Workshop on Asynchronous Many-Task Systems 2025

Venue:
Lee Lecture Hall, Room 001
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.

- 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

- 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 Innovative Computing Lab at the University of Tennessee, Knoxville, on the UT campus located in Knoxville, Tennessee. I am delighted to announce our support and co-sponsorship of this workshop, alongside other contributors such as Tactical Computation Labs, LSU and ORNL.
I would like to express my appreciation for the exceptional efforts of the workshop organizers: the local organizers, Dr. Joseph Schuchart and Dr. George Bosilca, as well as the workshop chairs Dr. Patrick Diehl and Dr. Pedro Valero-Lara. 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,

Jack Dongarra

Contents

Industrial talk

Orienteering RISC-V for HPC

20^{th}

Chris Taylor
Tactical Computing Lab

RISC-V is making an inroads to commodity and HPC computing. This presentation provides an introduction to the current state of the RISC-V HPC environment. The presentation specifically introduces the RISC-V instruction set architecture, available hardware platforms, the state of HPC RISC-V software, venues to track RISC-V HPC related work, and additional information about the RISC-V foundation working groups, technical groups, and special interest groups.

Session Chairs

- Keynote 1, Pedro Valero
- Session 1 (Wed 10:30 – 12:00), Omri Mor
- Session 2 (Wed 1:00 – 2:30), Mathieu Faverge
- Session 3 (Wed 3:00 – 4:30), Christoph Junghans
- Keynote 2, Patrick Diehl
- Session 3 (Thu 10:30 to 12:00), Qinglei Cao
- Session 4 (Thu 1:00 to 2:30), Aurelien Bouteiller
- Session 5 (Thu 3:00 to 4:30), Thomas Herault
- Keynote 3, Thomas Herault
- Session 6 (Fr 10:30 to 12:30), Patrick Diehl

Talks

Note that blue names indicate virtual talks.

TBA

19^{th}
9:00 AM–10:00 AM

Alan Edelman
Massachusetts Institute of Technology

TBA

19^{th}
10:30 AM–11:00 AM

Contemplating a Lightweight Communication Interface for Asynchronous Many-Task Systems

Jiakun Yan
University of Illinois Urbana-Champaign

MPI has been around for 30 years. Mainstream MPI applications feature coarse-grained, collective-style communication, no or limited overlap within a single global synchronization step, and single-thread communication. They have driven the MPI communitys optimization direction. Unfortunately, AMT systems communicate in a different manner: multiple threads can launch communication; messages are mostly point-to-point and fine-grained; there can be a large number of pending communication requests at the same time and no global synchronization points.

The Lightweight Communication Interface (LCI) is an experimental communication library aiming to push for efficient multithreaded irregular communication support. It has been integrated into two AMTs (HPX and PaRSEC) and has shown significant performance improvement in microbenchmarks and real-world applications at large scales. In this talk, we will present the newly developed LCI version 2 interface based on C++. It has the following main features (a) a simple but expressive interface to unify common communication primitives and completion mechanisms (b) a three-level resource hierarchy for library interoperating, multithreaded performance isolation, and large message multi-channel aggregation (c) explicit progress control and optional memory registration interface (d) a set of compilation and runtime variables to customize runtime behavior. We will also talk about how this interface maps to low-level network functionality and runtime behavior and how it fits into high-level AMT communication abstraction.

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

19^{th}
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.

19^{th}
11:30 AM–12:00 AM

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

19^{th}
1:00 PM–1:30 PM

Querol Vicenc
Barcelona Supercomputing Center

The OpenMP 6.0 standard introduces the Free Agents feature, which supports a more dynamic execution model to improve application malleability and resource utilization. With Free Agents, when a thread is idle within a parallel region, it can be dynamically reused to execute tasks from another parallel region. However, implementing this model within production-grade runtimes like LLVM libomp poses performance and complexity challenges due to its departure from conventional 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 demonstrate that libompv achieves performance on par with the original libomp runtime while offering increased flexibility and functionality.

19th
1:30 PM–2:00 PM

Futures in Task Graphs Extending Taskflow With Dynamic Data Dependencies

Rudiger 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

19^{th}
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

Mia Reitz
University of Kassel

Modern supercomputers nowadays consist of millions of compute cores. This growing number increases the likelihood of process failures, making fault tolerant programs essential.

Traditionally, fault tolerance is realized using Checkpoint/Restart (C/R), where process states are periodically saved to disk, and a collective restart is performed after failure. This approach is transparent to application programmers, but incurs a high running time overhead.

While C/R is a general-purpose technique, this paper deals with a specific fault tolerance technique for Asynchronous Many-Task (AMT) programs on clusters. In AMT, the computation is divided into tasks that are processed by worker processes running on different cluster nodes. For load balancing, the workers may employ cooperative or coordinated work stealing, where idle workers steal tasks from others.

AMT is well-amenable to fault tolerance, because the runtime may save the clearly defined task interfaces instead of process states. This approach, called task-level checkpointing (TC), is more efficient than C/R, and transparent to application programmers as well. So-far, TC has only been proposed for independent and nested fork-join tasks, which impose strict rules for task communication, and for cooperative work stealing. Future-based cooperation (FBC) is an increasingly popular and less strict variant of AMT, where tasks communicate through futures. Further, a recent study with independent tasks reported significant performance gains from deploying coordinated instead of cooperative work stealing.

This paper proposes a TC scheme for FBC programs with coordinated work stealing. In first experiments, we observed low running time overheads.

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

19^{th}
3:30 PM–4:00 PM

Anthony Cabrera
Oak Ridge National Laboratory

Heterogeneous systems are ubiquitous and predicted to host more diverse architectures including quantum computing hardware. This short paper strives to design an asynchronous task-based runtime solution that can encapsulate both classical and quantum computing environments in the heterogenous execution paradigm by exploring various integration possibilities between the task-based runtime IRIS, the quantum programming framework XACC, and the Quantum Intermediate Representation Execution Engine (QIR-EE). This could be made possible by utilizing task-based kernels designed to generate optimized codes via the Multi-Level Intermediate Representation (MLIR) which can then in principle be lowered to QIR. The need for asynchronous task-based execution is motivated by examples that require the co-existence of classical and quantum computing hardware. We hope to further highlight challenges that we would need to overcome to make such a solution.

20^{th}
9:00 AM–10:00 AM

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

20^{th}
10:30 AM–11:00 AM

Jed Brown
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.

20^{th}
11:00 AM–11:30 AM

Comparing Kernel Fusion Strategies with SYCL

Gregor Daiss
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)

20^{th}
11:30 AM–12:00 AM

Jonas Posner
University of Kassel

Dynamic resource management allows programs running on supercomputers to adjust their number of compute node allocations at runtime. This dynamism offers potential improvements in both individual program efficiency and overall system 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. Moreover, the development of flexible programs introduces significant increased programming complexity compared to static programs.

While recent research improved MPI's resource flexibility, significant challenges in programmability remain. In addition, MPI-based solutions typically focus on iterative parallelization, which limits their effectiveness for irregular and dynamic workloads.

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

In this work, we compare the MPI proposal "Dynamic Processes with PSets (DPP)" and the APGAS+GLB AMT runtime system. We implement benchmarks in both environments to evaluate programmability and performance. We conduct experiments on 16 nodes to evaluate static and dynamic programs coupled with custom prototype resource managers. Results show that APGAS+GLB facilitates programmer productivity by transparently providing both load balancing and resource flexibility. In contrast, DPP achieves superior performance in handling node changes, but at the cost of increased programming complexity.

20^{th}
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

20^{th}
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.

20^{th}
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 static timing analysis and data similarity search. However, existing solutions only deal with certain types of applications in which data dependencies exist between preceding data and succeeding data in a forward direction. Some applications, such as video encoding, exhibit data dependencies in both forward and backward directions and cannot be processed with existing solutions. To address the limitation, we introduce a token dependency awareness pipeline framework. Our framework associates each data element with a token as its identifier, supports explicit definitions of forward and backward token dependencies with an expressive programming model, resolves dependencies 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 dependencies. For example, our framework is 11% faster than PARSEC's implementation in x.264 video encoding applications.

GPXPy: The Gaussian Process Python Library built on Asynchronous Tasks

20^{th}
3:00 PM–3:30 PM

Maksim Helmann
University of Toronto

Python is the de-facto language for software development in artificial intelligence (AI). Commonly used libraries, like PyTorch and TensorFlow, rely on parallelization built into their BLAS backends to achieve speed-up on CPUs. However, only applying parallelization in a low-level backend can lead to performance and scaling degradation. In this work, we propose a novel way of binding task-based C++ code built on the asynchronous runtime model HPX to a high-level Python API. We develop a parallel Gaussian Process (GP) library as an application. The resulting Python library GPXPy combines the ease of use of commonly available GP libraries with the performance and scalability of asynchronous task execution. We observe almost no overhead when binding the asynchronous HPX code using pybind. In addition, GPXPy shows good scaling up to 64 cores on an AMD EPYC 7742 CPU. Compared to GPyTorch and GPflow, our library achieves a prediction speed-up of factor 10.5 and 16.5, respectively.s Even compared to GPyTorch with LOVE, which rapidly approximates the predictive covariance matrix, our approach shows a speed-up of up to factor 2.8. These results showcase the potential of using asynchronous tasks within Python-based AI applications.

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

20^{th}
3:30 PM–4:00 PM

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.

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

21^{th}
9:00 AM–10:00 AM

Hatem Ltaief
King Abdullah University of Science and Technology

TBA

21^{th}
10:30 AM–11:00 AM

Scalable Block-Sparse Matrix Multiplication Using Template Task Graphs

Joseph Schuchart
Stony Brook University

Block-sparse matrix-matrix multiplication is an important operation in many scientific fields, including quantum physics and general tensor operations. Template Task Graph (TTG) is a programming model for flow graph-based composition of high-performance algorithms executable on distributed heterogeneous computer platforms. The TTG API abstracts out the details of the underlying task and data flow runtime. This paper describes the implementation of a distributed block-sparse matrix multiplication in the Template Task Graph (TTG) programming model. We will discuss the structure of the task graph and the operations involved, including strategies for controlling the distribution of matrix blocks. We further provide a performance evaluation of our implementation.

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

UT Student Union Room (Room 169)
1502 Cumberland Ave, Knoxville, TN 37916

Hotel

Hilton Knoxville, 501 W Church Ave, Knoxville, TN 37902
(865) 523-2300

Banquet

Calhoun's On The River, 400 Neyland Dr, Knoxville, TN 37902
(865) 673-3355

Restaurants

Walking distance

Knoxville's market square and Gay street offer a wide selection of restaurants and bars so this list is far from exhaustive:

- Tupelo Honey – Southern comfort food with a creative twist, plus craft beers & cocktails.: 1 Market Square, Knoxville, TN 37902 (865-522-0004)
- Cafe4 – Refined Southern classics, wine & beer in a comfortable dining room with a mezzanine coffee shop: 4 Market Square, Knoxville, TN 37902 (865-544-4144)
- Stock & Barrel – Stylish restaurant featuring thoughtfully sourced burgers & an extensive selection of bourbons: 35 Market Square, Knoxville, TN 37902 (865-766-2075)
- Vida – Vibrant restaurant in a historic bank, offering Latin dishes & a swanky cocktail lounge in a vault: The Holston, 531 S Gay St, Knoxville, TN 37902 (865-544-8564)

The Old City

A short walk away from the hotel lies Knoxville's Old City, a vibrant night life area with good food and bar choices:

- Kaizen – Popular place serving steamed buns, Asian dishes & draft beer in a chill space: 127 S Central St, Knoxville, TN 37902 (865-409-4444)
- Boyd's Jig & Reel: This cozy pub serving traditional meals & libations hosts live Scottish, Irish & Appalachian music: 101 S Central St, Knoxville, TN 37902 (865-247-7066)
- Lofty taproom doles out Mexican & Southern-inspired eats & draft beers: 100 Broadway SW, Knoxville, TN 37902 (865-999-5015)

Author Index