Performance Analysis of GPU-accelerated Applications with HPCToolkit

John Mellor-Crummey
Rice University

ALCF Hands-on HPC Workshop

Slides available in ALCF-Workshops Slack

STEP SOFTWARE TOOLS ECOSYSTEM PROJECT
National Nuclear Security Administration
U.S. DEPARTMENT OF ENERGY | Office of Science
RICE

Plan for This Session

- Prepare for hands-on
- Introduce HPCToolkit tools and workflow
  - Measurement (hpcrun)
  - Post-mortem analysis tools (hpcstruct, hpcprof)
  - Graphical user interface (hpcviewer)
- Illustrate hpctoolkit's use with some case studies
- Working with hands-on examples
  - Exploring pre-collected performance databases
  - Full contact

    scripted measurement, analysis, visualization of examples

Prepare to Explore Performance Data on your Laptop

- Download and install hpcviewer
  - See https://hpctoolkit.org/download.html
    - —Select the right one for your laptop: MacOS (Apple Silicon, Intel), Windows, Linux
    - —If you don't have Java JDK 17 or 21, you can install one easily from Adoptium (https://adoptium.net)
      - —It's a one-click install. Install Java JDK before running hpcviewer
- User manual for hpcviewer

  https://hpctoolkit.gitlab.io/hpctoolkit/users/hpcviewer/hpcviewer.html

Acquire a Copy of HPCToolkit Hands-on Examples for Aurora

Starting from scratch

```
git clone https://github.com/argonne-lcf/ALCF_Hands_on_HPC_Workshop.git
```

Updating your existing directory

```
cd ALCF_Hands_on_HPC_Workshop
git pull
```

Today's examples

```
cd ALCF_Hands_on_HPC_Workshop/tools/hpctoolkit
```

Acquire a Compute Node to use Interatively

Getting a compute node

```
cd ALCF_Hands_on_HPC_Workshop/tools/hpctoolkit
source sourceme-get-compute-node.sh
```

Configuring a compute node's environment for hands-on examples

```
cd ALCF_Hands_on_HPC_Workshop/tools/hpctoolkit
source sourceme-on-compute-node.sh
```

Plan for This Session

- Prepare for hands-on
- Introduce HPCToolkit tools and workflow
  - Measurement (hpcrun)
  - Post-mortem analysis tools (hpcstruct, hpcprof)
  - Graphical user interface (hpcviewer)
- Illustrate hpctoolkit's use with some case studies
- Working with hands-on examples
  - Exploring pre-collected performance databases
  - Full contact

    scripted measurement, analysis, visualization of examples

Linux Foundation's HPCToolkit Performance Tools

- Collect profiles and traces of unmodified parallel CPU and GPU-accelerated applications
- Understand where an application spends its time and why
  - —call path profiles associate metrics with application source code contexts
    - analyze instruction-level performance within GPU kernels and attribute it to your source code
  - —hierarchical traces to understand execution dynamics
- Parallel programming models
  - —across nodes: MPI, SHMEM, UPC++, …
  - —within nodes: OpenMP, Kokkos, RAJA, HIP, DPC++, Sycl, CUDA, OpenACC, …
- Languages
  - —C, C++, Fortran, Python, …
- Hardware
  - —CPU cores and GPUs within a node
    - —CPU: x86_64, Power, ARM
    - —GPU: NVIDIA, AMD, Intel

Why HPCToolkit?

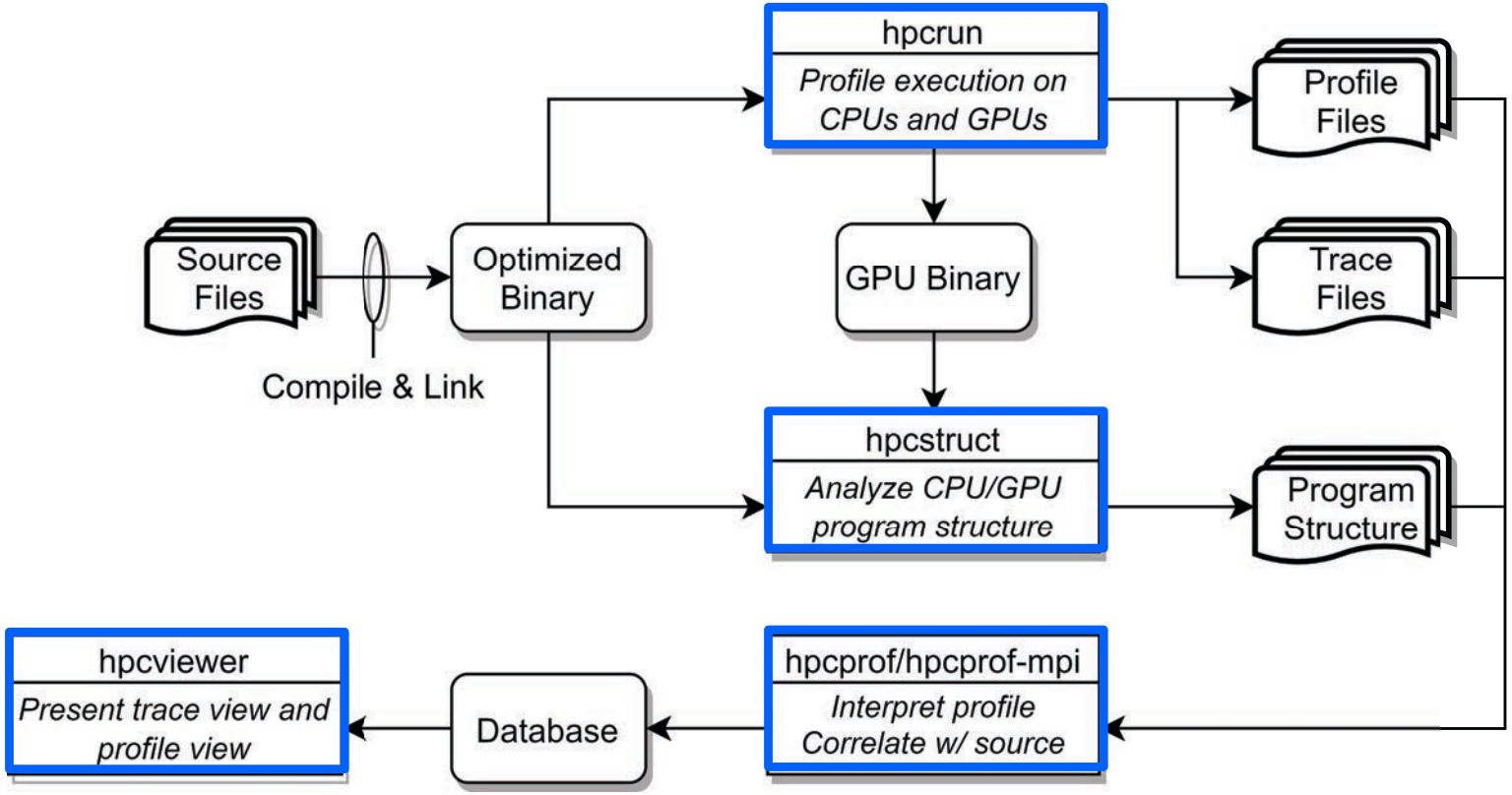
- Measure and analyze performance of CPU and GPU-accelerated applications
- Easy: profile unmodified application binaries
- Fast: low-overhead measurement
- Informative: understand where an application spends its time and why
  - —call path profiles associate metrics with application source code contexts
  - —optional hierarchical traces to understand execution dynamics
- Broad audience
  - —application developers
  - —framework developers
  - —runtime and tool developers

- Unlike vendor tools, it works with a wide range of CPUs and GPUs

How does HPCToolkit Differ from Vendor Tools?

- NVIDIA NSight Systems
  - —tracing of CPU and GPU streams
  - —analyze traces when you open them with the GUI
    - long running traces are huge and thus extremely slow to analyze, limiting scalability
  - —designed for measurement and analysis within a node
- NVIDIA NSight Compute
  - —detailed measurement of kernels with counters and execution replay
  - —very slow measurement
  - —flat display of measurements within GPU kernels
- Intel VTune: designed for analysis of performance on a single node
- AMD Omnitrace: designed for analysis of performance on a single node
- HPCToolkit
  - —more scalable tracing than vendor tools
    - measure exascale executions across many nodes and GPUs
    - GUI can render trace data measured in TB
  - —scalable, parallel post-mortem analysis vs. non-scalable in-GUI analysis
  - —detailed reconstruction of estimates for calling context profiles within GPU kernels

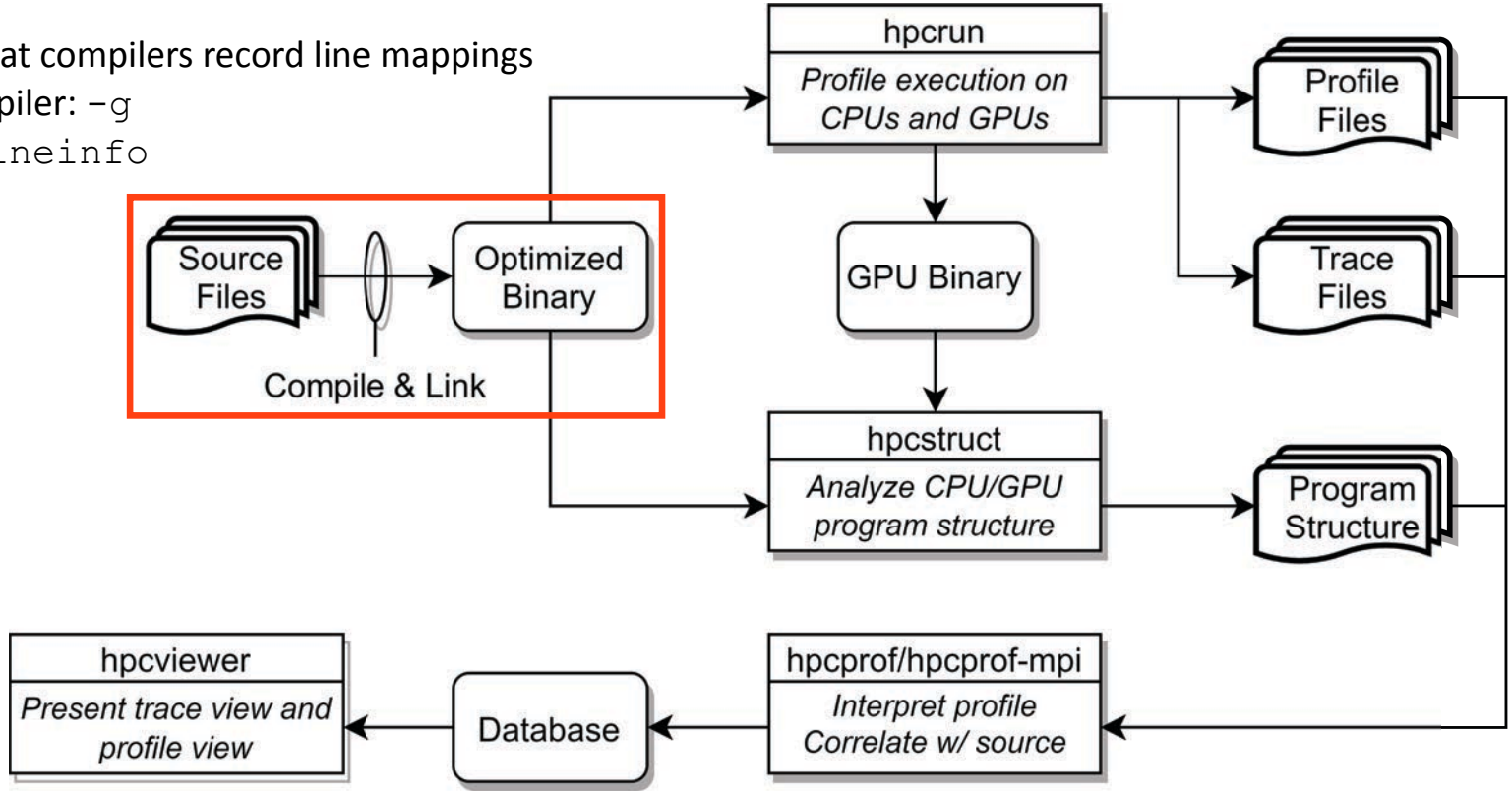
HPCToolkit's Workflow for GPU-accelerated Applications



HPCToolkit's Workflow for GPU-accelerated Applications

Step 1:

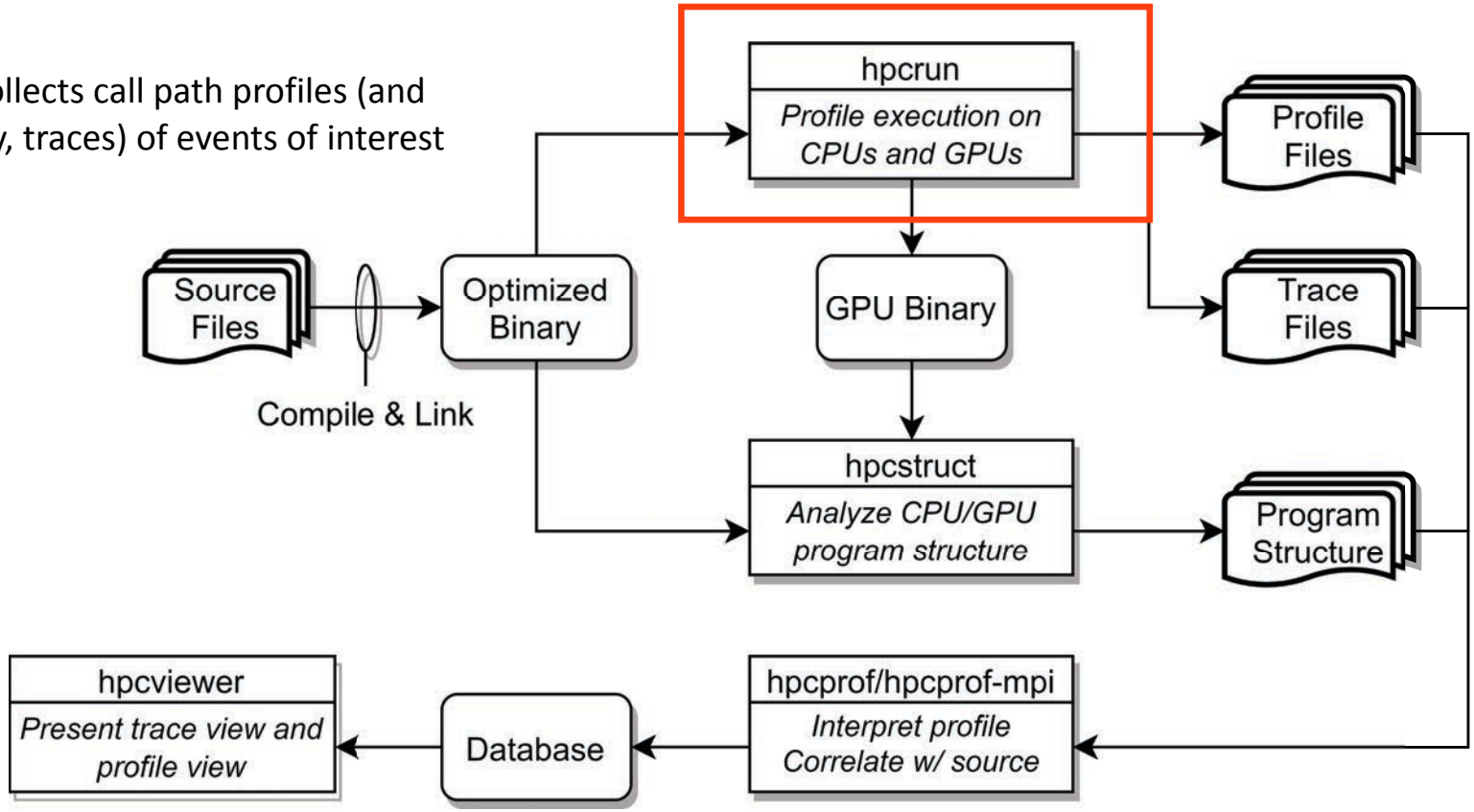
- Ensure that compilers record line mappings
- host compiler: `-g`
- nvcc: `-lineinfo`



HPCToolkit's Workflow for GPU-accelerated Applications

Step 2:

- *hpcrun* collects call path profiles (and optionally, traces) of events of interest



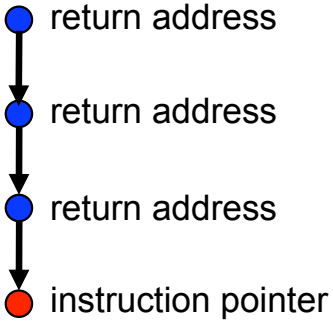
Measurement of CPU and GPU-accelerated Applications

- Sampling using Linux timers and hardware counter overflows on the CPU
- Callbacks when GPU operations are launched
- Event stream or callbacks for GPU operation completion
- PC Samples: AMD, NVIDIA, Intel
- Binary instrumentation of GPU kernels on Intel GPUs for fine-grain measurement

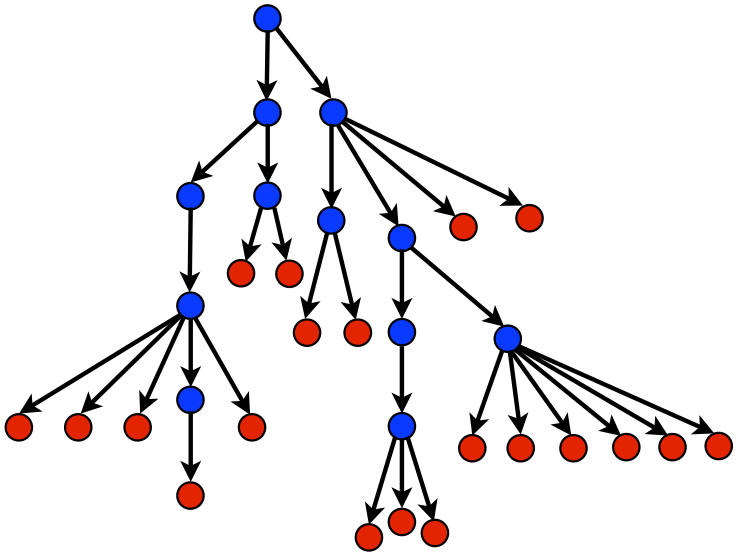
Call Stack Unwinding to Attribute Costs in Context

- Unwind when timer or hardware counter overflows
  - —measurement overhead proportional to sampling frequency rather than call frequency
- Unwind to capture context for events such as GPU kernel launches

Call path sample



Calling context tree



hpcrun: Measure CPU and/or GPU activity

- GPU profiling
  - `hpcrun -e gpu=xxx <app> ….`
- GPU PC sampling
  - `hpcrun -e gpu=yyy,pc <app>`
- CPU and GPU Tracing (in addition to profiling)
  - `hpcrun -e CPUTIME -e gpu=xxx -tt <app>`
- Use hpcrun with MPI on Polaris or Aurora
  - `mpiexec -n <ranks> … hpcrun -e gpu=xxx <app>`

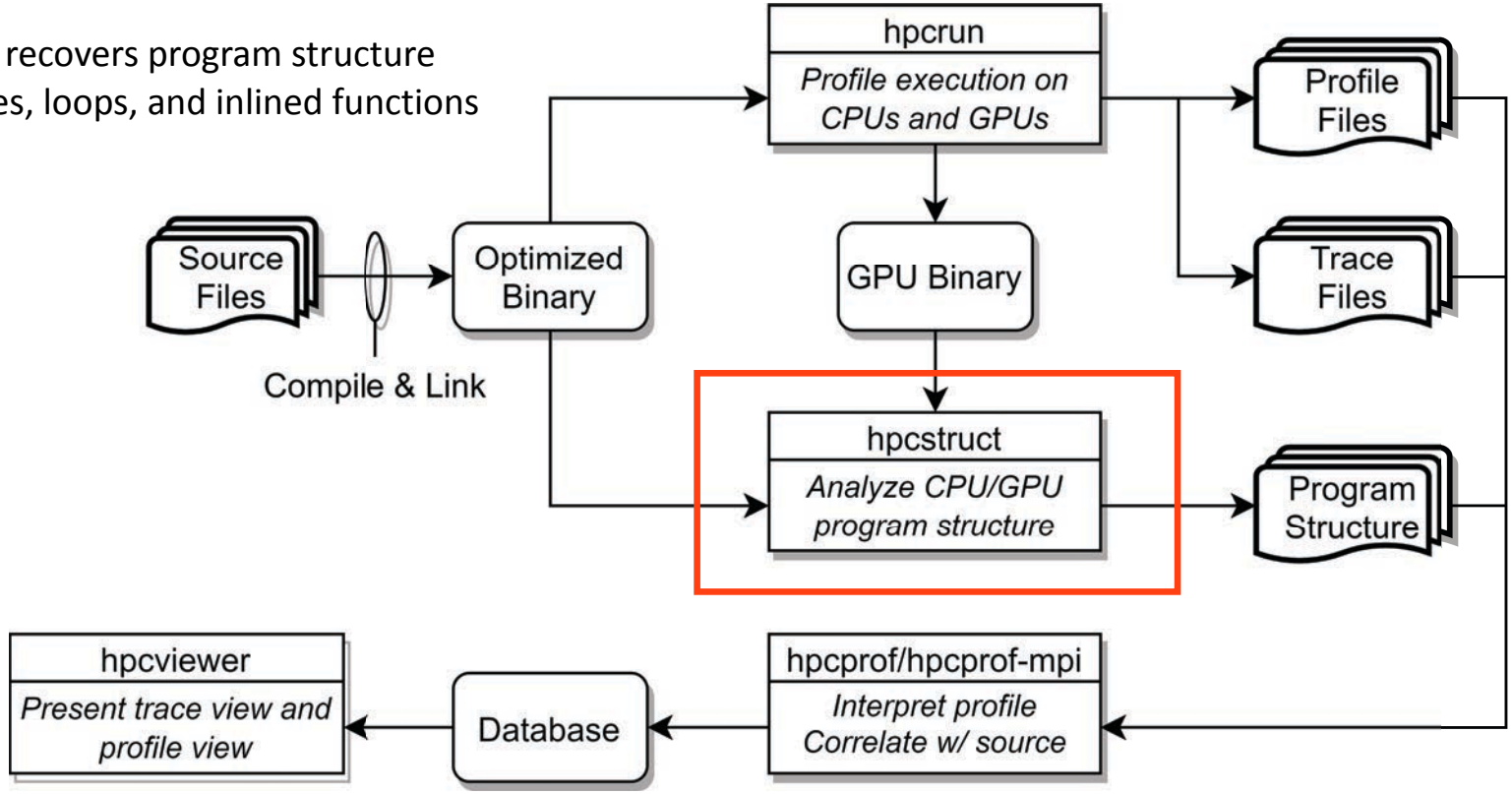
xxx ∈ *{cuda,rocm,opencl,level0}*
yyy ∈ *{cuda,rocm,level0}*

Note:
Directions here apply to HPCToolkit 2025.1.0
Prior versions used
nvidia instead of cuda
amd instead of rocm

HPCToolkit's Workflow for GPU-accelerated Applications

Step 3:

- *hpcstruct* recovers program structure about lines, loops, and inlined functions



hpcstruct: Analyze CPU and GPU Binaries Using Multiple Threads

- Usage

```
hpcstruct [--gpucfg yes] <measurement-directory>
```

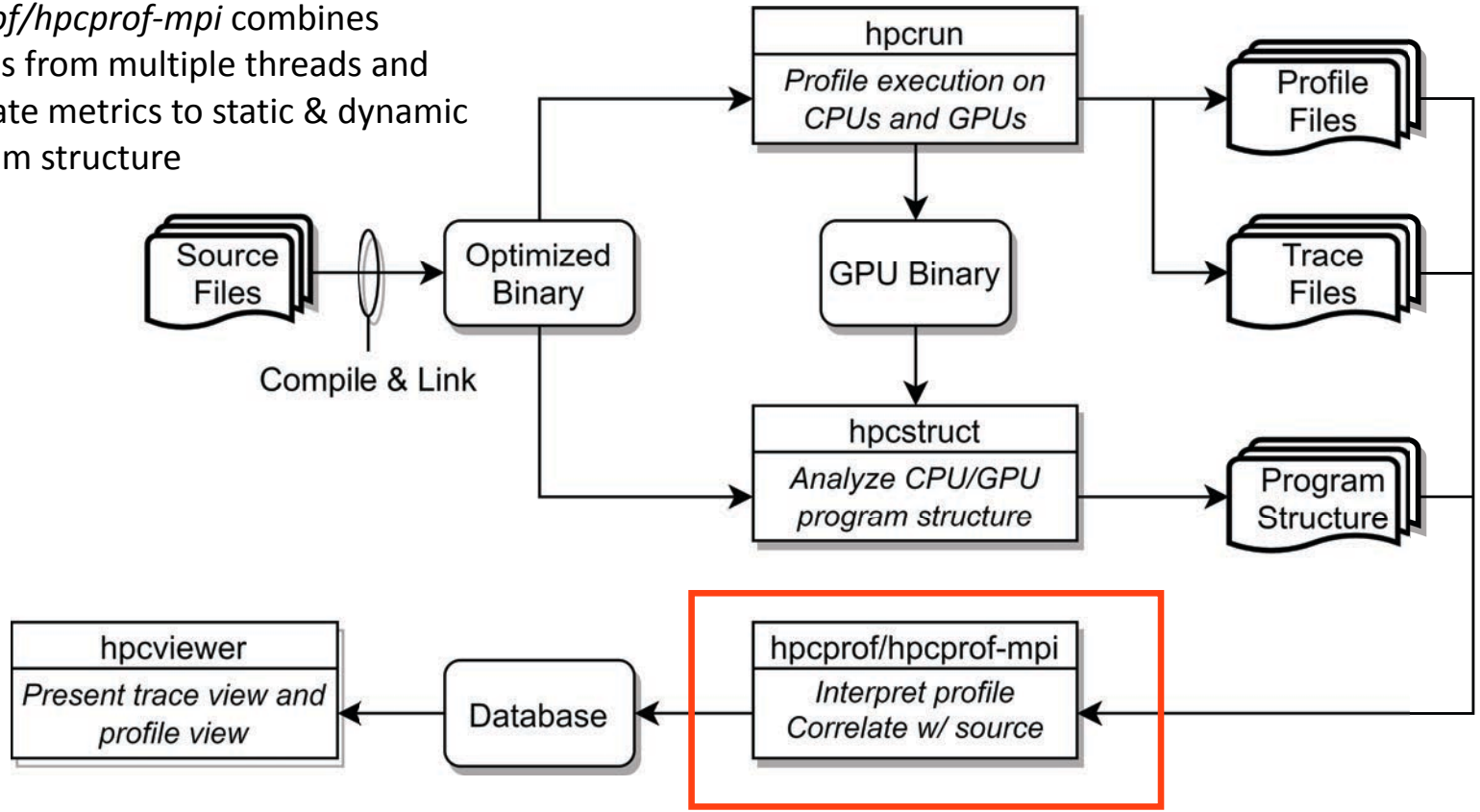
- What it does
  - Recover program structure information
    - Files, functions, inlined templates or functions, loops, source lines
  - In parallel, analyze all CPU and GPU binaries that were measured by HPCToolkit
    - —typically analyze large application binaries with 16 threads
    - —typically analyze multiple small application binaries concurrently with 2 threads each
  - Cache binary analysis results for reuse when analyzing other executions

NOTE: `--gpucfg yes` needed for analysis of PC samples on NVIDIA and AMD GPUs

HPCToolkit's Workflow for GPU-accelerated Applications

Step 4:

- *hpcprof/hpcprof-mpi* combines profiles from multiple threads and correlate metrics to static & dynamic program structure



hpcprof/hpcprof-mpi: Associate Measurements with Program Structure

- Analyze data from modest executions with multithreading (moderate scale)

```
hpcprof <measurement-directory>
```

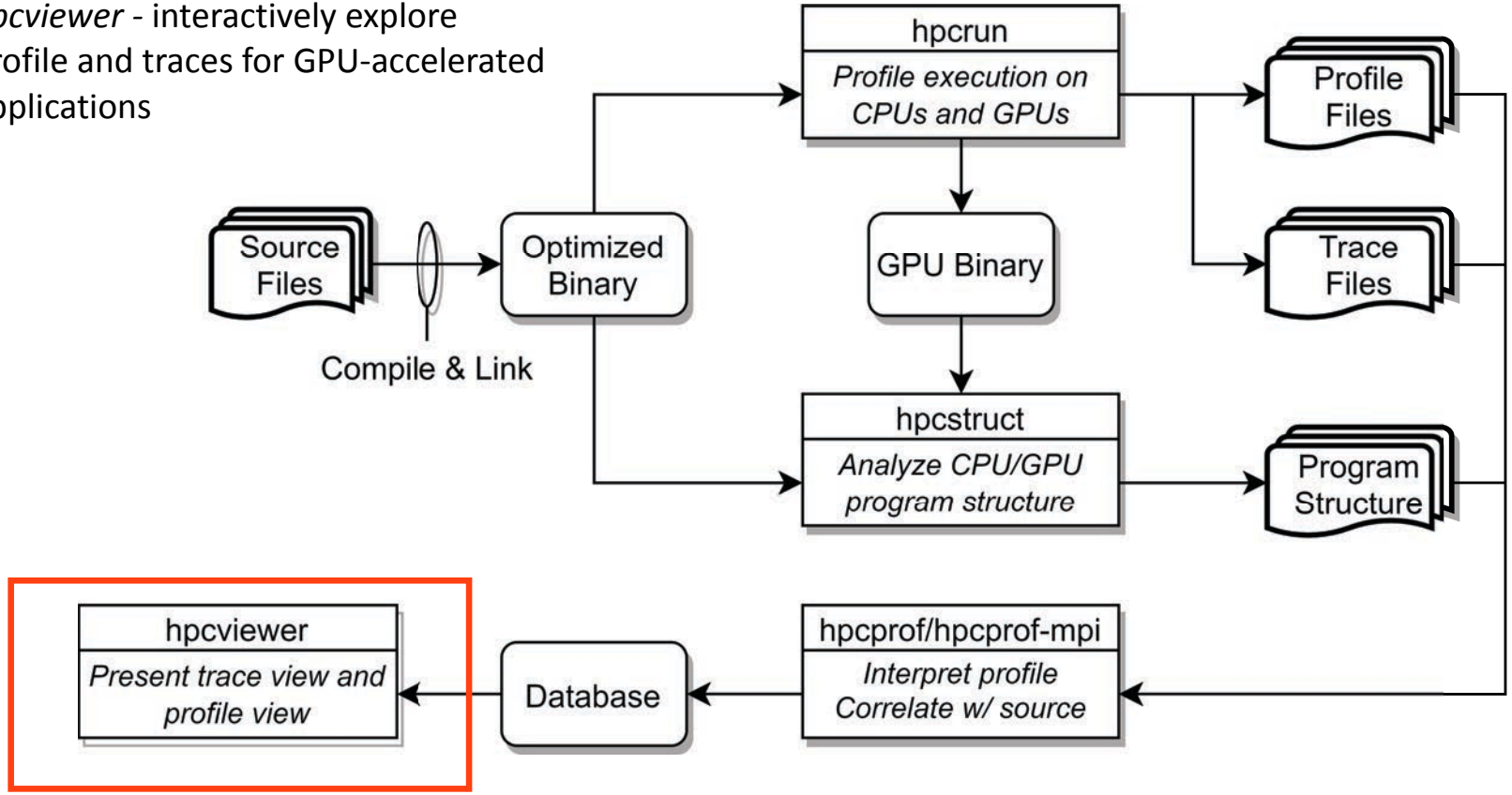
- Analyze data from large executions with distributed-memory parallelism + multithreading (large scale)

```
mpiexec -n ${NODES} --ppn 1 –depth=128 \
    hpcprof-mpi <measurement-directory>
```

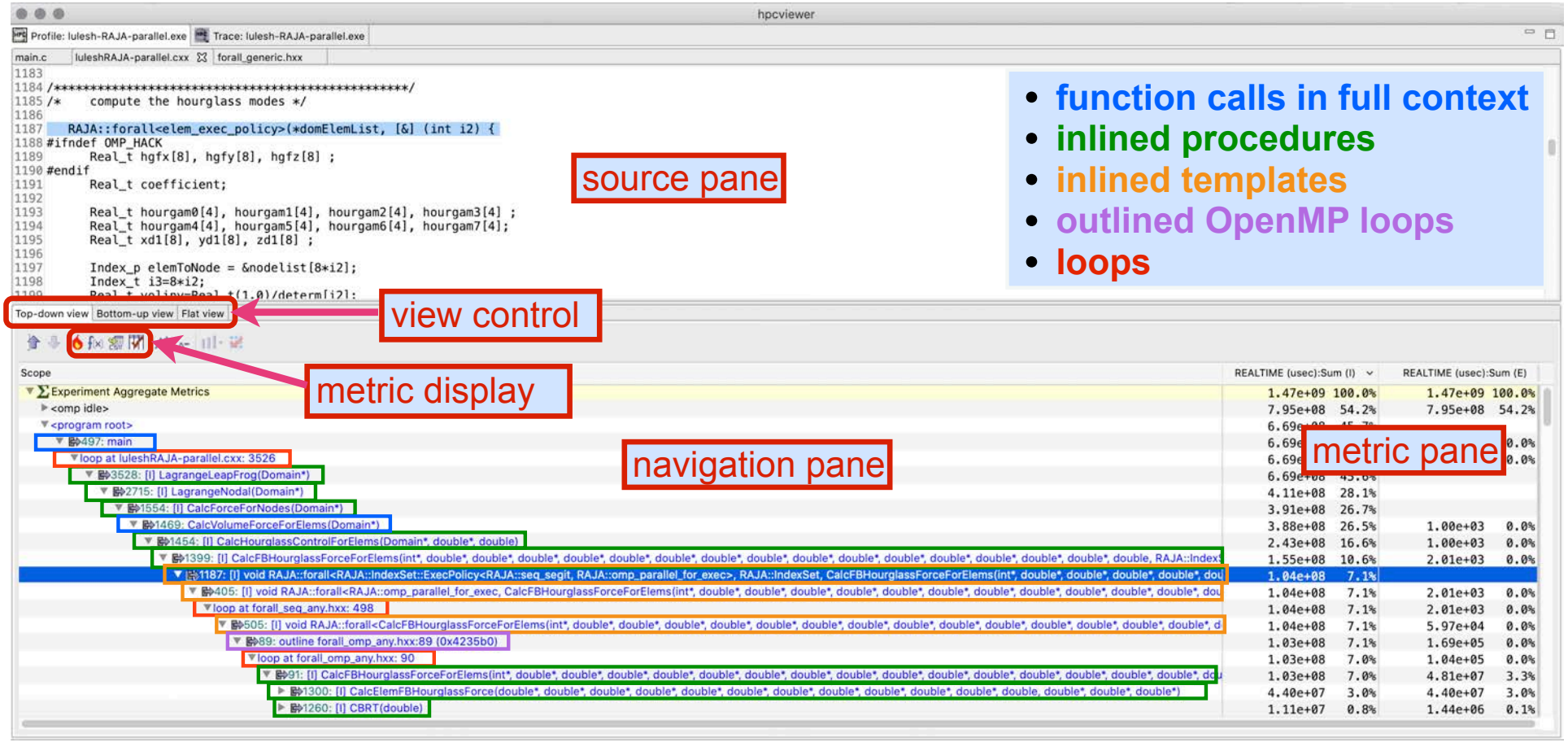
HPCToolkit's Workflow for GPU-accelerated Applications

Step 4:

- *hpcviewer* - interactively explore profile and traces for GPU-accelerated applications

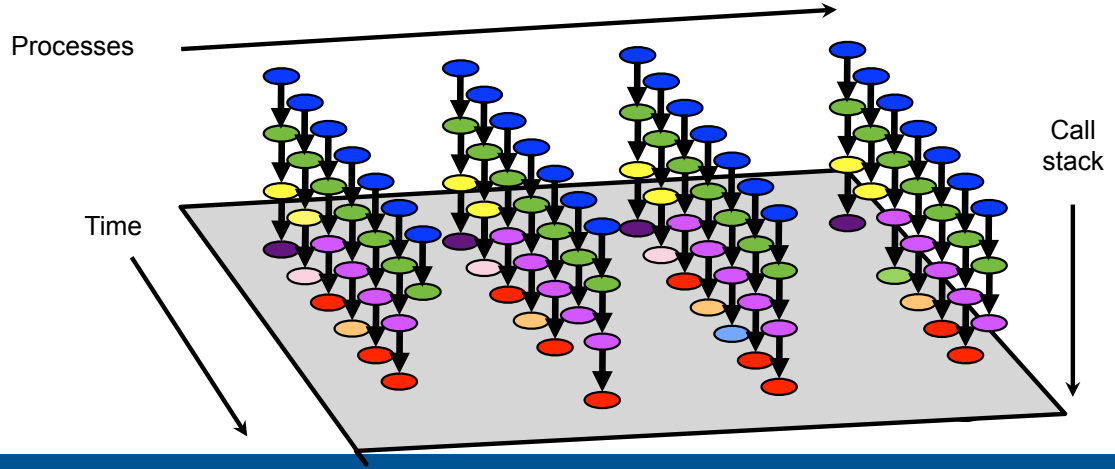


Code-centric Analysis with hpcviewer

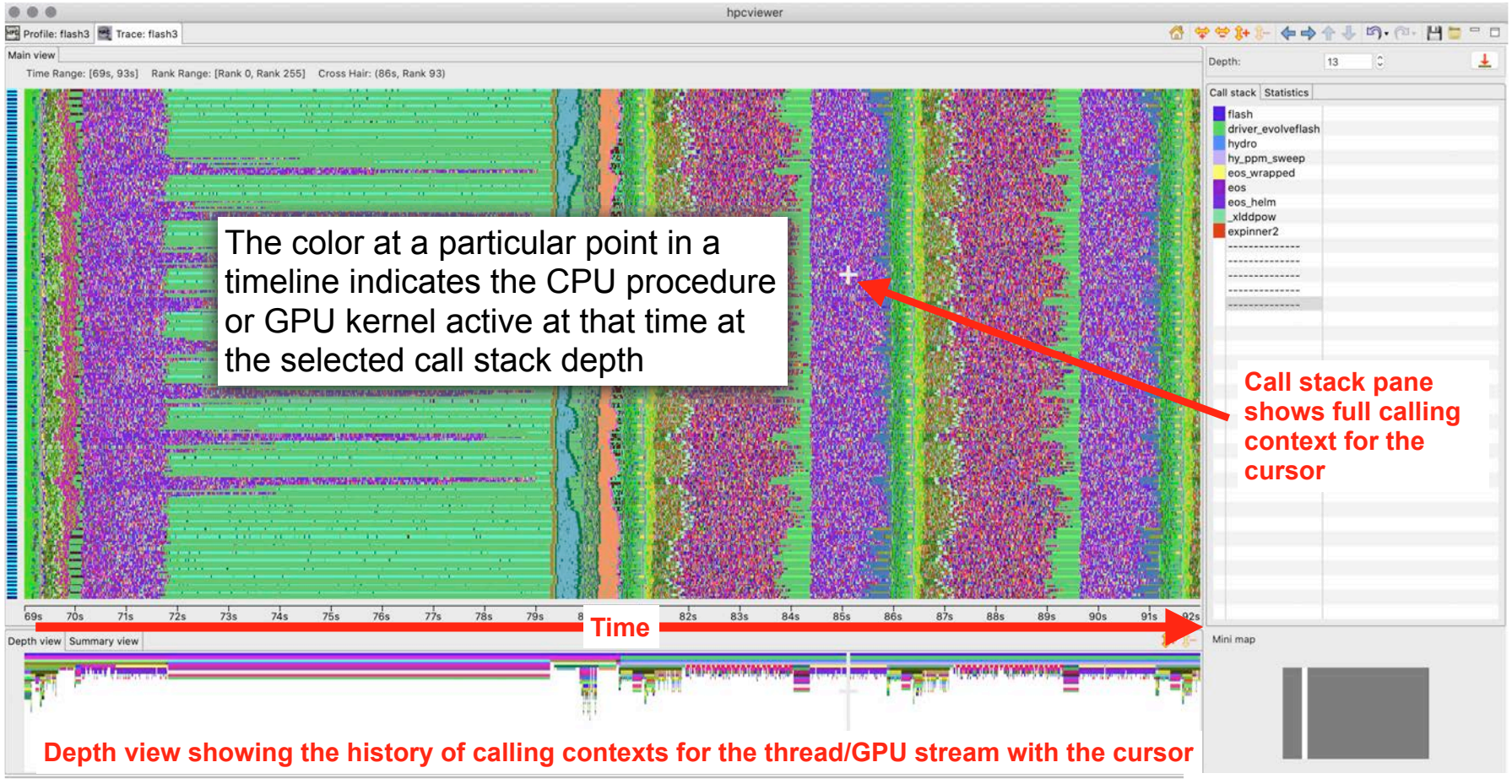


Understanding Temporal Behavior

- Profiling compresses out the temporal dimension
  - —Temporal patterns, e.g. serial sections and dynamic load imbalance are invisible in profiles
- What can we do? Trace call path samples
  - —N times per second, take a call path sample of each thread
  - —Organize the samples for each thread along a time line
  - —View how the execution evolves left to right
  - —What do we view? assign each procedure a color; view a depth slice of an execution



Understanding hpcviewer's trace view



A multi-level call stack based view of execution over time

Plan for This Session

- Prepare for hands-on
- Introduce HPCToolkit tools and workflow
  - Measurement (hpcrun)
  - Post-mortem analysis tools (hpcstruct, hpcprof)
  - Graphical user interface (hpcviewer)
- Illustrate hpctoolkit's use with some case studies
- Working with hands-on examples
  - Exploring pre-collected performance databases
  - Full contact

    scripted measurement, analysis, visualization of examples

Case Studies

- ExaWind (Nalu-Wind + AMRWind) - Wind turbine and wind farm simulation
- PeleLMeX - Adaptive mesh hydrodynamics code for low mach number reacting flows
- GAMESS (OpenMP) -
- Quicksilver (CUDA) -
- LAMMPS (Kokkos) at exascale

ExaWind: Modeling Turbine Wake Formation

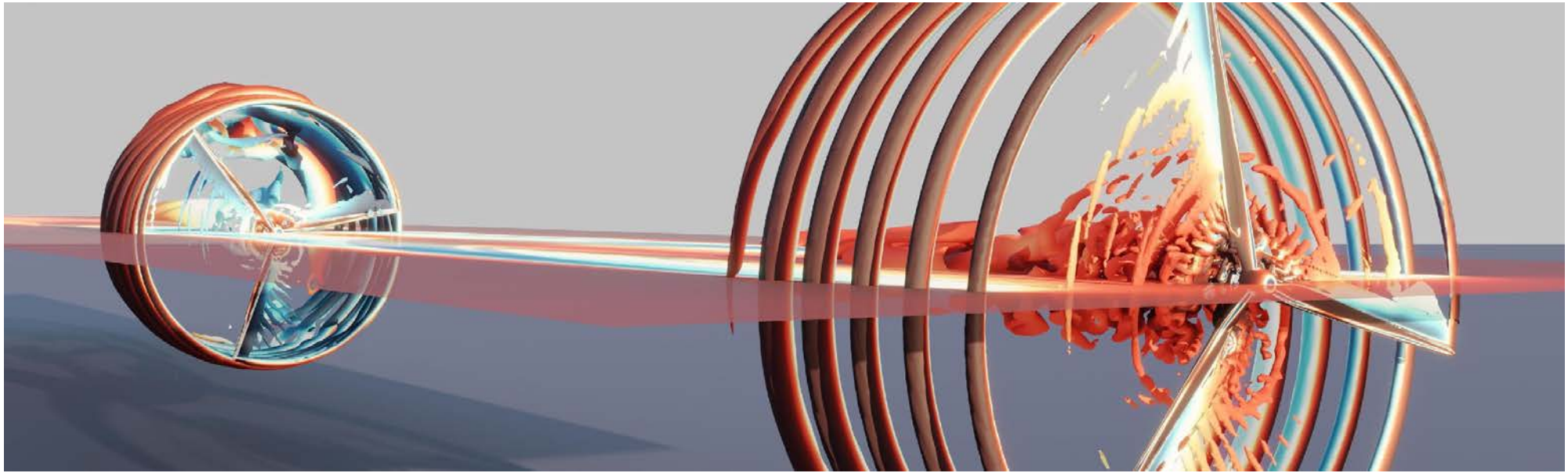


Figure credit: Jon Rood, NREL

ExaWind: Wakes from Three Turbines over Time

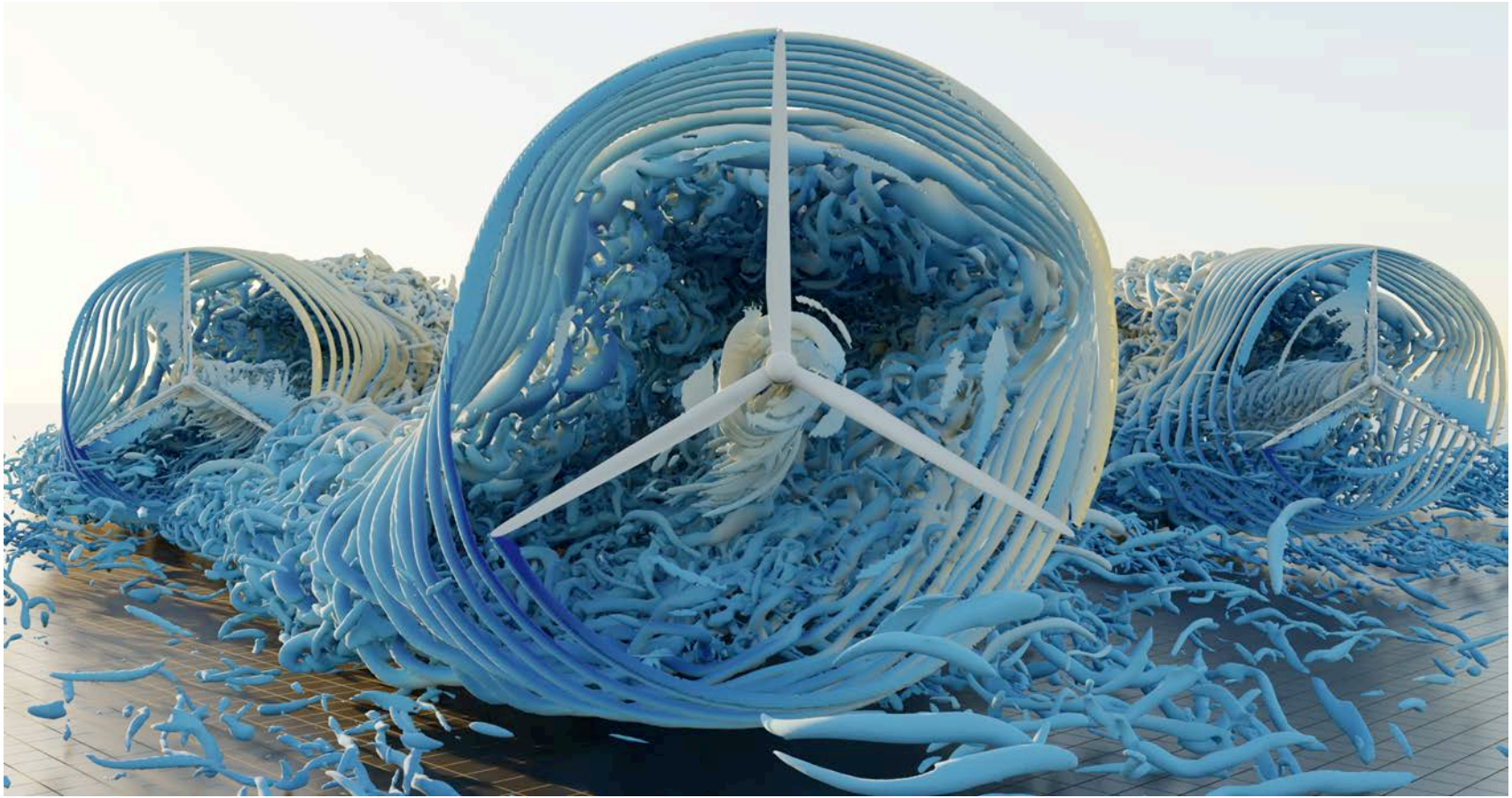


Figure credit: Jon Rood, NREL

ExaWind: Visualization of a Wind Farm Simulation

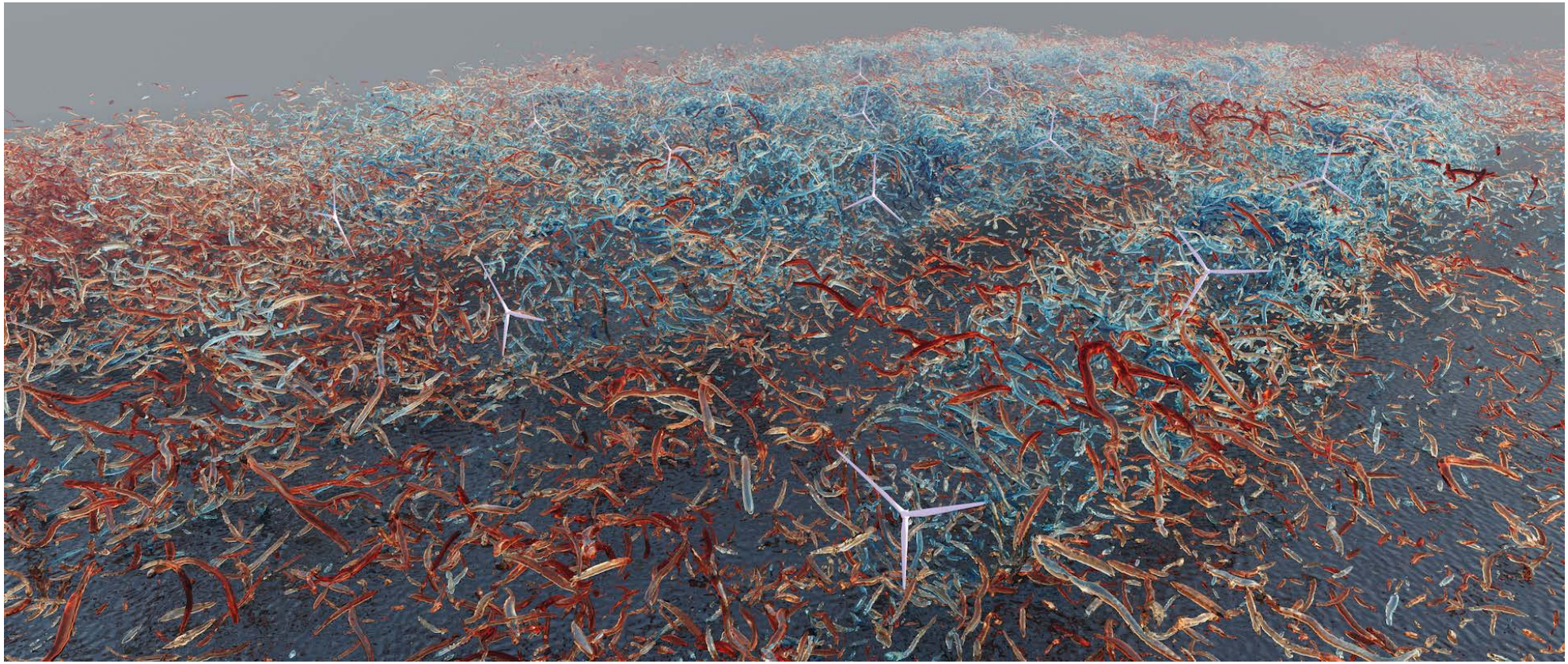


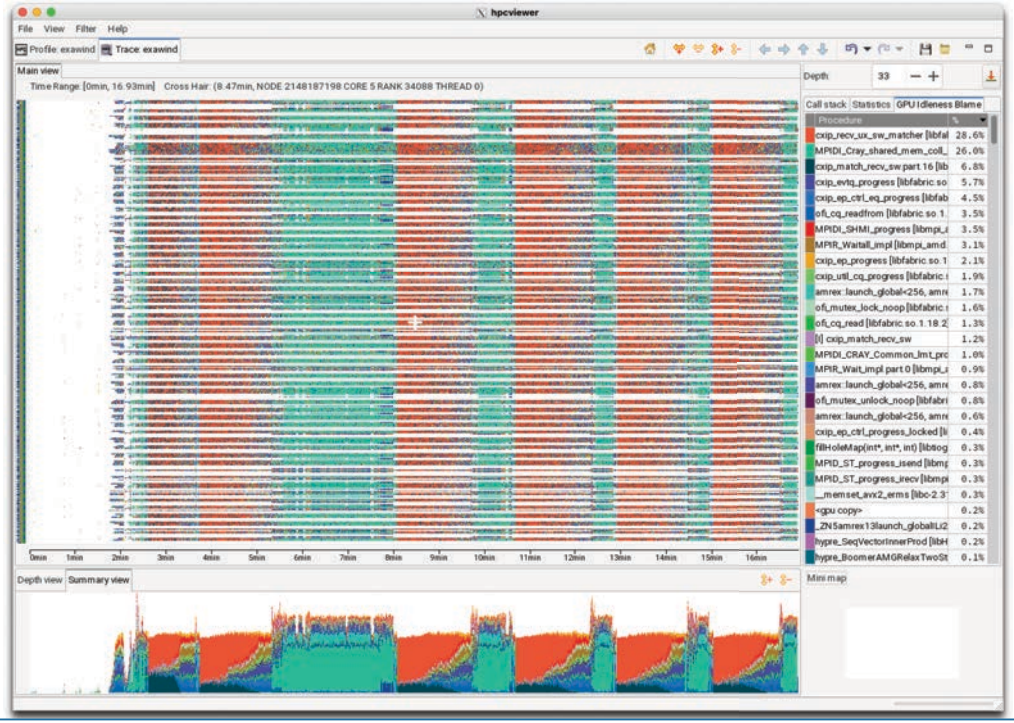
Figure credit: Jon Rood, NREL

ExaWind: Execution Traces on Frontier Collected with HPCToolkit

Traces on roughly 64K MPI ranks + 8K GPUs for ~17minutes

Before: MPI waiting (bad), shown in red

After: MPI overhead negligible*



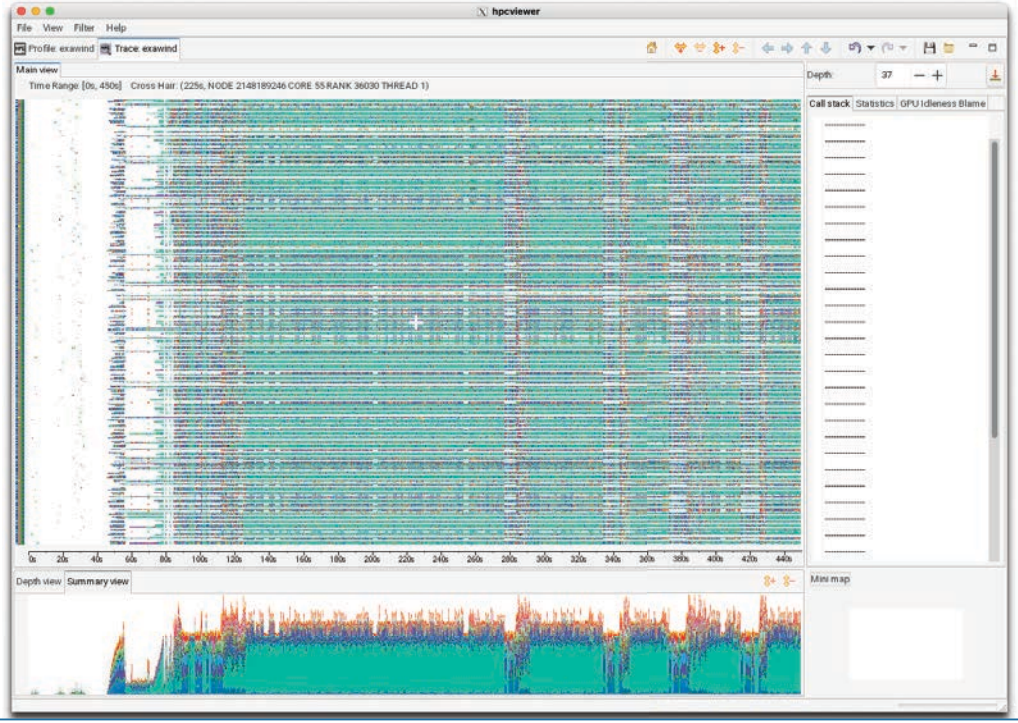


Figure credits: Jon Rood, NREL

*replaced non-blocking send/recv with ialltoallv

ExaWind Testimonials for HPCToolkit

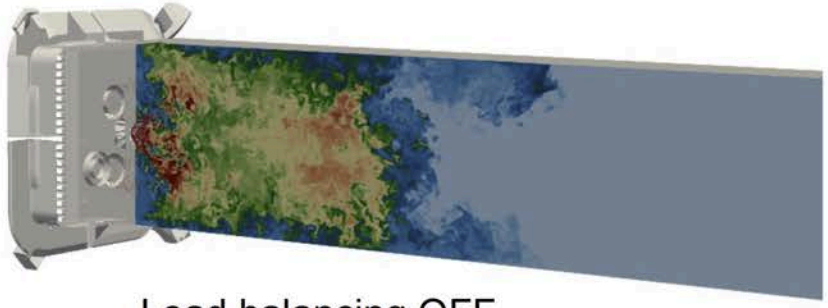
I just wanted to mention we've been using HPCToolkit a lot for our ExaWind application on Frontier, which is a hugely complicated code, and your profiler is one of the only ones we've found that really lets us easily instrument and then browse what our application is doing at runtime including GPUs. As an example, during a recent hackathon we had, we improved our large scale performance by 24x by understanding our code better with HPCToolkit and running it on 1000s of nodes while profiling. We also recently improved upon this by 10% for our total runtime.

- Jon Rood NREL (5/31/2024)

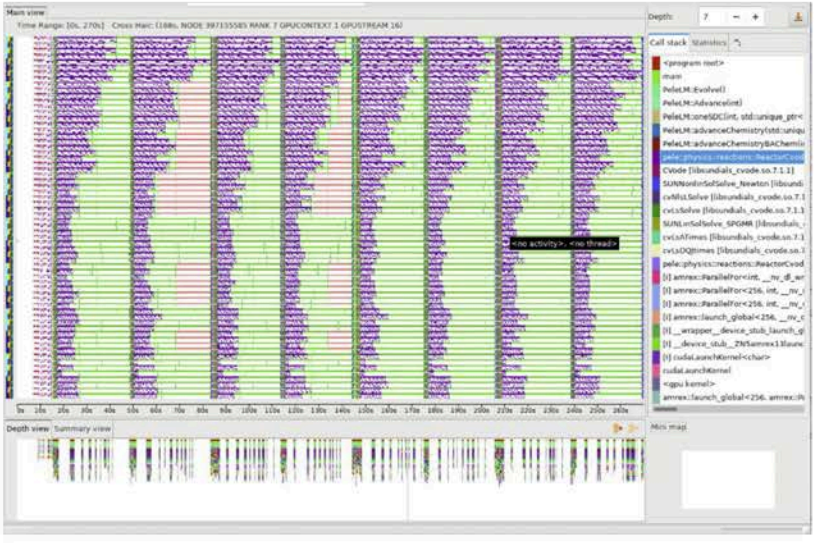
One big thing for us is that we can't overstate how complicated ExaWind is in general, and how complicated it is to build, so finding out that HPCToolkit could easily profile our entire application without a ton of instrumentation during the build process, and be able to profile it on a huge amount of Frontier with line numbers and visualizing the trace was really amazing to us.

- Jon Rood NREL (6/3/2024)

Full SAF Case: Load Balancing Issues



Load balancing OFF
(pink-> Cvode calls)



Load balancing ON

Confirmed on Frontier production runs: 60% speedup

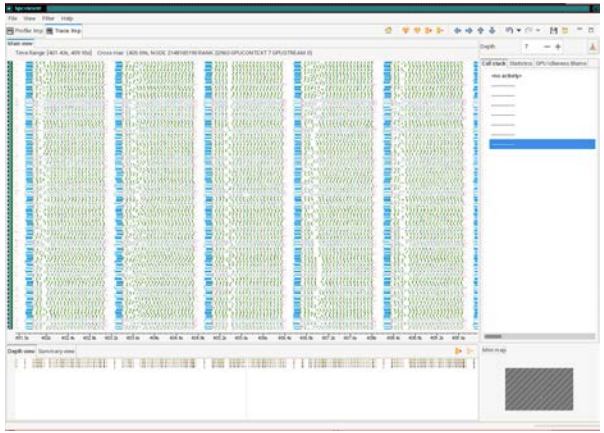
Grid from checkpoint file before regridding, Avg. Time/dt = 92.3 s

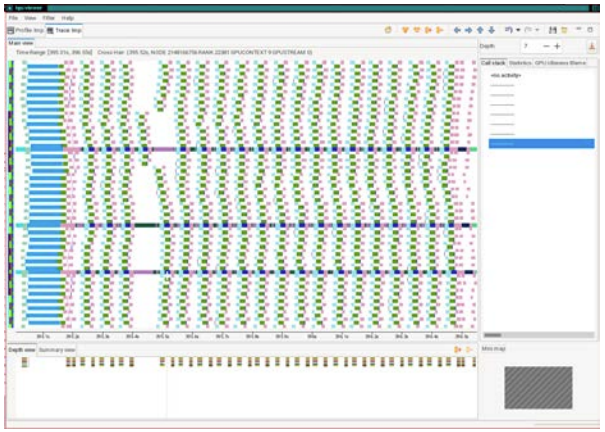
After regridding, Avg. Time/dt = 62.5s
- Down to 40s when setting amr.max_grid_size=32

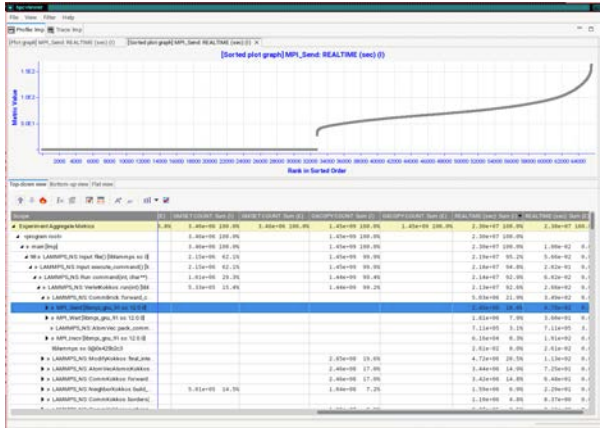
OpenACC
OPEN HACKATHONS

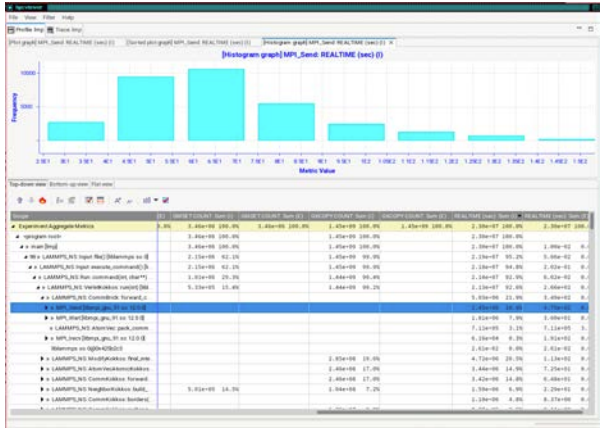
Figure credit: PeleLMEX Team, NREL Hackathon, February 2025

LAMMPS on Frontier: Executions with Kernel Duration of Milliseconds

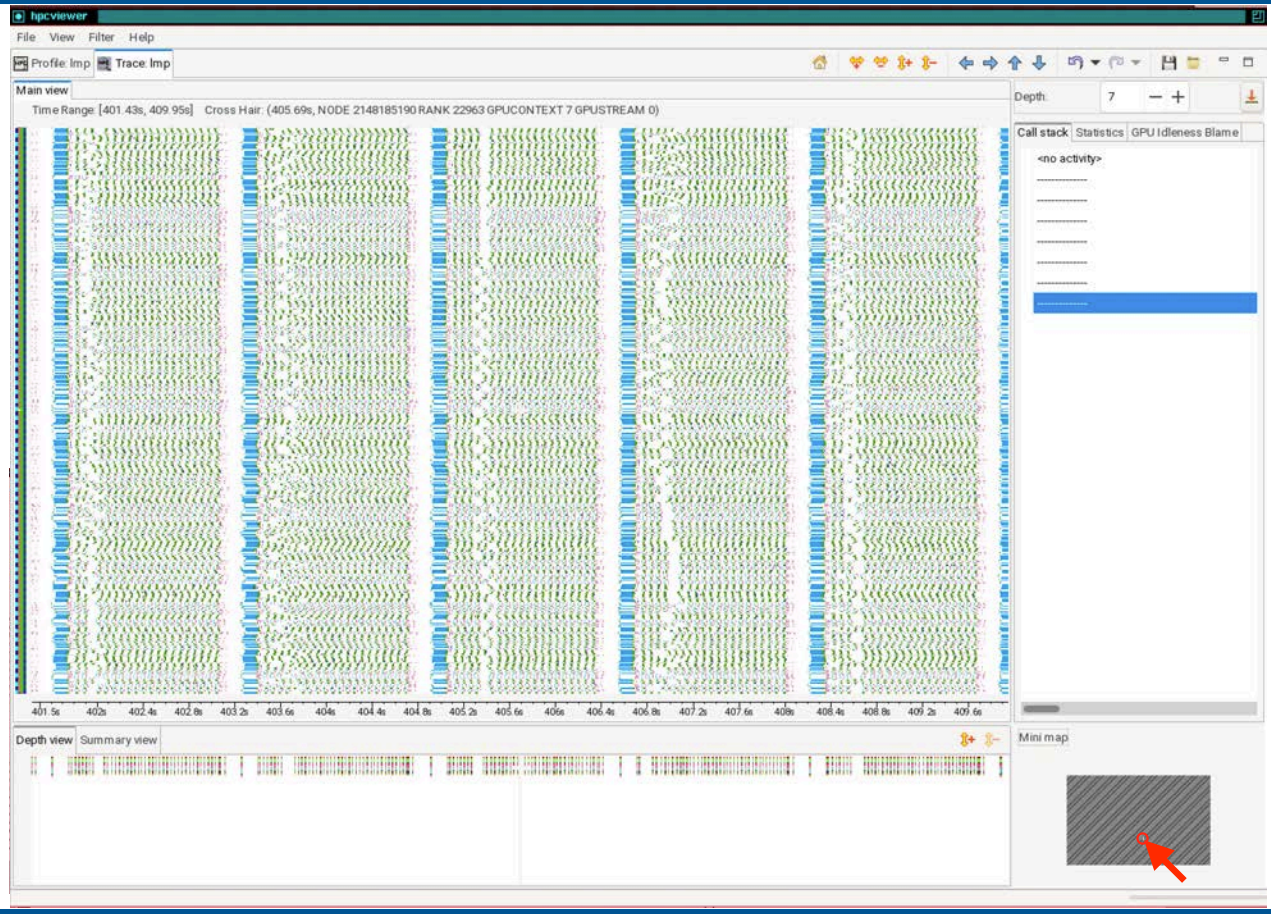




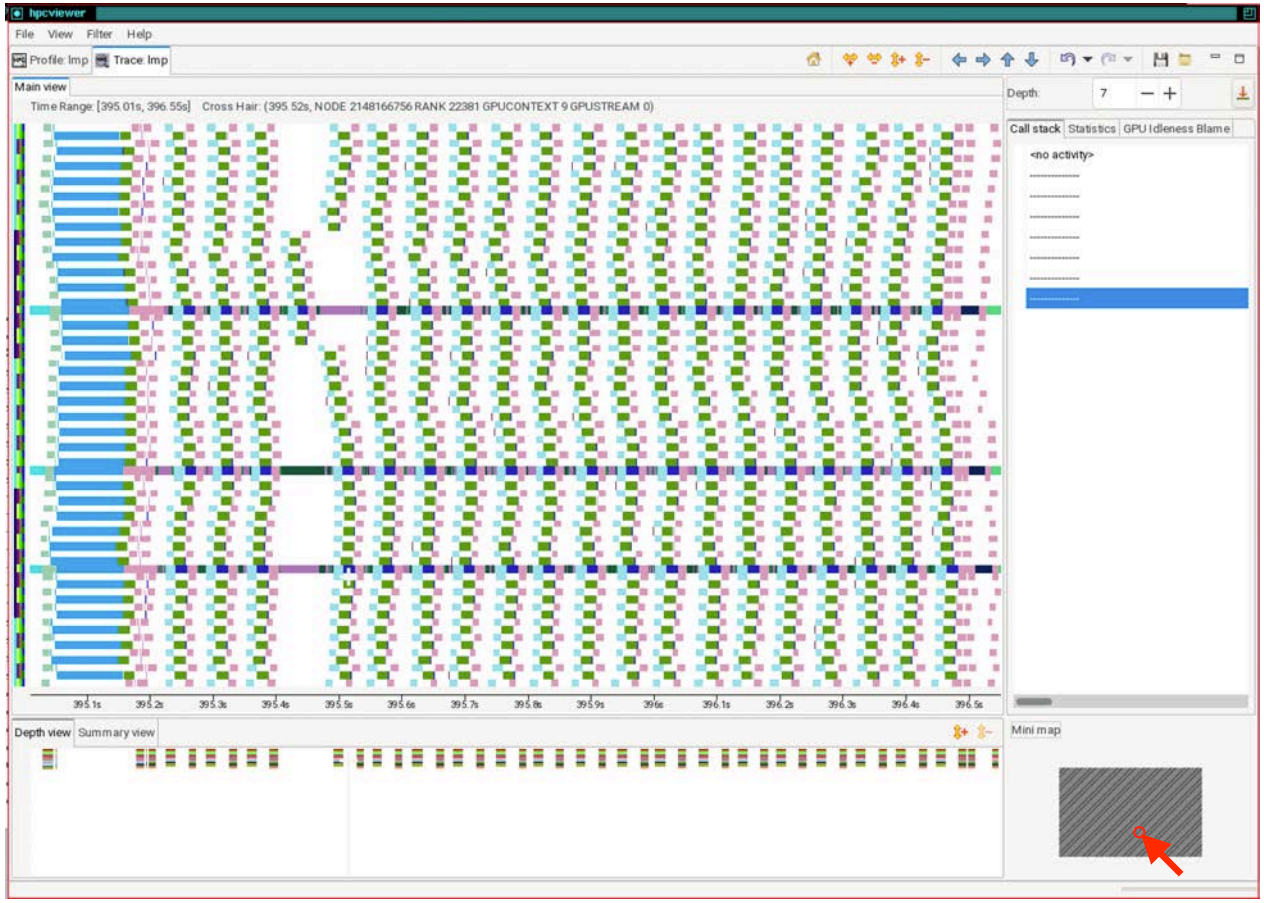




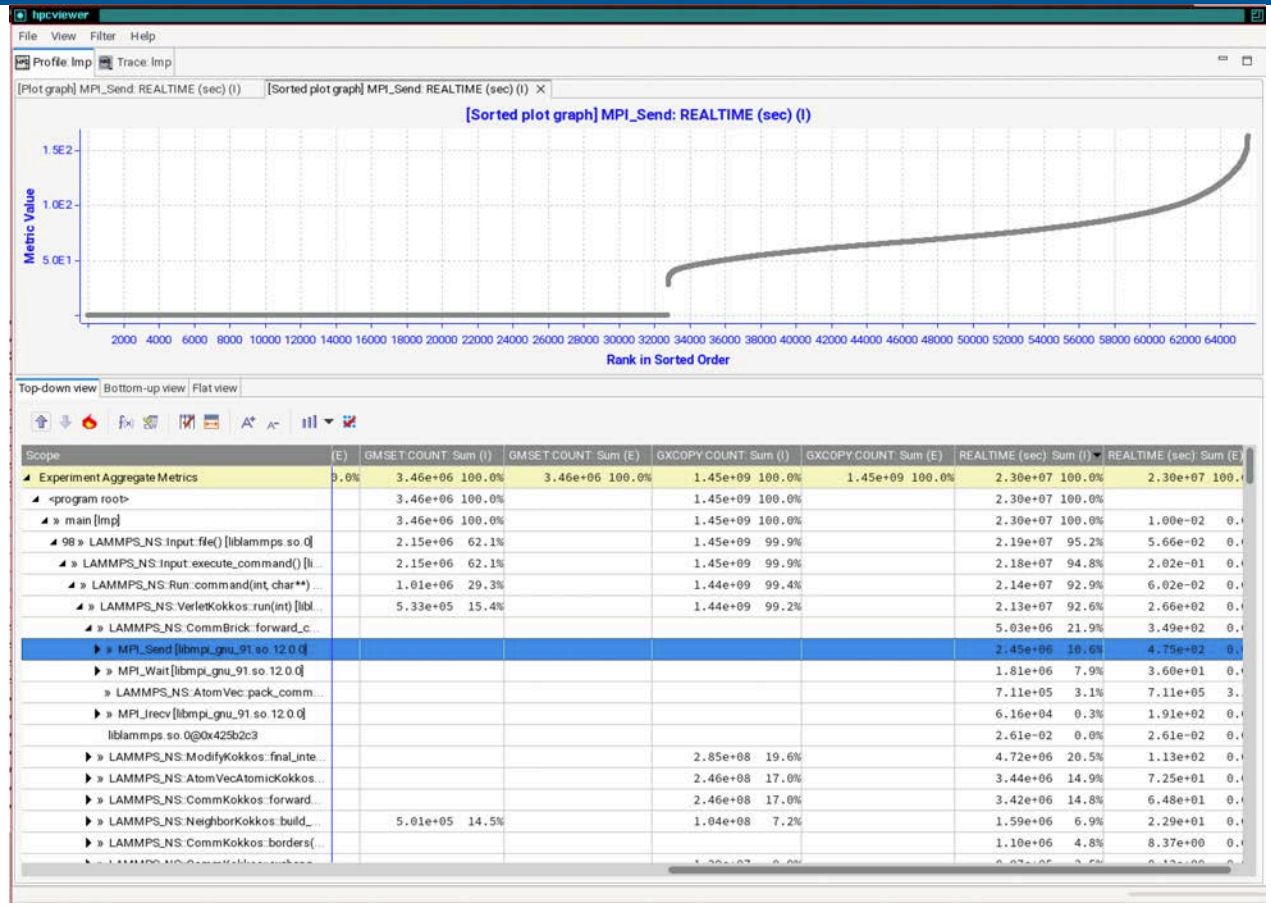
LAMMPS on Frontier: Executions with Kernel Duration of Milliseconds



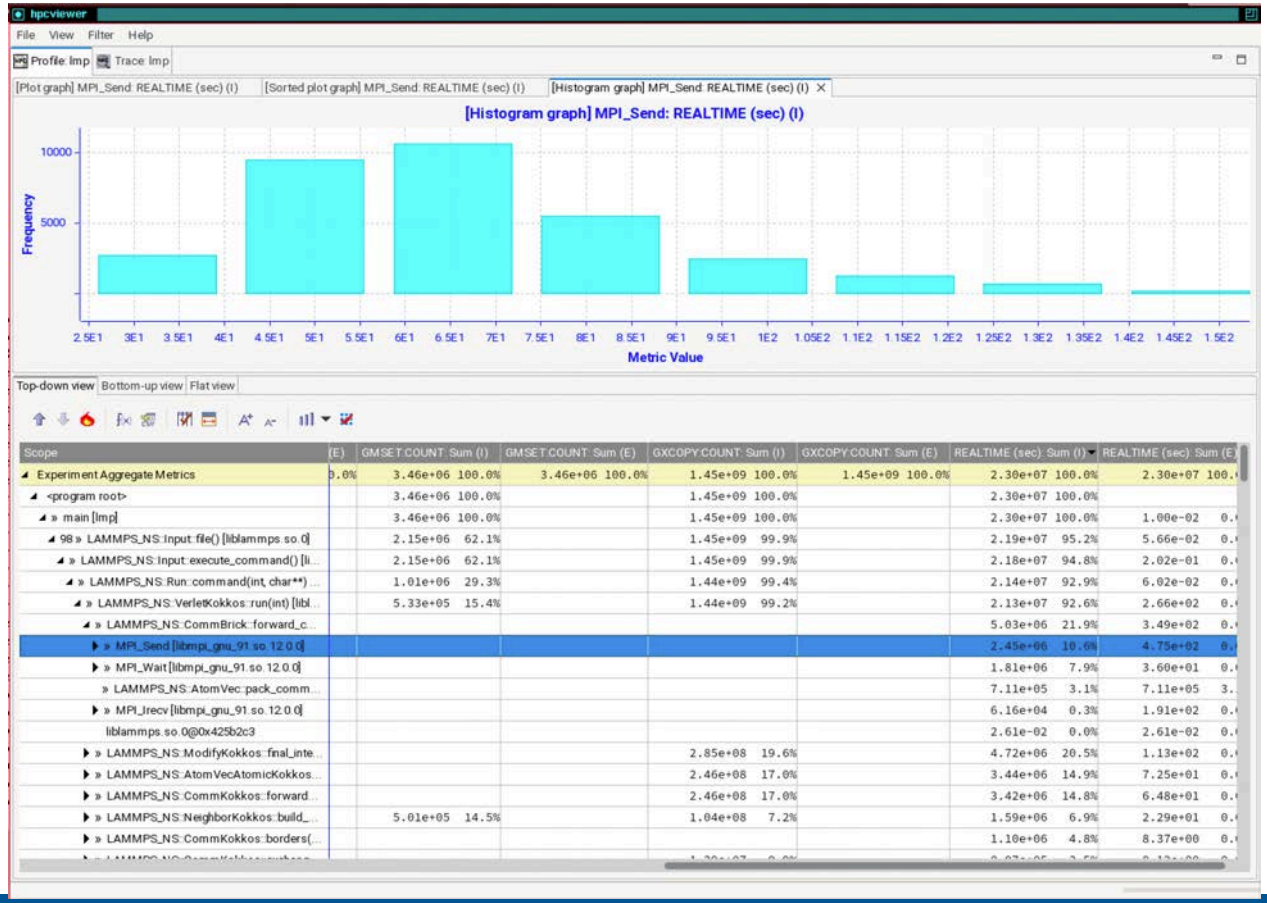
LAMMPS on Frontier: Executions with Kernel Duration of Milliseconds



LAMMPS on Frontier: Executions with Kernel Duration of Milliseconds

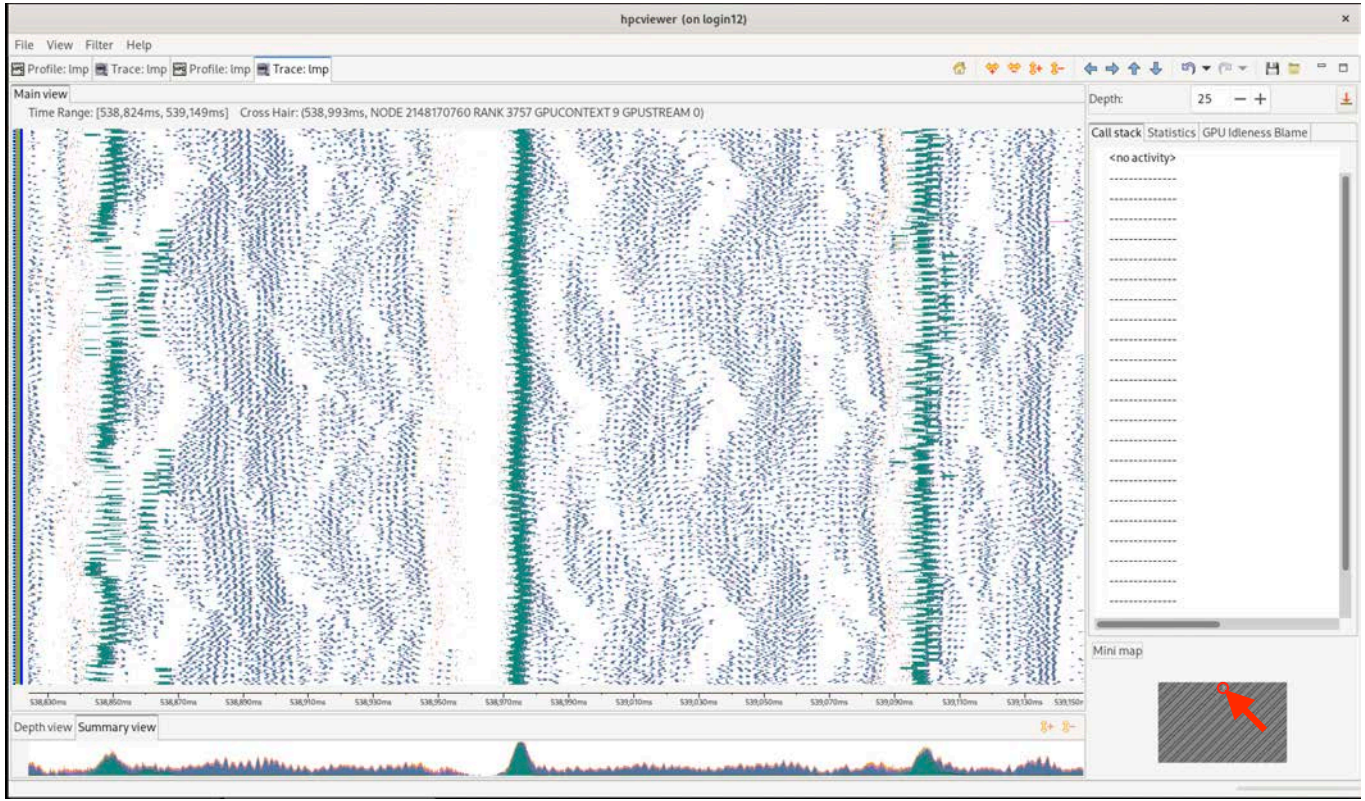


LAMMPS on Frontier: Executions with Kernel Duration of Milliseconds



LAMMPS on Frontier: 8K nodes, 64K MPI ranks + 64K GPU tiles

Kernel duration of microseconds



Case Study: GAMESS

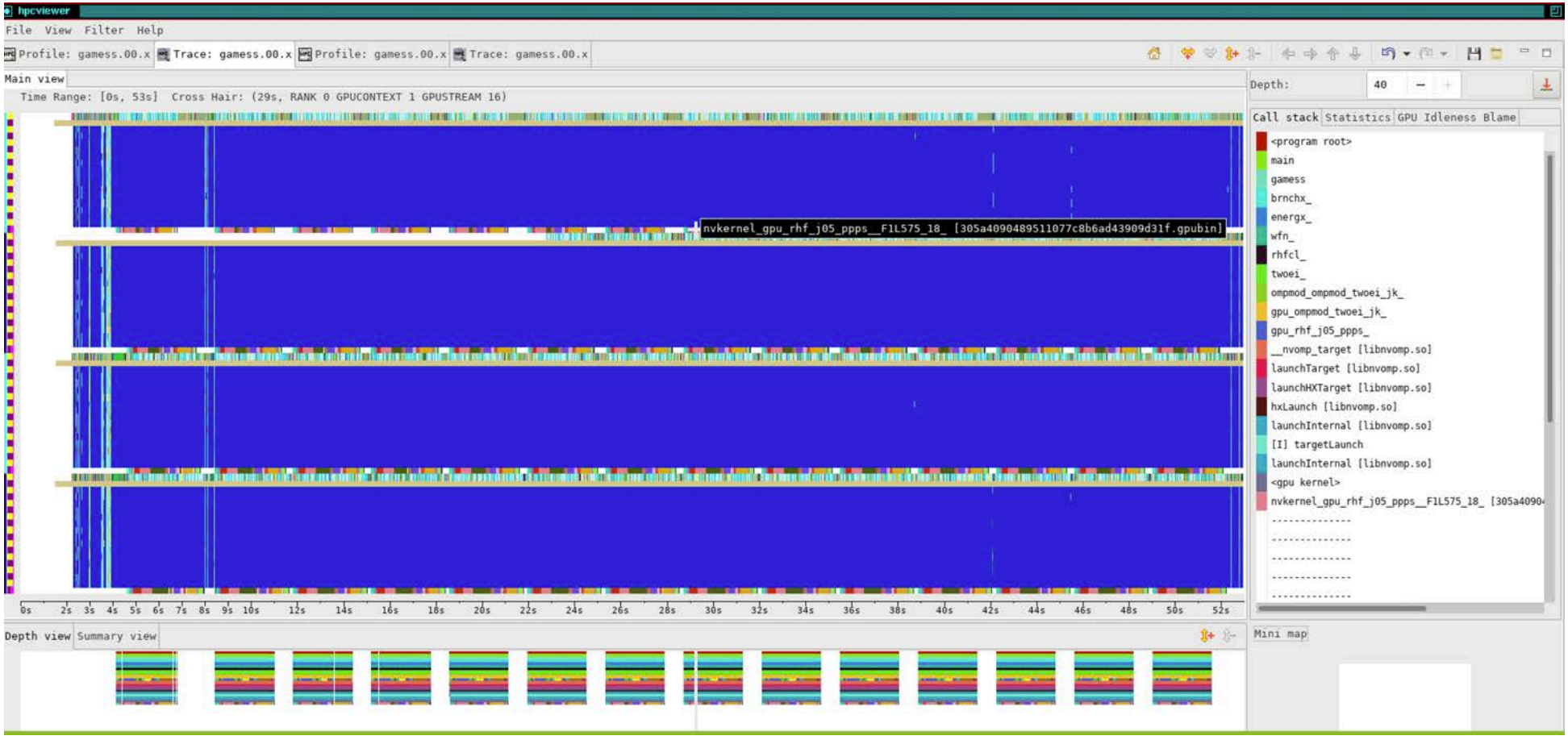
- General Atomic and Molecular Electronic Structure System (GAMESS)
  —general *ab initio* quantum chemistry package
- Calculates the energies, structures, and properties of a wide range of chemical systems

- Experiments
  - GPU-accelerated nodes at a prior Perlmutter hackathon
    - Single node with 4 GPUs
    - Five nodes with 20 GPUs

Perlmutter node at a glance

AMD Milan CPU
4 NVIDIA A100 GPUs
256 GB memory

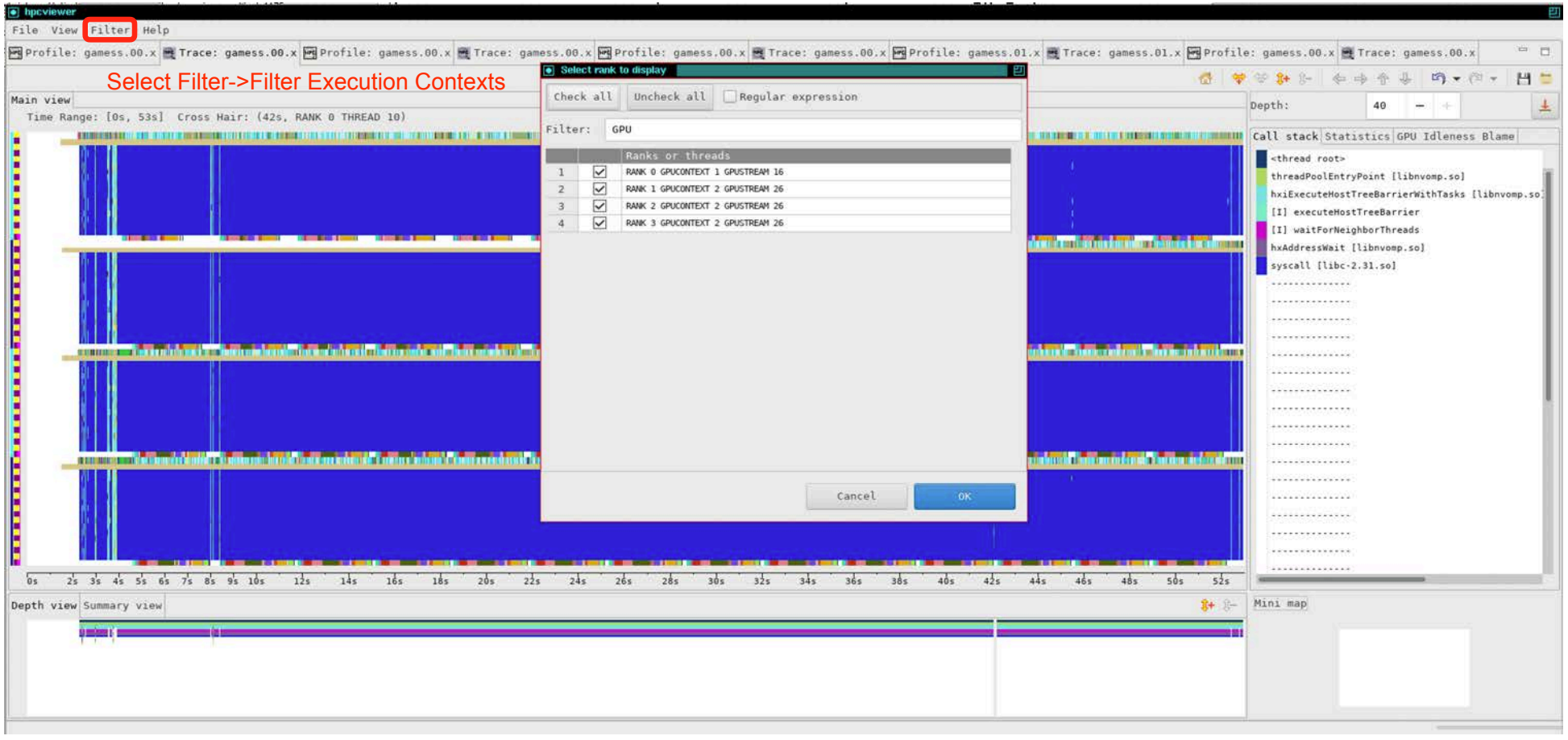
Time-centric Analysis: GAMESS 4 ranks, 4 GPUs on Perlmutter



GAMESS original

All CPU threads and GPU streams

Time-centric Analysis: GAMESS 4 ranks, 4 GPUs on Perlmutter



GAMESS original

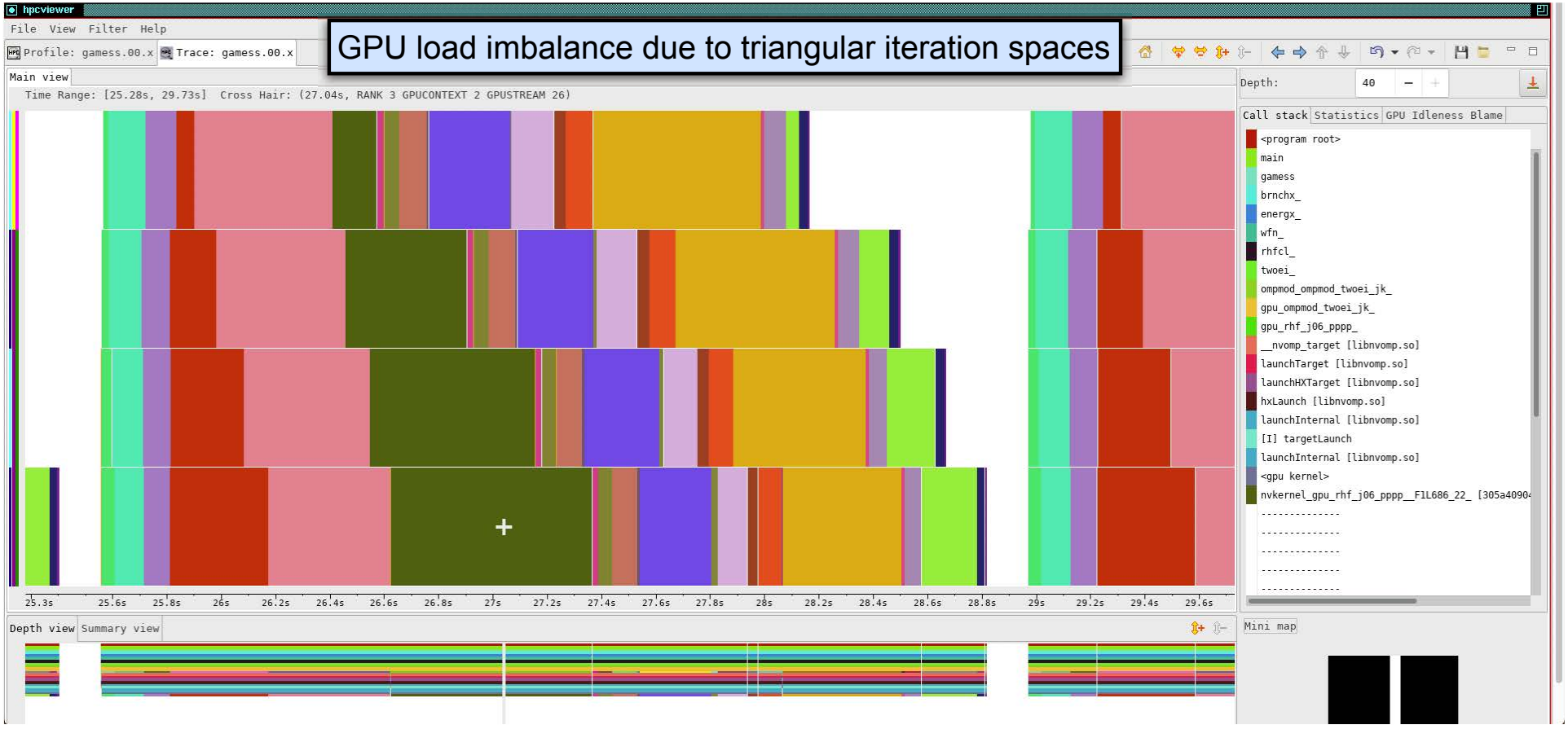
All CPU threads and GPU streams

Time-centric Analysis: GAMESS 4 ranks, 4 GPUs on Perlmutter

GAMESS original

All GPU streams; whole execution

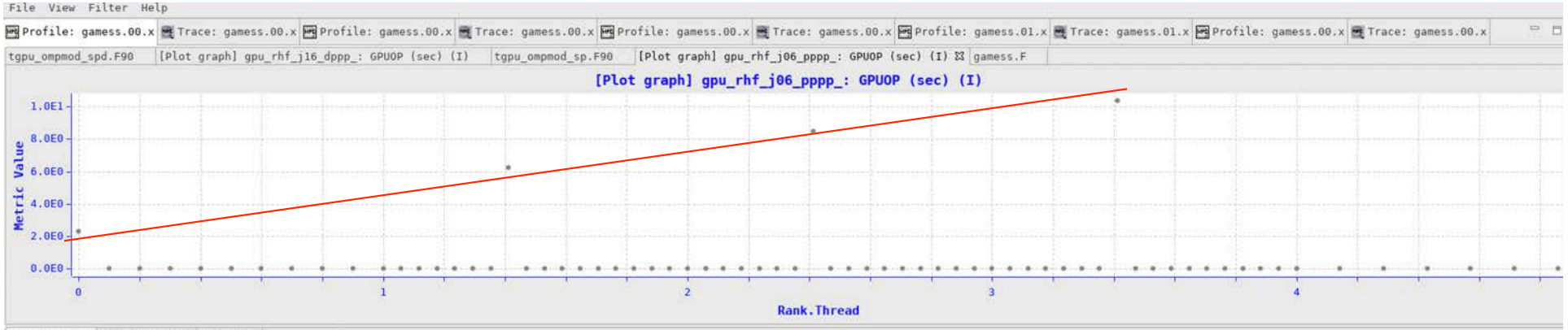
Time-centric Analysis: GAMESS 4 ranks, 4 GPUs on Perlmutter



GAMESS original

GPU streams: 1 iteration

Time-centric Analysis: GAMESS 4 ranks, 4 GPUs on Perlmutter

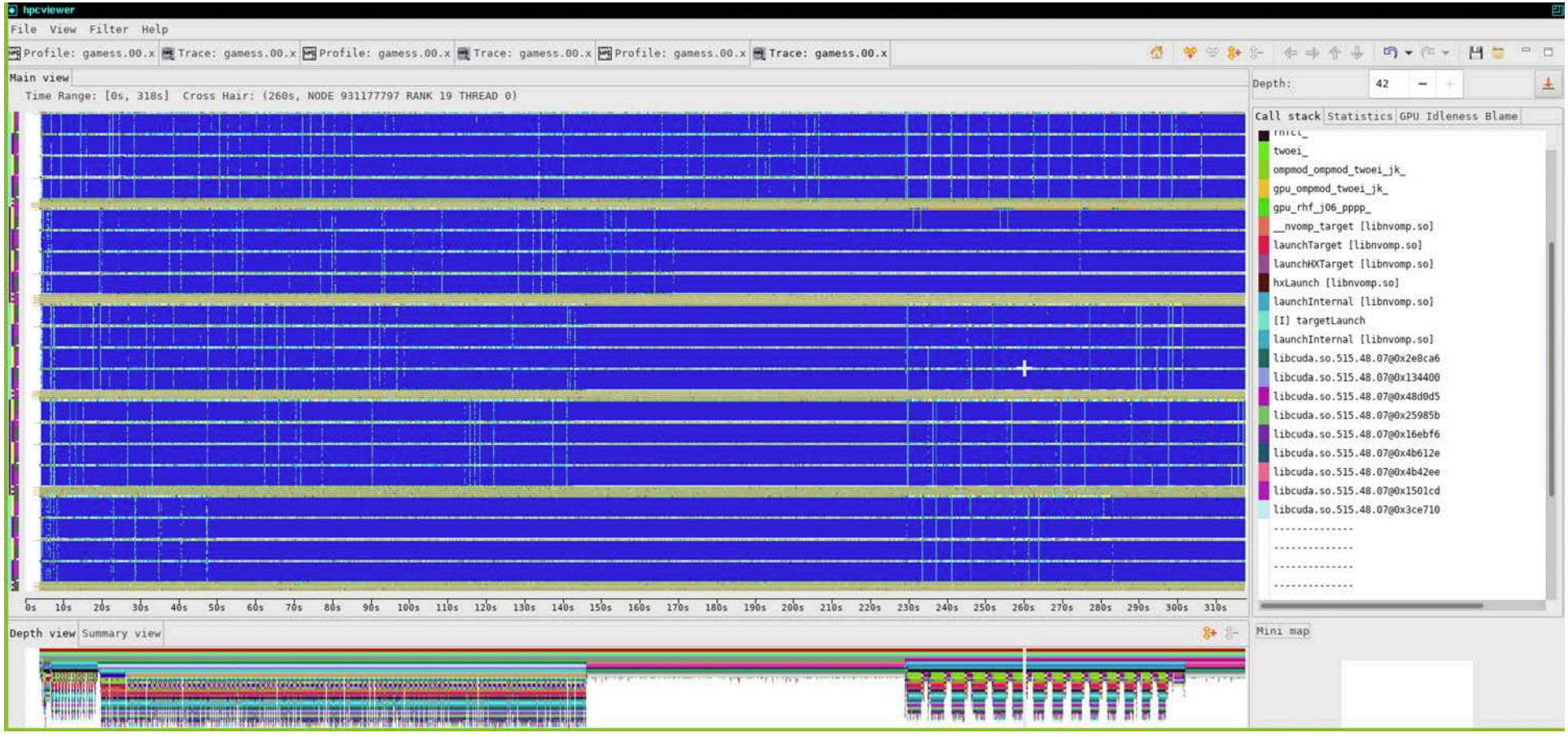


Top-down view | Bottom-up view | Flat view

Scope	REALTIME (sec): Sum (I)	REALTIME (sec): Sum (E)	GPUOP (sec): Sum (I)	GPUOP (sec): Sum (E)	GKER (sec): Sum (I)	GKER (sec): Sum (E)	GXCOPY (sec): Sum (I)	GXCOPY (sec): Sum (E)
Experiment Aggregate Metrics	3.48e+03 100.0%	3.48e+03 100.0%	1.66e+02 100.0%	1.66e+02 100.0%	1.66e+02 100.0%	1.66e+02 100.0%	2.64e-01 100.0%	2.64e-01 10
<program root>	2.04e+02 5.9%		1.66e+02 100.0%		1.66e+02 100.0%		2.64e-01 100.0%	
» main [gamess.00.x]	2.04e+02 5.9%		1.66e+02 100.0%		1.66e+02 100.0%		2.64e-01 100.0%	
» gamess	2.04e+02 5.9%		1.66e+02 100.0%		1.66e+02 100.0%		2.64e-01 100.0%	
772 » brnchx_	2.03e+02 5.8%		1.66e+02 100.0%		1.66e+02 100.0%		2.64e-01 100.0%	
1095 » energx_	2.03e+02 5.8%		1.66e+02 100.0%		1.66e+02 100.0%		2.64e-01 100.0%	
loop at gamess.F: 1316	2.03e+02 5.8%		1.66e+02 100.0%		1.66e+02 100.0%		2.64e-01 100.0%	
loop at gamess.F: 1436	1.97e+02 5.7%		1.66e+02 100.0%		1.66e+02 100.0%		2.64e-01 100.0%	
loop at gamess.F: 1436	1.97e+02 5.7%		1.66e+02 100.0%		1.66e+02 100.0%		2.64e-01 100.0%	
1440 » wfn_	1.97e+02 5.7%		1.66e+02 100.0%		1.66e+02 100.0%		2.64e-01 100.0%	
loop at gamess.F: 2645	1.97e+02 5.7%		1.66e+02 100.0%		1.66e+02 100.0%		2.64e-01 100.0%	
2568 » rhfcl_	1.97e+02 5.7%	1.04e-02 0.0%	1.66e+02 100.0%		1.66e+02 100.0%		2.64e-01 100.0%	
loop at rhfuhf.f: 2678	1.96e+02 5.7%	5.25e-03 0.0%	1.66e+02 100.0%		1.66e+02 100.0%		2.64e-01 100.0%	
loop at rhfuhf.f: 2723	1.96e+02 5.6%		1.66e+02 100.0%		1.66e+02 100.0%		2.64e-01 100.0%	
loop at rhfuhf.f: 2723	1.96e+02 5.6%		1.66e+02 100.0%		1.66e+02 100.0%		2.64e-01 100.0%	
2859 » twoei_	1.69e+02 4.9%		1.66e+02 100.0%		1.66e+02 100.0%		2.64e-01 100.0%	
3994 » ompmod_ompmod_twoei_jk_	1.69e+02 4.9%		1.66e+02 100.0%		1.66e+02 100.0%		2.64e-01 100.0%	
246 » gpu_ompmod_twoei_jk_	1.69e+02 4.9%		1.66e+02 100.0%		1.66e+02 100.0%		2.64e-01 100.0%	
656 » gpu_rhf_j16_dppp_	2.98e+01 0.9%	1.00e-02 0.0%	2.97e+01 17.9%		2.97e+01 17.9%		1.07e-02 4.1%	
589 » gpu_rhf_j06_pppp_	2.74e+01 0.8%	5.00e-03 0.0%	2.74e+01 16.5%		2.74e+01 16.5%		8.56e-03 3.2%	

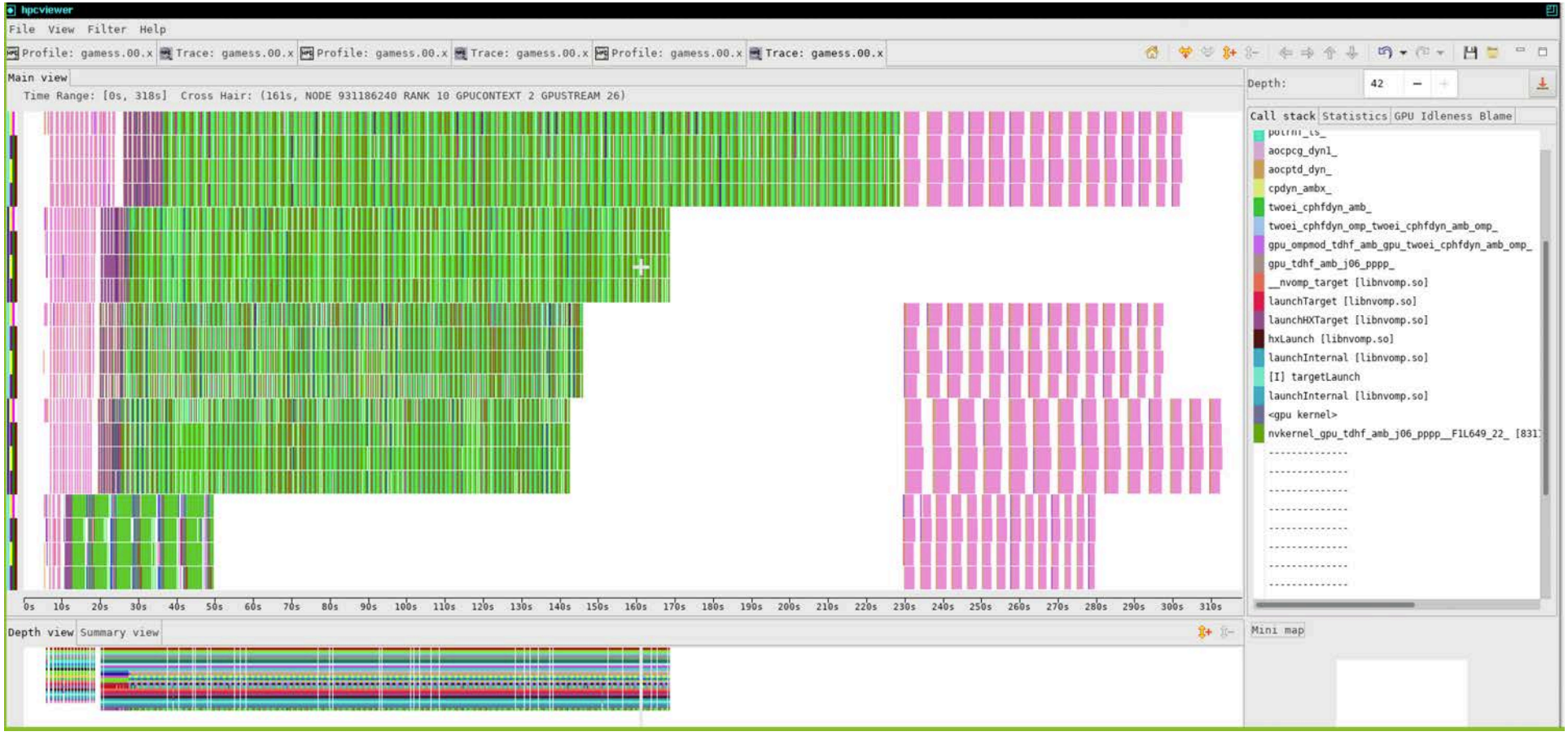
GAMESS original

Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter

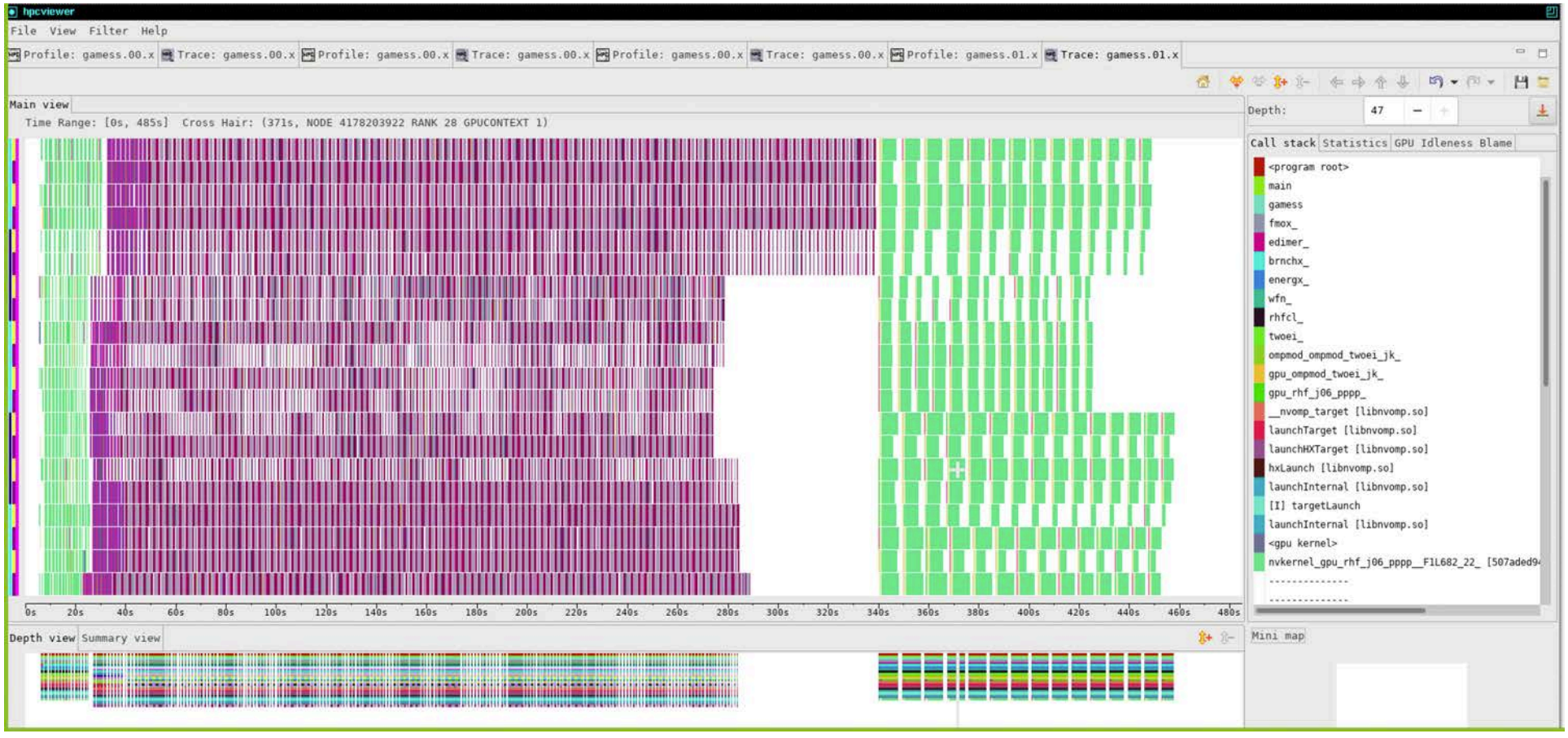


CPU Threads and GPU Streams

Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter

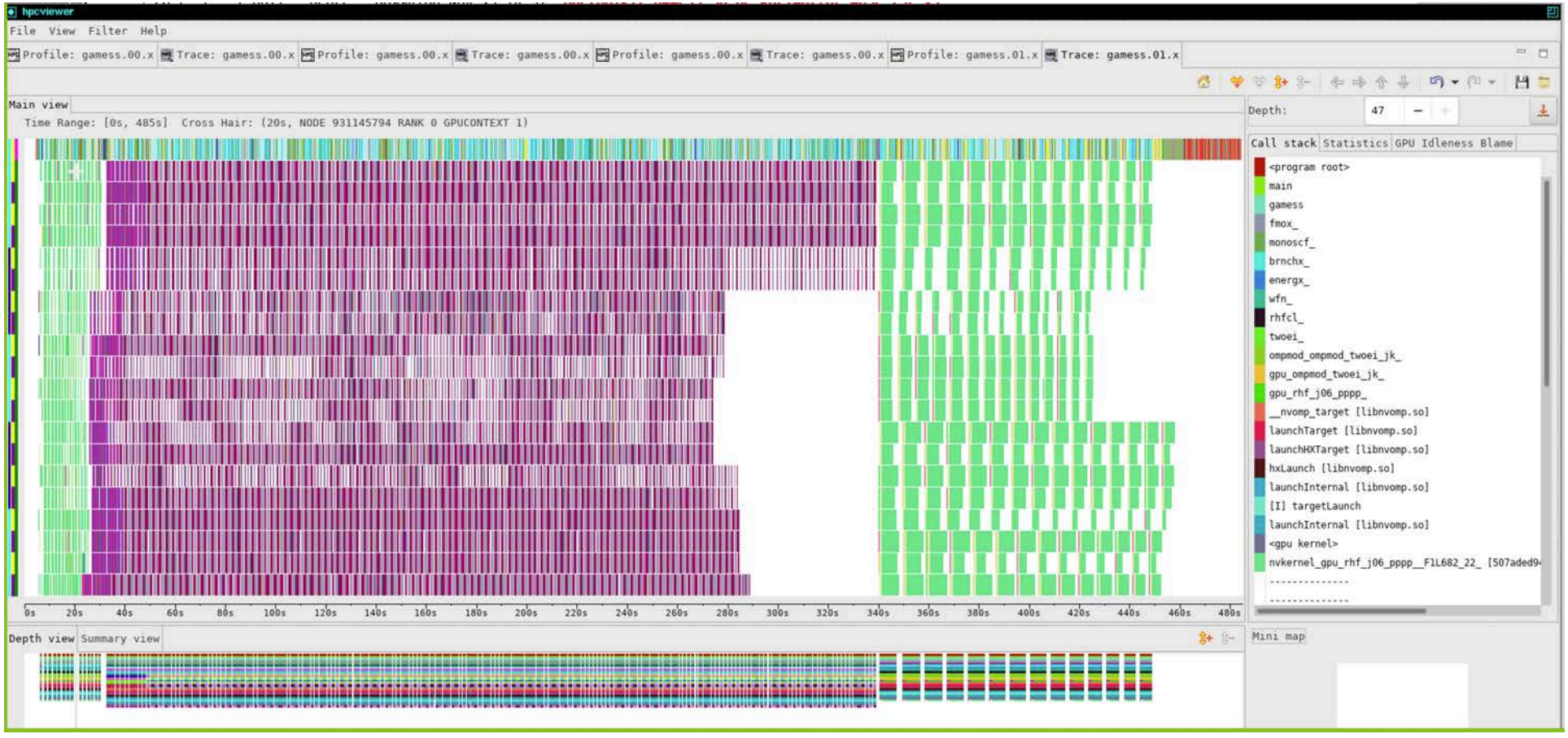


Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter



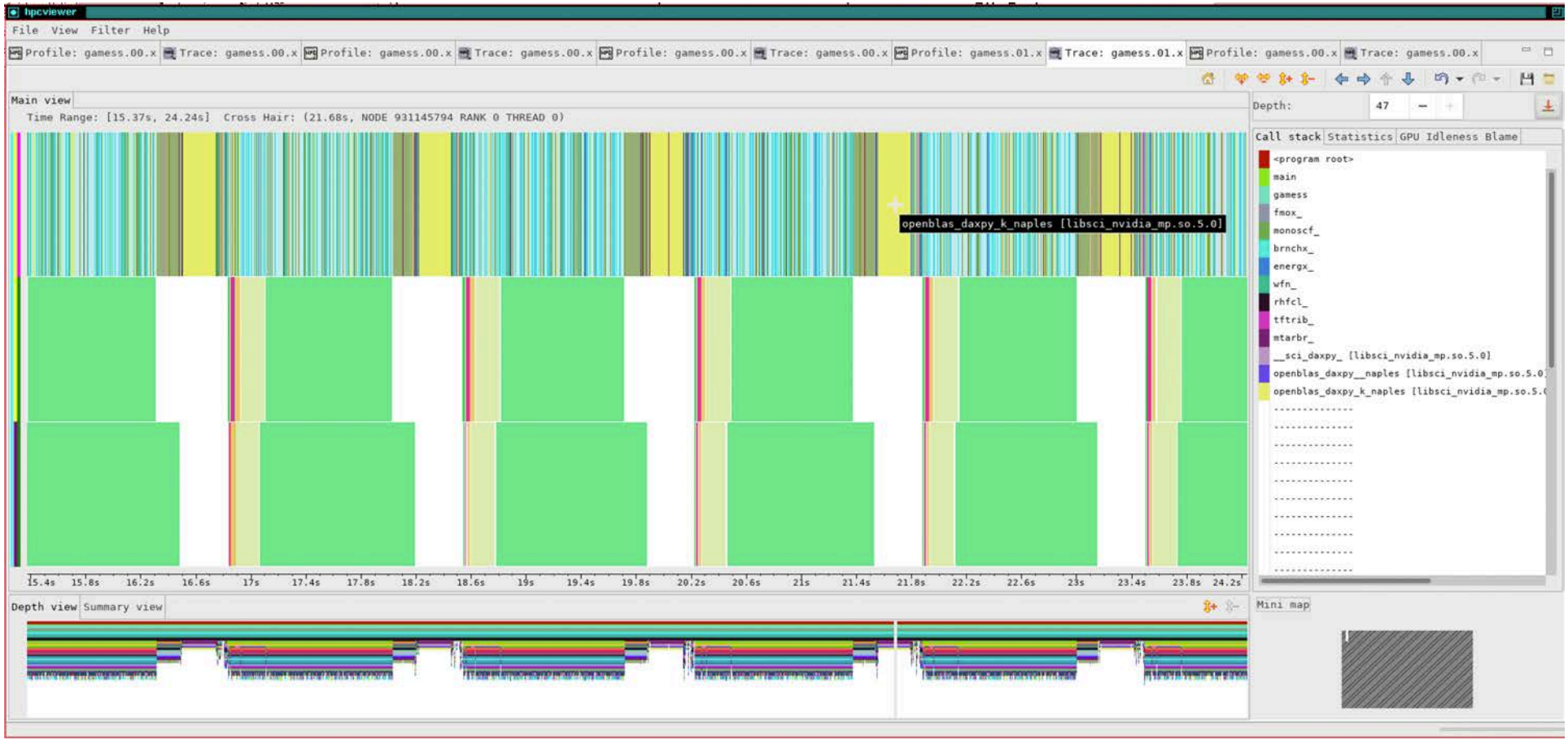
GAMESS improved with better manual distribution of work in input

Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter



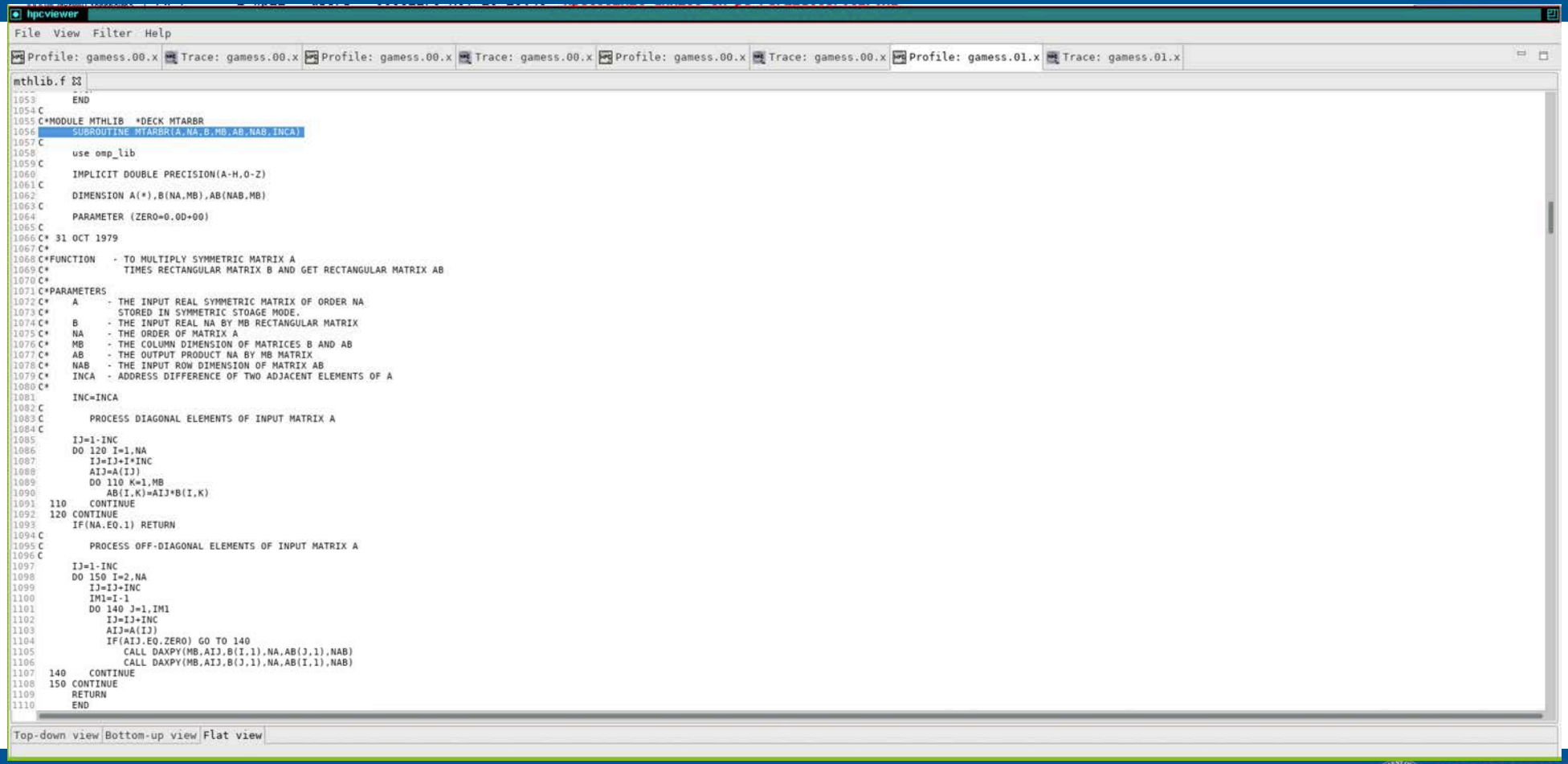
GAMESS improved adding Rank 0 Thread 0 to GPU streams

Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter

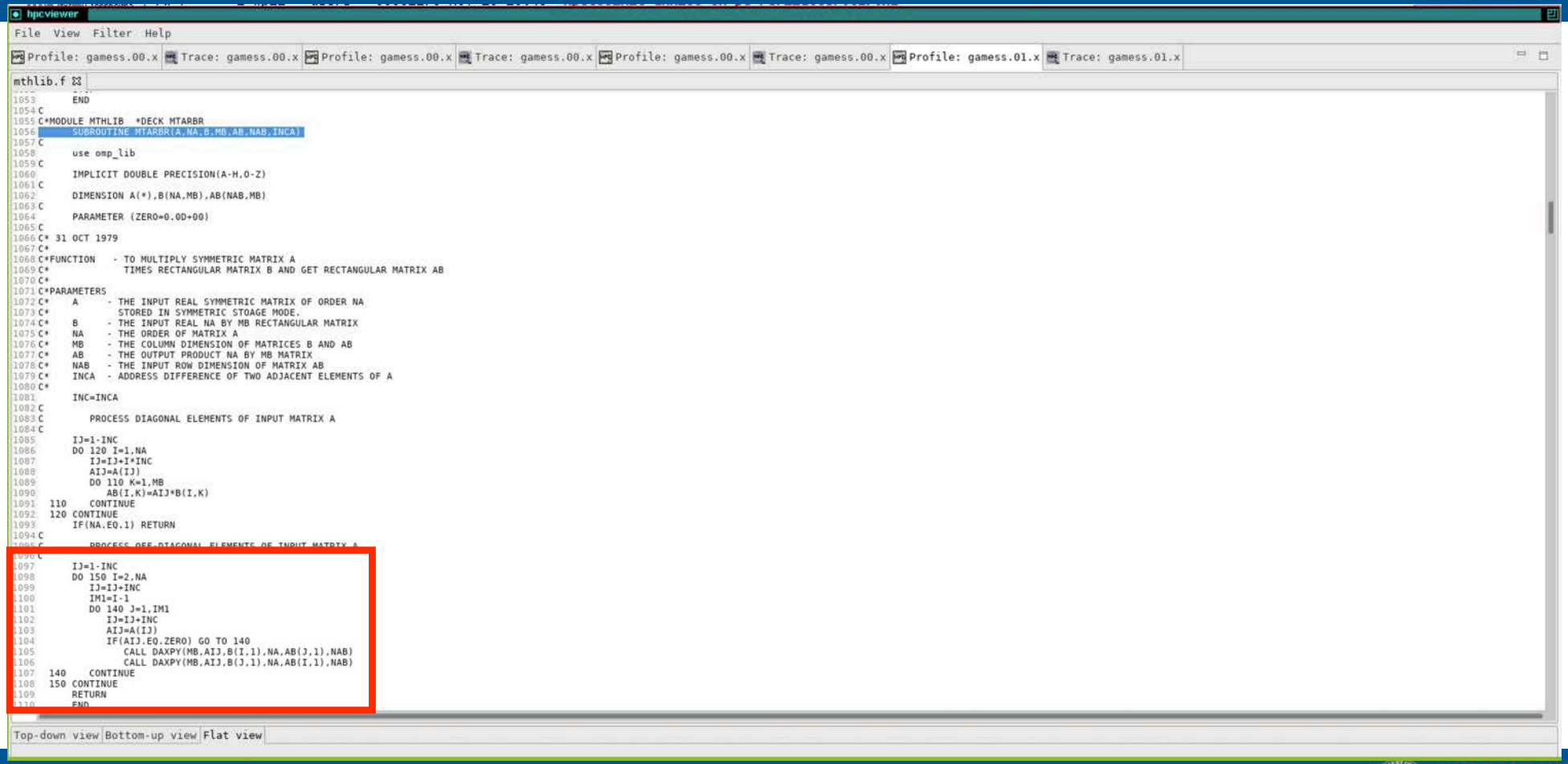


1 CPU Stream, 2 GPU Streams: 6 Iterations

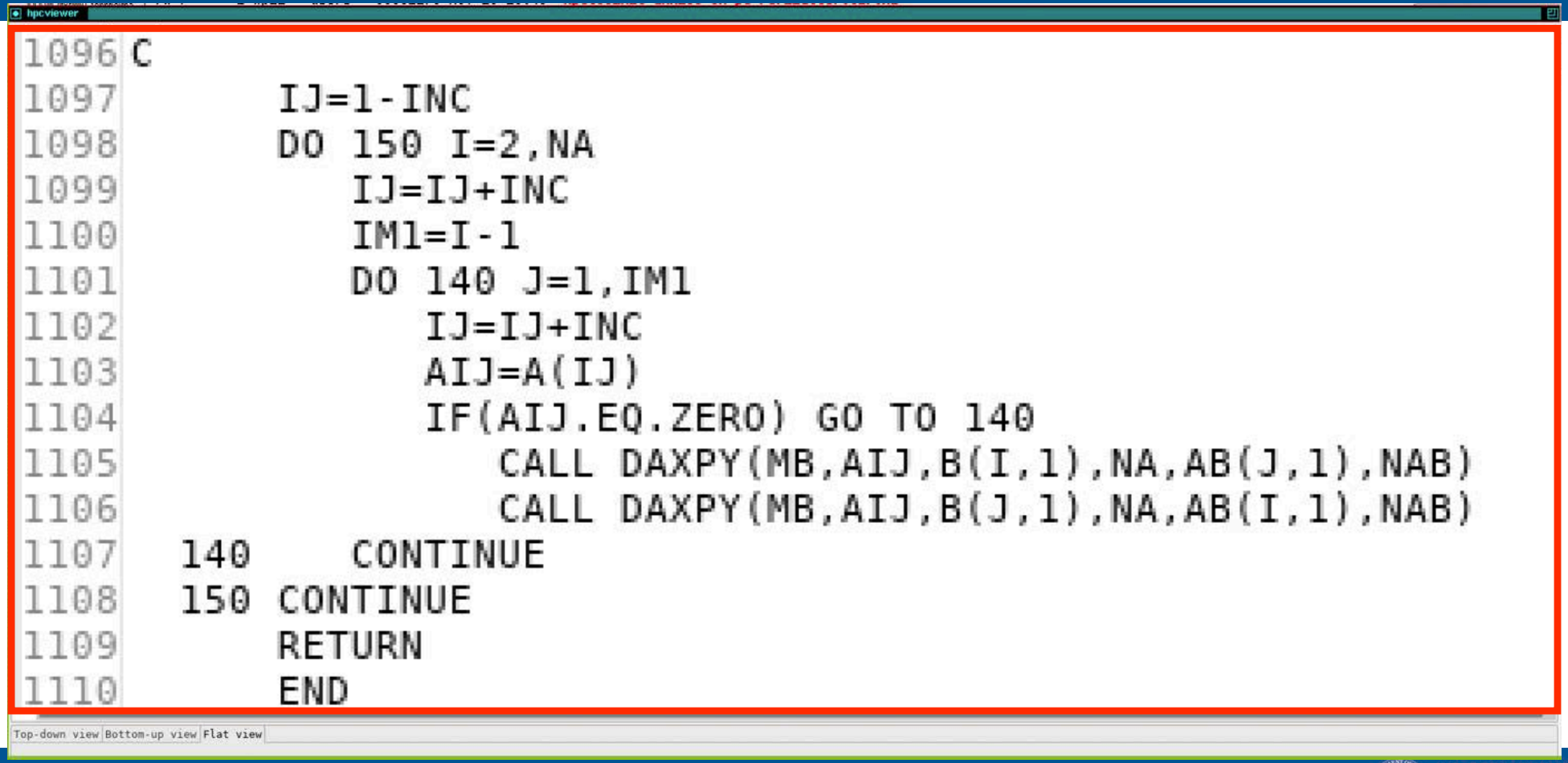
Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter



Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter



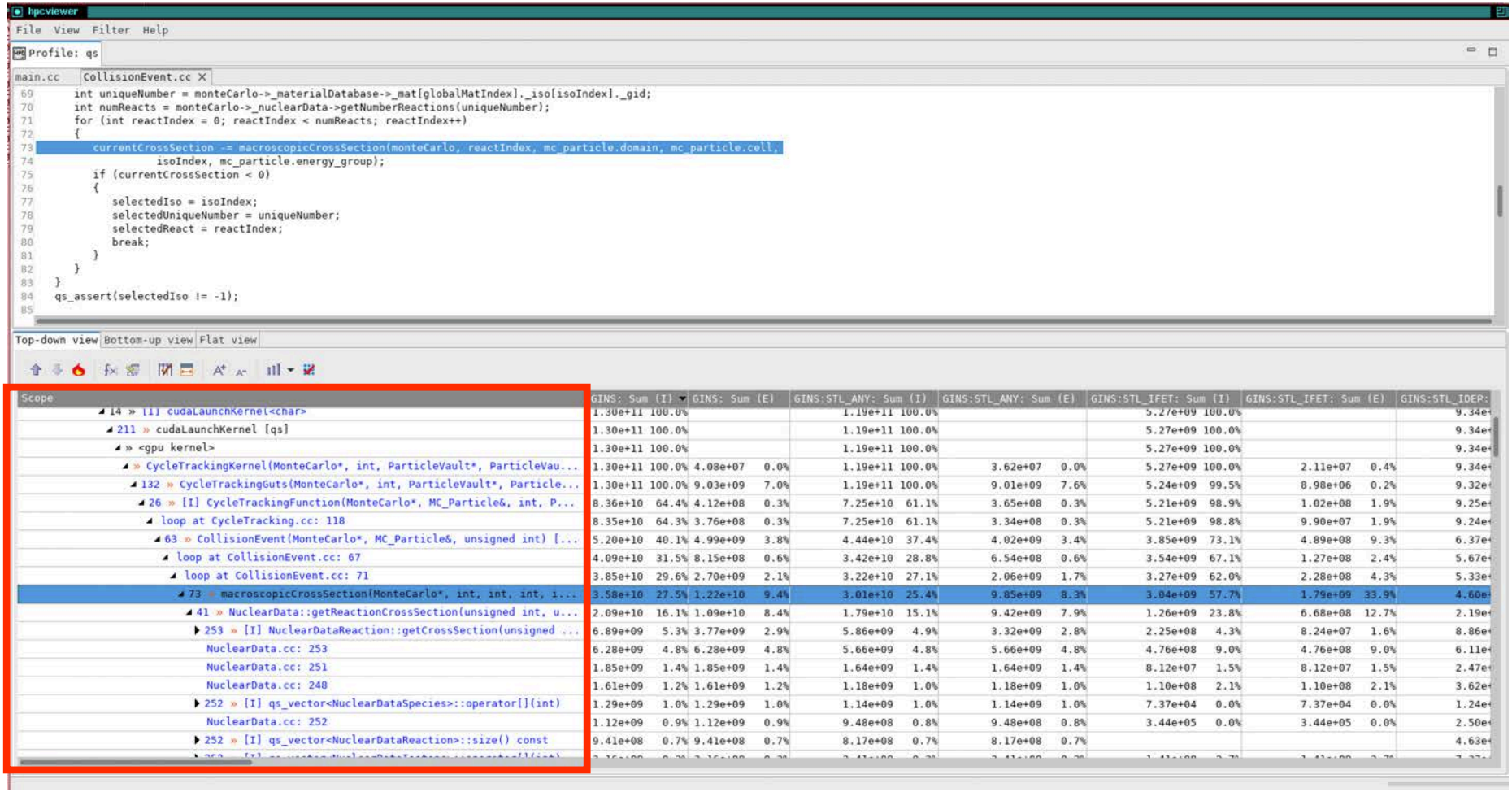
Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter



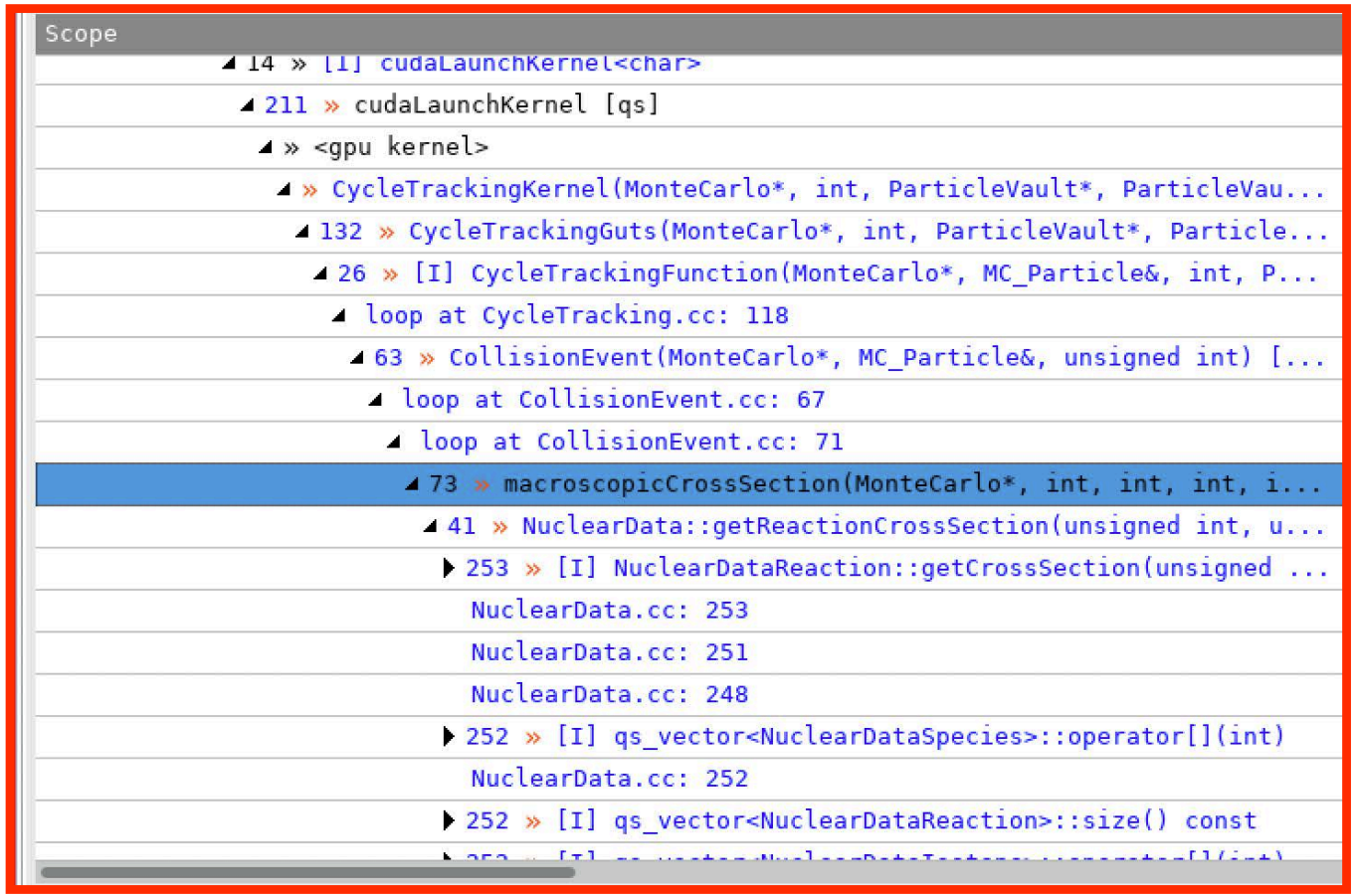
Case Study: Quicksilver

- Proxy application that represents some elements of LLNL's Mercury code
- Solves a simplified dynamic Monte Carlo particle transport problem
  - Attempts to replicate memory access patterns, communication patterns, and branching or divergence of Mercury for problems using multigroup cross sections
- Parallelization: MPI, OpenMP, and CUDA
- Performance Issues
  - load imbalance (for canned example)
  - latency bound table look-ups
  - a highly branchy/divergent code path
  - poor vectorization potential

Quicksilver: Detailed analysis within a Kernel using PC Sampling



Quicksilver: Detailed analysis within a Kernel using PC Sampling



Plan for This Session

- Prepare for hands-on
- Introduce HPCToolkit tools and workflow
  - Measurement (hpcrun)
  - Post-mortem analysis tools (hpcstruct, hpcprof)
  - Graphical user interface (hpcviewer)
- Illustrate hpctoolkit's use with some case studies
- Working with hands-on examples
  - Exploring pre-collected performance databases
  - Full contact

    scripted measurement, analysis, visualization of examples

HPCToolkit Resources

- Documentation
  - —User manual for HPCToolkit: https://hpctoolkit.gitlab.io/hpctoolkit
  - —Cheat sheet: https://gitlab.com/hpctoolkit/hpctoolkit/-/wikis/HPCToolkit-cheat-sheet
  - —User manual for hpcviewer: https://hpctoolkit.gitlab.io/hpctoolkit/users/hpcviewer/hpcviewer.html
  - —Tutorial videos
    - http://hpctoolkit.org/training.html
      - recorded demo of GPU analysis of Quicksilver: https://youtu.be/vixa3hGDuGg
      - recorded tutorial presentation including demo with GPU analysis of GAMESS: https://vimeo.com/781264043
- Software
  - —Download hpcviewer GUI binaries for your laptop, desktop, cluster, or supercomputer
    - OS: Linux, Windows, MacOS
    - Processors: x86_64, aarch64, ppc64le
    - http://hpctoolkit.org/download.html
  - —Install HPCToolkit on your Linux desktop, cluster, or supercomputer using Spack
    - http://hpctoolkit.org/software.html

Hands-on Options

- Pre-collected databases to explore
  - gain experience using hpctoolkit's hpcviewer graphical user interface to analyze performance data
- Hands-on examples
  - scripts to build, run, and view several examples for the full experience
    - hpcrun: measure an application as it executes
    - hpcstruct: recover program structure information for mapping measurements to source code
    - hpcprof: combine measurements with program structure information
    - hpcviewer: explore profiles and traces

Prepare to Explore Performance Data on your Laptop

REVIEW

- Download and install hpcviewer
  - See https://hpctoolkit.org/download.html
    - —Select the right one for your laptop: MacOS (Apple Silicon, Intel), Windows, Linux
    - —If you don't have Java JDK 17 or 21, you can install one easily from Adoptium (https://adoptium.net)
      - —It's a one-click install. Install Java JDK before running hpcviewer
- User manual for hpcviewer

  https://hpctoolkit.gitlab.io/hpctoolkit/users/hpcviewer/hpcviewer.html

Acquire a Copy of HPCToolkit Hands-on Examples for Aurora

Starting from scratch

REVIEW

```
git clone https://github.com/argonne-lcf/ALCF_Hands_on_HPC_Workshop.git
```

Updating your existing directory

```
cd ALCF_Hands_on_HPC_Workshop
git pull
```

Today's examples

```
cd ALCF_Hands_on_HPC_Workshop/tools/hpctoolkit
```

Acquire a Compute Node to use Interactively

REVIEW

Getting a compute node

```
cd ALCF_Hands_on_HPC_Workshop/tools/hpctoolkit
source sourceme-get-compute-node.sh
```

Configuring a compute node's environment for hands-on examples

```
cd ALCF_Hands_on_HPC_Workshop/tools/hpctoolkit
source sourceme-on-compute-node.sh
```

Available Performance Databases

See /flare/alcf_training/hpctoolkit-examples/databases

- quicksilver: Monte Carlo particle transport proxy application **(C++ + CUDA)**
  - hpctoolkit-qs-gpu-cuda.d - profile and trace on 4 CPUs + 4 GPUs
  - hpctoolkit-qs-gpu-cuda-pc.d - instruction-level measurements within kernels using PC sampling
- pelelmex: Adaptive mesh hydrodynamics simulation code for low Mach number reacting flows **(C++ + AMReX)**
  - pelelmex.db - a large trace with load imbalance from 2025 NERSC hackathon run on 16 CPUs + 16 GPU
- gamess: General Atomic and Molecular Electronic Structure System **(Fortran + OpenMP)**
  - 1.singlegroup-unbalanced/hpctoolkit-gamess-1n-chol-noDS.d
  - 2.singlegroup-balanced/hpctoolkit-gamess-1n-chol-fix_load_balance_noDS.d/
  - 3.multigroup-unbalanced-mtarbr/hpctoolkit-gamess-5n.d/
  - 4.multigroup-balanced/hpctoolkit-gamess-5n-manualbalance.d/
  - 5.multigroup-unbalanced-pc/hpctoolkit-gamess-5n-pc.d/
  - 6.scale/hpctoolkit-gamess-22n-test.d/
- qmcpack - CPU and GPU simulations
- minitest **(OpenMP TARGET; SYCL: Profiling+tracing, PC sampling, Binary instrumentation)**

Viewing Performance Data

- Copy a performance database directory to your laptop and open it locally
- Open a performance database on a remote system

Note: using a HPCViewer with a remote system presumes that hpcserver has already been installed on the remote system

- hpcserver has been installed on Aurora
- you can download and install hpcserver on your local cluster as well (ask in Slack for directions)

Configuring Hpcviewer Remote Access

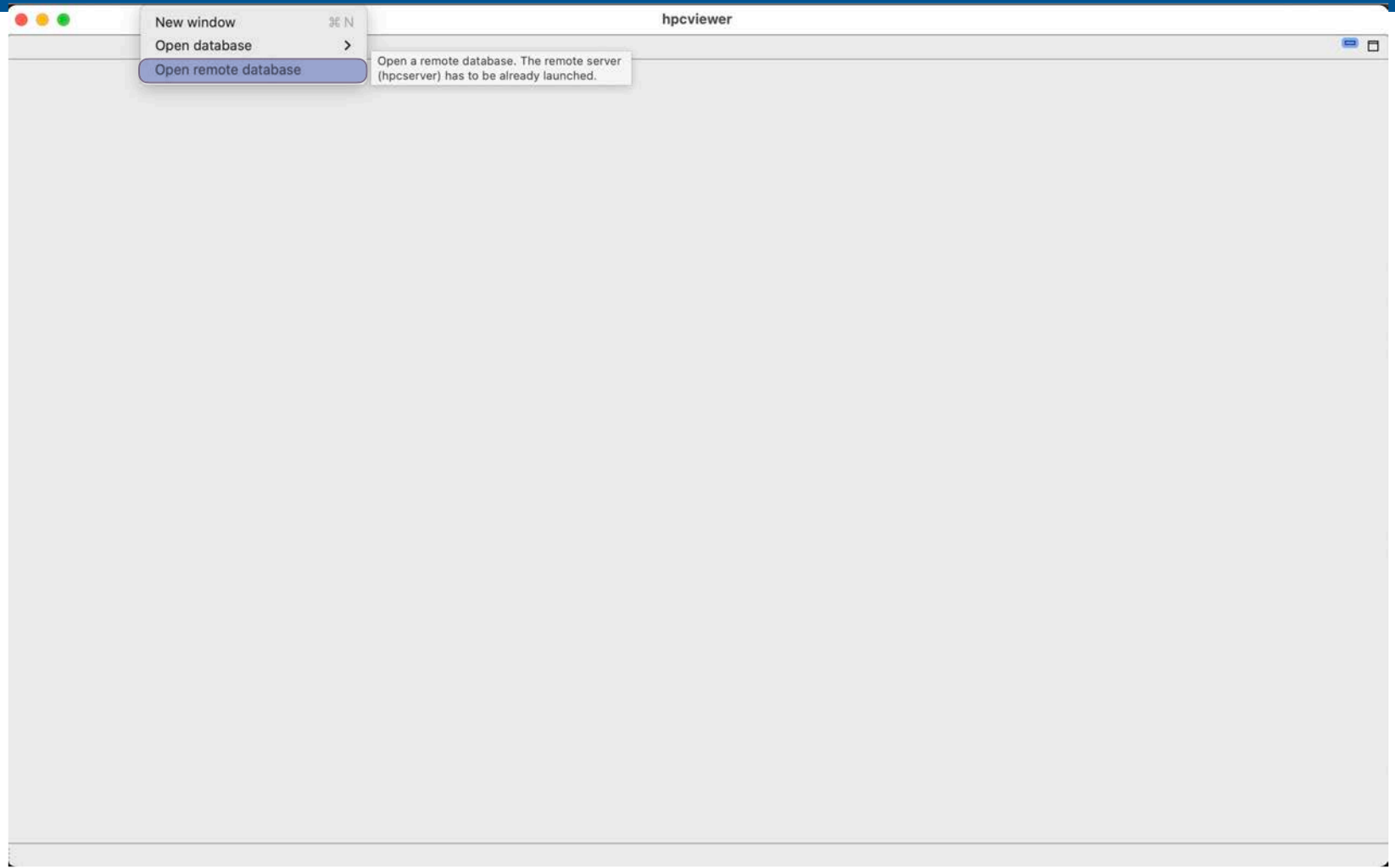
- Run hpcviewer
- From the file menu, select “Open remote database”
- Fill in the hostname/IP address: aurora.alcf.anl.gov
- Fill in your username on Aurora
- Fill in the remote installation directory for hpcviewer’s server:

```
/soft/perftools/hpctoolkit/hpcserver
```

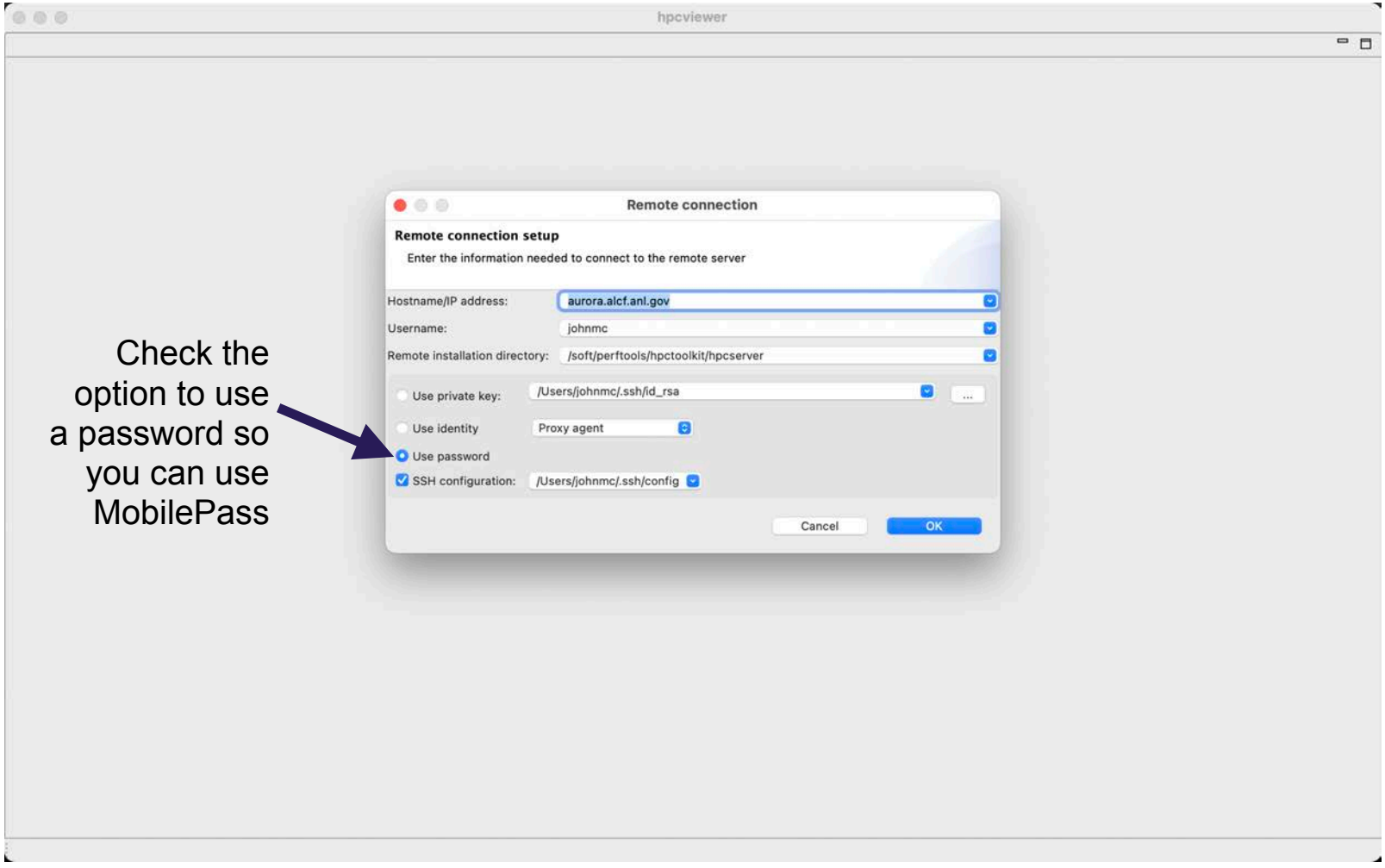
- Select the authentication method: “Use password”
- Click “OK”
- Authenticate using your token as you normally do
- Navigate to a database with the file chooser to

  /flare/ATPESC2025/EXAMPLES/track6-tools/hpctoolkit/data

Opening a Remote Database

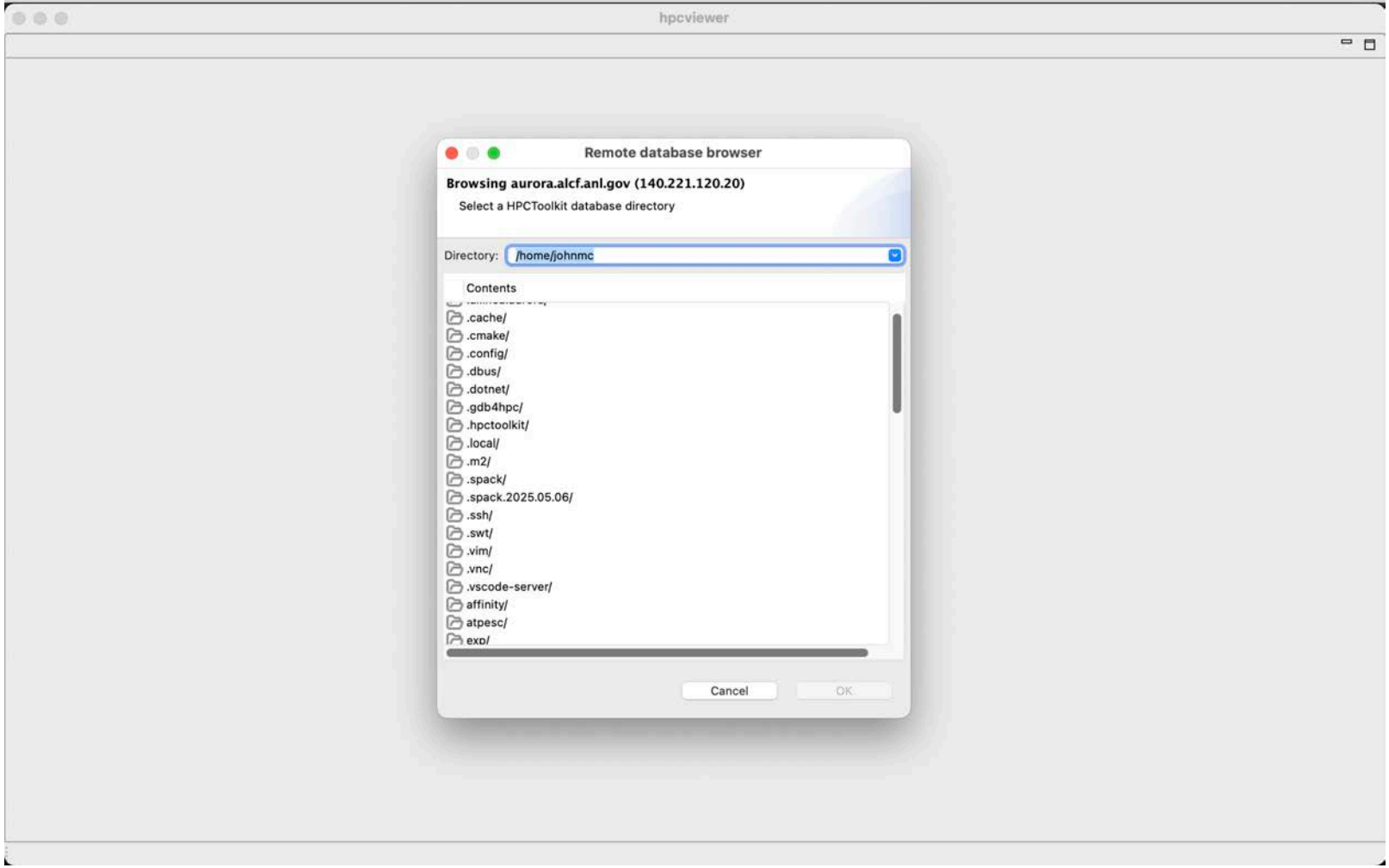


Configuring for remote access to Aurora using hpcserver

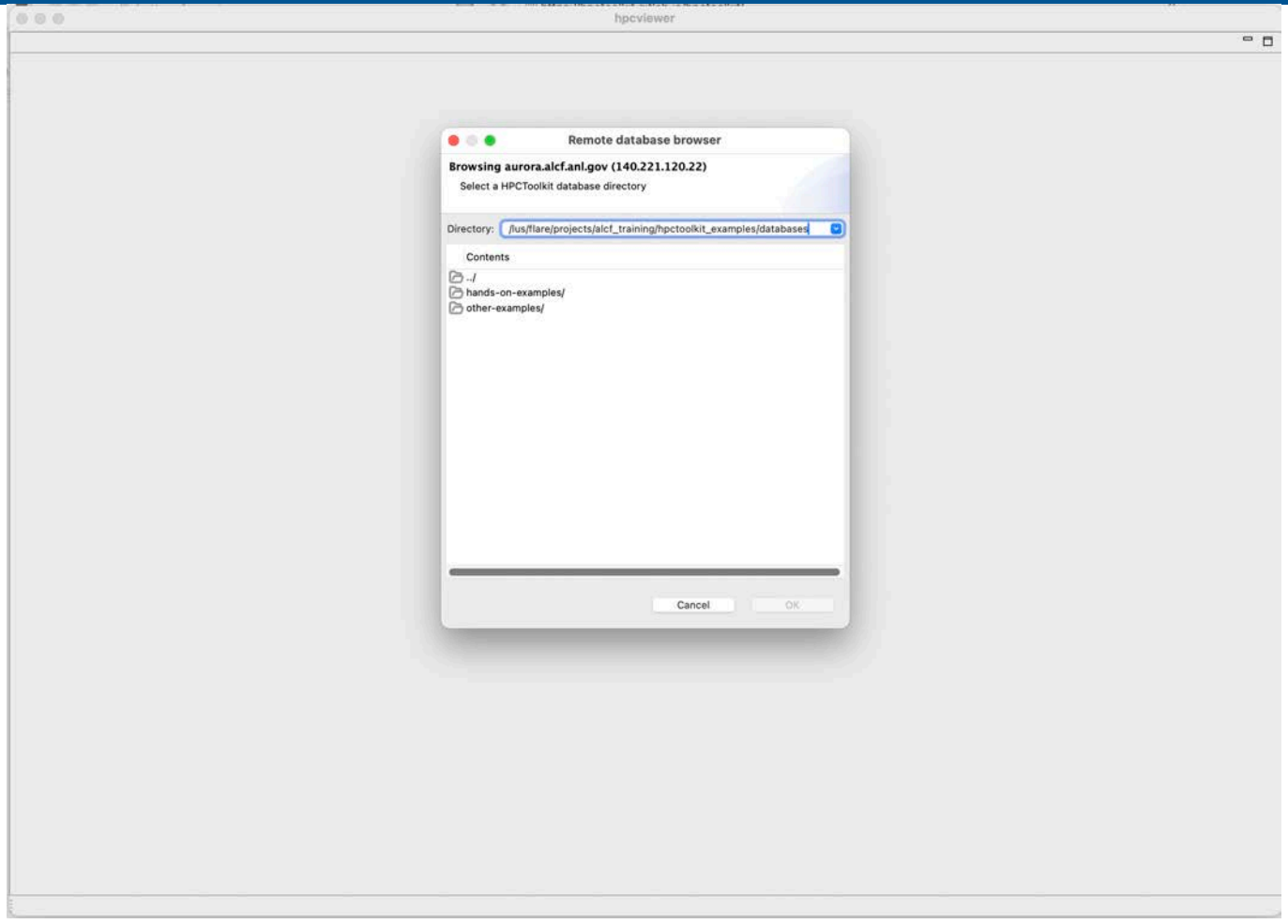


Check the option to use a password so you can use MobilePass

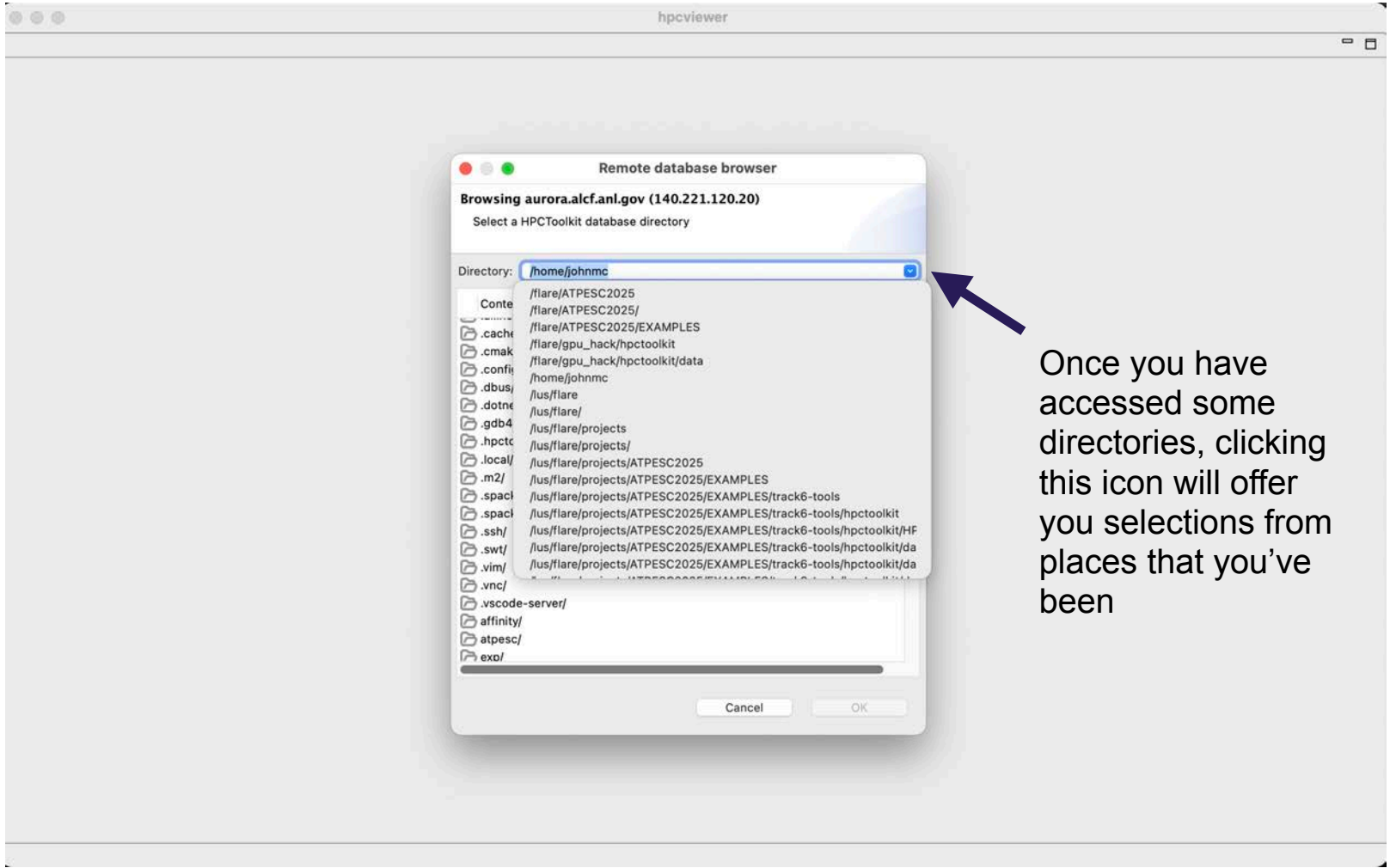
First View of Aurora: Your Home Directory



Navigate to Example Databases

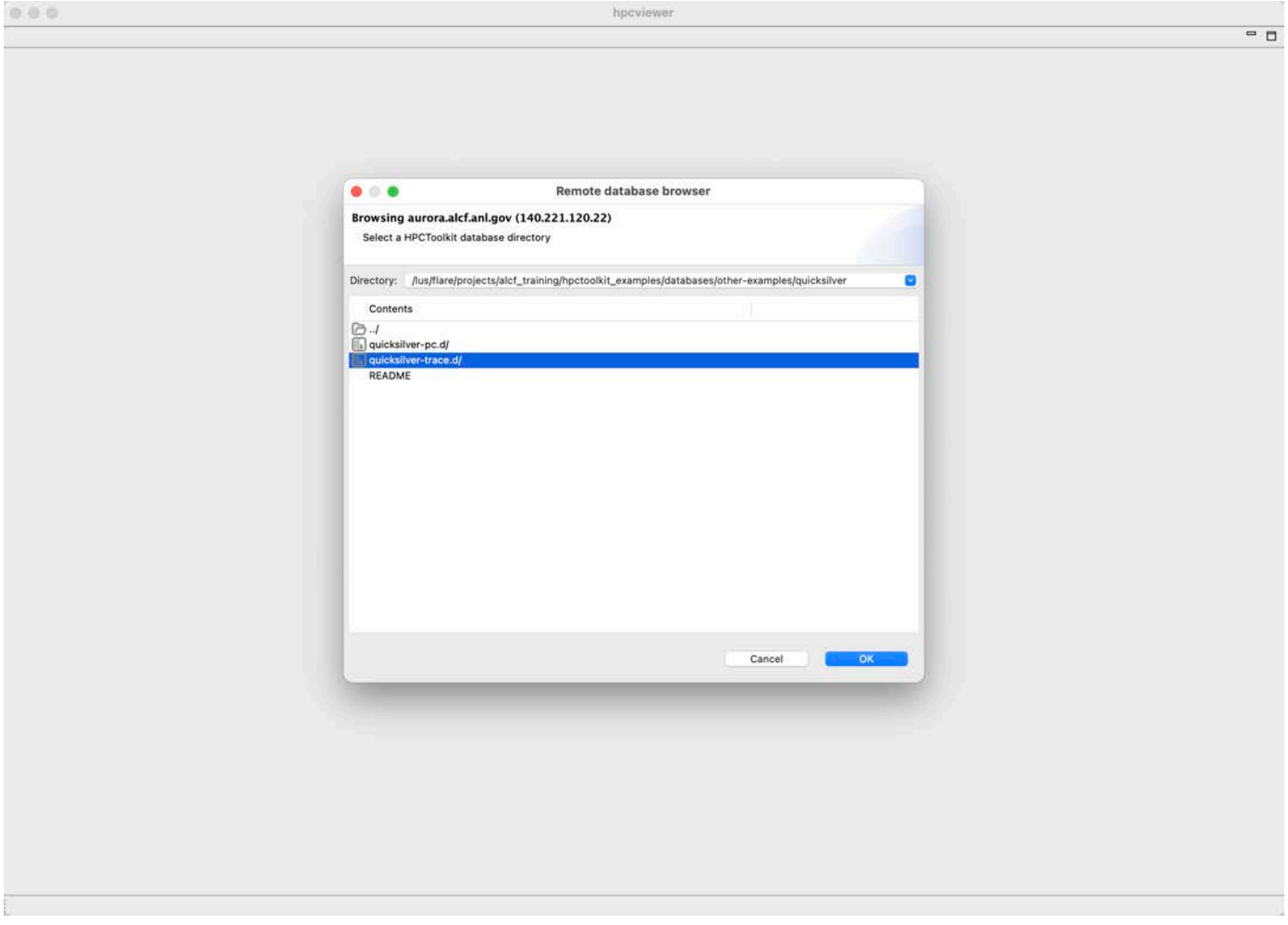


Navigate to Example Databases

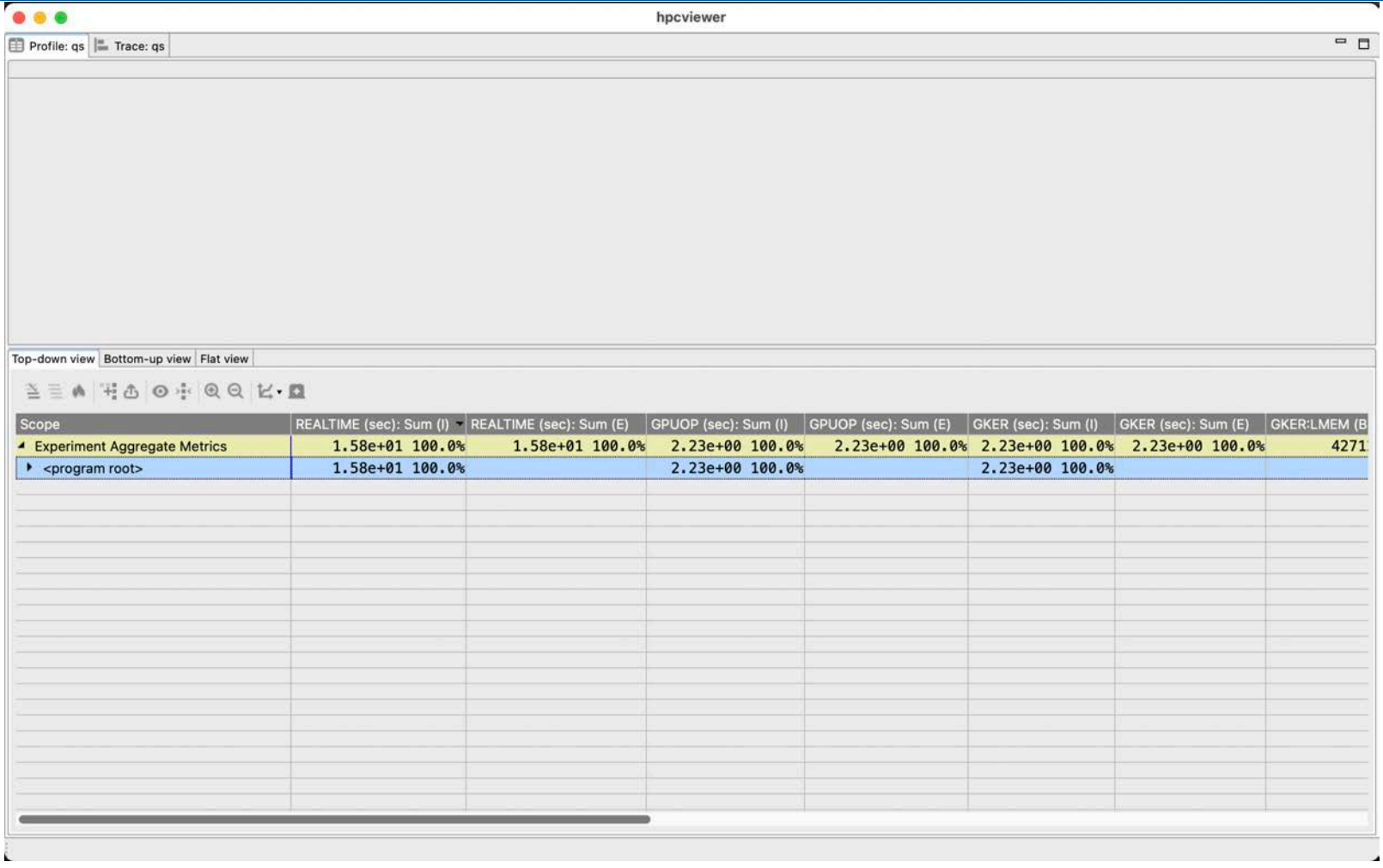


Once you have accessed some directories, clicking this icon will offer you selections from places that you've been

Select a Quicksilver Database with Traces

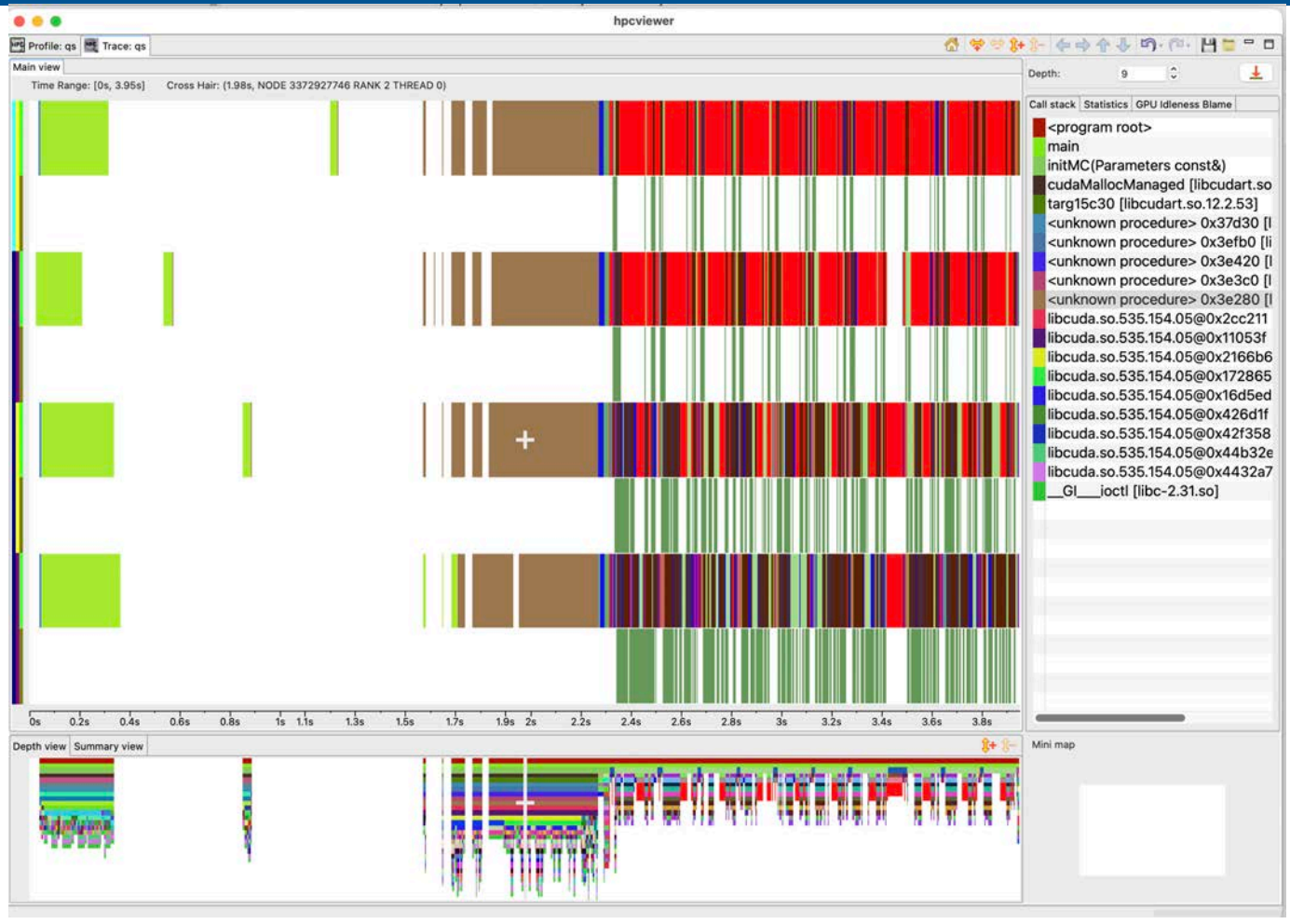


After Selecting quicksilver-trace.d

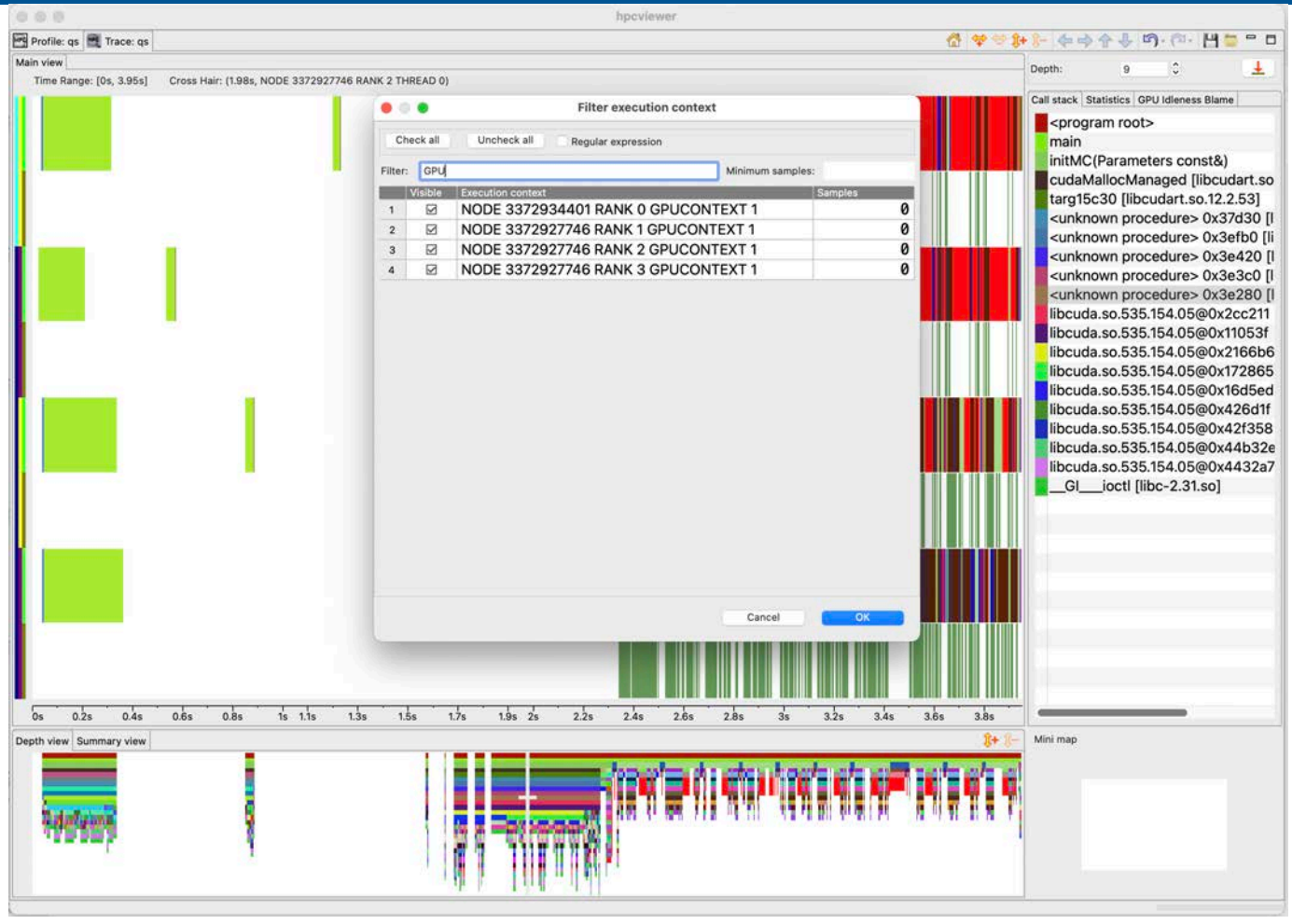


Inspecting Pre-collected Quicksilver Data

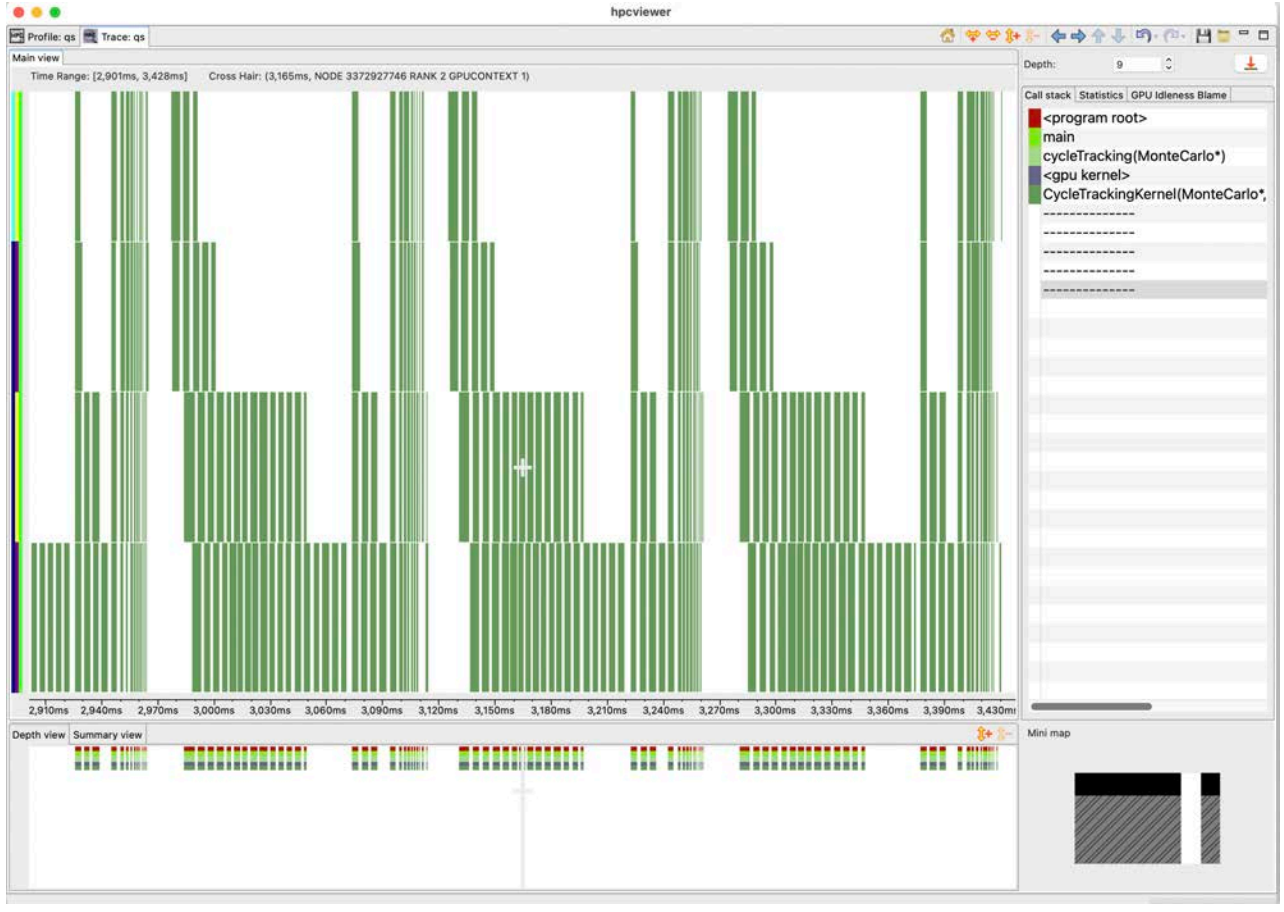
Select the Tab “Trace: qs”



Use the Filter to “Uncheck all” and Check “GPU” streams



Zoom in on GPU Work to See Load Imbalance Across the Four GPUs



Analyzing Quicksilver Traces

Using a measurement database with profiles and traces

- Select the Trace tab “Trace: qs”
- Identifying the traces
  - Select a pixel on a trace line
  - Look at legend on the top of the display, which reports the location of the “cross hair”
  - Is this a CPU or GPU trace line?
  - Repeat this a few times to identify what each of the trace lines represents
- Notice that each time you select a colored pixel on a trace line, you will be shown the function call stack in the rightmost pane
- At the top of the pane is a “depth” indicator, that indicates what level in the call stack you are viewing. The selected level will also be highlighted
- You can change the depth of your view by using the depth up/down, typing a depth, or simply selecting a frame in the call stack at the desired depth
- You can select  above the call stack frame to show the call stacks at the deepest depth
  - If a sample doesn’t have an entry at the selected depth, its deepest frame will be shown

Analyzing Quicksilver Traces

Using a measurement database with profiles and traces

- Zoom in on a region in a trace by selecting it in the trace display
- Use the back button  to undo a zoom
- Use the control buttons  at the top of the trace pane to
  - expand or contract the pane
  - move left, right, up, or down
- Keep an eye on the minimap in the lower right corner of the display to know what part of the trace you are viewing
- Use the home button  to reset the trace view to show the whole trace

Analyzing Quicksilver Traces

Using a measurement database with profiles and traces

- Select the Trace tab “Trace: qs”
- Configure filtering
  - Use the Filter menu to select Filter Execution Contexts
  - In the filtering menu, select "Uncheck all"
  - Now, in the empty box preceded by "Filter:", type "GPU" and then click "Check all”
  - Select "OK".
  - Now, the Trace View will show only trace lines for the GPUs.
- Inspect the trace data
  - Is the work load balanced across the GPUs? How can you tell?
  - Bring up the filter menu again. Select "Uncheck all". Type in "RANK 3" in the Filter box. Select thread 0 and the GPU context. Select “OK”.
  - Move the call stack to depth 2
    - What CPU function is Rank 3 thread 0 executing when the GPU is idle?
    - Does this suggest any optimization opportunities?

Analyzing the Quicksilver Summary Profile

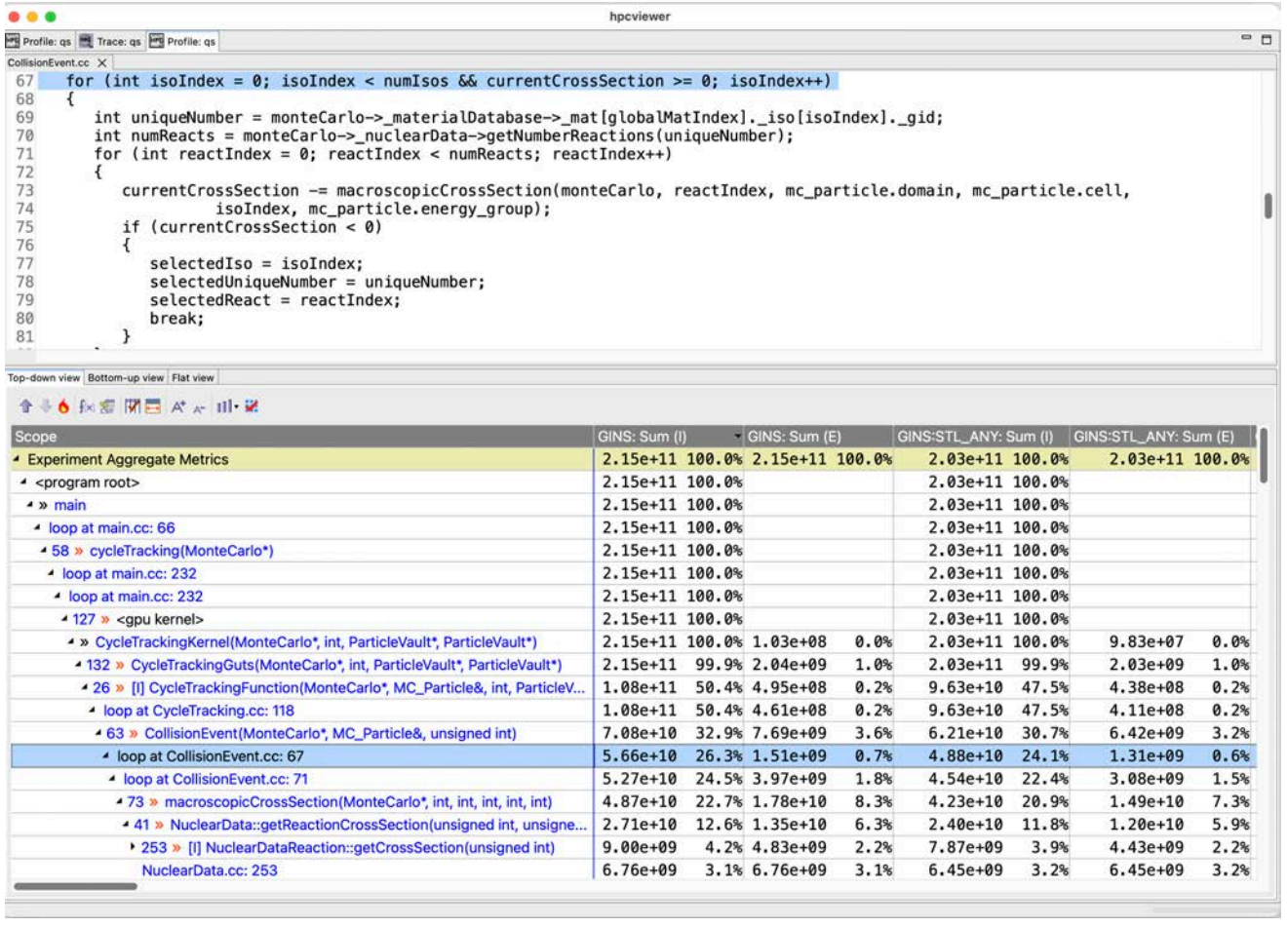
Using a measurement database with profiles and traces

- Select the Profile Tab “Profile: qs”
- Use the column selector  to deselect and hide the two REALTIME columns
- Select the GPU OPS column, which represents time spent in all GPU operations
- Select the  button to show the “hot path” according to the selected column
  - the hot path of parent will continue into a child as long as the child accounts for 50% or more of the parent’s cost
- The hot path will select “CycleTrackingKernel” — a GPU kernel that consumes 100% of the GPU cost in this profile
- Use the  button to graph “GPU OPS (I)” — inclusive GPU operations across the profiles
  - Are the GPU operations balanced or not across the execution contexts (ranks)?

Analyzing the Quicksilver Summary Profile

- You will notice that for quicksilver, HPCToolkit doesn't report any data copies between the host and device
  - The quicksilver code uses "unified memory" so that all of the data movement occurs between CPU and GPU using page faults rather than explicit copies
  - Today's GPU hardware doesn't support attribution of page faults to individual instructions
    - We could profile them (and do on AMD GPUs in a forthcoming release), but the GPUs lack support to attribute them to the code that triggered the faults

The Profile View in the other “PC Sampling” Database



Analyzing Quicksilver PC Samples

Using a measurement database with traces that was collected *with* PC sampling enabled

Using the default top-down view of the profile

- Select the column “GINS (I)” to focus on the measurement of inclusive GPU Instructions
- Select use the flame button to look at where the instructions are executed
- In the call stack revealed, you will <gpu kernel> placeholder that separates CPU activity (above) from GPU kernel activity (below)
- Below the <gpu kernel> placeholder you will see the function calls, inlined functions, loops and statements in HPCToolkit’s reconstruction of calling contexts within the CycleTrackingKernel

- Using the bottom-up view of the profile
  - Select the bottom-up tab of above the control pane
  - Select the GINS STL_ANY (E) column, which will sort the functions by the exclusive GPU instruction stalls within that function
  - Scroll right to see which of the types of contributing types of stalls accounts for most of the STL_ANY amount
  - Select the function that has the most exclusive stalls
  - Select the the hot path to see where this function is called from.
    - Where do the calls to the costly function come from?
    - Does there appear to be an opportunity to reduce the number of calls to this function?

Exercise the whole workflow: Measurement, Post-mortem analysis, Interactive exploration

Hands-on Tutorial Examples on Aurora - 2

hpcstruct - hpctoolkit's multithreaded binary analysis tool

qmcpack - a quantum monte-carlo materials simulation code from the exascale computing project. A big, pre-built code that offloads to the GPU using OpenMP

minitest.omp - a simple MPI + OpenMP offloading code

minitest.sycl - a simple MPI + SYCL offloading code

minitest.sycl.gtpin - use Intel's GTPin binary instrumentation tool to collect dynamic instruction counts for GPU instructions and map them back to source code

minitest.sycl.pc - use hardware support to sample GPU instructions during execution. Map samples and stall reasons back to the source code.

Hands-on Tutorial Examples on Aurora - 1

Go to an example directory, e.g. MINITEST/minitest.sycl

1. make run # use hpcrun to measure, hpsctruct for binary analysis, hpcprof to integrate
2. (a) make view # launch hpcviewer on Aurora to examine the resulting database using X11
   (b) see the next slide for how to view performance data directly on your laptop

Some hpcviewer tips

Information for Using Hpcviewer

- Filtering GPU traces
  - Can use the filter menu to select what execution traces you want to see
    - cpu only, gpu, a mix
    - type a string or a regular expression in the chooser select or unselect the new set
    - only traces that exceed a minimum number of samples
- Filtering GPU calling context tree nodes to hide clutter
  - hide individual CCT nodes: e.g. lines that have no source code mapping library@0x0f450
  - hide subtrees: MPI implementation, implementation of CUDA primitives
- When inspecting GPU activity, be aware that hpcviewer has two modes
  - expose GPU traces or not
    - means: when displaying GPU trace lines, don't just show GPU activity if the time in the middle of a pixel is in a GPU operation. instead, show the first (if any) GPU operation between the time in the middle of the pixel and the middle of the next pixel
    - why? GPU activity is so short, it may be hard to find if we don't "expose" where it is
    - downside: makes the GPU appear more active than it is
      - you can correct hpcviewer's trace-pane statistics by turning off the "Expose GPU traces" mode
  - mode can be selected from <File>:<Preferences>:<Traces>

Filtering Tips to Hide Unwanted Implementation Details

- Filter “descendants-only” of CCT nodes with names *MPI* to hide the details of MPI implementation in profiles and traces
- Filter internal details of RAJA and SYCL templates to suppress unwanted detail using a “self-only” filter

Troubleshooting tips

On Linux, hpcviewer crashes at startup!

State saved by different versions of hpcviewer is unfortunately incompatible on Linux. Namely, state saved by hpcviewer/2025.1 and hpcviewer/2025.2 is incompatible.

Typically, removing hpcviewer's saved state, as shown below

```
rm -rf $HOME/.hpctoolkit/hpcviewer
```

will fix the problem and allow you to launch the version of hpcviewer you are trying to use.

Note: when running hpcviewer from Aurora, you must be logged in using "ssh -X" and have a value for the environment variable DISPLAY that indicates a valid X11 display.

Why can’t I see Source Code in hpcviewer?

To relate performance measurements in detail to your application source code, your code must be compiled with a “-g” option in addition to your preferred optimization flags. Otherwise, the compiler doesn’t record the information that tools need to map performance to anything finer grain than procedures

- For instance, if you are building with cmake, you will want to build **RelWithDebInfo** rather than **Release** for detailed correlation with source code

I got the following WARNING from hpcprof on Aurora

WARNING: Trace for a thread is unexpectedly extremely unordered, falling back to an in-memory sort.

This may indicate an issue during measurement, and WILL significantly increase memory usage!

Affected thread: NODE(BOTH){1930040613, 0} RANK(SINGLE){3} GPUCONTEXT(SINGLE){0} GPUSTREAM(SINGLE){0}

It appears that Level Zero time stamps wrap rather than counting monotonically. This can make your traces look surprising where GPU activity is rendered a few minutes behind the rest of application activity.

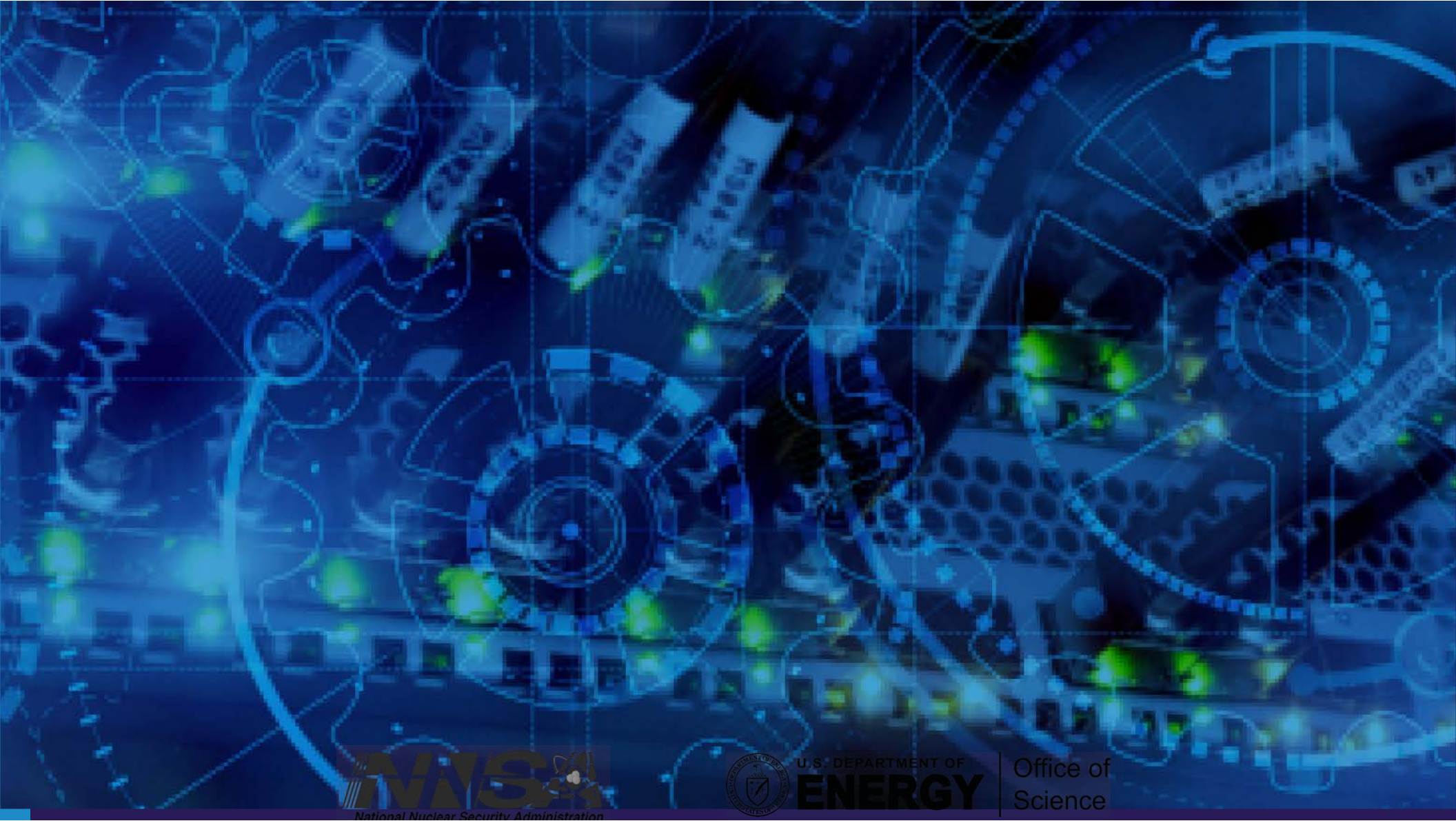
For short executions, you can rerun your application and the problem may disappear.

(Next slide compares a disordered trace and a correct trace)

Disordered vs. Correct Trace for qmcpack

Disordered (some or all GPU timestamps shifted left)

Proper (correct) trace alignment between CPU & GPU


National Nuclear Security Administration
U.S. DEPARTMENT OF
ENERGY
Office of
Science

Plan B Hands-on Examples for Aurora

Starting from scratch

```
git clone https://github.com/jmellorcrummey/ALCF_Hands_on_HPC_Workshop.git
```

Updating your existing directory

```
cd ALCF_Hands_on_HPC_Workshop
git pull
```

Today's examples

```
cd ALCF_Hands_on_HPC_Workshop/tools/hpctoolkit
```