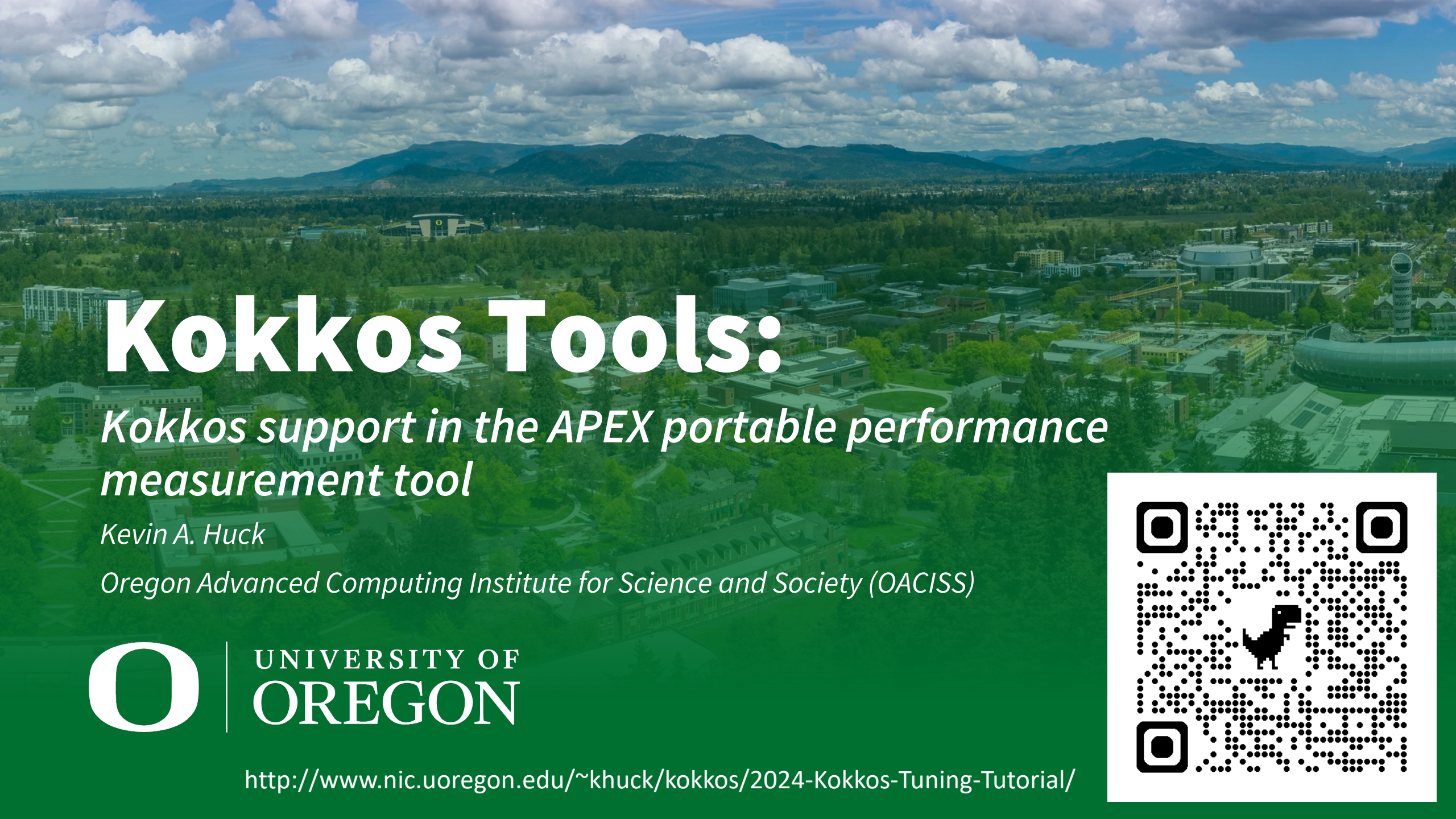


Kokkos Tools:

Kokkos support in the APEX portable performance measurement tool

Kevin A. Huck

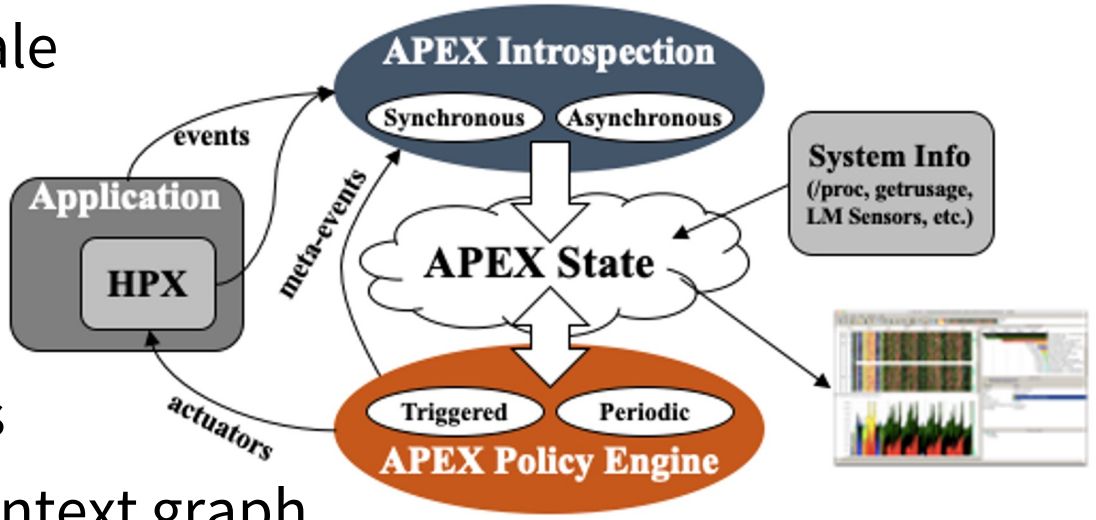
Oregon Advanced Computing Institute for Science and Society (OACISS)

UNIVERSITY OF OREGON

http://www.nic.uoregon.edu/~khuck/kokkos/2024-Kokkos-Tuning-Tutorial/

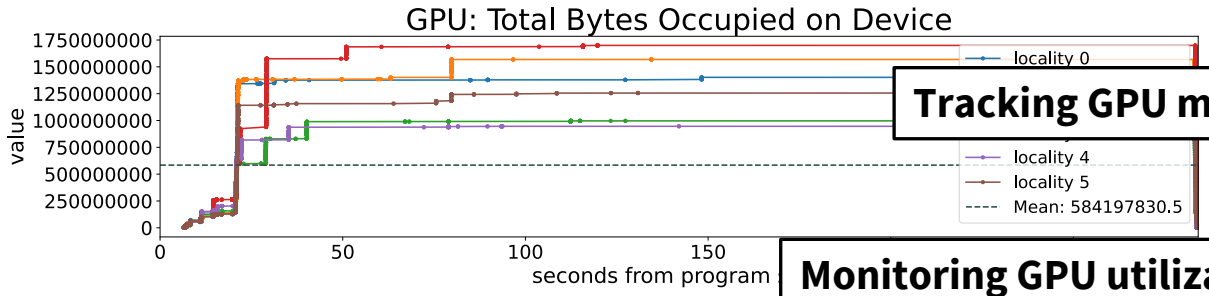
Autonomic Performance Environment for Exascale (APEX)

- 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 for parametric search methods

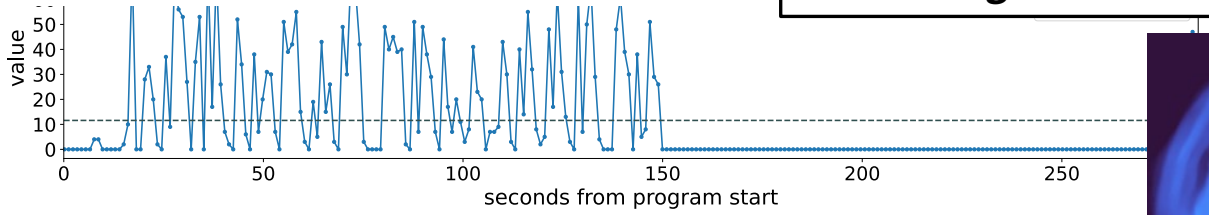


https://doi.org/10.1109/ESPM256814.2022.00008 : "Broad Performance Measurement Support for Asynchronous Multi-Tasking with APEX", Huck, ESPM, 2022

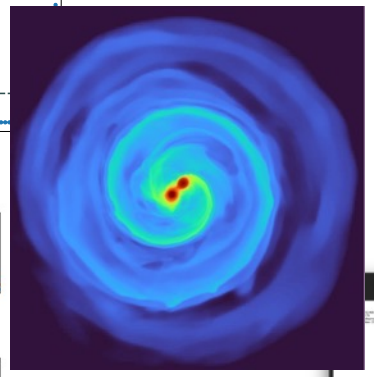
APEX example – Octo-Tiger (Octree astrophysics in HPX, Kokkos)

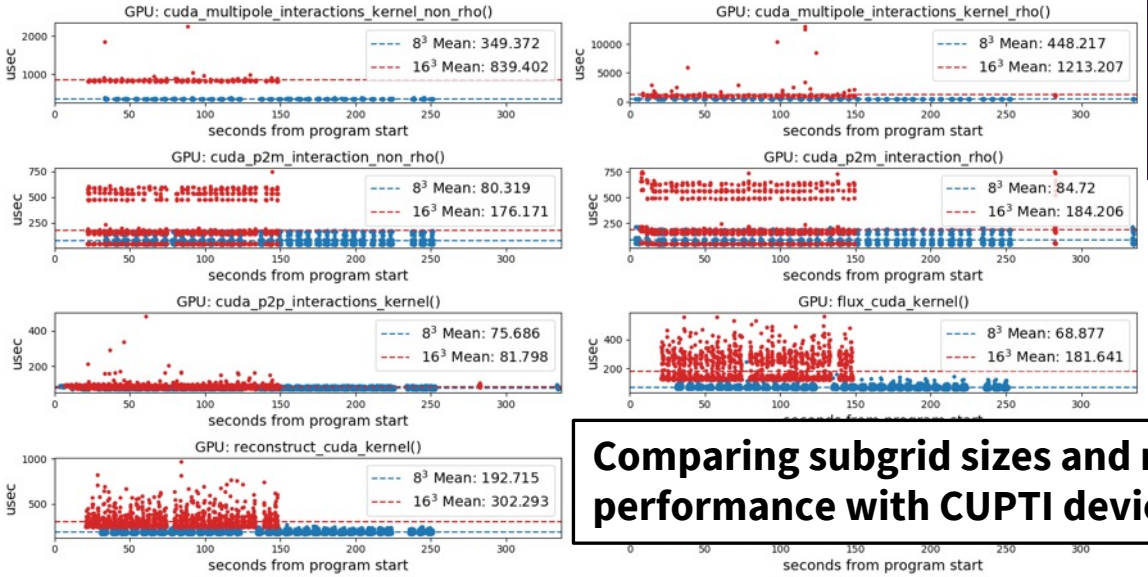


Tracking GPU memory usage with CUPTI

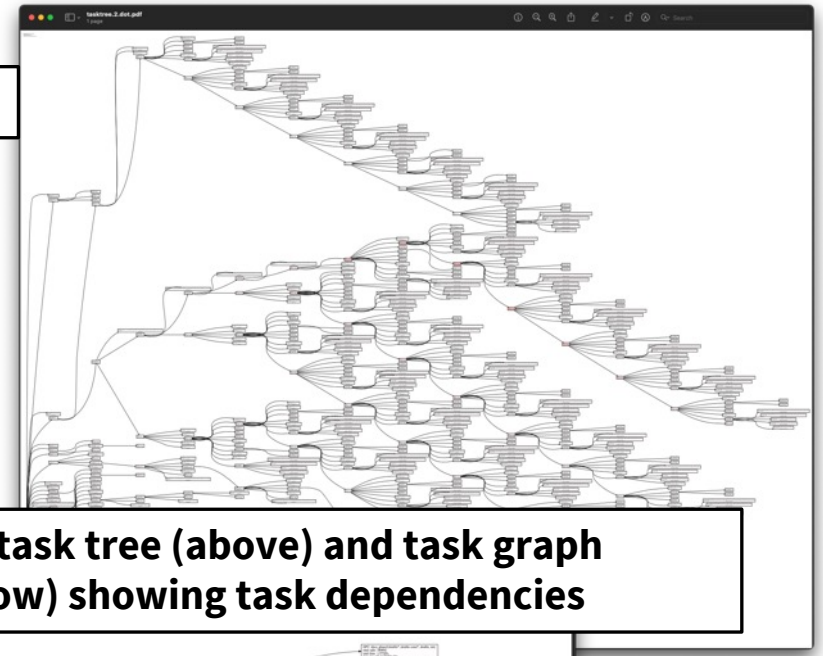


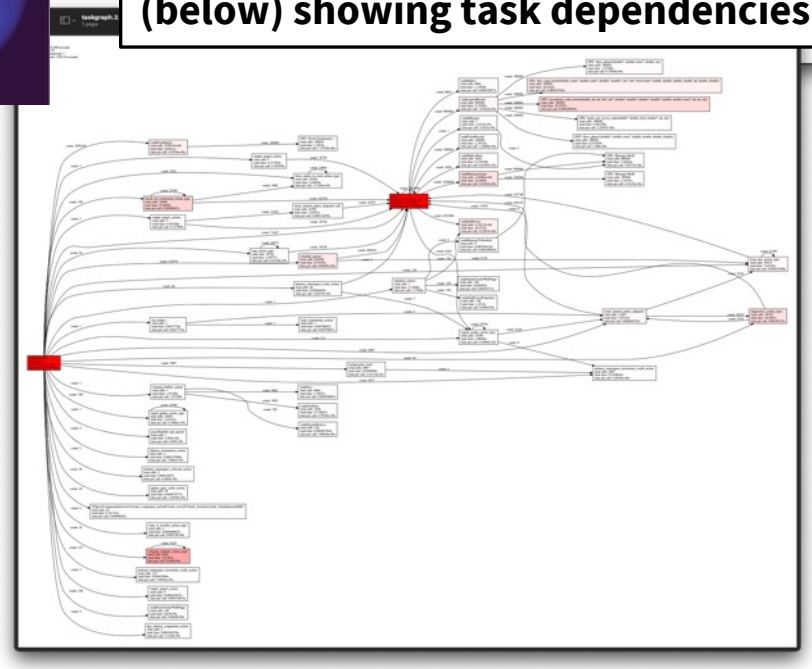
Monitoring GPU utilization with NVML library





Comparing subgrid sizes and relative kernel performance with CUPTI device activity



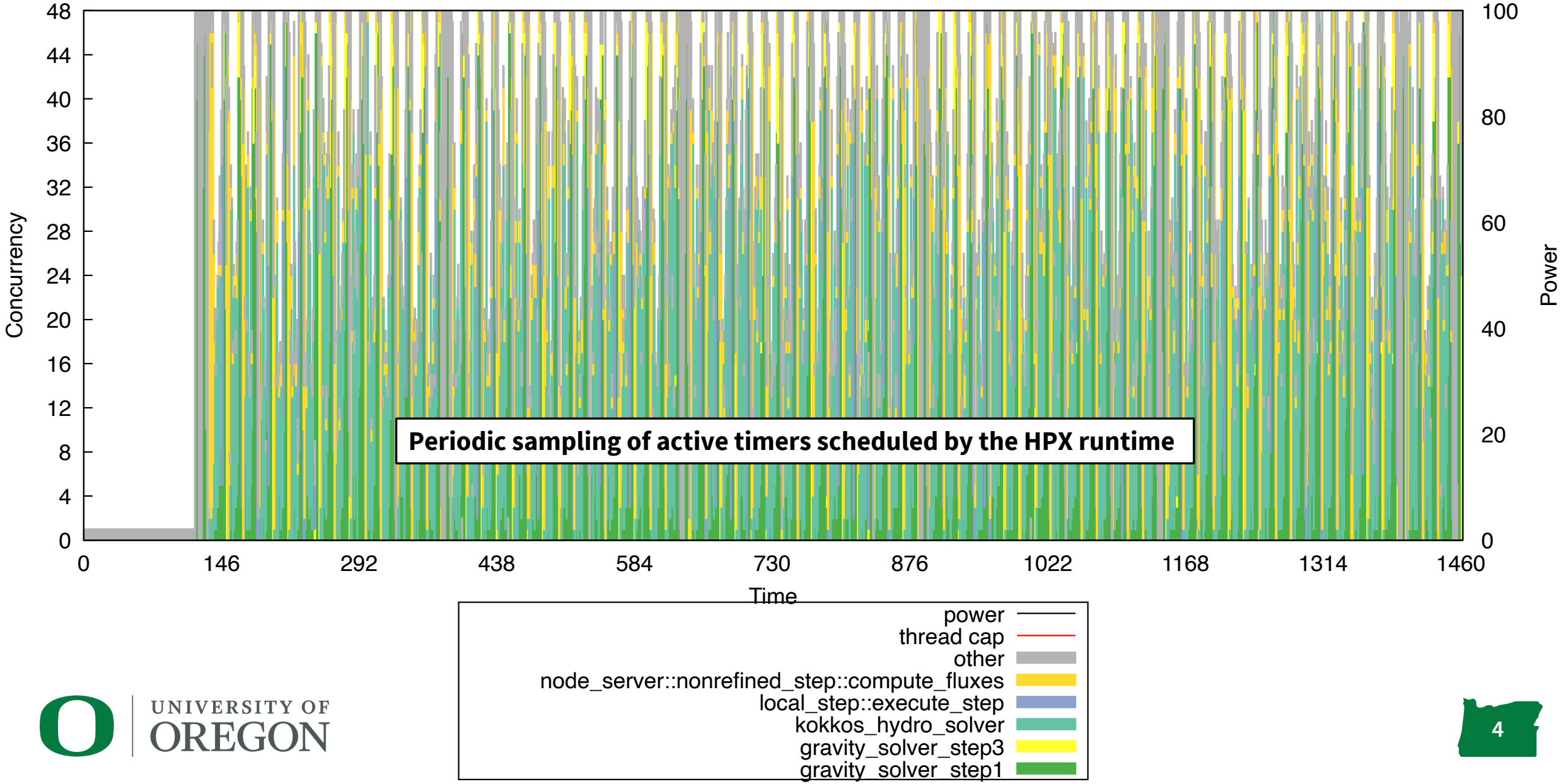


Full task tree (above) and task graph (below) showing task dependencies

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

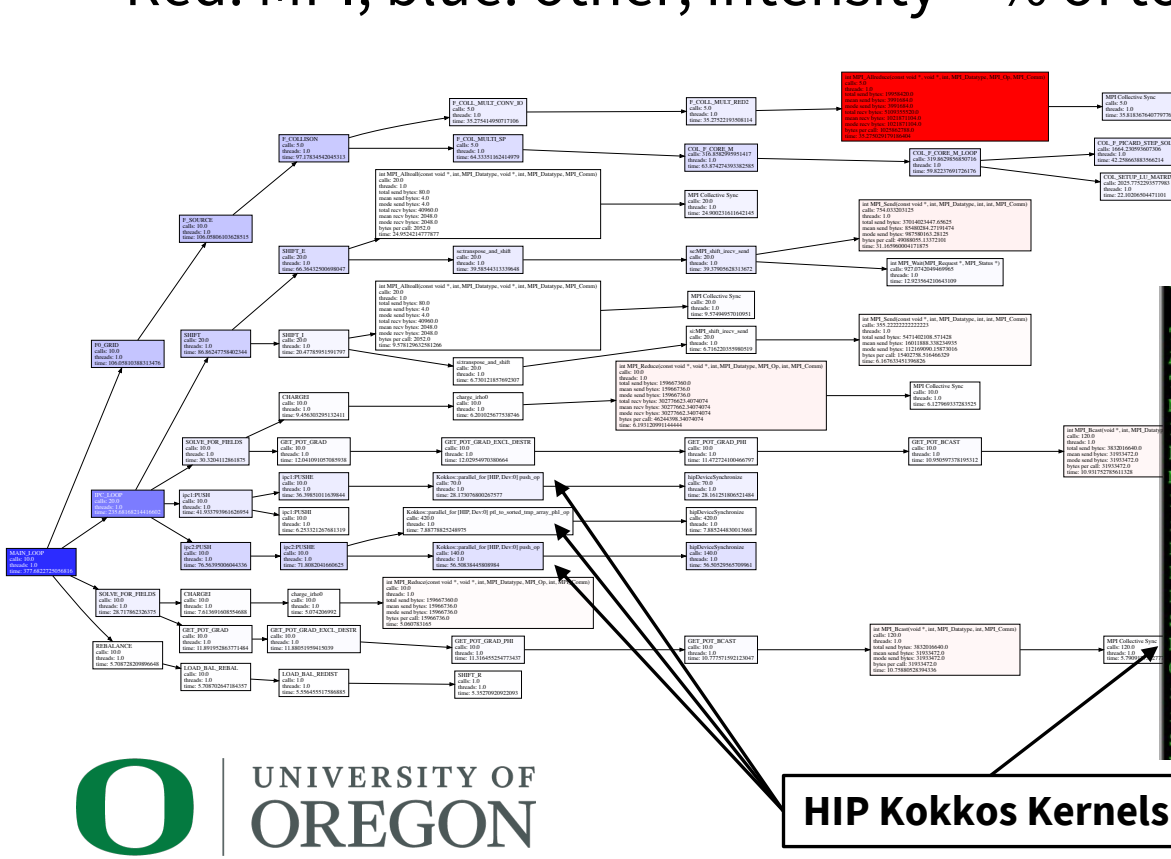
UNIVERSITY OF OREGON

Octotiger on Fugaku – recent result



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



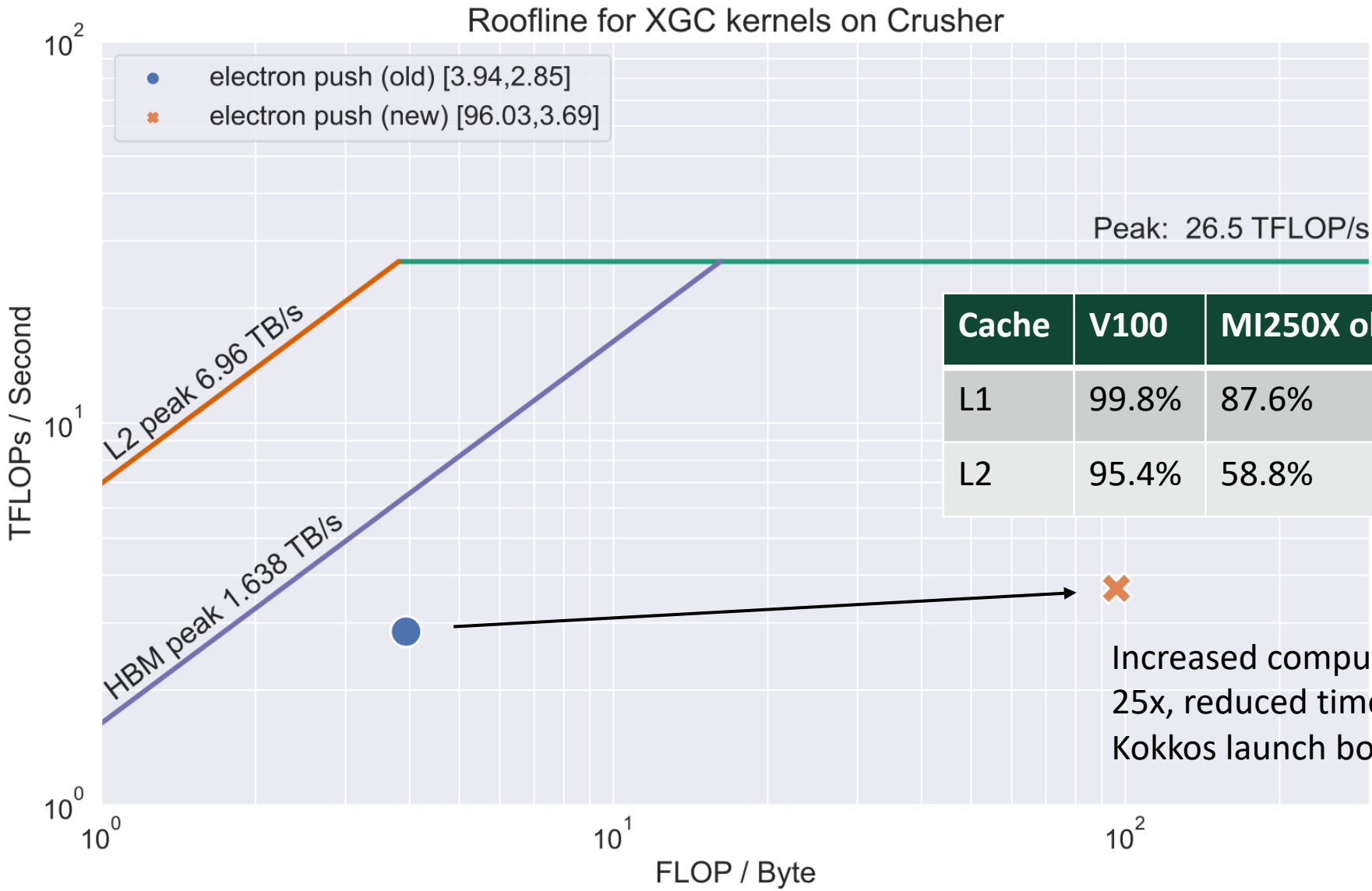
OpenMP Kokkos Kernels

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

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

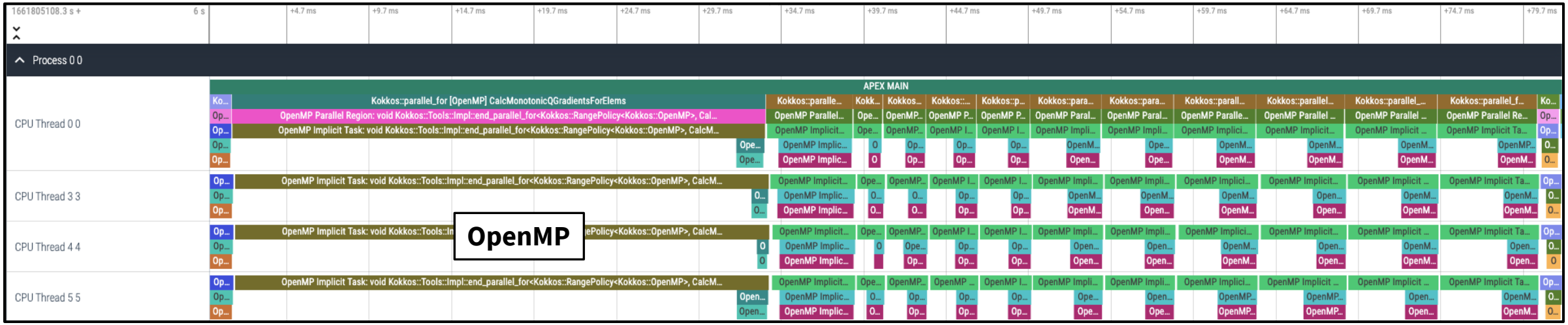
Kokkos + APEX + PAPI + rocprofiler = ☺

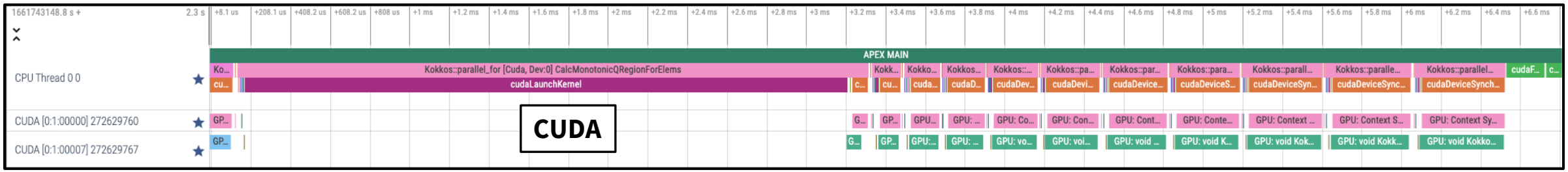


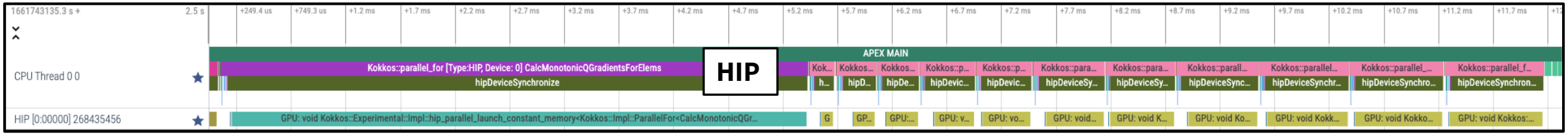
Cache	V100	MI250X old	MI250X new
L1	99.8%	87.6%	98.7%
L2	95.4%	58.8%	94.1%

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

Kokkos Lulesh and APEX Tracing – OpenMP, CUDA, HIP back ends







Kokkos Support in APEX

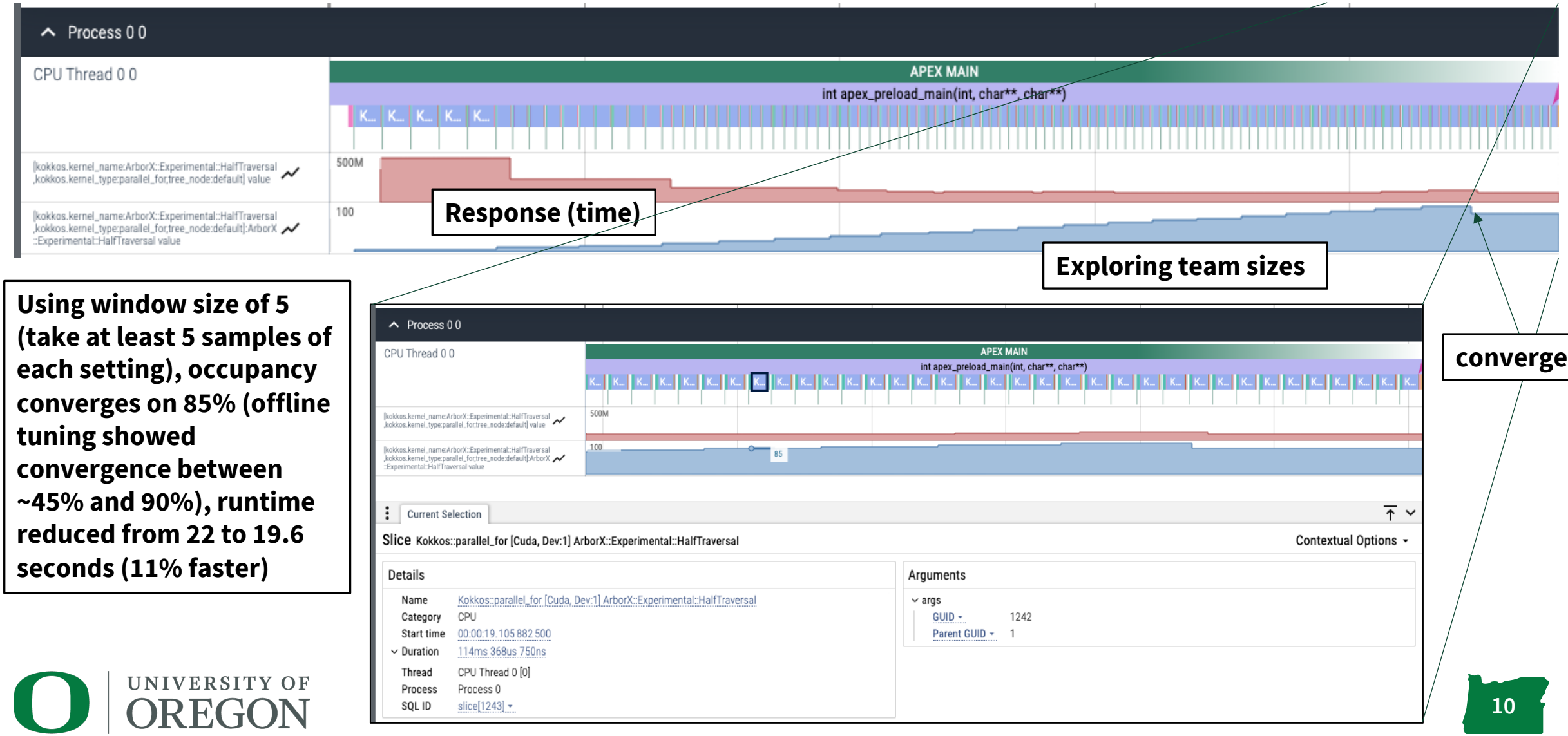
- APEX implements the same profiling API that TAU does, *and…*
- APEX provides runtime auto-tuning (parametric search) support
- Kokkos provides the ability to autotune with:

  `-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
  - RangePolicy*: block size
  - All policies*: occupancy

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

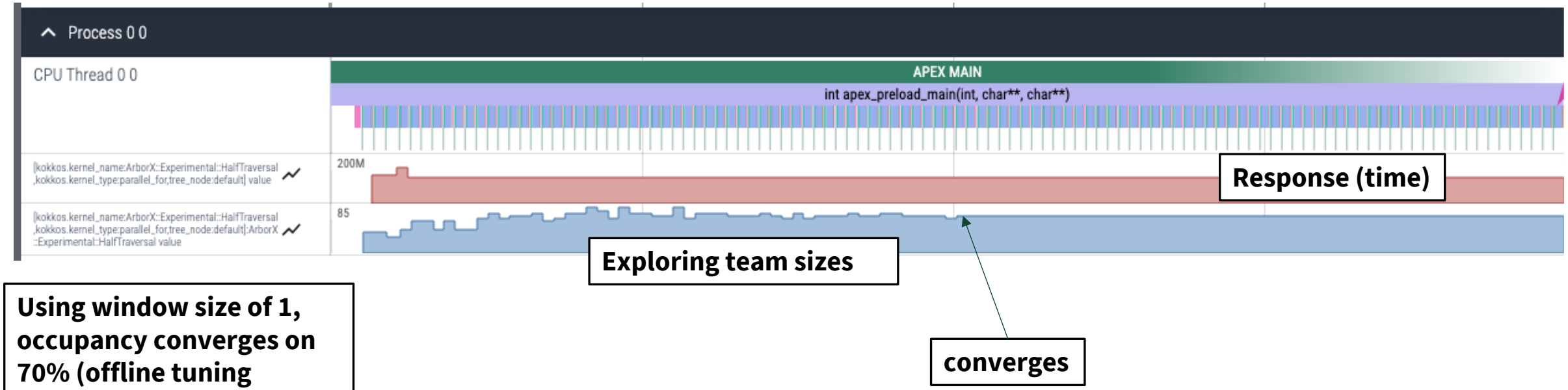
APEX Auto-tuning of ArborX DBScan with exhaustive search



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)

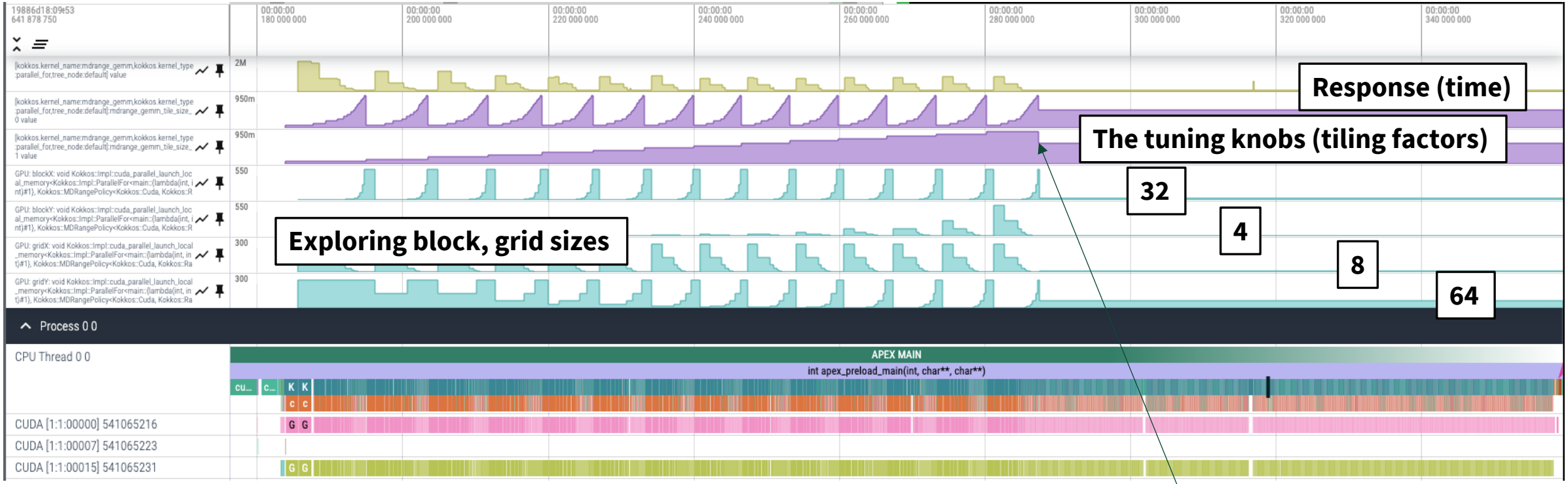
Auto-Tuning test cases / examples

- 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 – ugly, 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 each for best configuration.
- Configure & Build:
  - See README.md for latest info (https://github.com/khuck/apex-kokkos-tuning/)

Building apex-kokkos-tuning

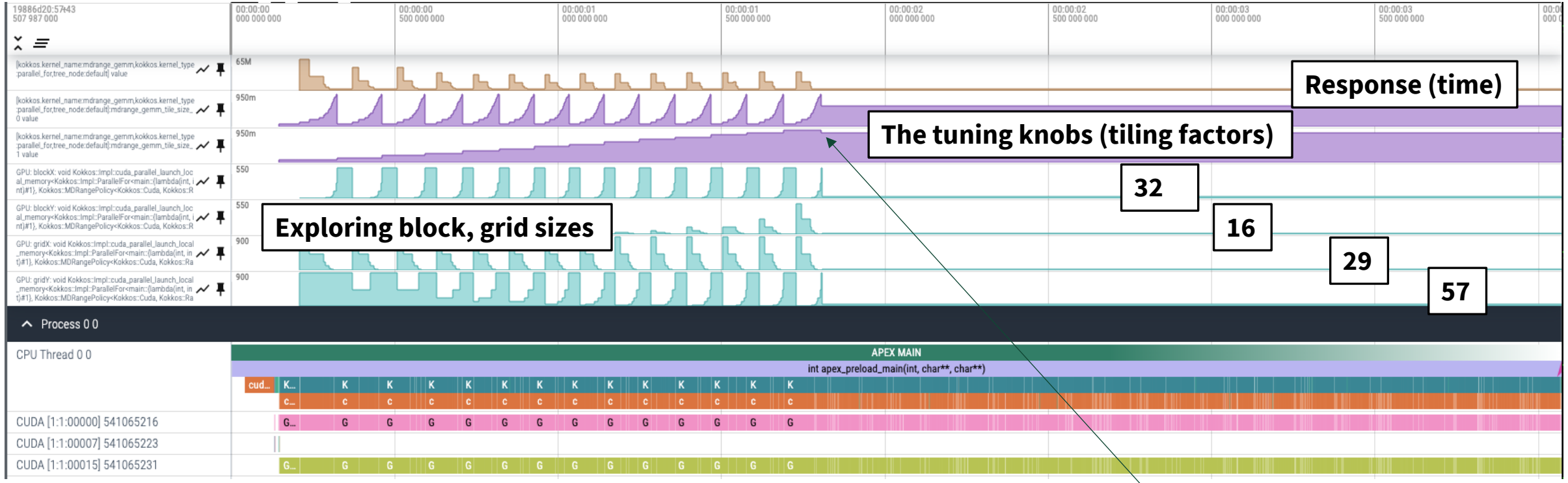
- `generic-cuda.sh` or `perlmutter.sh` for CUDA examples
- `frontier.sh` for ROCm example
- General idea: this is just a CMake compound project including the Kokkos and APEX projects. Any CMake variables you would pass in to those projects you pass in to this one. With specific notes:
  - `-DKokkos_ENABLE_TUNING=ON` is required for tuning
  - This repo uses a submodule that is a fork/branch of the main Kokkos repository, waiting on a PR to be merged

Example: testing mdrange_gemm 256x256 with exhaustive search



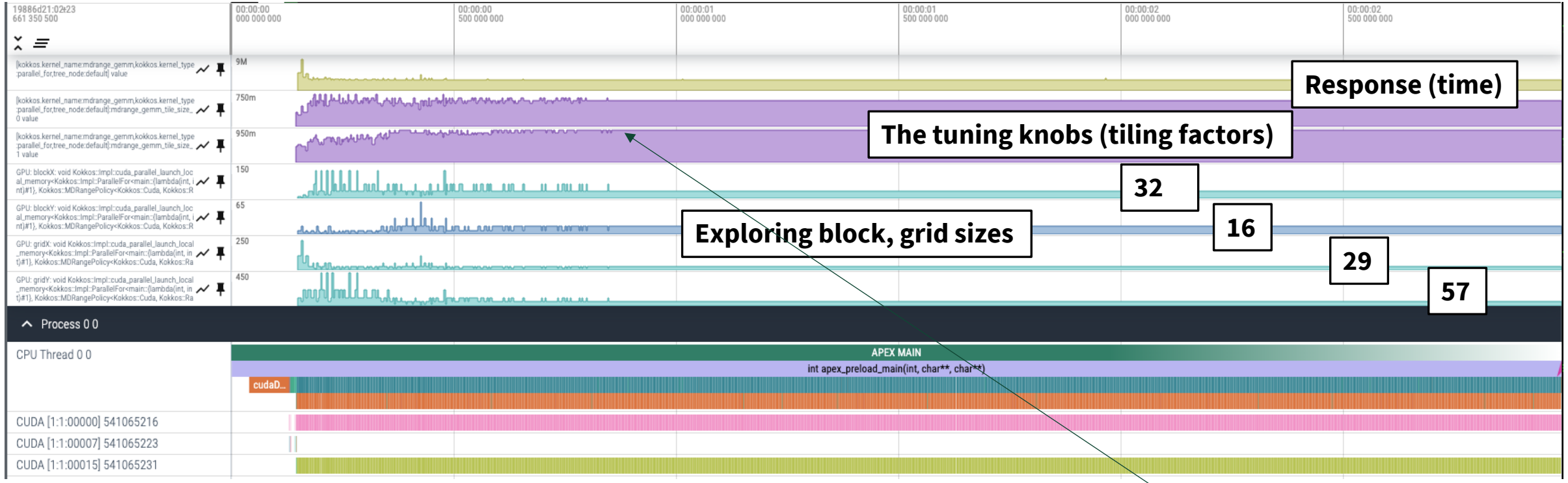
Using window size of 2, tiling converges on block dimensions of (32,4,1) and grid dimensions (8,64,1), the defaults are (16,2,1) and (16,128,1) respectively – simulated annealing converges to the same values

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