Technische Universität Braunschweig

DIGITAL SCIENCE SUPPORT LAB

HPSF Community Summit

A European Workshop on high-performance software

February 25th- February 27th, 2026 Braunschweig (Germany)

Keynote Speaker:

Hartwig Anzt (Technical University Munic)

HPSF
HIGH PERFORMANCE SOFTWARE FOUNDATION

Hochschule.digital Niedersachsen

Wednesday February 25th

08:00-09:00 — Okerhochhaus PK3.4
Registration & Welcome Reception

09:00-09:30 — Okerhochhaus PK3.4
Opening

09:30-11:30 — Okerhochhaus PK3.4
HPSF & Kokkos Overview

- Introduction to HPSF, general overview
  M. Mayr
- What's new in Kokkos
  D. Arndt, C. Chevalier

11:30-13:00 — Okerhochhaus PK3.4
Lunch Break

13:00-15:00 — Okerhochhaus PK3.4
Kokkos I

- FleCSI: Portable C++ Multiphysics Application Development
  P. Edelmann
- DDC: A Performance Portable Library Abstracting Computation on Discrete Domains
  T. Morvany, T. Padioleau
- Kokkos-FFT
  P. Gannay, Y. Asahi
- Krokkos, an experimental Rust API for Kokkos
  C. Chevalier, R. Clisson

15:00-16:00 — Okerhochhaus PK3.4
Coffee Break

16:00-18:00 — Okerhochhaus PK3.4
Kokkos II

- ReProspect - A framework for reproducible prospecting of CUDA applications
  R. Tomasetti, M. Arnst
- Improving the Efficiency of Kokkos Multi-Dimensional Range Policy for GPUs
  A. Taberner
- IPPL: A Kokkos based Performance Portable Library for Particle-Mesh Methods
  A. Adelmann
- A Massively Parallel Performance Portable Free-Space Spectral Poisson Solver
  S. Mayani, A. Adelmann, A. Cerfon, M. Frey, S. Muralikrishnan, V. Montanaro

18:00-20:30 — Okerhochhaus Aquarium
Poster Session & Networking

Thursday February 26th

08:00-09:30 Okerhochhaus PK3.4
Coffee Break

09:30-11:30 Okerhochhaus PK3.4
Keynote & Trilinos Overview

Keynote: Hardware is changing- Do we need to change the way we design simulation software?
H. Anzt

What's new in Trilinos
M. Mayr

11:30-13:00 Okerhochhaus PK3.4
Lunch Break

13:00-15:00 Okerhochhaus PK3.4
Trilinos & Kokkos III

A User Perspective on Physics-Oriented Block Preconditioning Using Trilinos for Mictrostructure-Resolved Solid-State Battery Simulations
C. P. Schmidt, M. Firmbach, M. Mayr W. A. Wall

Modernizing the 4C linear algebra backend: from Epetra to Tpetra
M. Mayr, M. Firmbach, M. Frank, V. Ivannikov

Trilinos in deal.II
D. Arndt

Kokkos Comm: Performance Portable Communication Interface for Distributed Kokkos Applications
G. Dos Santos, C. Chevalier, C. Pearson, N. Avans

15:00-16:00 Okerhochhaus PK3.4
Coffee Break

16:00-18:00 Okerhochhaus PK3.4
AMReX & WarpX

What's new in AMReX & WarpX
L. Fedeli

Accelerating axion physics with AMReX
M. Buschmann

On the interoperability of AMReX and Kokkos
N. Schild, C. Lalescu, E. Poulsen

19:00-21:00 Nah am Wasser
Dinner

Friday February 27th

08:00-09:30 Okerhochhaus PK3.4
Coffee Break

09:30-11:30 Okerhochhaus PK3.4
HPC ressource management

What's new in Spack
M. Culpo

JUBE: An Environment for systematic benchmarking and scientific workflows
P. Steinbach, T. Breuer, F. Guimaraes, J. Mirus, W. Frings

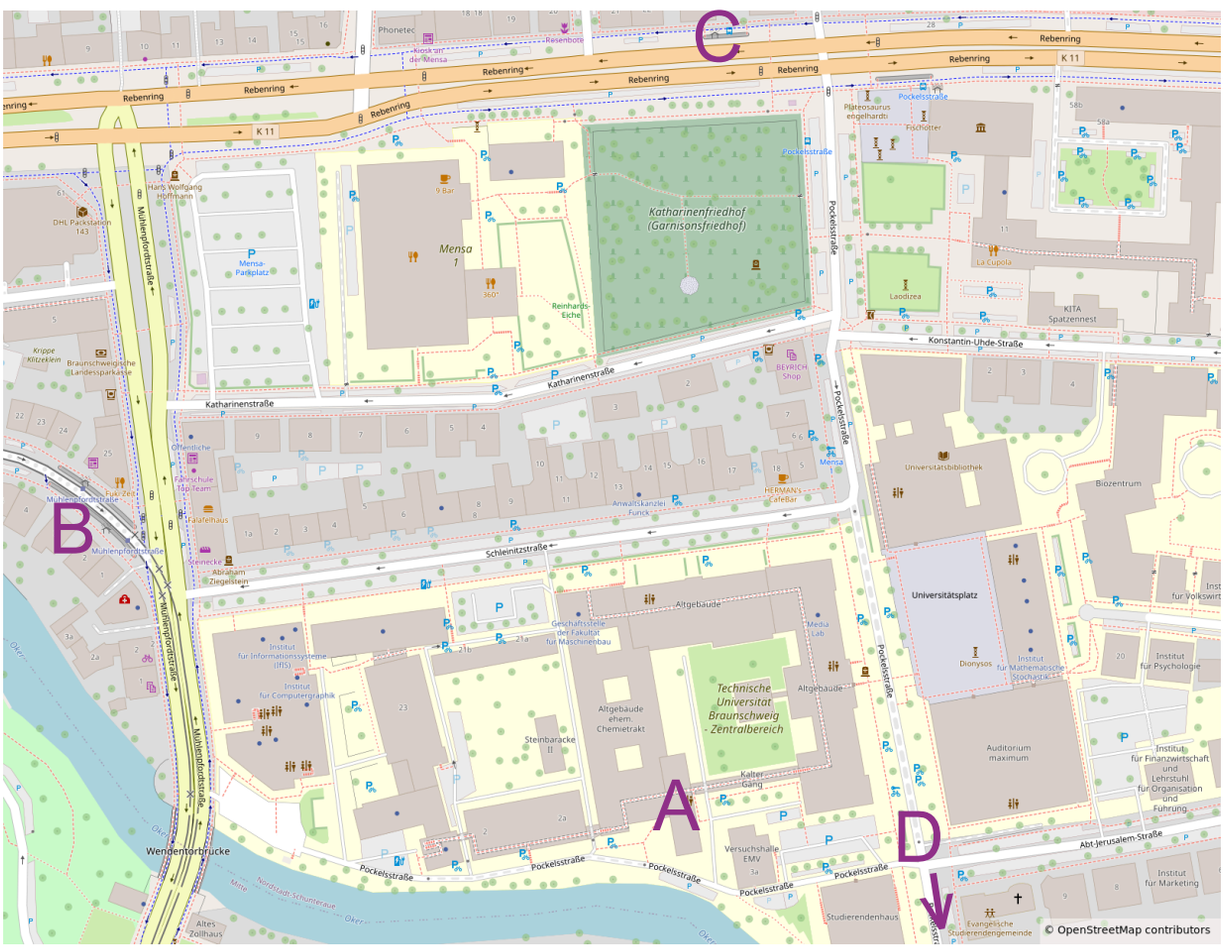
Accelerating HPC Control Plane Development: Deploying OpenCHAMI Test Systems rom Local VMs to Bare Metal
A. Escoubas

11:30-12:00
Closing

12:00-13:30 Okerhochhaus PK3.4
Lunch Break

Main Campus

A: Okerhochhaus
B: Tram Station Mühlenpfordstraße
C: Bus Stop Pockelsstraße
D: To restaurant (by foot)



Public transport options from main Station (Braunschweig Hbf)
Tram:
To station Mühlenpfordstraße (B) with lines
1 (Wenden), 2 (Siegfriedviertel) or 10 (Rühme)
Bus:
To station Pockelsstraße (C) with lines
419 (Amalienplatz > Cyriakusring) or 436 (Flughafen)

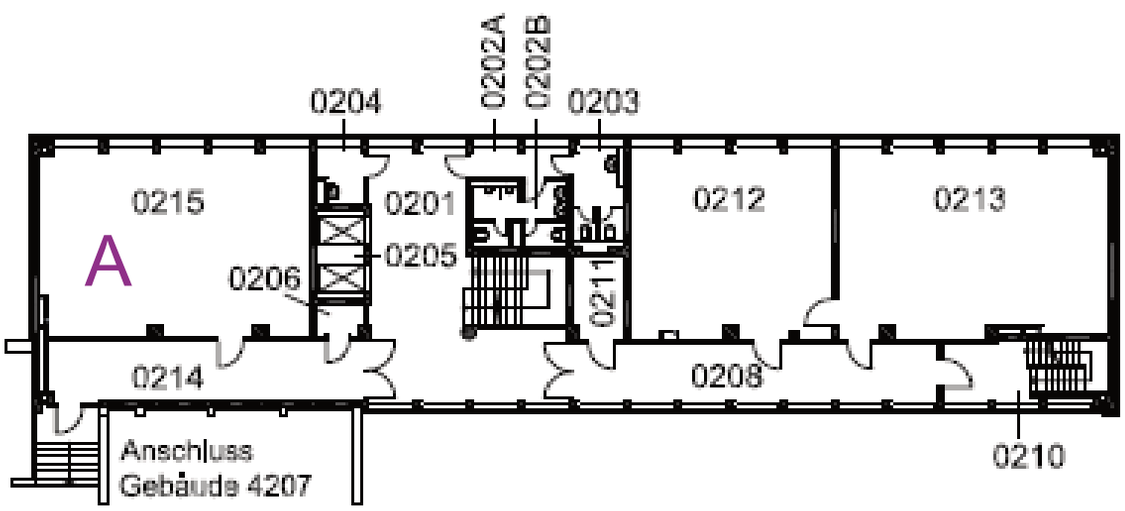
HPSF
HIGH PERFORMANCE SOFTWARE FOUNDATION

Hochschule.digital Niedersachsen

Okerhochhaus (PK3)

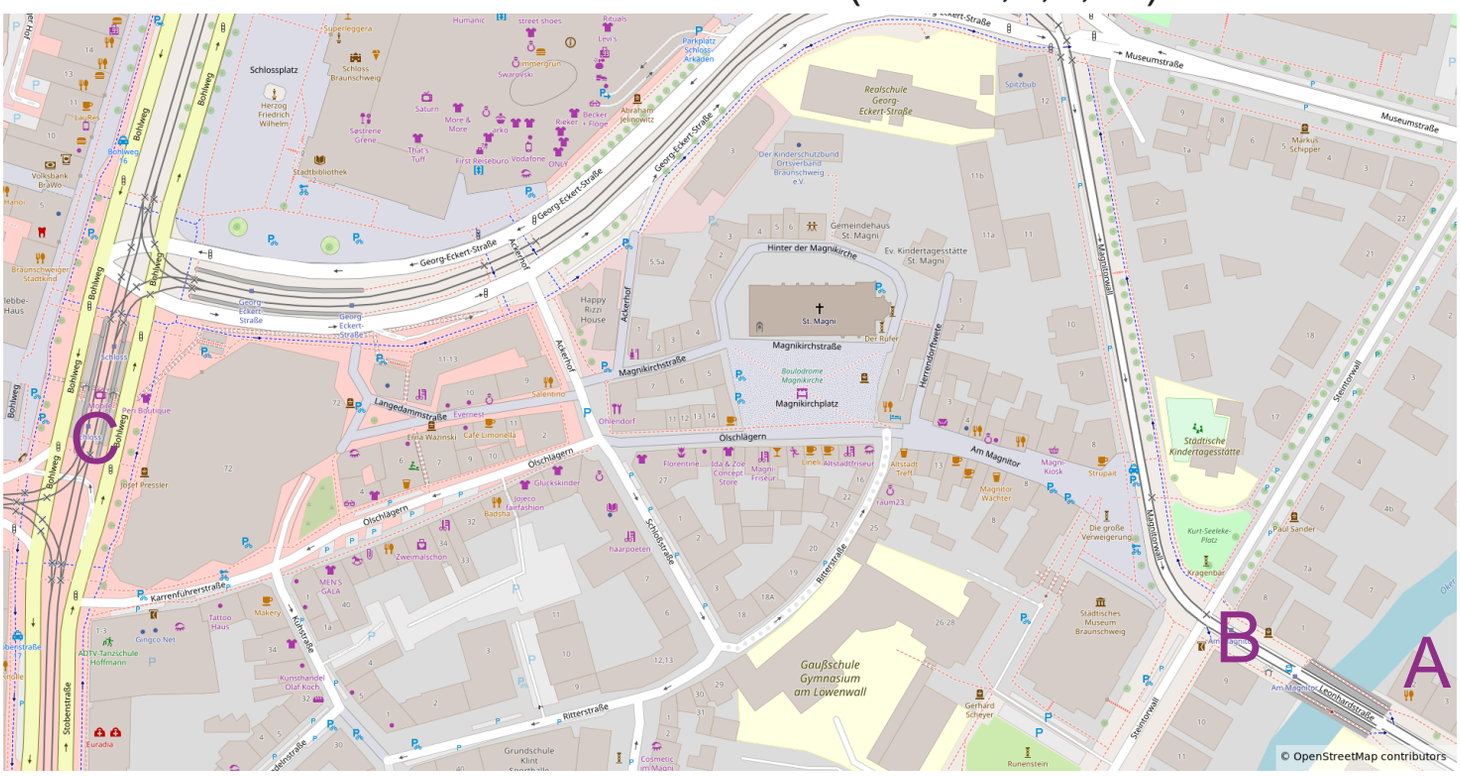
A: Room PK3.4 (2nd floor)

Aquarium: ground floor foyer



Dinner

26.02.2026 19:00
NAW - Nah am Wasser
Leonhardstraße 2 (A)
B: Tram station "Am Magnitor" (line 5)
C: Tram station "Schloss" (lines 1,2,5,10)



The restaraunt is close to the city center and many of the hotels.

Public transport:
Take lines 1,2 or 10 from "Mühlenpfordstraße" to "Schloss" (C)
Take line 5 to "Am Magnitor" (B) going from the opposite side

Walking:
starting from Campus (D) go straight down Pockelsstraße across the Oker
walk straight ahead until reaching "Staatstheater Braunschweig"
continue down "Magnitorwall" leading to B

A group will walk from Campus after the last session if you want to join.

Wed, 25.02.2026 09:30-10:30
Introduction to HPSF, general overview

Matthias Mayr (Universität der Bundeswehr München)

This talk gives a general overview on the high performance software foundation

Wed, 25.02.2026 10:30-11:30
What's new in Kokkos

Daniel Arndt (Oak Ridge National Laboratory), Cédric Chevalier (CEA)

Wed, 25.02.2026 13:00-13:30
FleCSI: Portable C++ Multiphysics Application Development

Philipp Edelmann (Los Alamos National Laboratory)

Developing multiphysics applications that run on the supercomputers of the exascale era is a daunting task. The varying workload in different regions of a problem can make load balancing difficult, especially when simultaneously handling GPU and CPU resources. Task-based parallelism is a promising way out of this dilemma but it introduces significant complexity for the application developer and has not seen widespread adoption for that reason

I will introduce FleCSI, which is an open-source C++ framework developed at Los Alamos National Laboratory as part of the Advanced Simulation and Computing Program of the U.S. Department of Energy to mitigate these issues. It provides an abstraction layer aimed at multiphysics application developers to make it easy to use different task-based parallelism frameworks. This is achieved by expressing data access permissions in the types of the function arguments. FleCSI currently supports Legion and HPX, while also providing an MPI backend as a fallback. Data movement to and from host/device memory spaces as well as halo updates is handled transparently for the user. Performance portability is achieved through Kokkos. It is possible to run CPU and GPU tasks simultaneously and have the data moved as needed. FleCSI offers several base topologies, upon which developers can build so-called specializations for their specific data layout. Among these are N-dimensional arrays, unstructured meshes, tree, and set topologies.

I will give some examples of the FleCSI programming model and show how we build the complex dependencies on various architectures through Spack. I will give an overview of several applications including the HARD code, an open-source radiation-hydrodynamics code on a structured mesh, and Moya, a low energy density ALE (Arbitrary Lagrangian Eulerian) code on an unstructured mesh. I will show benchmark results obtained on the AMD MI300A machines at Lawrence Livermore National Laboratory.

Wed, 25.02.2026 13:30-14:00
DDC: A Performance Portable Library Abstracting Computation on Discrete Domains

Trévis Morvany (CEA), Thomas Padioleau (CEA)

The Discrete Domain Computation (DDC) library is a modern C++ library that aims to offer to the C++ world an equivalent to the xarray.DataArray Python environment. The Xarray library introduces labeled multidimensional arrays, enabling more intuitive data manipulation by associating dimensions with user-provided names rather than relying on positional indexing. This approach simplifies indexing, slicing, and broadcasting while reducing common indexing errors. Inspired by these ideas, DDC extends the Kokkos library by providing zero-overhead dimension labeling for multidimensional arrays along with performance-portable multidimensional algorithms. This labeling mechanism enables compile-time detection of indexing and slicing errors, ensuring safer and more expressive array operations in C++. In this presentation, we will introduce the core concepts of DDC and demonstrate its usage through a simple example that highlights its key features.

Wed, 25.02.2026 14:00-14:30
Kokkos-FFT

Paul Gannay (CEA), Yuuichi Asahi

Kokkos-FFT is a library which started development in December 2023, as a part of the CExA project, and is now an official part of the Kokkos project. Its initial goals were to offer access to a Kokkos-aware performance-driven and portable FFT, with an API as simple as possible. It has been done by building an API inspired by NumPy FFT, which dispatch work to existing FFT libs (such as cufft, hipfft, FFTW, or onemkl), choosing the appropriate one depending on the current Kokkos backend. This API is designed to be safe to use, with as much compile-time and runtime checks as possible. Kokkos-FFT takes advantage of Kokkos to offer support for multi dimensional arrays with up to 8 dimensions. It also supports real to real, real to complex and complex to real transformations.

Kokkos-FFT is thoroughly tested and offers an extensive documentation, it is also easy to build, most of the compilation flags coming directly from the Kokkos installation you link against. It recently reached an important milestone with the release of the 1.0 version, making it production ready.

This presentation will serve to:
- showcase the API, its ease of use, and the way it integrates in a Kokkos project,
- present performances results,
- present ongoing work regarding the distribution of FFT across several nodes using MPI and other collective communication libraries.

Wed, 25.02.2026 14:30-15:00

Krokkos, an experimental Rust API for Kokkos

Cédric Chevalier (CEA), Raphael Clisson

In this presentation, we'll introduce our work on Krokkos, an experimental library designed to bridge the gap between Kokkos and Rust.

Rust, a modern systems programming language, excels in multi-threaded and memory-safe applications. Its ownership model guarantees that data races are caught at compile time, making it an attractive choice for scientific software development on shared-memory architectures. However, Rust's ecosystem still lags behind when it comes to vendor-agnostic GPU support for general-purpose computations. Most of the available GPU backends are either experimental, tied to a single vendor, or require cumbersome unsafe wrappers that erode Rust's safety guarantees.

Kokkos, on the other hand, is a well-established solution for shared-memory parallel programming across CPU and accelerators. It achieves performance portability by providing an abstract way to express loops, automatically generating tailored kernels for the chosen backend—whether that be a multi-core CPU or an GPU. A key feature of Kokkos is its custom data structures, Kokkos::Views, which encapsulate multi-dimensional arrays with a layout that adapts to the execution platform. This abstraction allows developers to write code once and run it efficiently on a wide range of hardware without manual tuning.

Our project's goal is to enable interoperability between Rust applications and Kokkos kernels. One initial use case involves creating a hybrid application that uses Rust for the general architecture—handling I/O, orchestration, and high-level logic—but delegates compute-intensive regions to Kokkos routines in order to leverage GPU hardware. Another use case demonstrates interoperability in the opposite direction: a C++ application built on Kokkos calls Rust routines to populate or manipulate Kokkos data structures, thereby taking advantage of Rust's safety and expressive type system for parts of the codebase that are less compute intensive (e.g. IO, interactions).

The primary technical challenge in both scenarios is interfacing Rust and C++, especially given Kokkos' extensive use of templates. Templates in C++ allow for compile-time specialisation and zero-overhead abstractions, but they also mean that the generated binary contains many distinct instantiations of a single generic function. Rust's foreign-function interface (FFI) traditionally relies on C-style ABI, which does not natively support templated types or overloaded functions. Consequently, we must devise a strategy that preserves Kokkos' strongly typed data structures and compile-time layout selection while exposing them safely to Rust.

To better align with Kokkos' logic and strongly typed data structures, which allow for memory representation selection at compile time, we've opted for a direct wrapper between Rust and C++. We have extended interoperability libraries such as the cxx crate to work with Kokkos. The cxx crate provides a safe, type-checked bridge between Rust and C++ by generating bindings that respect ownership semantics. By leveraging this crate, we can expose Kokkos::Views and kernel launch functions to Rust without resorting to raw pointers or manual memory management. The wrapper also handles the translation of Kokkos' execution policies into Rust enums, ensuring that the user can select the desired backend in a type-safe manner.

An alternative solution would have been to use a lingua franca with a stable ABI like C. However, this approach involves fully opaque data exchanges: the Rust side would receive a "void*" pointer and would have to perform runtime checks or casts to interpret the data correctly. Such opaque exchanges can degrade performance because they force the compiler to generate generic code that cannot be optimised as aggressively, and they also reintroduce potential safety pitfalls that Rust is designed to avoid.

We'll spend time explaining the design of the translated Kokkos API in Rust, particularly its memory safety features. Our goal is to create a Rust API that doesn't expose unsafe code to users. For example, we wrap Kokkos::Views in a struct that implements Rust's Deref and Index traits, allowing idiomatic indexing while guaranteeing that the underlying memory remains valid for the lifetime of the view.

We'll then delve into the limitations of our work using more advanced concepts like subviews. Subviews allow a user to create a view into a subset of an existing Kokkos::View, which is essential for algorithms that operate on slices or blocks of data. In C++, subviews are implemented as templated types that inherit from the parent view, leading to a combinatorial explosion of instantiations. In Rust, however, function overloading is disallowed, and generic functions must be monomorphized at compile time. This makes it difficult to provide a clean, ergonomic API for subviews without generating an unmanageable number of functions. We discuss potential workarounds, such as using trait objects or a macro-based approach to generate the necessary boilerplate, and we evaluate their trade-offs in terms of compile time and binary size.

This project is aiming at integrating the HPSF Kokkos project, adding portable GPU support for Rust. By contributing our wrapper back to the upstream repository, we can broadly expose Rust bindings, thereby paving the way for wider adoption of a more productive and secure language that is compatible with modern HPC hardware. This integration will also allow Rust developers to benefit from Kokkos' mature performance tuning tools, such as profiling and auto-tuning, while maintaining the safety guarantees that Rust provides.

Wed, 25.02.2026 16:00-16:30

ReProspect - A framework for reproducible prospecting of CUDA applications

Romin Tomasetti (GNOI), Maarten Arnst (GNOI)

ReProspect is a Python framework designed to support reproducible prospecting of CUDA code - that is, the systematic analysis of CUDA-based libraries and software components through API tracing, kernel profiling, and binary analysis.

ReProspect builds on NVIDIA tools: Nsight Systems, Nsight Compute, and the CUDA binary utilities. It streamlines data collection and extraction using these tools, and it complements them with new functionalities for a fully programmatic analysis of these data, thus making it possible to encapsulate the entire prospecting analysis in a single Python script.

ReProspect provides a practical foundation for developing novel use cases of CUDA code prospecting. It supports collaborative code development by enabling developers to share concise, reproducible analyses that motivate design decisions and help reviewers grasp the impact of proposed changes. It also enables new types of tests that go beyond traditional output-correctness validation in CI/CD pipelines, such as validating the presence of instruction patterns in binaries or confirming expected API call sequences for key library functionalities. Additionally, ReProspect can act as a framework for structuring research artifacts and documenting analyses, enabling others to reproduce the work and build upon it more effectively.

The talk will expose the needs that ReProspect addresses, how the community would benefit from using it, and demonstrate it with concrete case studies inspired by contributions to Kokkos.

Wed, 25.02.2026 16:30-17:00

Improving the Efficiency of Kokkos Multi-Dimensional Range Policy for GPUs

Adrien Taberner (CEA)

Kokkos MDRangePolicy provides a high-level abstraction for iterating over multi-dimensional index spaces. Used with parallel_for and parallel_reduce constructs, it enables computations over N-dimensional spaces (up to 6 dimensions). MDRangePolicy is the most intuitive and commonly used approach for iterating over multi-dimensional arrays and implementing stencil computations in scientific applications. As Kokkos adoption continues to grow, optimizing this core functionality directly benefits a large portion of the user community.

This presentation covers the current performance limitations of MDRangePolicy and explores the default tiling strategies employed within Kokkos for device backends (CUDA, HIP, etc.). Our investigation identified several areas for improvement: suboptimal default block sizes, complicated code paths, and excessive register pressure, which lead to occupancy limitations on GPUs. We present ongoing work to enhance the MDRangePolicy implementation, focusing on reducing register pressure, optimizing default block sizes, and improving overall GPU performance.

We will present profiling reports and benchmark results comparing current and improved implementations across various GPU architectures, demonstrating measurable performance gains (from 1.1x to 2x speedup). Importantly, all improvements maintain full compatibility, so existing user code requires no modifications to benefit from the enhanced performance. Beyond performance improvements, this work has provided opportunities for code refactoring, simplification, and modernization of the Kokkos codebase.

Finally, we discuss the tradeoffs between maintaining high-level portable abstractions and addressing low-level performance concerns.

Wed, 25.02.2026 17:00-17:30

IPPL: A Kokkos based Performance Portable Library for Particle-Mesh Methods

Andreas Adelmann (Paul Scherrer Institute & ETH Zürich)

Particle–mesh methods such as Particle-in-Cell (PIC) remain central to plasma, beam, and astrophysical simulation. We present the current state of the IPPL (Independent Parallel Particle Layer) library, which provides performance portable and dimension independent building blocks for scientific simulations requiring particle-mesh methods. IPPL makes use of Kokkos, HeFFTe, and MPI (Message Passing Interface) to deliver a portable, massively parallel toolkit supporting simulations in one to six dimensions, mixed precision, and asynchronous execution in different execution spaces (e.g. CPUs and GPUs).

IPPL is very well suited for research on novel numerical solvers, and as showcases we elaborate our recent work on a novel spectrally accurate free-space Poisson solver [1] and IPPL-based mini-apps for kinetic plasma simulations [2]. We achieve high particle throughput on GPUs while maintaining scaling across multiple nodes. The mini-apps also help us to identify the dominant costs in particle migration, halo exchange, and field solvers.

[1] S Mayani et al., A massively parallel performance portable free-space spectral Poisson solver,
ACM Transactions on Mathematical Software (51)3, 2025, https://dl.acm.org/doi/10.1145/3748815
[2] S Muralikrishnan et al., Scaling and performance portability of the particle-in-cell scheme for plasma physics applications through mini-apps targeting exascale architectures, SIAM Parallel Processing, 2024, https://doi.org/10.1137/1.9781611977967.3 and https://arxiv.org/abs/2205.11052

Wed, 25.02.2026 17:30-18:00

A Massively Parallel Performance Portable Free-Space Spectral Poisson Solver

Sonali Mayani (Paul Scherrer Institute & ETH Zürich), Andreas Adelmann (Paul Scherrer Institute & ETH Zürich), Antoine Cerfon (Type One Energy Group), Matthias Frey (University of St. Andrews), Srinamkrishnan Muralikrishnan (Forschungszentrum Jülich), Veronica Montanaro (ETH Zürich)

This research is in the context of IPPL (Independent Parallel Particle Layer), an open-source C++ framework providing performance-portable, dimension-independent building blocks for particle–mesh simulations, which combine Eulerian field solvers and Lagrangian particle dynamics. IPPL leverages Kokkos for on-node portability, MPI for distributed parallelism, and heFFTe for scalable FFTs, enabling massively parallel particle–mesh methods on modern heterogeneous systems.

Within IPPL, we implement a performance-portable free-space Poisson solver (Mayani et al., 2025) based on the spectrally accurate algorithm proposed by Vico et al. (2016). The commonly used Hockney-Eastwood method (1988) has second-order convergence at best, requiring fine grids to achieve high accuracy. In contrast, the Vico-Greengard-Ferrando (2016) algorithm converges spectrally for sufficiently smooth functions i.e. faster than any fixed order in the number of grid points. This allows the same accuracy to be achieved on coarser grids, significantly reducing the memory footprint. However, due to the mathematical nature of the Vico-Greengard-Ferrando algorithm, a pre-computation of a Green's function on a four-fold grid is required, which is still memory intensive. We remove this bottleneck by exploiting the realness and the symmetry of the Green's function using a Discrete Fourier Transform, implemented in HeFFTe. This reduces the memory of the new solver based on the Vico-Greengard-Ferrando algorithm to be comparable to the standard Hockney-Eastwood for the same grid-size, all while improving the accuracy. We demonstrate scaling on Perlmutter (NERSC) for both CPUs and GPUs. On GPU, the strong scaling efficiency stays above 80% for a problem size of , and above 75% for weak scaling. Additionally, we perform scaling studies on the GH200 nodes at Alps (CSCS) and the AMD MI250X nodes at LUMI (CSC) to showcase portability.

Thu, 26.02.2026 09:30-10:30

Keynote: Hardware is changing - Do we need to change the way we design simulation software?

Hartwig Anzt (Technical University Munich)

The AI boom is reshaping processor design. Hardware vendors now prioritize high throughput for matrix multiplications and dense linear algebra, optimize aggressively for low precision, and integrate specialized units such as tensor cores. Meanwhile, compute performance continues to grow much faster than memory bandwidth, and latency improvements lag behind both. The result is a widening gap between computation and data movement — a shift with profound implications for scientific simulation software.

Traditional simulation codes rely heavily on IEEE double precision and were designed for architectures with more balanced compute and memory characteristics. In an AI-driven hardware landscape, these assumptions are increasingly challenged. Should we embrace mixed precision? Is double precision becoming a luxury? Do we need emulation for higher precision, or decouple memory and arithmetic precision through compression? Should we trade communication for recomputation to reduce data movement?

This talk explores how hardware trends driven by AI affect simulation software design and argues that we must rethink — though not abandon — our numerical and architectural assumptions to remain efficient on emerging platforms.

Thu, 26.02.2026 10:30-11:30

What's new in Trilinos

Matthias Mayr (Universität der Bundeswehr München)

Trilinos is an advanced software framework designed to facilitate the development of high-performance scientific applications. It provides a comprehensive suite of libraries and tools that support a wide range of computational tasks, from linear algebra and optimization to differential equations and mesh generation. Particular emphasis is put on large-scale parallel software and algorithm development ranging from distributed data structures for sparse linear algebra over finite element evaluation and assembly managers all the way to various nonlinear and linear solvers.

With a focus on scalability and efficiency, Trilinos enables researchers and engineers to tackle complex problems across various fields, including engineering, physics, and applied mathematics. Its modular architecture allows users to customize and extend functionalities, ensuring that Trilinos meets the evolving needs of the scientific community.

This presentation will give an overview of Trilinos, its vision and potential use cases and will include updates on recent developments and achievements across Trilinos.

Thu, 26.02.2026 13:00-13:30

A User Perspective on Physics-Oriented Block Preconditioning Using Trilinos for Microstructure-Resolved Solid-State Battery Simulations

Christoph P. Schmidt (Technical University of Munich), Max Firmbach (Universität der Bundeswehr München), Matthias Mayr (Universität der Bundeswehr München), Wolfgang A. Wall (Technical University of Munich)

Solid-state batteries (SSBs) are a promising technology to overcome physicochemical limitations of the currently dominant battery technology, lithium-ion batteries with liquid electrolytes. However, the interaction between solid mechanics and electrochemical phenomena remains an unresolved challenge in these systems. To gain a deeper understanding, microstructure-resolved computational models that incorporate the relevant phenomena of solid mechanics and electrochemistry capture heterogeneities and their influence on local fields and global cell behavior, but yield large, strongly coupled systems of nonlinear partial differential equations. Efficient solution strategies are thus essential to make such simulations tractable on HPC platforms.

We first demonstrate the underlying physics relevant to an electro-chemo-mechanically coupled SSB model [1]. Then, we outline how the open-source multiphysics software framework 4C [2] can be used to efficiently solve microstructure-resolved SSB models by leveraging physics-oriented block preconditioning techniques implemented using the Trilinos [3] packages Teko, MueLu, and Ifpack, and the GMRES method adopted as linear solver. The preconditioning of the monolithic system of linear equations exploits the system's inherent block structure by dividing it into blocks corresponding to physical fields or areas with similar physical properties. This block structure is then preserved during preconditioning, enabling a tailored preconditioner setup for each block. We recently replaced an in-house block preconditioning implementation with Teko to reduce manual maintenance efforts and benefit from community developments, e.g., the transition to Tpetra for heterogeneous hardware architectures. Finally, we also briefly discuss issues we encountered during this process and suggest additional capabilities that might be helpful from a user perspective.

References

[1] Schmidt et. al. A three-dimensional finite element formulation coupling electrochemistry and solid mechanics on resolved microstructures of all-solid-state lithium-ion batteries, Comput. Method Appl. M. 417, 116468, 2023.
[2] 4C: A Comprehensive Multiphysics Simulation Framework, https://www.4c-multiphysics.org, 2026.
[3] M. Mayr et. al. Trilinos: Enabling Scientific Computing Across Diverse Hardware Architectures at Scale, arXiv:2503.08126, 2025.

Thu, 26.02.2026 13:30-14:00

Modernizing the 4C linear algebra backend: from Epetra to Tpetra

Matthias Mayr (Universität der Bundeswehr München), Max Firmbach (Universität der Bundeswehr München), Martin Frank, Vladimir Ivannikov

The 4C (Comprehensive Computational Community Code) multiphysics simulation framework has been developed to address complex physical phenomena across various scientific and engineering domains. From its inception, 4C has relied on the Trilinos project, an open-source software library for scalable numerical computations, as its backbone for sparse linear algebra and MPI-parallel computing. This integration enhances 4C's computational capabilities and, more importantly, allows the 4C developers to focus on their core research interest: the numerical modeling of multiphysics systems. At the same time, ongoing developments within Trilinos necessitate continuous adaptation and maintenance of the 4C code base.

A major recent change in Trilinos has been the deprecation and removal of the long-standing Epetra linear algebra package. Its successor, Tpetra, introduces templating over scalar and ordinal types and, crucially, integrates tightly with the Kokkos ecosystem to enable performance portability across diverse and heterogeneous hardware architectures.

In this presentation, we report on our experience migrating a large, mature multiphysics code base from Epetra to Tpetra. We describe our strategy of encapsulating Trilinos data structures behind dedicated abstraction layers to limit their scope within the application code. Achievements, encountered challenges, and design trade-offs arising from both 4C- and Trilinos-specific software patterns are discussed, with the aim of providing guidance and practical insights for other Trilinos-based application developers facing similar transitions.

Thu, 26.02.2026 14:00-15:30

Trilinos in deal.II

Daniel Arndt (Oak Ridge National Laboratory)

deal.II has been using Trilinos for linear algebra for more than 15 years. This talk will give a brief overview over the history through that time span and highlight issues with moving from Epetra to Tpetra.

Thu, 26.02.2026 14:30-15:00

Kokkos Comm: Performance Portable Communication Interface for Distributed Kokkos Applications

Gabriel Dos Santos (CEA), Cédric Chevalier (CEA), Carl Pearson (Sandia National Laboratories), Nicole Avanse (Tennessee Technological University)

Kokkos Comm is a lightweight C++ library providing performance-portable explicit communication primitives for distributed Kokkos applications. It aims to eliminate code duplication across the Kokkos ecosystem by centralizing solutions to common pain points. Kokkos Comm addresses critical integration challenges between the Kokkos execution model and distributed memory programming by automatically handling GPU awareness, non-contiguous data marshalling, and view lifetime management. It features a minimal asynchronous API that exposes a streamlined subset of the usual point-to-point and collective operations, while preserving flexibility and abstracting backend-specific complexity. The design is centered around simple, composable, and extensible interfaces, ensuring near-zero overhead compared to hand-rolled solutions. Currently, MPI and NCCL backends are supported, with other GPU-oriented (RCCL, oneCLL) and NIC-oriented (OFI, UCX, Portals) backends being explored. Built with C++20, Kokkos Comm maintains performance portability across complex heterogeneous systems (multi-GPU, multi-NIC) while serving as a research platform for advanced communication patterns, parallel programming models and standardization efforts. Kokkos Comm's philosophy is simple: easy to use, hard to misuse.

Thu, 26.02.2026 16:00-17:00

What's new in AMReX & WarpX

Luca Fedeli (CEA)

Thu, 26.02.2026 17:00-17:30

Accelerating axion physics with AMReX

Malte Buschmann (DESY)

Approximately 70% of the matter content of our Universe consists of dark matter, yet its fundamental nature remains unknown. Among the many proposed candidates, the axion stands out as particularly compelling because it not only provides a viable explanation for dark matter but also offers an elegant solution to the strong CP problem, one of the most persistent puzzles in the Standard Model of particle physics. As a result, a broad experimental program has emerged worldwide to search for axions using a variety of detection strategies.

A major challenge faced by many of these experiments is that the axion mass is not known. Because detection relies on resonant enhancement, experiments must scan over a wide range of possible masses, making searches time-consuming and costly. Reliable theoretical predictions of the axion mass would dramatically improve the efficiency of these efforts. However, obtaining such predictions requires understanding the highly non-linear dynamics of the axion field in the early Universe. This regime can only be studied through large-scale numerical simulations of coupled scalar fields.

Conceptually, these simulations are simple and only involve solving classical field equations on a three-dimensional mesh. In practice, they are extremely demanding. The formation of topological defects known as axion strings introduces a severe separation of physical scales. Accurately resolving string cores while simultaneously capturing large cosmological volumes requires enormous dynamic range. Uniform-grid simulations have reached trillions of cells, yet still struggle to capture the full evolution with controlled systematics.

To address this challenge, we employ adaptive mesh refinement (AMR), which dynamically concentrates resolution only where it is required. We developed sledgehamr, a simulation framework built on AMReX, to make large-scale scalar-field simulations with AMR efficient and accessible on both GPU and CPU systems. The code provides a flexible and modular infrastructure originally designed for axion string networks but readily extensible to other problems, such as gravitational waves sourced by scalar-field dynamics.

Using this framework, we performed the largest three-dimensional simulation of axion strings to date on the NERSC Perlmutter system (see https://tinyurl.com/AxionStringsAMR for an animation). The simulation follows the evolution of the axion field during the first microseconds after the Big Bang, when strings form and dominate the dynamics. It consumed approximately 25 million CPU core-hours and produced roughly 500 TB of output data. By leveraging AMR, we achieved a dynamic range about 3000 times larger than previous studies. Reaching a comparable resolution with a static grid would have required more than 10^{16} cells and would be computationally infeasible.

This unprecedented scale substantially reduces systematic uncertainties in string evolution and axion production, enabling the most robust prediction to date of the axion dark-matter mass and helping to narrow the search space for experiments. More broadly, our results demonstrate that adaptive mesh refinement is essential for tackling the extreme multiscale challenges that arise in early-Universe field dynamics and highlight AMReX as a powerful platform for next-generation HPC cosmology simulations.

Thu, 26.02.2026 17:30-18:00

On the interoperability of AMReX and Kokkos

Nils Schild (IPP), Christian Lalescu (IPP), Emil Poulsen (IPP)

In this talk we report our initial steps on the interoperability between Kokkos and AMReX within an internal codebase. Our goal is to explore how Kokkos capabilities can be integrated into an existing AMReX-based code. In addition to CUDA and HIP, the parallel dispatch provided by Kokkos::parallel_for enables the use of OpenMP for parallel loop constructs. Furthermore, we investigate whether cache local arrays and nested parallelism can enhance the performance of selected kernels. Finally, we consider the interoperability of the Kokkos::View with std::mdspan introduced in C++23. The application of interest is our geometric particle-in-cell code GEMPICX which is currently based on the AMReX framework.

We start by adding the Kokkos library to our build system implemented through CMake, which already facilitates AMReX. Afterwards, we identify overlapping data structures and how AMReX data structures map to Kokkos data structures. Our work is concluded by comparing the runtime of the selected kernels.

Fri, 27.02.2026 09:30-10:30

What's new in Spack

Massimiliano Culpo (CEA)

Fri, 27.02.2026 10:30-11:00

JUBE: An Environment for systematic benchmarking and scientific workflows

Pit Steinbach (Forschungszentrum Jülich, FZJ), Thomas Breuer (FZJ), Filipe Guimaraes (FZJ), Jan-Oliver Mirus (FZJ), Wolfgang Frings (FZJ)

A key aspect of developing research software is testing the installation and the expected results on various configurations, as well as benchmarking the performance preferably continuously. This applies especially to software that targets high-performance computing (HPC) installations around Europe and the world. For these applications performance, scalability, and efficiency are key metrics that need to be monitored and compared among systems. Due to the complexity of these technical installations, individual scripts written for a specific system lack portability, reusability and reproducibility.

These challenges were addressed by the development of the Jülich Benchmarking Environment (JUBE) [1] at the Jülich Supercomputing Centre (JSC). JUBE is a generic and lightweight framework that automates the systematic execution, monitoring, and analysis of applications. It is a free, open-source software [2] implemented in Python that operates on a "declarative configuration" paradigm, where experiments are defined in human-readable YAML/XML files, automating script generation, job submission, and result analysis. Due to its standardized configuration format, it simplifies collaboration and usability of research software. JUBE integrates seamlessly with CI/CD pipelines, enabling automated regression testing, performance tracking, and benchmarking as part of HPC software development workflows.

The entry barrier of JUBE is relatively low as it builds upon basic knowledge of the Linux shell and either XML or YAML, and an extensive documentation including tutorials and advanced examples is available [2]. Offering a high degree of flexibility, JUBE may be used in every phase of the HPC software development pipeline. Example use cases comprise standard benchmarks to track a project's development in terms of performance, or systematic studies to explore parameter combinations---including orchestrating scaling experiments, which has already been shown to streamline the application process for HPC compute resources [3]. JUBE has been previously used to successfully automate a large variety of scientific codes and standard HPC benchmarks, with configurations available open-source [4]. The software can be easily installed, with existing configurations also available for the software managers EasyBuild [5] and Spack [6]. Further projects have been built ontop of JUBE [7,8].

In conclusion, JUBE is a well-established software, which has already been used in several national and international projects and on numerous and diverse HPC systems [9-16]. Given its broad scope and range of applications, JUBE is likely to be of interest to those working in the HPC software sector.

This presentation will provide an overview of JUBE, covering its core principles, current status, and future development roadmap. Additionally, two illustrative use cases will be presented to demonstrate JUBE's practical applications. The first is benchmarking as part of the procurement of JUPITER, Europe's first exascale supercomputer [3]; the second is continuous insight into HPC system health through the regular execution of applications, and the subsequent graphical presentation of their results.

[1] https://apps.fz-juelich.de/jsc/jube/docu/index.html
[2] https://github.com/FZJ-JSC/JUBE
[3] https://www.fz-juelich.de/en/jsc/jupiter/jureap
[4] https://github.com/FZJ-JSC/jubench
[5] EasyBuild: https://github.com/easybuilders/easybuild-easyconfigs/tree/develop/easybuild/easyconfigs/j/JUBE
[6] Spack: https://packages.spack.io/package.html?name=jube
[7] https://github.com/edf-hpc/unclebench
[8] https://dl.acm.org/doi/10.1145/3733723.3733740
[9] MAX CoE: https://max-centre.eu/impact-outcomes/key-achievements/benchmarking-and-profiling/
[10] RISC2: https://risc2-project.eu/?p=2251
[11] EoCoE: https://www.eocoe.eu/technical-challenges/programming-models/
[12] DEEP: https://deep-projects.eu/modular-supercomputing/software/benchmarking-and-tools/
[13] DEEP-EST: https://cordis.europa.eu/project/id/754304/reporting
[14] IO-SEA: https://cordis.europa.eu/project/id/955811/results
[15] EPICURE: https://epicure-hpc.eu/wp-content/uploads/2025/07/EPICURE-BEST-PRACTICE-GUIDE-Power-measurements-in-EuroHPC-machines_v1.0.pdf
[16] UNSEEN: https://juser.fz-juelich.de/record/1007796/files/UNSEEN_ISC_2023_Poster.pdf

Fri, 27.02.2026 11:00-11:30

Accelerating HPC Control Plane Development: Deploying OpenCHAMI Test Systems from Local VMs to Bare Metal

Alexandre Escoubas (ETH Zürich / Swiss National Supercomputing Center)

OpenCHAMI, the open-source successor to the Cray System Management (CSM) stack, reimagines HPC system management through a modular, microservices-based architecture. While this shift offers flexibility, it introduces complexity in replicating the environment for development and testing. To address this, we present a versatile tooling suite designed to empower developers by tightening the feedback loop, enabling the deployment of full-featured OpenCHAMI control planes on commodity workstations.

This presentation explores three distinct deployment workflows tailored to different stages of the development lifecycle:

1. Minikube-based Simulation: A self-contained environment using Libvirt to model an entire cluster—including virtual compute nodes and emulated Redfish BMCs—on a single laptop.

2. Docker Compose Quickstart: A rapid instantiation method for stand-alone service testing.

3. Podman Quadlets: A systemd-integrated approach for robust, production-like service management.

Crucially, we demonstrate how this setup is not limited to simulation but is capable of booting and provisioning physical hardware nodes directly from a developer's machine. Key DevOps discussions will cover the "Sidecar" pattern for synchronizing DHCP and State Management Database (SMD) state, the automation of custom boot image generation, and the orchestration of network services to handle hybrid virtual/physical environments. By democratizing access to complex infrastructure, these tools allow contributors to validate code changes instantly, accelerating the evolution of the next generation of HPC system management.