Workshop on Asynchronous Many-Task Systems 2026

Venue:
Garching near Munich, Germany

February 16-18, 2026

This workshop is sponsored by

- Max Planck Computing & Data Facility
- Technical University of Munich
- Oak Ridge National Laboratory (ORNL)
- AMD Lenovo

Abstract

As our compute capacity grows, science simulations are not only becoming bigger, but more complex. Simulations are carried out at multiple scales and using multiple kinds of physics at once. Boundaries are irregular, grids are irregular, computational domains can be dynamic and complex. In such scenarios, the ideal way to parallelize often cannot be statically determined. At the same time, hardware is becoming more heterogeneous and difficult to program. Increasingly, scientists are turning to asynchronous, dynamic parallelism in order to make the best use of increasingly challenging hardware. As a result, numerous frameworks, platforms, and specialized languages have sprung up to answer this need.

The objectives of this workshop are to bring together experts in asynchronous many-task frameworks, developers of science codes, performance experts, and hardware vendors to discuss the state-of-the-art techniques needed to program, analyze, benchmark, and profile these codes to achieve maximum performance possible from modern machines. This workshop will promote a dialogue between these communities, and help identify challenges and opportunities for advancement in all the disciplines they represent.

Organizing committee

- Patrick Diehl, Los Alamos National Laboratory (USA)
- Martin Schulz, Technical University of Munich (Germany)
- Erwin Laure, Max Planck Computing and Data Facility (Germany)
- Markus Rampp, Max Planck Computing and Data Facility (Germany)

Scientific committee

- Alex Aiken, Stanford (USA)
- Erwin Laure, Max Planck Computing & Data Facility (Germany)
- Christoph Junghans, Los Alamos National Laboratory (USA)
- Bryce Adelstein Lelbach, NVIDIA (USA)
- Laxmikant V. Kale, University of Illinois at Urbana-Champaign (USA)
- Brad Chamberlain, HPE and University of Washington (USA)
- John D. Leidel, Tactical Computing Laboratories (USA)

Technical program chair

- Patrick Diehl, Los Alamos National Laboratory (USA)
- Martin Schulz, Technical University of Munich (Germany)
- Erwin Laure, Max Planck Computing and Data Facility (Germany)
- Markus Rampp, Max Planck Computing and Data Facility (Germany)

Technical program

- Kevin Huck, University of Oregon (USA)
- Dirk Pflüger, University of Stuttgart (Germany)
- Huda Ibeid, Intel
- Dirk Pleiter, KTH Royal Institute of Technology (Sweden)
- Keita Teranishi, Sandia National Laboratories (USA)
- Gregor DaiSS, University of Stuttgart (Germany)
- Najoude Nader, Louisiana State University (USA)
- Hartmut Kaiser, Louisiana State University (USA)
- Steven R. Brandt, Louisiana State University (USA)
- Narasinga Rao Miniskar, Oak Ridge National Laboratory (USA)
- Markus Rampp, Max Planck Computing and Data Facility (Germany)
- Sumathi Lakshmiranganatha, Los Alamos National Laboratory (USA)
- Nikunj Gupta, Amazon (USA)
- Jonas Posner, University of Kassel (Germany)
- Chris Taylor, Tactical Computing Laboratories (USA)
- Aurelien Bouteiller, University of Tennessee, Knoxville (USA)
- Joseph Schuchart, University of Tennessee, Knoxville (USA)
- Rabab Alomairy, Massachusetts Institute of Technology (USA)
- Julian Samaroo, Massachusetts Institute of Technology (USA)

Logistics

- Friederike Neu, Max Planck Computing and Data Facility (Germany)

Welcome Address

Greetings, With great pleasure I send a warm welcome to all participants of the Workshop on Asynchronous Many-Task Systems and Applications, scheduled from February 16 to 18, 2026 at the Research Campus Garching near Munich. The event is hosted by the Max Planck Computing and Data Facility together with the Technical University Munich and the Leibniz Supercomputing center.

I would like to express my appreciation for the exceptional efforts of the workshop co-organizers, Dr. Markus Rampp, Prof. Martin Schulz and in particular Dr. Patrick Diehl. The objectives of this workshop are to bring together experts in asynchronous many-task frameworks, developers of science codes, performance experts, and hardware vendors to discuss the state-of-the-art techniques needed to program, analyze, benchmark, and profile these codes to achieve maximum performance possible from modern machines. This workshop will promote a dialogue between these communities, and help identify challenges and opportunities for advancement in various disciplines.

A special thanks goes to our distinguished keynote speakers: Dr. Rosa Badia from the Barcelona Supercomputing Center, Dr. Nick Brown from the Edinburgh Parallel Computing Centre, and Dr. Michael Klemm from AMD. Their expertise in the field promises valuable insights, and we are grateful for their willingness to share them with us. I would also like to thank our financial sponsers, Lenovo and AMD.
Finally, I extend my gratitude to all participants. I sincerely hope that you find enjoyment and benefit from the unique discussions and sessions planned over the next two and a half days.

With best wishes,

Prof. Dr. Erwin Laure Max Planck Computing and Data Facility & Technical University Munich

Contents

Session Chairs

- Keynote 1, Erwin
- Session 1 (Wed 10:30 – 12:00), Markus Rampp
- Session 2 (Wed 1:00 – 2:30), Christoph Junghans
- Keynote 2, Martin Schulz
- Session 3 (Thu 10:30 to 12:00), Jonas Posner
- Session 4 (Thu 1:00 to 2:30), Markus Rampp
- Session 5 (Thu 3:00 to 4:30), Christoph Junghans
- Keynote 3, Christoph Junghans
- Session 6 (Fr 10:30 to 12:30), Markus Rampp

Talks

Keynote 1: Fortran and the OpenMP API in a Modern GPU World

16^{th}
9:00 AM–10:00 AM

Michael Klemm
AMD

Modern day supercomputers are massively parallel, heterogeneous systems that employ accelerators (mostly GPU) to provide additional compute and memory performance to applications. While C/C++, but also Python, gain traction in the HPC domain, Fortran continues to have a large developer base with new high-performance code written every day. In this world, the OpenMP Application Programming Interface is one of the key components to support application developers and their need to write portable and performant code for such systems, especially in the context of large Fortran codes. We will review the evolution of the OpenMP API from its early days in 1997 to the present day and how it supports large scale, heterogeneous applications. We will recap how OpenMP initially supported portable multi-threading (for Fortran) and how it was extended to support task parallelism, single-instruction multiple-data, and heterogeneous computing. We will also shortly touch on future plans for the OpenMP API versions 6.1 and 7.0. The review of OpenMP features will be embedded in the journey of AMD to build the first and second exascale system, based on AMD EPYC(tm) Processors and AMD Instinct(tm) accelerators as well as a modern Fortran compiler based on LLVM Flang. Buckle up and enjoy the ride!

16^{th}
10:30 AM–11:00 AM

Concurrent Scheduling of High-Level Parallel Programs on Multi-GPU Systems

Peter Thoman
University of Innsbruck

Parallel programming models can encourage performance portability by moving the responsibility for work assignment and data distribution from the programmer to a runtime system. However, analyzing the resulting implicit memory allocations, coherence operations and their interdependencies can quickly introduce delays into the latency-sensitive execution pipeline of a distributed-memory application.

In this paper, we show how graph-based intermediate representations help moving such scheduling work out of the critical path. In the context of SYCL programs distributed onto accelerator clusters, we introduce instruction graph scheduling, a method based a low-level representation that preserves full concurrency between memory management, data transfers, MPI peer-to-peer communication and kernel invocations.

Through integration within the Celerity runtime, we demonstrate how instruction-graph scheduling enables a system architecture that performs this analysis concurrently with execution. Using a scheduler lookahead mechanism, we further detect changing access patterns to optimize memory allocation in the presence of virtualized buffers. We show the effectiveness of our method through strong-scaling benchmarks with multiple applications on up to 128 GPUs.

Accelerating GPRat: Asynchronous, Task-Based GPU Acceleration for Gaussian Process Regression with HPX

16^{th}
11:00 AM–11:30 AM

Alexander Strack
University of Stuttgart

Gaussian processes are a widely used regression tool, but exact computation is limited by the cubic complexity of standard solvers. To address this challenge, we extend the GPRat library by incorporating GPU acceleration. GPRat is an HPX-based framework that combines task-based parallelism, an intuitive Python API, and portability across x86, ARM, and RISC-V. We implement tiled algorithms for the Gaussian process prediction pipeline using optimized CUDA libraries, thereby exploiting massive parallelism for linear algebra operations. Performance is evaluated on a dual-socket AMD EPYC 9274F system with an NVIDIA A30 GPU. Results show that GPU acceleration provides speedups for datasets which exceed 128 training samples. Comparisons with GPyTorch and GPflow demonstrate that GPRat achieves superior prediction performance on CPUs and remains competitive on GPUs. Furthermore, combining HPX with multiple CUDA streams allows GPRat to match, and in some cases slightly surpass, native cuSolver implementations.

16^{th}
11:30 AM–12:00 PM

Hierarchical Scheduling for Composable Affinity-Aware Policies in Task-Based Runtimes

Vincent Arcila
Barcelona Supercomputing Center

iModern many-core processors combine deep memory hierarchies with a large number of cores, making both data locality and load balance critical for performance. Forkjoin constructs with static scheduling can preserve locality through fixed work-to-core mappings but may suffer severe load imbalance, whereas task-based runtimes balance irregular workloads via dynamic scheduling while offering limited control over task placement and execution order, often at the expense of locality. As different kernels may prioritize locality, load balance, or an intermediate trade-off, current scheduling mechanisms are insufficient to adapt to diverse and evolving application needs.

In this paper, we address this limitation by introducing a flexible scheduling framework built on two core abstractions: predefined scheduling policies and taskgroups. Taskgroups act as independent scheduling domains, allowing tasks from a specific kernel or subcomputation to be executed under a locally defined policy. By composing and nesting these domains, the framework enables complex and dynamic scheduling strategies, naturally supporting hierarchical and multi-kernel applications. The framework supports FIFO, LIFO, priority, and affinity policies and their combinations, with affinity exposed as a first-class mechanism to tune the trade-off between data locality and load balance. We demonstrate the integration of the framework in the OmpSs-2 tasking model, while keeping the design general enough to be applicable to other programming models. An evaluation using synthetic benchmarks shows that application behavior and performance can vary substantially depending on the chosen scheduling policy, and that the proposed framework makes such choices easy to express and modify. Overall, this work provides a practical and extensible path toward application-tailored scheduling in task-based runtime systems.

Evaluating MPI and OpenMP for global task dependencies across processes.

16^{th}
1:00 PM–1:30 PM

Jose Gracia

High Performance Computing Center Stuttgart, University of Stuttgart

MPI and OpenMP are still considered the work horses of high performance computing. OpenMP tasking allows to schedule interdependent tasks of various granularities within a process thereby avoiding process-global synchronisation which potentially increases system utilisation and efficiency. On the other hand, MPI's asynchronous communication or one-sided communication primitives allow to orchestrate communication such as to overlap with computation whenever possible. However, in practise it is surprisingly complex to encapsulate MPI communication in dependent tasks without risking deadlocks. In this paper we explore the potential of combining MPI continuations with OpenMP's detached tasks to construct dependencies between tasks across processes. MPI continuations are a proposed extension to MPI which in a nutshell replaces user code polling or waiting on request handles with the MPI library notifying completion through user-provided callbacks. The detach clause on OpenMP task construct allows to block otherwise ready tasks until user-code signals the occurrence of an event such as completion of communication. In combination, these techniques facilitate writing HPC applications with minimal synchronisation both within and across processes.

16th
1:30 PM–2:00 PM

Exploring Performance-Productivity Trade-offs in AMT Runtimes: A Task Bench Study of Itoyori, and MPI

Jonas Posner
Fulda University of Applied Sciences

Asynchronous Many-Task (AMT) runtimes offer a productive alternative to the established Message Passing Interface (MPI). However, the diverse landscape of AMTs makes fair comparisons challenging. The Task Bench benchmark, proposed by Slaughter et al., addresses this by providing a common framework to evaluate parallel programming systems.

This work integrates two recent cluster AMTs, Itoyori and ItoyoriFBC, into Task Bench for the first time, enabling a comprehensive evaluation against the well-known MPI and HPX. Itoyori employs a PGAS-style global memory and RDMA-based work stealing. ItoyoriFBC extends Itoyori with future-based synchronization.

We evaluate these systems in terms of both performance and programmer productivity. Performance is assessed across various configurations including compute-bound kernels, weak scaling, and both imbalanced and communication-intensive patterns. Performance is quantified using application efficiency, i.e., the percentage of maximum performance achieved, and the Minimum Effective Task Granularity (METG), i.e., the smallest task duration before runtime overheads dominate. Programmer productivity is quantified using lines of code (LOC) and the number of library constructs (NLC).

Our results reveal strengths and weaknesses of each system. MPI delivers the highest efficiency in well-balanced, communication-light configurations at the cost of verbose, low-level code. HPX sustains high, stable efficiency under imbalance over varying node counts. However, HPX surprisingly ranks last in our productivity metrics, indicating that AMTs may not always improve productivity over MPI. Itoyori demonstrates the highest efficiency on communication-intensive configurations and leads in programmer productivity. ItoyoriFBC is slightly less efficient than Itoyori, yet its future-based synchronization promises high programmer productivity for irregular workloads.

Managing Explicit Communications Within a Sequential Task Flow Paradigm

16^{th}
2:00 PM–2:30 PM

Nicolas Brieuc
Inria

Task-based runtime systems that follow the Sequential Task Flow paradigm rely on reconstructing a dependency graph from a sequential description of tasks. In the StarPU framework, which targets distributed-memory systems, each MPI process inserts tasks in sequence, specifying how they access data (read, write, or modify). From this description, StarPU infers the global Directed Acyclic Graph (DAG) of tasks and automatically determines the communications needed between processes. This model offers simplicity and portability, but gives the programmer little control over the placement and behavior of communications—sometimes at the expense of performance.

In this work, we introduce a new routine, starpu_mpi_data_cpy, which allows programmers to explicitly insert data transfers into the DAG. By specifying source and destination handles on the corresponding processes, applications can directly receive data in place, thus avoiding the additional memory allocations normally introduced by implicit STF communications. This mechanism provides an effective way to improve efficiency in communication-intensive regions, while still being integrated into the task-based execution model.

Explicitly inserting communications, however, breaks the pure STF flow and risks introducing subtle anti-dependencies. We describe the mechanisms that StarPU implements to reconcile these explicit operations with the inferred DAG, preserving correctness despite the asynchronous nature of execution. We then evaluate the benefits of this approach on representative distributed workloads, showing that the additional control enabled by starpu_mpi_data_cpy can translate into substantial performance improvements when used judiciously.

17^{th}
9:00 AM–10:00 AM

Keynote 2: Task-based programming models: tradeoffs between granularity and computing platforms

Rosa Badia
BSC-CNS

Task-based programming defends that provides a simple programming approach that leverages the computing infrastructure and reduces global synchronizations. Since its inception has been extensively applied to different computing platforms. In particular, at BSC we have developed instances of programming models for so diverse infrastructures as the grid, multicore or the Cell processor, clusters and large supercomputers, to the digital continuum or hybrid classic-quantum approaches. However, is the programming model so simple? What the end-users say about it? What are the issues that each computing platform pose? Is an autonomous runtime taking all decisions appreciated by end-users? How can we debug or improve the performance of distributed applications? The talk will present a perspective overview of these topics and our views for the next future.

Towards Dynamic Resource Management with Charm++

17^{th}
10:30 AM–11:00 AM

Dominik Huber
Technical University Munich

Dynamic Resource Management (DRM) can increase scheduling flexibility, benefiting both global system performance and individual job execution.

However, DRM requires adaptations of application codes to support dynamic changes of assigned resources during runtime. In this context, higher-level programming models, such as task-based programming models, could lower the bar for application developers to support DRM by hiding the complexities of dynamic process management and data redistribution.

In this work, we describe and evaluate our approach to integrate the task-based, object-oriented parallel programming model Charm++ into the generic Dynamic Processes with PSets (DPP) design. To this end, we extend the adaptive Charm++ features with a PMIx-based process manager, acting as a proxy between the Charm++ runtime and the dynamic resource manager. We evaluate our implementation on the SuperMUC-Ng cluster on up to 16 nodes, demonstrating its applicability in certain DRM scenarios.

17th
11:00 AM–11:30 AM

Finding Conservation Laws of Large Dynamical Systems with Tasks and Futures: A Case Study in Utilizing Dynamic Data Dependencies

Rüdiger Nather
Universität Kassel

Conservation laws provide valuable insights into the structure of dynamical systems that arise in physics and other fields. Recent research has shown that machine learning approaches, such as neural networks, can discover conservation laws from numerical data. Nevertheless, these approaches are still challenging to scale to large dynamical systems due to memory, data, and processing requirements.

A recently introduced technique based on kernel regression has addressed the high data requirements. Nevertheless, it lacks a high-performance implementation and still demands substantial memory for storing high-dimensional, dense matrices. Those memory requirements can be managed, for example, by utilizing hierarchical matrix representations, which also introduce a high degree of parallelism. However, those representations come at the cost of a (controllable) numerical approximation error and complex dependency patterns.

This work presents a task-based parallel algorithm for discovering conservation laws in dynamical systems that addresses all three aspects of the scaling problem. The algorithm leverages the kernel regression approach, exploits the parallelism introduced by hierarchical matrices, and uses futures, which are placeholders for asynchronously computed values, to manage dependencies. Weak-scaling experiments on an implementation of this algorithm in an extended version of the Taskflow runtime system demonstrate its efficiency in discovering conservation laws in systems with hundreds of dimensions within a reasonable timeframe.

Using C++ Generators in Dataflow Broadcast

17^{th}
11:30 AM–12:00 PM

Joseph Schuchart
Stony Brook University

Template Task Graph TTG) is a multiparadigm programming model built from task flowgraphs. Each task in TTG produces data for a single consumer (send) or multiple consumers (broadcast), possibly along multiple flowgraph edges. Until now identifiers of broadcast consumers had to be encoded as explicit dynamically-computed sequences, which is crucial for composing data-dependent algorithms. In this work, we split the task discovery from the main task body and introduce the use of generators available since C++23 to generate the successor list. This improves composability by allowing us to implement multiple broadcast strategies using the on-demand key generation.

17th
1:00 PM–1:30 PM

From Promts to Performance: Evaluating LLMs for Task-based Parallel Code Generation

Linus Batel
University of Stuttgart

Large Language Models (LLMs) show strong abilities in code generation, but their skill in creating efficient parallel programs is less studied. This paper explores how LLMs generate task-based parallel code from three kinds of input: natural language problem descriptions, pseudo-code, and sequential reference implementations. We focus on three programming paradigms: OpenMP Tasking, C++ standard parallelism, and the High Performance ParalleX (HPX) runtime. Each offers different levels of abstraction and control for task execution. We evaluate LLM-generated solutions for correctness, efficiency, and scalability, and compare them to manually written code. Our results reveal both strengths and weaknesses in task decomposition, dependency handling, and load balancing. Finally, we discuss what these findings mean for future LLM-assisted development in high-performance and scientific computing.

First Experiments to Evaluate the Relevance of Task-based Runtime Systems to Implement Large Language Model Applications

17^{th}
1:30 PM–2:00 PM

Lionel Eyraud-Dubois
Inria

During the last decade, a new kind of computation-intensive and complex application has emerged: Large Language Models (LLM). These applications require the computing power offered by HPC clusters and are complex to implement efficiently: several kinds of parallelisms are possible, limited available memory is often a major constraint, accelerators (GPUs, TPUs, NPUs, ...) need to schedule data transfers and computations, the problem size imposes distributed executions, ... All these challenges are already well-known by developers of the first-class applications running on HPC clusters: linear algebra, numerical simulation, ... Addressing these challenges has led to the development of task-based runtime systems, to ease the writing of HPC applications by providing an abstraction of the machine and its efficient programming. Despite task-based runtime systems being used for a long time now for classic HPC applications, they are generally not used to implement LLM applications. In this paper, we present our first experiments to analyze the relevance of using task-based runtime systems for LLM applications (both training and inference). We describe our implementation of a small LLM with StarPU, discuss the different choices we had to make, evaluate performance and summarize helpful and missing features of StarPU to implement an LLM.

Silent Data Corruption Protection through Efficient Task Replication

17^{th}
2:00 PM–2:30 PM

Mia Reitz
University of Kassel

The trend of increasing cluster sizes of supercomputers leads to a growing susceptibility to Silent Data Corruption (SDC) that can invalidate program results. A common strategy for SDC protection is replication, where the computation is repeated, and the correct result is determined as the one that is the same in at least two different computations. Applying replication to cluster Asynchronous Many-Task (AMT) runtimes is challenging due to dynamic task spawning and work stealing, which complicate the identification of replicated tasks.

To address the challenge, this paper introduces a novel replication scheme that detects and corrects SDCs for two classes of task cooperation: nested fork-join (NFJ) and future-based cooperation (FBC). Our approach replicates the computation and, upon a mismatch in the final result, traverses the task dependency graph top-down beginning at the final result to identify the frontier between correct and corrupted task results. Recovery is then performed by recomputing all tasks above the frontier, while the results of correct subtrees below it are reused.

We demonstrate our implementation within the ItoyoriFBC cluster AMT runtime. Our results show that the time to identify the affected tasks and reprocess them is negligible.

Keynote 3: Asychronous tasks: the cornerstone of targeting HPC on emerging architectures

18^{th}
9:00 AM–10:00 AM

Nick Brown
EPCC

Whilst emerging architectures, such as the Cerebras WSE, were initially designed with AI/ML workloads in mind their underlying capability results in great potential for traditional HPC too. However, a major challenge is in the programming of these technologies not-least because approaching these architectures as traditional imperative/BSP-style is a poor fit and results in slow performance. In this talk I will describe work targeting these architectures for existing HPC workloads, written in Fortran, with programmer's codes requiring minimal modifications. I will describe how we see that the target execution model naturally follows that of highly asynchronous many-tasks, and how we can automate via MLIR the mapping to this paradigm from the imperative view.

erformance Analysis of Task-Based Parallelism for 3D Acoustic Full Waveform Inversion on HPC Infrastructure

18^{th}
10:30 AM–11:00 AM

Vanderlei Munhoz Pereira Filho
INRIA

Full Waveform Inversion (FWI) is a computationally intensive geophysical method for estimating subsurface properties by iteratively minimizing the mismatch between observed and simulated seismic data. Traditional implementations rely on data-parallel approaches using MPI and OpenMP, distributing work through loop-level parallelism over seismic shots and spatial grids. This work presents a fundamentally different approach: a novel implementation of 3D acoustic FWI using StarPU, a task-based runtime system that decomposes computation into fine-grained tasks with explicit dependencies and provides dynamic, data-aware scheduling across heterogeneous CPU/GPU resources. We benchmark our task-based implementation against two established academic software packages employing conventional data-parallel paradigms: Mamute, a C++ framework using MPI for distributed memory parallelism and OpenMP for shared memory with custom workload balancing strategies, and simwave, a Python/C package implementing straightforward loop parallelism through OpenMP and OpenACC compiler directives. Performance experiments are conducted on (the) Grid'5000 Platform using standard synthetic 3D velocity models. This comparison directly contrasts task-based runtime systems against traditional loop-parallel approaches for large-scale geophysical inverse problems, analyzing execution time, scalability, load balancing efficiency, and implementation complexity. Our results provide insights into how modern task-based programming models perform relative to established data-parallel techniques for computationally demanding seismic imaging applications.

Multiresolution Analysis using Data-Flow Programming

18^{th}
11:00 AM–11:30 AM

Nilesh Chaturvedi
Stony Brook University

Multiresolution analysis (MRA) is widely used across scientific disciplines for its ability to represent both local and global data features effectively. Representations in MRA are tree-based with levels corresponding to the scale at which the function is represented. The operations that construct these representations and act on them are irregular and data-dependent and require traversal of the tree. The Template Task Graph (TTG) programming model enables data-dependent task discovery and can represent large-scale task graphs compactly by instantiating only those tasks for which inputs are available. We introduce a framework that implements MRA using TTG. Our work explores the representation of functions, operators, and the application of operators on functions in a multiresolution multiwavelet basis. Our construction tackles a variety of feasibility, efficiency, and numerical challenges in higher dimensions.

18^{th}
11:30 AM–12:00 PM

Abstract communicators for task based runtime systems

Nicolas Ducarton
University of Bordeaux

Task-based systems are outstanding at exploiting massively parallel and heterogeneous architectures. They are used for ever longer running problems, and on ever larger supercomputers. This leads to a heightened risk of failure, making fault-tolerance capabilities for such systems a desirable addition. To this end, StarPU has started including checkpoint-restart mechanisms directly inside the runtime system. But to tolerate actual failures, a distributed memory communication paradigm that enables the writing of fault-tolerant applications must be used. The MPI Forum has recently started including the User-Level Failure Mitigation (ULFM) extension in the MPI Standard precisely to achieve this goal. ULFM defines a set of routines that help support operations after failures, and the behavior of MPI when failures occur. The communication engine, and the task-based runtime system must thus be adapted to this specification. In particular, in order to recover the full communication capabilities, it becomes necessary to dynamically replace communicators (which are distributed MPI objects that are required by most MPI routines) that include failed processes by renewed communicators. To this end, in this work we introduce abstract communicators, a mechanism to replace MPI communicators during execution, including failed communicators. This enables the application to continue submission without changing its communicators, even when a failure occurs. The runtime system re-orders processes without the application noticing. We then present implementation challenges raised by this mechanism, notably on the process of handling requests on old communicators, and switching to replacements while still accepting new requests submitted by the application on its original communicators. We validate our approach with a working implementation in StarPU.

Merging Parameterized Task Graphs in PaRSEC through Recursive JDF Composition

18^{th}
1:00 PM–1:30 PM

Zhuowei Gu
Saint Louis University

The Parameterized Task Graph (PTG) interface in the PaRSEC runtime system provides a powerful mechanism to express scalable, parameterized task dependencies for distributed heterogeneous computing. However, as applications become increasingly complex and hierarchical, developers often need to compose or merge multiple JDF (Job Data Flow) descriptionseach representing a subgraphinto a single coherent PTG, a process that is currently manual, error-prone, and difficult to generalize. In this work, we introduce a Python- and AI-assisted framework to automate the merge-to-JDF process in PaRSEC. The framework recursively analyzes two or more JDFs, identifies overlapping dataflows, and merges their task and data definitions into a unified PTG while preserving correctness and dependency semantics. We demonstrate this approach on representative PTG applications, including multi-stage solvers and AI models such as neural networks, convolutional networks, and Transformers, showing that the automated merge framework substantially reduces development effort while maintaining runtime efficiency.

18^{th}
1:30 PM–2:00 PM

Leveraging PaRSEC for Task-Based Heterogeneous Computing in Python and Julia

Qiao Zhang
Saint Louis University

We present Python and Julia bindings for the PaRSEC task-based runtime system, enabling high-level access to its efficient and portable execution model on distributed heterogeneous architectures. The Python interface (PyPaRSEC) integrates seamlessly with NumPy and PyTorch, while the Julia interface (PaRSEC.jl) leverages Julias multiple dispatch and composability to express dynamic task graphs naturally. Both interfaces expose PaRSECs core abstractionsincluding data management, task creation, and device schedulingallowing domain scientists to construct parallel workflows easily across diverse architectures. We evaluate these bindings using Cholesky factorization and general matrixmatrix multiplication (GEMM) on multi-core CPUs, NVIDIA GPUs, and AMD GPUs. The results show that the Python and Julia interfaces achieve performance comparable to native C implementations, with minimal overhead and strong scalability across devices. This work demonstrates that PaRSECs high-performance runtime can be effectively leveraged from productive, high-level languages, bridging the gap between scientific prototyping and exascale execution.

Additional information

Addresses

Workshop venue

Max Planck Computing & Data Facility on Research Campus Garching, GieSSenbachstrasse 2 85748 Garching near Munich, Germany. Access via Isarstrasse. Lecture hall on ground level next to the main entrance to building D2.

Hotel

Courtyard Hotel Munich Garching Walter-von-Dyk-Str. 12 85748 Garching near Munich, Germany

Banquet

Bayerische Staatsbrauerei Weihenstephan, Alte Akademie 2, 85354 Freising, Germany

Author Index

This work is licensed under a Creative Commons "Attribution-NonCommercial-NoDerivatives 4.0 International" license.