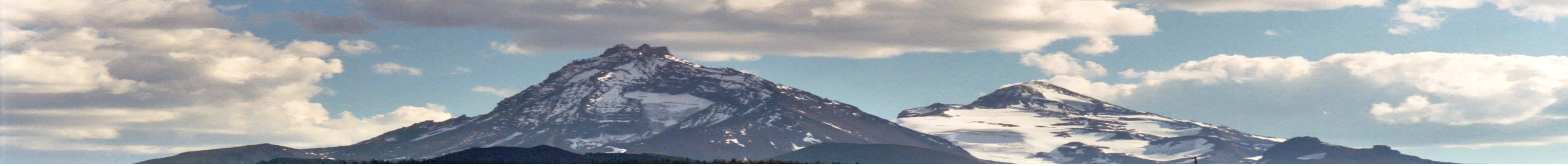


Observability, Monitoring, and In Situ Analytics in Exascale Applications

Dewi Yokelson[1], Oskar Lappi[2], Srinivasan Ramesh[5]
Miika Väisälä[3], Kevin Huck[1], Touko Puro[2]
Boyana Norris[1], Maarit Korpi-Lagg[2,4], Keijo Heljanko[2], Allen D. Malony[1]

University of Oregon
Eugene, Oregon, USA[1]

University of Helsinki
Helsinki, Finland[2]

Institute of Astronomy & Physics
Academia Sinica, Taiwan[3]

Aalto University
Espoo, Finland[4]

NVIDIA Corporation
Santa Clara, California, USA[5]

Motivation

- Heterogeneous HPC for exascale computing
  - Concerns for *performance variability* and *performance portability*
- Important to integrate HPC performance analysis technologies
  - Robust scalable and portable performance tools
  - Standard post-mortem analysis can miss interesting dynamics
- Growing interest in greater *observability* of HPC applications
  - Online observation (measurement) + monitoring + analytics
  - HPC applications are not generally designed to be observable
- Consider how to build performance observation support
  - Integrate TAU Performance System with Astaroth application
  - Design service-based observation, monitoring, analysis (SOMA)
  - Create SOMA prototype and test with Astaroth on LUMI

Opportunity

- Fulbright-Nokia Distinguished Chair
  - Study performance of HPC scientific and big data applications on LUMI supercomputer
  - University of Helsinki (Prof. Keijo Heljanko)
  - CSC–IT Center for Science (TAU on LUMI)
- Collaboration with Prof. Maarit Korpi-Lagg
  - Department of Computer Science, Aalto University
  - Developing LUMI-G application (LUMI-G pilot)
    - magnetized astrophysical plasmas based on Astaroth
  - Began work together in January 2023
- Paper reports research progress thus far

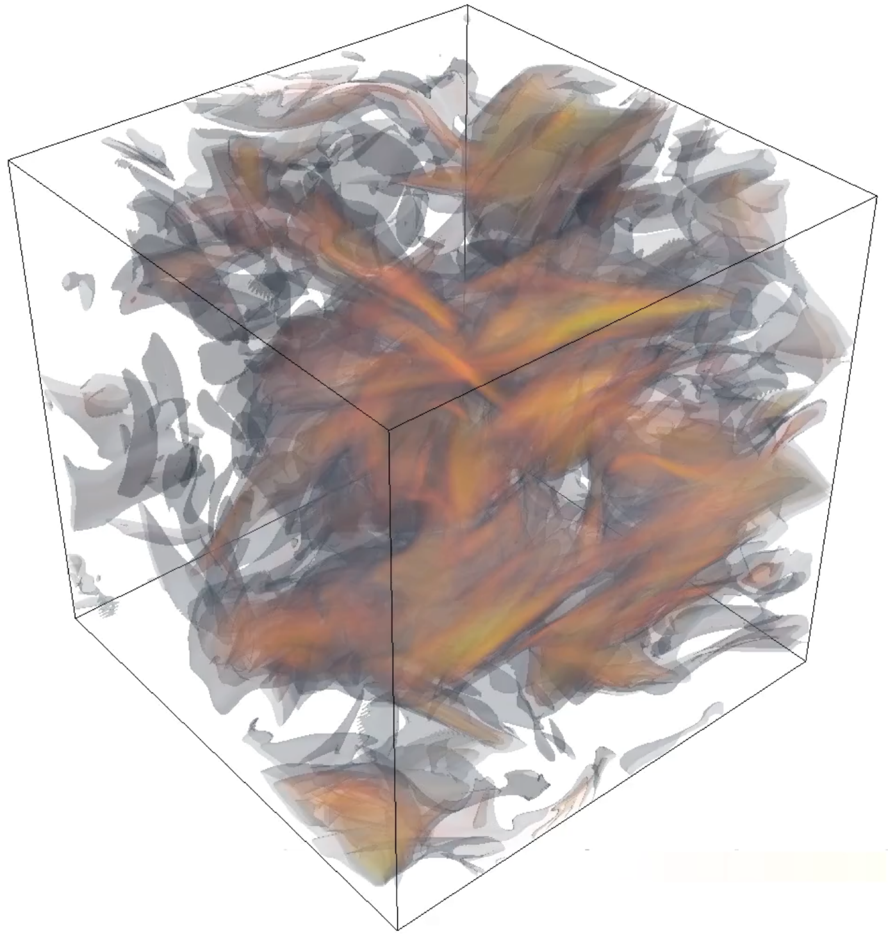
FULBRIGHT FINLAND FOUNDATION

NOKIA

CSC

Introduction to Astaroth

- Multi-GPU high-order stencil library
  - Host-level interface (C, C++, Fortran)
  - Domain specific language (DSL) for physics
  - Multi-GPU MPI (GPUDirect RDMA)
- Magnetized astrophysical plasmas framework
  - Magnetohydrodynamics (MHD) regime
- Astaroth operates in SPMD manner
  - One MPI rank per GPU device
  - LUMI-G node has 4x AMD MI250x GPUs (8 “devices”)
  - No explicit CPU multithreading
  - Logical tasking and scheduling of GPU kernels
- Astaroth-based LUMI-G pilot “hero run”
  - 16K devices (2K nodes) for 12 hours



Magnetic field lines (dark red streamlines) and intensity (volume-rendered colours) in a dynamo-active Astaroth simulation

Astaroth Motivation for In Situ

- Astaroth allows unprecedented resolution
  - Analysis and movement of data major bottleneck
  - Need more in situ data reduction and analytics
- Framework solves PDEs for MHD
  - Continuity, angular momentum, entropy, induction
  - Conditions in astrophysical plasmas
  - Non-conservative versus flux-conserving
- Methods used do not necessarily guarantee conserved quantities are accurate to the machine precision
  - Conserved up to the discretization error
  - However, requires constant monitoring of the conserved quantities during execution

SPMD HPC Applications and Monitoring

- SPMD is the dominant HPC programming model
  - Shared memory parallelism (e.g., OpenMP)
  - Distributed memory parallelism (e.g., MPI)
  - Accelerator parallelism (e.g., CUDA, HIP)
- Performance tools mainly developed for SPMD applications
- Integrating monitoring infrastructure is problematic
  - Difficult to express monitor operations in SPMD model
  - Requires asynchronous execution and dynamic resource use
  - Encounters implementation restrictions with MPI or systems
- Limits ability to take advantage of underutilized capacity

Special Case of Free Cores

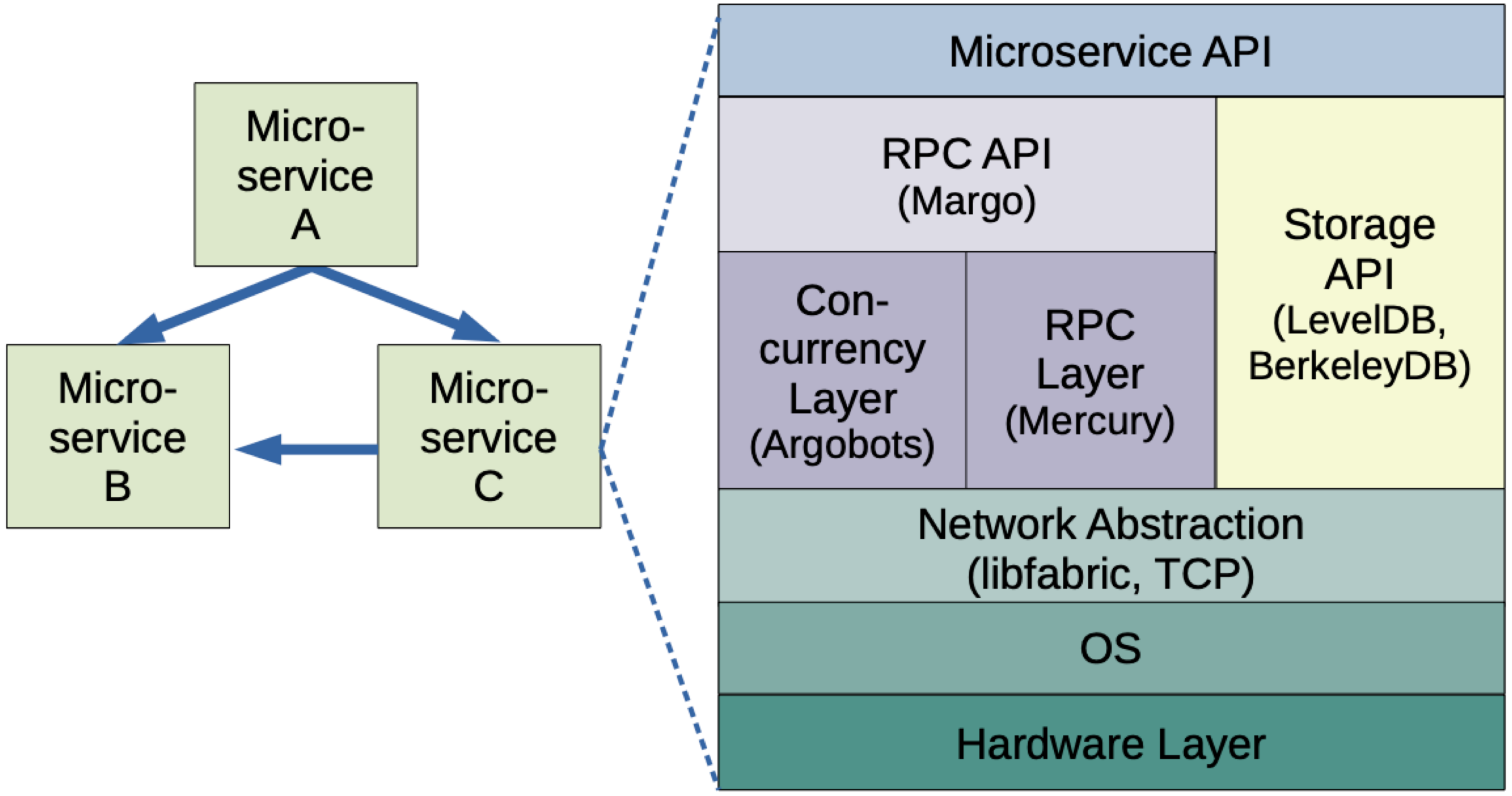
- Heterogeneous accelerated-node applications emphasize GPU use and could leave CPU cores idle
- How to take advantage for monitoring purposes
- Consider MPI application with following attributes:
  - ***R*** total ranks on *N* nodes (***r*=*R*/*N*** ranks per node)
  - ***C*** CPU cores per node (***c*** cores unused by application)
  - Desire ***M*** total monitoring processes (***m*=*M*/*N*** per node)
- How to create monitoring processes and configure them?
  - Solution A: Splitting of *MPI_Comm_World*
  - Solution B: Using MPI MPMD support (if available)
  - Solution C: Running separate programs with job scheduler
- Different solutions have different tradeoffs

High-Performance Services

- Creating monitoring processes is only part of solution
- How to run monitor code and interact with application?
- HPC data services emerged for couple HPC codes
  - High-performance microservice technologies
  - Utilize interconnection technology and fabrics
- Mochi software stack for developing data services
  - Argonne National Lab (https://www.mcs.anl.gov/research/projects/mochi/)
  - Mochi used in HPC data and visualization services

Mochi Software Stack

- Mercury RPC library
  - High-performance
  - RDMA
- Argobots
  - Light-weight threading
  - High concurrency
- Margo
  - Programming abstraction for Mercy
- Thallium
  - Header-only C++ interface to Margo



Mochi

Prior Research with Mochi

- Investigate Mochi microservices for observability
- SYMBIOMON demonstrated monitor was possible
  - Deployed internally in Mochi
  - Not flexible enough for general purpose
- SERVIZ applied approach to in situ visualization
  - Highlighted data models in microservices
  - Utilized Conduit technology for visualization data
  - Not developed as a monitoring solution
- Seer in situ analysis with Jupiter frontend
- Colza elastic in situ visualization of HPC simulations

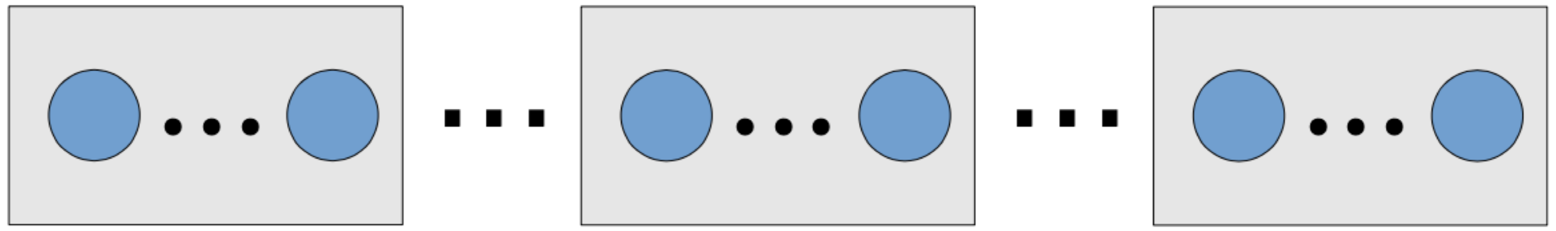
Conduit

- What data is sent and how it is represented?
- Conduit designed to simplify data description and sharing across HPC sim tools
  - Provides an API for data description
  - C, C++, Python, Fortran interfaces
- Hierarchical variant type call a *Node*
  - Capture and represent arbitrary nested data
- Use Conduit to represent performance data
  - TAU and APEX profiles
- Use Conduit to represent application diagnostic data

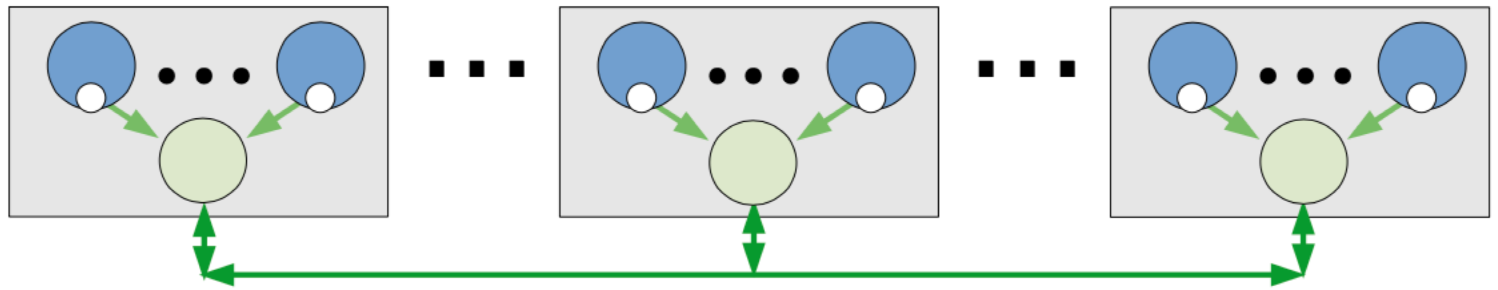
SOMA Framework

- Consider a service-based observation, monitoring, and analysis (SOMA) framework
- Implement using Mochi technology
  - Create *“collector” client* within an application rank
    - gathers performance data to send by RPC to monitoring layer
  - Create *“collector” service instance* to receive data
    - endpoint of RPC
  - Configure clients and service with application
    - Discovery and registration
- Use Conduit for performance and application data
- Develop SOMA programming stack and API

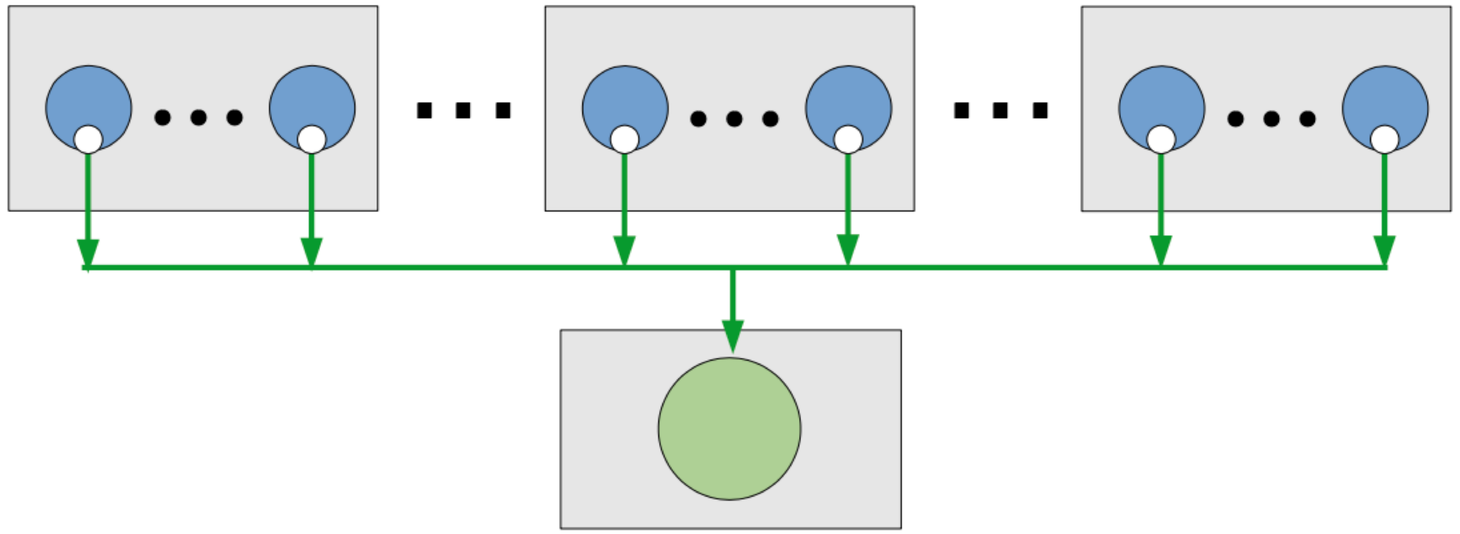
SOMA Configuration Examples



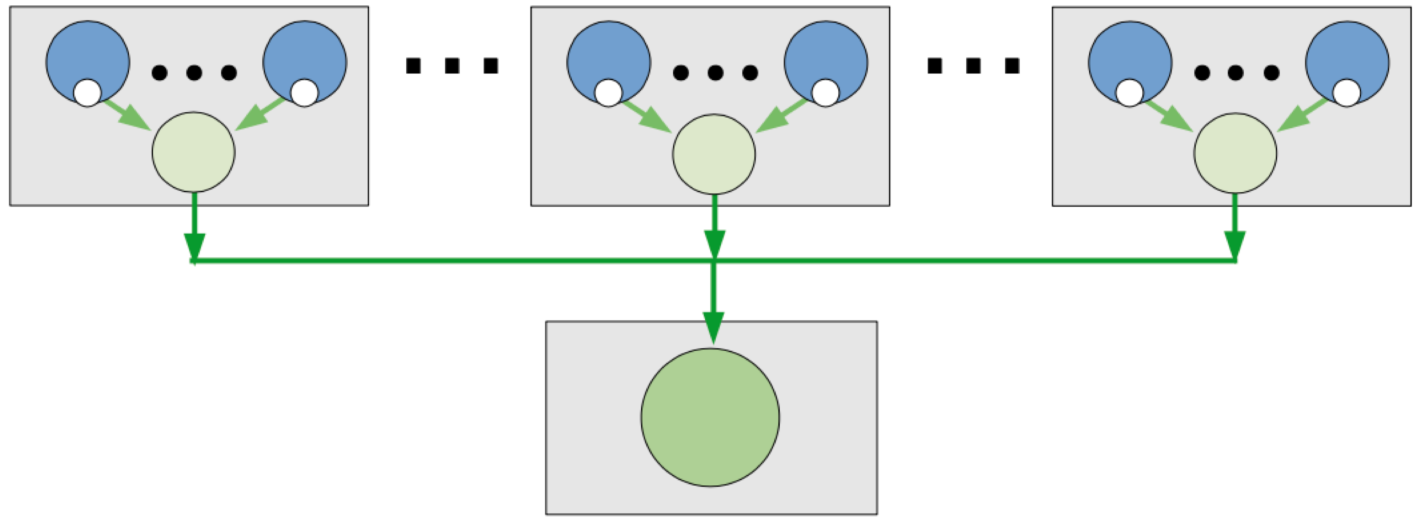
- Application MPI process (rank)
- Application compute node



- Collector client (within application process)
- Collector service instance (local, own process)



- Collector client (within application process)
- Collector service instance (global, own node)



- Collector client (within application process)
- Collector service instance (local, own process)
- Collector service instance (global, own node)

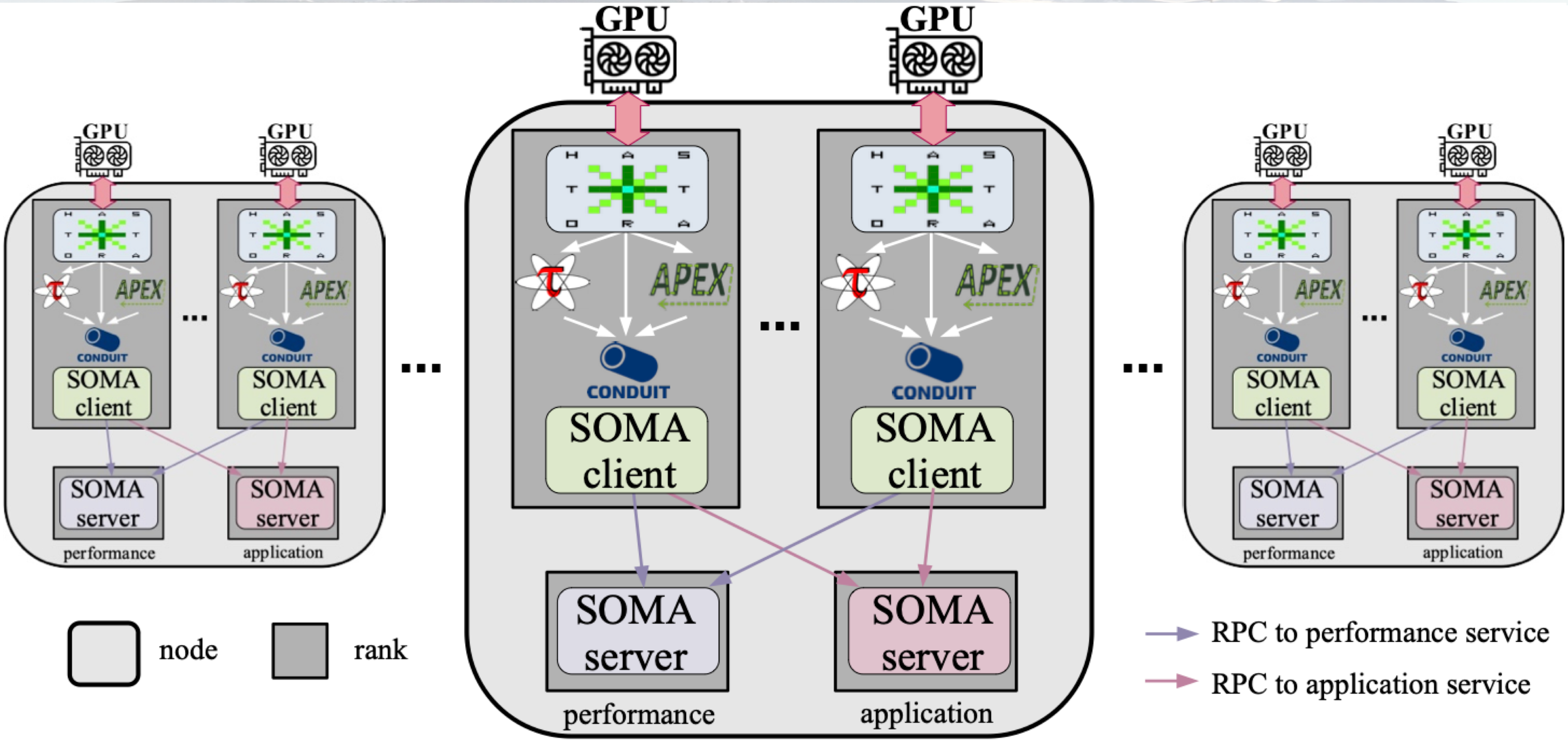
TAU Project at the University of Oregon

- Research and development effort spanning 30+ years
- Focus on parallel performance problems and technologies
- Performance problem solving framework for HPC research
  - Integrated, scalable, flexible, portable
  - Target all parallel programming / execution paradigms
- Integrated performance toolkit (TAU Performance System®)
  - Multi-level performance instrumentation
  - Flexible and configurable performance measurement
  - Widely-ported performance profiling / tracing system
  - Performance data management and data mining
  - Open source (BSD-style license)
- Broadly used for performance analysis and engineering in complex software, systems, applications

TAU Performance System

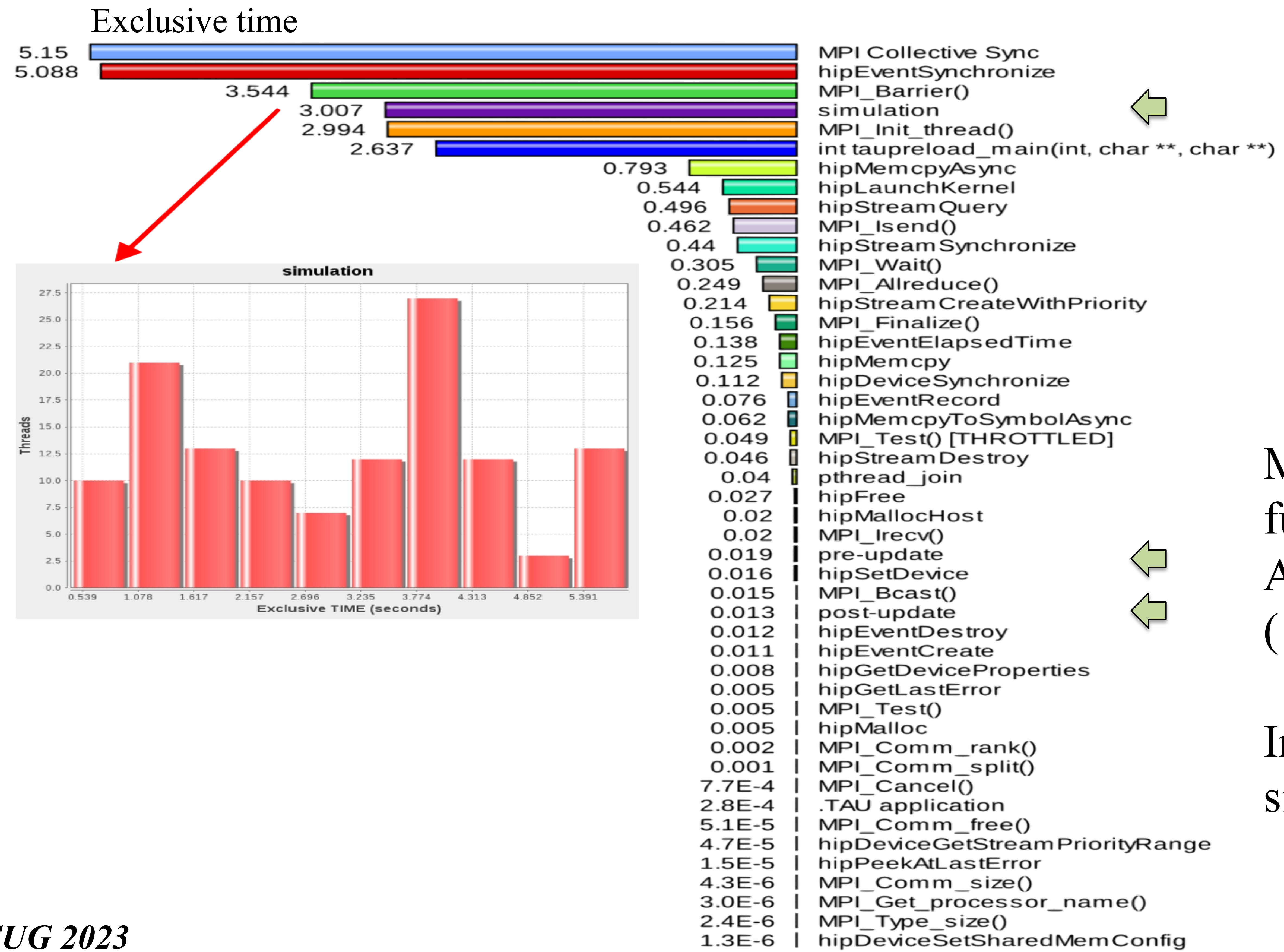
- Incorporates two performance toolkits
  - Each provides measurement and analysis support
  - ***TAU*** (Tuning and Analysis Utilities)
  - ***APEX*** (Autonomic Performance Environment for Exascale)
- Differ in respects to observation perspective
  - TAU: who is doing the "work" (per thread measurement)
  - APEX: what "work" (task) is done (per task measurement)
- Used individually or together

SOMA Framework with Astaroth



Use MPI communicator splitting approach

Astaroth Performance Analysis with TAU



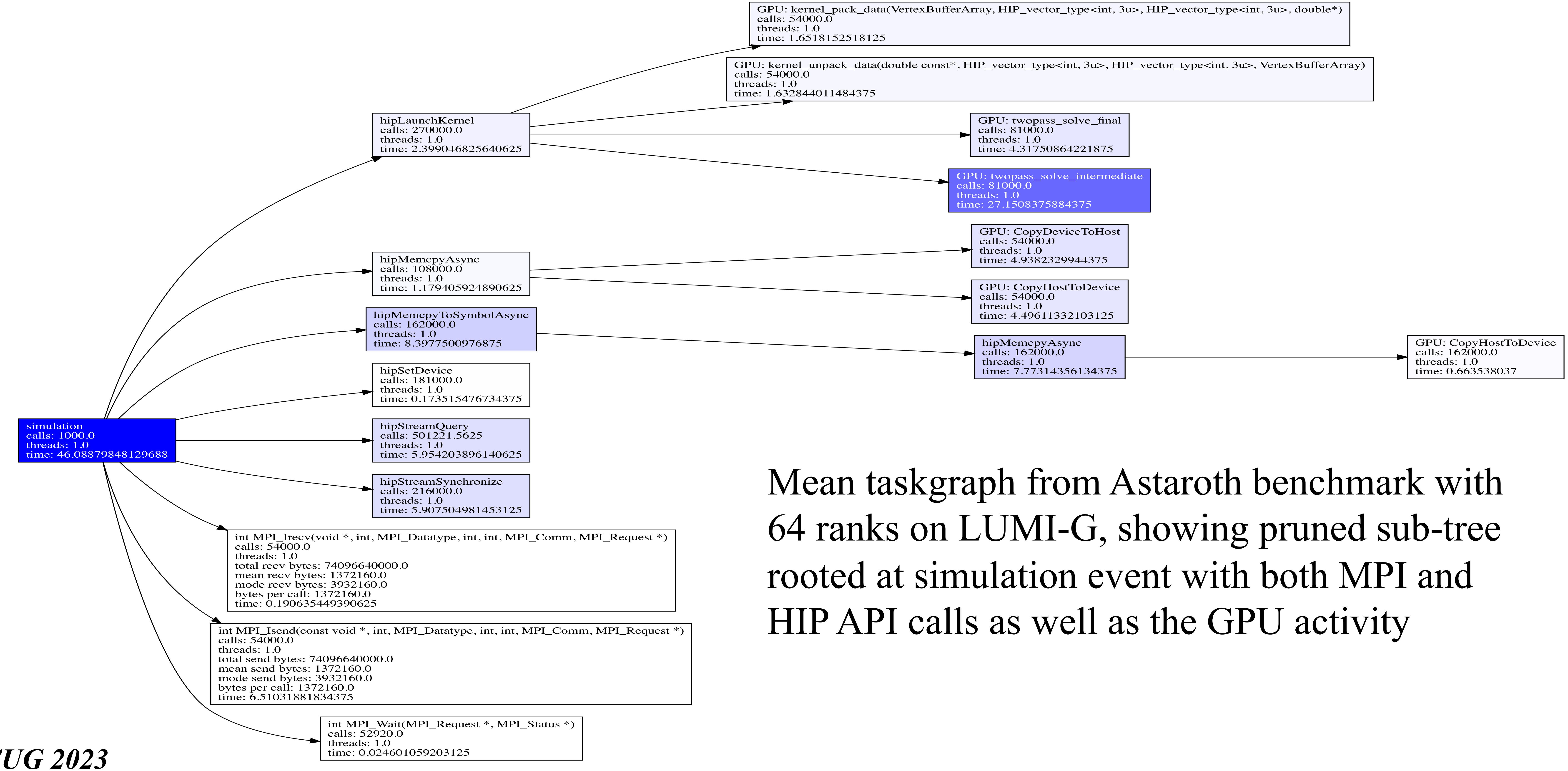
Application events (⇦) (instrument w/ Perfstubs):

- *pre-update*
- *simulation*
- *post-update*

Mean time spent per Astaroth function across all ranks for an Astaroth execution on LUMI-G (16-node, 128-GPU, 128 ranks)

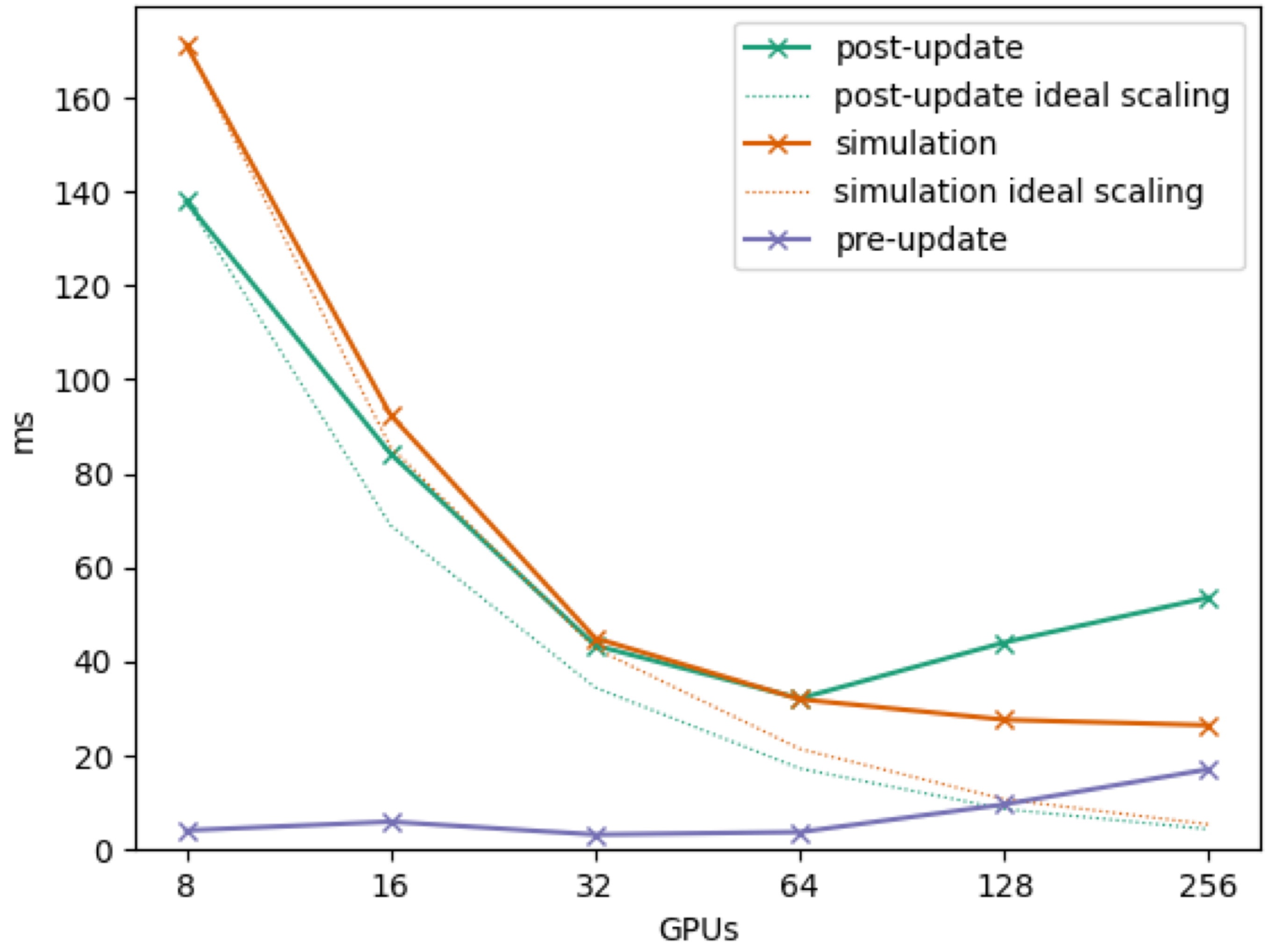
Inset shows the distribution of the simulation timer across ranks

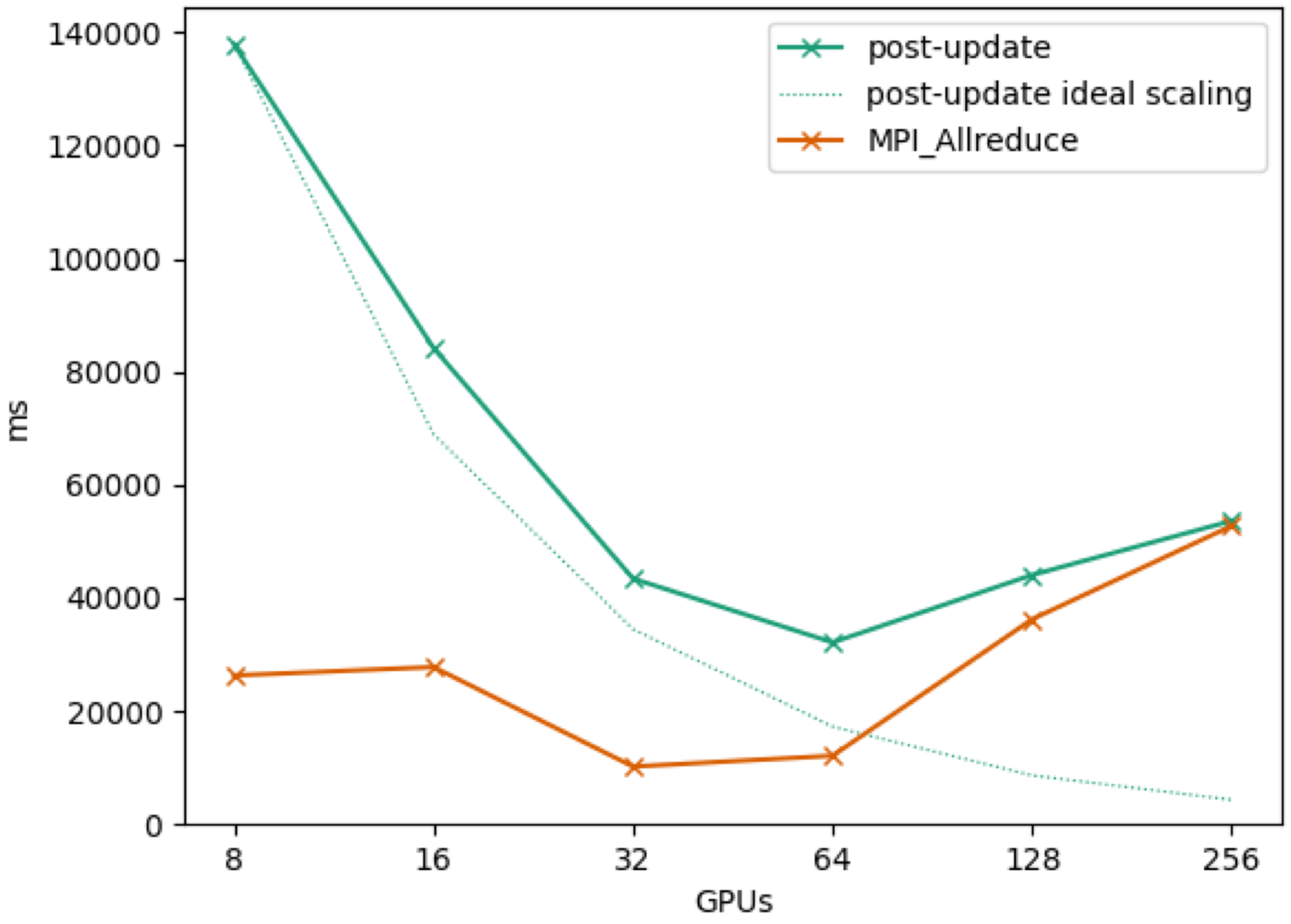
Astaroth Performance Analysis with Apex



Mean taskgraph from Astaroth benchmark with 64 ranks on LUMI-G, showing pruned sub-tree rooted at simulation event with both MPI and HIP API calls as well as the GPU activity

Astaroth Performance Scaling





- Scaling behavior of the simulation and GPU activity
- GPU kernel has near perfect scaling
  - Additional overheads in the timers
  - Not accounted for by MPI or HIP calls

Conduit Data Models for Astaroth Monitoring

TAU performance data

```
{
    TAU:
        MPI:
            Allreduce: fp64, fp64,
            MPI_Wait: fp64,
        Application Functions:
            Foo: fp64,
        Counters:
        MPI_Message_Sizes:
        ...
}
```

Astaroth diagnostic data

```
{
          pid: uint32,
     timestep: uint64,
simulation_time: fp64,
   local_mass: fp64,
      FIELD_1: {
       min: { value: fp32, location: [uint16] x 3 },
       max: { value: fp32, location: [uint16] x 3 },
       nan: { value: bool, location: [uint16] x 3 }
      },
      FIELD_2: { min: ..., max: ..., nan: ... },
      ...
}
```

- ❑ TAU performance data model is application agnostic
- ❑ SOMA also accepts any Conduit Node schema

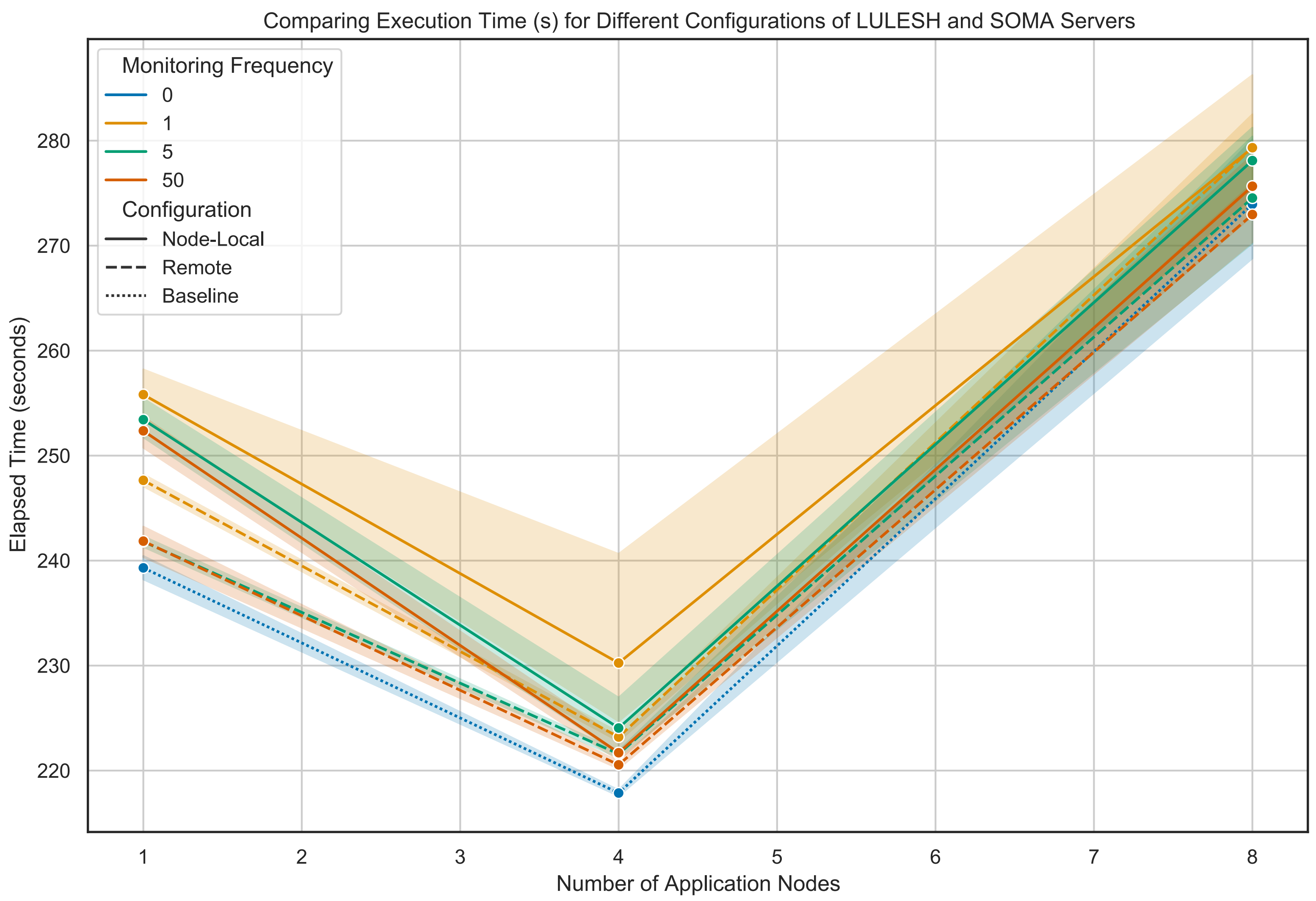
Monitoring Overhead Experiment Setup

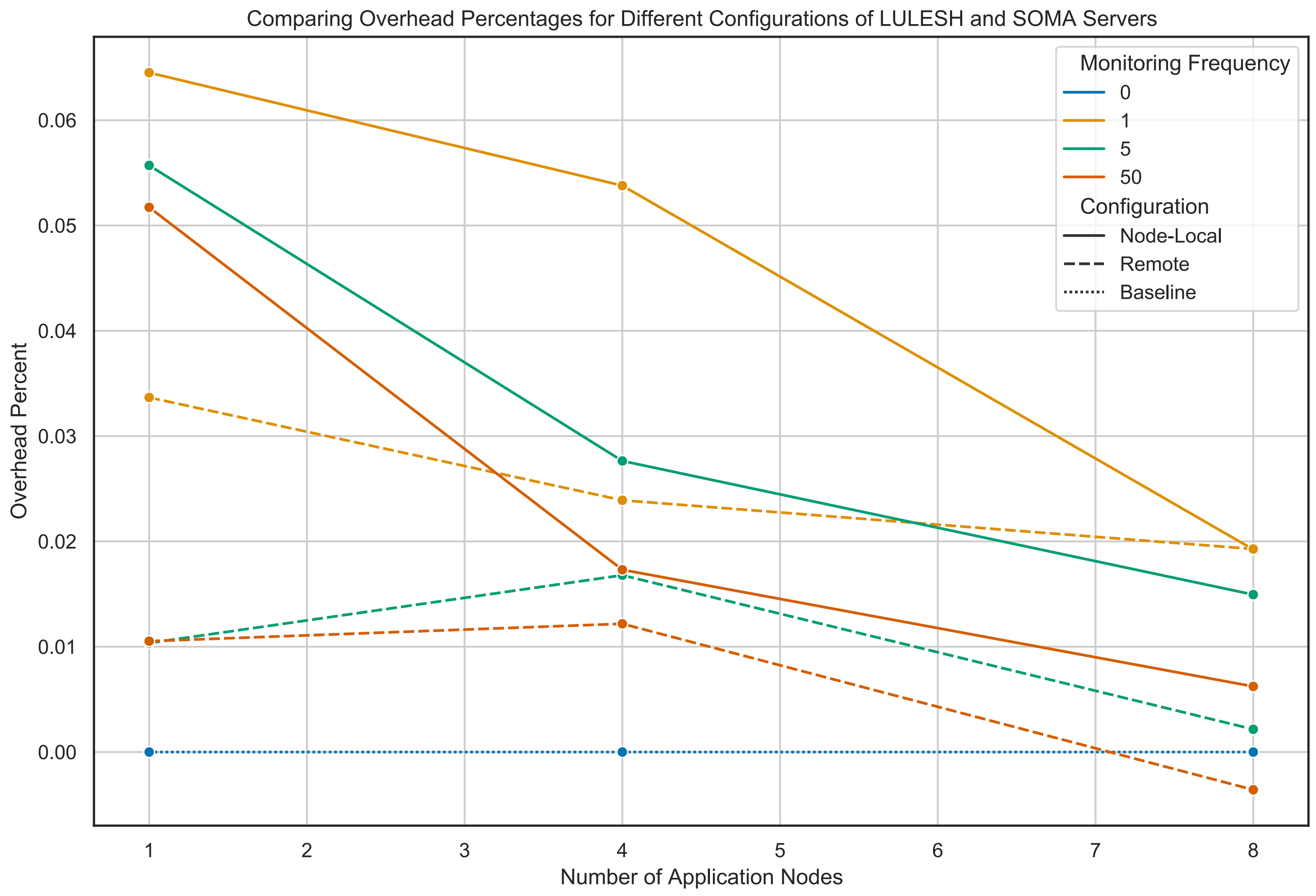
Name	System	CPU	Total CPU Cores	Memory (GB)	# GPUs	GPU Arch
LUMI-G	HPE Cray EX	AMD EPYC 7653	64	512	8	MI250X
MAHTI	Atos BullSequana XH2000	AMD Rome 7H12	128	256	4	NVIDIA A100

- LULESH
  - Mahti
  - 1,4,8 nodes
  - Scaled problem size (total elements) per rank
  - Varied monitoring frequency

- Astaroth
  - LUMI-G
  - 1,2,4,8,16 nodes (8,16,32,64,128 GPUs)
  - Increased global grid dimensions for strong scaling

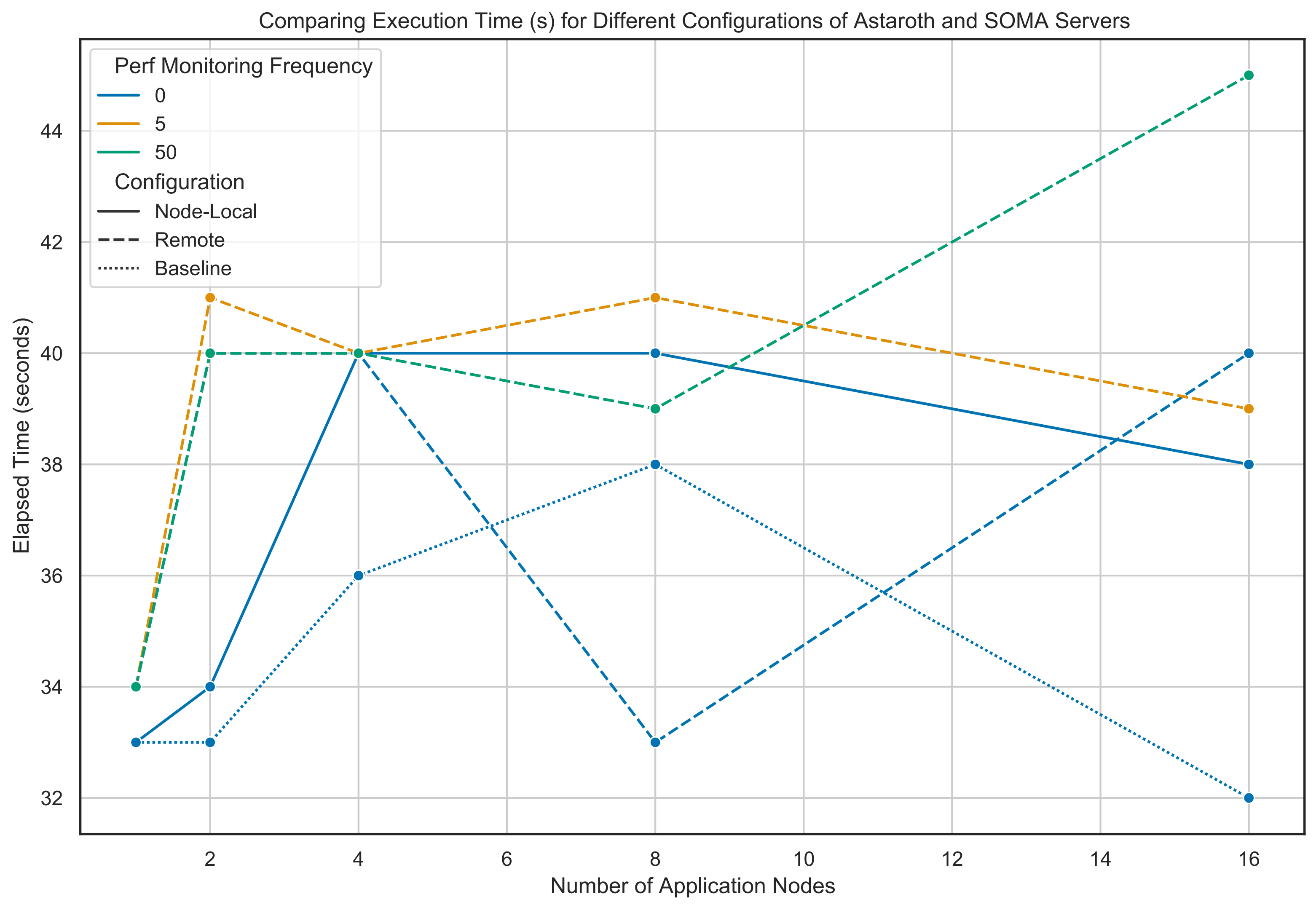
Monitoring Overhead - LULESH

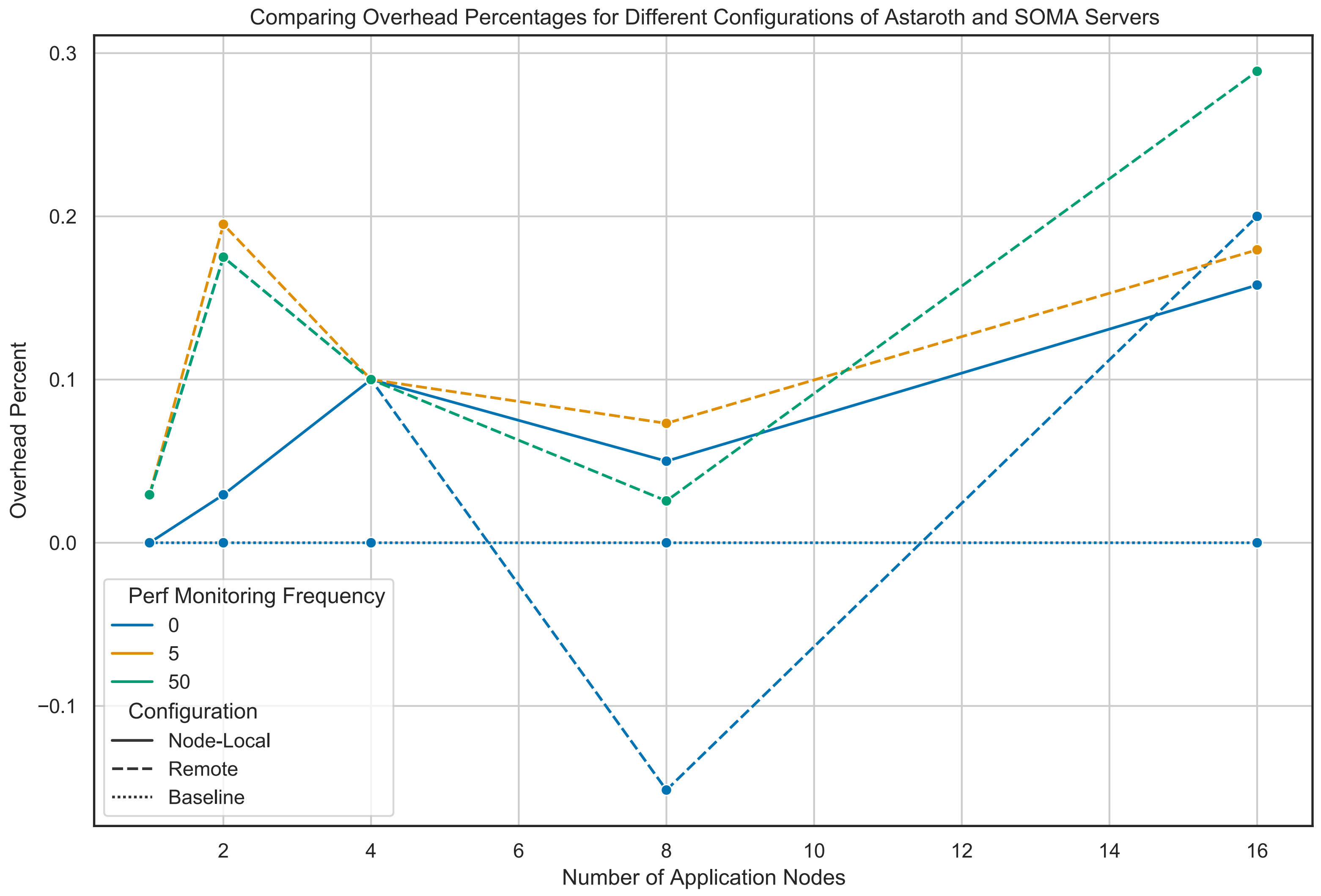




- Node-local and remote configurations of SOMA and LULESH
  - 64 ranks per node
- CSC Mahti (128 CPU cores per node)

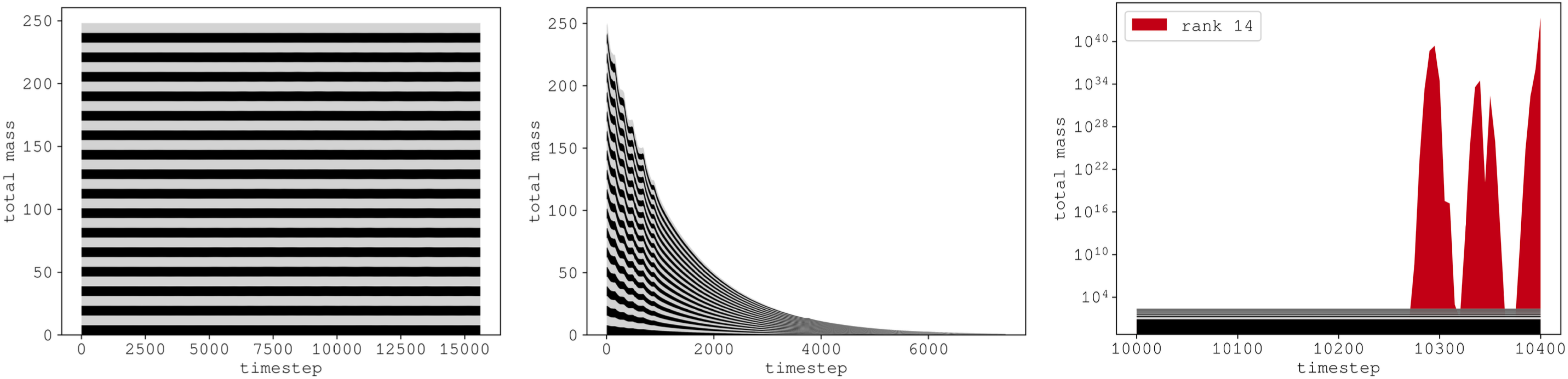
Monitoring Overhead - Astaroth





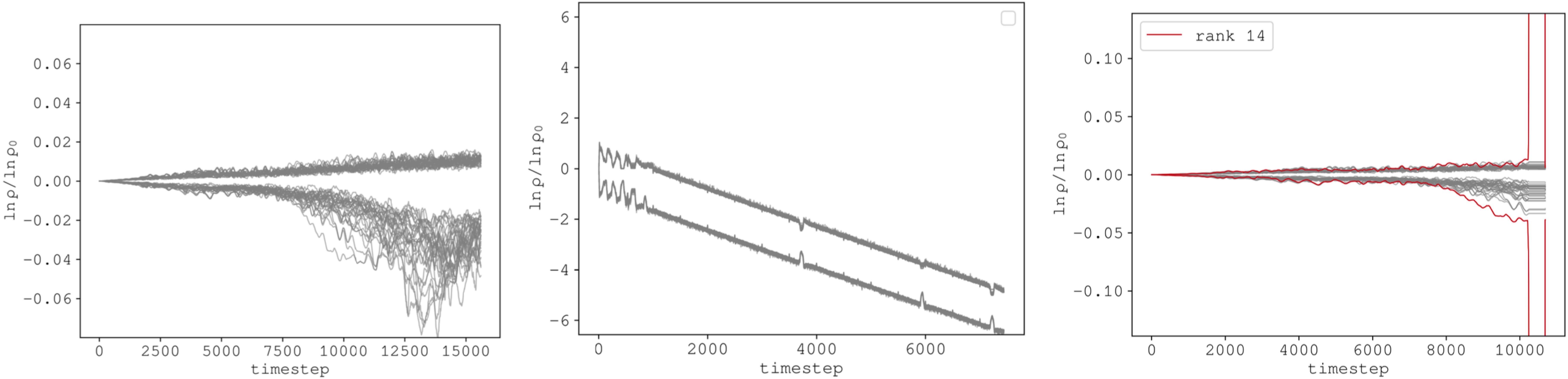
- Node-local and remote configurations of SOMA and Astaroth
  - 8 ranks per node
- CSC LUMI-G (64 CPU cores per node)

Astaroth Mass Conservation Diagnostics



- Shaded regions are the local mass in a rank over simulation
- Left: Mass is conserved (healthy simulation)
- Middle: Time step too large and mass disappears (bad simulation)
- Right: Viscosity too low resulting in numerical instability and mass gain (bad simulation)

Astaroth Density Extrema Diagnostics



- Density evolution same three simulations
  - Each gray curve is a min or max of a rank's density field over simulation
- Left: Density field develops naturally (no systematic error)
- Middle: Density systematically decreases in the system (bad)
- Right: Catastrophic mass increase due to a numerical instability

Conclusion

- Early results from a productive collaborative effort
- Successful integration of Astaroth with TAU/APEX
  - More to be done on kernel tasking measurements
- SOMA approach for observability proves promising
  - Look at different configurations and evaluate performance
  - Investigate use of asynchronous RPC
  - Additional opportunities to run at larger-scale
  - Potential for feedback between Astaroth and SOMA
- Leverage other Mochi-based infrastructure for Astaroth