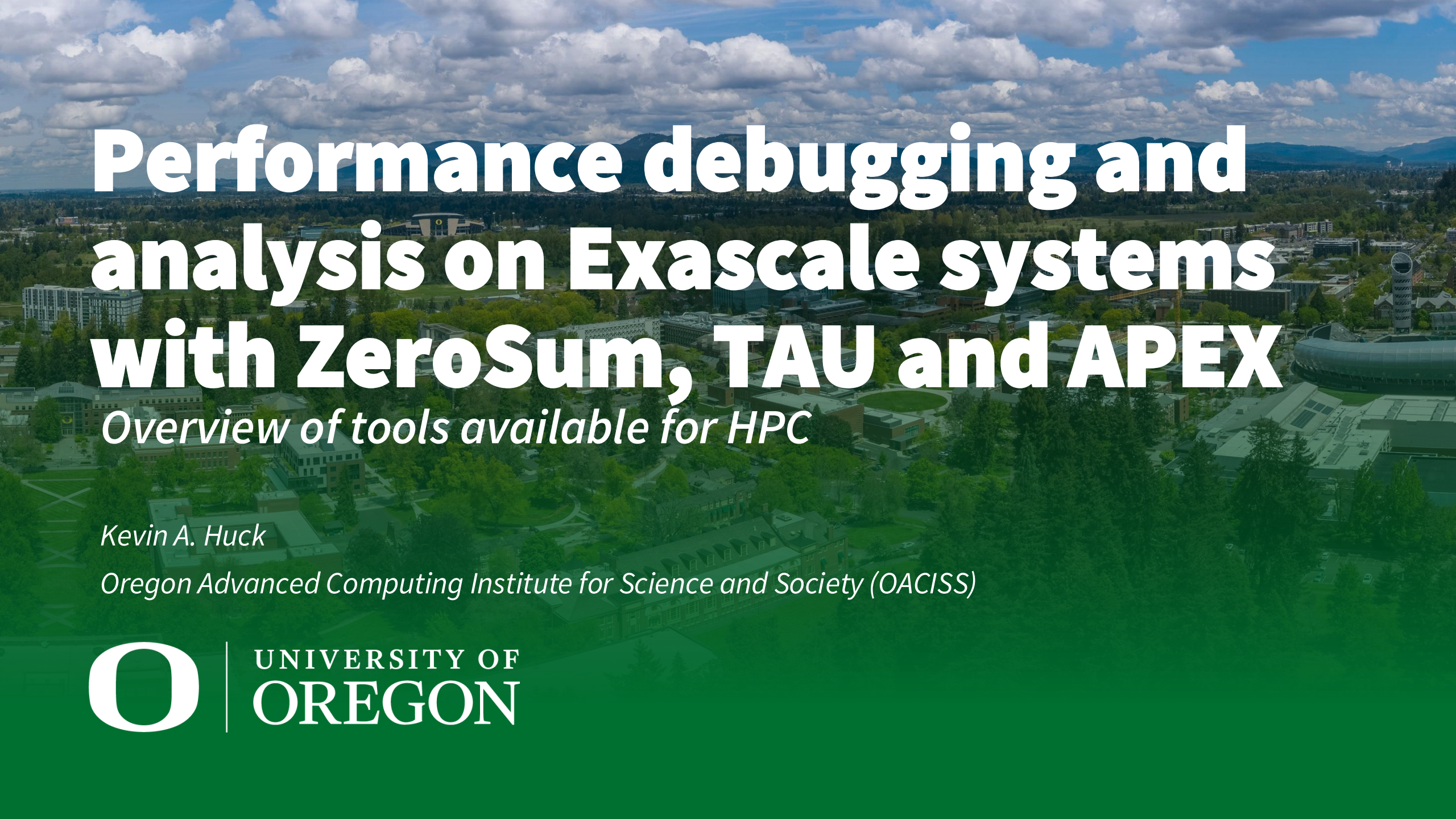


Performance debugging and analysis on Exascale systems with ZeroSum, TAU and APEX

Overview of tools available for HPC

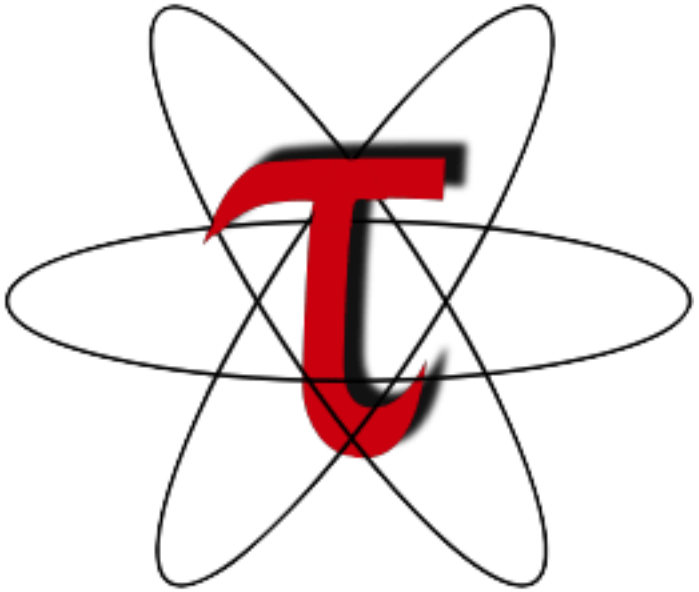
Kevin A. Huck

Oregon Advanced Computing Institute for Science and Society (OACISS)

UNIVERSITY OF OREGON

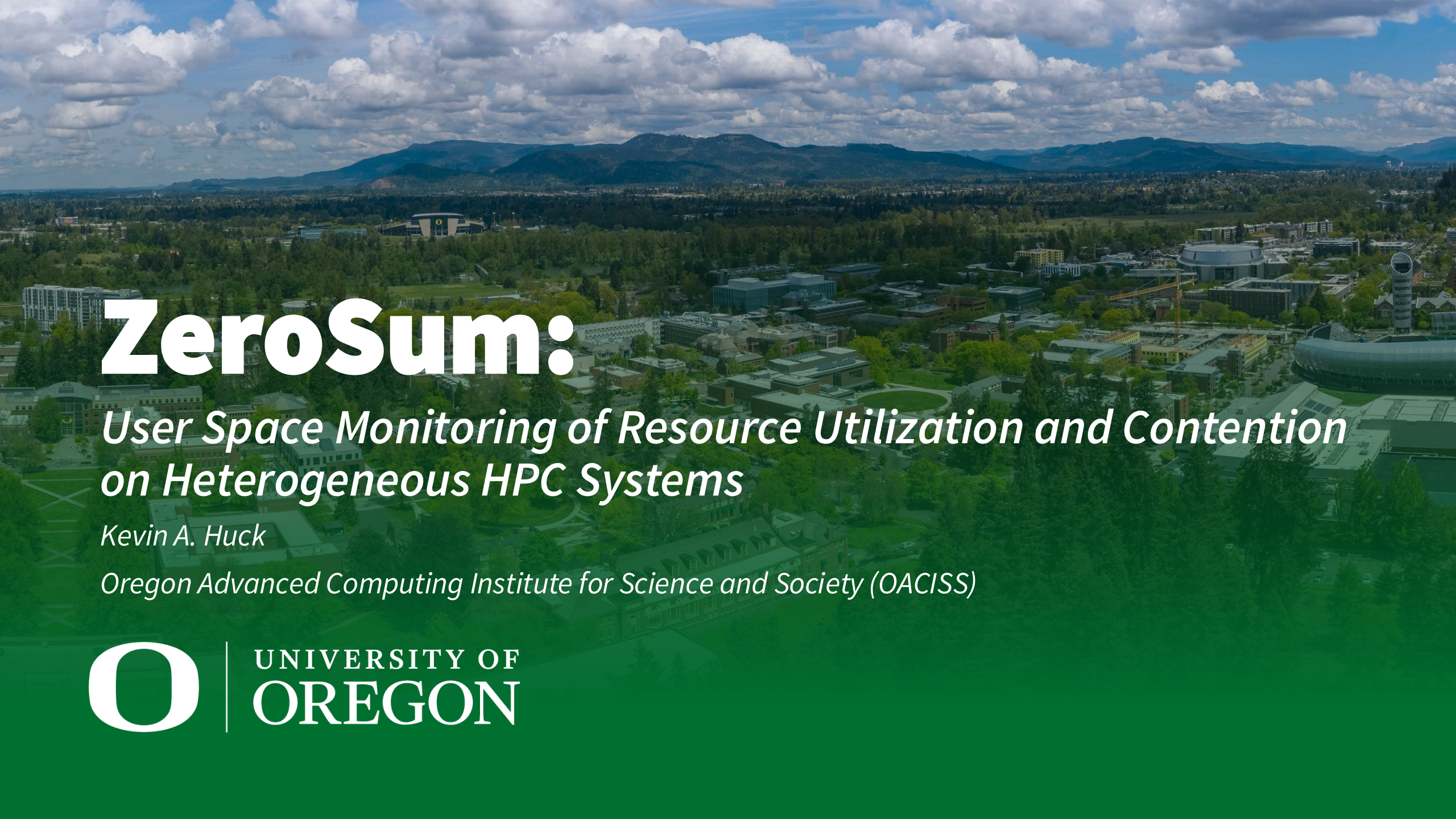
Who am I (Who are we)

- Senior Research Associate at University of Oregon
- OACISS: Oregon Advanced Computing Institute for Science and Society https://oaciss.uoregon.edu
- Performance Research Lab: focused on the analysis of applications on large-scale HPC systems



Projects of Interest today

- HPC Application monitoring with ZeroSum
- Application / library instrumentation with PerfStubs
- Performance measurement with TAU
- Performance measurement and runtime auto-tuning with APEX
- Focusing on Frontier today:
  - `module load tau`
  - `module load ums ums002/vanilla ; module load zerosum`
  - `module load ums ums002/vanilla ; module load apex`
  - `module load ums ums002/vanilla ; module load perfstubs`
- TAU, ZeroSum and APEX tied to specific MPI vendor/version, and ROCm version – see '`module avail apex`' for example



ZeroSum:

User Space Monitoring of Resource Utilization and Contention on Heterogeneous HPC Systems

Kevin A. Huck

Oregon Advanced Computing Institute for Science and Society (OACISS)

UNIVERSITY OF OREGON

Performance Analysis Perspective on Monitoring

Three main classes of performance optimizations:

1. Algorithmic replacement (usually a high level of difficulty)
   - E.g. replace $O(n^2)$ with O(nlogn)
   - Can involve data structure changes, new dependencies, major rewrites
2. Code optimization (usually a medium difficulty)
   - Improve cache reuse, reduce stalls (branching, instructions, I/O, etc)
3. Optimized launch configuration (low difficulty, high embarrassment potential)
   - Misconfiguration
   - Wrong assumptions from another system/application
   - Changes to system policies/defaults (e.g. reserved cores)

Behavioral Economics & Computer Science

"People aren't dumb – the world is hard."

– Richard Thaler, behavioral economist and co-author (with Cass Sunstein) of *Nudge: Improving Decisions About Health, Wealth, and Happiness*, 2008

People make bad choices for good reasons…

If you want to change behavior, change the defaults.

(not always possible – "one size fits most")

Motivation: Why do users (want to) monitor at runtime?

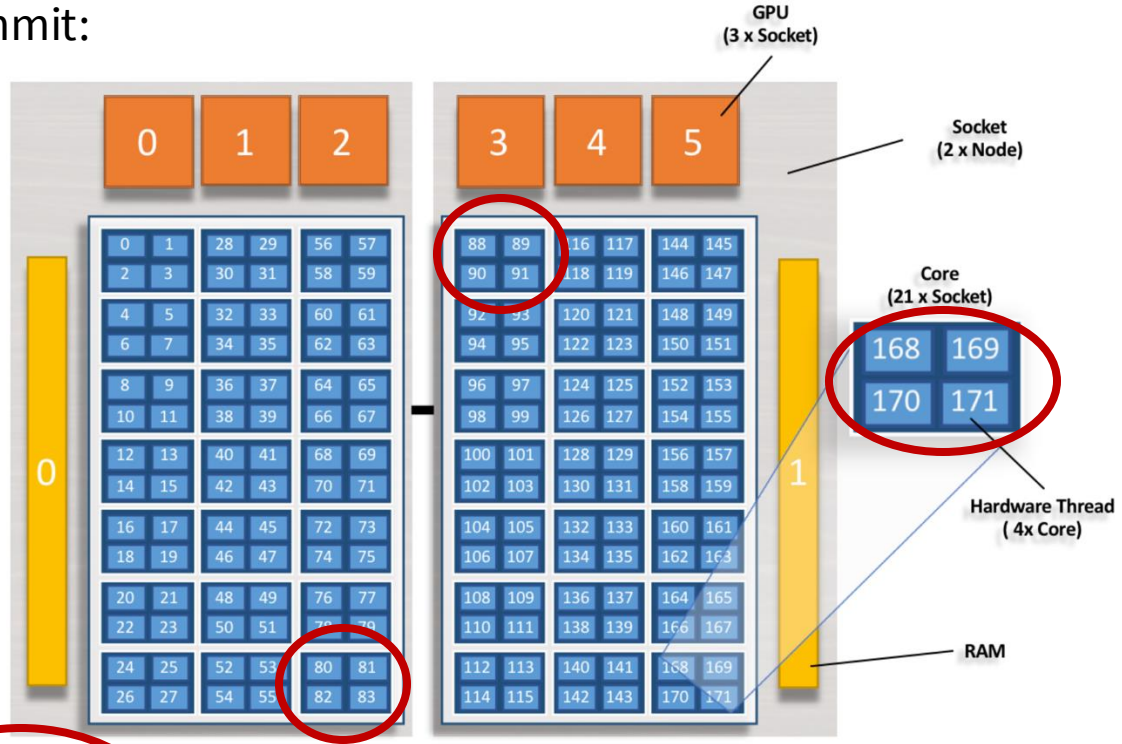
- **Sanity check** / curiosity / impatience (logging, essentially)
- **Check for misconfiguration**
- **Check for efficient utilization**
- Confirmation of expected hardware / operating system behavior
- **Identify cause of failure – deadlock, crash, stalls, etc.**
- Adaptation / computational steering / feedback & control
- ~~Identify system failures~~ (out of scope)

(Mis)Configuration – what could *possibly* go wrong?

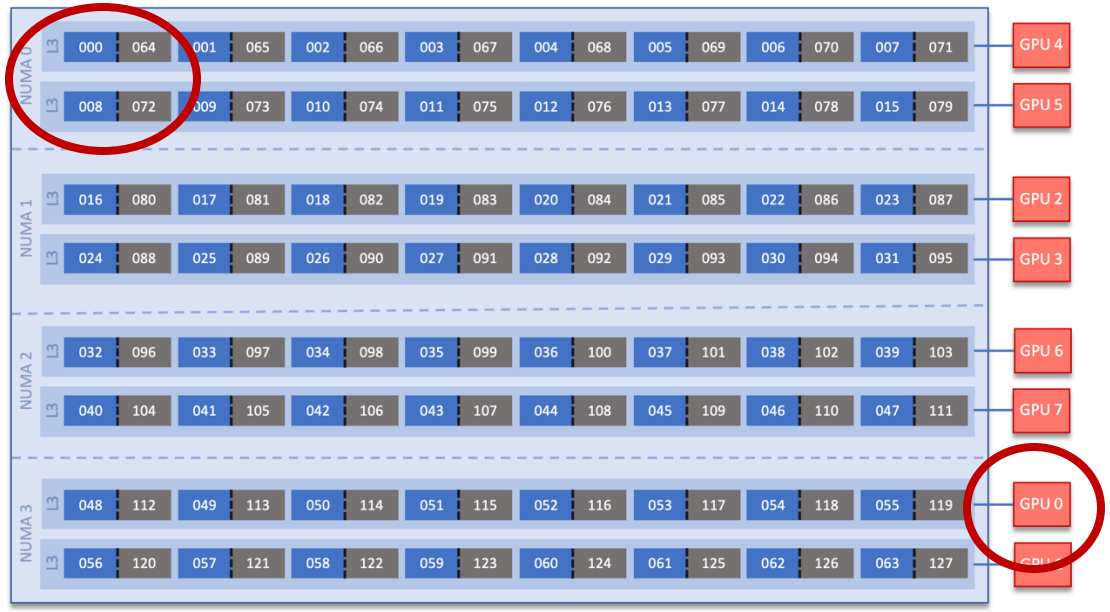
- Process placement:
  - Logical/physical mappings, resources assigned/constrained to each process
  - Are there reserved core(s) for system? What is the GPU mapping?
- Thread placement: Socket, NUMA domain, core, thread (HWT)
- Undersubscribing: Wasted hardware, energy, time (under-utilization)
  - Can provide better performance in some situations (memory-bound code)
- Oversubscribing: Increased contention with no realized benefit
- Imbalances
  - What is the communication frequency/volume between pairwise MPI ranks?
- Slurm/PBS/Alps/Torque/Flux are *complicated to use*
  - …especially when combined with MPI, OpenMP, GPUs, or other model settings

Although accurate, system documentation can be confusing

Summit:



Frontier:



- 1 node
- 2 sockets (grey)
- 42 physical cores* (dark blue)
- 168 hardware cores (light blue)
- 6 GPUs (orange)
- 2 Memory blocks (yellow)

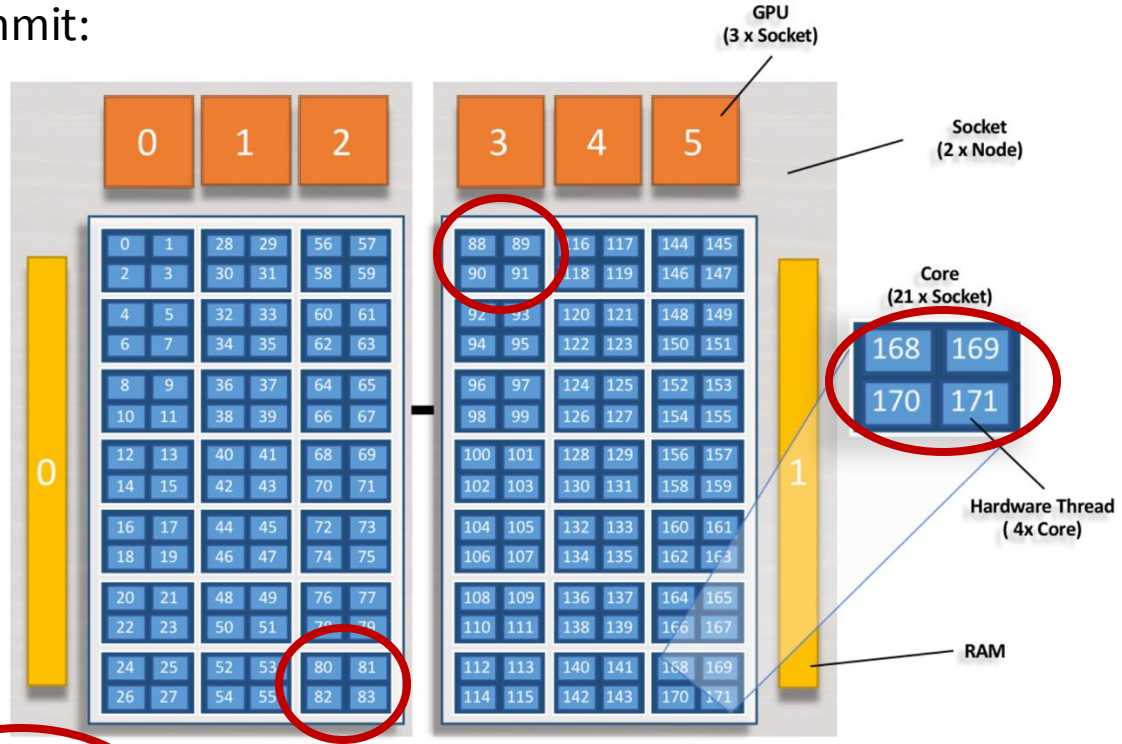
Core indexing, HWT indexing, GPU mapping not universal (or even intuitive)

***Core Isolation:** 1 core on each socket has been set aside for overhead and is not available for allocation through jsrun. The core has been omitted and is not shown in the above image.

Sources: https://docs.olcf.ornl.gov/systems/summit_user_guide.html , https://docs.olcf.ornl.gov/systems/frontier_user_guide.html

Although accurate, system documentation can be confusing

Summit:



- 1 node
- 2 sockets (grey)
- 42 physical cores* (dark blue)
- 168 hardware cores (light blue)
- 6 GPUs (orange)
- 2 Memory blocks (yellow)

Frontier:

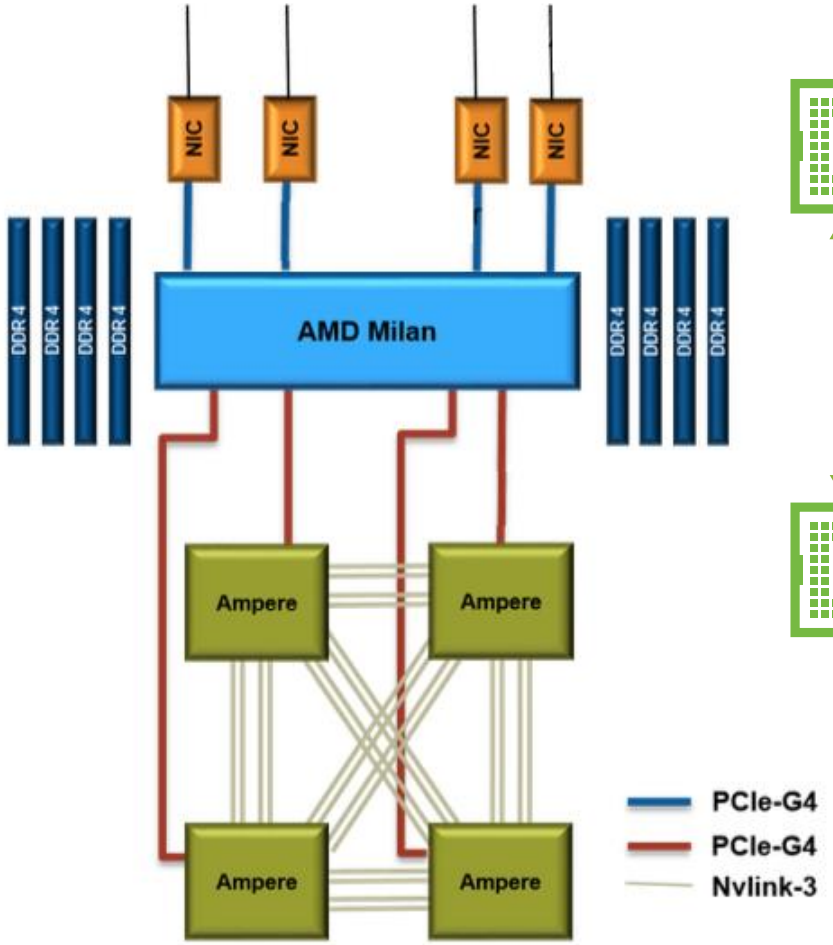
Modified default: system reserves 8 cores…but user controllable

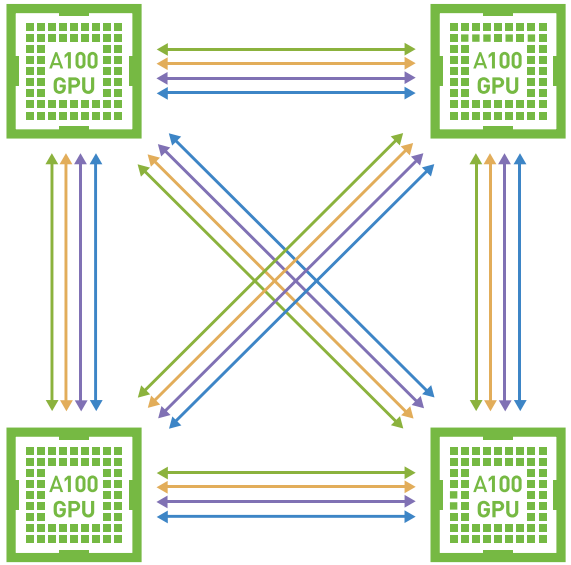
***Core Isolation:** 1 core on each socket has been set aside for overhead and is not available for allocation through jsrun. The core has been omitted and is not shown in the above image.

Sources: https://docs.olcf.ornl.gov/systems/summit_user_guide.html , https://docs.olcf.ornl.gov/systems/frontier_user_guide.html

More systems – information missing

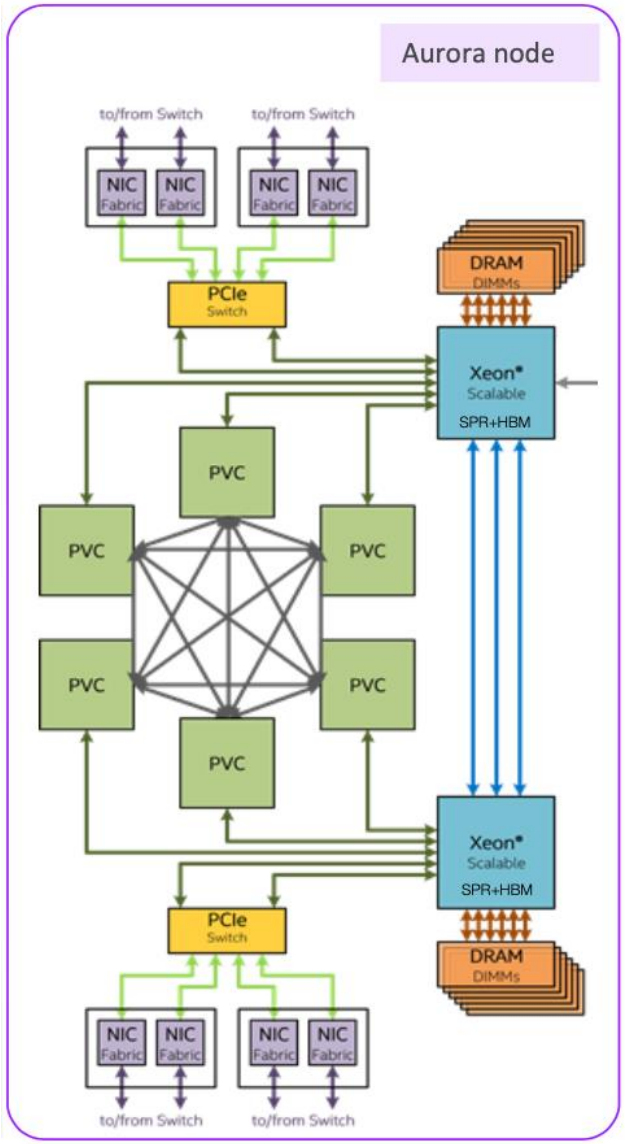
Perlmutter:





No physical layout information available in the documentation, no core or GPU indexing…

Aurora:



UNIVERSITY OF OREGON

Sources: https://docs.nersc.gov/systems/perlmutter/architecture/
https://extremecomputingtraining.anl.gov/wp-content/uploads/sites/96/2022/11/ATPESC-2022-Track-1-Talk-9-Muralidharan-Aurora.pdf

ZeroSum: User Space Monitoring of Resource Utilization and Contention

- Inspired by the *hello_jsub* program from Tom Papatheodore: https://code.ornl.gov/t4p/Hello_jsrun
- Why "ZeroSum"? – An benefit for one side results in an equivalent loss for the other
  - the fixed number of resources available in an allocation – no elasticity
  - the need to periodically use *some* resources to monitor
- Available on GitHub: https://github.com/UO-OACISS/zerosum
- Monitors application threads (LWP), CPU hardware (HWT), Memory, and GPU hardware for all processes, all nodes in the allocation
- Uses hwloc library to extract node topology information

ZeroSum Functionality

- ✓ **Detect the initial/changing configuration of the application**
- ❑ Evaluate the configuration to automatically detect misconfigurations
- ✓ Provide runtime feedback to the user that the program is progressing
- ✓ **Provide a report of how effectively the hardware was utilized**
- ✓ **Provide a report of how much contention was identified in the execution**
- ❑ Provide a way to expose the observed data to other tools that can perform computational steering / runtime optimization / reconfiguration

✓ Implemented
❑ Future work

How to use ZeroSum

- Wrapper script(s) – `zerosum` and `zerosum-mpi`
  - Periodicity – default 1 second
  - Detailed/verbose output – true/false, default false
  - Heartbeat (memory consumption) – true/false, default false
  - Register signal handler – true/false, default false
  - Run in debugger – true/false, default false (also specify executable name)
  - Detect deadlocks – true/false, default false
  - Deadlock period – how many periods of "inactivity" is considered "deadlock"
  - HWT/Core for async thread – defaults to last core/HWT in affinity list for process
- Preloads the library, wraps `__libc_start_main` or creates global static constructor/destructor functions

What does it do? …Configuration Detection

- Query `/proc/[self|pid]/status` to get the allowable cores
- Query `/proc/meminfo` to get total memory available
- Query MPI rank, size, hostname (after `MPI_Initialized()` returns `true`)
- If available, use hwloc to query hardware topology (`lstopo`)
- Asynchronous background thread is started, it periodically queries:
  - `/proc/[self|pid]/status` to get process thread count, memory usage
  - `/proc/[self|pid]/task` directory to get all thread IDs
  - For each thread, query the affinity list for that thread (it may change!), utilization, state, core/HWT it's running on, context switches, other metrics
  - `/proc/stat` to query core utilization of all cores
- OMP-Tools (v5.0+) callback used to identify OpenMP threads at creation*
- NVML/ROCm-SMI/SYCL libraries used to query/monitor GPU(s)

Utilization Report

- Rank 0 writes a summary report to the screen
- All ranks write a report to a log file – including full time series data as CSV
- All observed threads (LWP) are reported – user/system/idle, context switches, affinity list
- All assigned cores (HWT) are reported – user/system/idle
- All assigned GPUs stats are reported

Example Output – miniQMC (OpenMP target offload) on Frontier (OLCF)

```
Duration of execution: 210.878 s

Process Summary:
MPI 000 - PID 51334 - Node frontier09085 - CPUs allowed: [1,2,3,4,5,6,7]

LWP (thread) Summary:
LWP 51334: Main,OpenMP - stime: 12.48, utime: 63.94, nv_ctx:   4, ctx: 365488, CPUs: [1]
LWP 51343:     ZeroSum - stime:  0.15, utime:  0.26, nv_ctx:   9, ctx:    679, CPUs: [7]
LWP 51374:       Other - stime:  0.00, utime:  0.00, nv_ctx:   0, ctx:      6, CPUs:
     [1-7,9-15,17-23,25-31,33-39,41-47,49-55,57-63,65-71,73-79,81-87,89-95,97-103,
      105-111,113-119,121-127]
LWP 51384:      OpenMP - stime: 12.60, utime: 64.00, nv_ctx:   3, ctx: 365742, CPUs: [3]
LWP 51385:      OpenMP - stime: 12.63, utime: 64.27, nv_ctx:   2, ctx: 352574, CPUs: [5]
LWP 51386:      OpenMP - stime: 12.74, utime: 63.76, nv_ctx: 473, ctx: 368585, CPUs: [7]
```

Process Summary

LWP (thread) Summary

Note: times are in jiffies – typically 0.01 seconds. For final summaries they are percentages.

Example Output – miniQMC (OpenMP target offload) on Frontier (OLCF)

HWT (core/thread) Summary

```
Hardware Summary:
CPU 001 - idle:   22.70, system:   12.42, user:   64.52
CPU 002 - idle:   99.82, system:    0.00, user:    0.00
CPU 003 - idle:   23.08, system:   12.60, user:   63.97
CPU 004 - idle:   99.83, system:    0.00, user:    0.00
CPU 005 - idle:   22.79, system:   12.62, user:   64.23
CPU 006 - idle:   99.83, system:    0.00, user:    0.00
CPU 007 - idle:   22.94, system:   12.89, user:   63.81
```

Note: times are in jiffies

GPU Summary

```
GPU 0 - (metric: min  avg  max)
    Clock Frequency, GLX (MHz): 800.000000 1614.691943 1700.000000
    Clock Frequency, SOC (MHz): 1090.000000 1090.000000 1090.000000
    Device Busy %: 0.000000 14.616114 52.000000
    Energy Average (J): 0.000000 8.328571 10.000000
    GFX Activity: 0.000000 17223.704762 38443.000000
    GFX Activity %: 0.000000 13.706161 41.000000
    Memory Activity: 0.000000 623.623810 1536.000000
    Memory Busy %: 0.000000 0.355450 3.000000
    Memory Controller Activity: 0.000000 0.303318 2.000000
    Power Average (W): 90.000000 126.483412 138.000000
    Temperature (C): 35.000000 37.909953 39.000000
    UVD|VCN Activity: 0.000000 0.000000 0.000000
    Used GTT Bytes: 11624448.000000 11624448.000000 11624448.000000
    Used VRAM Bytes: 15044608.000000 4743346651.601895 4839596032.000000
    Used Visible VRAM Bytes: 15044608.000000 4743346884.549763 4839596032.000000
    Voltage (mV): 806.000000 891.848341 906.000000
```

Logs have full time series of all samples

Contention Detection Support

- Voluntary/non-voluntary context switches (analysis todo)
- Minor/major page faults, pages swapped (analysis todo)
- System time analysis (analysis todo)
- Comparing affinity lists – across threads *and* across processes (todo)
- Memory consumption (analysis todo)
- GPU memory consumption (analysis todo)
- However, **can detect deadlocks** – both **active** (i.e. spinning at MPI collective) and **passive** (i.e. waiting for mutex)

Evaluation / Example usage

MPI+OpenMP version of miniQMC on Frontier, 8 processes, 7 threads (64 cores, 1 thread per core, 8 cores reserved)

LWP	Type	stime	utime	nvctx	ctx	CPUs
18351	Main†	1.54	15.17	332905	1838	1
18356	*ZeroSum*	0.42	1.10	194	1007	1
18385	Other	0.00	0.00	0	41	1-127‡
18405	OpenMP	0.31	13.09	232689	5	1
18407	OpenMP	0.44	12.93	353365	11	1
18408	OpenMP	0.21	13.22	92528	3	1
18409	OpenMP	0.47	12.93	394014	10	1
18410	OpenMP	0.37	13.03	302371	7	1
18411	OpenMP	0.41	12.97	348829	10	1

Table 1: Frontier results, default configuration. †indicates that the main thread is also an OpenMP thread. ‡indicates that the first core of each L3 region was set aside for system processes, not all threads in the sequence 1-127 are allowed but summarized for brevity in the table (see LWP 51274 in Listing 2).

LWP	Type	stime	utime	nvctx	ctx	CPUs
18552	Main†	3.13	88.40	5	704	1-7
18561	*ZeroSum*	0.79	2.64	2	2790	7
18588	Other	0.00	0.00	0	41	1-127‡
18589	OpenMP	1.10	90.00	9	716	1-7
18590	OpenMP	1.10	93.00	8	724	1-7
18591	OpenMP	1.07	90.52	9	692	1-7
18592	OpenMP	1.10	89.83	14	766	1-7
18593	OpenMP	1.10	90.48	7	728	1-7
18594	OpenMP	1.10	91.93	300	849	1-7

Table 2: Frontier results, configuration requesting 7 cores per process. †indicates that the main thread is also an OpenMP thread. ‡indicates that the first core of each L3 region was set aside for system processes, not all threads in the sequence 1-127 are allowed but summarized for brevity in the table (see LWP 51274 in Listing 2).

LWP	Type	stime	utime	nvctx	ctx	CPUs
18948	Main†	3.07	88.57	2	386	1
18954	*ZeroSum*	0.71	2.57	2	291	7
18981	Other	0.00	0.00	0	41	1-127‡
18992	OpenMP	1.18	96.36	0	422	2
18993	OpenMP	1.14	96.50	1	391	3
18994	OpenMP	1.18	96.46	0	381	4
18995	OpenMP	1.11	93.89	0	324	5
18996	OpenMP	1.14	93.29	0	370	6
18997	OpenMP	1.14	95.54	208	358	7

Table 3: Frontier results, configuration requesting 7 cores per process and binding OpenMP threads to cores. †indicates that the main thread is also an OpenMP thread. ‡indicates that the first core of each L3 region was set aside for system processes, not all threads in the sequence 1-127 are allowed but summarized for brevity in the table (see LWP 51274 in Listing 2).

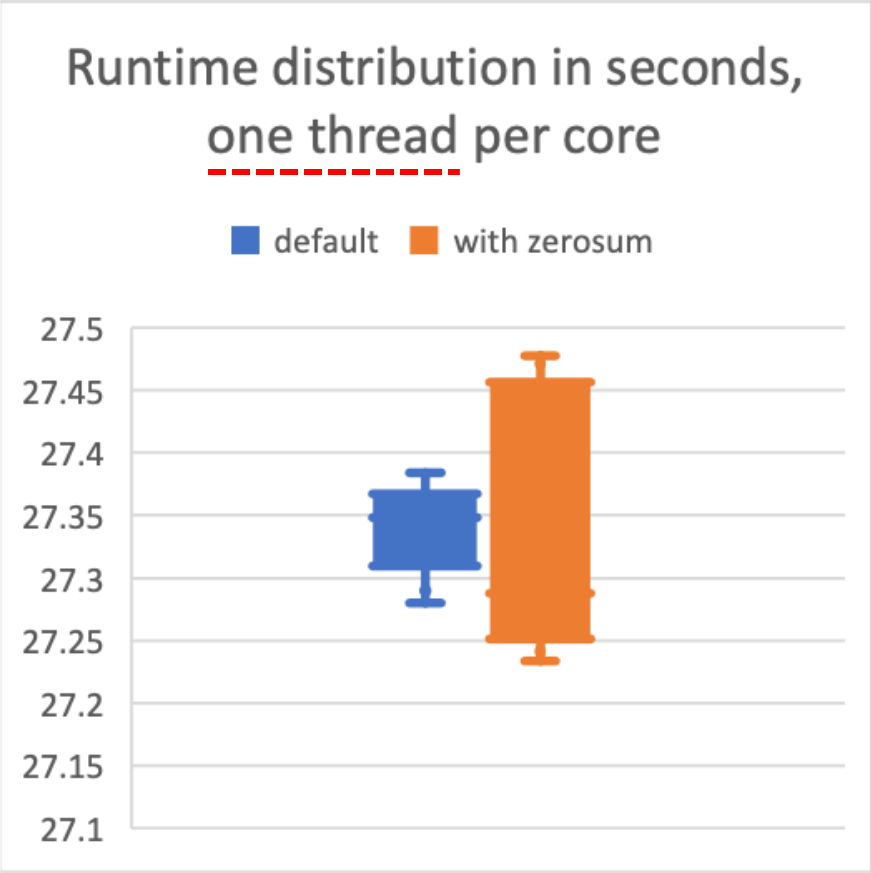
```
export OMP_NUM_THREADS=7
srun -n8 zerosum-mpi miniqmc
```

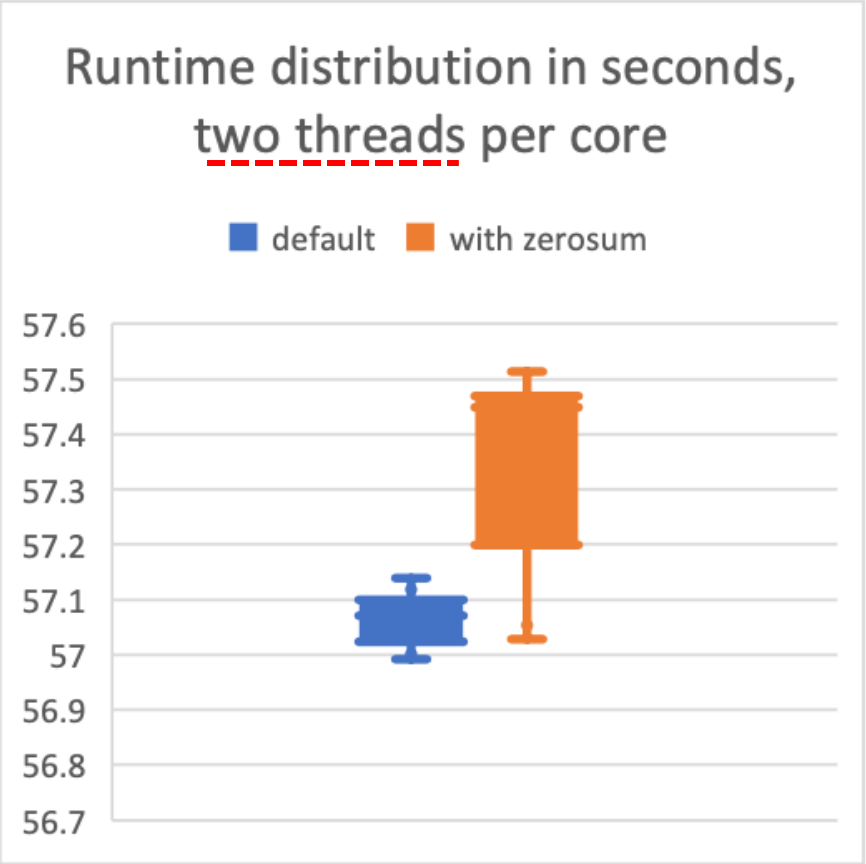
```
export OMP_NUM_THREADS=7
export OMP_PROC_BIND=spread
export OMP_PLACES=cores
srun -n8 -c7 zerosum-mpi miniqmc
```

```
export OMP_NUM_THREADS=7
srun -n8 -c7 zerosum-mpi miniqmc
```

Note: stime/utime reported as % of total time

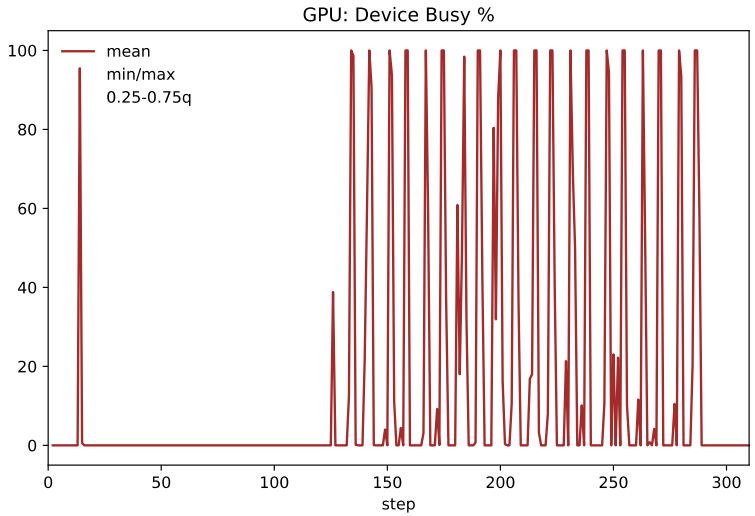
Overhead – less than 0.5% in resource constrained example

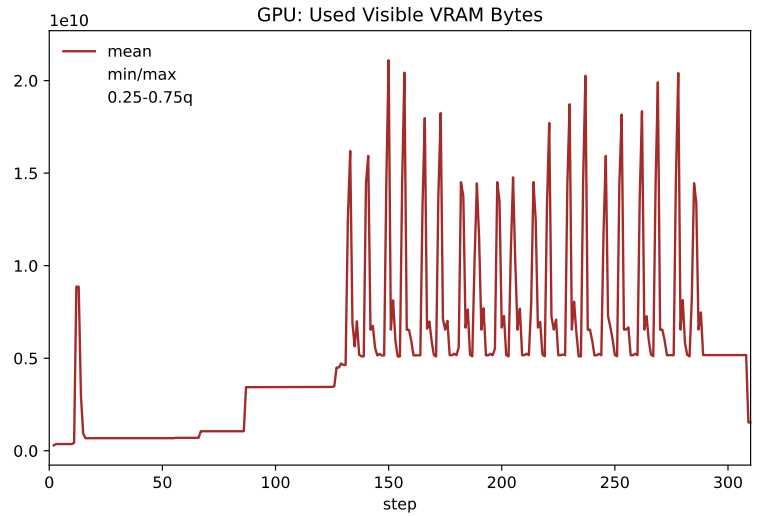


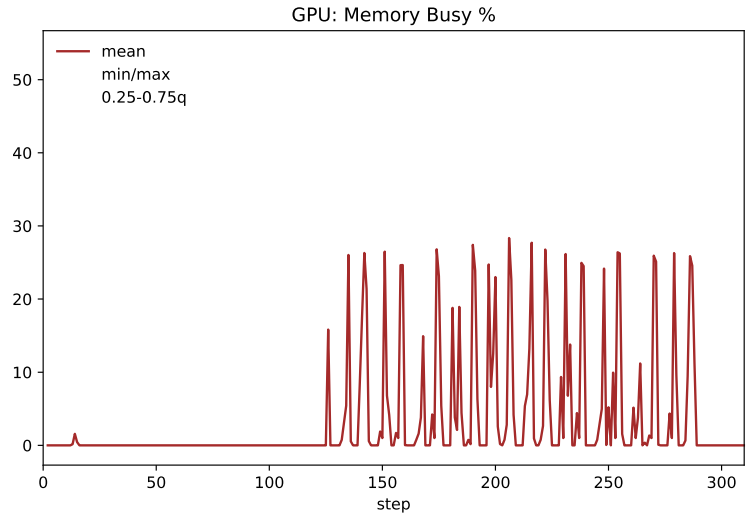


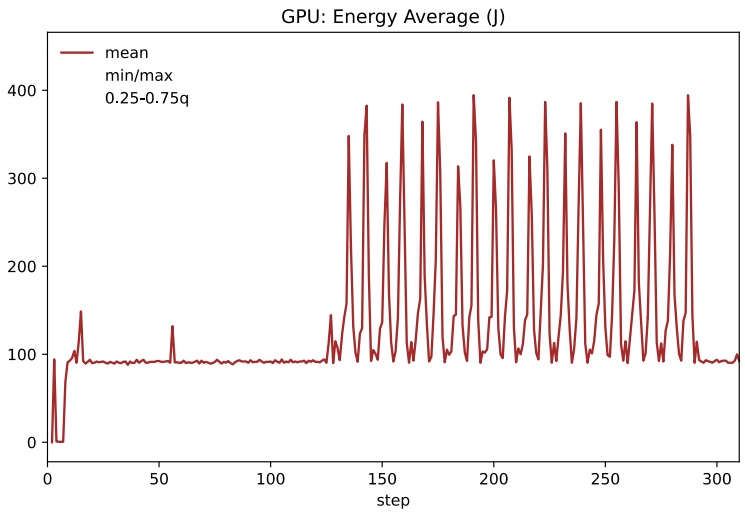
miniQMC time distributions executed 10 times using one OpenMP thread per core (left). In this comparison, the distribution of times with ZeroSum is noisier, but there is no significant observation of measurable overhead. The right figure shows the time distributions using two OpenMP threads per core. In this comparison, the distribution of times with ZeroSum is both noisier and longer tailed, and does show an observation of overhead, averaging about 0.2752 seconds, or 0.5%.

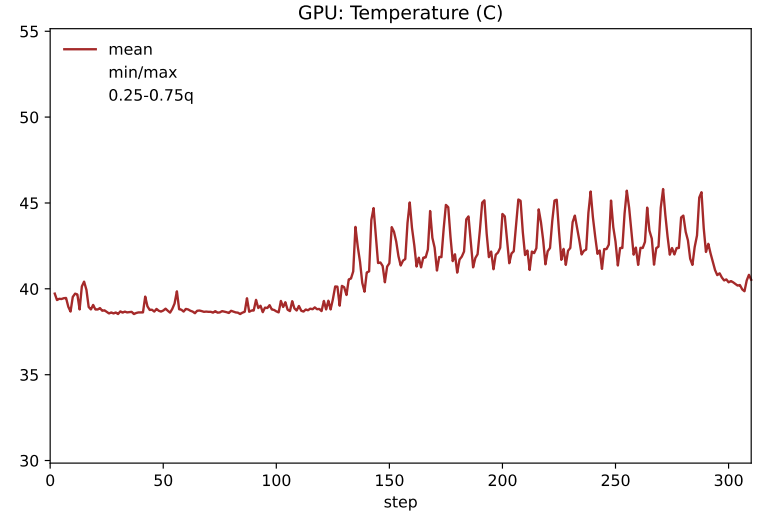
Summary views of collected data – 64 GCDs, XGC running on Frontier

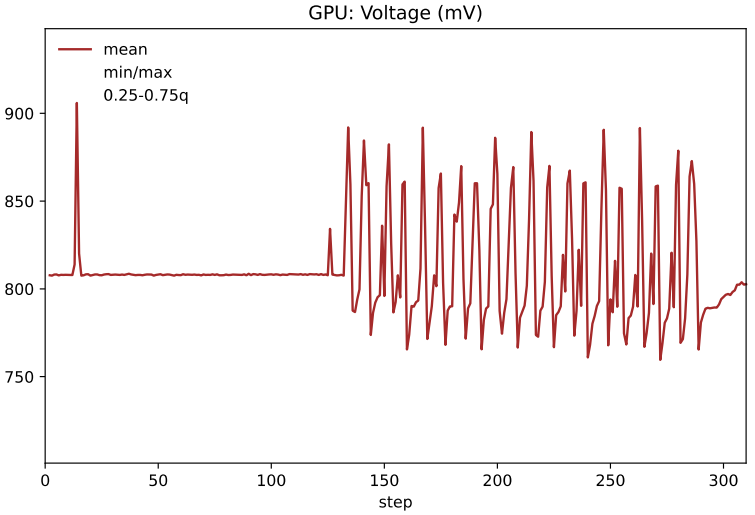










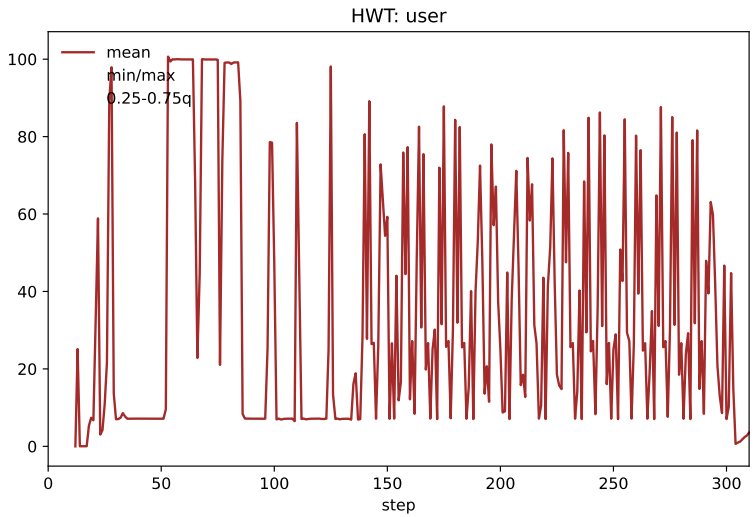


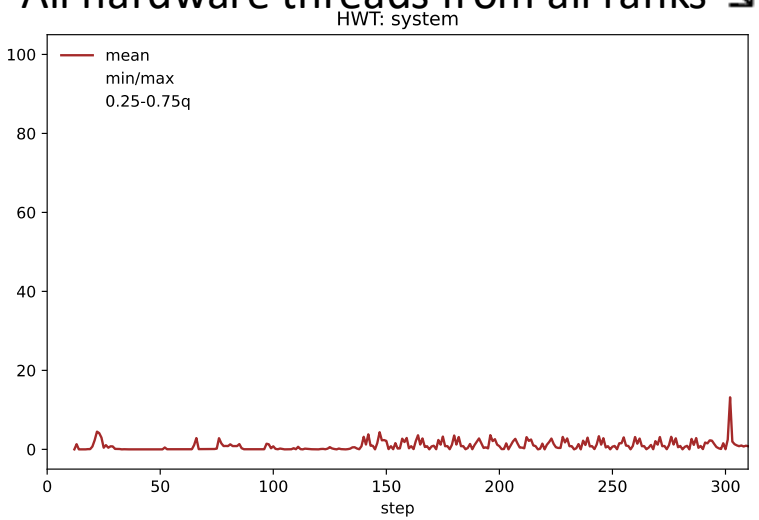
UNIVERSITY OF OREGON

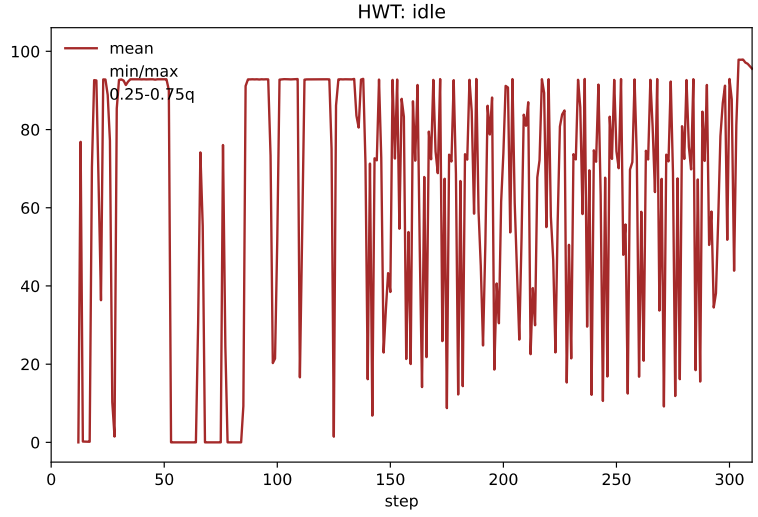
Data post-processed using Python scripts

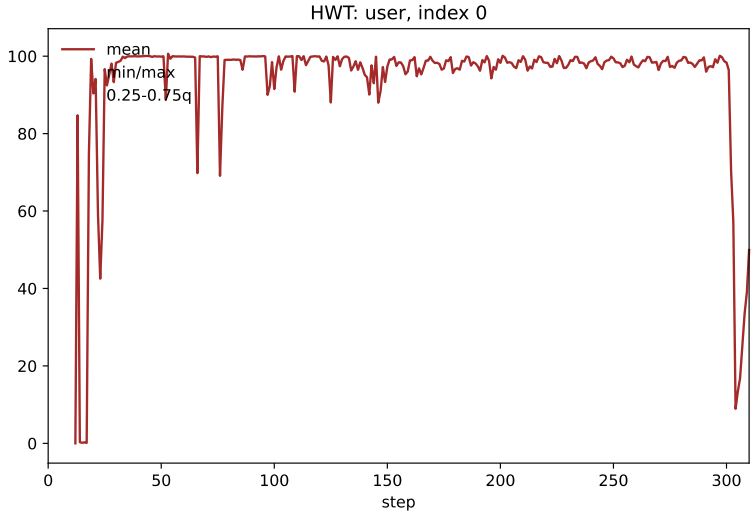
Summary views of collected data – 896 HWTs, XGC running on Frontier

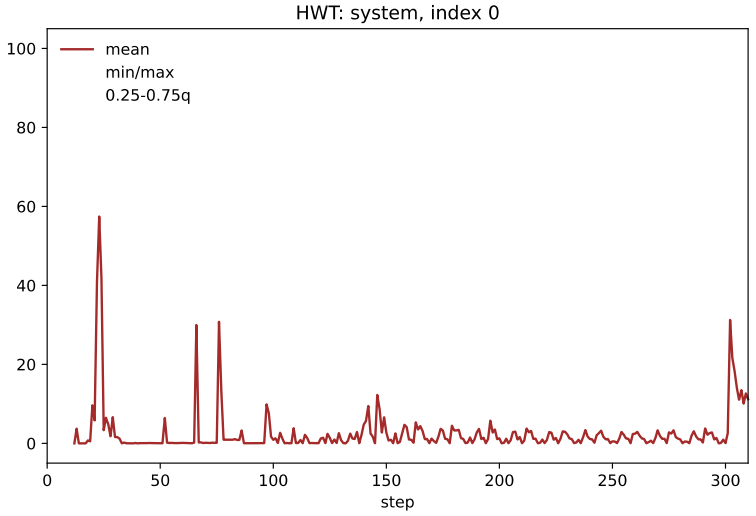
↙ All hardware threads from all ranks ↘

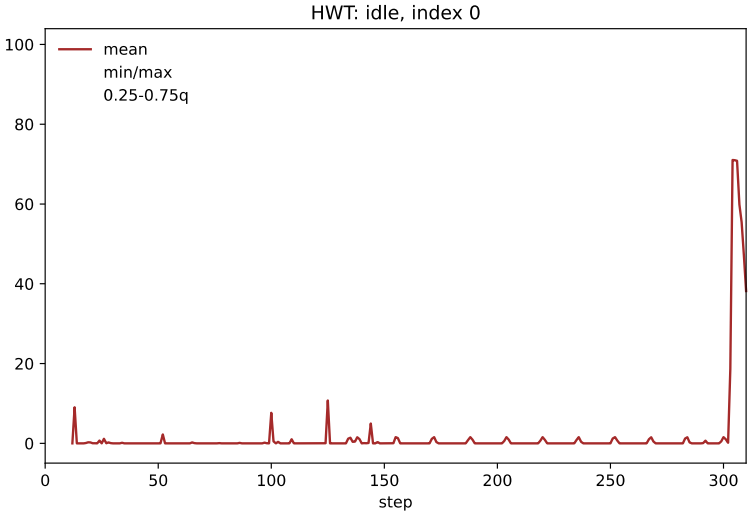










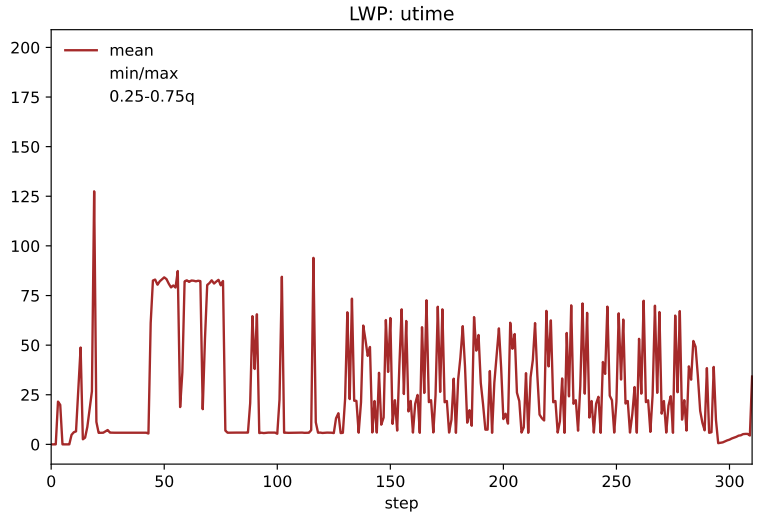


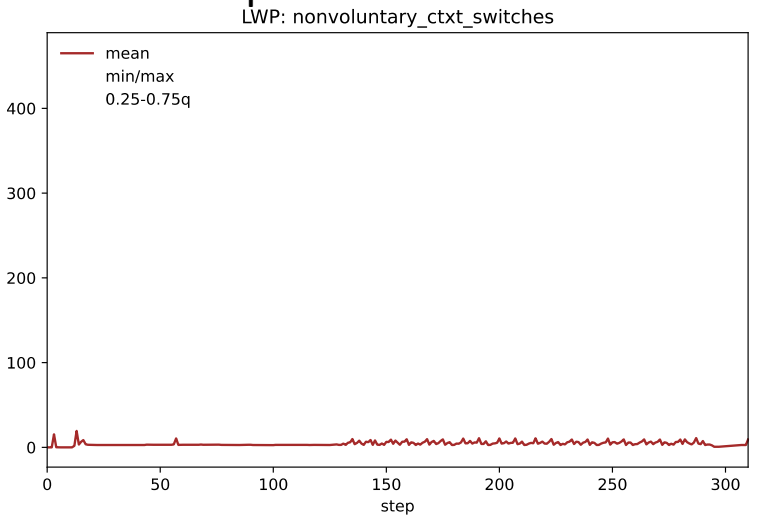
UNIVERSITY OF OREGON

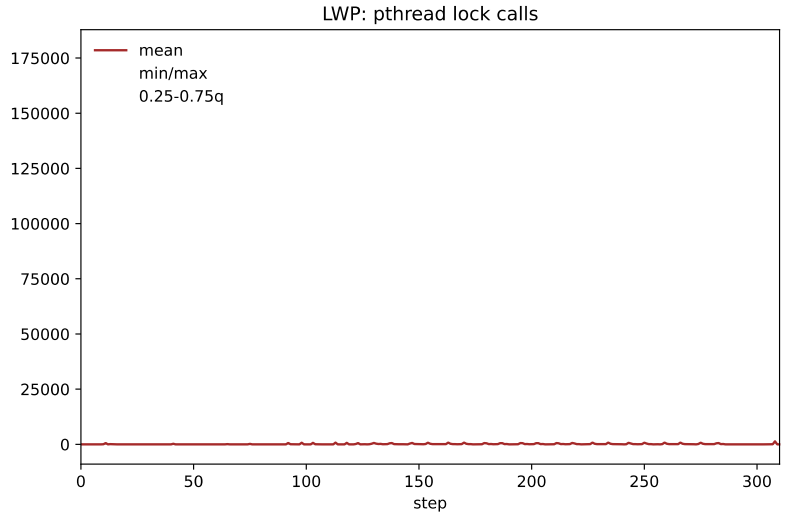
↖ Only main hardware thread from all ranks ↗

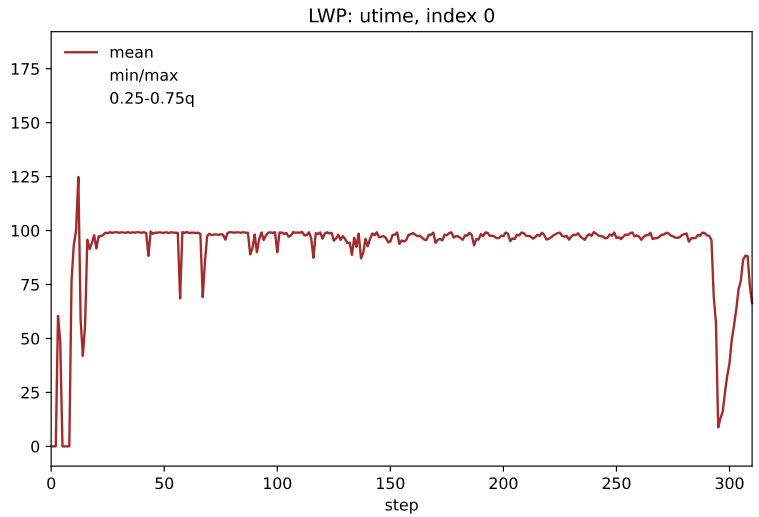
Summary views of collected data – 1088 LWPs, XGC running on Frontier

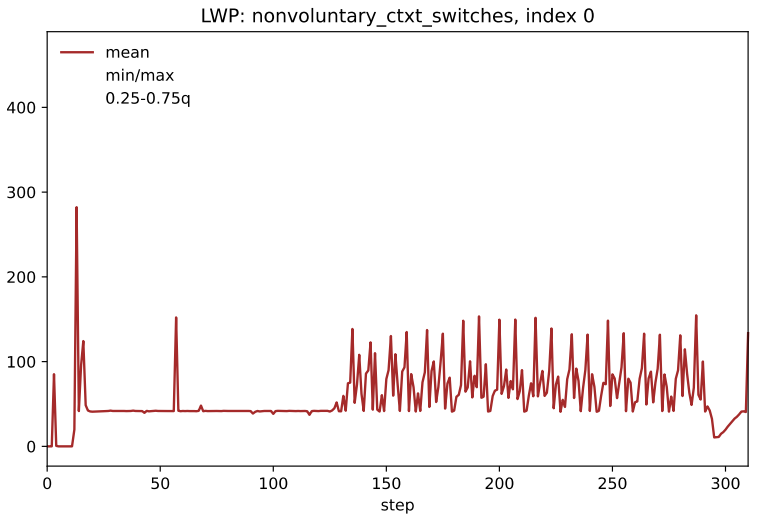
↙ All process threads ↘

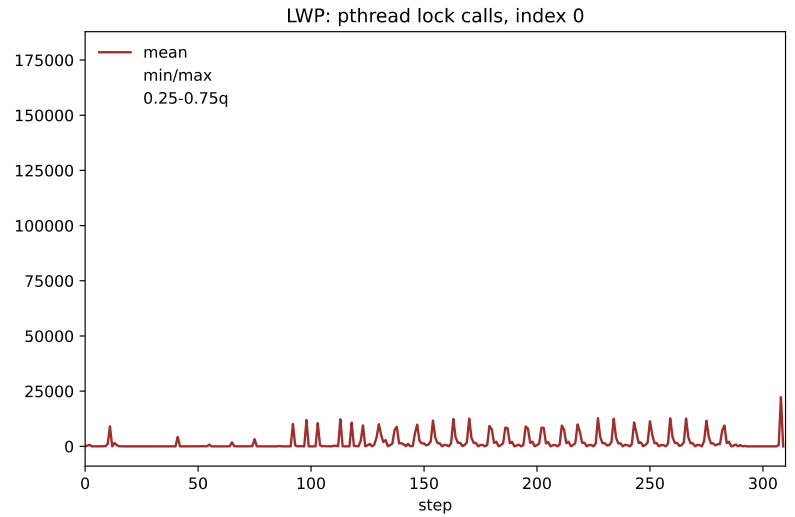












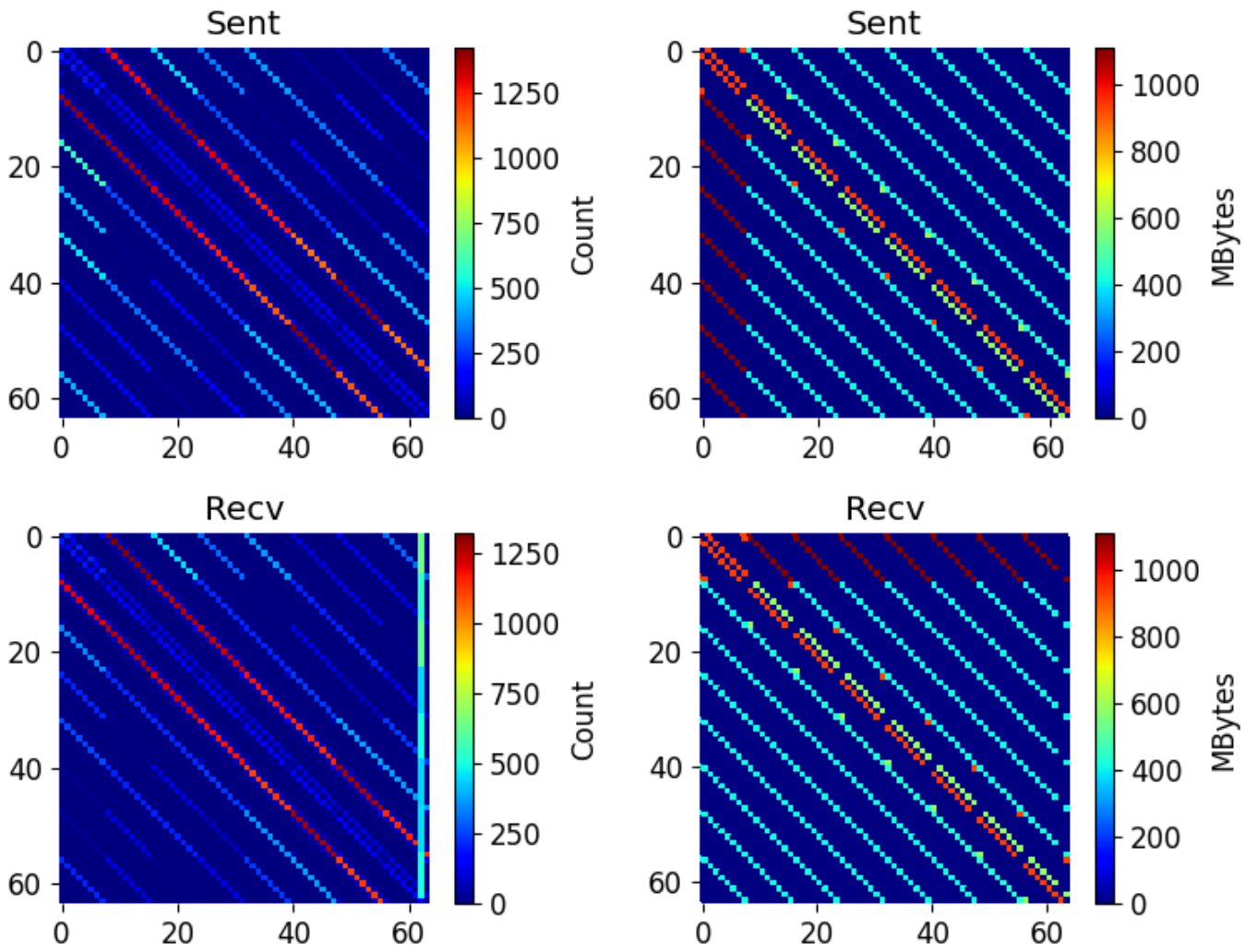
UNIVERSITY OF OREGON

↖ Only main process thread ↗

MPI P2P data

- MPI implementation includes wrappers for `MPI_Recv/Irecv`, `MPI_Sendrecv`, `MPI_Send/Bsend/Isend/Rsend/Ssend` to capture P2P frequency, volume
- Obviously can be done by other/better performance measurement tools, but for a quick view, can be useful

- Apply analysis from “Optimizing Process-to-Core Mappings for Application Level Multi-dimensional MPI Communications”, Karlsson et al. 2012. https://doi.org/10.1109/CLUSTER.2012.47



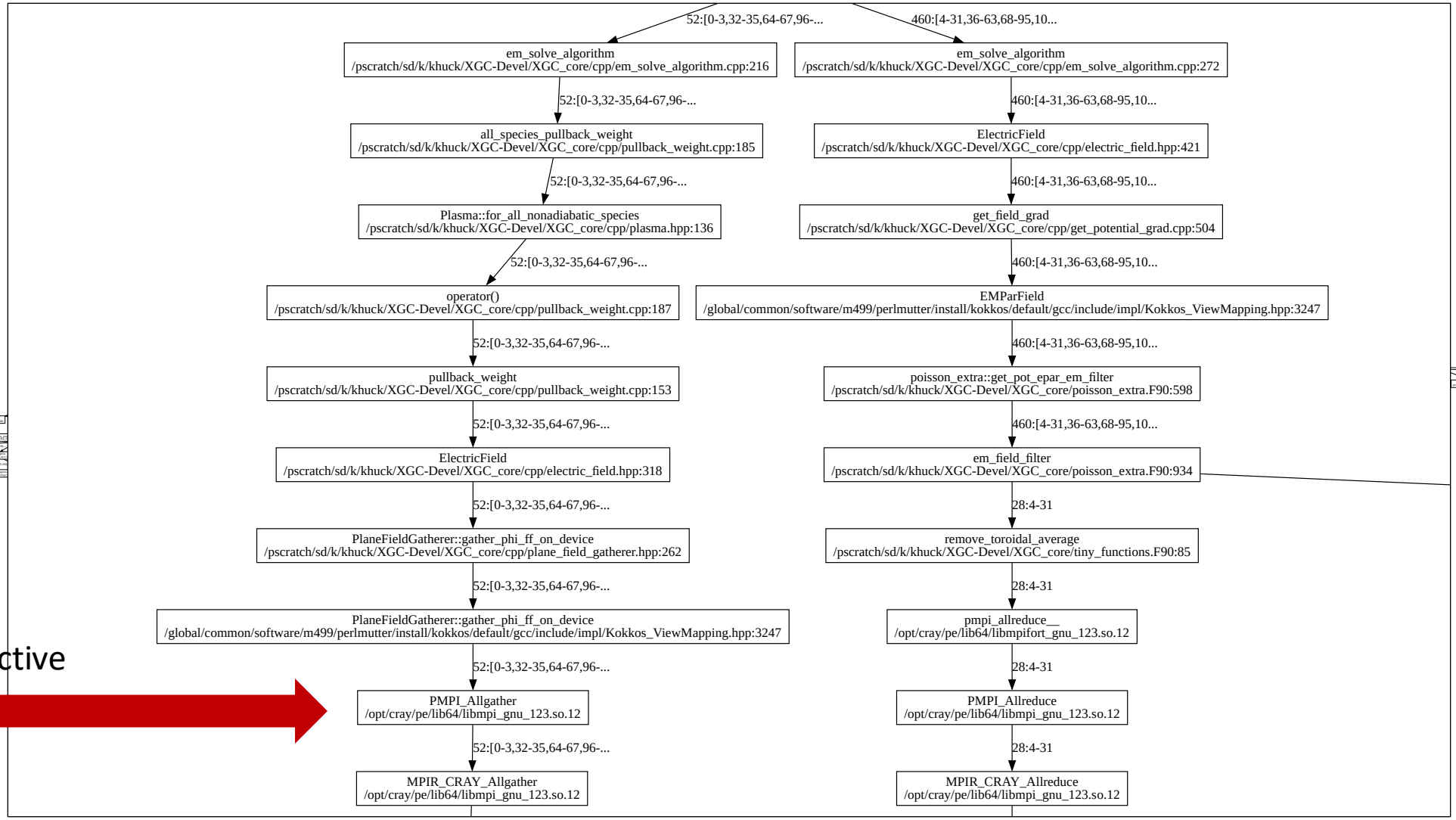
Successful Use Cases

- Octo-Tiger (HPX) thread placement on Fugaku
  - Detected unusual core indexing, needs to be mapped using hwloc information
- Argobots example
  - Detected that the user was running client & server on same node, with overlapping affinity lists
- Grid (https://github.com/paboyle/Grid) on Frontier (lattice QCD library)
  - Helped find/understand ideal resource mapping with custom bind arguments
- XGC on Frontier
  - Empirically demonstrated that 2 threads per core provided better performance than 1 thread per core
  - Helped detect deadlock in GPU-aware libfabric/MPI implementation
  - Helped detect algorithmic deadlock in MPI (application bug)

Deadlock detection – at scale

- Runs application in gdb/cuda-gdb/rocgdb/gdb-oneapi in “batch” mode – sequence of commands are scripted
- ZeroSum monitors all threads during execution:
  - If all sleeping for X seconds (X is user-configurable):
    - pthread_kill(SIGQUIT) the main thread
  - If all but 1 sleeping for X seconds, *and* that thread doesn’t have at least one minor page fault during that time, it’s *probably* deadlocked at an MPI collective:
    - pthread_kill(SIGQUIT) the main thread
- Gdb will intercept the signal and is scripted to get a backtrace from all threads
- Post-process the threads and make a tree (just like STAT/Cray-STAT)
  - *However*, ZeroSum is automated - queue times on Perlmutter were around 8 hours, user didn’t want to monitor the 128 node slurm request

Example: XGC deadlocked on Perlmutter with 512 ranks



Only 52 of 64 in communicator participated in collective

Conclusions, Future Work

- ZeroSum addresses the *configuration optimization* problem
  - https://github.com/UO-OACISS/zerosum
- Still need automated misconfiguration/contention detection
- Output data could be better – use ADIOS2 BP5? HDF5? SQLite?
  - Depends on analysis needs/wants
  - Current log file "format" means analysis process is manual/ad hoc (via Python)
- Streaming data to Mochi (SOMA) or other service (LDMS)?
  - Enables robust monitoring approach – but likely redundant in many cases
- Integration with performance tools (TAU, APEX, etc)
  - Analysis of application performance data in context of system monitoring data
- Input for automated feedback/control (APEX, Argobots, SOMA, application)

PerfStubs:

Profiling API for adding external tool instrumentation support to any project

Kevin A. Huck

Oregon Advanced Computing Institute for Science and Society (OACISS)

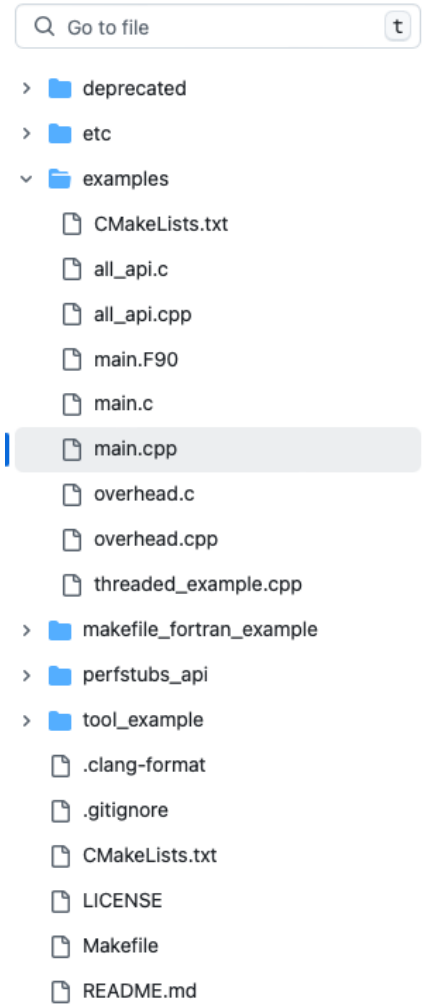
UNIVERSITY OF OREGON

PerfStubs

- PerfStubs is a “frictionless” instrumentation library
  - https://github.com/UO-OACISS/perfstubs
  - One source file, three headers
  - Provides a plugin interface for performance tools
  - Can be compiled away if desired
- Integrated into several libraries (so far) as a git submodule or direct
  - CAMTIMERS
  - PETSc
  - Ginkgo
  - ADIOS2
  - Others?
- Provides runtime integration with TAU & APEX

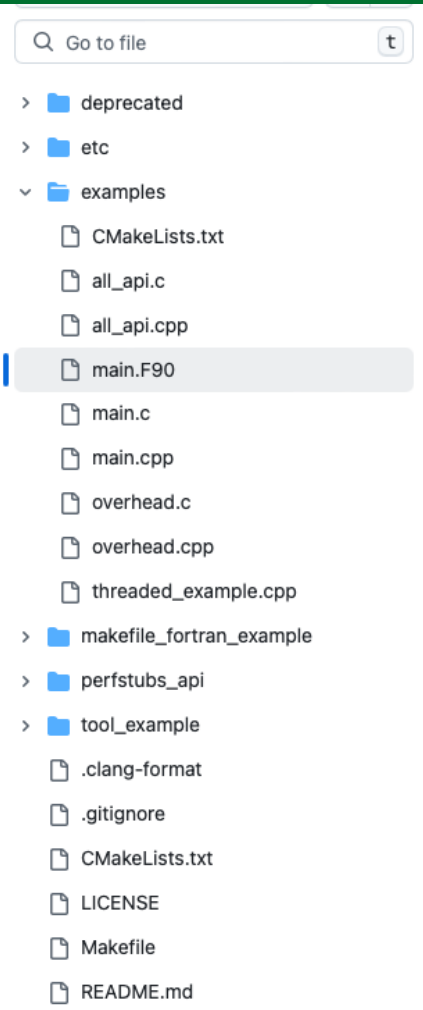
Boehme, Huck, Madsen, Weidendorfer,
“The Case for a Common Instrumentation Interface for HPC Codes”
https://doi.org/10.1109/ProTools49597.2019.00010, 2019

C++ example




```
43
44  int main(int argc, char *argv[])
45  {
46      PERFSTUBS_INITIALIZE();
47      PERFSTUBS_SCOPED_TIMER_FUNC();
48
49      {
50          PERFSTUBS_SCOPED_TIMER("Argument Validation");
51          if (argc < 2) {
52              std::cerr << argv[0] << " Version "
53                      << PerfStubs_VERSION_MAJOR << "."
54                      << PerfStubs_VERSION_MINOR << std::endl;
55              std::cerr << "Usage: " << argv[0] << " number" << std::endl;
56              return 1;
57          }
58      }
59      std::thread example_thread(threaded_function, nullptr);
60
61      double inputValue = atof(argv[1]);
62      PERFSTUBS_SAMPLE_COUNTER("input", inputValue);
63
64      double outputValue;
65      for (int i = 0; i < 5; i++)
66      {
67          PERFSTUBS_DYNAMIC_PHASE_START("Loop", i);
68          outputValue = compute(inputValue);
69          outputValue = compute2(inputValue);
70          outputValue = compute3(inputValue);
71          PERFSTUBS_DYNAMIC_PHASE_STOP("Loop", i);
72      }
73
74      example_thread.join();
75      std::cout << "The square root of " << inputValue << " is " << outputValue
76                << std::endl;
77
78      PERFSTUBS_DUMP_DATA();
79      PERFSTUBS_FINALIZE();
80      return 0;
81  }
```

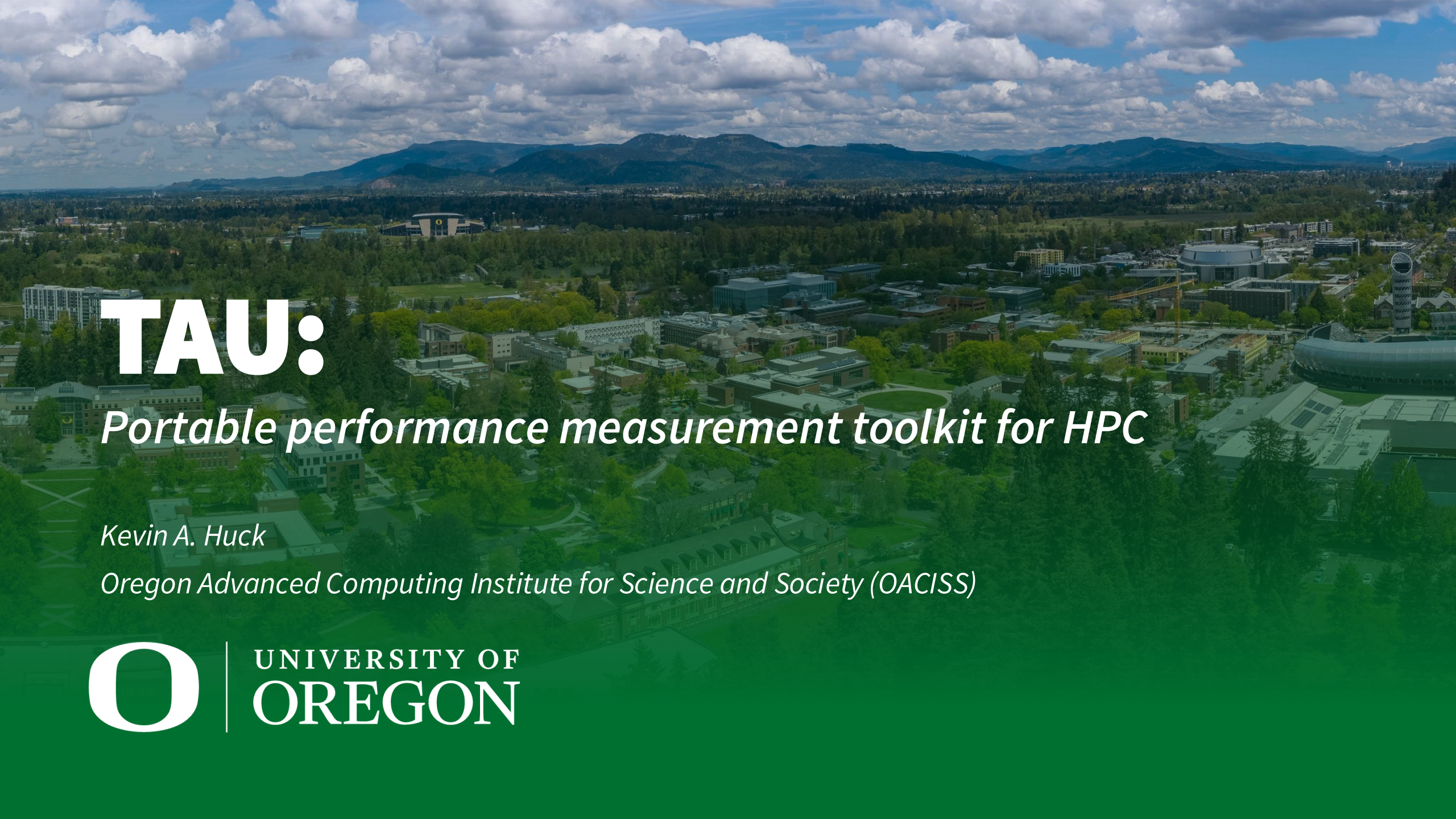
Fortran example:




```
24
25          PERFSTUBS_START_STRING('HELLOWORLD2')
26          print *, "Iteration = ", iVal
27          PERFSTUBS_STOP_CURRENT()
28        end
29
30        subroutine HELLOWORLD3(iVal)
31          integer iVal
32
33          PERFSTUBS_START_STRING('HELLOWORLD3')
34          print *, "Iteration = ", iVal
35          PERFSTUBS_STOP_STRING('HELLOWORLD3')
36        end
37
38        program main
39          integer i
40          integer profiler(2)
41          save    profiler
42          integer counter(2)
43          save    counter
44
45          PERFSTUBS_INITIALIZE()
46          PERFSTUBS_TIMER_CREATE(profiler, 'main')
47          PERFSTUBS_TIMER_START(profiler)
48          PERFSTUBS_CREATE_COUNTER(counter, 'test counter')
49          PERFSTUBS_SAMPLE_COUNTER(counter, 25.0)
50          PERFSTUBS_METADATA('foo', 'bar')
51
52          print *, "test program"
53
54          do 10, i = 1, 10
55              call HELLOWORLD(i)
56              call HELLOWORLD2(i)
57              call HELLOWORLD3(i)
58    10      continue
59          PERFSTUBS_TIMER_STOP(profiler)
60          PERFSTUBS_DUMP_DATA()
61          PERFSTUBS_FINALIZE()
62        end
```

Comparison to other instrumentation libraries

- Similar to NVTX (CUDA), ROC-TX (HIP), etc. – but portable
- Can be compiled away completely (macro-based)
- If no tool attached, very little overhead – but that depends on how much instrumentation you add, and where (don't put it inside tight loops, for example)
- Similar library TaskStubs being developed for asynchronous tasking runtimes like PaRSEC, StarPU, IRIS
- Module installed on Frontier:
  - `module load ums ums002/vanilla ; module load perfstubs`



TAU:

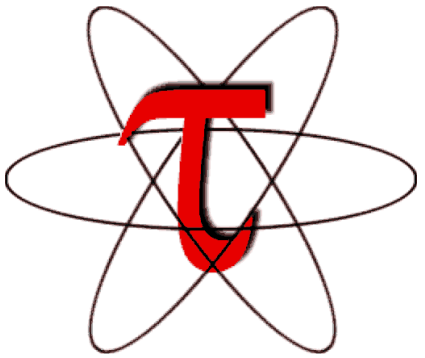
Portable performance measurement toolkit for HPC

Kevin A. Huck

Oregon Advanced Computing Institute for Science and Society (OACISS)

UNIVERSITY OF OREGON

TAU Performance System



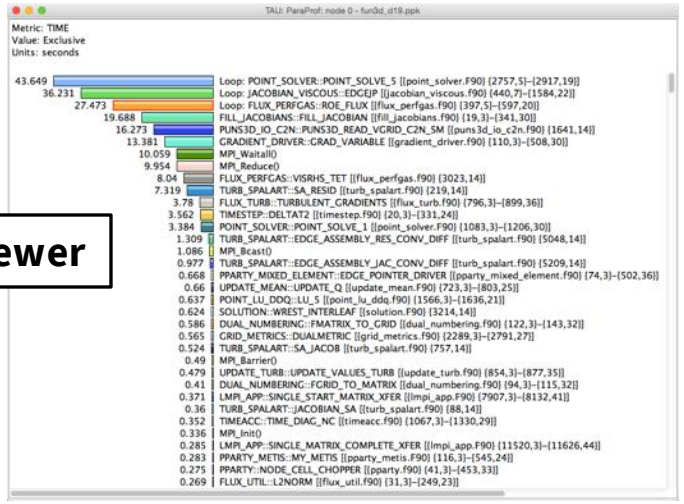
- **T**uning and **A**nalysis **U**tilities (29+ year project)
- Integrated performance toolkit:
  - Multi-level performance instrumentation
  - Highly configurable
  - Widely ported performance profiling / tracing system
  - Portable (java, python) visualization / exploration / analysis tools
- Supports all major HPC programming models
- MPI/SHMEM, OpenMP, OpenACC, CUDA, HIP, SYCL/OneAPI, Kokkos…
- Support for ML/AI frameworks: TensorFlow, pyTorch, Horovod, Python
- Integrated with PAPI, LIKWID for hardware counter support
- https://tau.uoregon.edu or https://github.com/UO-OACISS/tau2 (public mirror)

Performance Measurement

- **Timers**
  - Requires instrumentation of some kind
    - Manual, automated
    - Source, compiler provided, binary
    - **Library callbacks**, API wrappers, weak symbol replacement
  - Simple to implement
- **Sampling**
  - Requires specialized system libraries / support (nearly universal)
    - Periodic signals, signal handler
    - Call stack unwinding
  - No modification to executable/library needed
  - Potential to interfere with system support (signal handlers)
  - Can mix with timers to generate a hybrid profile

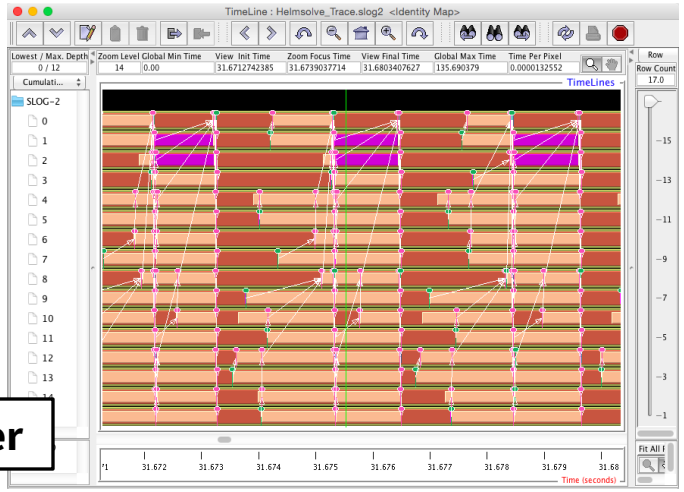
Profiling and Tracing

- **Profiling**: how much time was spent in each measured function on each thread in each process?
  - Collapses the time axis
  - No ordering or causal event information
  - Small summary per thread/process, regardless of execution time – only grows with number of timers & threads/processes



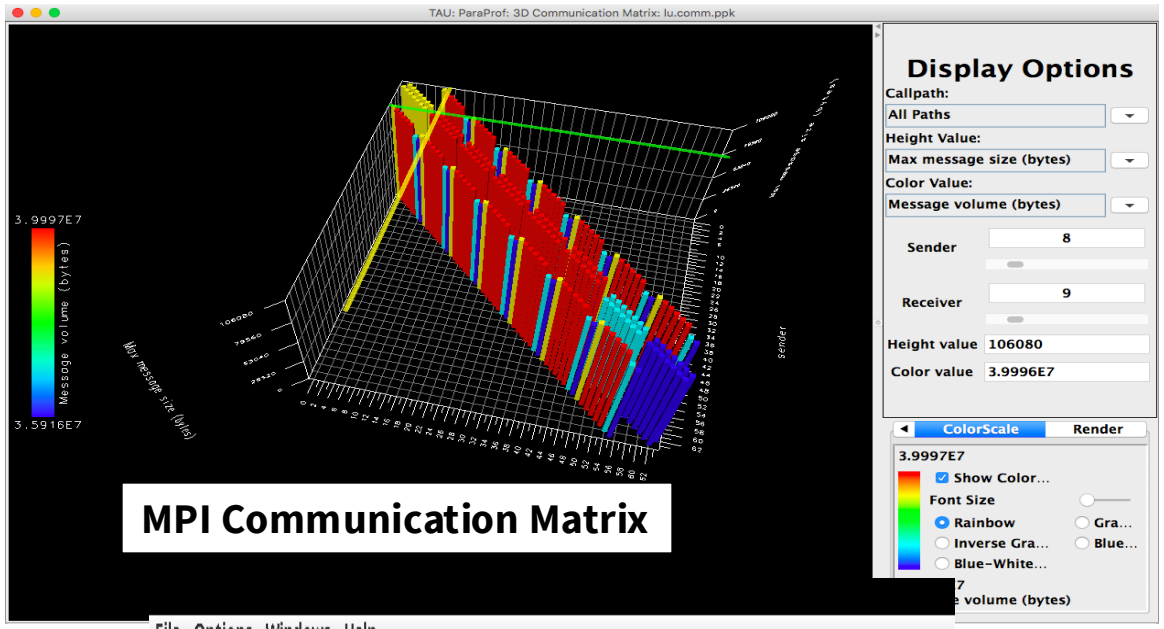
ParaProf profile viewer

- **Tracing**: record all function entry & exit events on a timeline
  - Detailed view of what happened
  - The longer the program runs, the bigger the trace

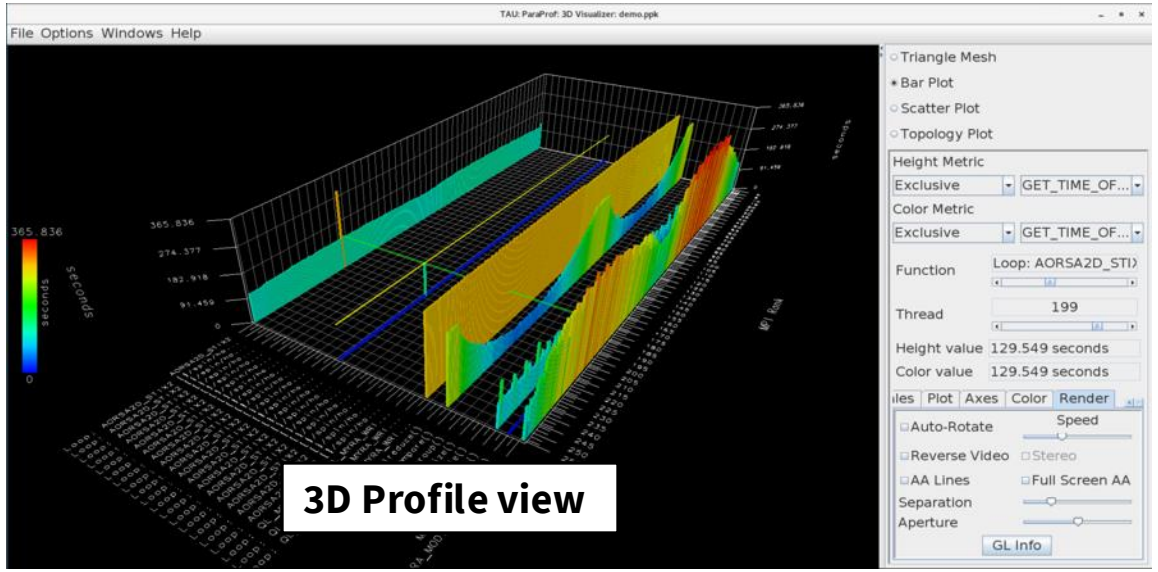


JumpShot trace viewer

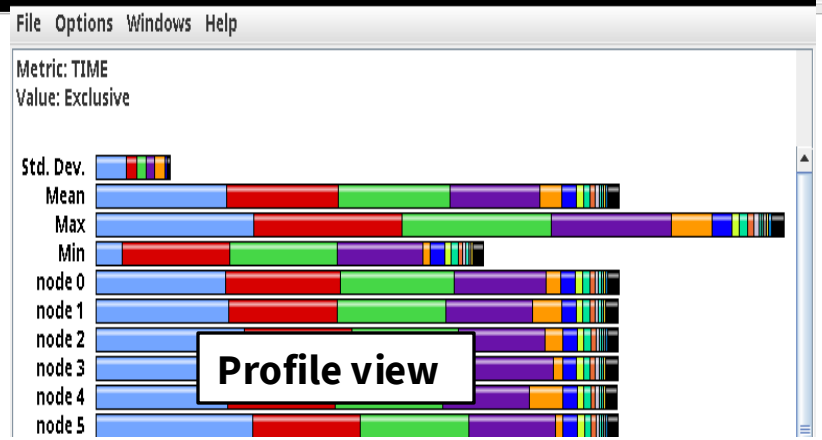
TAU Analysis Tools: ParaProf, Vampir



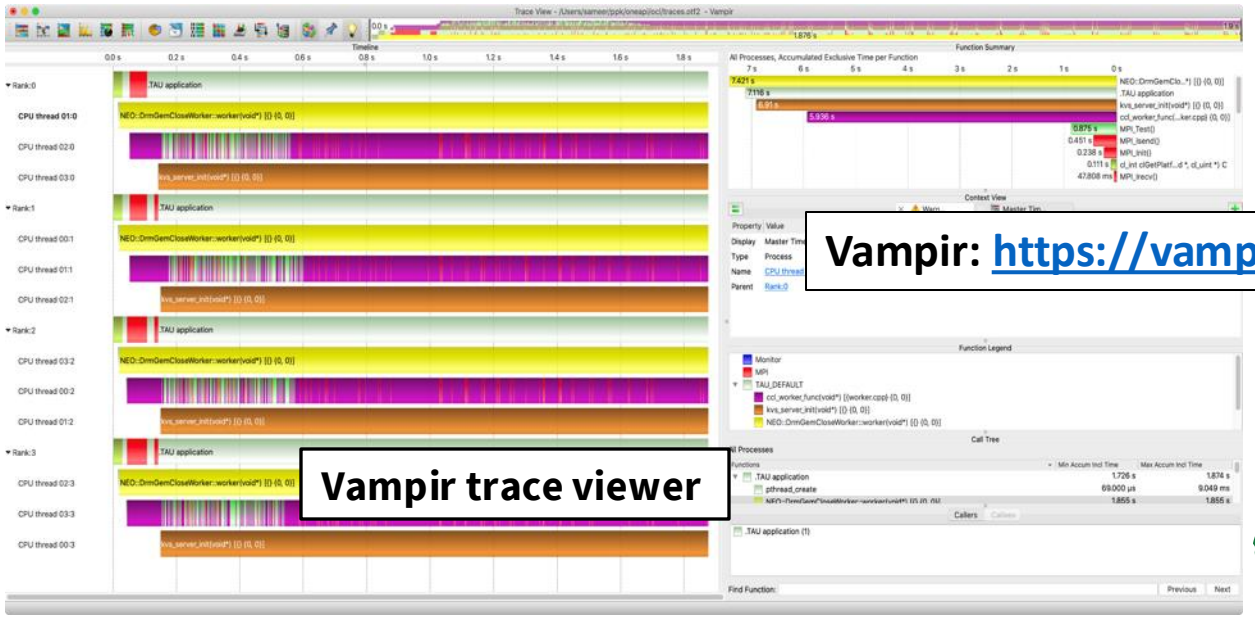
MPI Communication Matrix



3D Profile view



Profile view



Vampir trace viewer

Vampir: https://vampir.eu

Programming Model Support in TAU

- MPI, SHMEM – interposition library to intercept all MPI calls
- OpenMP – OMPT support (provided by all compilers except GCC), including target offload support
- OpenACC – callback API similar to OpenMP
- CUDA – measurement support through CUPTI, NVML libraries
- HIP – measurement support through Rocprofiler, Roctracer, Rocm-SMI libraries
- SYCL/Level0 – support provided through SYCL, Level0 APIs
- Python – integration with cProfile, updated support in development for Python 3.12+

Using TAU on Frontier

- `module load tau`
- `srun <srun options> tau_exec -T <config> <options> your_executable`
- Useful options:
  - `-io` : track I/O operations
  - `-rocm` : enables HIP profiling support
  - `-cuda` : enables CUDA profiling support
  - `-ompt` : enables OpenMP Tools support (most compilers, not GCC)
  - `-ebs` : enables sampling support
  - `-ebs_period=<count>` : Sampling period in microseconds (default 10000)
  - `-ebs_resolution=<file|function|line>` : choose sampling granularity
  - `-monitoring` : enables periodic monitoring of system resources (a la ZeroSum)
- `tau-config --list-matching` : shows installed TAU configurations

Kokkos support in TAU

- TAU implements the Kokkos Profiling API (`Kokkos_Profiling_C_Interface.h`)
- TAU sets an environment variable KOKKOS_TOOLS_LIBS to tell Kokkos that it should enable profiling and enable function callbacks to the TAU implementations
- TAU implements
  - `kokkosp_[init|finalize]_library`
  - `kokkosp_[begin|end]_parallel_[for|scan|reduce]`
  - `kokkosp_[push|pop]_profile_region`
- Names for regions are passed to the tools to provide intelligent labels
- In addition, TAU also implements support for native Pthreads, OpenMP, OpenACC, CUDA, HIP, SYCL back-end measurement – no code changes necessary
- Raja support: if you have a Raja application, and Raja is configured with `-DRAJA_ENABLE_RUNTIME_PLUGINS`, Raja implements the same callback API

TAU Example – Kokkos Lulesh (from kokkos-miniapps)

Metric: TAUGPU_TIME
Value: Exclusive
Units: seconds

5.537	cudaDeviceSynchronize
2.692	cudaMallocAsync
1.08	cudaMemcpy
1.005	cudaStreamSynchronize
0.806	Kokkos::parallel_for Kokkos::View::initialization [_mirror] via memset [type = OpenMP, device = 0]
0.111	Kokkos::parallel_for Kokkos::View::initialization [nodeElemCornerList_mirror] via memset [type = OpenMP, device = 0]
0.106	cudaLaunchKernel
0.08	cudaMemcpyToSymbolAsync
0.078	cudaSetDevice [THROTTLED]
0.046	cudaEventRecord
0.044	Kokkos::parallel_for CalcEnergyForElems [type = Cuda, device = 0]
0.042	Kokkos::parallel_for EvalEOSForElems A [type = Cuda, device = 0]
0.025	cudaEventSynchronize
0.018	Kokkos::parallel_reduce 18__nv_hdl_wrapper_tlLb0ELb0E11__nv_dl_tagIPFvR6DomainiidRdEXadL_ZN72_INTERNAL_50__tmp_tmpxft_002
0.017	Kokkos::parallel_reduce 18__nv_hdl_wrapper_tlLb0ELb0E11__nv_dl_tagIPFvR6DomainiidRdEXadL_ZN72_INTERNAL_50__tmp_tmpxft_002
0.015	Kokkos::parallel_for Kokkos::View::initialization [m_nodeElemStart_mirror] via memset [type = OpenMP, device = 0]
0.015	Kokkos::parallel_for Kokkos::View::initialization [nodeElemCount] via memset [type = OpenMP, device = 0]
0.014	Kokkos::parallel_for Kokkos::View::initialization [regElemlist::entries_mirror] via memset [type = OpenMP, device = 0]
0.013	Kokkos::parallel_for CalcMonotonicQRegionForElems [type = Cuda, device = 0]
0.013	Kokkos::parallel_for EvalEOSForElems F [type = Cuda, device = 0]
0.013	Kokkos::parallel_for CalcSoundSpeedForElems [type = Cuda, device = 0]
0.012	cudaMemsetAsync
0.01	cudaMemcpyAsync
0.005	cudaFuncGetAttributes
0.005	Kokkos::parallel_for Kokkos::View::initialization [] via memset [type = Cuda, device = 0]
0.003	cudaFreeAsync
0.003	Kokkos::parallel_for Kokkos::ViewCopy-1D [type = Cuda, device = 0]
0.002	Kokkos::parallel_reduce 18__nv_hdl_wrapper_tlLb0ELb0E11__nv_dl_tagIPFvR6DomainEXadL_ZN72_INTERNAL_50__tmp_tmpxft_00210c2
0.002	Kokkos::parallel_reduce 18__nv_hdl_wrapper_tlLb0ELb0E11__nv_dl_tagIPFvR6DomainEXadL_ZN72_INTERNAL_50__tmp_tmpxft_00210c2
0.002	Kokkos::parallel_reduce 18__nv_hdl_wrapper_tlLb0ELb0E11__nv_dl_tagIPFvR6DomainPddEXadL_ZN72_INTERNAL_50__tmp_tmpxft_002
0.002	Kokkos::parallel_reduce 18__nv_hdl_wrapper_tlLb0ELb0E11__nv_dl_tagIPFvR6DomainEXadL_ZN72_INTERNAL_50__tmp_tmpxft_00210c2
0.002	Kokkos::parallel_for CalcMonotonicQGradientsForElems [type = Cuda, device = 0]
0.002	Kokkos::parallel_for CalcForceForNodes [type = Cuda, device = 0]
0.002	Kokkos::parallel_for InitStressTermsForElems [type = Cuda, device = 0]
0.002	Kokkos::parallel_for CalcAccelerationForNodes [type = Cuda, device = 0]
0.001	Kokkos::parallel_for UpdateVolumesForElems [type = Cuda, device = 0]
0.001	Kokkos::parallel_for CalcKinematicsForElems [type = Cuda, device = 0]
0.001	Kokkos::parallel_for ApplyMaterialPropertiesForElems A [type = Cuda, device = 0]
0.001	Kokkos::parallel_for IntegrateStressForElems A [type = Cuda, device = 0]
0.001	cudaGetDeviceProperties
0.001	Kokkos::parallel_for IntegrateStressForElems B [type = Cuda, device = 0]
0.001	Kokkos::parallel_for ApplyAccelerationBoundaryConditionsForNodes A [type = Cuda, device = 0]
0.001	Kokkos::parallel_for CalcVelocityForNodes [type = Cuda, device = 0]
0.001	Kokkos::parallel_for CalcFBHourglassForceForElems A [type = Cuda, device = 0]
0.001	Kokkos::parallel_for ApplyAccelerationBoundaryConditionsForNodes B [type = Cuda, device = 0]

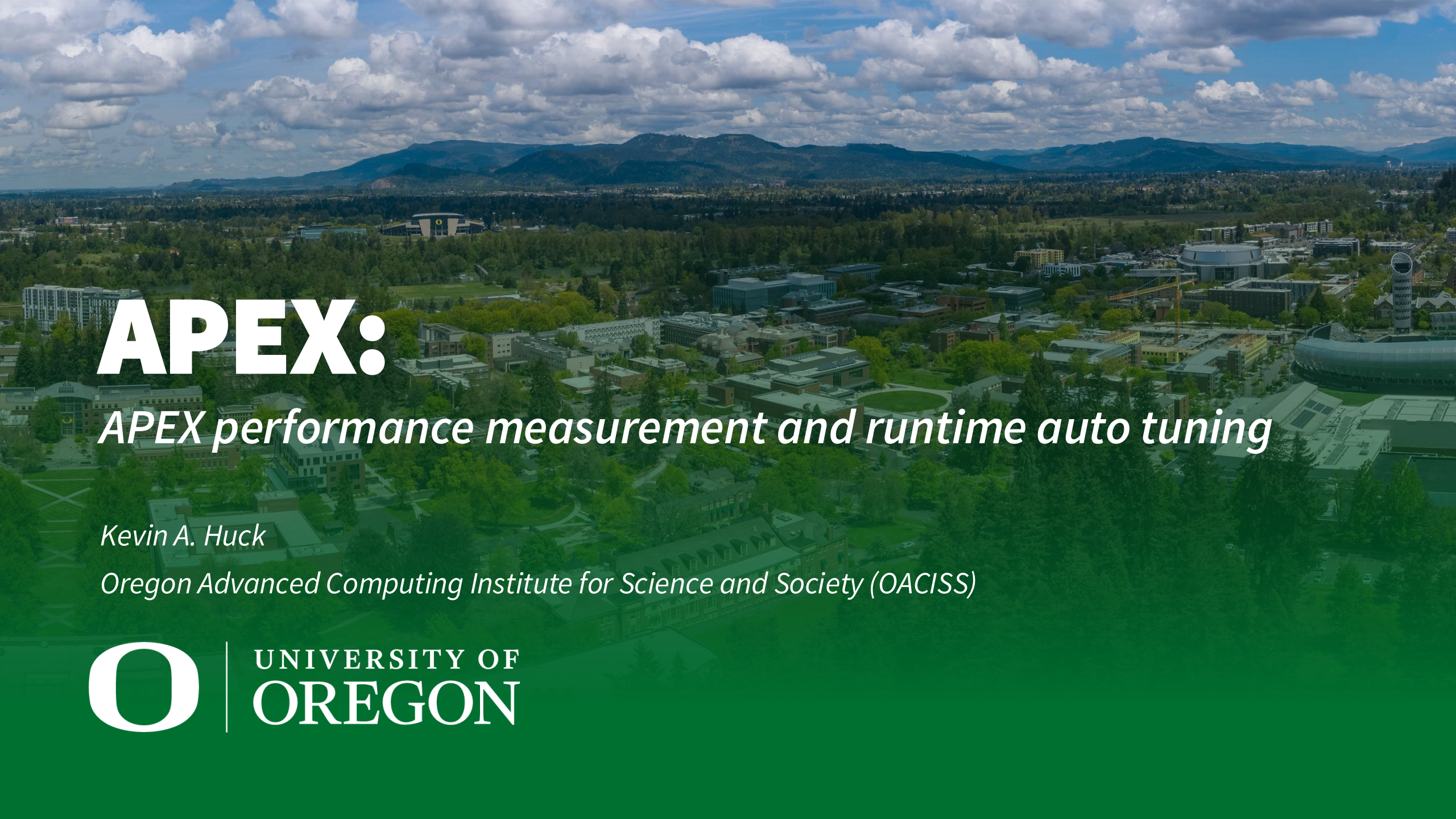
Main thread launching kernels

Metric: TAUGPU_TIME
Value: Exclusive
Units: seconds

0.876	void Kokkos::Impl::cuda_parallel_launch_constant_memory<Kokkos::Impl::ParallelFor<_INTERNAL_019bd3e3_9_lulesl
0.75	void Kokkos::Impl::cuda_parallel_launch_constant_memory<Kokkos::Impl::ParallelReduce<Kokkos::Impl::CombinedFu
0.746	void Kokkos::Impl::cuda_parallel_launch_constant_memory<Kokkos::Impl::ParallelFor<_INTERNAL_019bd3e3_9_lulesl
0.726	void Kokkos::Impl::cuda_parallel_launch_constant_memory<Kokkos::Impl::ParallelFor<_INTERNAL_019bd3e3_9_lulesl
0.687	Context Synchronize
0.654	Memory copy Host to Device
0.637	void Kokkos::Impl::cuda_parallel_launch_constant_memory<Kokkos::Impl::ParallelFor<_INTERNAL_019bd3e3_9_lulesl
0.444	Memory copy Device to Host
0.317	void Kokkos::Impl::cuda_parallel_launch_constant_memory<Kokkos::Impl::ParallelFor<_INTERNAL_019bd3e3_9_lulesl
0.315	void Kokkos::Impl::cuda_parallel_launch_constant_memory<Kokkos::Impl::ParallelFor<_INTERNAL_019bd3e3_9_lulesl
0.239	void Kokkos::Impl::cuda_parallel_launch_constant_memory<Kokkos::Impl::ParallelFor<_INTERNAL_019bd3e3_9_lulesl
0.225	void Kokkos::Impl::cuda_parallel_launch_constant_memory<Kokkos::Impl::ParallelFor<CalcKinematicsForElems(Domai
0.164	void Kokkos::Impl::cuda_parallel_launch_constant_memory<Kokkos::Impl::ParallelFor<_INTERNAL_019bd3e3_9_lulesl
0.082	void Kokkos::Impl::cuda_parallel_launch_constant_memory<Kokkos::Impl::ParallelReduce<Kokkos::Impl::CombinedFu
0.079	void Kokkos::Impl::cuda_parallel_launch_constant_memory<Kokkos::Impl::ParallelFor<_INTERNAL_019bd3e3_9_lulesl
0.077	void Kokkos::Impl::cuda_parallel_launch_constant_memory<Kokkos::Impl::ParallelFor<_INTERNAL_019bd3e3_9_lulesl
0.076	void Kokkos::Impl::cuda_parallel_launch_constant_memory<Kokkos::Impl::ParallelFor<_INTERNAL_019bd3e3_9_lulesl
0.073	void Kokkos::Impl::cuda_parallel_launch_constant_memory<Kokkos::Impl::ParallelFor<_INTERNAL_019bd3e3_9_lulesl
0.067	void Kokkos::Impl::cuda_parallel_launch_constant_memory<Kokkos::Impl::ParallelReduce<Kokkos::Impl::CombinedFu
0.061	void Kokkos::Impl::cuda_parallel_launch_constant_memory<Kokkos::Impl::ParallelFor<_INTERNAL_019bd3e3_9_lulesl
0.046	void Kokkos::Impl::cuda_parallel_launch_constant_memory<Kokkos::Impl::ParallelReduce<Kokkos::Impl::CombinedFu
0.043	void Kokkos::Impl::cuda_parallel_launch_constant_memory<Kokkos::Impl::ParallelFor<_INTERNAL_019bd3e3_9_lulesl
0.022	void Kokkos::Impl::cuda_parallel_launch_constant_memory<Kokkos::Impl::ParallelFor<_INTERNAL_019bd3e3_9_lulesl
0.022	Stream Synchronize
0.019	void Kokkos::Impl::cuda_parallel_launch_constant_memory<Kokkos::Impl::ParallelReduce<Kokkos::Impl::CombinedFu
0.019	void Kokkos::Impl::cuda_parallel_launch_constant_memory<Kokkos::Impl::ParallelReduce<Kokkos::Impl::CombinedFu
0.018	void Kokkos::Impl::cuda_parallel_launch_local_memory<Kokkos::Impl::ParallelReduce<Kokkos::Impl::CombinedFuncto
0.017	void Kokkos::Impl::cuda_parallel_launch_constant_memory<Kokkos::Impl::ParallelFor<_INTERNAL_019bd3e3_9_lulesl
0.016	void Kokkos::Impl::cuda_parallel_launch_constant_memory<Kokkos::Impl::ParallelFor<_INTERNAL_019bd3e3_9_lulesl
0.012	Event Synchronize
0.011	void Kokkos::Impl::cuda_parallel_launch_local_memory<Kokkos::Impl::ParallelFor<_INTERNAL_019bd3e3_9_lulesh_cc
0.01	void Kokkos::Impl::cuda_parallel_launch_local_memory<Kokkos::Impl::ParallelFor<_INTERNAL_019bd3e3_9_lulesh_cc
9.5E-4	void Kokkos::Impl::cuda_parallel_launch_constant_memory<Kokkos::Impl::ParallelFor<_INTERNAL_019bd3e3_9_lulesl
5.2E-4	void Kokkos::Impl::cuda_parallel_launch_constant_memory<Kokkos::Impl::ParallelFor<_INTERNAL_019bd3e3_9_lulesl
5.0E-4	void Kokkos::Impl::cuda_parallel_launch_constant_memory<Kokkos::Impl::ParallelFor<_INTERNAL_019bd3e3_9_lulesl
1.0E-4	void Kokkos::Impl::cuda_parallel_launch_local_memory<Kokkos::Impl::ParallelFor<Kokkos::Impl::ViewFill<Kokkos::Vie
4.4E-5	desul::(anonymous namespace)::init_lock_arrays_cuda_kernel()
4.0E-6	Kokkos::Impl::(anonymous namespace)::query_cuda_kernel_arch(int*)
3.8E-6	void Kokkos::Impl::cuda_parallel_launch_local_memory<Kokkos::Impl::ParallelFor<Kokkos::Impl::ViewFill<Kokkos::Vie

Virtual thread with CUDA activity

Intel Xeon system with NVIDIA A100, size 256, 100 iterations, one rank



APEX:

APEX performance measurement and runtime auto tuning

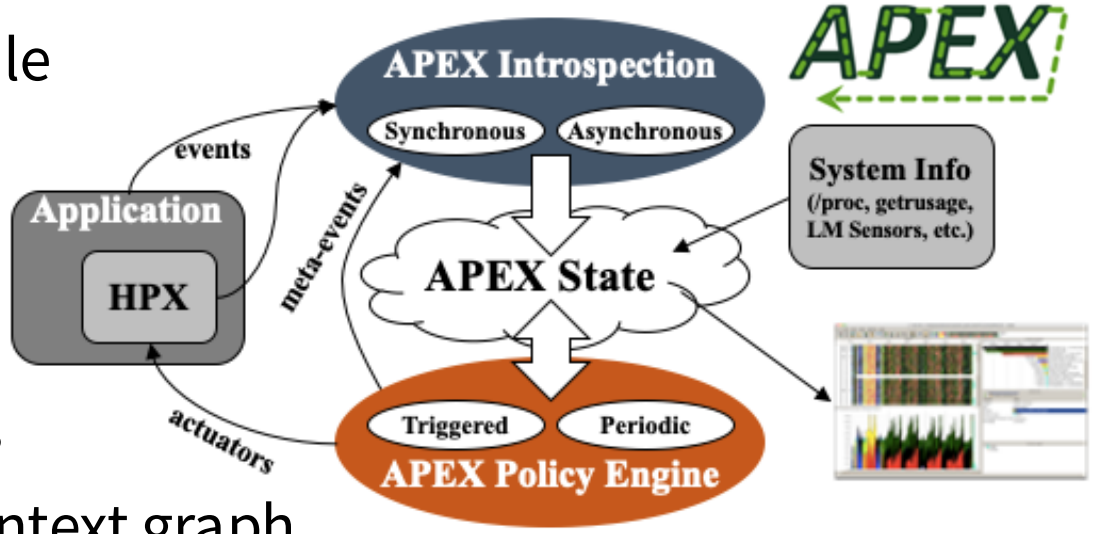
Kevin A. Huck

Oregon Advanced Computing Institute for Science and Society (OACISS)

UNIVERSITY OF OREGON

Autonomic Performance Environment for Exascale (APEX)

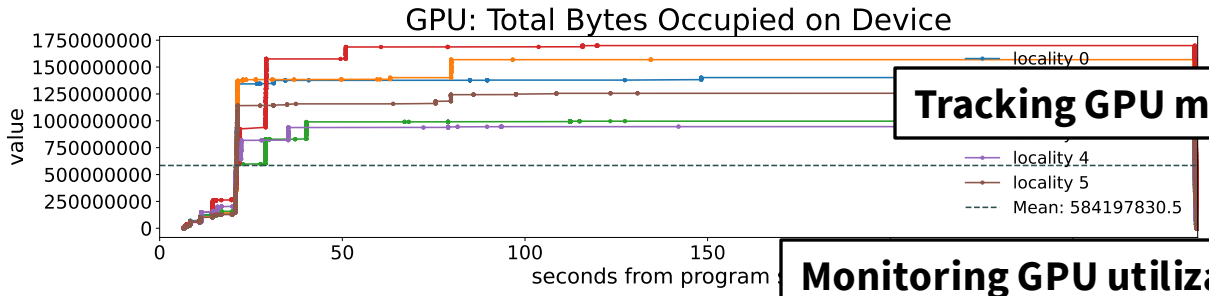
- Autonomic Performance Environment for eXascale
- Performance Measurement
- **Runtime Adaptation**
- Designed for AMT runtimes (originally HPX)
  - but works with conventional parallel models



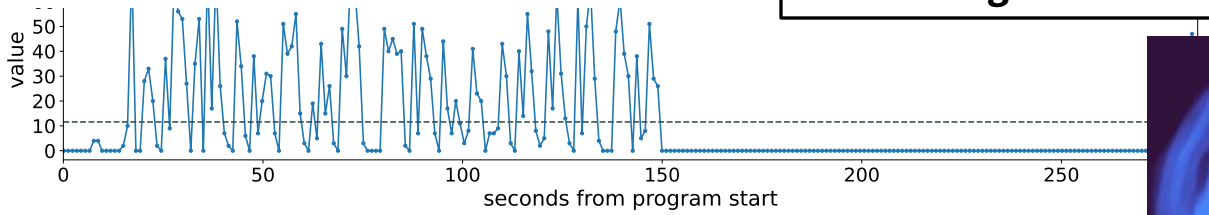
- Focus on **task dependency** graph, not calling context graph
- Supports HPX, C/C++ threads, OpenMP, OpenACC, Kokkos, Raja, CUDA, HIP, SYCL, StarPU... Working on Iris, PaRSEC
- https://github.com/UO-OACISS/apex and https://github.com/khuck/apex-tutorial
- Active Harmony* (Nelder Mead), simulated annealing, genetic search, hill climbing, native Nelder-Mead for parametric search methods

https://doi.org/10.1109/ESPM256814.2022.00008 : "Broad Performance Measurement Support for Asynchronous Multi-Tasking with APEX", Huck, ESPM, 2022

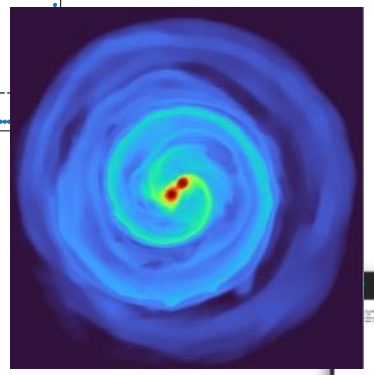
APEX example – Octo-Tiger (Octree astrophysics in HPX, Kokkos)

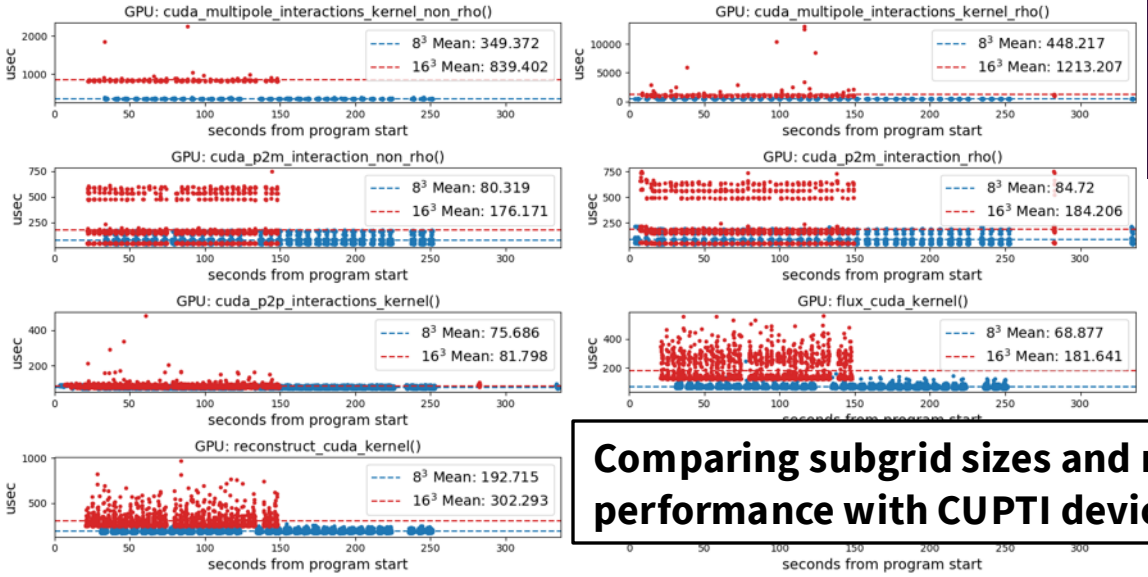


Tracking GPU memory usage with CUPTI

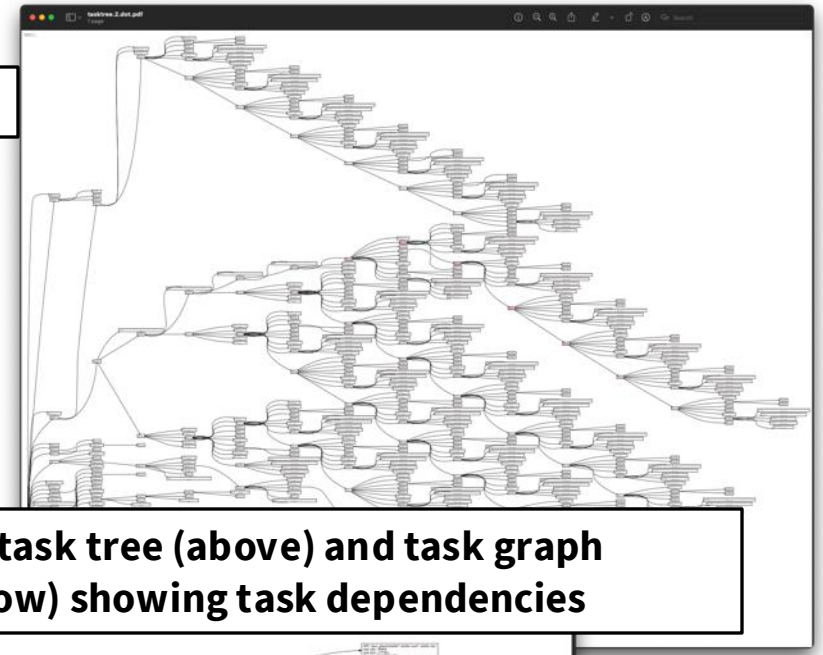


Monitoring GPU utilization with NVML library

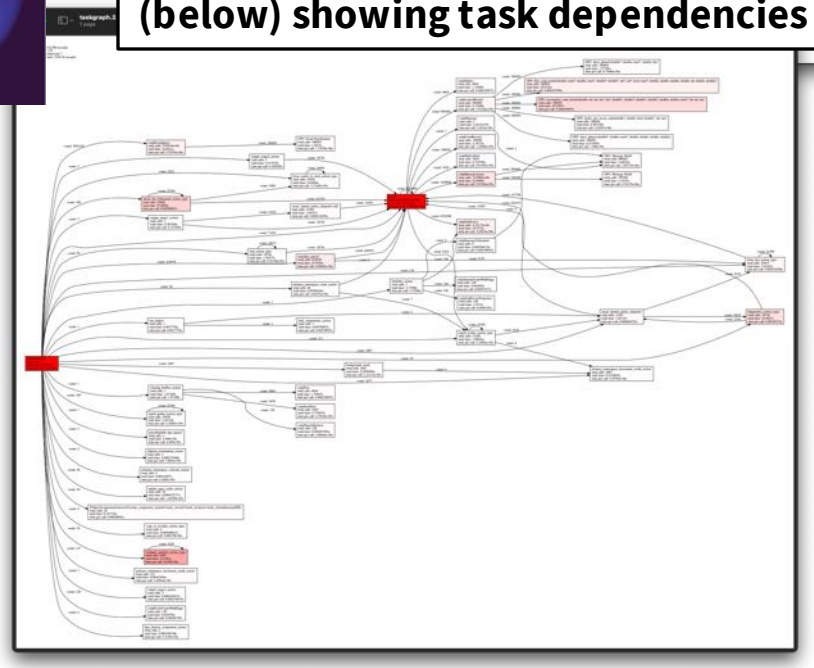




Comparing subgrid sizes and relative kernel performance with CUPTI device activity



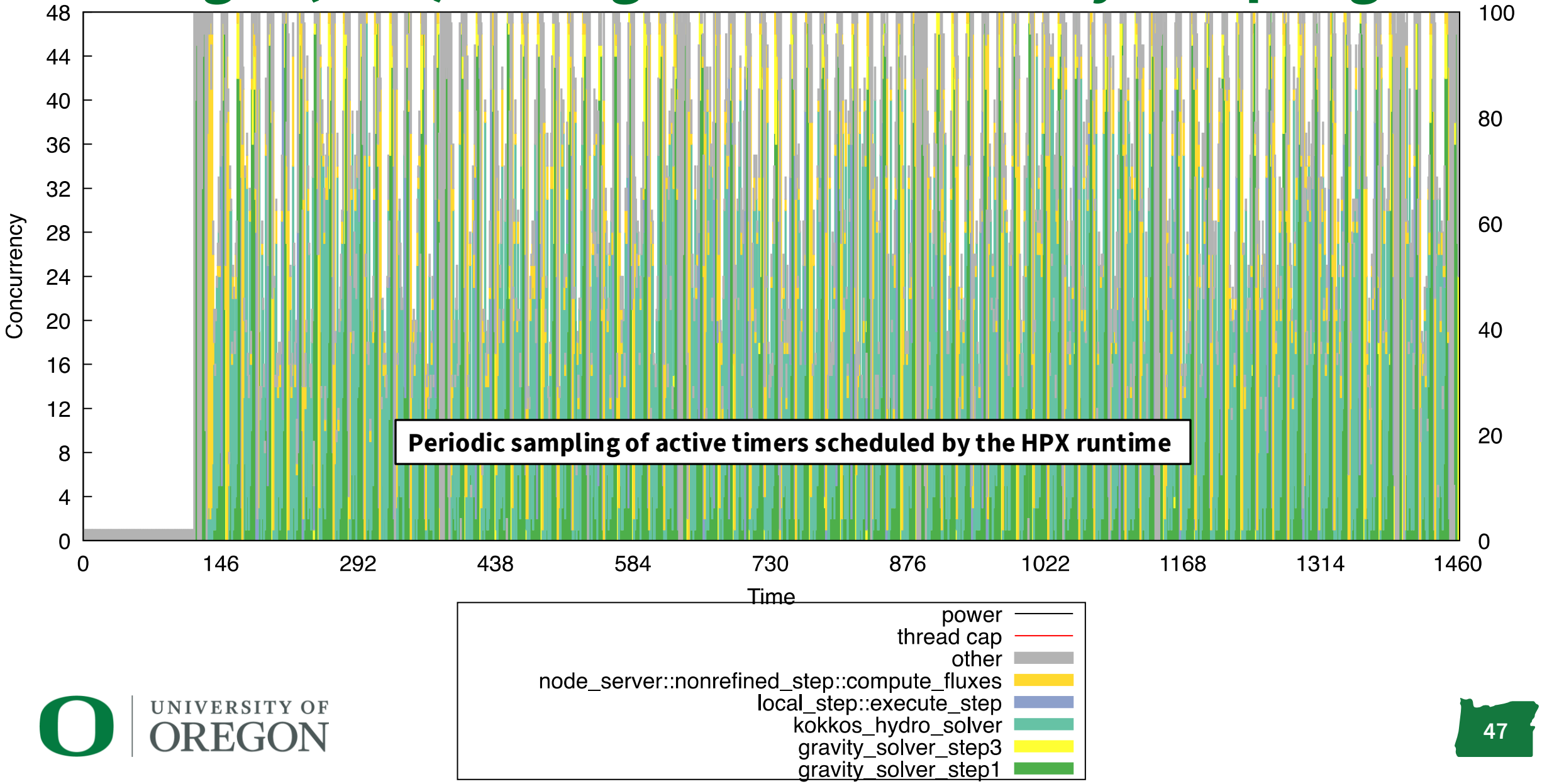
Full task tree (above) and task graph (below) showing task dependencies



https://github.com/STEllAR-GROUP/octotiger
https://github.com/STEllAR-GROUP/hpx

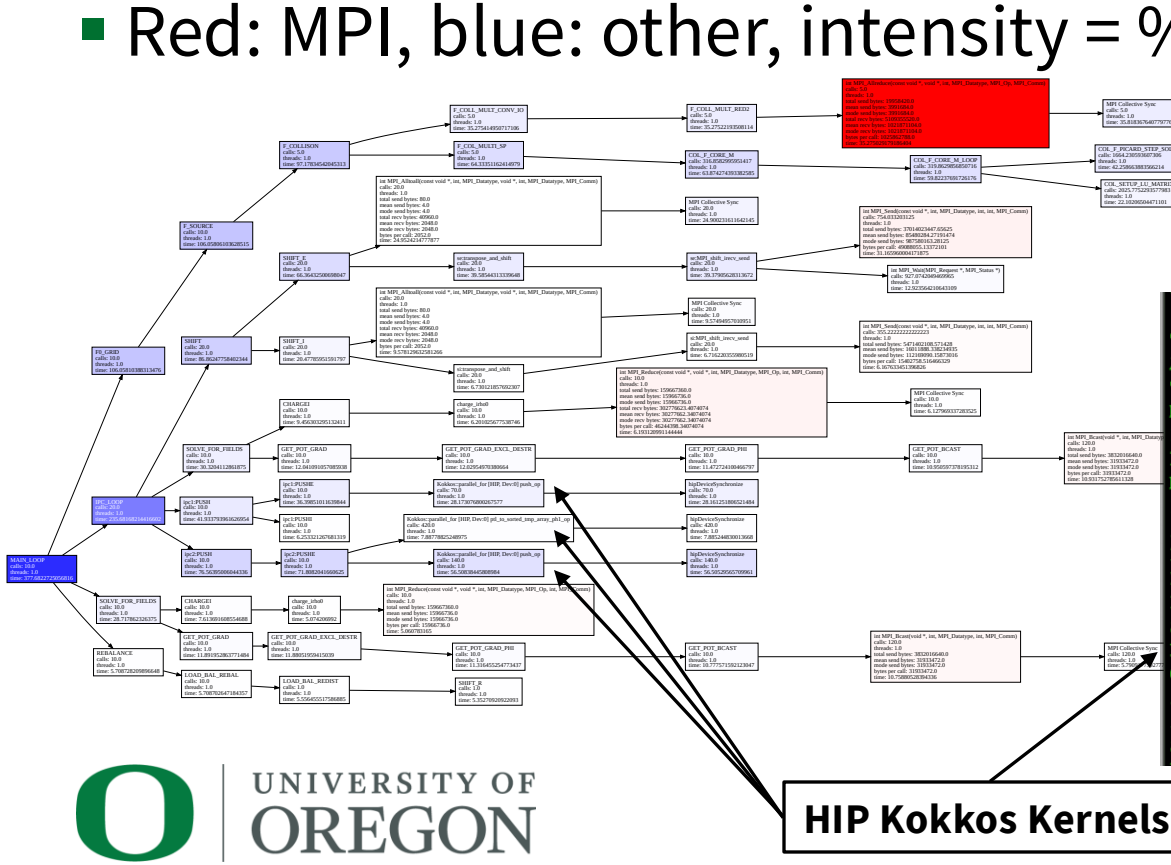
UNIVERSITY OF OREGON

Octotiger (HPX) on Fugaku – concurrency sampling



Example: XGC (tokamak plasma fusion PIC) on Frontier, 512 ranks

- Uses support for MPI, OpenMP-Tools, PerfStubs, Kokkos, Hip
- Post-processing view of MAIN_LOOP subtree, only with accumulated times > 5.0 seconds (only 72 nodes of 6298 of full task tree)
- Red: MPI, blue: other, intensity = % of total subtree



OpenMP Kokkos Kernels

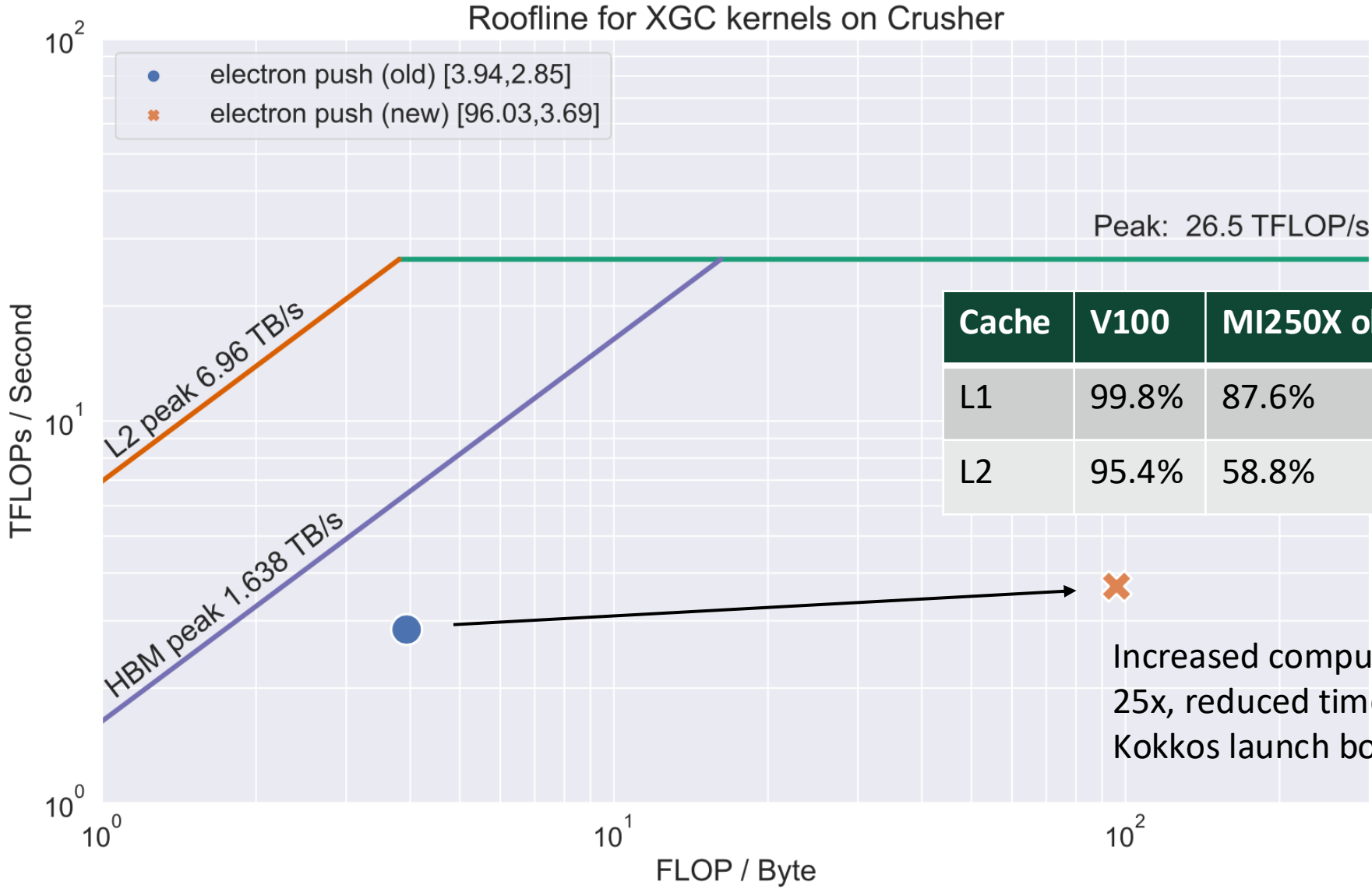
HIP Kokkos Kernels

Timer	total	calls	tot/call	yields	threads	tot/thr	%total	%wall
APEX MAIN	588.25	1.00	588.25	0.00	1.00	588.25	100.00	100.00
TOTAL	587.73	1.00	587.73	0.00	1.00	587.73	99.91	99.91
MAIN_LOOP	377.68	10.00	37.77	0.00	1.00	377.68	64.20	64.20
IPC_LOOP	235.68	20.00	11.78	0.00	1.00	235.68	40.06	40.06
hipDeviceSynchronize	149.40	168600.08	0.00	0.00	1.00	149.40	25.40	25.40
MPI Collective Sync	132.45	20064.08	0.01	0.00	1.00	132.45	22.52	22.52
INIT	125.85	1.00	125.85	0.00	1.00	125.85	21.39	21.39
i:SETUP	125.85	1.00	125.85	0.00	1.00	125.85	21.39	21.39
F0_GRID	106.06	10.00	10.61	0.00	1.00	106.06	18.03	18.03
F_SOURCE	106.06	10.00	10.61	0.00	1.00	106.06	18.03	18.03
F_COLLISON	97.18	5.00	19.44	0.00	1.00	97.18	16.52	16.52
SHIFT	86.86	20.00	4.34	0.00	1.00	86.86	14.77	14.77
Kokkos::parallel_for [HIP, Dev:0] pus...	86.76	230.00	0.38	0.00	1.00	86.76	14.75	14.75
GPU: void Kokkos::Impl::hip_parallel_...	84.64	210.00	0.40	0.00	1.00	84.64	14.39	14.39
ipc2:PUSH	76.56	10.00	7.66	0.00	1.00	76.56	13.02	13.02
ipc2:PUSHE	71.81	10.00	7.18	0.00	1.00	71.81	12.21	12.21
SHIFT_E	68.95	23.00	3.00	0.00	1.00	68.95	11.72	11.72

Profiles collected with apex_exec, then post-processed with Python

XGC: Push Kernel on Crusher/Frontier, Kokkos helped generate roofline

Kokkos + APEX + PAPI + rocprofiler = ☺

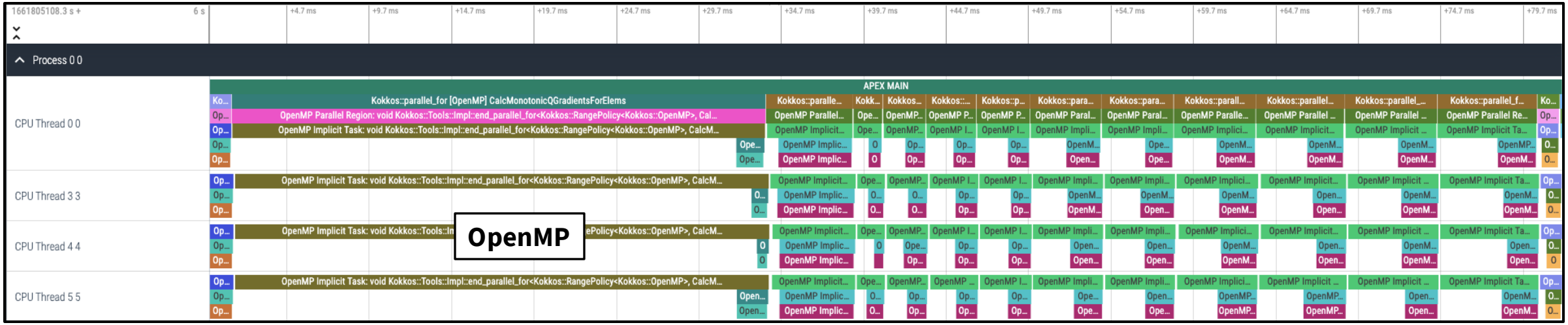


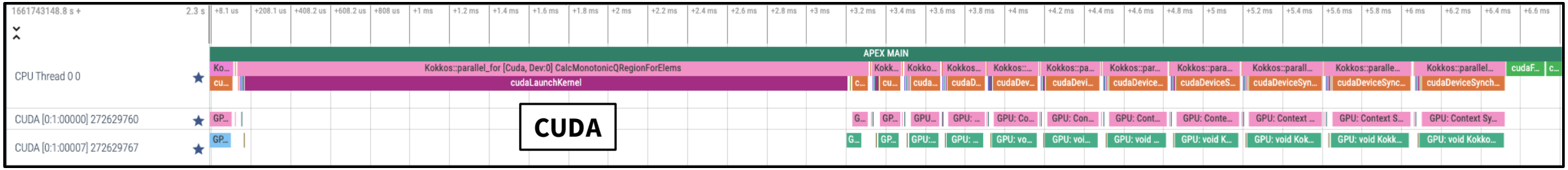
Cache	V100	MI250X old	MI250X new
L1	99.8%	87.6%	98.7%
L2	95.4%	58.8%	94.1%

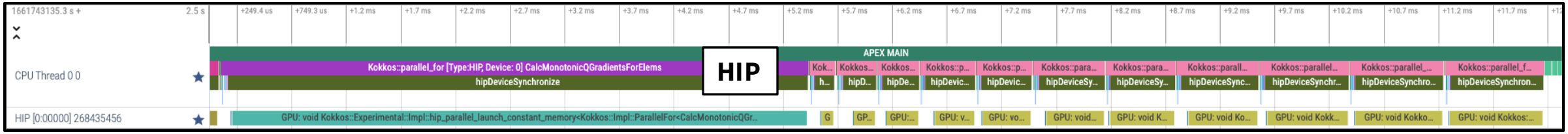
Increased computational intensity 25x, reduced time by 22% by using Kokkos launch bounds of <256,2>

Profiles collected with apex_exec, then post-processed with Python

Kokkos Lulesh and APEX Tracing – OpenMP, CUDA, HIP







apex_exec --apex:gtrace --apex:[ompt,cuda,hip] …
Perfetto trace viewer: https://ui.perfetto.dev/

Using APEX on Frontier

- module load ums ums002/vanilla ; module load apex
- `srun <srun-options> apex_exec <apex-options> executable <args>`
- Run ‘`apex_exec`’ without arguments for full list of options
- For demo exercises: https://github.com/khuck/apex-tutorial
- For runtime auto-tuning examples: https://github.com/khuck/apex-kokkos-tuning

Kokkos Support in APEX

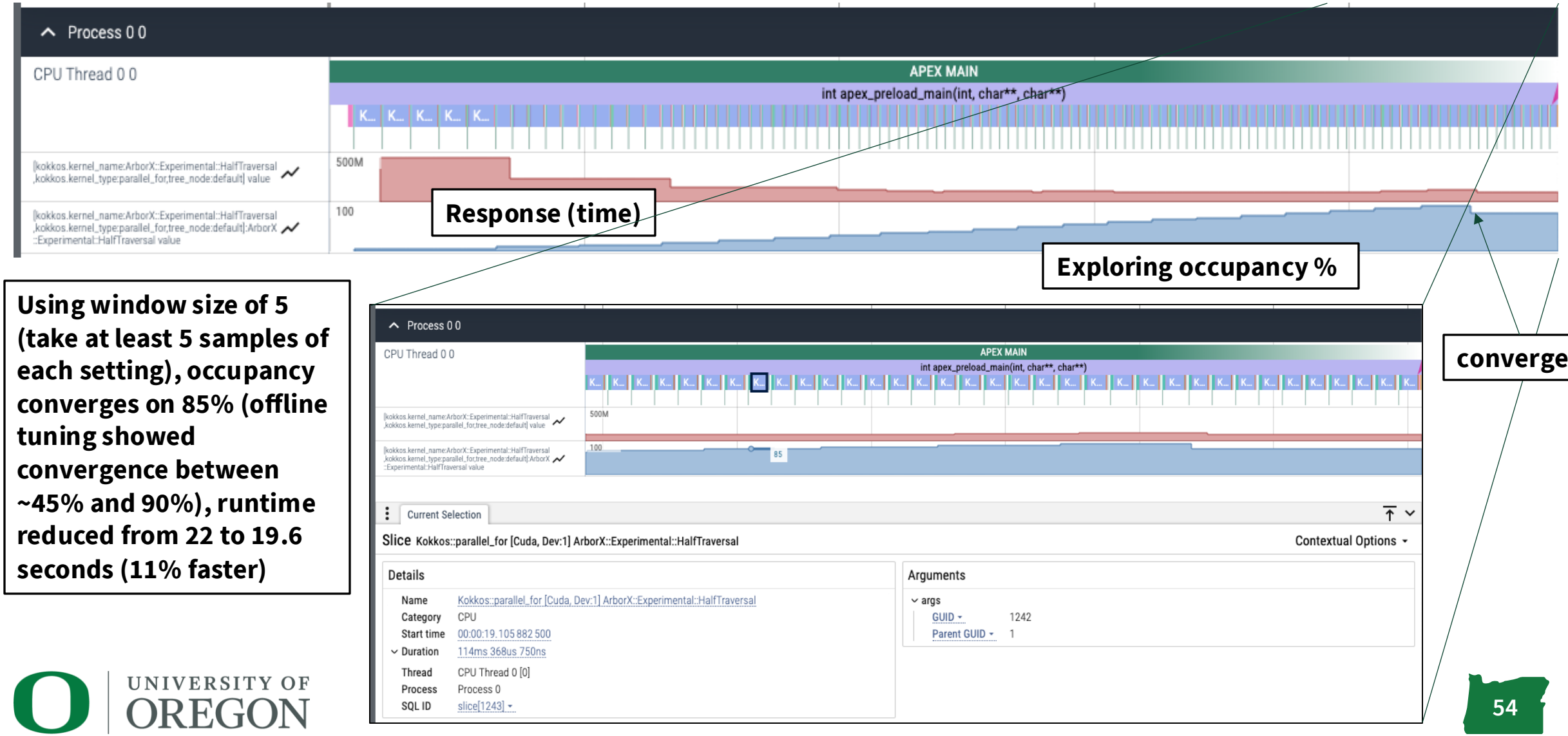
- APEX implements the same profiling API that TAU does, *and…*
- APEX provides runtime auto-tuning (parametric search) support
- Kokkos provides the ability to autotune with:

```
-DKokkos_ENABLE_TUNING=ON
```

- Automatically provides input and context variables for parallel_for, parallel_reduce, parallel_scan, parallel_copy.
  - TeamPolicy: team size and vector length
  - MDRangePolicy: tile sizes (block.x, block.y, block.z), 2 through 6 dimensions
  - RangePolicy*: block size
  - All policies: occupancy

*(in a long-dormant development fork/branch…working on new integration because many kernels use RangePolicy)

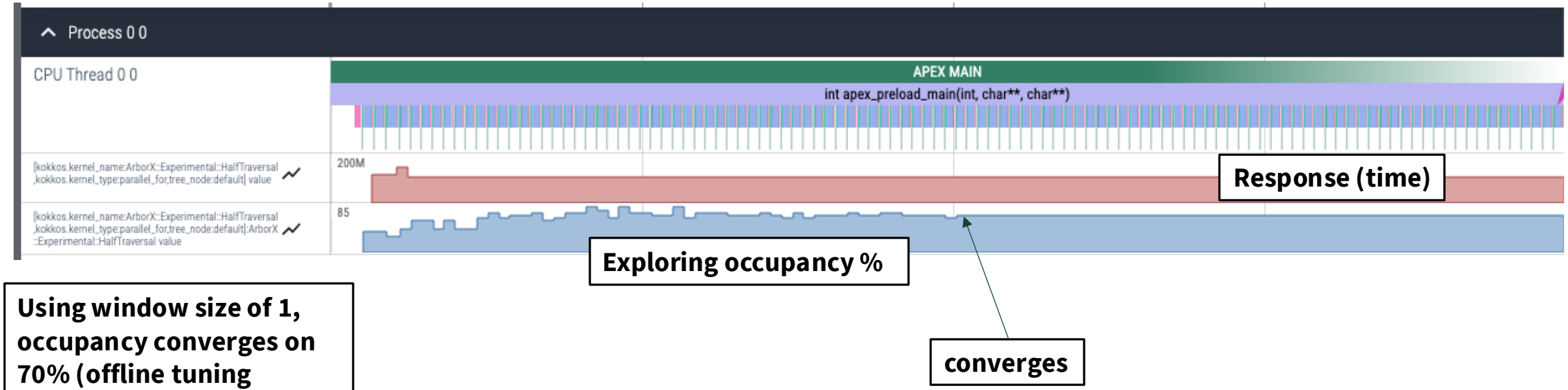
APEX Auto-tuning of ArborX DBScan with exhaustive search



Using window size of 5 (take at least 5 samples of each setting), occupancy converges on 85% (offline tuning showed convergence between ~45% and 90%), runtime reduced from 22 to 19.6 seconds (11% faster)

UNIVERSITY OF OREGON

APEX Auto-tuning of ArborX DBScan with simulated annealing search



Using window size of 1, occupancy converges on 70% (offline tuning showed convergence between ~45% and 90%), runtime reduced from 22 to 19.6 seconds (11% faster)

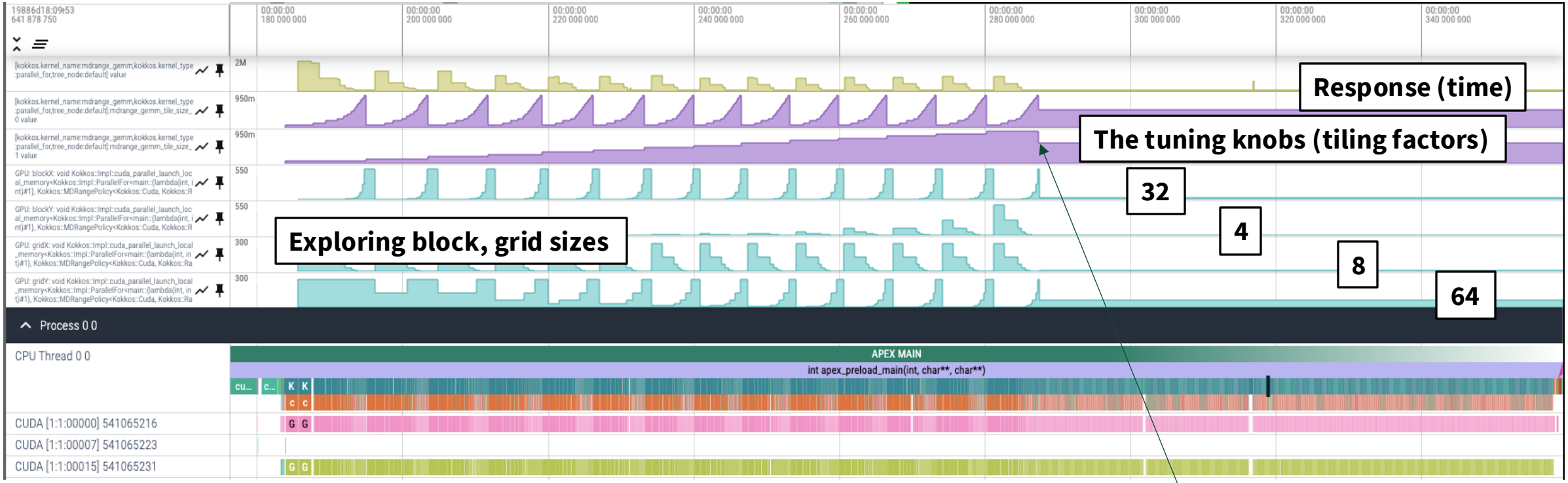
Auto-Tuning test cases / examples

- https://github.com/khuck/apex-kokkos-tuning
  - mdrange_gemm – demonstrates tuning of tiling parameters for MDRangePolicy
  - mdrange_gemm_occupancy – demonstrates occupancy tuning for MDRangePolicy
  - occupancy – demonstrates occupancy tuning for RangePolicy
  - mm2d_tiling – complex explicit tuning example for OpenMP parameters
    - Thread count, schedule, chunk size
  - idk_jmm – elegant, complex example that selects between two matrix multiplication implementations (TeamPolicy or MDRangePolicy), while tuning internals each for best configuration.
  - 1d_stencil, 2d_stencil, deep_copy examples
- Configure & Build:
  - See README.md for latest info (https://github.com/khuck/apex-kokkos-tuning/)

Building apex-kokkos-tuning

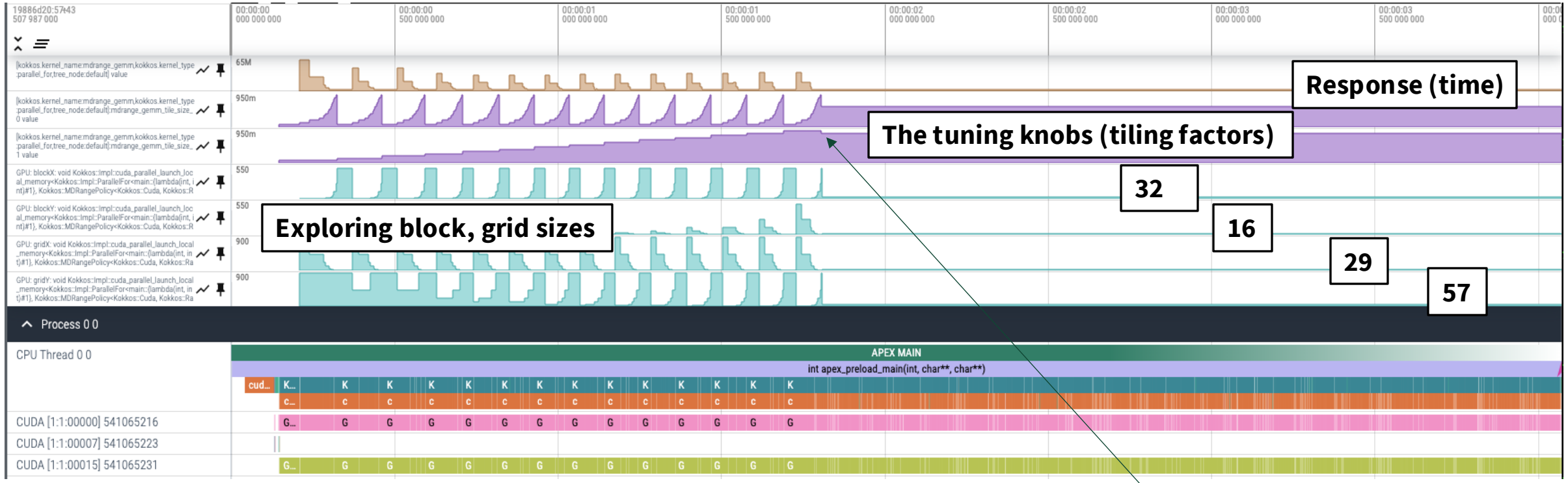
- `generic-cuda.sh` or `perlmutter.sh` for CUDA examples
- `frontier.sh` for ROCm example
- General idea: this is just a CMake compound project including the Kokkos and APEX projects. Any CMake variables you would pass in to those projects you pass in to this one. With specific notes:
  - `-DKokkos_ENABLE_TUNING=ON` is required for tuning

Example: testing mdrange_gemm 256x256 with exhaustive search



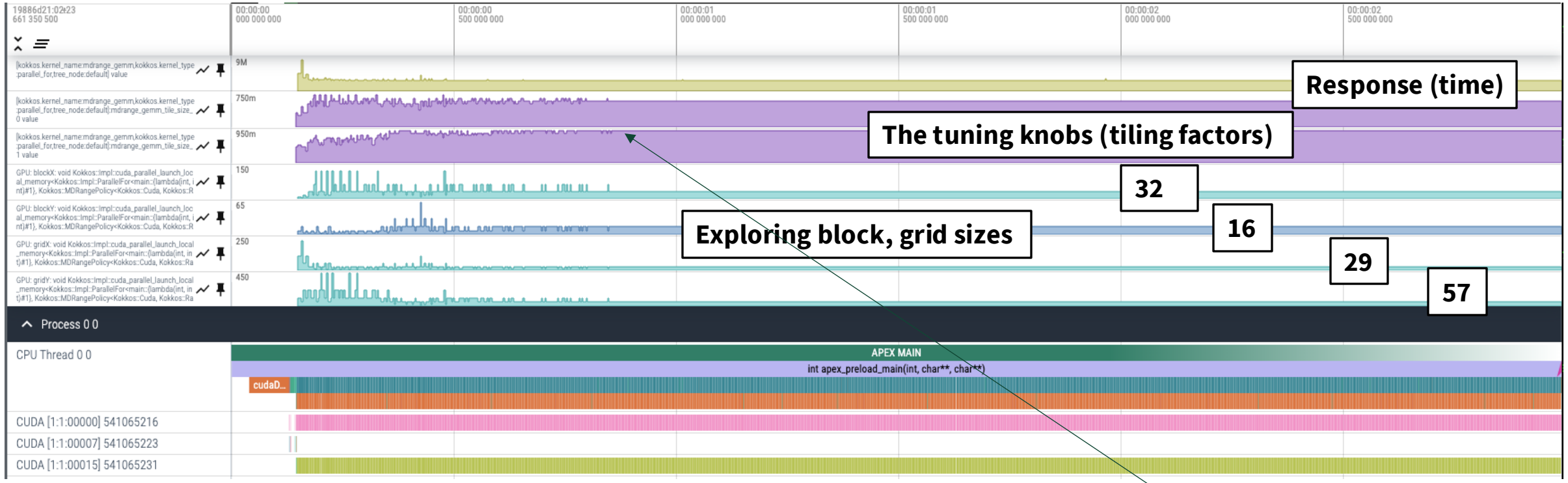
Using window size of 2, tiling converges on block dimensions of (32,4,1) and grid dimensions (8,64,1), the defaults are (16,2,1) and (16,128,1) respectively – simulated annealing converges to the same values

Example: testing mdrange_gemm 900x900 with exhaustive search



Using window size of 2, tiling converges on block dimensions of (32,16,1) and grid dimensions (29,57,1), the defaults are (16,2,1) and (57,450,1) respectively. The runtime improved from a baseline of 3.95 seconds to 3.04 seconds using cached tuning results!

Example: testing mdrange_gemm 900x900 with simulated annealing



Using window size of 2, tiling converges on block dimensions of (32,16,1) and grid dimensions (29,57,1), the defaults are (16,2,1) and (57,450,1) respectively. The runtime improved from a baseline of 3.95 seconds to 3.17 seconds using runtime simulated annealing!

Acknowledgements

Parts of this research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative.

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

UNIVERSITY OF OREGON

Extra slides

Users need help…

NERSC slack:

can users get fine-grained statistics of their usage? Let's say I run a GPU job and I'd like to have a look retrospectively on things like memory usage, how much the GPUs are consuming in Watts or whatever, and other statistics like that. In a previous institution I was associated with, there was like a weird dashboard where one could get all sorts of interesting info, and it was pretty helpful to track stuff

I was hoping for something more like nvidia-smi...

Hi everybody, I am fairly new to the perlmutter system. I am wondering if there is some tool to monitor node performances like the jupiter online one.
My ideal use case is: launch a job, ssh to node running -> monitor performances [on cpu systems I would generally use htop].
This is not thought for continuous use, just to make sure that the settings are exploiting all node resources that they can. Thank you very much!

ALCF slack:

Are there any tools on Aurora to monitor GPU usage similar to `nvidia-smi` or `rocm-smi` ?

Quick quiz:

- Where does MPI rank 0 of the following get allocated/scheduled on 1 Frontier node?

salloc -N 1 --threads-per-core=1 …

```
Process Summary:
MPI 000 - PID 64341 - Node frontier00765 - CPUs allowed: [1,2,3,4,5,6,7]
```

srun -n1 -c7 --gpus-per-task=1 --gpu-bind=closest

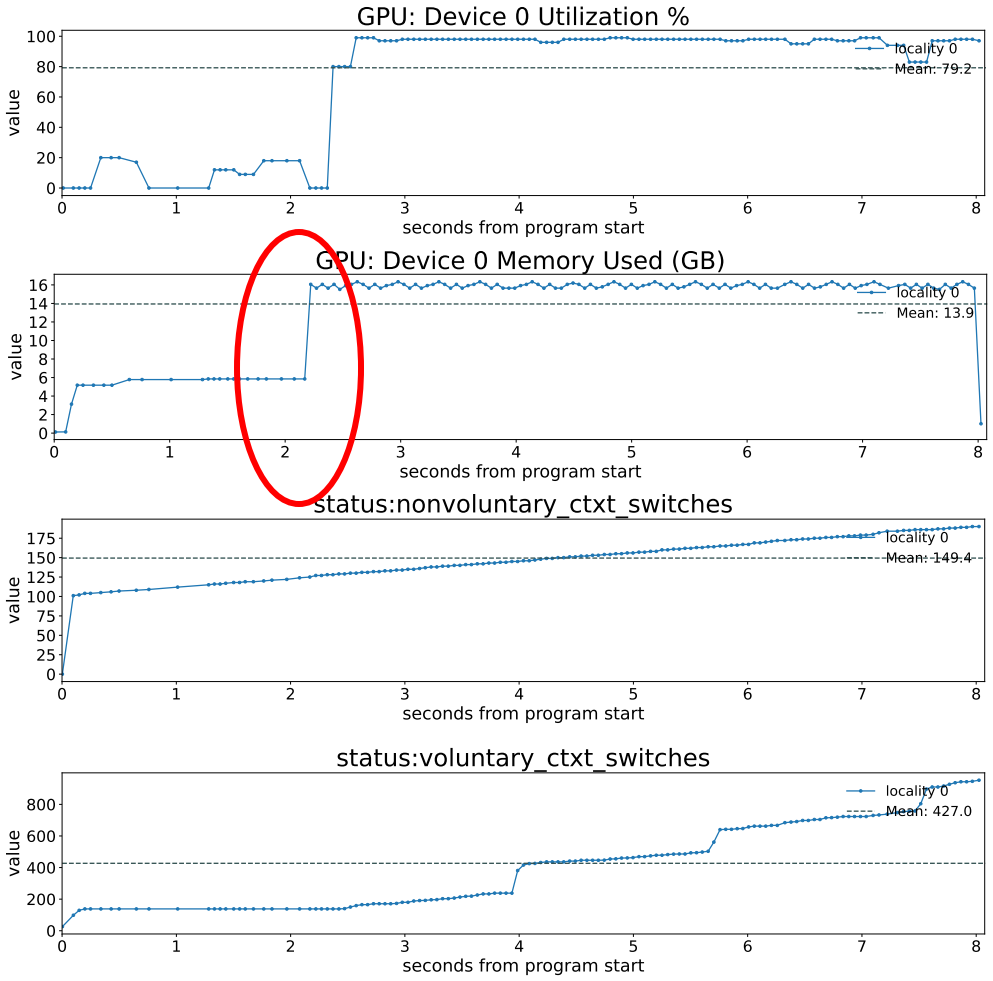
```
Process Summary:
MPI 000 - PID 64384 - Node frontier00765 - CPUs allowed: [49,50,51,52,53,54,55]

MPI 000 - Node frontier00765 - RT_GPU_ID 0 - GPU_ID 0 - Bus_ID 0000:c1:00.0
```

srun -n8 -c7 --gpus-per-task=1 --gpu-bind=closest

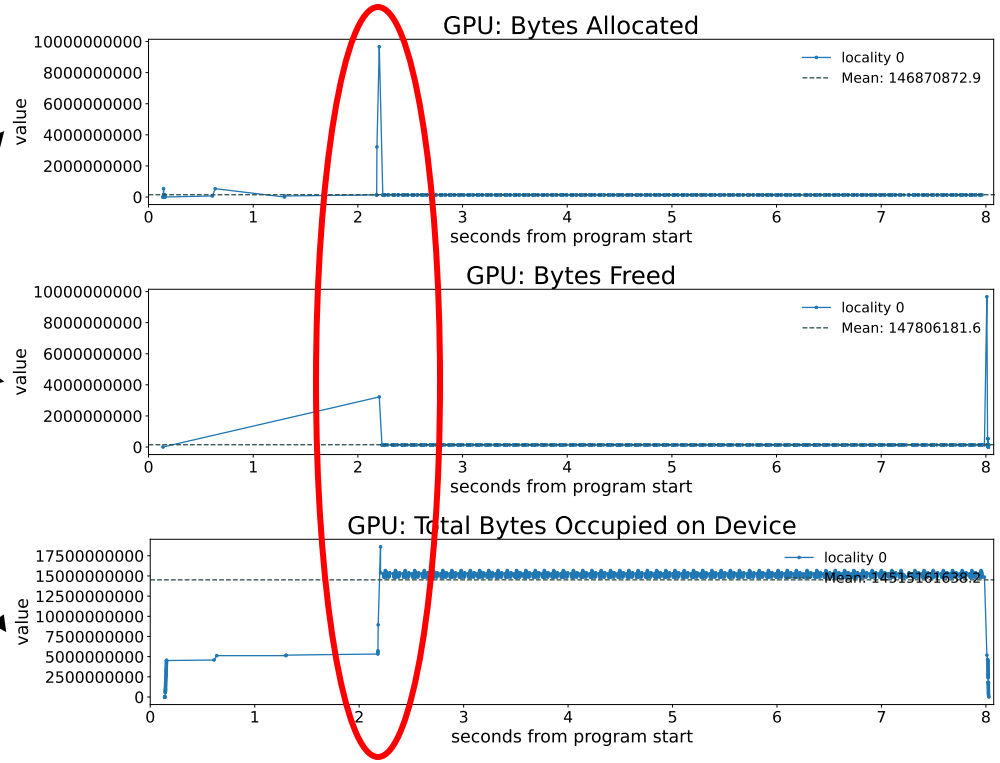
```
Process Summary:
MPI 000 - PID 64516 - Node frontier00765 - CPUs allowed: [1,2,3,4,5,6,7]

MPI 000 - Node frontier00765 - RT_GPU_ID 0 - GPU_ID 4 - Bus_ID 0000:d1:00.0
```

Caveat: Monitoring != Explicit Measurement

- Monitoring Data (periodic)



- Progress Data (events)



Example: APEX data from Lulesh Kokkos with CUDA back end, visualized with Python – clear need for GPU HWM

What is the hardware / operating system doing?

- Kernel monitoring is popular and useful (strace, ptrace, dtrace, dtruss, ftrace, KUtrace, kprobe, system-tap, KTAU, STaKTAU, eBPF, bpftrace, BCC, …)
- Nataraj, Morris, Malony, Sottile, and Beckman. "The Ghost in the Machine: Observing the Effects of Kernel Operation on Parallel Application Performance." In *SC2007*, pp. 1-12. 2007.
- Subsystem monitoring (Darshan)
- Did I ask for the right thing?
- Will I run out of a limited resource?
- Are there alternatives?
- Can I get this information *without being root*? *Without spawning another process* to monitor my process? *Without kernel patches/support*? Low/no overhead?

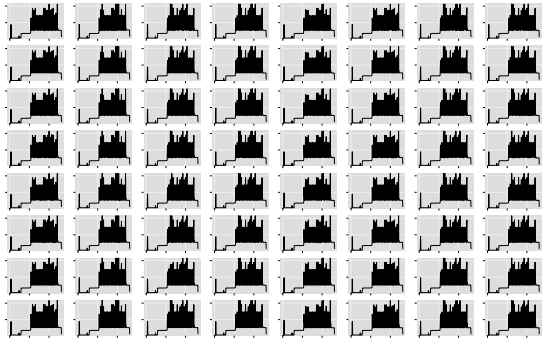
What (we think) users DON'T want:

(BTW, I am happy to ask for some help visualizing/simplifying the data…sparklines aren't going to cut it)

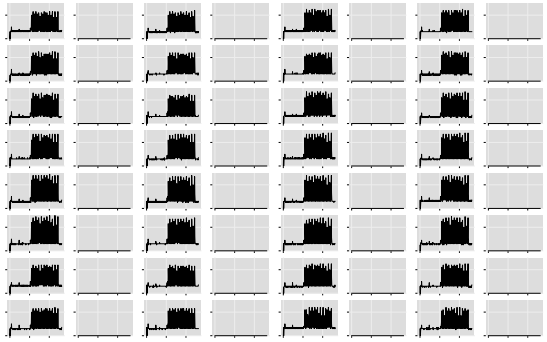
Sources: https://www.space.com/22652-nasa-redesigns-mission-control.html

Sparklines: 64 MPI Ranks on Frontier – need better

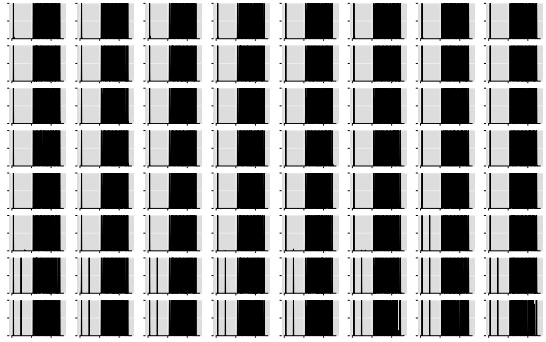
Used VRAM Bytes, range: [295,063,552, 21,668,823,040]



Energy Average (J), range: [0, 30]

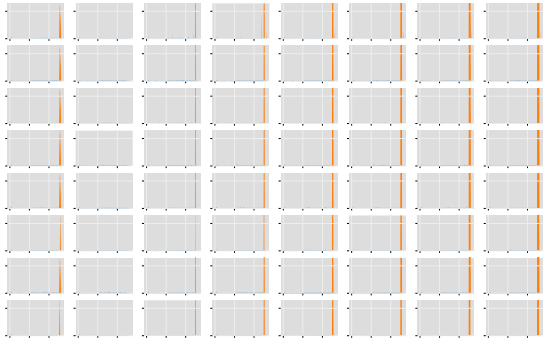


Device Busy %, range: [0, 100]

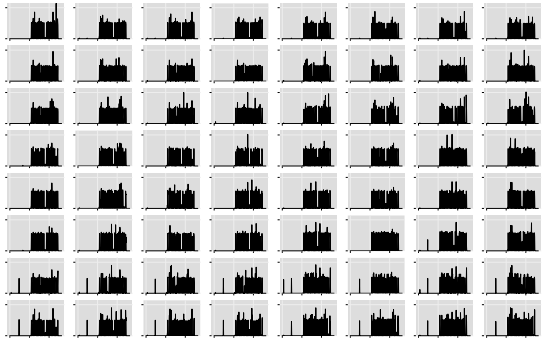


Voltage (mV), range: [712.0, 943.0]

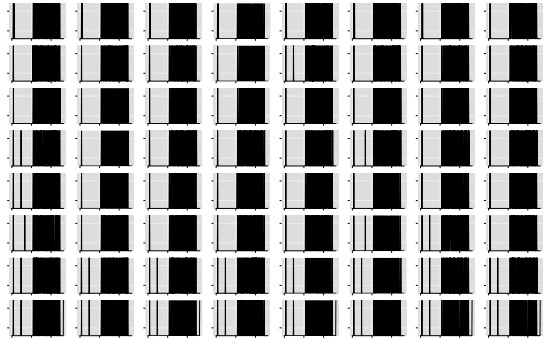
Context Switches, range: [0, 6,452]



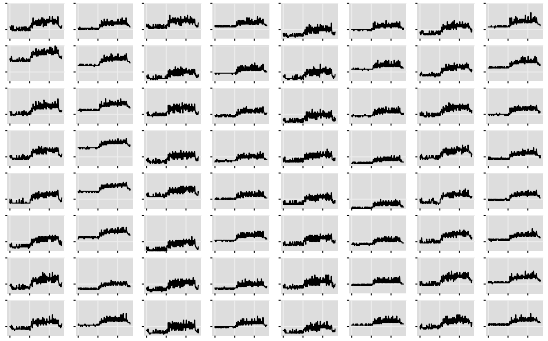
Memory Busy %, range: [0, 57]



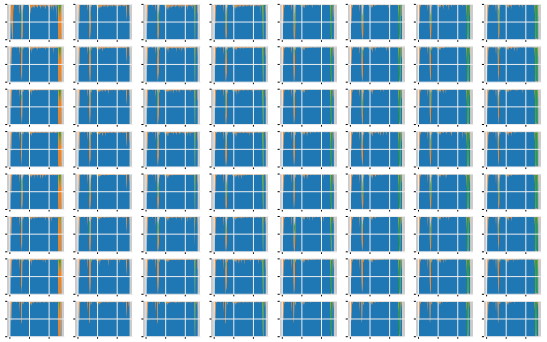
Clock Frequency, GLX (MHz), range: [800, 1,700]



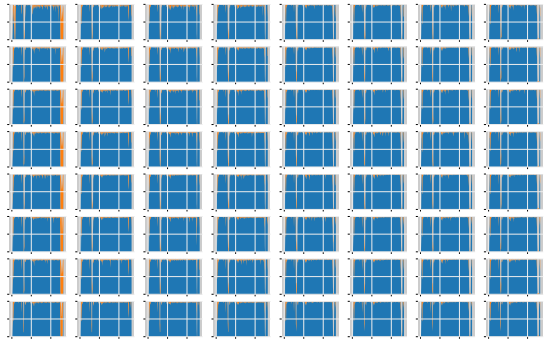
Temperature (C), range: [33.0, 60.0]



HWT utilization

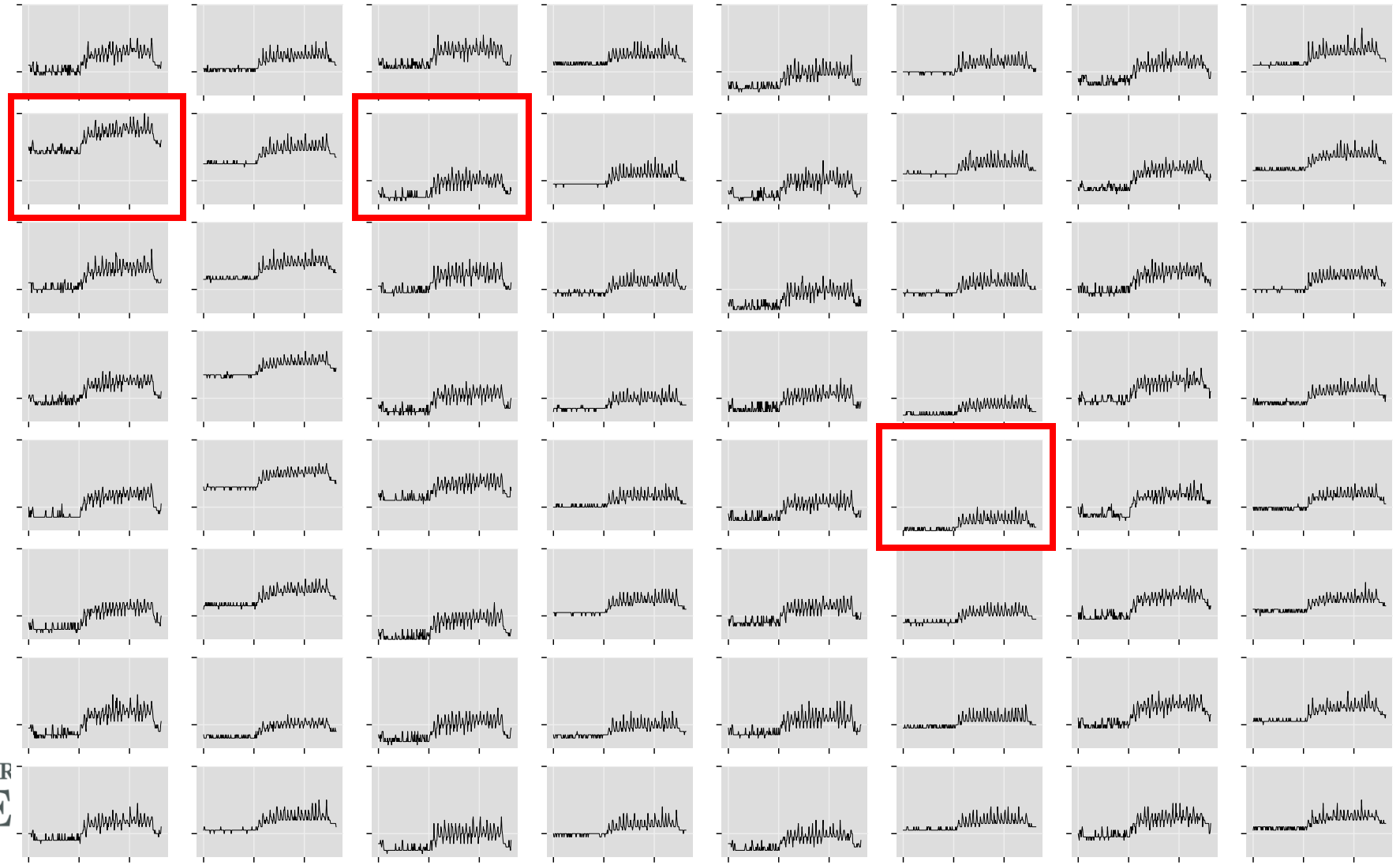


LWP utilization



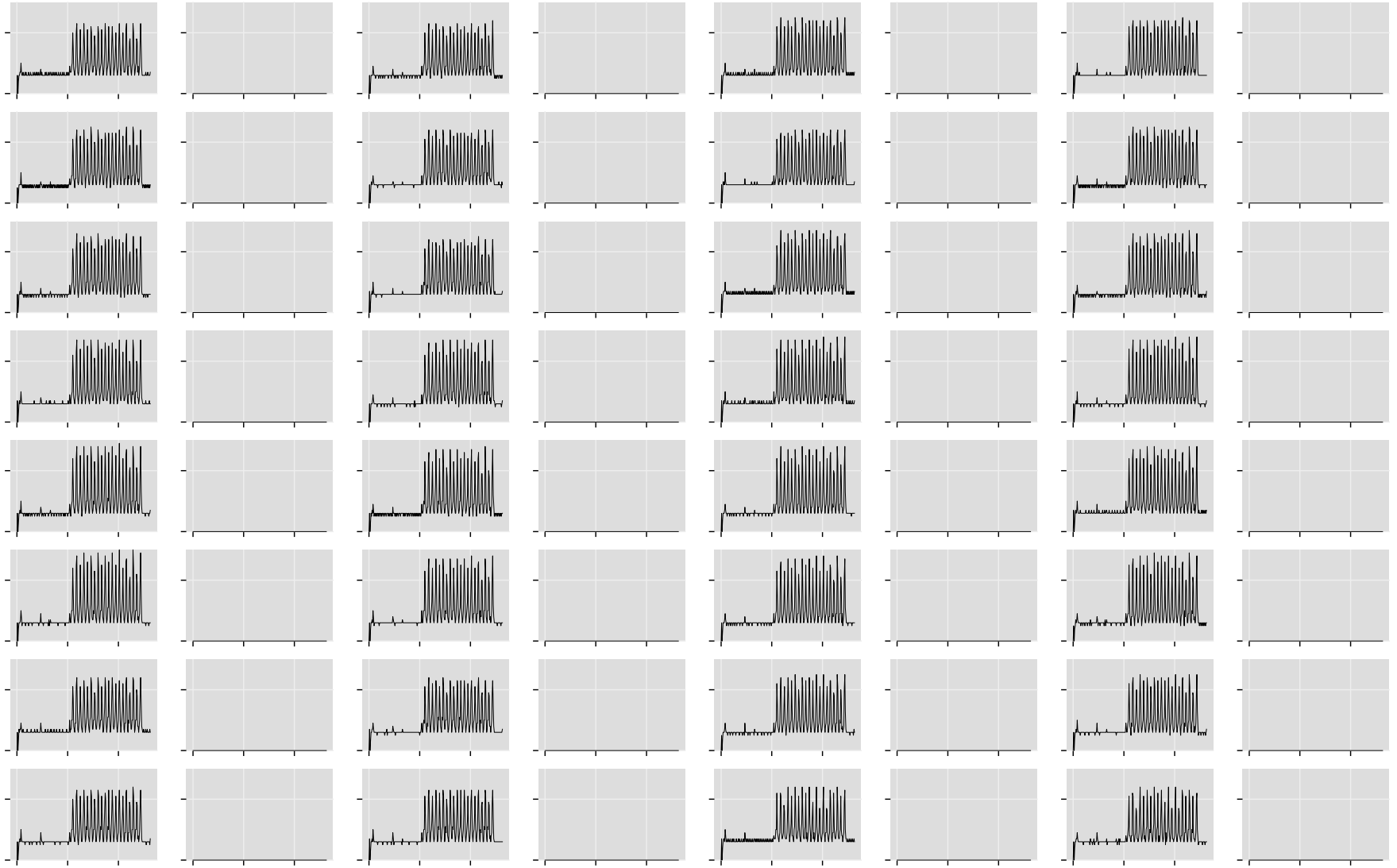
Sparklines: 64 MPI Ranks on Frontier

Temperature (C), range: [33.0, 60.0]



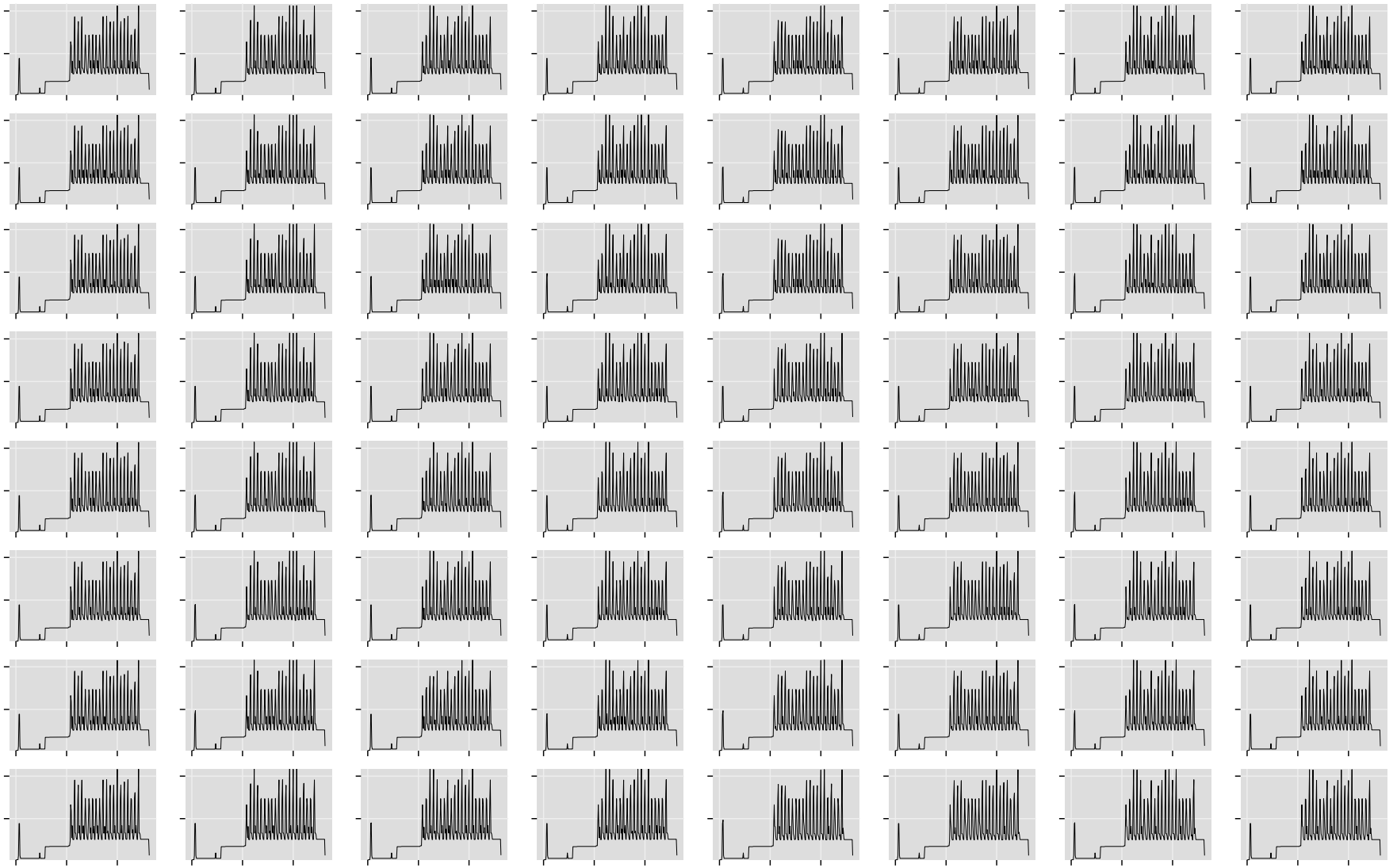
Sparklines: 64 MPI Ranks on Frontier

Energy Average (J), range: [0, 30]

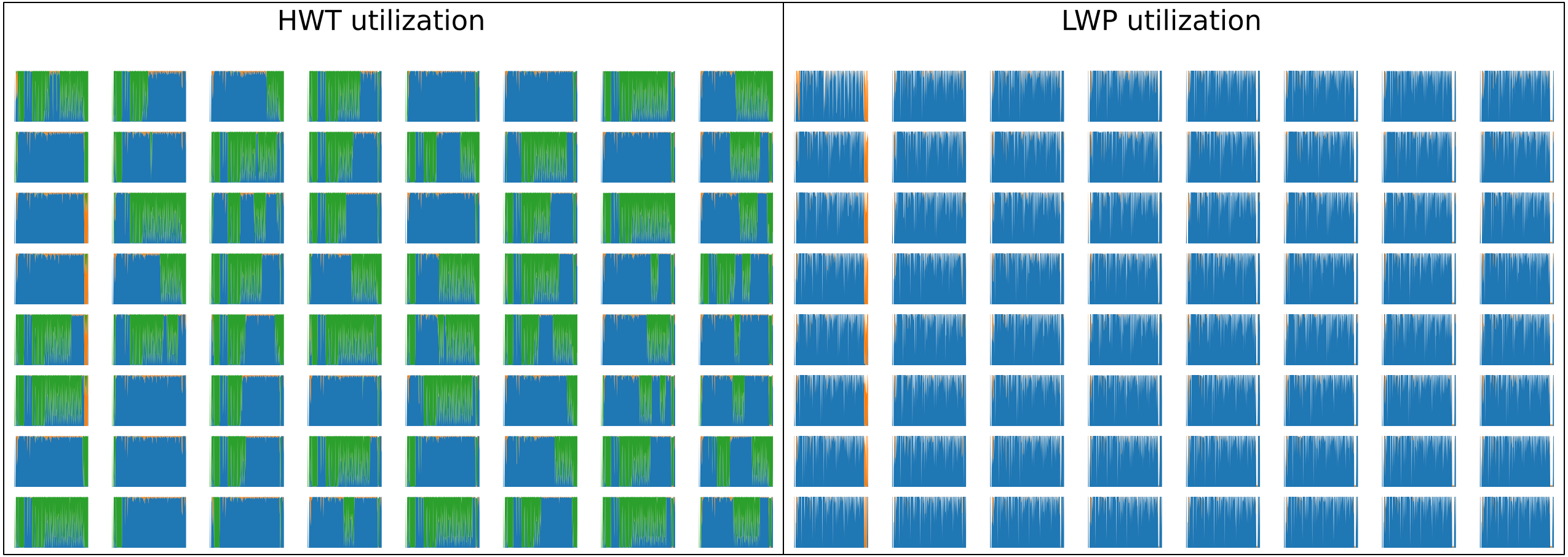


Sparklines: 64 MPI Ranks on Frontier

Used VRAM Bytes, range: [295,063,552, 21,668,823,040]



Sparklines: 64 MPI Ranks on Frontier – OMP_BIND=cores



Sparklines: 64 MPI Ranks on Frontier – OMP_BIND=threads