Workshop on Asynchronous Many-Task Systems 2025

Venue:
Saint Louis University

February 19-21, 2025

This workshop is sponsored by

- Saint Louis University

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)
- Qinglei Cao, Saint Louis University (USA)

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)
- George Bosilca, NVIDIA (USA)
- Thomas Herault, University of Tennessee, Knoxville (USA)

- Qinglei Cao, Saint Louis University (USA)

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)
- Didem Unat, Koç University (Turkey)
- Keita Teranishi, Sandia National Laboratories (USA)
- Gregor DaiSS, University of Stuttgart (Germany)
- Najoude Nader, Louisiana State University (USA)
- Weile Wei, Lawrence Berkeley National Laboratory (USA)
- Jeff Hammond, NVIDIA (Finland)
- Hartmut Kaiser, Louisiana State University (USA)
- J. Ram Ramanujam, 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)
- Wei Wu, NVIDIA (USA)

Logistics

- Andrew Flanagan, Saint Louis University (USA)
- Qinglei Cao, Saint Louis University (USA)

Welcome Address

Greetings,

It gives me great pleasure to extend a warm welcome to all participants of the Workshop on Asynchronous Many-task Systems and Applications, scheduled from February 19 to 21, 2025. The event is being hosted by the Computer Science Department at Saint Louis University. I am delighted to announce our support and co-sponsorship of this workshop.

I would like to express my appreciation for the exceptional efforts of the workshop organizers: Dr. Qinglei Cao and Dr. Patrick Diehl. The primary objective of this workshop is bringing together experts in asynchronous many-task frameworks, scientific code developers, performance engineering specialists, and hardware vendors and to create an environment where discussions and new ideas flourish. Our aim is to facilitate discussions on cutting-edge techniques necessary for the development, analysis, benchmarking, and profiling of task-based applications to achieve optimal performance on modern architectures. This workshop serves as a platform for fostering dialogue among these communities, identifying challenges, and exploring opportunities for advancement across various disciplines.

A special thanks goes to our distinguished keynote speakers: Prof. Alan Edelman from Massachusetts Institute of Technology, Dr. Christian Trott from Sandia National Laboratories, and Dr. Hatem Ltaief from King Abdullah University of Science and Technology. Their expertise in the field promises valuable insights, and we are grateful for their willingness to share them with us. 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,
Min Choi Ph.D.
Professor and Chair of Computer Science
Saint Louis University

Contents

Session Chairs

- Keynote 1, Qinglei Cao
- Session 1 (Wed 10:30 – 12:00), Christoph Junghans
- Session 2 (Wed 1:00 – 2:30), Vicki Carrica
- Session 3 (Wed 3:00 – 4:30), Joseph Schuchart
- Keynote 2, Patrick Diehl
- Session 3 (Thu 10:30 to 12:00), Patrick Diehl
- Session 4 (Thu 1:00 to 2:30), Jiakun Yan
- Session 5 (Thu 3:00 to 4:30), Aurelien Bouteiller
- Keynote 3, Qinglei Cao
- Session 6 (Fr 10:30 to 12:30), Qinglei Cao

Talks

Keynote I: Improving the HPC experience, did Julia get it right or will AI hide the problem (or both)?

19th
9:00 AM–10:00 AM

Alan Edelman

Massachusetts Institute of Technology

For years I was sad that not enough hard work went into making HPC much easier on the human, and much more portable. I kind of felt that there was too much chasing machoflops and not enough software engineering that would result in bringing more programmers to HPC. My benchmark was how many undergraduates are using HPC for interesting projects. Another observation was that it seemed infrastructure was rarely funded or encouraged.

Julia has been and is making strides in this direction, but new to the table is AI (something this tech lover turned luddite wanted to avoid always fearful this bandwagon is going to collapse and collapse hard.) I will talk about the original ongoing efforts and our SmartSolve (Funded by DARPA DIAL) project that we hope will bring everything together for HPC. Most importantly we will encourage everyone to participate; we can not do this entirely ourselves and value your contributions.

19th
10:30 AM–11:00 AM

Contemplating a Lightweight Communication Interface for Asynchronous Many-Task Systems

Jiakun Yan
University of Illinois Urbana-Champaign

AMTs exhibit different communication patterns than traditional HPC applications, characterized by asynchrony, concurrency, and multithreading. Existing communication libraries usually do not support AMTs' communication requirements in the most direct and efficient ways. The Lightweight Communication Interface (LCI) is an experimental communication library aiming to push for efficient communication support for AMTs. This paper presents the design for a new LCI C++ interface and its rationale. With a new C++ objectized flexible functions idiom, the new interface aims for the following features: (a) a concise but expressive interface for all common communication primitives and completion mechanisms, (b) a fine-grained resource mapping scheme for library interoperation, multithreaded performance isolation, and flexibility (c) a set of optional parameters and overridable classes for users to fine-tune the runtime behavior incrementally.

Comparing and Contrasting User and Runtime Directed Data Placement Strategies for Owner-Compute, Multi-Accelerator Distributed Task Based Scheduling

19th
11:00 AM–11:30 AM

Aurelien Bouteiller
University of Tennessee Knoxville

The current dominance of accelerators in leadership class High Performance Computing systems has motivated the emergence of the task-based programming style. This programming model enables the dynamic execution and mapping of the computation on computing resources and a greater asynchronous execution of tasks, henceforce enabling the execution to reach a higher portion of the computational peak. Another important aspect is the overlap between computation and the motion of data between nodes and memory hierarchies. The task-based programming paradigm, as employed in the PaRSEC micro-task runtime system, enables the decoupling of the data distribution and mapping of computation to resources from the base expression of the algorithm. As a consequence, it becomes easy to modify these mappings and explore the performance impact between a number of strategies, some automatic and runtime directed, and some user-directed. In this paper, we focus on the comparison and contrast of different data placement strategies for the owner-compute scheduling model in the context of split-memory accelerators–that is, when host memory and accelerator memory are separate domains, or when accessing host memory through Unified Virtual Memory (UVM) incurs a significant cost. We implement in the same algorithms (for example LLT Cholesky factorization, tensor contraction, mixed-precision algorithms) three different strategies for data and task mapping: a randomized first-touch policy that assigns data randomly to an accelerator, a load-balancing strategy that assings data to the accelerator with the lowest load, and we compare it to an user-directed strategy that minimizes cross-accelerator traffic by placing tasks according to a cross-memory bandwidth minimizing strategy. We also compare these strategies to using UVM to let the hardware position data on-demand on the site of computation as a baseline. An important consideration that we take into account is the positioning of data received from the network in distributed systems that are capable of depositing message payload directly in accelerator memory, this is critical for example on the Frontier system where network interfaces have closer affinity with accelerator memory banks than host memory banks. Evaluation will be carried out on a variety of multi-GPU accelerated systems (using the PaRSEC capabilities to schedule tasks with CUDA, HIP, OneAPI), including the Frontier system. We finally discuss how these strategies, including user-directed strategies can be implemented in a manner that maintain the separation of concerns between expressing the correctness of the algorithm and the mapping of tasks on resources.

19th
11:30 AM–12:00 PM

Chplx an Asynchronous Many Task Runtime Foundation for Chapel

Shreyas Atre
lsu center for computing and technology

A previous study demonstrated Chapel and HPX yielded best performance with respect to code complexity ratios for a 1 dimensional heat equation code. The study showed that HPX has a code complexity gap with respect to Chapel. This paper presents results of a study to close HPX's code complexity gap by way of a new source-to-source compiler called chplx. chplx converts Chapel to HPX application software. The chplx compiler creates ISO C++ application code using a new ISO C++20 library implemented using the HPX asynchronous many task (AMT) runtime system. The new ISO C++20 library closely mirrors functionality found in the Chapel programming language. The results of this study show performance and code complexity ratios between Chapel and HPX measured across 3 benchmarks and the COnstructive COst MOdel (COCOMO). The 3 benchmarks used for this study are GUPS, Streaming Triad, and a heat equation kernel. Performance is measures across 3 architectures: x86, aarch64, and risc-v (rv64g). Our performance results demonstrate chplx can maintain high performance and close the code complexity gap.

Supporting OpenMP Free Agents by Leveraging the nOS-V Threading Library

19th
1:00 PM–1:30 PM

Vicenc Beltran Querol
Barcelona Supercomputing Center

The OpenMP 6.0 standard introduces the free-agents feature, which enables a more dynamic execution model to improve application malleability and resource utilization. With free-agents, idle threads within a parallel region can be dynamically reassigned to execute tasks from other parallel regions. However, integrating this feature into complex runtimes, such as LLVMs libomp, presents significant challenges due to its departure from traditional threading models.

This paper presents our implementation of free-agents within the LLVM libomp runtime, which significantly reduces the complexity of implementing free-agents in libomp by leveraging the nOS-V threading and tasking library. Moreover, nOS-V also provides support for Task-Aware (TA) libraries and the co-execution of applications. Our enhanced libomp runtime, ported on top of nOS-V (libompv), also leverages these features. With libompv, OpenMP tasks and threads from any parallel region can safely call TA libraries, and independent processes can be efficiently co-executed to improve resource utilization or mitigate load imbalance issues. Experimental results show that libompv+fa delivers better performance than the original libomp for load-imbalanced or co-executed applications, while introducing no additional overhead for workloads that do not benefit from these new features.

19th
1:30 PM–2:00 PM

Futures in Task Graphs Extending Taskflow With Dynamic Data Dependencies

Rüdiger Nather
University of Kassel

Task parallel programming is a common approach to using modern multicore architectures efficiently. However, tasks can have dependencies that need to be accounted for by the scheduler. Particularly difficult to handle are dynamic dependencies, which may be discovered only at runtime. Dynamic dependencies can be expressed with the future construct, which has several variants. Previous research on the LU decomposition of hierarchical matrices suggested that futures should have properties such as the ability to encapsulate other futures, and that of being movable to another task for being filled. Such powerful future types have not yet been implemented in current runtime systems.

This paper fills the gap and implements a powerful future type in Taskflow, which is a parallel runtime system that supports, e.g., heterogeneous tasks, static dependencies, and task-level control flow. We present our extension of Taskflow by futures, show how futures integrate into the programming model of Taskflow, and evaluate our implementation with the LU decomposition of hierarchical matrices.

Adaptively Optimizing the Performance of HPX's Parallel Algorithms

19th
2:00 PM–2:30 PM

Karame Mohammadiporshokooh
Louisiana State University

Executors in C++ abstract concurrency management across diverse hardware architectures, simplifying development by providing a consistent interface for task execution. While this abstraction facilitates portability and uniformity of the user-facing interfaces, it can also lead to performance inefficiencies by imperfectly matching the workloads or by not fully leveraging specific hardware capabilities. To mitigate this, dynamic optimizations can be incorporated into executors, enabling them to adjust their behavior based on the supplied workload, code complexity, and hardware they operate on, optimizing scheduling, resource allocation, and task distribution. We developed a chunking (workload) optimization integrated into HPXs executor API that dynamically determines optimal workload distribution and resource allocation based on runtime metrics without introducing undue overheads. Evaluated within the adjacent difference algorithm, this approach demonstrates improvements in execution time and efficiency across various configurations and workloads, offering a promising solution for improved parallel performance through a user-friendly API.

19th
3:00 PM–3:30 PM

Fail-stop Failure Protection for Coordinated Work Stealing of Tasks that Communicate through Futures

Claudia Fohry
University of Kassel

Modern supercomputers nowadays consist of millions of compute cores. This still growing number increases the likelihood of process failures, making fault tolerant programs essential, especially on clusters. Traditionally, fault tolerance is achieved with Checkpoint/Restart (C/R), where process states are periodically saved to disk, and a collective restart is performed after failure. This general-purpose approach is transparent to application programmers, but incurs a high running time overhead.

The present paper, in contrast, deals with a specific fault tolerance technique for Asynchronous Many-Task (AMT) programs, called task-level checkpointing (TC). This technique is more efficient than C/R, and transparent to application programmers, as well. AMT programs divide the computation into tasks that are processed by worker processes running on, e.g., different cluster nodes. For load balancing, the workers often employ work stealing, in which idle workers steal tasks from other workers. TC operates in the runtime system and saves the data of clearly defined task interfaces instead of process states.

So far, TC has only been applied to restricted classes of AMT runtimes, chiefly to runtimes with independent or nested fork-join tasks under cooperative work stealing, i.e., victims participate in steals. This paper adapts the technique to a runtime with task communication through futures under coordinated work stealing, i.e., thieves directly take data from victim memory. We present and evaluate first checkpointing algorithms for this setting, observing overheads of up to 12% at 1280 workers.

Q-IRIS: The Evolution of the IRIS Task-Based Runtime to Enable Classical-Quantum Workflows

19th
3:30 PM–4:00 PM

Anthony Cabrera
Oak Ridge National Laboratory

Heterogeneous systems are ubiquitous and are growing to include even more diverse paradigms, such as quantum computing. This paper strives to design an asynchronous task-based runtime solution that can encapsulate both classical and quantum computing environments in the heterogeneous execution paradigm by exploring a few integration possibilities between the task-based runtime IRIS, the quantum programming framework XACC, and the Quantum Intermediate Representation Execution Engine (QIR-EE). The need for asynchronous task-based execution is motivated by examples that require the coexistence of classical and quantum computing hardware. To show a proof-of-concept for motivating future study, we describe the principles of integrating quantum runtimes with a task-based runtime and demonstrate its capability of parallelizing quantum circuit execution by decomposing a four-qubit circuit into a collection of smaller circuits, which lowers the quantum simulation load during execution. We hope that this will further highlight challenges that we would need to overcome to make such a solution effectively scalable while simultaneously capturing classical-quantum and quantum-quantum interactions.

20th
9:00 AM–10:00 AM

Keynote II: Task-Graphs: Why aren't we all using them?

Christian Trott
Sandia National Laboratories

Task-Graphs are a very attractive concept to express complex algorithms, manage asynchronicity and expose available concurrency. However, very few HPC applications are written in terms of task-graphs. In this talk I will provide reflections on what I believe some of the reasons are, based on the experience gained with Kokkos adoption in the last decade. To ground the discussion, this talk will provide an overview of the Kokkos::Graph design - a Kokkos capability that despite being introduced 4 years ago, has found very little adoption. A comparison with CUDA Graphs will demonstrate design tradeoffs in this space, and help highlight adoption hurdles and correctness pitfalls that are likely an important part of the reason why task-graphs are not more widely used. Last but not least the talk will touch on how the recently approved ISO C++ 26 Execution framework (Senders and Receivers) can help overcome some of these adoption hurdles, and allow us to realize the promise of a task-graph based application design.

Type-level invariants for SPMD programming with Rust

20th
10:30 AM–11:00 AM

Nafees Iqbal
University of Colorado Boulder

Rust is a systems programming language that emphasizes memory- and type-safety, and provides a strong type system. A library or interface is considered "sound" if this safety cannot be breached without (mis)using the unsafe keyword, but safe code using sound libraries can still deadlock or yield incorrect results. We consider some representative bugs arising in parallel software development and explain in terms of invariant violation. We consider ways of addressing such bugs using Rust's type system in the context of the Rust interface for MPI as well as GPU and threaded programming paradigms with nontrivial data partitions, and give an outlook on improving the reliability and productivity of parallel software development.

20th
11:00 AM–11:30 AM

Evaluating AI-generated code for C++, Fortran, Go, Java, Julia, Matlab, Python, R, and Rust

Patrick Diehl
University of Stuttgart

GPU implementations for algorithms that involve many but fine-grained compute kernels often struggle with both GPU API overheads and device starvation. One way to overcome this is kernel fusion. In this work, we aim to compare two distinct strategies for kernel fusion with SYCL. The first one is using the SYCL graph extension, which allows developers to record a graph of kernel invocations and launch the recorded subset as one single kernel upon repetition. The second strategy uses a task-based runtime system, HPX, and a kernel-fusion executor from the library CPPuddle. Here, similar kernel invocations are fused into a single kernel on-the-fly without requiring any graph recording. Instead, this scheme relies on additional input from the developer and uses other criteria to decide whether to fuse kernels together during the runtime. Both strategies come with their own upsides and downsides, as well as different runtime overheads. Thus, in this work, compare both these solutions to kernel fusion. We focus primarily on the performance using a sample algorithm, however, we also discuss their usability for different scenarios.

Dynamic Resource Management: Comparison of Asynchronous Many-Task (AMT) and Dynamic Processes with PSets (DPP)

20th
11:30 AM–12:00 PM

Jonas Posner
University of Kassel

Dynamic resource management allows programs running on supercomputers to adjust resource allocations at runtime. This dynamism offers potential improvements in both individual program efficiency and overall supercomputer utilization.

Despite growing interest in recent years, the adoption of dynamic resource management remains limited due to inadequate support from widely used resource managers, such as Slurm, and programming environments, such as MPI. Furthermore, developing flexible programs introduces substantially higher programming complexity compared to static programs.

While recent research has improved MPI's resource flexibility, significant programmability challenges remain. Additionally, MPI-based solutions rely on low-level message-passing primitives, which are particularly challenging to use for non-iterative workloads.

Asynchronous Many-Task (AMT) programming offers a promising alternative to MPI. By decomposing computations into tasks that are dynamically scheduled by the runtime system, AMT is well suited to handling irregular and dynamic workloads. AMT's transparent resource management is ideal for dynamic resources, allowing the runtime system to seamlessly redistribute tasks in response to node changes without requiring additional programmer effort.

In this work, we compare the "Dynamic Processes with PSets (DPP)" design principle implemented in an MPI-based environment and the APGAS+GLB AMT runtime system. We implement benchmarks in both environments to evaluate programmability and perform experiments on up to 16 nodes to analyze the performance of static and flexible programs. Results demonstrate that GLB simplifies programming with built-in load balancing and resource flexibility. In contrast, the MPI-DPP implementation achieves superior performance in handling node changes but at the cost of increased programming complexity.

20th
1:00 PM–1:30 PM

Unifying the Architecture and Implementation of Task-Aware Libraries

Amadeu Moya
Barcelona Supercomputing Center

Modern computing platforms require the coordination of CPU, GPU, network, and storage devices, among others. This heterogeneity forces application developers to use several APIs to leverage those devices. Tasking models are a promising method for orchestrating such heterogeneity by serving as the parallelism backbone in those programs. An application can be represented as a set of direct acyclic graphs (DAG) where vertices represent tasks comprising CPU computation, GPU offloading, and I/O operations, and the edges represent the data dependencies between the tasks.

The Task-Aware Libraries (TA-X) is a software ecosystem that allows the tasks of an application to perform those time-consuming operations efficiently. The ecosystem includes a task-aware library per API supported, such as MPI, CUDA, and I/O. Each library provides task-aware operations that are linked to the underlying API and avoid the issues of mixing blocking/non-blocking operations with tasks. Also, these libraries are supported by any task-based runtime system that implements the Asynchronous Low-level Programming Interface (ALPI). Underneath, most TA-X libraries have similar architecture and implementation. However, developing new TA-X libraries is still tedious and repetitive, which leads to code duplication, increased maintainability, and non-portable performance.

In this paper, we analyze the architectures of the existing TA-X libraries and propose a high-performance unified design for current and future libraries. With our design, TA-X libraries share most of the code, and they only have to implement a small set of API-specific functionalities. This facilitates the rapid and straightforward development of novel TA-X libraries, while the ALPI interface facilitates their portability over task-based runtime systems. On top of that, TA-X libraries can directly benefit from the optimizations applied to the common components. We demonstrate the benefits of our design by porting the existing TA-X libraries, which cover communications, offloading of HPC code and graphic computation to GPUs, and I/O operations.

Data Sparsity in Global and Compact Support Radial Basis Functions for 3D Unstructured Mesh Deformation

20th
1:30 PM–2:00 PM

Rabab Alomairy
Massachusetts Institute of Technology

We explore the data sparsity characteristics of various commonly used Radial Basis Function (RBF) kernels in the context of 3D unstructured mesh deformation. While RBF interpolation is a powerful method for generating high-quality adaptive meshes, solving the resulting boundary problems leads to large, dense linear systems that are computationally expensive and memory-intensive due to their cubic complexity. To address these challenges, we exploit the rank structure of the matrix operators by employing a Tile Low-Rank Cholesky-based solver, which approximates off-diagonal matrix tiles up to an application-specific accuracy threshold. Our study compares global support RBFs and compact support RBFs, focusing on their effects on rank distribution and numerical accuracy. Using realistic 3D geometries of SARS-CoV-2 viruses from the Protein Data Bank, we evaluate various RBF kernels, analyze the corresponding matrix rank structures, and assess the backward error resulting from low-rank approximations for different kernel types. We conduct experiments on various shared and distributed systems, demonstrating the performance scalability on massively parallel architectures. Leveraging the Hierarchical Computations on Manycore Architectures (HiCMA) library and the PaRSEC runtime system, we show how data sparsity accelerates large-scale mesh adaptation, providing valuable insights into the balance between computational efficiency and numerical accuracy.

20th
2:00 PM–2:30 PM

A Task-parallel Pipeline Programming Framework with Token Dependency

Cheng-Hsiang Chiu
University of Wisconsin at Madison

Task-parallel pipeline framework explores pipeline parallelism in applications and is critical in many parallel and heterogeneous areas, such as VLSI static timing analysis and data similarity search. However, existing solutions only deal with certain types of applications in which data dependency exists between preceding data and succeeding data in a forward direction. Some applications, such as video encoding, exhibit data dependency in both forward and backward directions and cannot be processed with existing solutions. To address the limitation, we introduce a token dependency-aware pipeline framework. Our framework associates each data element with a token as its identifier, supports explicit definitions of forward and backward token dependency with an expressive programming model, resolves token dependency using simple data structures, and schedules tokens with lightweight atomic counters. We have evaluated the framework on applications that exhibit both forward and backward token dependency. For example, our framework is 8.6% faster than PARSECs implementation in x.264 video encoding applications.

GPRat: Gaussian Process Regression with Asynchronous Tasks

20th
3:00 PM–3:30 PM

Alexander Strack
University of Toronto

Python is the de-facto language for software development in artificial intelligence (AI). Commonly used libraries, such as PyTorch and TensorFlow, rely on parallelization built into their BLAS backends to achieve speedup on CPUs. However, only applying parallelization in a low-level backend can lead to performance and scaling degradation. In this work, we present a novel way of binding task-based C++ code built on the asynchronous runtime model HPX to a high-level Python API using pybind11. We develop a parallel Gaussian process (GP) library as an application. The resulting Python library GPRat combines the ease of use of commonly available GP libraries with the performance and scalability of asynchronous runtime systems. We evaluate the performance on a mass-spring-damper system, a standard benchmark from control theory, with varying number of regressors (features). Results show almost no overhead when binding the asynchronous HPX code using pybind11. Compared to GPyTorch and GPflow, GPRat shows superior scaling on up to 64 cores on an AMD EPYC 7742 CPU for training. Furthermore, our library achieves a prediction speedup of up to factor 7.63 over GPyTorch and 25.25 over GPflow. If we increase the number of features from eight to 128, we observe speedups of up to factor 29.62 and 21.19, respectively. These results showcase the potential of using asynchronous tasks within Python-based AI applications.

20th
3:30 PM–4:00 PM

Julia-Unified Recursive Implementation of TRMM and TRSM for GPU Acceleration

Maxwell Onyango
Massachusetts Institute of Technology

This paper presents an innovative approach to implementing the triangular matrix-matrix multiplication (TRMM) and triangular solve matrix (TRSM) operations using Julia for GPUs, leveraging the KernelAbstraction.jl framework. TRMM is crucial in solving systems of equations with triangular matrices, while TRSM is essential for inverting triangular matrices, both forming the backbone of many linear algebra algorithms. This work is based on an existing recursive implementation for TRMM and TRSM, which restructures the operations to include general matrix-matrix multiplication (GEMM) calls. This restructuring reduces memory traffic, increases data reuse, and enhances concurrency, facilitating better utilization of the GPU memory hierarchy and reducing latency overhead. The unified implementation in Julia harnesses the language's LLVM-based compilation and type inference capabilities, enabling efficient and hardware-agnostic execution across different GPU architectures. By supporting a consistent API, this implementation allows users to seamlessly switch between different GPU backends, achieving performance comparable to vendor-optimized libraries. This portability and performance consistency make the proposed Julia-native approach ideal for high-performance computing applications that require adaptable and scalable solutions across heterogeneous computing environments.

Keynote III: Chaos to Cosmos: Orchestrating Complex Scientific Applications with Dynamic Runtime Systems

21th
9:00 AM–10:00 AM

Hatem Ltaief
King Abdullah University of Science and Technology

Scientific computing is increasingly confronted with the challenge of orchestrating highly complex simulations and AI-driven workflows on heterogeneous hardware systems. The sheer scale of modern scientific applicationsranging from computational astronomy to genomic analysis and climate modelingintroduces difficult data dependencies, irregular execution patterns, and severe performance bottlenecks. Traditional static execution models when faced with complex irregular workflows struggle to efficiently harness the computational power of emerging hardware architectures, leading to suboptimal resource utilization and scalability limitations.

In this talk, we explore how dynamic runtime systems provide a paradigm shift, transforming computational chaos into a structured and efficient cosmos. We delve into research on task-based programming models, asynchronous execution, and mixed-precision algorithms, drawing from state-of-the-art developments in the field. By leveraging dynamic scheduling, data locality optimizations, and adaptive precision techniques, we demonstrate how these runtime systems mitigate bottlenecks, improve energy efficiency, and enable unprecedented scalability on modern supercomputing platforms.

By bridging the gap between numerical methods, system software, and cutting-edge hardware, this talk provides a roadmap for orchestrating complex scientific applications with dynamic, intelligent, and scalable runtime systemsturning computational disorder into scientific discovery.

21th
10:30 AM–11:00 AM

Scalable Block-Sparse Matrix Multiplication Using Template Task Graphs

Joseph Schuchart
Stony Brook University

Block-sparse matrix operations are a special case of general sparse algebra where the matrix is sparsely populated with dense blocks, e.g., in sparse tensor algebra for quantum chemistry. One of the challenges of implementing distributed matrix multiplication $C = A \times B$ in general is the management of communication flows since both input matrices A and B are readily available and must be distributed to the processes computing the relevant blocks of C. In this paper, we propose an addition to the Template Task Graph programming model that allows applications to constrain the execution of tasks using a flexible API. We show that such constraints can be used in a pure dataflow model to replace artificial control flow with a more structured approach. In the context of sparse matrix multiplication, we found that constraints allow us to limit the number of concurrent communications and thus avoid creating a bottleneck in the network.

Leveraging Hardware-Aware Computation in Mixed-Precision Matrix Multiply: A Tile-Centric Approach

21th
11:00 AM–11:30 AM

Qiao Zhang
Saint Louis University

General Matrix Multiplication (GEMM) is a critical operation underpinning a wide range of applications in high-performance computing (HPC) and artificial intelligence (AI). The emergence of hardware optimized for low-precision arithmetic necessitates a reevaluation of numerical algorithms to leverage mixed-precision computations, achieving improved performance and energy efficiency. This research introduces an adaptive mixed-precision GEMM framework that supports different precision formats at fine-grained tile/block levels. We utilize the PaRSEC runtime system to balance workloads across various architectures. The performance scales well on ARM CPU-based Fugaku supercomputer, Nvidia GPU-based A100 DGX, and AMD GPU-based Frontier supercomputer. This research aims to enhance computational efficiency and accuracy by bridging algorithmic advancements and hardware innovations, driving transformative progress in various applications.

Posters

HPX+MPICH: Bridging the Gap between Asynchronous Many-Task Systems and MPI with VCIs and Continuation

Jiakunf Yan
University of Illinois Urbana-Champaign

The MPI standard lacks direct support for communication functionalities AMTs need. Therefore, AMT communication subsystems often resort to significant efforts to implement their needs using the limited capabilities provided by MPI, leading to compromises in both programmability and performance. In this work, we study two of the gaps that have been shown to have large application-level impacts: efficient management of many pending operations and communication resource replication. We test existing and newly developed MPICH extensions to see how well they fit into the AMTs communication model and how much they can mitigate the aforementioned overheads. They include MPIX Continuation to efficiently manage a large number of pending communication operations by introducing a callback-based completion mechanism and MPICHs VCI-mapped communicators to alleviate thread contention on internal communication resources by replicating and mapping them to different communicators. The new MPI parcelport outperforms the existing one and greatly shrinks the performance gap between the LCI and MPI parcelports. We also identified areas for improvement in the MPICH extensions.

Additional information

Addresses

Workshop venue

Busch Student Center, 20 N Grand Blvd, St. Louis, MO 63103

Hotel

Angad Arts Hotel, 3550 Samuel Shepard Dr, St. Louis, MO 63103

Banquet

On campus

Author Index