREVIATIONS ARE USED: **B_e** – EXECUTION ON GPUS OF JUPITER BOOSTER, **B_s** – EXECUTION ON THE CPUS OF JUPITER BOOSTER, **C_e** – EXECUTION ON THE CPUS OF JUPITER CLUSTER, **M_{GC}** – MSA EXECUTION WITH CPUS OF JUPITER CLUSTER AND GPUS OF JUPITER BOOSTER.

Application Features			Execution Targets			
Benchmark Name	Progr. Language, [Libraries,]Prog. Models	Licence	Nodes Base	Nodes High-Scale $N^{\text{Mem Vars}}$	Module/Device B_e B_s C_e M_{GC}	
Application	Amber*	Fortran, CUDA	Custom	1		✓
	Arbor	C++, CUDA/HIP	BSD-3-Clause	8	642 ^{T,S,M,L}	✓
	Chroma-QCD	C++, QUDA, CUDA/HIP	JLab	8	512 ^{S,M,L}	✓
	GROMACS	C++, CUDA/SYCL	LGPLv2.1	3/128		✓
	ICON	Fortran/C, OpenACC/CUDA/HIP	BSD-3-Clause	120/300		✓
	JUQCS	Fortran, CUDA/OpenMP	None	8	512 ^{S,L}	✓
	nekRS	C++/C, OCCA, CUDA/HIP/SYCL	BSD-3-Clause	8	642 ^{S,M,L}	✓
	ParFlow*	C, Hypre, CUDA/HIP	LGPL	4		✓
	PICongPU	C++, Alpaka, CUDA/HIP	GPLv3+	8	640 ^{S,M,L}	✓
	Quantum Espresso	Fortran, ELPA, OpenACC/CUF	GPL	8		✓
	SOMA*	C, OpenACC	LGPL	8		✓
	MMoCLIP	Python, PyTorch, CUDA/ROCm ¹	MIT	8		✓
	Megatron-LM	Python, PyTorch/Apex, CUDA/ROCm ¹	BSD-3-Clause	96		✓
	ResNet*	Python, TensorFlow, CUDA/ROCm ¹	Apache-2.0	10		✓
	DynQCD	C, OpenMP	None	8		✓
Synthetic	NASdA	C++, MPI	MPL-2.0	8		✓
	Graph500	C, MPI	MIT	4/16/all		✓
	HPCG	C++, OpenMP, CUDA/HIP	BSD-3-Clause	1/4/all		✓
	HPL	C, BLAS, OpenMP, CUDA/HIP	BSD-4-Clause	1/16/all		✓
	IOR	C, MPI	GPLv2	-> 64 [§]		✓
	LinkTest	C++, MPI/SIONlib	BSD-4-Clause+	all		✓
	OSU	C, MPI, CUDA	BSD	1/2		✓
	STREAM	C, CUDA/ROCm/OpenACC	Custom	1		✓
						✓
Benchmark Name	Progr. Language, [Libraries,]Prog. Models	Licence	Nodes Base	Nodes High-Scale $N^{\text{Mem Vars}}$	Module/Device B_e B_s C_e M_{GC}	

* The benchmarks were prepared for the procurement, but not actually used.

¹ For PyTorch and TensorFlow, CUDA and ROCm backends are available; through *extensions*, also backends for Intel GPUs exist (not in mainline repositories).

[§] IOR features two sub-benchmarks, *easy* and *hard*. The number of nodes is a free parameter in easy. In hard, it can also be chosen freely, as long as more than 64 nodes are taken.

detailed instructions for execution and verification, sample results, and concluding commitment requests. In all relevant sections, relations to the JUBE scripts are made in addition to the textual descriptions. For the vendors, the use of JUBE is recommended but not mandatory.

PDFs generated from the benchmark descriptions are part of the committed procurement documentation, including hashes of archived benchmark repositories.

D. Git and Submodules

All components of a benchmark are available in a single Git repository as a single source of truth. A common structure is established, containing description, JUBE scripts, auxiliary scripts, benchmark results, and the source code of the benchmarked application. Utilizing the attached issue tracker, project management and collaboration are facilitated.

Per default, the sources are included as references in the form of Git Submodules. Submodules enable a direct linkage

to well-defined versions of source code, but do not unnecessarily clutter the benchmark repository by static, potentially extensive copies. They are well integrated into the Git workflow and easily updated. In cases where inclusion as a Git Submodule is not possible, scripts and detailed instructions for download are provided.

For delivery as part of the procurement specification package, each benchmark repository is archived as a tar file.

If too large for inclusion in the Git repository, input data is provided as a separate download, including a verifying hash.

E. Project Management

The benchmark suite development efforts were supported efficiently by clear project management workflows over several months.

A core team of HPC specialists and scientific researchers initiated the process early, curating a list of potential benchmarks based on their expertise and experience with existing

HPC systems, while also incorporating insights from previous procurements. In close collaboration with domain scientists, this list was gradually refined to ensure a balanced and diverse selection that met the requirements outlined in subsection II-A.

Competent teams, each led by a team captain, were responsible for individual applications. GitLab issues were used to document biweekly meetings and track per-application progress in the form of a pre-defined checklist with 11 points (ranging from source code availability, over JUBE integration, to description creation). Collaborative hack days facilitated collaboration while running the benchmarks on the preparation system.

IV. BENCHMARKS

This section describes the JUPITER Benchmark Suite, containing 23 benchmarks: 7 synthetic and 16 application benchmarks. In the procurement process, the number of application benchmarks was reduced to 12. Table II gives an overview of all benchmarks, including application features and execution targets.

With this work, the JUPITER Benchmark Suite is released as open source software at <https://github.com/FZJ-JSC/jubench>, with individual repositories for each benchmark [30]–[52].

A. Application Benchmarks

In the following, we describe in detail eleven of the application benchmarks, divided into Base and High-Scaling benchmarks. The benchmarks are based on prominent workloads in the HPC community and were specifically developed for the suite. Given their complex computational dynamics, a certain level of technical and scientific background is necessary and will be provided accordingly. The remaining benchmarks, Amber, ParFlow, SOMA, ResNet, and DynQCD, are briefly introduced first for completeness; they are either using closed-source software (DynQCD) or were ultimately not used for the JUPITER procurement (the others).

- *Amber* [53], [54] is a popular commercial molecular dynamics code for biomolecules. The Satellite Tobacco Mosaic Virus (STMV) case from the Amber20 benchmark suite [55] (1 067 095 atoms) is chosen. The code is mainly optimized for single GPU calculations and is not intended to scale beyond a single node.
- *ParFlow* [56], [57] is a massively-parallel, open source, integrated hydrology model for surface and subsurface flow simulation. The ClayL test from ParFlow’s test suite (simulating infiltration into clay soil) is selected, with a problem size of $1008 \times 1008 \times 240$ cells [58].
- *SOMA* [59] performs Monte Carlo simulations for the “Single Chain in Mean Field” model [60], studying the behaviour of soft coarse-grained polymer chains in a solution.
- *ResNet* [61] uses convolutions and residual connections for training deep neural networks and serves as a reference model in computer vision tasks. The suite includes ResNet50, implemented in TensorFlow with Horovod.

- *DynQCD* [62] is a CPU-only code which performs numerical simulations for Lattice Quantum-Chromodynamics (LQCD). The benchmark generates 600 quark propagators using a conjugate gradient solver for sparse LQCD fermion matrices, with high demands to the memory sub-system.

1) *Base Benchmarks (Selection)*: The Base benchmarks are designed to incentivize system designs that optimize the time to solution. They are first executed on a reference number of nodes on the preparation system (see subsection II-C). Figure 2 gives an overview of application runtimes and respective strong scaling behaviors for surrounding number of nodes. While the absolute number can be used to judge system designs quantitatively, the strong scaling behavior can be used as an additional data point to understand the overall design qualitatively. Note the example for reading the graph in the figure caption.

a) *GROMACS*: GROMACS [63]–[66] is a versatile package to perform molecular dynamics simulations, focusing on biochemical molecules and soft condensed matter systems. The application integrates Newton’s equations of motion for systems with hundreds to millions of particles and provides time-resolved trajectories. Two biological systems from the Unified European Applications Benchmark Suite (UEABS) [21] are used, test cases A and C. Test case A simulates a GluCl ion channel embedded in a membrane. Test case C contains 27 replicas of the STMV with about 28 000 000 atoms and allows testing the scalability of system-supplied Fast Fourier Transform (FFT) libraries.

b) *ICON*: The ICOSahedral Non-hydrostatic model (ICON) [67] is a modelling framework for weather, climate, and environmental prediction used for operational weather forecasting at the German Weather Service. ICON also provides an Earth System Model for climate simulations, i.e., a general circulation model of the atmosphere, including a land module [68] and an ocean model [69]. While the atmosphere part has been ported to GPUs [70], the ocean component is still running on CPUs only. ICON is available under a permissive open source licence. The JUPITER benchmark case is based on the atmosphere component, with global forecast simulation in two resolutions, resulting in two sub-benchmarks: R02B09 (5 km grid point distance) and R02B10 (2.5 km grid point distance) [71]. The coarser resolution is targeted for execution on 120 nodes, and the finer resolution is for 300 nodes. While reasonable scaling to $2\times$ the node count (240 nodes and 600 nodes, respectively) is possible, this is not the usual mode of operation for ICON. These simulations are crucial for ICON’s development towards a storm-resolving climate model with 1 km resolution or even less [72]. A unique aspect of the ICON benchmark is its large input dataset: R02B09 requires 1.8 TB of data, R02B10 needs 4.5 TB. Therefore, the ICON benchmark also tests the performance of I/O operations on a system.

c) *Megatron-LM*: Megatron-LM [73] is a prominent codebase in Natural Language Processing (NLP), known for its vast scale and performance capabilities. It employs the

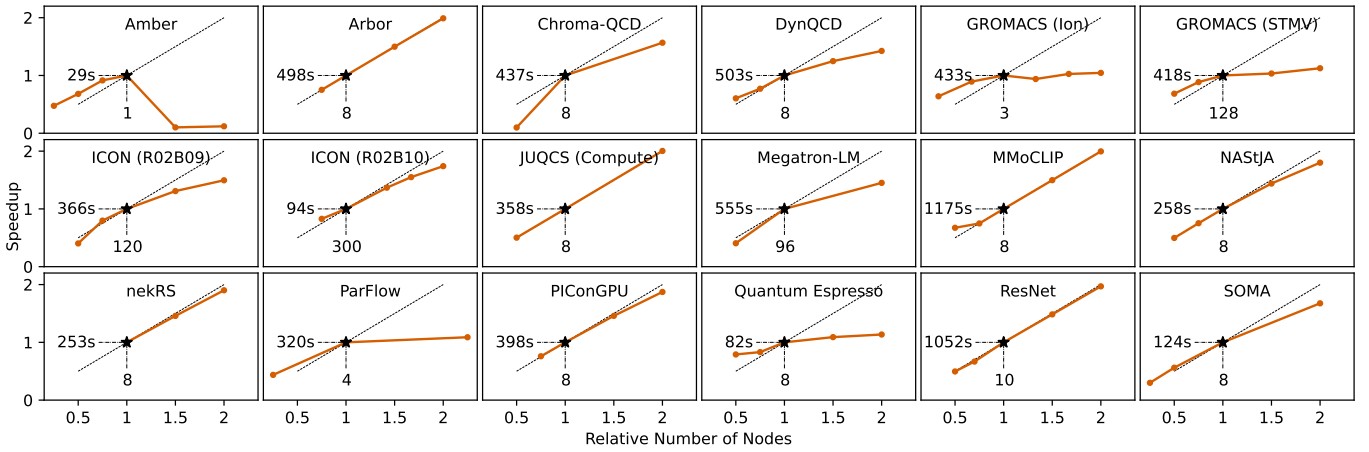


Fig. 2. Overview of relative runtimes of all Base applications on the reference system, JUWELS Booster. Shown at (1,1) is the execution on the reference number of nodes with the reference runtime, with the absolute values shown close to the horizontal and vertical axes, respectively. Beyond the reference execution, strong-scaled relative runtimes (with respect to the reference runtime) on the surrounding number of nodes are given (usually $0.5\times$, $0.75\times$, $1.5\times$, and $2\times$ the reference; some benchmarks deviate). As an example, consider Arbor: the reference number of nodes (8) is noted at horizontal 1, the reference runtime of 498 s at vertical 1; further data-points are given for 4 nodes (663 s), 12 nodes (332 s), and 16 nodes (250 s) – $0.5\times$, $1.5\times$, and $2\times$ the reference number of nodes of 8. See also Table II.

transformer model architecture [74] and leverages various parallelization techniques and optimizations [75]–[78] through PyTorch to achieve high hardware utilization with excellent efficiency. The training of various recent open source GPT-like models was carried out with Megatron-LM [79], making this benchmark crucial to assess a system’s capability to handle disruptive generative AI workloads. The benchmark trains a 175 billion parameter model, converting the usual throughput metric (tokens per time) to a hypothetical time-to-solution FOM by training 20 million tokens.

d) *MMoCLIP*: Contrastive Language-Image Pre-training (CLIP) [80] is a method that conducts self-supervised learning on weakly-aligned image-text pairs with open vocabulary language, resulting in language-vision foundation models. The approach enables usage of substantially increased datasets, like web-scale datasets [81], [82]. The OpenCLIP [83], [84] codebase, an open source implementation of CLIP, enables efficiently distributed training of CLIP models by using multiple data parallelism schemes through PyTorch, scaling to more than a thousand GPUs. Due to its strong transferability and robustness, OpenCLIP is used in diverse multi-modal learning approaches and downstream applications [85]–[87], and efficient training is crucial for the machine learning community dealing with open, fully reproducible foundation models.

The MMoCLIP benchmark is curated from OpenCLIP. It trains an ViT-L-14 model on a synthetic dataset of 3 200 000 image-text pairs and records the total training time as a FOM.

e) *Quantum ESPRESSO*: Quantum ESPRESSO (QE) [88]–[90] is an open source, density-functional-theory-based electronic structure software used both in academia and industry. QE calculates different material properties using a plane wave basis set and pseudo-potentials and can exploit novel accelerators well [91]. The dominant kernel in

QE performs a three-dimensional FFT, which is usually a memory-bound kernel and is communication-bound for large systems [91].

For the benchmark suite, the *Car-Parrinello Molecular Dynamics* model was chosen. The benchmark is based on a use case created in the MaX project [92], [93] and does calculations for a slab of ZrO₂ with 792 atoms.

f) *NASTJA*: The Neoteric Autonomous Stencil code for Jolly Algorithms (NASTJA) is a massively-parallel simulation framework of biological tissues using a Cellular Potts Model [94], [95]. This model relies on nearest neighbour interactions and is parallelized by dividing the overall workload into multiple sub-regions, called blocks. Each block is treated independently by an MPI process, with boundaries being exchanged. Using NASTJA, tissues composed of thousands to millions of cells can be simulated at subcellular resolution [96]. As a test case, adhesion-driven cell sorting is used, a common process in tissue development and segregation [97]. The benchmark investigates the first 5050 Monte Carlo (MC) steps of a system of size $720 \times 720 \times 1152 \mu\text{m}^3$, containing roughly 600 000 cells. NASTJA utilizes MPI for parallelization/distribution and is one of the few CPU-only benchmarks in the suite. The application exhibits an irregular memory access pattern at each iteration, which is not suitable for GPU execution.

2) *High-Scaling Benchmarks*: In this section, we describe the five High-Scaling benchmarks in detail.

Figure 3 juxtaposes the weak-scaling behaviours of the applications over a wide range of node numbers, using the reference HPC system JUWELS Booster.

a) *Arbor*: Arbor is a library for simulating biophysically-realistic neural networks, bridging the gap between point and nanoscale models [98]. Developed in the HBP [99], it aims at efficient use of modern HPC hardware behind an

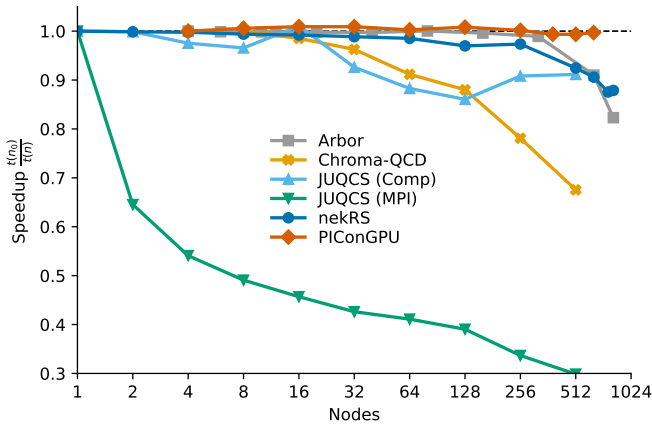


Fig. 3. Weak scaling efficiency of the five High-Scaling benchmarks over a wide range of JUWELS Booster node numbers. For JUQCS, two lines are drawn; one for the computation and one for the communication (see section IV-A2c).

intuitive interface. Neurons are modelled by morphology, ion channels, and connections. Arbor is written in C++ with a Python interface and available under a permissive open source license. The user-centric description is discretized and aggregated to optimize data layouts for individual hardware. At runtime, the *cable equation* is integrated alternating with a system of ODEs for the channels. Users model channels via a domain-specific language that must be compiled for the target hardware. Communication is performed, concurrently with time evolution, every n steps, determined by neural delays. The benchmark is parameterized to fill the GPU memory in the variants T, S, M, L. To differentiate from point models, it is weighted heavily towards computation, emphasized by a sparse connectivity. A complex cell from the Allen Institute was selected and adapted to random morphologies of fixed depth [100]. Cells are organized into rings propagating a single spike. Rings are interconnected to place load on the network without altering dynamics, yielding a deterministic, scalable workload. Profiling shows two cost centers: 52% ion channels and 33% cable equation; hiding communication completely. The Base version runs on 32 JUWELS Booster nodes, filling all 4 GPUs' memory. This was scaled up to the full Booster to verify efficient resource usage and extrapolated to 1 EFLOP. The number of generated spikes is used for validation.

b) Chroma: Chroma [101] is an all-purpose application for LQCD computations. It is compiled on top of the USQCD software stack [102], which provides LQCD-specific libraries for communication, data-parallelism, I/O, and, importantly, sparse linear solver libraries optimized for different architectures. Key libraries used in this benchmark include QMP for MPI wrapping, QDP-JIT for data-parallelism and parallel-I/O via QIO, and the GPU-targeted QUDA solver library [103]. Chroma and the USQCD stack are open source and community-developed. Chroma is one of the most widely used LQCD suites and is representative of LQCD codes in general.

LQCD calculations generally depend heavily on solving very large, regular, sparse linear systems (dimension $10^6 - 10^9$ generally). Due to the regularity of the data and the calculation, LQCD lends itself to many levels of concurrency.

The Chroma LQCD benchmark in the JUPITER Benchmark Suite contains the representative benchmarks for the Hybrid Monte-Carlo (HMC) component of the LQCD simulations. In the benchmark, a number of HMC update trajectories are performed using the 3 + 1 flavours of Clover Wilson fermions — three light quark flavours with identical mass, and a fourth flavour with heavier mass — and the Lüscher-Weisz gauge action. The 4D lattice is initialized with a random $SU(3)$ element on each link. Checkpointing is disabled by a source patch to remove the I/O overhead for the calculations. The benchmark also contains a fix to Chroma, allowing simulation of 4D lattice volumes greater than 2^{31} and among the largest LQCD simulations anywhere to date. The benchmark performance is sensitive to the decomposition configuration used for distributing the 4D lattice to different tasks and to the affinity between the CPU, GPU, NUMA domains, and the network controller for each task.

The benchmark is validated by comparing the output with a reference solution with a tolerance of 10^{-10} for the Base benchmark and 10^{-8} for High-Scaling benchmarks.

The relevant metric (FOM) is the total time spent in HMC updates, excluding the first update, which includes overhead for tuning QUDA parameters. So a minimum of two updates must be prescribed.

c) JUQCS: JUQCS is a massively parallel simulator for universal gate-based Quantum Computers (QCs) [104] written in Fortran 90 using MPI, OpenMP, and CUDA. During the past decades, JUQCS has been used to benchmark some of the largest supercomputers worldwide, including the Sunway TaihuLight and the K computer [105] as well as JUWELS Booster [106], and it was part of Google's quantum supremacy demonstration [107]. Various versions of JUQCS are available in binary form as part of a container [108]; a light version with available sources was created specifically for this benchmark suite [109]. JUQCS simulates an n -qubit gate-based QC by iteratively updating a rank- n tensor of 2^n complex numbers (state vector) stored in double precision and distributed over the supercomputer's memory. The total available memory determines the size of the largest QC that can be simulated. For instance, a universal simulation of $n = 45$ qubits requires a little over $16 \times 2^{45} \text{ B} = 0.5 \text{ PiB}$. Many operations require the transfer of half of all memory, i.e., $2^n/2$ complex double-precision numbers, across the network, which can help to assess the performance of a supercomputer's communication network [105], [106]. As Fig. 3 shows, the deviation of JUQCS w.r.t. the theoretically expected linear scaling (green triangles) reveals both a drop in performance from intra-node to inter-node GPU communication (from 1 to 2 nodes) and another drop when communication enters the large-scale regime at 256 nodes.

All present JUQCS benchmarks simulate successive applications of a single-qubit quantum gate that requires large

memory transfers. The Base benchmark simulates $n = 36$ qubits requiring 1 TiB of GPU memory. The High-Scaling benchmark contains two memory variants: a large memory variant with $n = 42$ qubits requiring 64 TiB (L) and a small memory variant with $n = 41$ qubits requiring 32 TiB of GPU memory (S). Rules are given for extrapolation to an Exascale setup using $n = 46$ (L) or $n = 45$ qubits (S). For all test cases, verification is done using the theoretically known results [105].

In addition, an MSA version of the JUQCS benchmark simulates $n = 34$ qubits on both JUWELS Cluster and Booster simultaneously. The total amount of memory is split into two parts, with 128 GiB residing on the CPU nodes and 128 GiB residing on the GPU nodes. MPI is used for communication between the Cluster and the Booster, and the number of MPI tasks is similarly split into two. On the Cluster, each MPI task launches 12 OpenMP threads, with one thread per CPU core. On the Booster, each MPI task controls one of the GPUs.

d) nekRS: nekRS [110] is a fast CFD solver designed for GPUs that solves the low-Mach Navier-Stokes equations (NSEs), potentially coupled with multiphysics effects. nekRS has been run at scale on many large supercomputers, featuring excellent time-to-solution due to its high GPU throughput, and was nominated for the 2023 Gordon-Bell Award [111]. nekRS uses high-order spectral elements [112] in which the solution, data, and test functions are represented as locally structured N^{th} -order tensor product polynomials on a set of E globally unstructured curvilinear hexahedral brick elements. There are two key benefits to this strategy. First, high-order polynomial expansions significantly reduce the number of unknowns ($n \approx EN^3$) needed to achieve engineering tolerances. Second, the locally structured forms allow tensor product sum factorization, which yields low $\mathcal{O}(n)$ storage cost and $\mathcal{O}(nN)$ work complexity [113]. The leading order $\mathcal{O}(nN)$ work terms can be cast as small dense matrix-matrix products with good computational intensity [114]. nekRS is written in C++ and the kernels are implemented using the portable Open Concurrent Compute Abstraction (OCCA) library [115] for abstraction between different parallel languages/hardware architectures.

The benchmark case is derived from a Rayleigh-Bénard convection (RBC) application [116], [117] which simulates turbulence induced by a temperature gradient — a typical case executed at scale. The simulation domain is a *sheet*. It is much more extended in the periodic directions than in the wall-bounded direction. The chosen polynomial order is 9 with 600 time steps per run. Verification is based on pre-computed results and derived tolerances. The High-Scaling benchmark variants use between 28 836 900 (*small*, $\sim 11\,229$ per GPU) and 57 760 000 (*large*, ~ 22.492 per GPU) elements, which is more than the minimum number of elements required for the "strong scaling limit" of 7000 – 8000 elements per GPU. The Base benchmark case uses 719 104 elements resulting in 22 472 elements per GPU.

e) PIConGPU: PIConGPU is an open source, fully relativistic particle-in-cell (PIC) code designed for studying laser-plasma interactions and astrophysical phenomena. It uses the PIC algorithm with several key components, namely, parti-

cle initialization, charge calculations using grid interpolation, field calculations using densities, and time-marching due to Lorentz force. This approach allows particles to interact via fields on the grid rather than direct pairwise interactions, reducing computational steps from N^2 to N for N particles. PIConGPU employs a unique data model with asynchronous data transfers to handle the computational challenge. It can simulate complex plasma systems with billions of particles on GPU clusters [118]. PIConGPU is developed with a hardware-agnostic approach using the Alpaka library [119], [120], providing outstanding performance across all supported platforms, like CPUs, AMD and NVIDIA GPUs, and FPGAs [121]. The benchmark suite uses a 3D test-case simulating the Kelvin-Helmholtz Instability (KHI), a non-relativistic shear-flow instability, utilizing a pre-ionized hydrogen plasma with periodic boundary conditions. While relevant for various research communities, the nature of shear-flows and the use of periodic boundary conditions does not impose a significant load imbalance throughout the simulation. Therefore, the performance of the code is based on its structure rather than the physics of the problem. In the KHI use case, the number of particles per cell is kept constant to 25, using as many cells as the GPU memory allows. A grid size of $\vec{x} = (4096, 2048, 1024)$ is chosen for the small memory variant, and extended to $(4096, 2048, 2048)$ (M) and $(4096, 4096, 2560)$ (L) for the larger variants. PIConGPU employs domain decomposition for distribution, dividing the computational domain into smaller subdomains along three dimensions. To distribute along these three dimensions, the maximum number of nodes that can be utilized is limited to 640, rather than 642.

B. Synthetic Benchmarks

The JUPITER Benchmark Suite also includes seven well-known synthetic benchmarks: Graph500, HPCG, HPL, IOR, LinkTest, OSU, and STREAM (including a GPU variant). The IOR and LinkTest implementation are presented in the following, highlighting some unique aspects of the setup.

a) IOR: IOR is the de facto standard for measuring I/O performance and is being used by the IO500 [122] to compare the I/O characteristics of storage systems. The benchmark provides a large list of parameters such as block size, transfer size, API, and task reordering, which in turn allows simulating multiple I/O patterns. To target the high-bandwidth, NVMe-based JUPITER storage module, the upper and lower bounds on the mean read and write bandwidth are our focus in the Benchmark Suite. Similar to IO500, two variants of the IOR are implemented, *Easy* and *Hard*. The Easy variant requires a transfer size of 16 MiB, with each process writing to its own file. The Hard variant uses a transfer size of 4 KiB and a block size of 4 KiB, with all processes writing and reading a single file. The setup forces multiple processes to write to the same file system data block, stressing the filesystem with the lock processes. The remaining parameters were selected to avoid caching effects. The number of nodes is a free parameter (with a lower bound) to allow for optimization on the level of parallelism that the underlying file system can provide.

b) *LinkTest*: LinkTest [123] is designed to test point-to-point connections between processes in serial or parallel mode and is capable of handling very large numbers of processes. It is an essential tool in network operations, used mostly internally by system administrators for acceptance testing, maintenance, and troubleshooting.

The JUPITER Benchmark Suite utilizes LinkTest’s *bisection* test, to concisely evaluate interconnectivity between parts of the system’s network, quantified by a single metric. In the bisection test, a number of test processes (one per high-speed network adapter) is separated to two equal halves of the system, and messages are bounced between partnering processes in parallel (bidirectional mode). To achieve optimal bandwidth, the message size is set to 16 MiB. An assessment is made mainly based on the minimum bisection bandwidth.

V. LESSONS LEARNED

We developed the JUPITER Benchmark Suite building upon our experience from previous HPC system procurements. The suite constitutes a substantial expansion from those earlier endeavours, and should be considered as a living object that will continue to evolve over time. In the following, we summarize the lessons learned, covering first the perspective of application developers, then the benchmark suite creation, and finally the overall procurement process.

A. Application Development

The applications of the JUPITER Benchmark Suite not only need to be executed on current large-scale resources like JUWELS Booster, but also need to be extrapolated to larger future resources, amplifying scaling challenges.

To understand the performance characteristics on a future system better, it proved useful for some application developers to create **models** of their applications. JUQCS, for example, has a non-trivial weak scaling behaviour. In the benchmark, the execution time is reported in relation to an ideal scenario, enabling comparability. A model was developed for nekRS to predict the performance of a later part of the simulation early in the process, allowing much shorter and more resource-efficient benchmarks. During scalability studies for PIconGPU, a model for the scaling behaviour could be developed, informing valid simulation parameters for the benchmark setup.

While the approximate scale of the future system is known, the details of the setup are not. When domain decomposition is important for performance, preliminary studies are usually employed to determine the best parameters for production runs. But **decomposition studies** are impossible in the benchmark context, especially for an unknown system design. Through labour- and resource-intensive investigation, estimates, rules, or scripts for ideal domain decomposition were devised, e.g., for Chroma-QCD, PIconGPU, NASTJA, and DynQCD. This also documented the experience of individual researchers, improving reproducibility. To understand different scaling regimes of the application, a network communication model was developed for JUQCS. The model can be employed

to understand topological aspects of the high-speed network of current and future systems, for example with respect to congestion.

The preparation for **future system designs** had a direct effect on application development. For example, it became apparent for Arbor developers that they need to optimize memory usage, as memory capacity and bandwidth will continue to extend more slowly compared to compute performance. During benchmark preparation, they also needed to trade highly-valued user experience for scalability, as the approach of referring to connection endpoints with labels did not scale as required. A short-term solution (using local indexing) was found for the suite, and a hash-based solution is being developed upstream. On a similar note, Chroma-QCD authors needed to patch the code to facilitate execution on the envisioned scale. Unexpected effects can occur depending on the extrapolation method to the future system. For Chroma-QCD, it was found out that the employed benchmark is not guaranteed to converge, and a cut-off after a certain number of iterations is a more robust approach.

Result **verification** is essential for a benchmark suite to ensure the validity of submitted results. Yet, the experience with verification during the suite preparation varied. Some results could be verified either exactly (JUQCS), or within a certain numerical limit by comparing to a pre-computed solution (Chroma-QCD); more involved simulations were verified by extracting key metrics from the computed solution for comparison to a model (ICON, nekRS). The verification of some applications with iterative algorithms, which were stopped before convergence, relied on framework-inherent verification and required key data in the output (PIconGPU, Megatron-LM) — arguably the weakest form of verification.

B. Benchmark Design

Creating a vast benchmark suite that picks up the status quo in workloads, and bringing it to mature levels, is a **resource-intensive endeavour**. Beyond the human resources, compute resources of the reference system constitute a significant investment. To use these resources efficiently, it is important to design benchmarks with runtimes as short as possible, while keeping it as large as needed. Short runtimes also enable swift turn-around times for rapid prototyping – especially useful in a large suite. The size of the input data and the files generated at runtime should be minimized to ensure that the benchmark suite is easy to handle. For reproducibility, non-core application parts like pre-processing or data-staging should be kept short. Modelling domain decomposition effects, also beyond typical production execution profiles, is further consuming resources.

Preparing for target system designs with **unknown details and scales** beyond available resources is a demanding task. Care should be taken to consider future hardware trends in the benchmark design. To explicitly accommodate different compute-to-memory ratios, up to four memory variants of benchmarks were introduced. On the preparation system, the memory variants can be used to study artificially-limited com-

pute profiles and determine possible bottleneck shifts on future systems. With unknown hardware, algorithmic behaviours might shift as well, and iterations may not converge. A more robust approach is to not compute until convergence, but stop after a predetermined amount of iterations.

Some parameters in the benchmark suite are free to chose, like the number of nodes, others are fixed, for example simulation parameters. Thorough **execution rules** and modification guidelines determine the envisioned outcome and need to be developed as part of the suite. Beyond general rules, benchmarks may explicitly deviate to either loosen or tighten rules. Parameter validation should be part of the overall verification process; further extending on the importance of the verification task.

C. Procurement

Using application benchmarks in the procurement of HPC systems is essential to realistically represent user requirements when deciding the configuration of the future system. An additional challenge was given by the particular system architecture of JUPITER, in which **two compute modules** of very disparate sizes, a small CPU-only module and an exascale GPU-accelerated module, are coupled together with a shared storage module. It took some discussions until finding the right number and balance between CPU and GPU benchmarks, which ended up being in the ratio of about 1:5.

To formulate and develop the benchmarks, it proved fruitful to **collaborate closely** with the domain researchers intending to utilize the system. Established relationships in joint projects are especially productive, while it is more difficult for new user domains. A fundamental limitation of our approach is the **reliance on existing application codes** executed on current systems. By design, disruptive approaches are not well covered and a tendency to favour evolutionary technology is introduced. However, considering the effort associated with adopting new technologies among HPC users, this focus on incremental developments is justified. Still, predicting future system **usage trends** is crucial — like the AI applications in the suite, which aim at representing a user domain expected to gain importance over time. However, the rapidly evolving software and algorithms in this domain make it hard to accurately estimate their future needs. It is therefore important to consider the most recent breakthroughs in AI beyond the HPC context, including also commercially-driven domains.

The **time window** for the development of the JUPITER Benchmark Suite was limited by the constraints of the procurement process. The endeavour started several months before the procurement, and required dedicated work by tens of people. Clear management structures and collaboration platforms were essential **tools for extensive collaboration**. In particular, transparent communication with all bidders was crucial, which was possible thanks to the dialogue phase that was part of the procurement. The suite development fostered collaboration, team-building, and knowledge-sharing. Code and environment optimizations were openly shared between benchmark developers and vendors, iteratively improving the

benchmarks further. The suite itself is now open source and can **benefit the wider HPC community**.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we presented the JUPITER Benchmark Suite, which has been successfully employed in the procurement process of the exascale system JUPITER. The suite has served as a valuable tool in assessing HPC system performance during the procurement and beyond. The benchmark applications (see section IV) were chosen to represent the workloads on the future system after careful consideration of requirements and constraints. Bidders used this suite to test different technologies, put together their proposals, and prepare and commit the associated performance numbers. The procuring entity selected the vendor based on these values, together with multiple additional evaluation criteria. At the time of writing, the JUPITER system is being installed. The benchmark suite will be employed again during the acceptance procedure.

The JUPITER Benchmark Suite lays a foundation to further extend and automate HPC system benchmarking. Facilitated by the reusable design of the suite, Continuous Benchmarking will be realized as future work, employing the CI/CD features of GitLab in conjunction with novel tools such as Jacamar [25]. Running the suite at regular intervals (e.g., after maintenances), we will ensure that the system does not see performance degradation over its lifetime or after updates. Application optimization for JUPITER will continue during the system deployment and installation phase, utilizing experiences gained and tools created. We will strive for further improvements regarding the reproducibility of individual benchmarks, including a focus on verification. Also, individual technical enhancements are in progress (for example, using git-annex for the large input data).

ACKNOWLEDGMENT

Many benchmarks presented in this work have roots in projects that received funding from national and European sources; for example, Quantum Espresso (from the MaX project [124]) or nekRS (from the CoEC project [125]). The authors thank the funding agencies for their support and the colleagues from the projects for being available for advice when designing the suite.

The authors would like to thank the following people for their contributions: Max Holicki and Yannik Müller, for their contributions to the LinkTest benchmark; Jonathan Windgassen and Christian Witzler for their support with the nekRS benchmark; Hans De Raedt for the discussions and help with the JUQCS benchmark.

The double-anonymous review of this publication was enabled by the *Anonymous GitHub* service at <https://anonymous.4open.science/>. The authors thank the creator, also for the immediate, last-minute support.

Finally, the authors would like to thank EuroHPC JU, BMBF, MKW-NRW, and GCS for the supportive atmosphere, productive discussions, and for the financial support of the JUPITER project.

REFERENCES

- [1] J. Fehr, J. Heiland, C. Himpe, and J. Saak, “Best practices for replicability, reproducibility and reusability of computer-based experiments exemplified by model reduction software,” *AIMS Mathematics*, vol. 1, no. 3, pp. 261–281, 2016, ISSN: 2473-6988. DOI: 10.3934/Math.2016.3.261.
- [2] E. Suarez, N. Eicker, T. Moschny, *et al.*, “Modular Supercomputing Architecture: A success story of European R&D,” *Tech. Rep. 9*, 2022, p. 24. DOI: 10.5281/ZENODO.6508394.
- [3] T. Breuer, J. Wellmann, F. Souza Mendes Guimarães, C. Himmels, and S. Luehrs, *JUBE*, version REL-2.6.1, Nov. 2023. DOI: 10.5281/zenodo.10228432.
- [4] *The TOP500 list*, <http://www.top500.org>, Accessed: 2024-03-13.
- [5] J. J. Dongarra, P. Luszczek, and A. Petitet, “The LINPACK benchmark: Past, present and future,” *Concurrency and Computation: Practice and Experience*, vol. 15, no. 9, pp. 803–820, 2003. DOI: <https://doi.org/10.1002/cpe.728>.
- [6] A. Sorokin, S. Malkovsky, G. Tsoy, A. Zatsarinnyy, and K. Volovich, “Comparative performance evaluation of modern heterogeneous high-performance computing systems CPUs,” *Electronics*, vol. 9, no. 6, 2020, ISSN: 2079-9292. DOI: 10.3390/electronics9061035.
- [7] R. Xu, X. Tian, S. Chandrasekaran, Y. Yan, and B. Chapman, “NAS parallel benchmarks for GPGPUs using a directive-based programming model,” in *Languages and Compilers for Parallel Computing*, J. Brodman and P. Tu, Eds., Cham: Springer International Publishing, 2015, pp. 67–81, ISBN: 978-3-319-17473-0. DOI: 10.1007/978-3-319-17473-0_5.
- [8] C. Bienia, S. Kumar, J. P. Singh, and K. Li, “The PARSEC benchmark suite: Characterization and architectural implications,” in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT ’08, Toronto, Ontario, Canada: Association for Computing Machinery, 2008, pp. 72–81, ISBN: 9781605582825. DOI: 10.1145/1454115.1454128.
- [9] A. Herten, T. Hater, W. Klijn, and D. Pleiter, “Performance comparison for neuroscience application benchmarks,” in *High Performance Computing*, M. Weiland, G. Juckeland, S. Alam, and H. Jagode, Eds., Cham: Springer International Publishing, 2019, pp. 418–431, ISBN: 978-3-030-34356-9. DOI: 10.1007/978-3-030-34356-9_31.
- [10] J. Albers, J. Pronold, A. C. Kurth, *et al.*, “A modular workflow for performance benchmarking of neuronal network simulations,” *Frontiers in Neuroinformatics*, vol. 16, 2022, ISSN: 1662-5196. DOI: 10.3389/fninf.2022.837549.
- [11] S. Farrell, M. Emani, J. Balma, *et al.*, “MLPerf™ HPC: A holistic benchmark suite for scientific machine learning on HPC systems,” in *2021 IEEE/ACM Workshop on Machine Learning in High Performance Computing Environments (MLHPC)*, Los Alamitos, CA, USA: IEEE Computer Society, Nov. 2021, pp. 33–45. DOI: 10.1109/MLHPC54614.2021.00009.
- [12] J. Dongarra and P. Luszczek, “HPC challenge: Design, history, and implementation highlights,” in *Contemporary High Performance Computing: From Petascale toward Exascale*, J. S. Vetter, Ed. Chapman and Hall/CRC, 2013, vol. 1, ch. 2, ISBN: 9781466568358.
- [13] J. A. Stratton, C. I. Rodrigues, I.-J. Sung, *et al.*, “Parboil: A revised benchmark suite for scientific and commercial throughput computing,” University of Illinois at Urbana-Champaign, Center for Reliable and High-Performance Computing, Technical Report IMPACT-12-01, 2012. [Online]. Available: <http://impact.crhc.illinois.edu/Shared/Docs/impact-12-01.parboil.pdf>.
- [14] A. Danalis, G. Marin, C. McCurdy, *et al.*, “The scalable heterogeneous computing (SHOC) benchmark suite,” in *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, ser. GPGPU-3, Pittsburgh, Pennsylvania, USA: Association for Computing Machinery, 2010, pp. 63–74, ISBN: 9781605589350. DOI: 10.1145/1735688.1735702.
- [15] S. Che, M. Boyer, J. Meng, *et al.*, “Rodinia: A benchmark suite for heterogeneous computing,” in *2009 IEEE International Symposium on Workload Characterization (IISWC)*, 2009, pp. 44–54. DOI: 10.1109/IISWC.2009.5306797.
- [16] K. Asanović, R. Bodik, B. C. Catanzaro, *et al.*, “The landscape of parallel computing research: A view from berkeley,” *Tech. Rep. UCB/EECS-2006-183*, Dec. 2006. [Online]. Available: <http://www2.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.html>.
- [17] S. Matsuoka, J. Domke, M. Wahib, *et al.*, “Preparing for the future—rethinking proxy applications,” *Computing in Science & Engineering*, vol. 24, no. 2, pp. 85–90, 2022. DOI: 10.1109/MCSE.2022.3153105.
- [18] G. Juckeland, W. Brantley, S. Chandrasekaran, *et al.*, “SPEC ACCEL: A standard application suite for measuring hardware accelerator performance,” in *High Performance Computing Systems. Performance Modeling, Benchmarking, and Simulation*, S. A. Jarvis, S. A. Wright, and S. D. Hammond, Eds., Cham: Springer International Publishing, 2015, pp. 46–67, ISBN: 978-3-319-17248-4. DOI: 10.1007/978-3-319-17248-4_3.
- [19] S. Boehm, S. Pophale, V. G. Vergara Larrea, and O. Hernandez, “Evaluating performance portability of accelerator programming models using SPEC ACCEL 1.2 benchmarks,” in *High Performance Computing*, R. Yokota, M. Weiland, J. Shalf, and S. Alam, Eds., Cham: Springer International Publishing, 2018,

- pp. 711–723, ISBN: 978-3-030-02465-9. DOI: 10.1007/978-3-030-02465-9_51.
- [20] N. Malaya, B. Messer, J. Glenski, *et al.*, “Experiences readying applications for exascale,” in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC ’23, Denver, CO, USA: Association for Computing Machinery, 2023, ISBN: 9798400701092. DOI: 10.1145/3581784.3607065.
 - [21] PRACE, *UEABS: Unified European Applications Benchmark Suite*, <https://repository.prace-ri.eu/git/UEABS/ueabs>, Accessed: 2024-03-13, 2022.
 - [22] *CORAL-2 benchmarks*, <https://asc.llnl.gov/coral-2-benchmarks>, Accessed: 2024-08-23.
 - [23] *NERSC-10 benchmark suite*, <https://www.nersc.gov/systems/nersc-10/benchmarks/>, Accessed: 2024-03-22.
 - [24] O. Pearce, A. Scott, G. Becker, *et al.*, “Towards collaborative continuous benchmarking for HPC,” in *Proceedings of the SC ’23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, ser. SC-W ’23, Denver, CO, USA: Association for Computing Machinery, 2023, pp. 627–635, ISBN: 9798400707858. DOI: 10.1145/3624062.3624135.
 - [25] R. Adamson, P. Bryant, D. Montoya, *et al.*, “Creating continuous integration infrastructure for software development on DOE HPC systems,” *Computing in Science & Engineering*, pp. 1–9, 2024. DOI: 10.1109/MCSE.2024.3362586.
 - [26] “TOP500 Nov 2023.” (Nov. 2023), [Online]. Available: <https://top500.org/lists/top500/2023/11/>.
 - [27] D. Alvarez, “JUWELS Cluster and Booster: Exascale pathfinder with Modular Supercomputing Architecture at Juelich Supercomputing Centre,” *Journal of large-scale research facilities JLSRF*, vol. 7, A183, Oct. 2021, ISSN: 2364-091X. DOI: 10.17815/jlsrf-7-183.
 - [28] S. Graf and O. Mextorf, “JUST: Large-scale multi-tier storage infrastructure at the Jülich Supercomputing Centre,” *Journal of large-scale research facilities JLSRF*, vol. 7, A180–A180, 2021. DOI: 10.17815/jlsrf-5-172.
 - [29] K. Hoste, J. Timmerman, A. Georges, and S. De Weirtdt, “EasyBuild: Building software with ease,” in *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, IEEE, 2012, pp. 572–582. DOI: 10.1109/SC.Companion.2012.81.
 - [30] K. Haghighi Mood, A. Herten, and S. Achilles, *Jupiter benchmark suite: Amber*, 2024. DOI: 10.5281/ZENODO.12737559.
 - [31] T. Hater, A. Herten, and J. Badwaik, *Jupiter benchmark suite: Arbor*, 2024. DOI: 10.5281/ZENODO.12737757.
 - [32] E. B. Gregory, A. Herten, and S. Achilles, *Jupiter benchmark suite: Chroma-lqcd*, 2024. DOI: 10.5281/ZENODO.12737626.
 - [33] J. H. Meinke, A. Strube, A. Herten, S. Achilles, and K. Haghighi Mood, *Jupiter benchmark suite: Gromacs*, 2024. DOI: 10.5281/ZENODO.12787776.
 - [34] C. I. Meyer, N. Nobre Wittwer, M. Römmer, *et al.*, *Jupiter benchmark suite: Icon*, 2024. DOI: 10.5281/ZENODO.12787963.
 - [35] D. Willsch, H. De Raedt, A. Herten, and S. Achilles, *Jupiter benchmark suite: Juqcs*, 2024. DOI: 10.5281/ZENODO.12788076.
 - [36] C. Witzler, J. Windgassen, M. Bode, A. Herten, and S. Achilles, *Jupiter benchmark suite: Nekrs*, 2024. DOI: 10.5281/ZENODO.12788254.
 - [37] A. Gonzalez-Nicolas, D. Caviedes-Voullième, A. Strube, *et al.*, *Jupiter benchmark suite: Parflow*, 2024. DOI: 10.5281/ZENODO.12788364.
 - [38] U. Sinha, J. Badwaik, A. Herten, and S. Achilles, *Jupiter benchmark suite: Picongpu*, 2024. DOI: 10.5281/ZENODO.12788381.
 - [39] K. Haghighi Mood, A. Herten, and S. Achilles, *Jupiter benchmark suite: Quantum espresso*, 2024. DOI: 10.5281/ZENODO.12788398.
 - [40] J. Badwaik, A. Smolenko, A. Herten, S. Achilles, and T. Breuer, *Jupiter benchmark suite: Soma*, 2024. DOI: 10.5281/ZENODO.12788506.
 - [41] M. Cherti, S. Achilles, A. Herten, C. John, and J. Badwaik, *Jupiter benchmark suite: Mmoclclip*, 2024. DOI: 10.5281/ZENODO.12788226.
 - [42] C. John, S. Kesselheim, C. Penke, *et al.*, *Jupiter benchmark suite: Megatron-lm*, 2024. DOI: 10.5281/ZENODO.12788115.
 - [43] C. John, A. Strube, S. Kesselheim, A. Herten, S. Achilles, and J. Ebert, *Jupiter benchmark suite: Resnet*, 2024. DOI: 10.5281/ZENODO.12788436.
 - [44] K. Szabo, A. Herten, S. Achilles, J. Badwaik, and S. Nassyr, *Jupiter benchmark suite: Dynqcd*, 2024. DOI: 10.5281/ZENODO.12737680.
 - [45] E. Behle, A. Herten, and S. Achilles, *Jupiter benchmark suite: Nastja*, 2024. DOI: 10.5281/ZENODO.12788527.
 - [46] D. Alvarez, A. Strube, A. Herten, and S. Achilles, *Jupiter benchmark suite: Graph500*, 2024. DOI: 10.5281/ZENODO.12788553.
 - [47] J.-O. Mirus, T. Breuer, S. Achilles, and A. Herten, *Jupiter benchmark suite: Hpcg*, 2024. DOI: 10.5281/ZENODO.12788585.
 - [48] S. Achilles, A. Herten, and K. Haghighi Mood, *Jupiter benchmark suite: Hpl*, 2024. DOI: 10.5281/ZENODO.12788612.
 - [49] Y. Müller, M. Holicki, A. Herten, and S. Achilles, *Jupiter benchmark suite: Linktest*, 2024. DOI: 10.5281/ZENODO.12788705.
 - [50] T. Breuer, A. Herten, and S. Achilles, *Jupiter benchmark suite: Osu micro-benchmarks*, 2024. DOI: 10.5281/ZENODO.12788751.

- [51] S. Achilles, T. Breuer, K. Thust, *et al.*, *Jupiter benchmark suite: Stream*, 2024. DOI: 10.5281/ZENODO.12788781.
- [52] S. Achilles, A. Herten, and K. Haghighi Mood, *Jupiter benchmark suite: Stream*, 2024. DOI: 10.5281/ZENODO.12788801.
- [53] R. Salomon-Ferrer, A. W. Götz, D. Poole, S. Le Grand, and R. C. Walker, "Routine microsecond molecular dynamics simulations with AMBER on GPUs. 2. explicit solvent particle mesh Ewald," *Journal of Chemical Theory and Computation*, vol. 9, no. 9, pp. 3878–3888, 2013. DOI: 10.1021/ct400314y.
- [54] D. A. Case, H. M. Aktulga, K. Belfon, *et al.*, "Amber-tools," *Journal of Chemical Information and Modeling*, vol. 63, no. 20, pp. 6183–6191, 2023. DOI: 10.1021/acs.jcim.3c01153.
- [55] D. Cerutti. "Amber20 benchmarks." (2020), [Online]. Available: https://ambermd.org/Amber20_Benchmark_Suite.tar.gz.
- [56] S. F. Ashby and R. D. Falgout, "A parallel multi-grid preconditioned conjugate gradient algorithm for groundwater flow simulations," *Nuclear science and engineering*, vol. 124, no. 1, pp. 145–159, 1996. DOI: 10.13182/NSE96-A24230.
- [57] J. E. Jones and C. S. Woodward, "Newton–Krylov-multigrid solvers for large-scale, highly heterogeneous, variably saturated flow problems," *Advances in water resources*, vol. 24, no. 7, pp. 763–774, 2001. DOI: 10.1016/S0309-1708(00)00075-0.
- [58] J. Hokkanen, S. Kollet, J. Kraus, A. Herten, M. Hrynian, and D. Pleiter, "Leveraging HPC accelerator architectures with modern techniques — hydrologic modeling on GPUs with ParFlow," *Computational Geosciences*, May 2021. DOI: 10.1007/s10596-021-10051-4.
- [59] L. Schneider and M. Müller, "Multi-architecture monte-carlo (mc) simulation of soft coarse-grained polymeric materials: Soft coarse grained monte-carlo acceleration (soma)," *Computer Physics Communications*, vol. 235, pp. 463–476, 2019, ISSN: 0010-4655. DOI: <https://doi.org/10.1016/j.cpc.2018.08.011>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0010465518303072>.
- [60] K. C. Daoulas and M. Müller, "Single chain in mean field simulations: Quasi-instantaneous field approximation and quantitative comparison with Monte Carlo simulations," *The Journal of Chemical Physics*, vol. 125, no. 18, 2006. DOI: 10.1063/1.2364506.
- [61] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.
- [62] S. Borsanyi, Z. Fodor, J. N. Guenther, *et al.*, "Leading hadronic contribution to the muon magnetic moment from lattice qed," *Nature*, vol. 593, no. 7857, pp. 51–55, Apr. 2021, ISSN: 1476-4687. DOI: 10.1038/s41586-021-03418-1. [Online]. Available: <http://dx.doi.org/10.1038/s41586-021-03418-1>.
- [63] H. J. C. Berendsen, D. van der Spoel, and R. van Drunen, "GROMACS: A message-passing parallel molecular dynamics implementation," *Computer Physics Communications*, vol. 91, no. 1, pp. 43–56, Sep. 1995, ISSN: 0010-4655. DOI: 10.1016/0010-4655(95)00042-E.
- [64] S. Páll, M. J. Abraham, C. Kutzner, B. Hess, and E. Lindahl, "Tackling exascale software challenges in molecular dynamics simulations with GROMACS," in *Solving Software Challenges for Exascale*, ser. Lecture Notes in Computer Science 8759, S. Markidis and E. Laure, Eds., Springer International Publishing, Apr. 2014, pp. 3–27, ISBN: 978-3-319-15976-8. DOI: 10.1007/978-3-319-15976-8_1.
- [65] M. J. Abraham, T. Murtola, R. Schulz, *et al.*, "GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers," *SoftwareX*, vol. 1–2, pp. 19–25, Sep. 2015, ISSN: 2352-7110. DOI: 10.1016/j.softx.2015.06.001.
- [66] S. Páll, A. Zhmurov, P. Bauer, *et al.*, "Heterogeneous parallelization and acceleration of molecular dynamics simulations in GROMACS," *The Journal of Chemical Physics*, vol. 153, no. 13, p. 134 110, Oct. 2020, ISSN: 0021-9606. DOI: 10.1063/5.0018516.
- [67] G. Zängl, D. Reinert, P. Rípodas, and M. Baldauf, "The ICON (ICOsahedral non-hydrostatic) modelling framework of DWD and MPI-M: Description of the non-hydrostatic dynamical core," *Quarterly Journal of the Royal Meteorological Society*, vol. 141, no. 687, pp. 563–579, 2015. DOI: 10.1002/qj.2378.
- [68] M. Giorgetta, R. Brokopf, T. Crueger, *et al.*, "ICON-A, the atmosphere component of the ICON earth system model. part i: Model description," *Journal of Advances in Modeling Earth Systems*, vol. 10, Jun. 2018. DOI: 10.1029/2017MS001242.
- [69] P. Korn, N. Brüggemann, J. H. Jungclaus, *et al.*, "ICON-O: The ocean component of the ICON earth system model—global simulation characteristics and local telescoping capability," *Journal of Advances in Modeling Earth Systems*, vol. 14, no. 10, 2022. DOI: 10.1029/2021MS002952.
- [70] M. A. Giorgetta, W. Sawyer, X. Lapillonne, *et al.*, "The ICON-A model for direct qbo simulations on gpus (version icon-cscs:baf28a514)," *Geoscientific Model Development*, vol. 15, no. 18, pp. 6985–7016, 2022. DOI: 10.5194/gmd-15-6985-2022.
- [71] C. Hohenegger, P. Korn, L. Linardakis, *et al.*, "ICON-Sapphire: Simulating the components of the earth system and their interactions at kilometer and sub-kilometer scales," *Geoscientific Model Development*, vol. 16, no. 2, pp. 779–811, 2023. DOI: 10.5194/gmd-2022-171.

- [72] B. Stevens, C. Acquistapace, A. Hansen, *et al.*, “The added value of large-eddy and storm-resolving models for simulating clouds and precipitation,” *Journal of the Meteorological Society of Japan. Ser. II*, vol. 98, no. 2, pp. 395–435, 2020. DOI: 10.2151/jmsj.2020-021.
- [73] M. Shoenybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, *Megatron-LM: Training multi-billion parameter language models using model parallelism*, 2020. arXiv: 1909.08053 [cs.CL].
- [74] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, “Attention is all you need,” in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, *et al.*, Eds., vol. 30, Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.
- [75] D. Narayanan, M. Shoenybi, J. Casper, *et al.*, “Efficient large-scale language model training on GPU clusters using Megatron-LM,” in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC ’21, St. Louis, Missouri: Association for Computing Machinery, 2021, ISBN: 9781450384421. DOI: 10.1145/3458817.3476209.
- [76] V. Korthikanti, J. Casper, S. Lym, *et al.*, *Reducing activation recomputation in large transformer models*, 2022. arXiv: 2205.05198 [cs.LG].
- [77] T. Dao, *FlashAttention-2: Faster attention with better parallelism and work partitioning*, 2023. arXiv: 2307.08691 [cs.LG].
- [78] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He, “ZeRO: Memory optimizations toward training trillion parameter models,” in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC ’20, Atlanta, Georgia: IEEE Press, 2020, ISBN: 9781728199986.
- [79] C. M. John, J. Ebert, C. Penke, S. Kesselheim, and A. Herten, “OpenGPT-X – Training Large Language Models on HPC Systems,” ISC High Performance 2023, Hamburg (Germany), 21 May 2023 - 25 May 2023, May 21, 2023. DOI: 10.34732/XDVBLG-SVNDMJ.
- [80] A. Radford, J. W. Kim, C. Hallacy, *et al.*, *Learning transferable visual models from natural language supervision*, 2021. arXiv: 2103.00020 [cs.CV].
- [81] C. Schuhmann, R. Vencu, R. Beaumont, *et al.*, “Laion-400m: Open dataset of clip-filtered 400 million image-text pairs,” *Data-Centric AI Workshop NeurIPS*, arXiv:2111.02114, 2021.
- [82] C. Schuhmann, R. Beaumont, R. Vencu, *et al.*, “LAION-5B: An open large-scale dataset for training next generation image-text models,” in *Thirty-sixth Conference on Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, vol. 35, 2022, pp. 25 278–25 294. [Online]. Available: <https://openreview.net/forum?id=M3Y74vmsMcY>.
- [83] G. Ilharco, M. Wortsman, R. Wightman, *et al.*, *Openclip*, version 0.1, Jul. 2021. DOI: 10.5281/zenodo.5143773.
- [84] M. Cherti, R. Beaumont, R. Wightman, *et al.*, “Reproducible scaling laws for contrastive language-image learning,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 2818–2829.
- [85] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, “High-resolution image synthesis with latent diffusion models,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 10 684–10 695.
- [86] H. Liu, C. Li, Q. Wu, and Y. J. Lee, “Visual instruction tuning,” in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. [Online]. Available: <https://openreview.net/forum?id=w0H2xGHlkW>.
- [87] Q. Sun, Q. Yu, Y. Cui, *et al.*, “Emu: Generative pre-training in multimodality,” in *The Twelfth International Conference on Learning Representations*, 2023.
- [88] P. Giannozzi, S. Baroni, N. Bonini, *et al.*, “QUANTUM ESPRESSO: A modular and open-source software project for quantum simulations of materials,” *Journal of Physics: Condensed Matter*, vol. 21, no. 39, p. 395 502, Sep. 2009. DOI: 10.1088/0953-8984/21/39/395502.
- [89] P. Giannozzi, O. Andreussi, T. Brumme, *et al.*, “Advanced capabilities for materials modelling with Quantum ESPRESSO,” *Journal of Physics: Condensed Matter*, vol. 29, no. 46, p. 465 901, Oct. 2017. DOI: 10.1088/1361-648X/aa8f79.
- [90] P. Giannozzi, O. Basergio, P. Bonfà, *et al.*, “Quantum ESPRESSO toward the exascale,” *The Journal of Chemical Physics*, vol. 152, no. 15, p. 154 105, Apr. 2020, ISSN: 0021-9606. DOI: 10.1063/5.0005082.
- [91] I. Carnimeo, F. Affinito, S. Baroni, *et al.*, “Quantum ESPRESSO: One further step toward the exascale,” *Journal of Chemical Theory and Computation*, vol. 19, no. 20, pp. 6992–7006, 2023. DOI: 10.1021/acs.jctc.3c00249.
- [92] *MaX benchmark repository*, https://gitlab.com/max-centre/benchmarks/-/blob/master/Quantum_Espresso/CP/ZrO2/supercell_11layer/.
- [93] *MaX: Materials at the eXascale. an EU Centre of Excellence for supercomputing applications*, <https://www.max-centre.eu/>, Accessed: 2024-03-14.
- [94] M. Berghoff and I. Kondov, “Non-collective scalable global network based on local communications,” in *2018 IEEE/ACM 9th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (scalA)*, 2018, pp. 25–32. DOI: 10.1109/ScalA.2018.00007.
- [95] F. Graner and J. A. Glazier, “Simulation of biological cell sorting using a two-dimensional extended Potts

- model,” *Phys. Rev. Lett.*, vol. 69, pp. 2013–2016, 13 Sep. 1992. DOI: 10.1103/PhysRevLett.69.2013.
- [96] M. Berghoff, J. Rosenbauer, F. Hoffmann, and A. Schug, “Cells in silico – introducing a high-performance framework for large-scale tissue modeling,” *BMC bioinformatics*, vol. 21, no. 1, pp. 1–21, 2020. DOI: 10.1186/s12859-020-03728-7.
- [97] M. S. Steinberg, “On the mechanism of tissue reconstruction by dissociated cells, I. Population kinetics, differential adhesiveness, and the absence of directed migration,” *Proceedings of the National Academy of Sciences*, vol. 48, no. 9, pp. 1577–1582, 1962. DOI: 10.1073/pnas.48.9.1577.
- [98] N. A. Akar, B. Cumming, V. Karakasis, *et al.*, “Arbor — a morphologically-detailed neural network simulation library for contemporary high-performance computing architectures,” in *2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, 2019, pp. 274–282. DOI: 10.1109/EMPDP.2019.8671560.
- [99] K. Amunts, A. C. Knoll, T. Lippert, *et al.*, “The Human Brain Project—synergy between neuroscience, computing, informatics, and brain-inspired technologies,” *PLoS Biology*, vol. 17, no. 7, e3000344, 2019. DOI: 10.1371/journal.pbio.3000344.
- [100] Y. N. Billeh, B. Cai, S. L. Gratiy, *et al.*, “Systematic integration of structural and functional data into multi-scale models of mouse primary visual cortex,” *Neuron*, vol. 106, no. 3, 388–403.e18, 2020, ISSN: 0896-6273. DOI: <https://doi.org/10.1016/j.neuron.2020.01.040>.
- [101] R. G. Edwards and B. Joo, “The Chroma software system for Lattice QCD,” *Nucl. Phys. B Proc. Suppl.*, vol. 140, G. T. Bodwin, D. K. Sinclair, E. Eichten, *et al.*, Eds., p. 832, 2005. DOI: 10.1016/j.nuclphysbps.2004.11.254.
- [102] *USQCD*, <https://usqcd-software.github.io/>.
- [103] M. A. Clark, R. Babich, K. Barros, R. C. Brower, and C. Rebbi, “Solving Lattice QCD systems of equations using mixed precision solvers on GPUs,” *Comput. Phys. Commun.*, vol. 181, pp. 1517–1528, 2010. DOI: 10.1016/j.cpc.2010.05.002.
- [104] K. De Raedt, K. Michielsen, H. De Raedt, *et al.*, “Massively parallel quantum computer simulator,” *Comput. Phys. Commun.*, vol. 176, p. 121, 2007. DOI: 10.1016/j.cpc.2006.08.007.
- [105] H. De Raedt, F. Jin, D. Willsch, *et al.*, “Massively parallel quantum computer simulator, eleven years later,” *Comput. Phys. Commun.*, vol. 237, pp. 47–61, 2019. DOI: 10.1016/j.cpc.2018.11.005.
- [106] D. Willsch, M. Willsch, F. Jin, K. Michielsen, and H. De Raedt, “GPU-accelerated simulations of quantum annealing and the quantum approximate optimization algorithm,” *Comput. Phys. Commun.*, vol. 278, p. 108411, 2022. DOI: 10.1016/j.cpc.2022.108411.
- [107] F. Arute, K. Arya, R. Babbush, *et al.*, “Quantum supremacy using a programmable superconducting processor,” *Nature*, vol. 574, pp. 505–510, 2019. DOI: 10.1038/s41586-019-1666-5.
- [108] H. De Raedt and D. Willsch. “Jülich universal quantum computer simulator (docker container).” (2021), [Online]. Available: <https://jugit.fz-juelich.de/qip/juqcs-docker>.
- [109] H. De Raedt and D. Willsch. “Jülich universal quantum computer simulator (light version).” (2024), [Online]. Available: <https://jugit.fz-juelich.de/qip/juqcs-light>.
- [110] P. Fischer, S. Kerkemeier, M. Min, *et al.*, “NekRS, a GPU-accelerated spectral element Navier–Stokes solver,” *Parallel Computing*, vol. 114, p. 102982, 2022. DOI: 10.1016/j.parco.2022.102982.
- [111] E. Merzari, S. Hamilton, T. Evans, *et al.*, “Exascale multiphysics nuclear reactor simulations for advanced designs,” in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC ’23, Denver, USA: Association for Computing Machinery, 2023, ISBN: 9798400701092. DOI: 10.1145/3581784.3627038.
- [112] A. T. Patera, “A spectral element method for fluid dynamics: Laminar flow in a channel expansion,” *Journal of Computational Physics*, vol. 54, pp. 468–488, 1984. DOI: 10.1016/0021-9991(84)90128-1.
- [113] S. A. Orszag, “Spectral methods for problems in complex geometries,” *Journal of Computational Physics*, vol. 37, pp. 70–92, 1980. DOI: 10.1016/0021-9991(80)90005-4.
- [114] M. O. Deville, P. F. Fischer, and E. H. Mund, *High-Order Methods for Incompressible Fluid Flow*. Cambridge Monographs on Applied and Computational Mathematics, Cambridge University Press, 2002. DOI: 10.1017/CBO9780511546792.
- [115] D. S. Medina, A. St-Cyr, and T. Warburton, “OCCA: A unified approach to multi-threading languages,” *arXiv preprint arXiv:1403.0968*, 2014.
- [116] Hidden (Blind Review), “Boundary layers in thermal convection are fluctuation-dominated,” *Journal of Fluid Mechanics (submitted)*, 2024.
- [117] Hidden (Blind Review), “A high-scaling in-situ workflow for deciphering boundary layer effects in high-Rayleigh-number convection,” *(submitted)*, 2024.
- [118] M. Bussmann, H. Burau, T. E. Cowan, *et al.*, “Radiative signatures of the relativistic Kelvin-Helmholtz instability,” in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC ’13, Denver, Colorado: ACM, 2013, 5:1–5:12, ISBN: 978-1-4503-2378-9. DOI: 10.1145/2503210.2504564.
- [119] E. Zenker, B. Worpitz, R. Widera, *et al.*, “Alpaka – an abstraction library for parallel kernel acceleration,” in *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2016, pp. 631–640. DOI: 10.1109/IPDPSW.2016.50.
- [120] B. Worpitz, *Investigating performance portability of a highly scalable particle-in-cell simulation code on*

various multi-core architectures, Master Thesis, Sep. 2015. DOI: 10.5281/zenodo.49768.

- [121] H. Burau, R. Widera, W. Hönig, *et al.*, “PIConGPU: A fully relativistic particle-in-cell code for a GPU cluster,” *IEEE Transactions on Plasma Science*, vol. 38, no. 10, pp. 2831–2839, 2010. DOI: 10.1109/TPS.2010.2064310.
- [122] J. Kunkel, G. Markomanolis, J. Bent, and J. Lofstead, *Vi4io/io-500-dev: Zenodo citation release*, version v1.1, Sep. 2018. DOI: 10.5281/zenodo.1422814.
- [123] *LinkTest: Communication API benchmarking tool*, <https://gitlab.jsc.fz-juelich.de/cstao-public/linktest/>, Accessed: 2024-03-22.
- [124] *MaX3 CoE*, <https://www.max-centre.eu/>, Accessed: 2024-08-23.
- [125] *CoEC*, <https://coec-project.eu/>, Accessed: 2024-08-23.