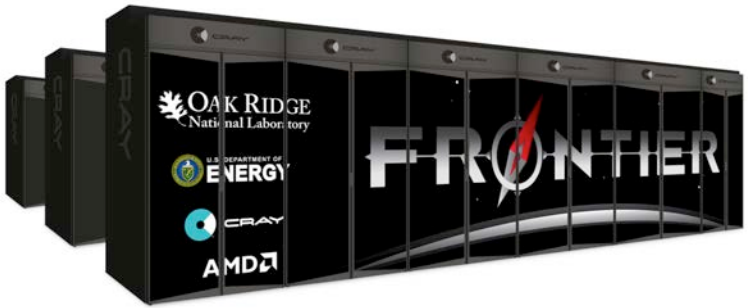
A Community-Driven Portable Profiler at Scale

HPCToolkit

John Mellor-Crummey
Rice University

DOE's GPU-Accelerated Exascale Platforms



- Frontier compute nodes (OLCF)
  - 1 AMD EPYC "Trento" CPU
  - 4 AMD MI250X GPUs
  - 4 Slingshot 11 endpoints
  - Unified memory architecture

- Aurora compute nodes (ALCF)
  - 2 Intel Xeon "Sapphire Rapids" processors
  - 6 Intel Data Center GPU Max 1500
  - 8 Slingshot 11 endpoints
  - Unified memory architecture

- El Capitan compute nodes (LLNL)
  - 4 AMD MI300A APU
  - 4 Slingshot 11 endpoints
  - Unified memory architecture

Outline

- Introduction to HPCToolkit performance tools
  - Overview of HPCToolkit components and their workflow
  - HPCToolkit's graphical user interfaces
- Analyzing the performance of GPU-accelerated codes with HPCToolkit
  - Slides: Exawind (AMReX)
  - Slides: LAMMPS at Exascale (Kokkos)
  - Demo: GAMESS (OpenMP)
- Hands-on after lunch
  - Explore available performance databases for a set of applications
  - Collect and explore some measurements for some prepared 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?

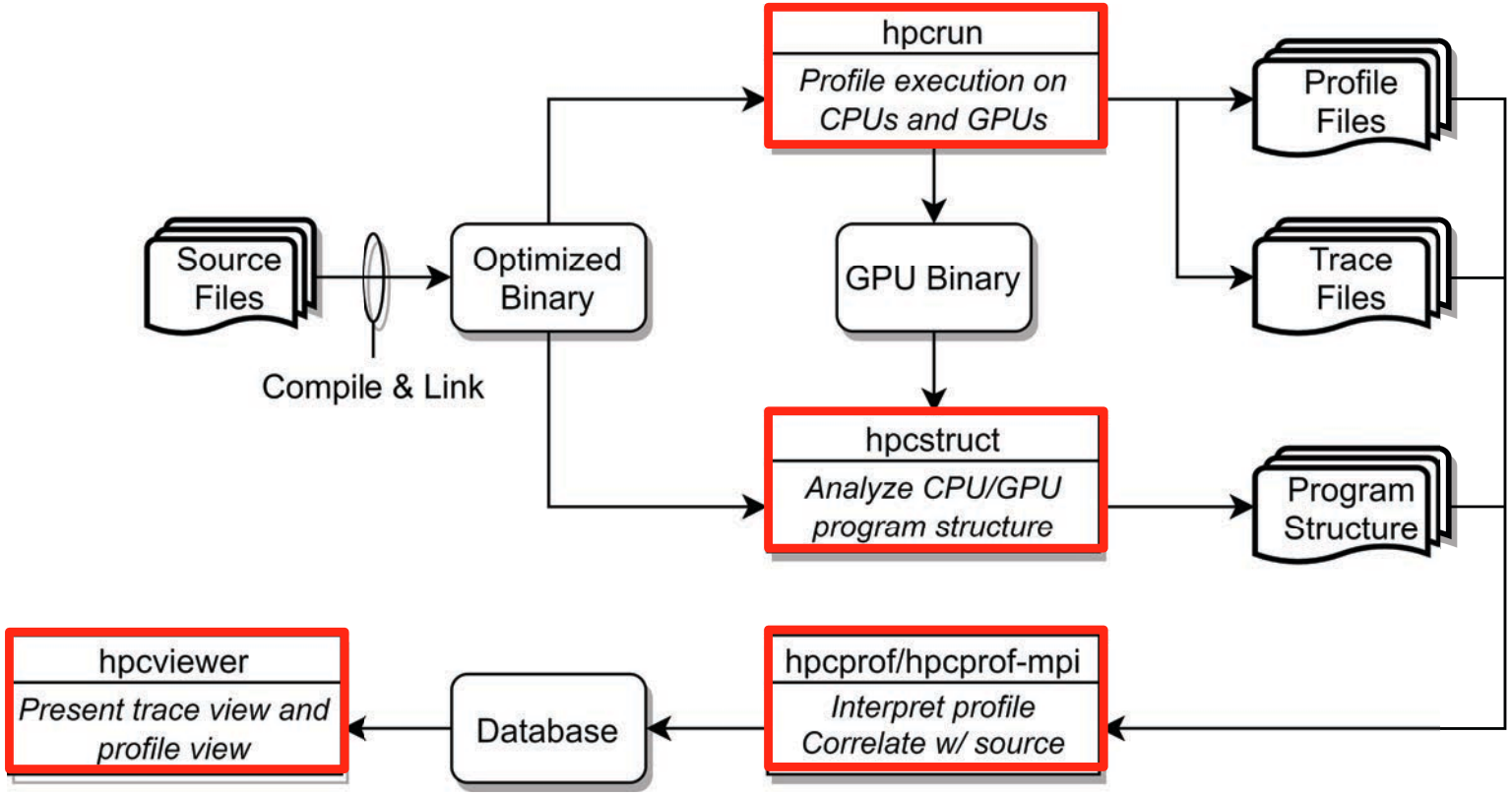
- Measure and analyze performance of CPU and GPU-accelerated applications
- Easy: profile unmodified application binaries
- Fast: low-overhead measurement
- Informative: understand where an application spends its time and why
  - —call path profiles associate metrics with application source code contexts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

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

How does HPCToolkit Differ from Vendor Tools?

- Intel VTune: designed for analysis of performance on a single node
- NVIDIA Nsight Systems and AMD ROCm Systems
  - —tracing of CPU and GPU streams
  - —AMD visualizes directly with Perfetto
  - —Nsight systems supports parallel analysis of measurement data with Dask
    - analyzing on more than one node requires setting up your own Dask cluster
- NVIDIA Nsight Compute and AMD ROCm Compute
  - —detailed measurement of kernels with counters and execution replay
  - —very slow measurement
  - —flat display of measurements within GPU kernels
- HPCToolkit
  - —supports more scalable tracing than vendor tools
    - measure exascale executions across many GPUs and nodes
  - —highly scalable, parallel post-mortem analysis
  - —detailed reconstruction of estimates for calling context profiles within GPU kernels

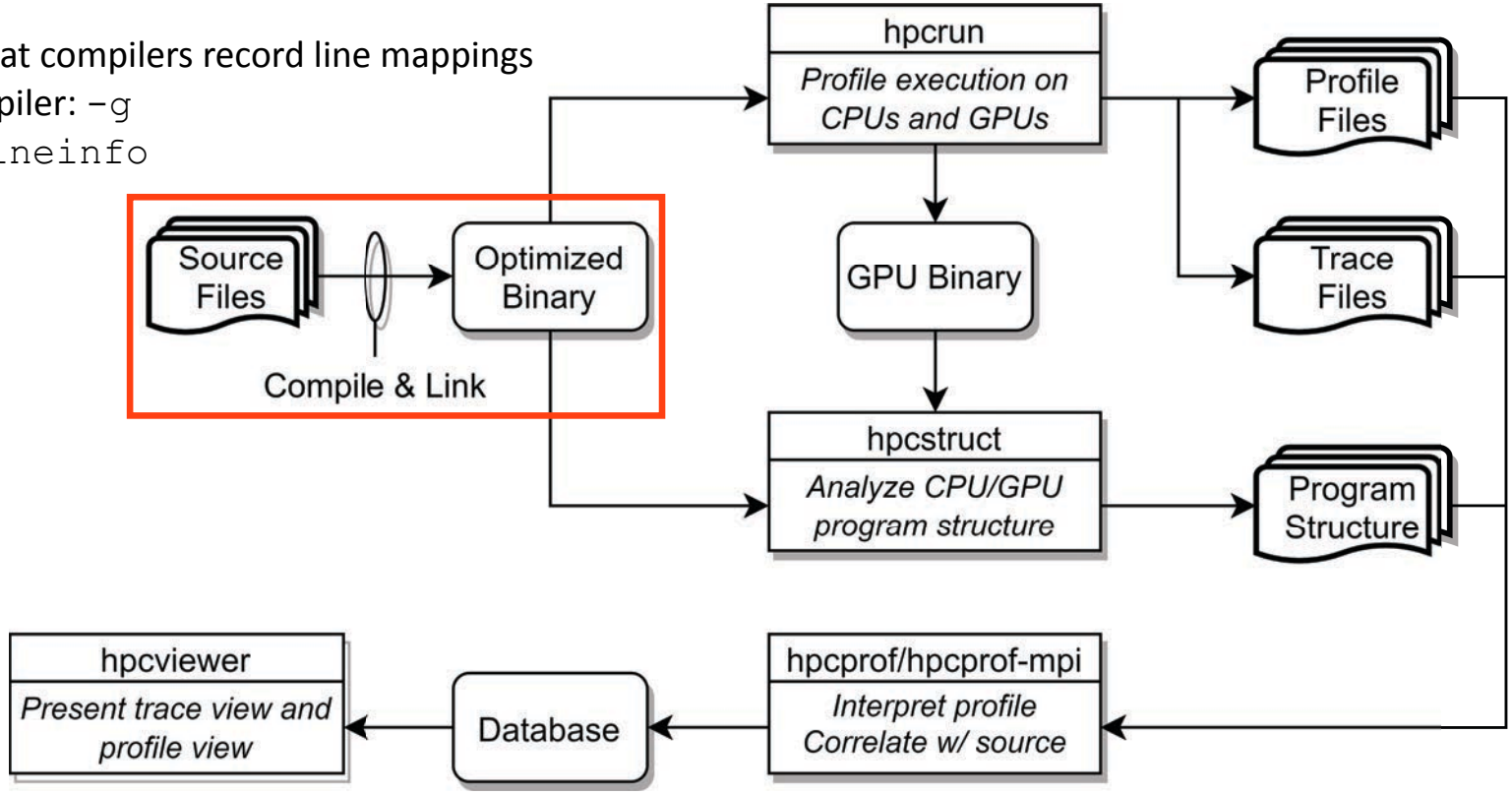
HPCToolkit's Workflow for GPU-accelerated Applications



HPCToolkit's Workflow for GPU-accelerated Applications

Step 1:

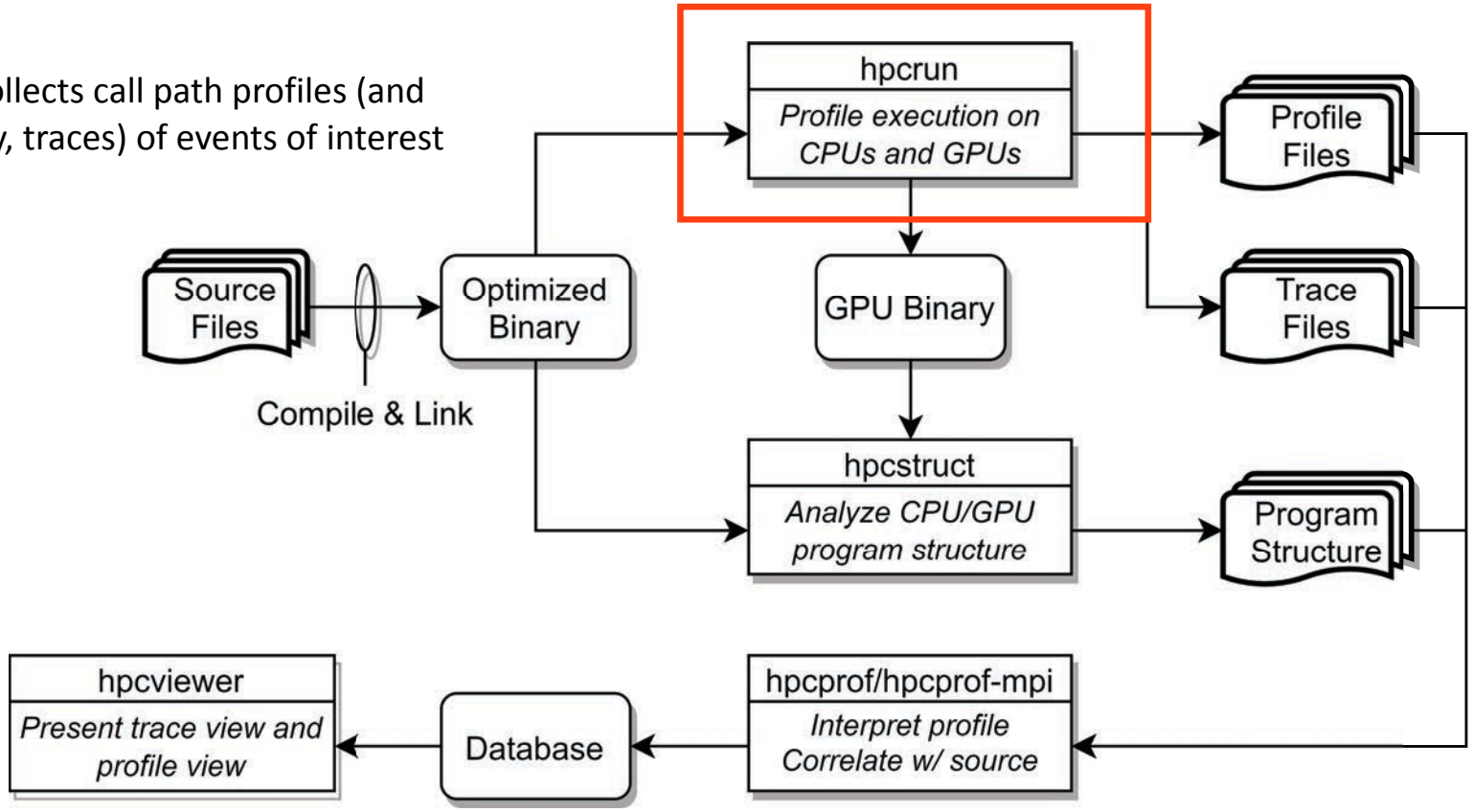
- Ensure that compilers record line mappings
- host compiler: `-g`
- nvcc: `-lineinfo`



HPCToolkit's Workflow for GPU-accelerated Applications

Step 2:

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



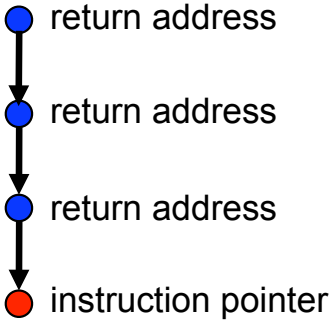
Measurement of CPU and GPU-accelerated Applications

- Sampling using Linux timers and hardware counter overflows on the CPU
- Callbacks when GPU operations are launched and (sometimes) completed
- Event stream for GPU operations
- Program Counter (PC) samples within kernels on AMD, Intel, and NVIDIA GPUs
- Binary instrumentation of GPU kernels on Intel GPUs for instruction-level measurements

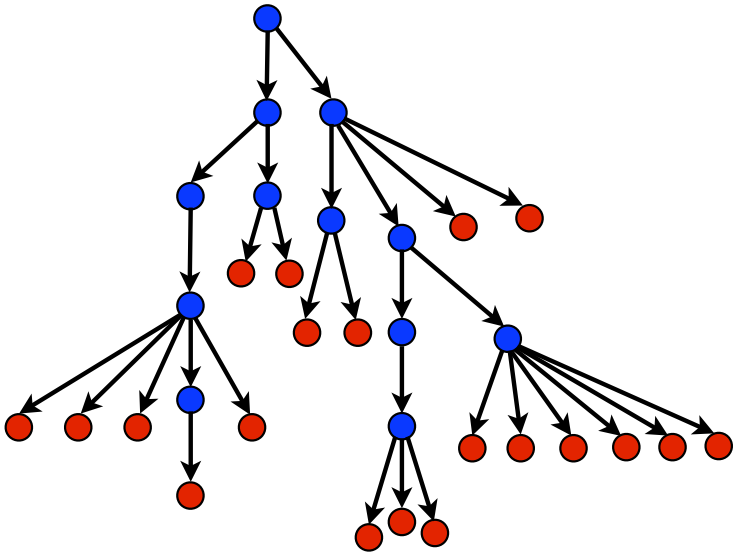
Call Stack Unwinding to Attribute Costs in Context

- 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>`

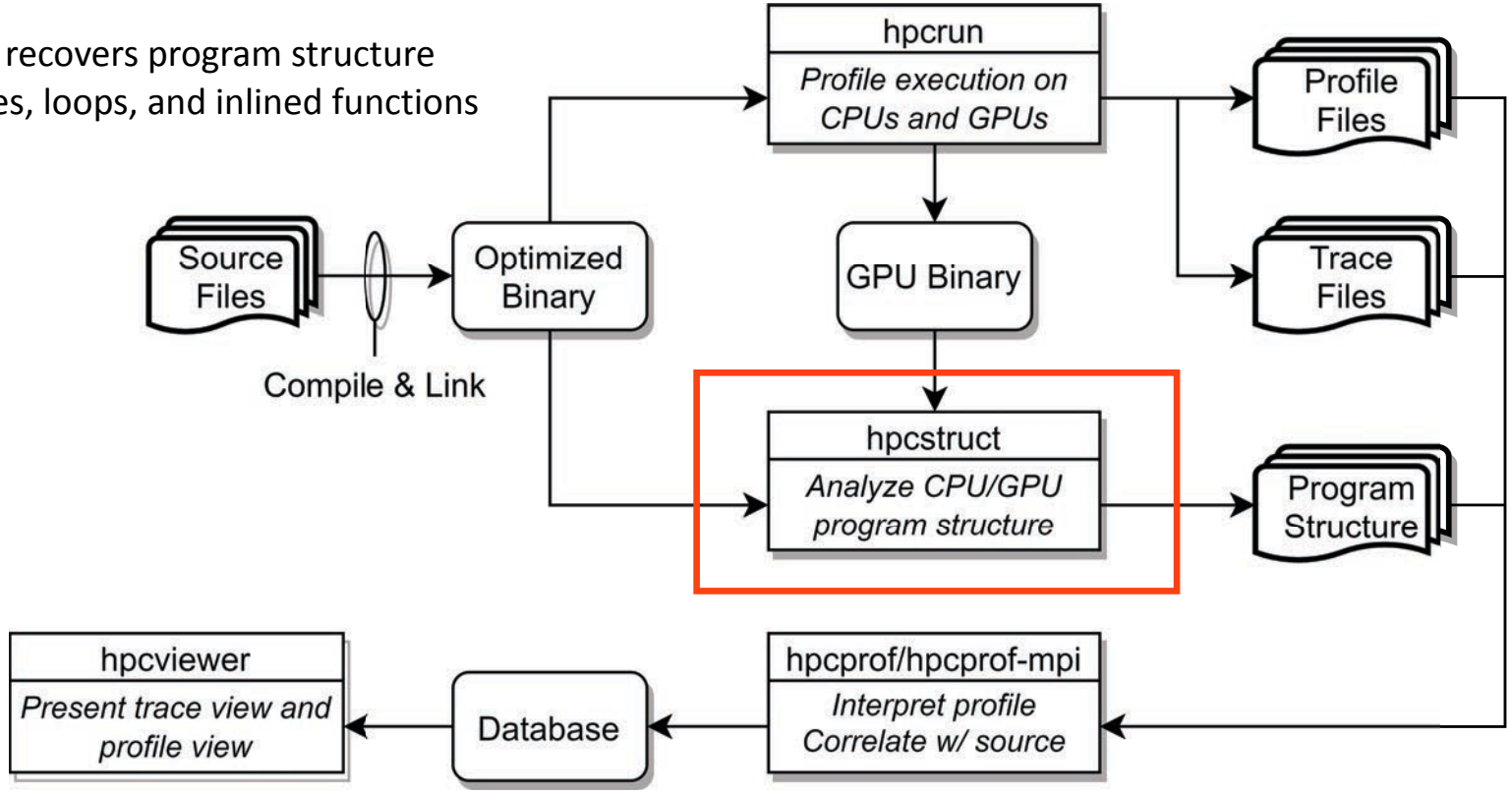
xxx ∈ *{cuda,rocm,opencl,level0}*

yyy ∈ *{cuda,rocm,level0}*

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>
```

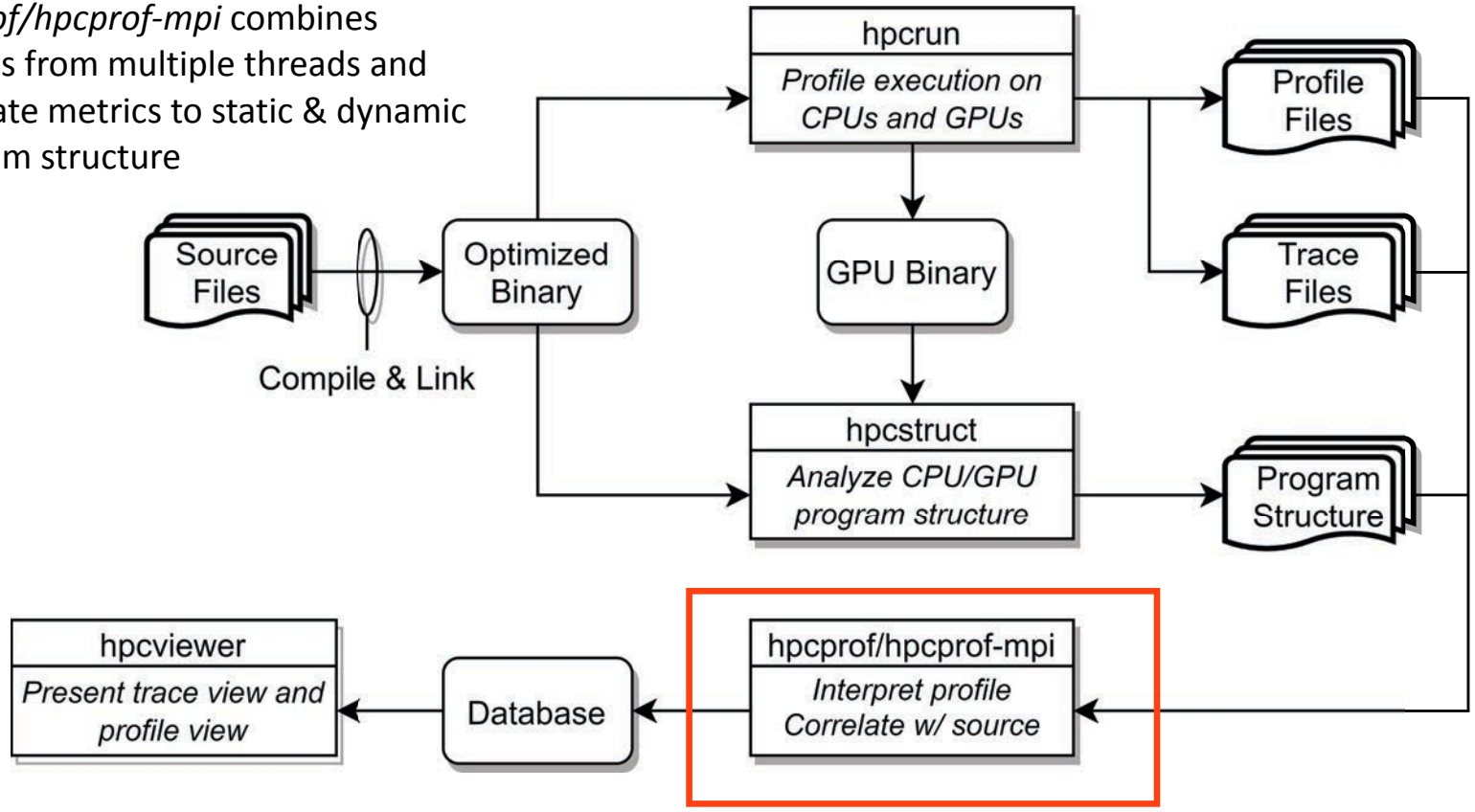
- What it does
  - Recover program structure information
    - Files, functions, inlined templates or functions, loops, source lines
  - In parallel, analyze all CPU and GPU binaries that were measured by HPCToolkit
    —analyze each binary with threads proportional to size of program text and symbol table
  - Cache binary analysis results for reuse when analyzing other executions

NOTE: `--gpucfg yes` needed only for analysis of GPU binaries for interpreting PC samples

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>
```

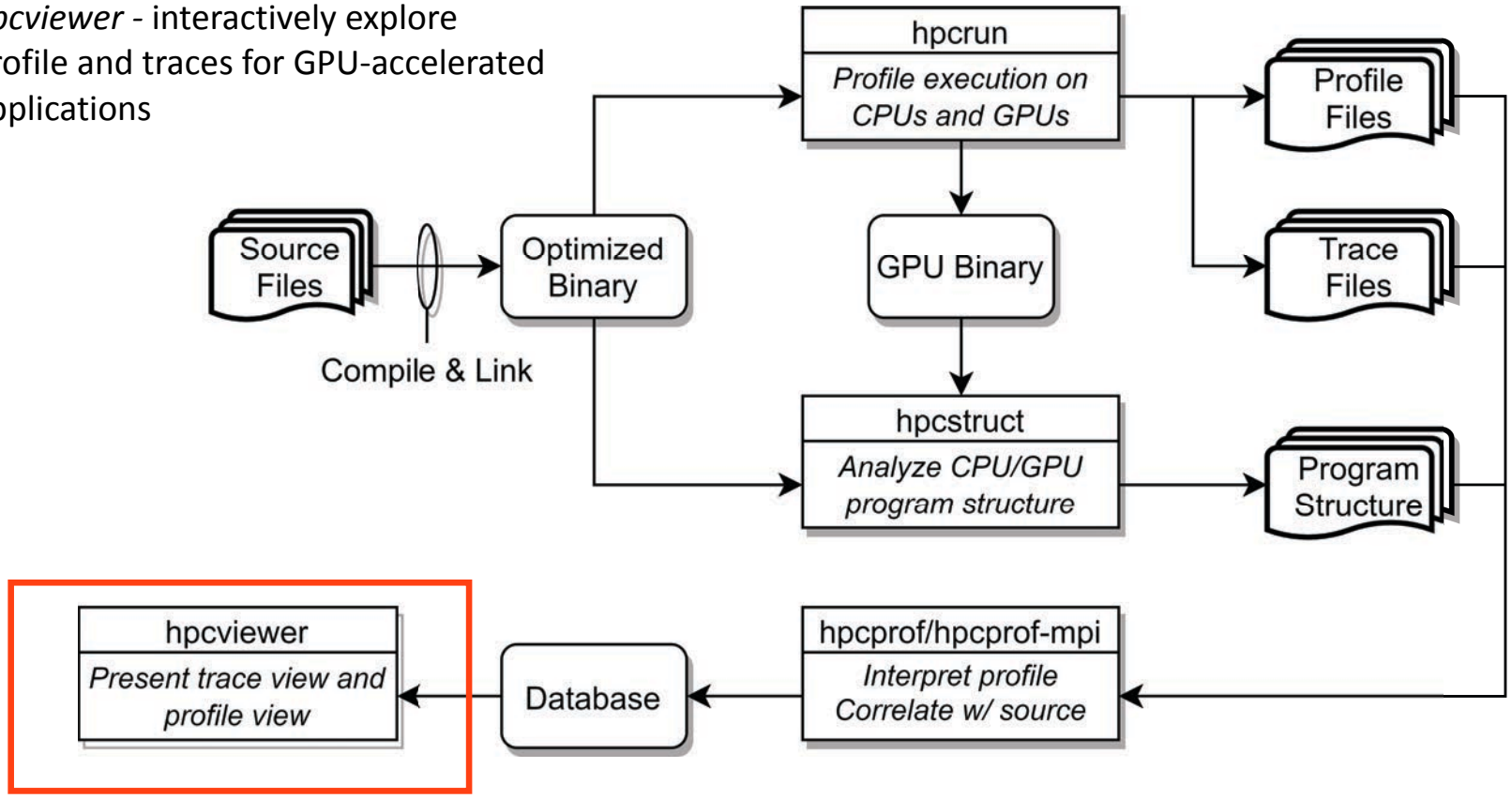
- Analyze data from large executions with distributed-memory parallelism + multithreading (large scale)

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

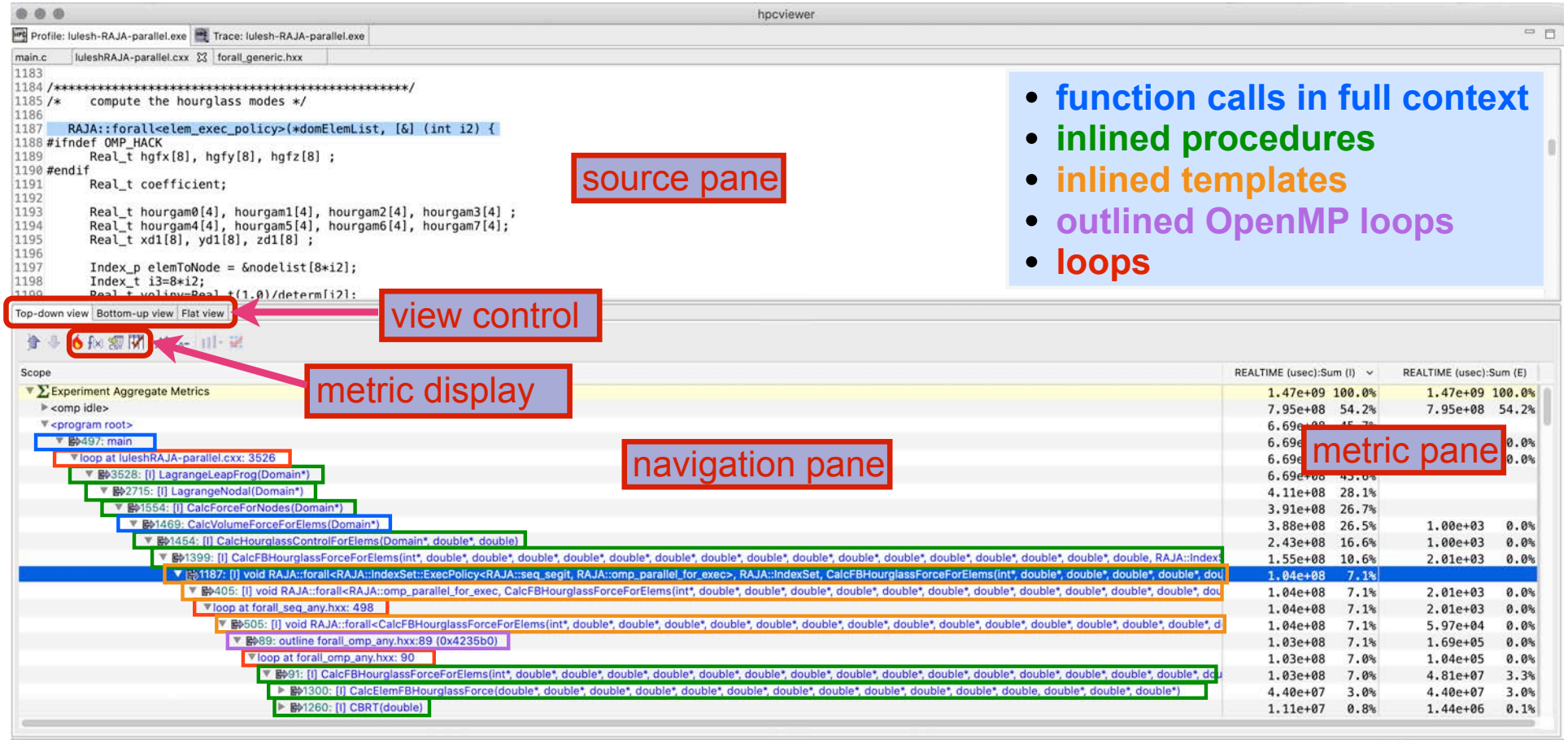
HPCToolkit's Workflow for GPU-accelerated Applications

Step 4:

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

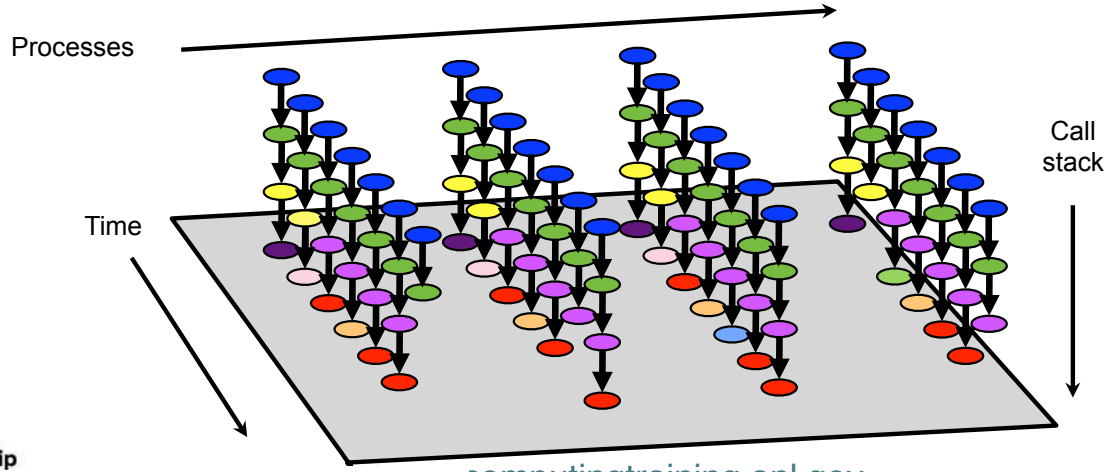


Code-centric Analysis with hpcviewer

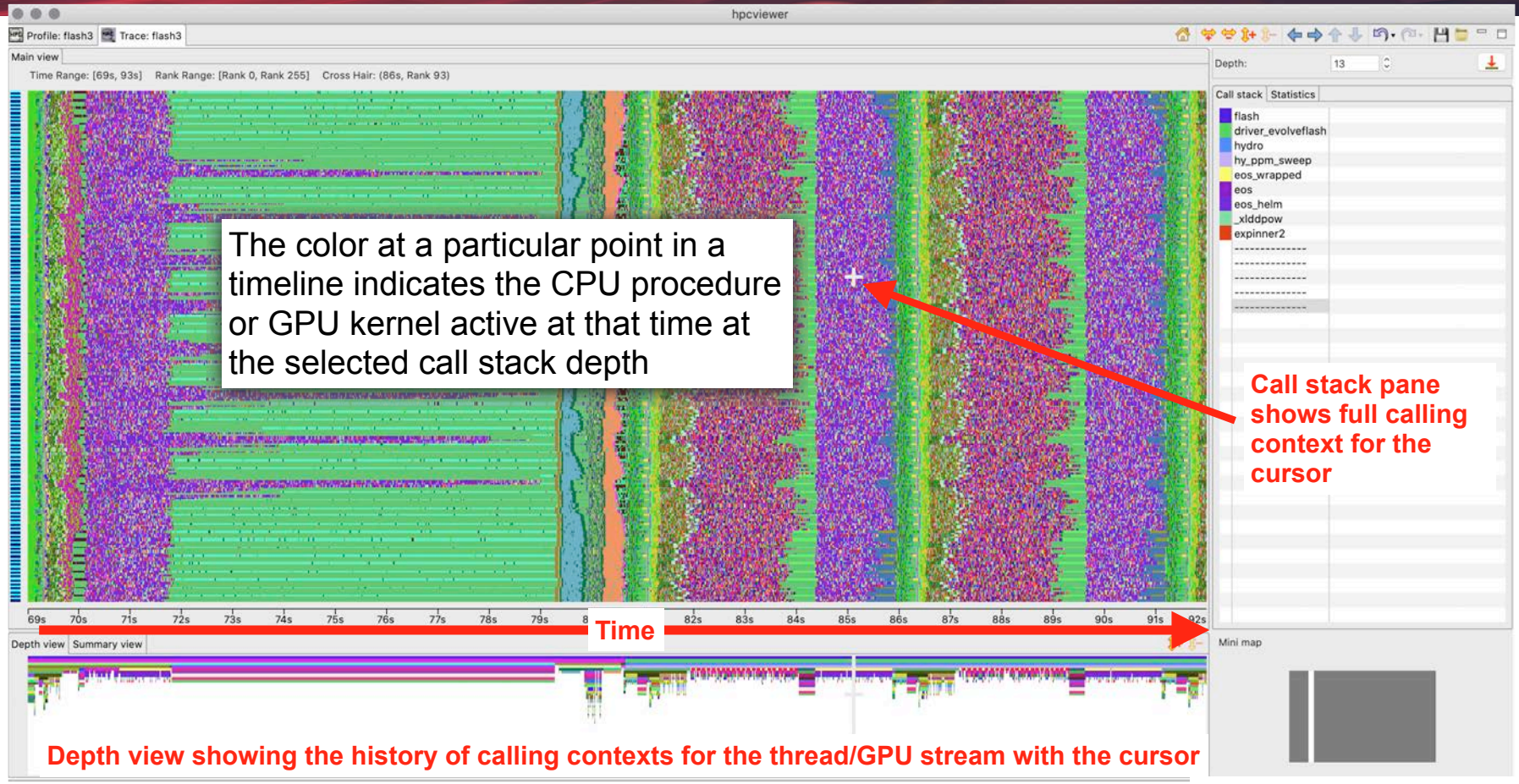


Understanding Temporal Behavior

- 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



Time-centric Analysis with hpcviewer



Minimap indicates part of execution trace shown

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

Case Studies

- ExaWind
- GAMESS (OpenMP)
- Quicksilver (CUDA)
- LAMMPS (Kokkos) at exascale

ExaWind: Wakes from Three Turbines over Time

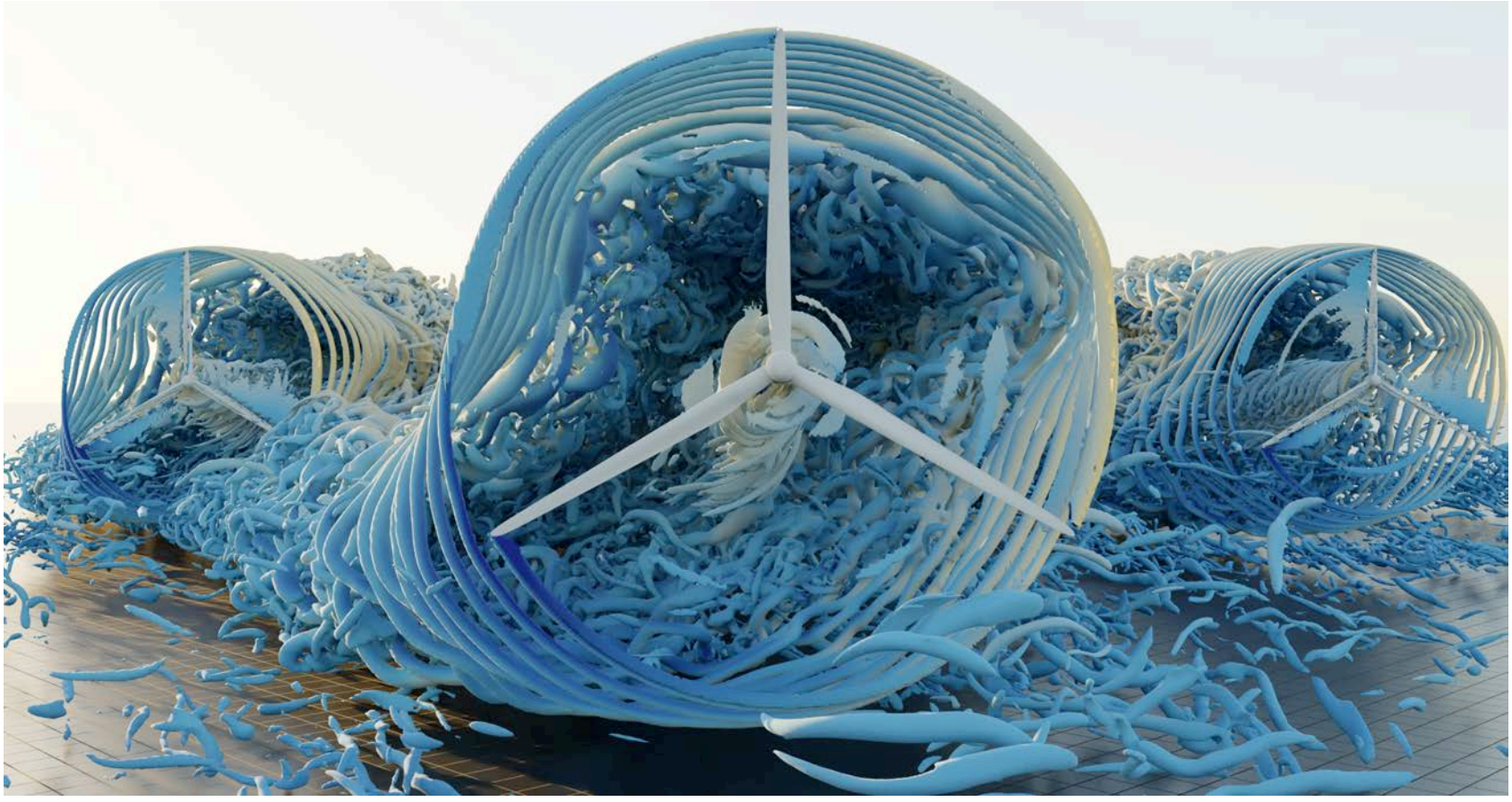


Figure credit: Jon Rood, NREL

ExaWind: Visualization of a Wind Farm Simulation

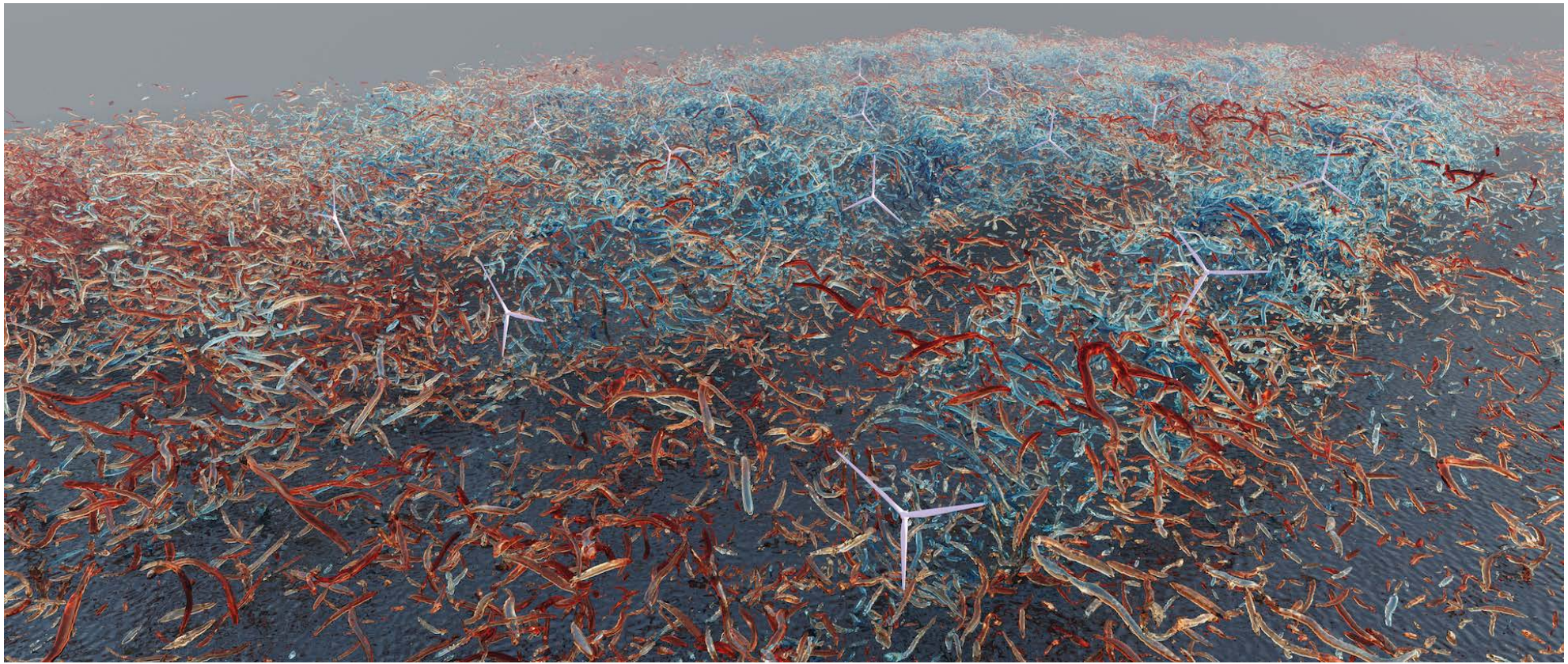


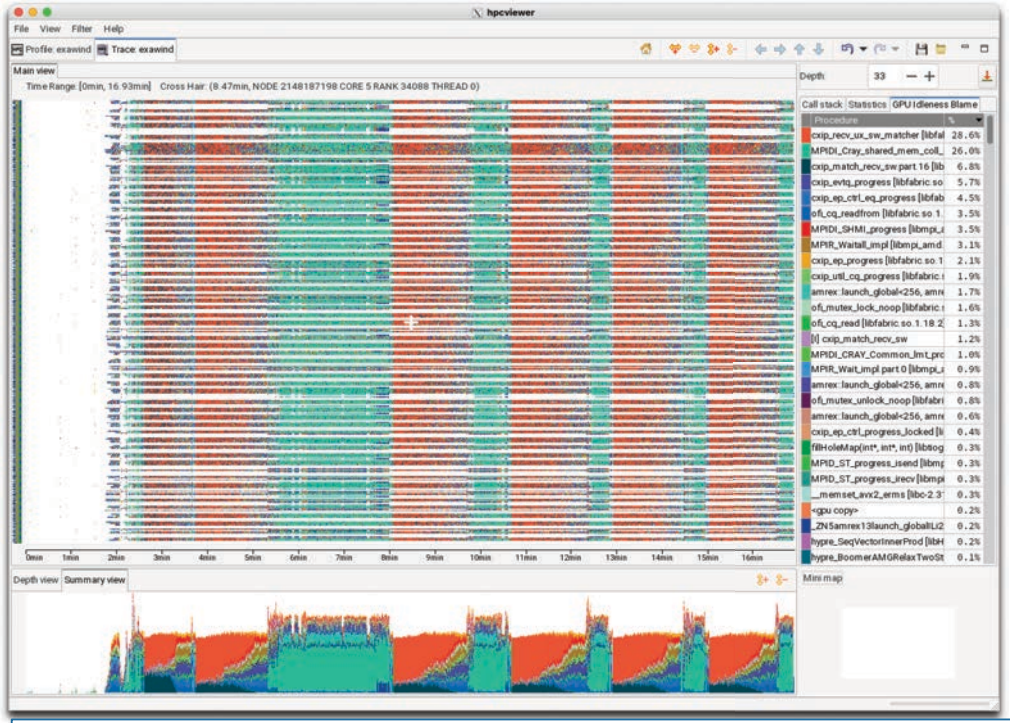
Figure credit: Jon Rood, NREL

ExaWind: Execution Traces on Frontier Collected with HPCToolkit

Traces on roughly ~70K MPI ranks for ~17minutes

Before: MPI waiting (bad), shown in red

After: MPI overhead negligible*



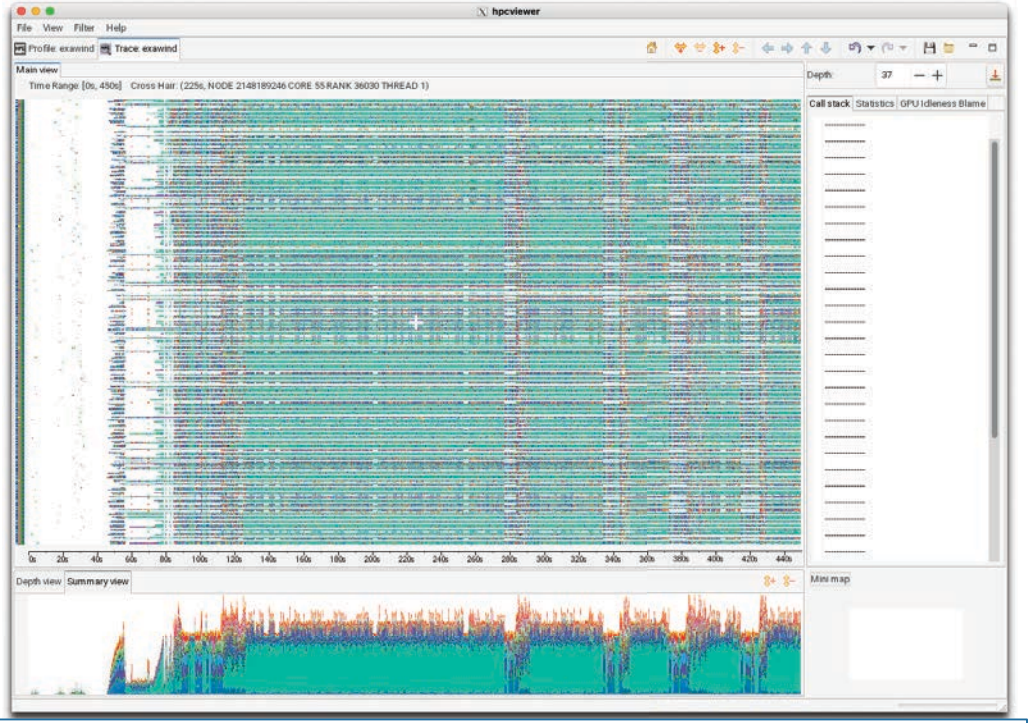


Figure credits: Jon Rood, NREL

*replaced non-blocking send/recv with ialltoallv

ExaWind Testimonials for HPCToolkit

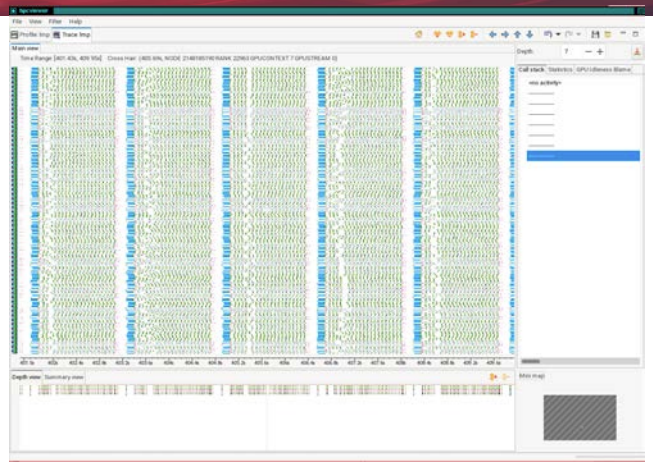
I just wanted to mention we've been using HPCToolkit a lot for our ExaWind application on Frontier, which is a hugely complicated code, and your profiler is one of the only ones we've found that really lets us easily instrument and then browse what our application is doing at runtime including GPUs. 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.

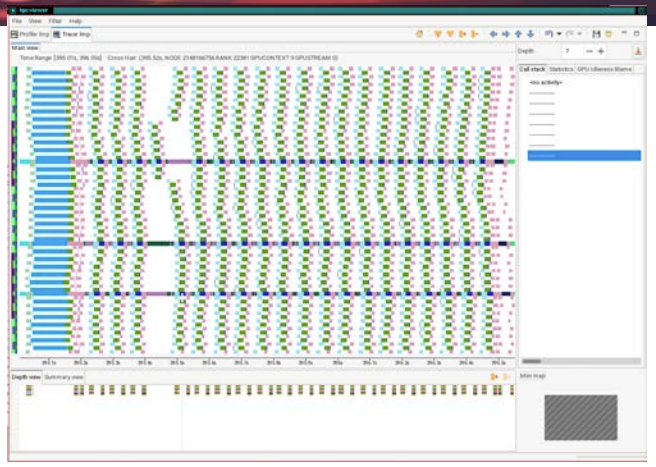
- Jon Rood NREL (5/31/2024)

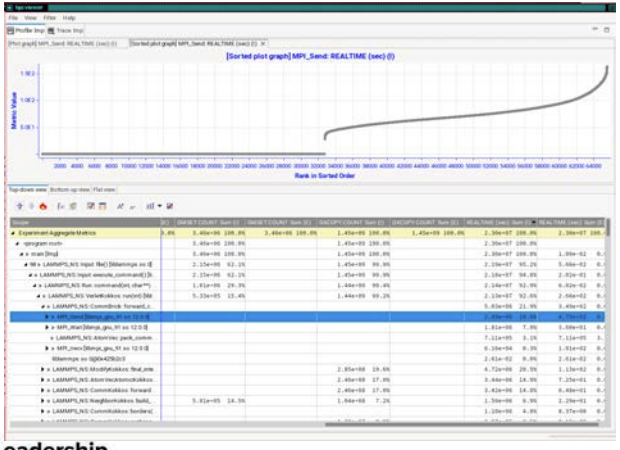
One big thing for us is that we can't overstate how complicated ExaWind is in general, and how complicated it is to build, so finding out that HPCToolkit could easily profile our entire application without a ton of instrumentation during the build process, and be able to profile it on a huge amount of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

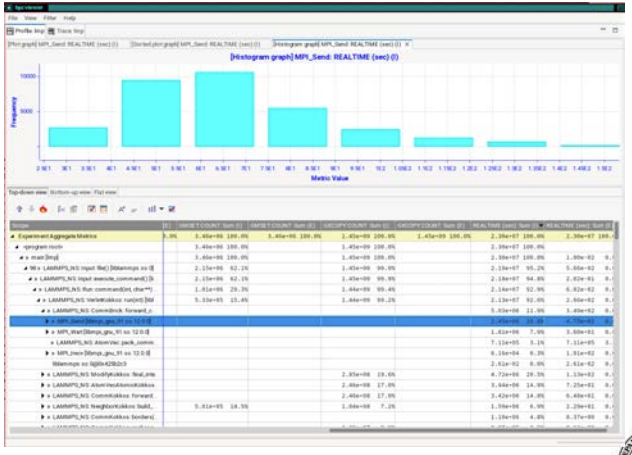
- Jon Rood NREL (6/3/2024)

LAMMPS on Frontier: Executions with Kernel Duration of Milliseconds

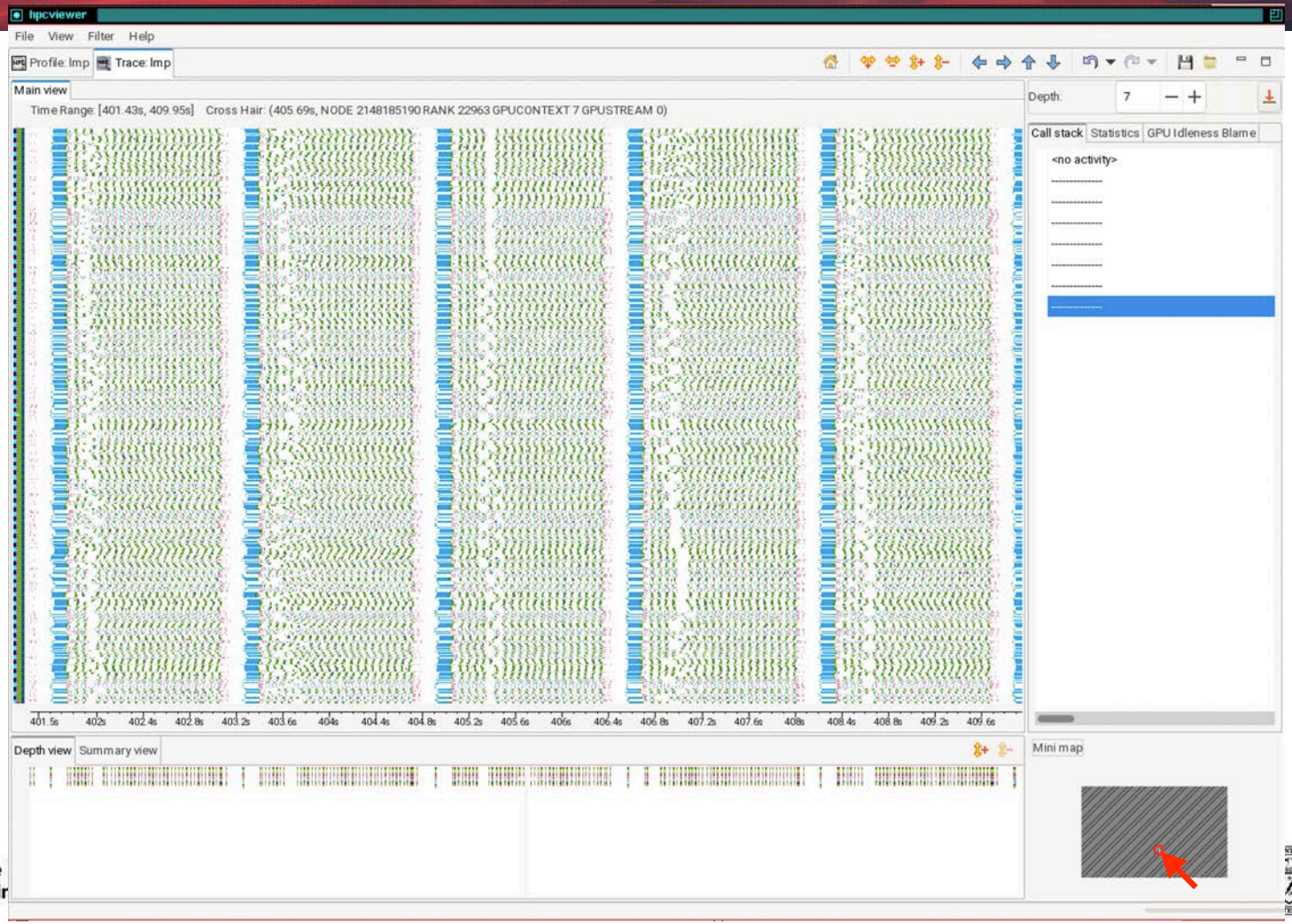




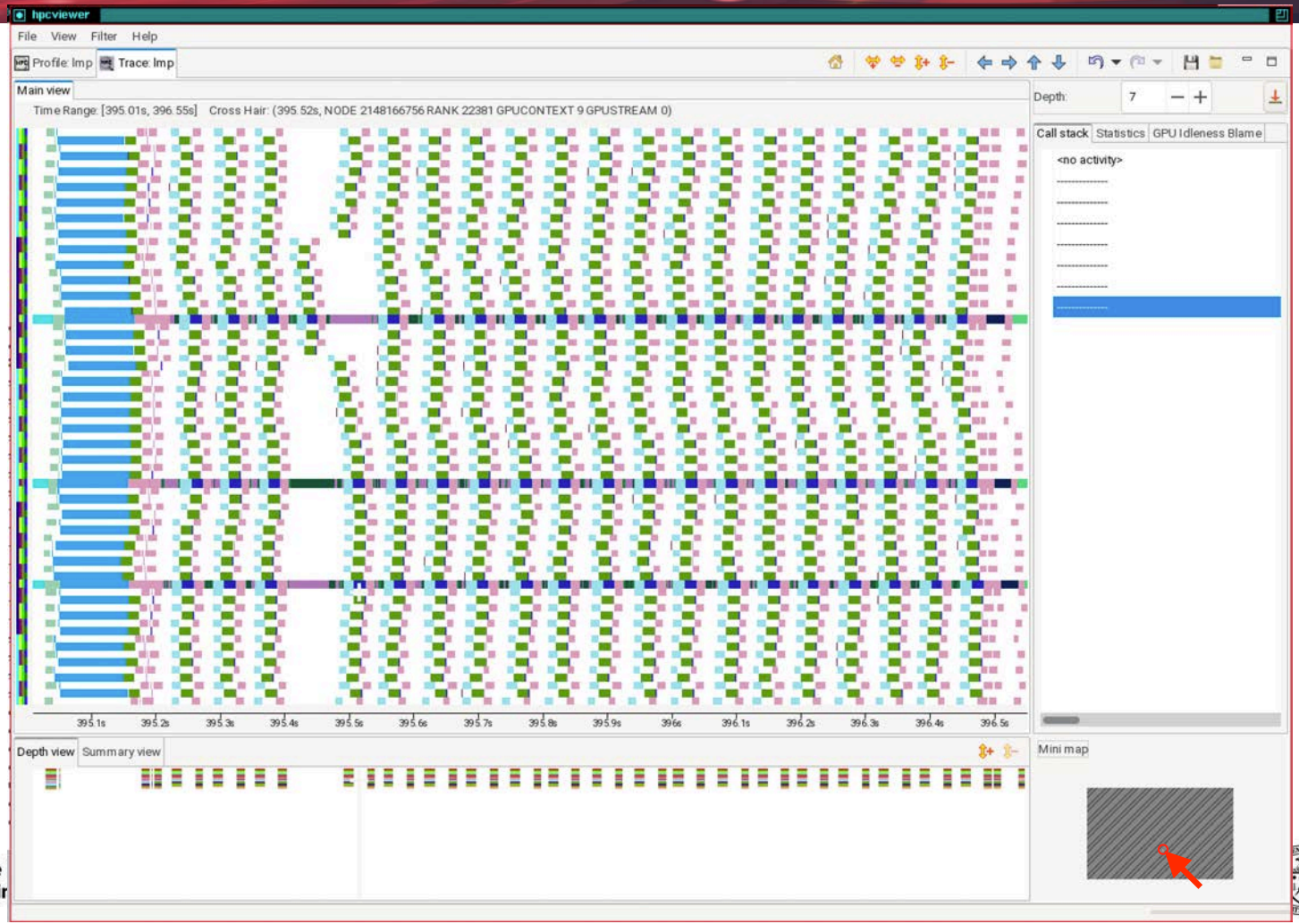




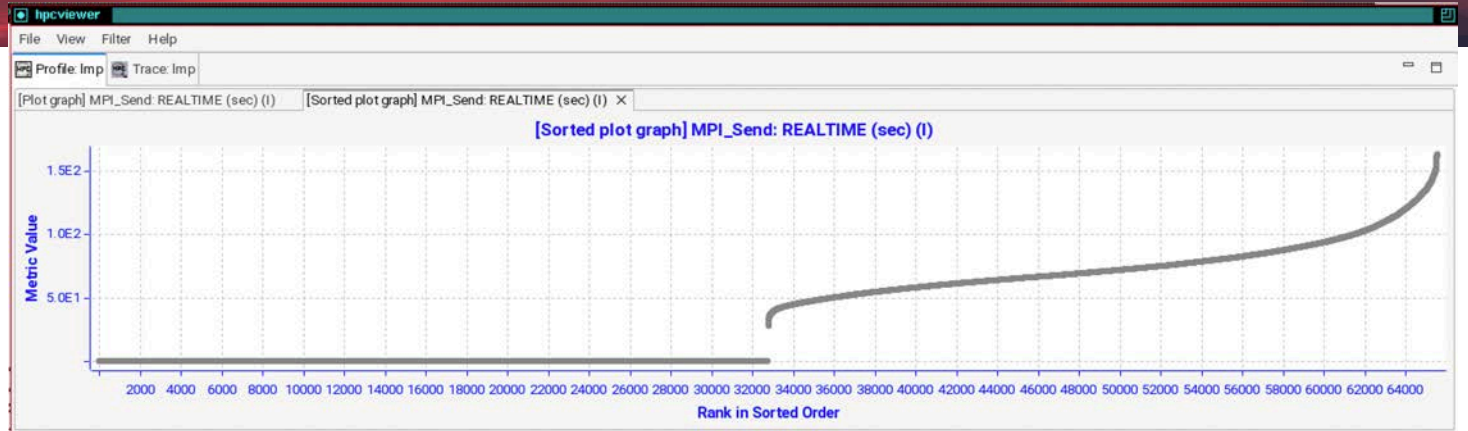
LAMMPS on Frontier: Executions with Kernel Duration of Milliseconds



LAMMPS on Frontier: Executions with Kernel Duration of Milliseconds

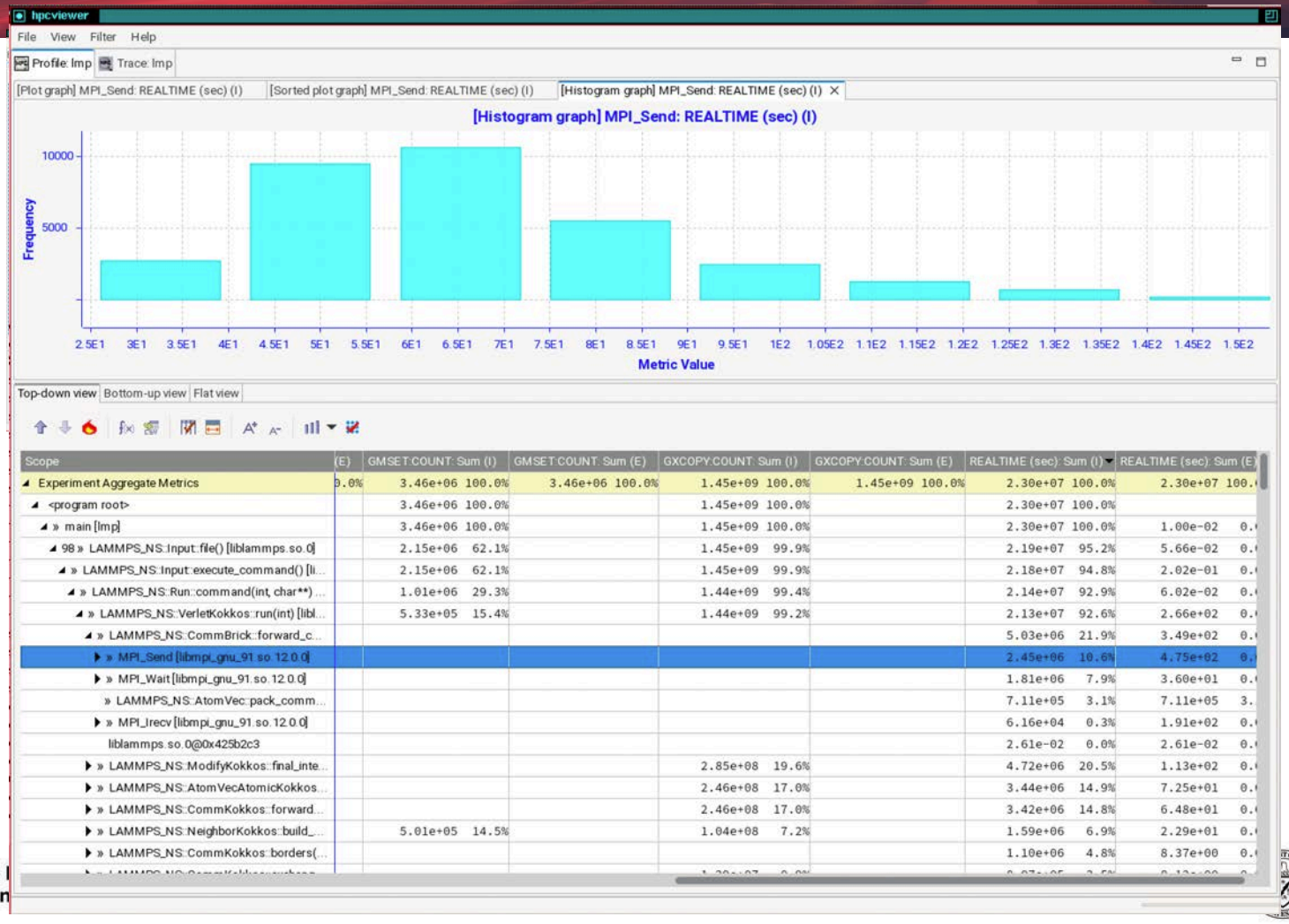


LAMMPS on Frontier: Executions with Kernel Duration of Milliseconds



Scope	(E)	GMSET:COUNT: Sum (I)	GMSET:COUNT: Sum (E)	GXCOPY:COUNT: Sum (I)	GXCOPY:COUNT: Sum (E)	REALTIME (sec): Sum (I)	REALTIME (sec): Sum (E)
Experiment Aggregate Metrics	0.0%	3.46e+06 100.0%	3.46e+06 100.0%	1.45e+09 100.0%	1.45e+09 100.0%	2.30e+07 100.0%	2.30e+07 100.0
<program root>		3.46e+06 100.0%		1.45e+09 100.0%		2.30e+07 100.0%	
» main [lmp]		3.46e+06 100.0%		1.45e+09 100.0%		2.30e+07 100.0%	1.00e-02 0.0
98 » LAMMPS_NS::Input::file() [liblammps.so.0]		2.15e+06 62.1%		1.45e+09 99.9%		2.19e+07 95.2%	5.66e-02 0.0
» LAMMPS_NS::Input::execute_command() [li...		2.15e+06 62.1%		1.45e+09 99.9%		2.18e+07 94.8%	2.02e-01 0.0
» LAMMPS_NS::Run::command(int, char**) ...		1.01e+06 29.3%		1.44e+09 99.4%		2.14e+07 92.9%	6.02e-02 0.0
» LAMMPS_NS::VerletKokkos::run(int) [libl...		5.33e+05 15.4%		1.44e+09 99.2%		2.13e+07 92.6%	2.66e+02 0.0
» LAMMPS_NS::CommBrick::forward_c...						5.03e+06 21.9%	3.49e+02 0.0
» MPI_Send [libmpi_gnu_91.so.12.0.0]						2.45e+06 10.6%	4.75e+02 0.0
» MPI_Wait [libmpi_gnu_91.so.12.0.0]						1.81e+06 7.9%	3.60e+01 0.0
» LAMMPS_NS::AtomVec::pack_comm...						7.11e+05 3.1%	7.11e+05 3.
» MPI_Irecv [libmpi_gnu_91.so.12.0.0]						6.16e+04 0.3%	1.91e+02 0.0
liblammps.so.0@0x425b2c3						2.61e-02 0.0%	2.61e-02 0.0
» LAMMPS_NS::ModifyKokkos::final_inte...				2.85e+08 19.6%		4.72e+06 20.5%	1.13e+02 0.0
» LAMMPS_NS::AtomVecAtomicKokkos...				2.46e+08 17.0%		3.44e+06 14.9%	7.25e+01 0.0
» LAMMPS_NS::CommKokkos::forward...				2.46e+08 17.0%		3.42e+06 14.8%	6.48e+01 0.0
» LAMMPS_NS::NeighborKokkos::build_...		5.01e+05 14.5%		1.04e+08 7.2%		1.59e+06 6.9%	2.29e+01 0.0
» LAMMPS_NS::CommKokkos::borders(...						1.10e+06 4.8%	8.37e+00 0.0

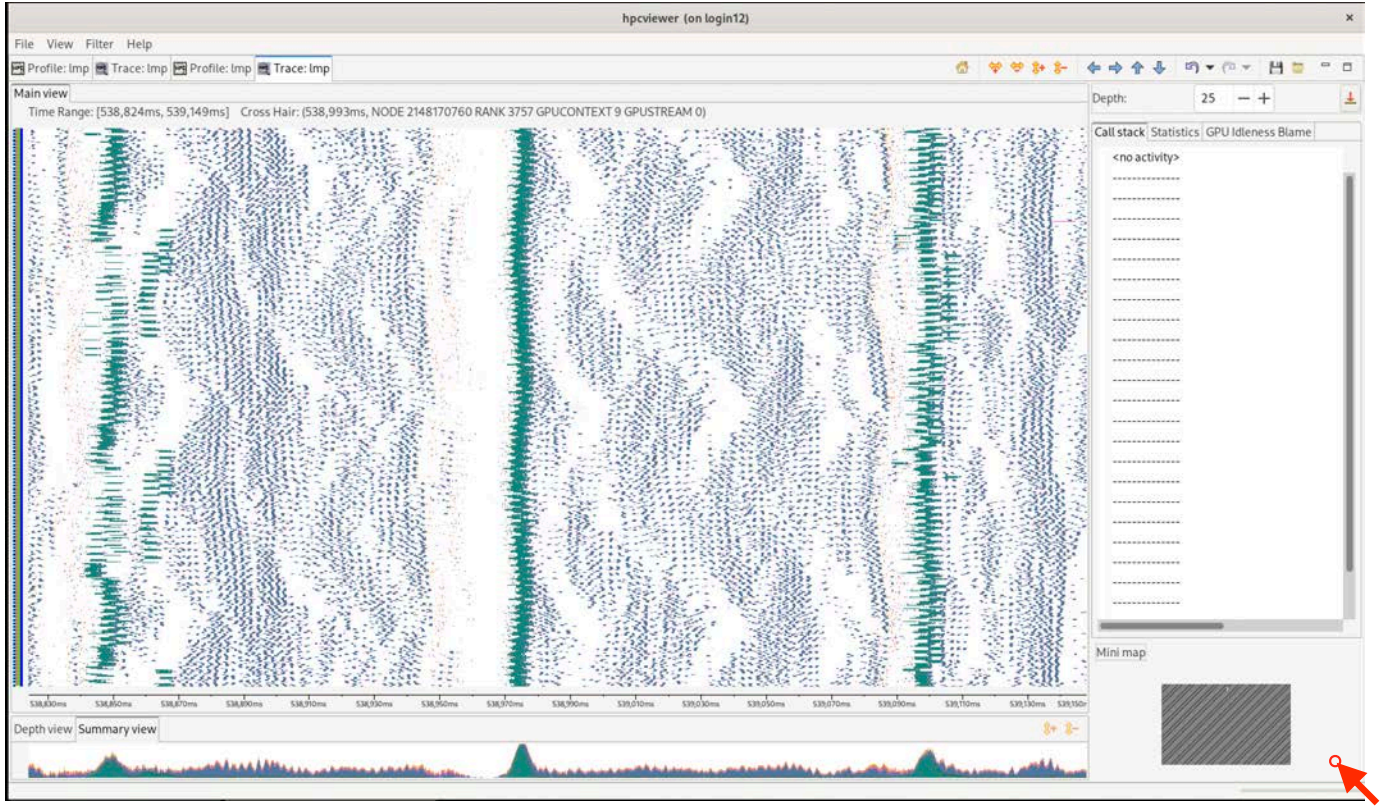
LAMMPS on Frontier: Executions with Kernel Duration of Milliseconds



Scope	(E)	GMSET:COUNT: Sum (I)	GMSET:COUNT: Sum (E)	GXCOPY:COUNT: Sum (I)	GXCOPY:COUNT: Sum (E)	REALTIME (sec): Sum (I)	REALTIME (sec): Sum (E)
Experiment Aggregate Metrics	0.0%	3.46e+06 100.0%	3.46e+06 100.0%	1.45e+09 100.0%	1.45e+09 100.0%	2.30e+07 100.0%	2.30e+07 100.0
<program root>		3.46e+06 100.0%		1.45e+09 100.0%		2.30e+07 100.0%	
» main [lmp]		3.46e+06 100.0%		1.45e+09 100.0%		2.30e+07 100.0%	1.00e-02 0.0
98 » LAMMPS_NS::Input::file() [liblammps.so.0]		2.15e+06 62.1%		1.45e+09 99.9%		2.19e+07 95.2%	5.66e-02 0.0
» LAMMPS_NS::Input::execute_command() [li...		2.15e+06 62.1%		1.45e+09 99.9%		2.18e+07 94.8%	2.02e-01 0.0
» LAMMPS_NS::Run::command(int, char**) ...		1.01e+06 29.3%		1.44e+09 99.4%		2.14e+07 92.9%	6.02e-02 0.0
» LAMMPS_NS::VerletKokkos::run(int) [libl...		5.33e+05 15.4%		1.44e+09 99.2%		2.13e+07 92.6%	2.66e+02 0.0
» LAMMPS_NS::CommBrick::forward_c...						5.03e+06 21.9%	3.49e+02 0.0
» MPI_Send [libmpi_gnu_91.so.12.0.0]						2.45e+06 10.6%	4.75e+02 0.0
» MPI_Wait [libmpi_gnu_91.so.12.0.0]						1.81e+06 7.9%	3.60e+01 0.0
» LAMMPS_NS::AtomVec::pack_comm...						7.11e+05 3.1%	7.11e+05 3.
» MPI_Irecv [libmpi_gnu_91.so.12.0.0]						6.16e+04 0.3%	1.91e+02 0.0
liblammps.so.0@0x425b2c3						2.61e-02 0.0%	2.61e-02 0.0
» LAMMPS_NS::ModifyKokkos::final_inte...				2.85e+08 19.6%		4.72e+06 20.5%	1.13e+02 0.0
» LAMMPS_NS::AtomVecAtomicKokkos...				2.46e+08 17.0%		3.44e+06 14.9%	7.25e+01 0.0
» LAMMPS_NS::CommKokkos::forward...				2.46e+08 17.0%		3.42e+06 14.8%	6.48e+01 0.0
» LAMMPS_NS::NeighborKokkos::build_...		5.01e+05 14.5%		1.04e+08 7.2%		1.59e+06 6.9%	2.29e+01 0.0
» LAMMPS_NS::CommKokkos::borders(...						1.10e+06 4.8%	8.37e+00 0.0

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

Kernel duration of microseconds



Case Study: GAMESS

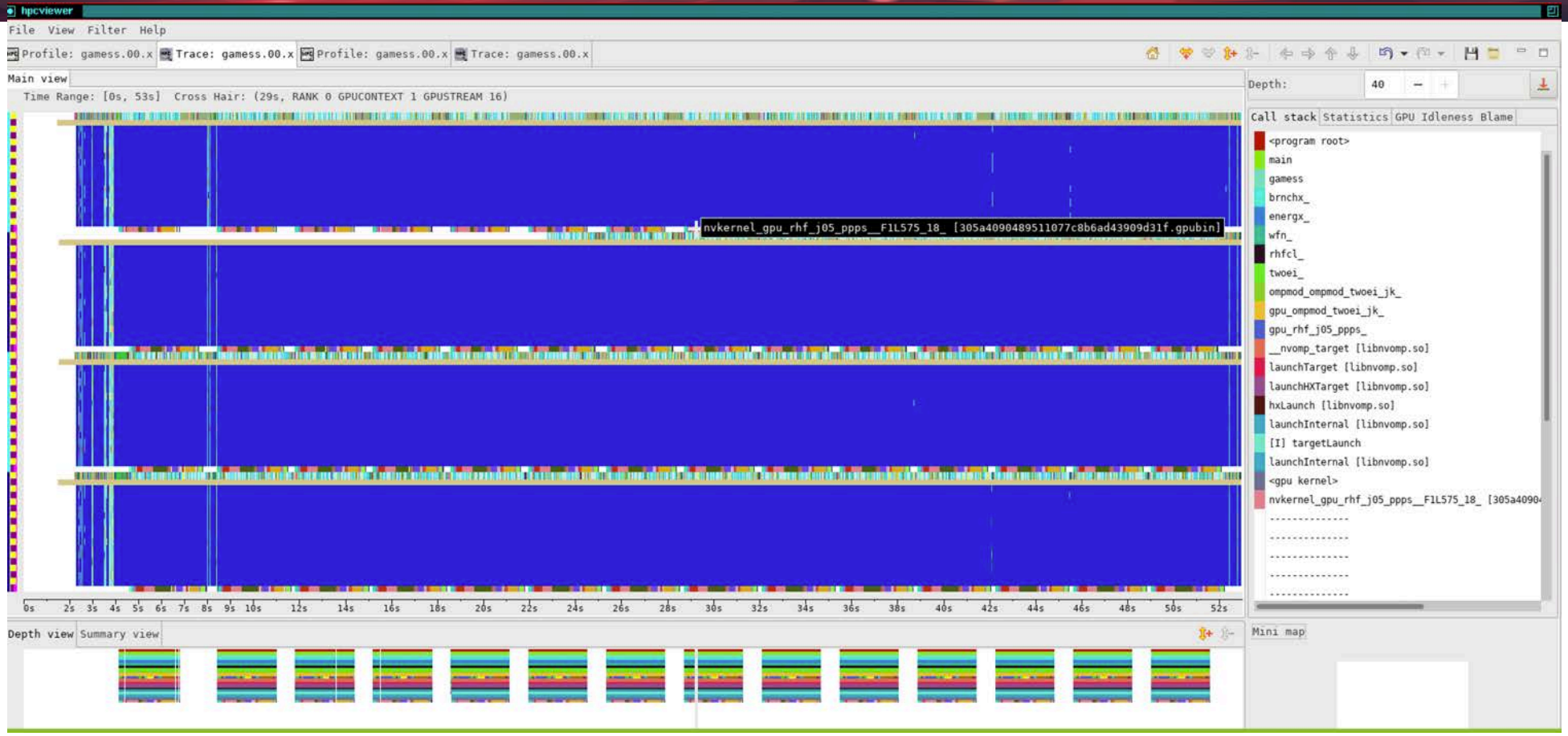
- General Atomic and Molecular Electronic Structure System (GAMESS)
  - —general *ab initio* quantum chemistry package
- Calculates the energies, structures, and properties of a wide range of chemical systems

- 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

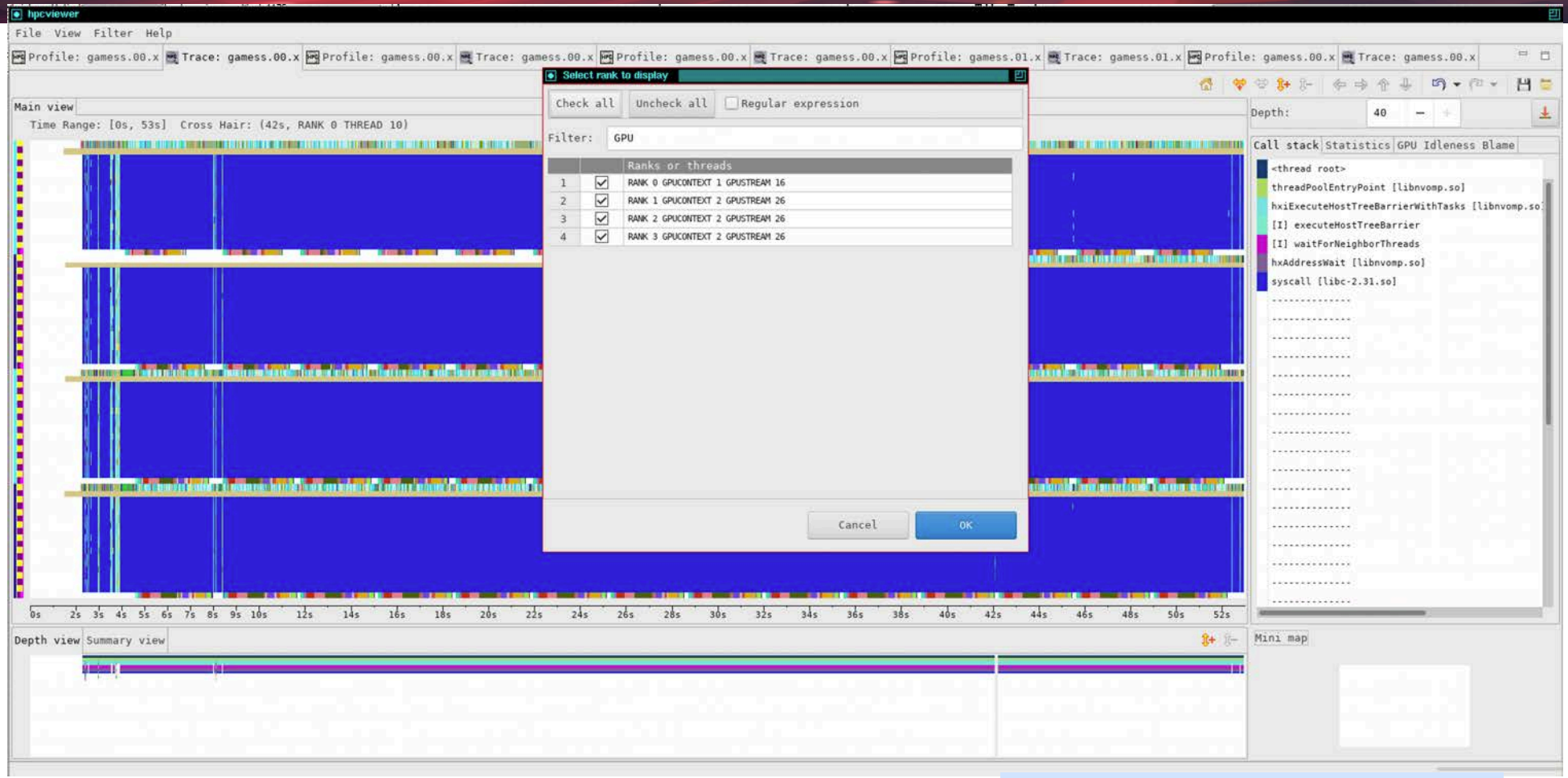
Time-centric Analysis: GAMESS 4 ranks, 4 GPUs on Perlmutter



GAMESS original

All CPU threads and GPU streams

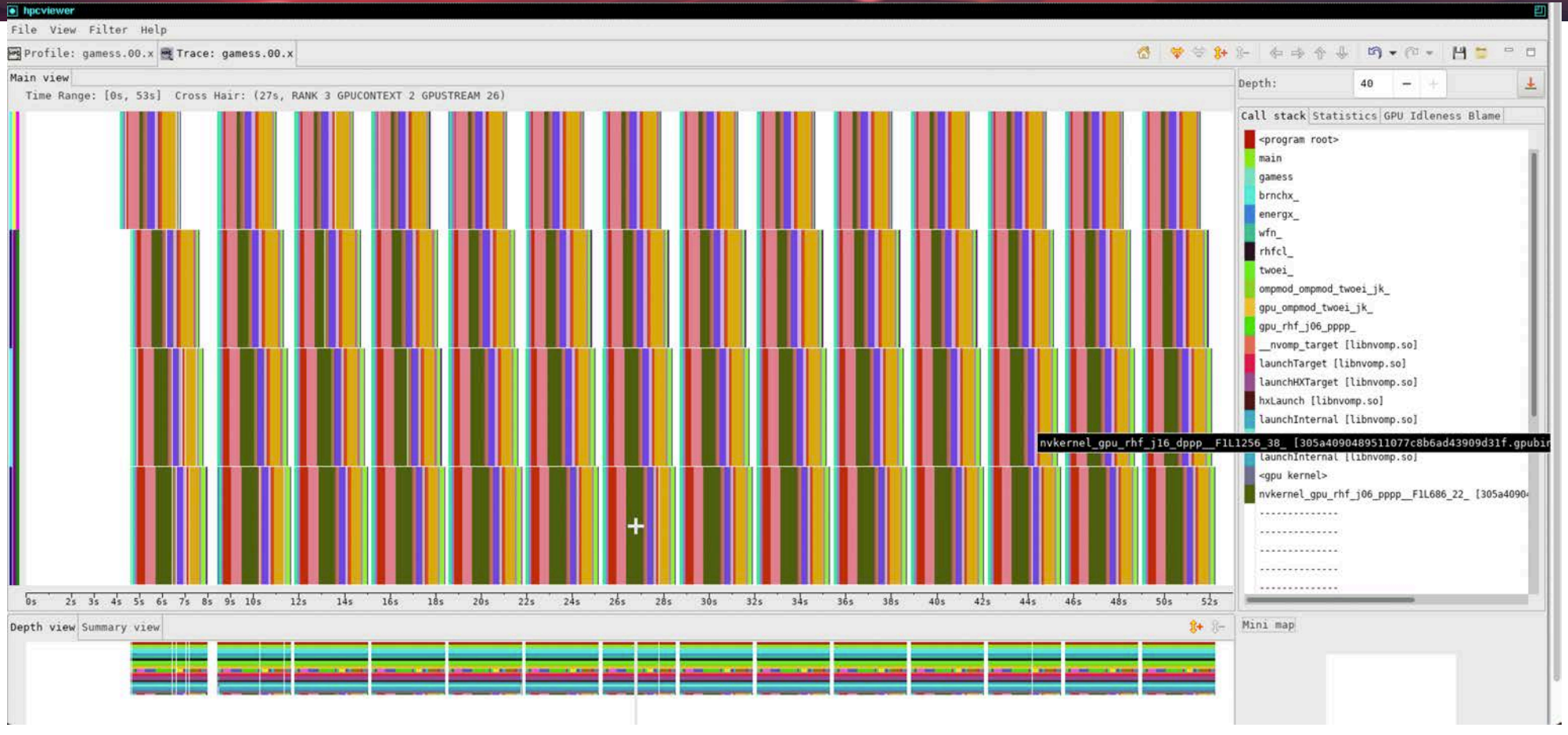
Time-centric Analysis: GAMESS 4 ranks, 4 GPUs on Perlmutter



GAMESS original

All CPU threads and GPU streams

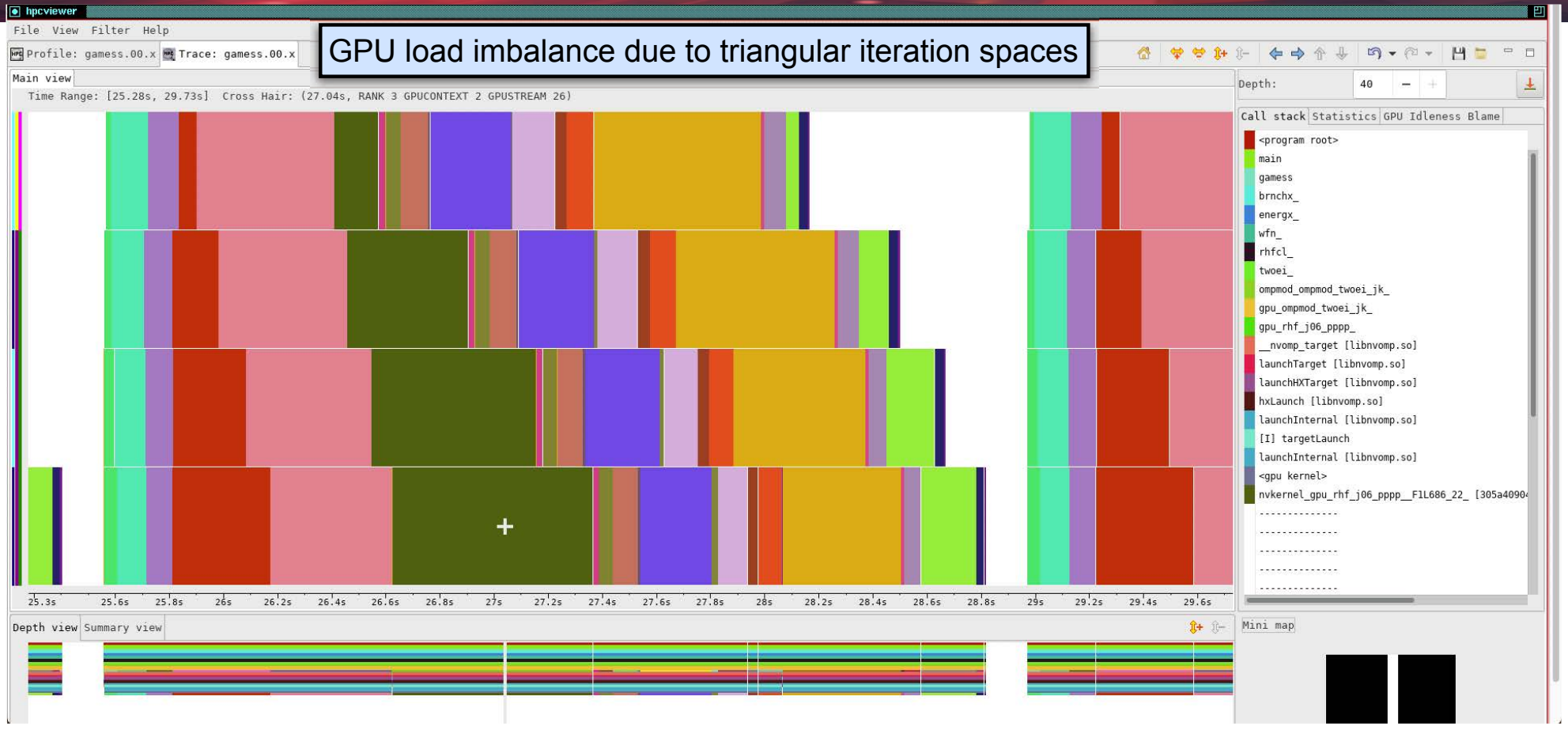
Time-centric Analysis: GAMESS 4 ranks, 4 GPUs on Perlmutter



GAMESS original

All GPU streams, whole execution

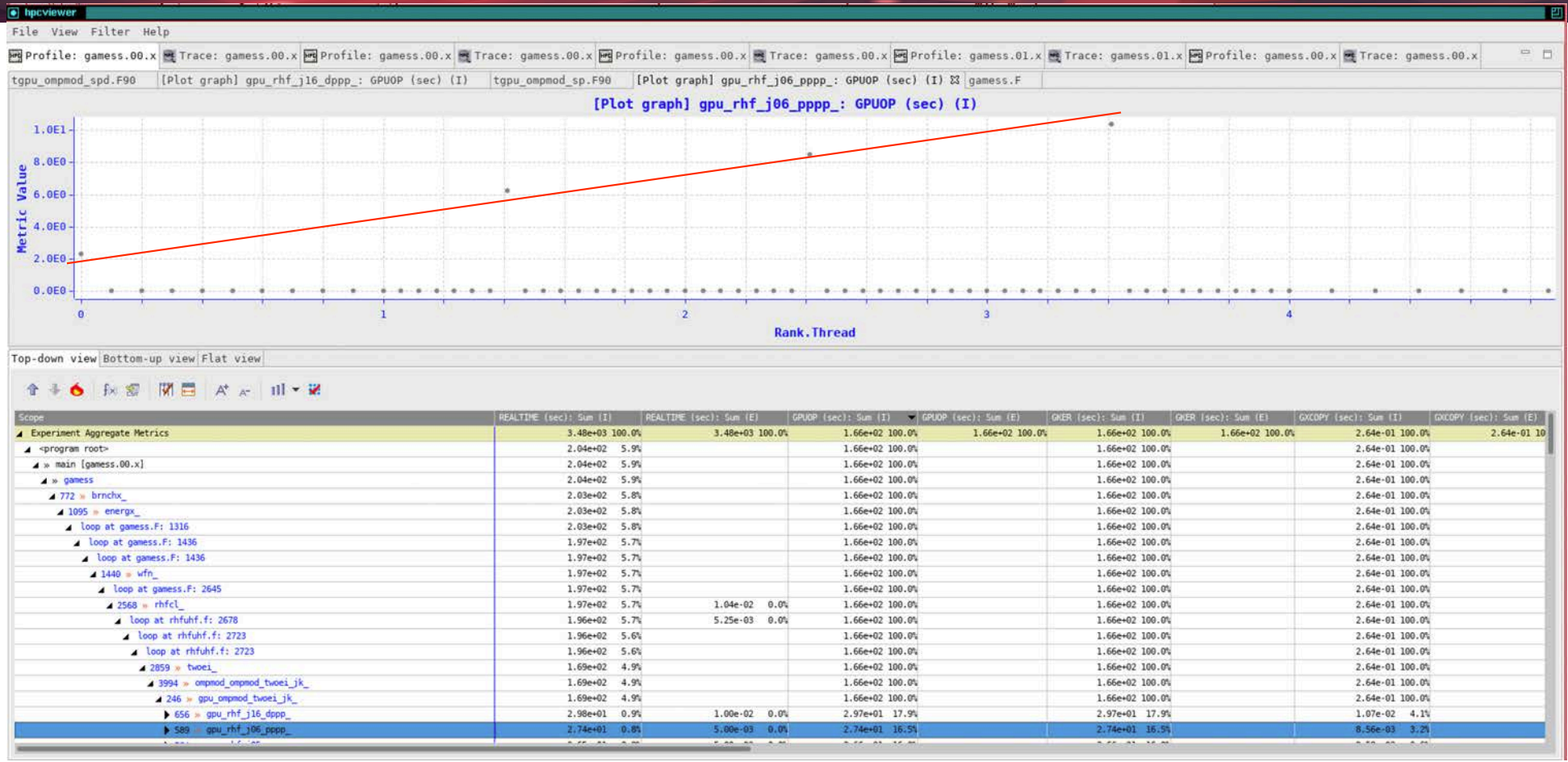
Time-centric Analysis: GAMESS 4 ranks, 4 GPUs on Perlmutter



GAMESS original

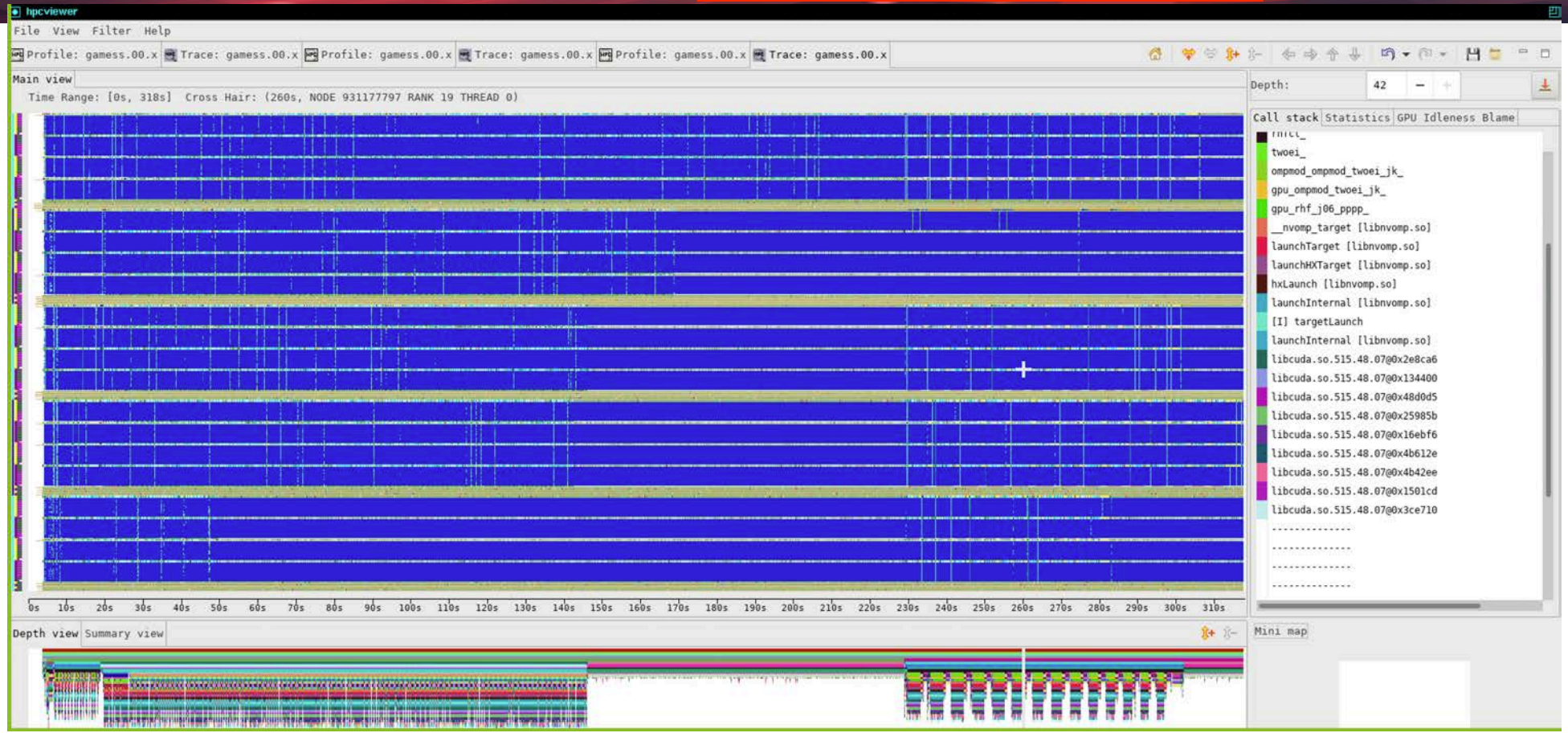
GPU streams: 1 iteration

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



GAMESS original

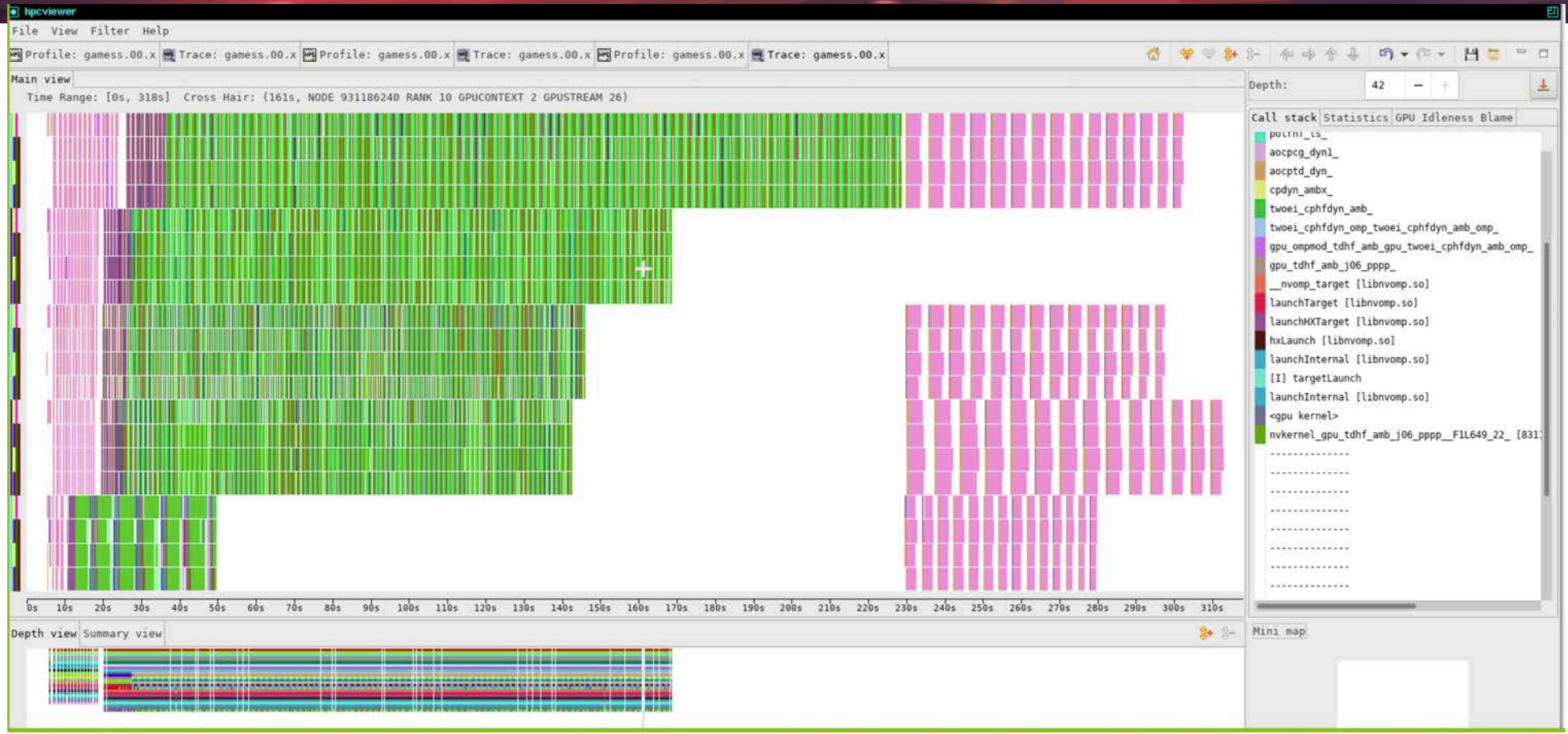
Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter



GAMESS improved

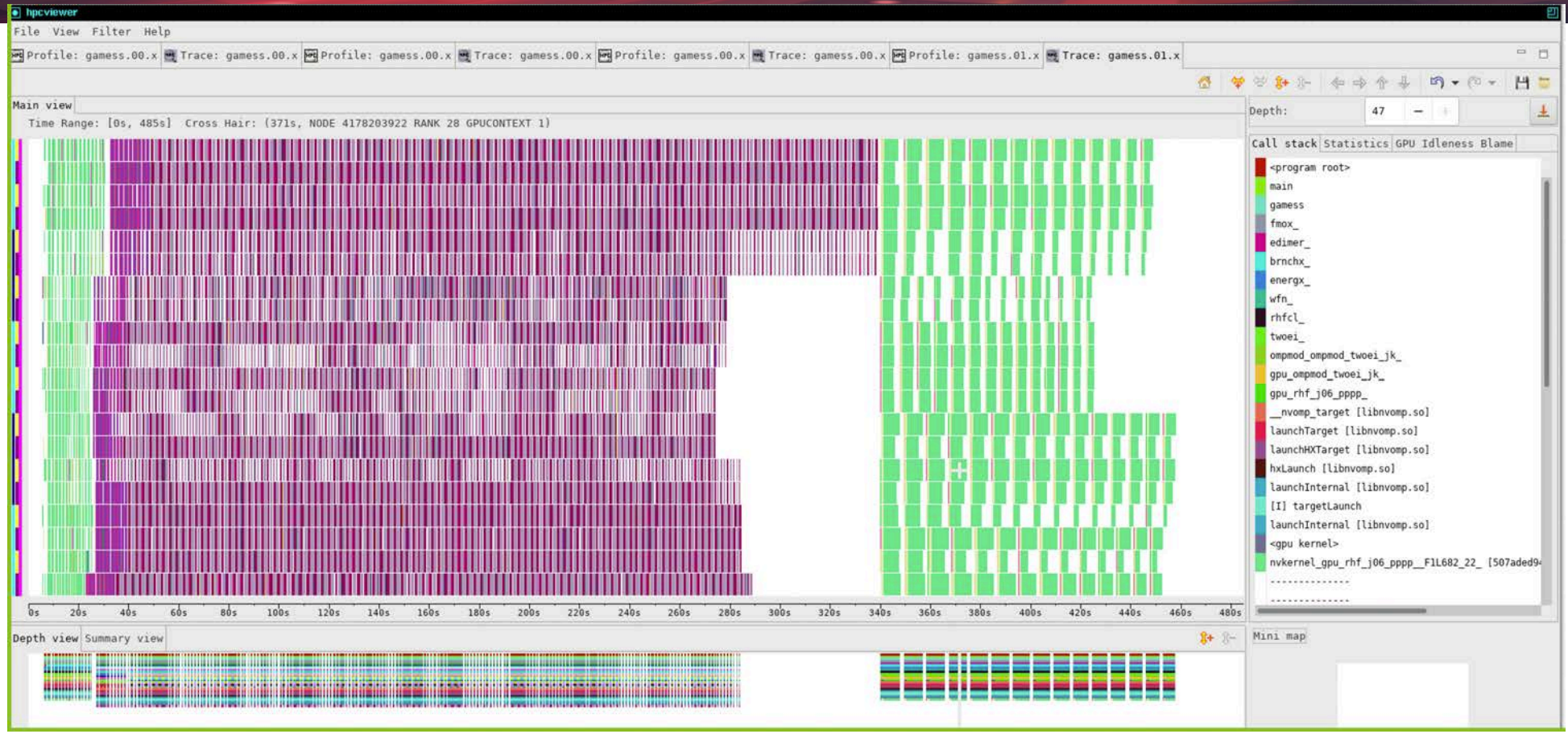
CPU Threads and GPU Streams

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



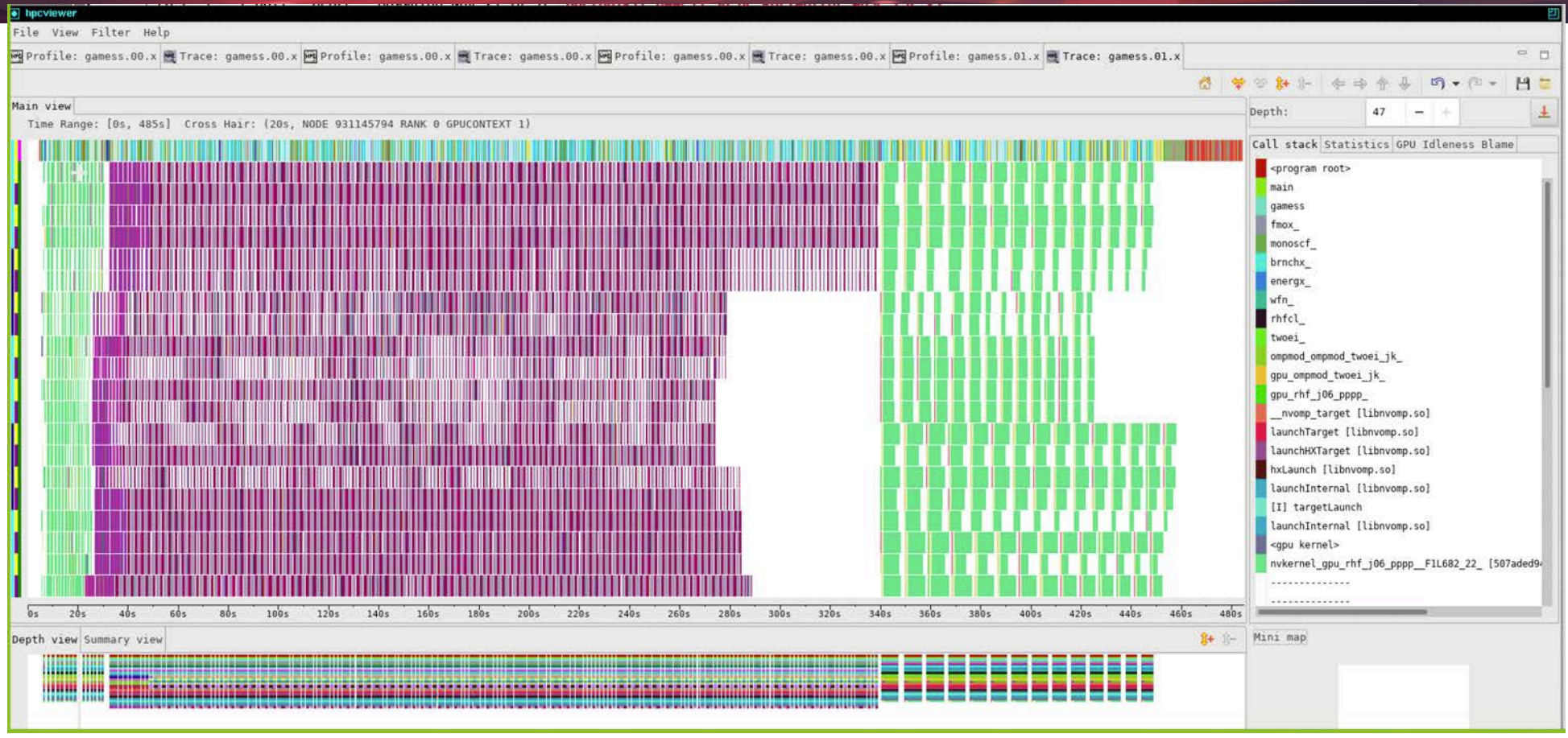
GAMESS improved

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



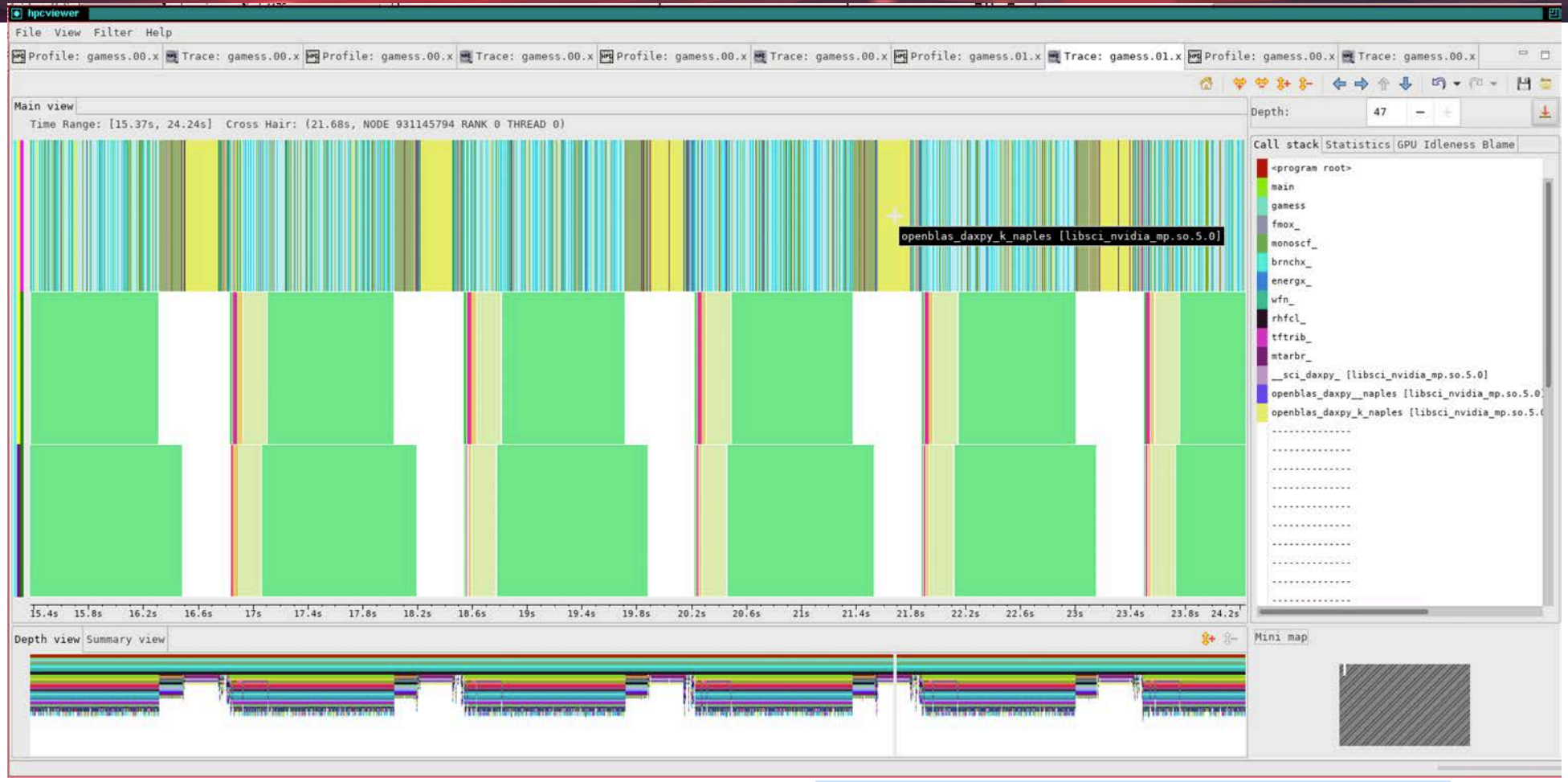
GAMESS improved with better manual distribution of work in input

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



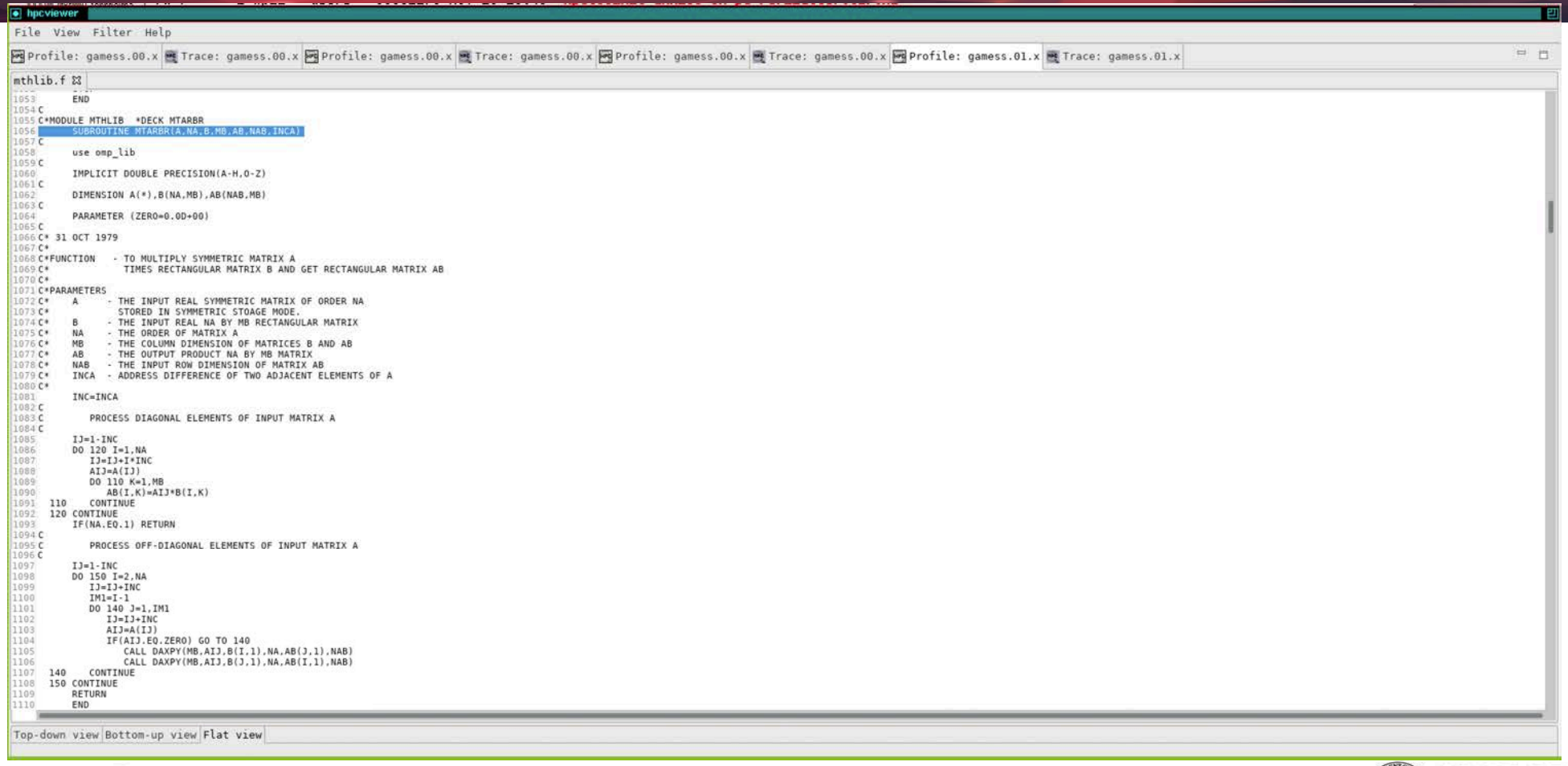
GAMESS improved adding Rank 0 Thread 0 to GPU streams

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

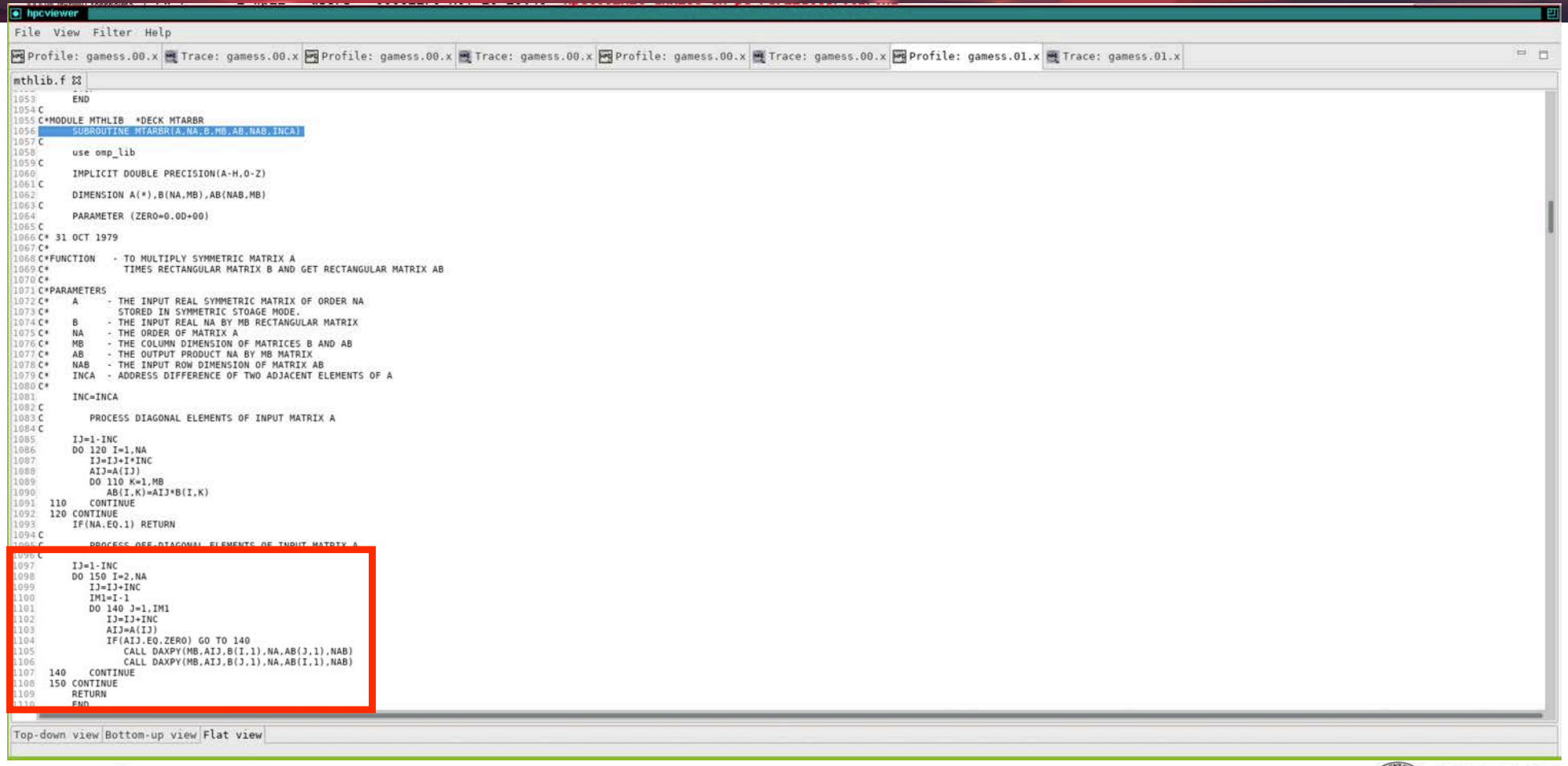


1 CPU Stream, 2 GPU Streams: 6 Iterations

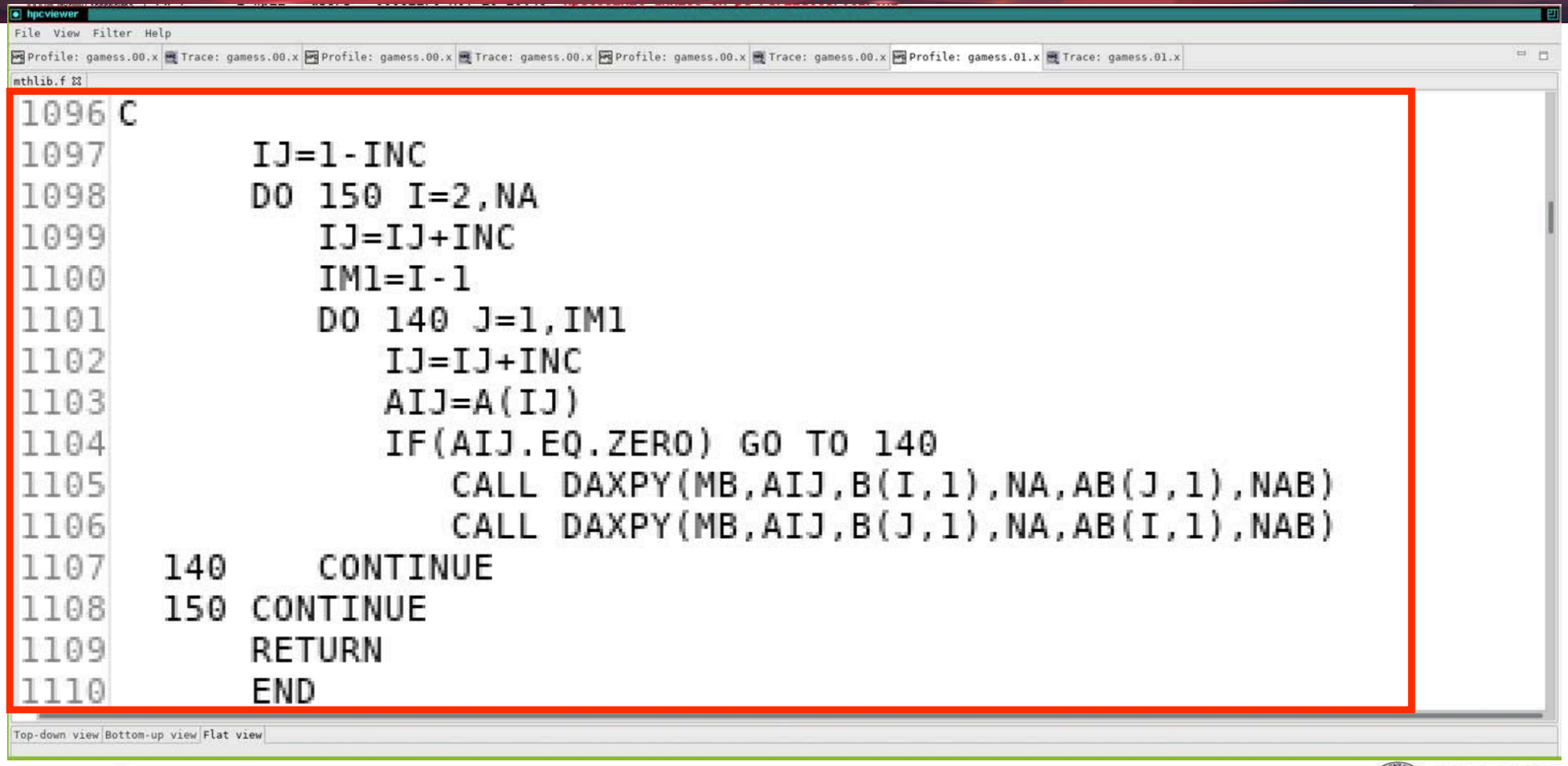
Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter



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



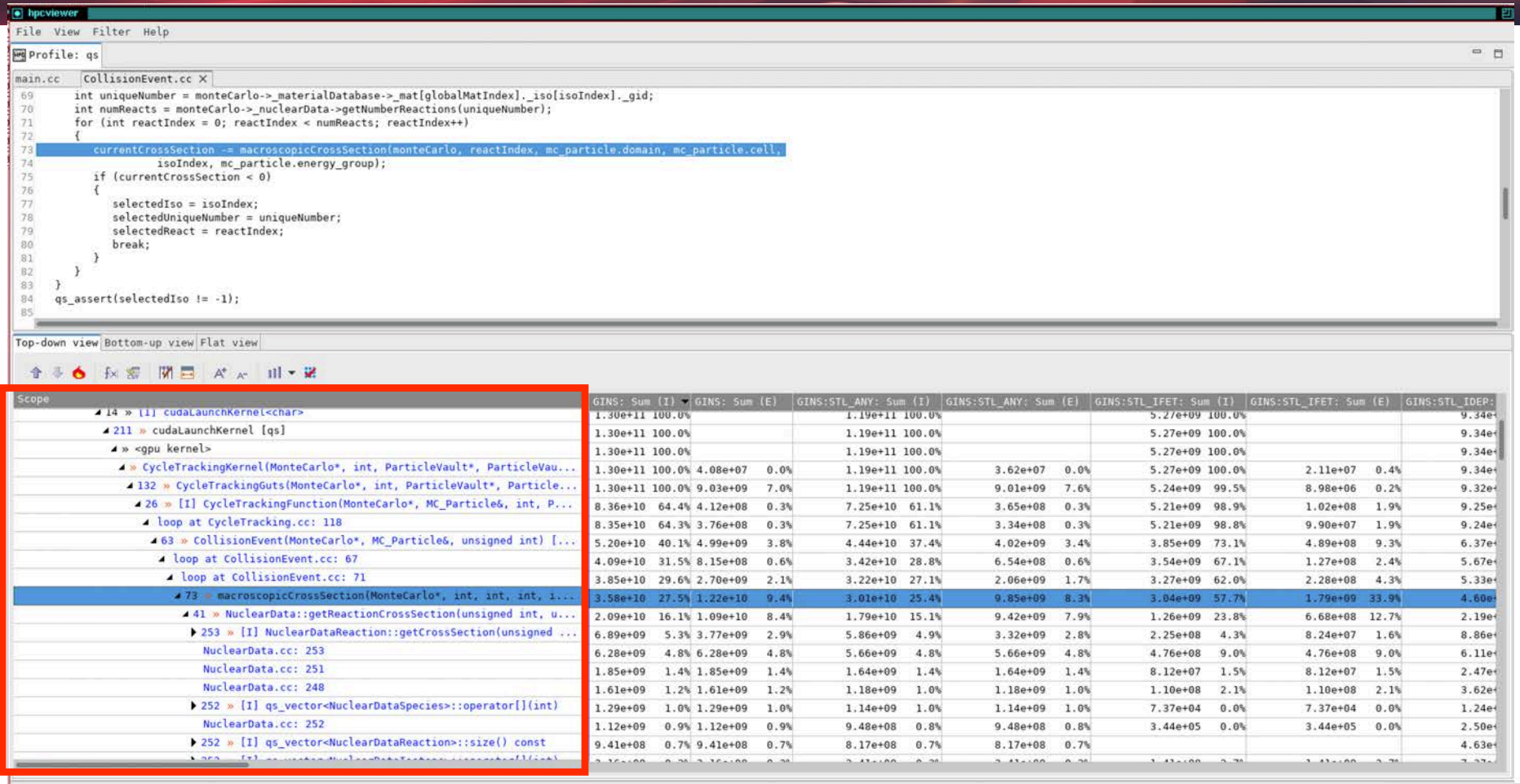
Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter



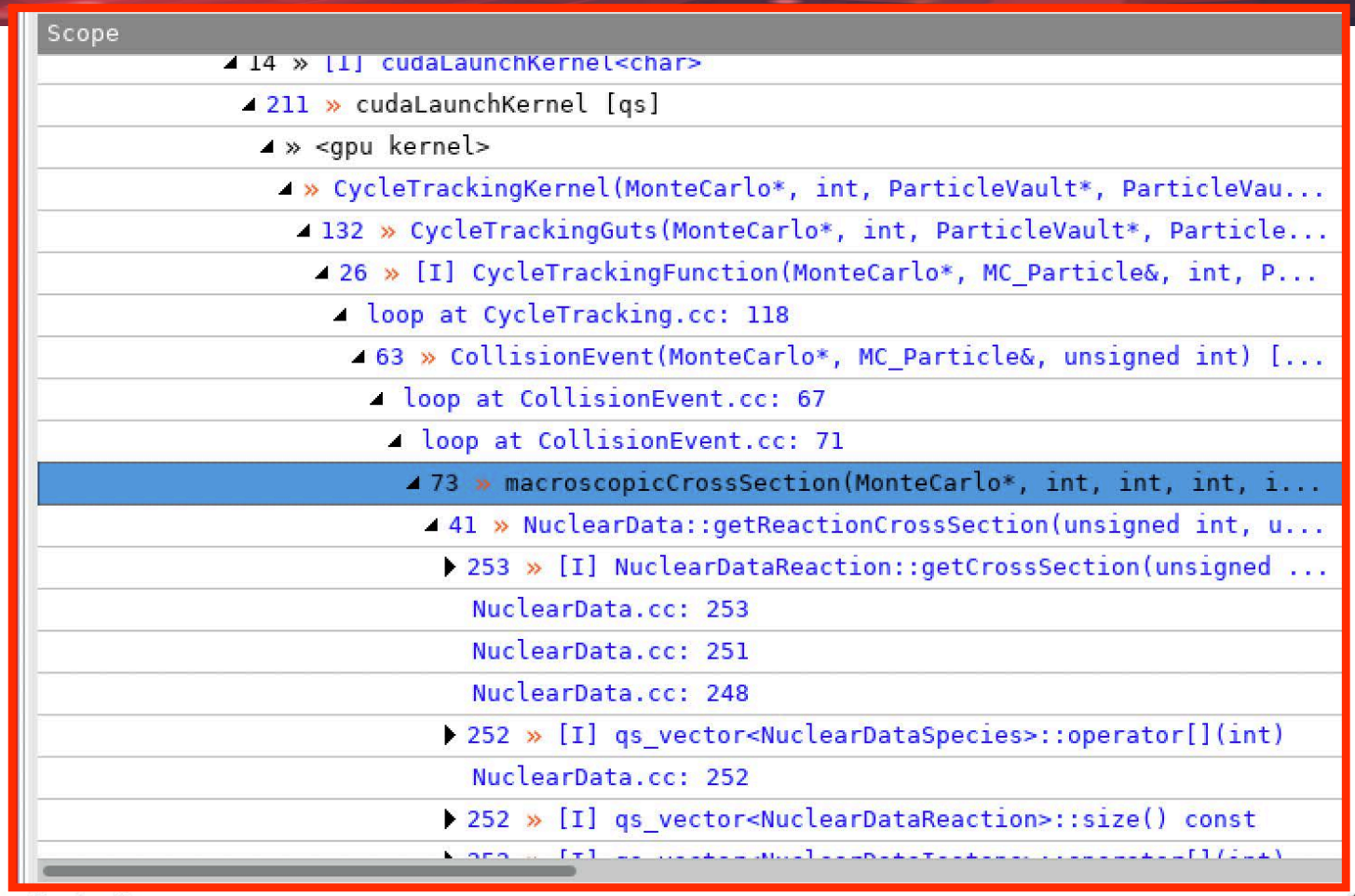
Case Study: Quicksilver

- Proxy application that represents some elements of LLNL's Mercury workload
- 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



HPCToolkit Resources

- Documentation
  - —User manual for HPCToolkit: https://hpctoolkit.gitlab.io/hpctoolkit
  - —Cheat sheet: https://gitlab.com/hpctoolkit/hpctoolkit/-/wikis/HPCToolkit-cheat-sheet
  - —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
    - https://hpctoolkit.gitlab.io/hpctoolkit/users/spack.html

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
      - can correct the statistics by turning the mode off
  - mode can be selected from <File>:<Preferences>:<Traces>

Hands-on Examples

Two Kinds of Hands-on Examples

- Pre-collected databases to explore
  - gain experience using hpctoolkit's hpcviewer graphical user interface to analyze performance data
- Hands-on examples
  - build, run, and view several codes to get 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

Performance Databases to Explore

- **On an aurora login node**

```
% qsub -I -l select=2,walltime=1:00:00,place=scatter -l
  filesystems=flare -A gpu_hack -q gpu_hack_prio -X
```

- **On an aurora compute node**

```
% module use /soft/perftools/hpctoolkit/modulefiles
% module load hpctoolkit
% cd /flare/ATPESC2026/EXAMPLES/hpctoolkit/DONT-COPY/data
% ls
      arborx  gamess  minitest  pelelmex  quicksilver
```

NOTE: all of the databases in these directories end with the suffix ".d". For the gamess examples, the hpctoolkit databases are one directory deeper, i.e. in subdirectories that begin with a number.

More about the Available Performance Databases

See /flare/gpu_hack/hpctoolkit/data for each of the following

- 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
  - EXERCISES
- 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/
- arborx **(C++ + Kokkos)**
- minitest **(SYCL and OpenMP TARGET)**

Hands-on Tutorial Examples on Aurora

```
% cd /flare/ATPESC2026/usr/${USER}
% cp -r /flare/ATPESC2026/EXAMPLES/hpctoolkit/hpctoolkit-hands-on .
% cd hpctoolkit-hands-on
% ls
   pcstruct  minitest  opensn  README
   sourceme-get-compute-node.sh  sourceme-on-compute-node.sh
 For your chosen example
 1. cd to the example directory
 2. source setup-env/aurora.sh  # custom for each example
 3. make build # build the code
 4. make run # measure and analyze code performance
    this will produce a directory <something>.db
 5. View the performance data with hpcviewer
    a. remotely from your laptop using an hpcviewer that you downloaded
    b. make view
       REQUIREMENTS:
       1. you connected to aurora using an X11 desktop
       2. you run "make view" on a login node
```

Downloading, Installing, and Using Hpcviewer Graphical User Interface on Your Laptop

Hpcviewer Graphical User Interface on Your Laptop

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
- User manual for hpcviewer: https://hpctoolkit.gitlab.io/hpctoolkit

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/1.4.0
```

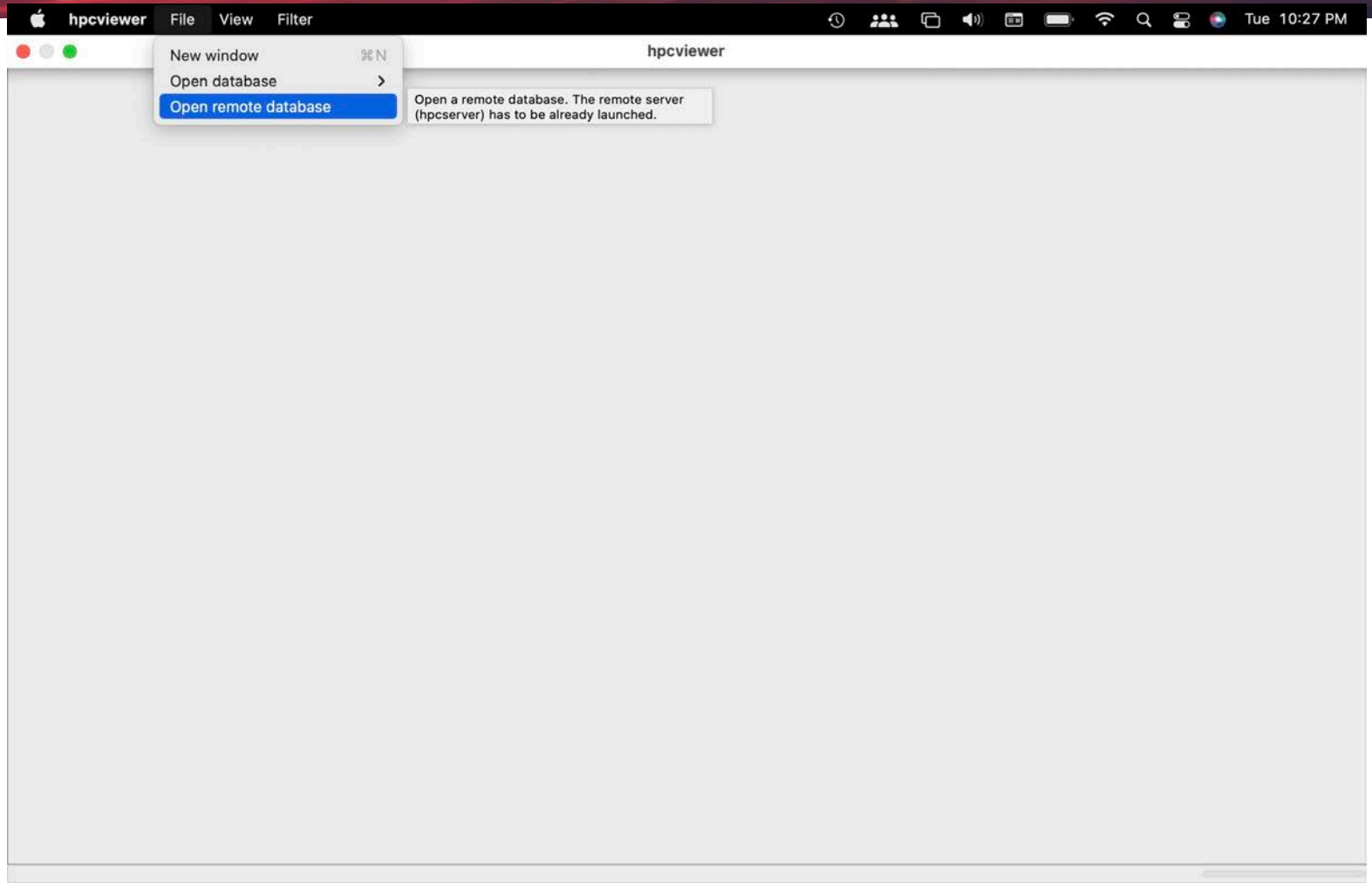
Select the authentication method: “Use password”

Click “OK”

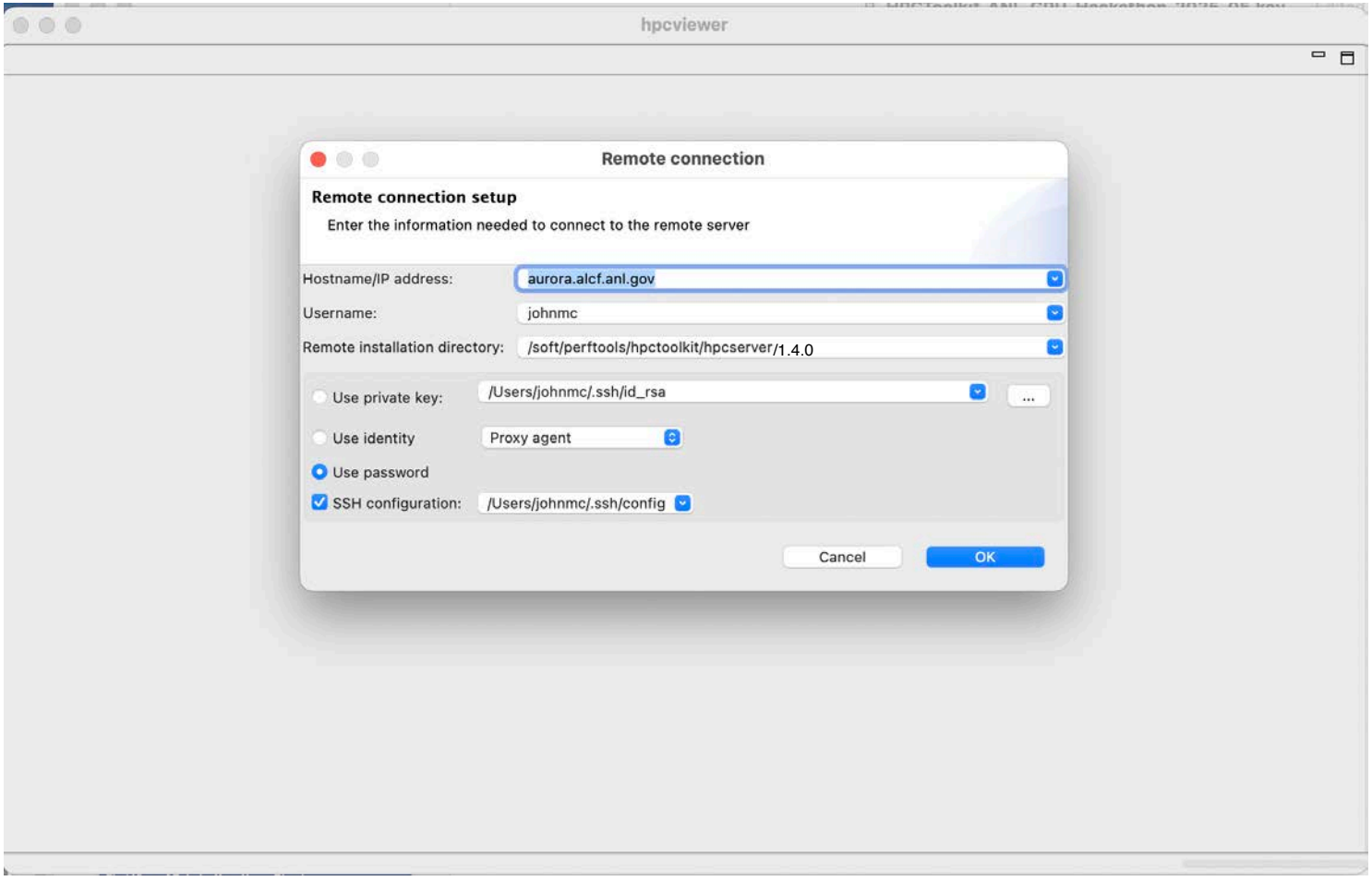
Authenticate using your token as you normally do

Navigate to a database with the file chooser in /flare/ATPESC2026/EXAMPLES/hpctoolkit/DONT-COPY/data

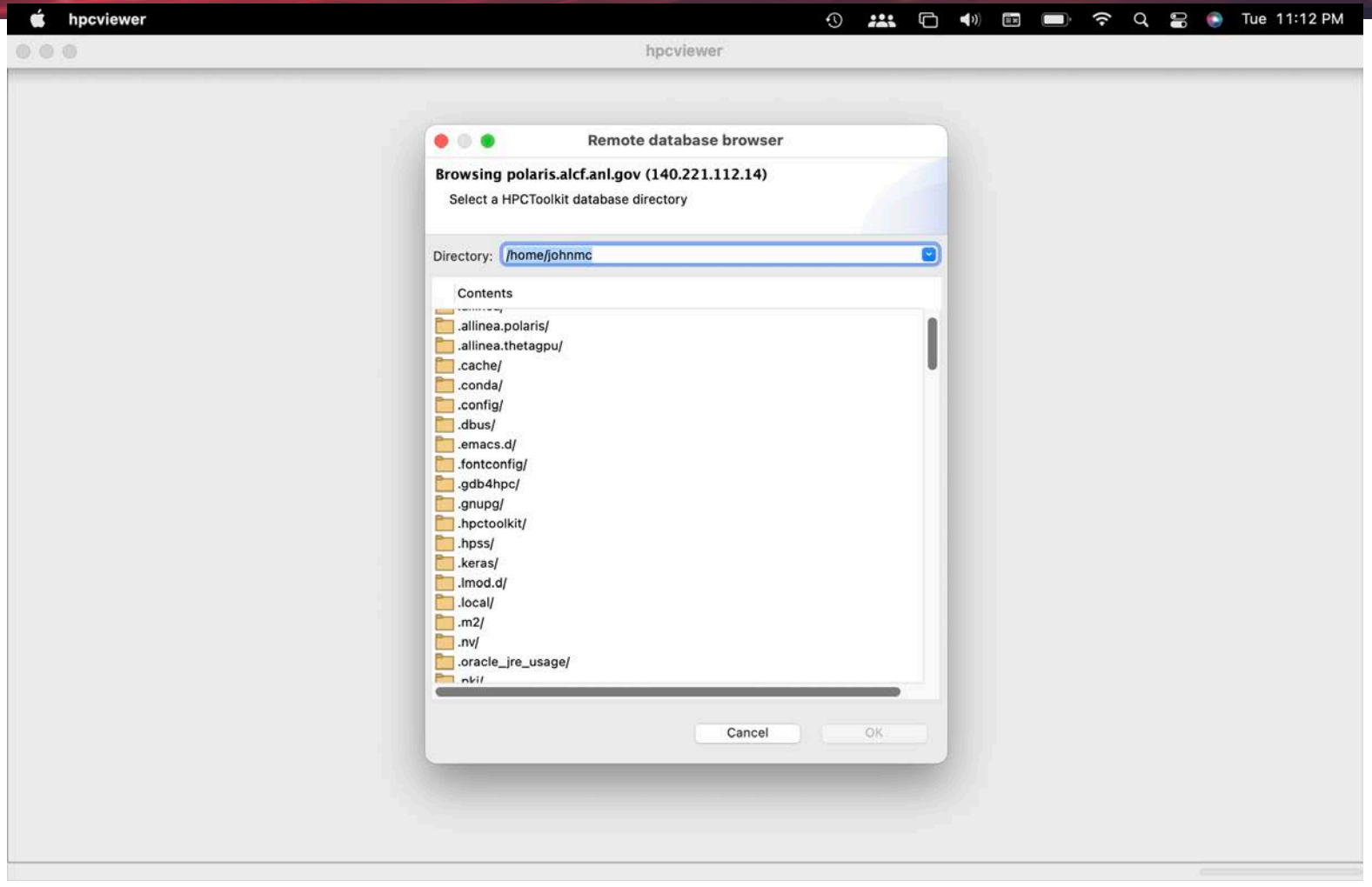
Opening a Remote Database



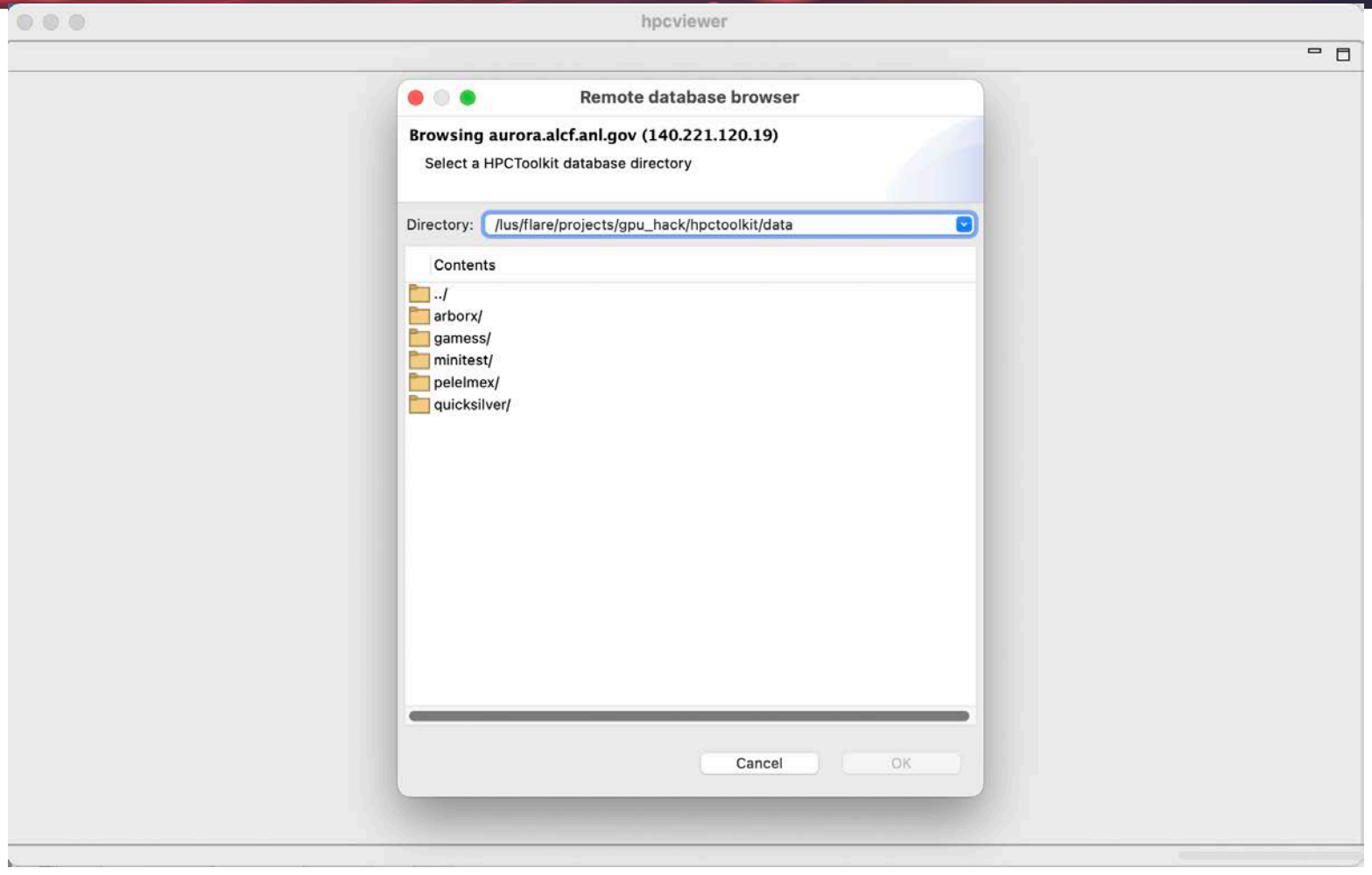
Configuring for remote access to Aurora using hpcserver



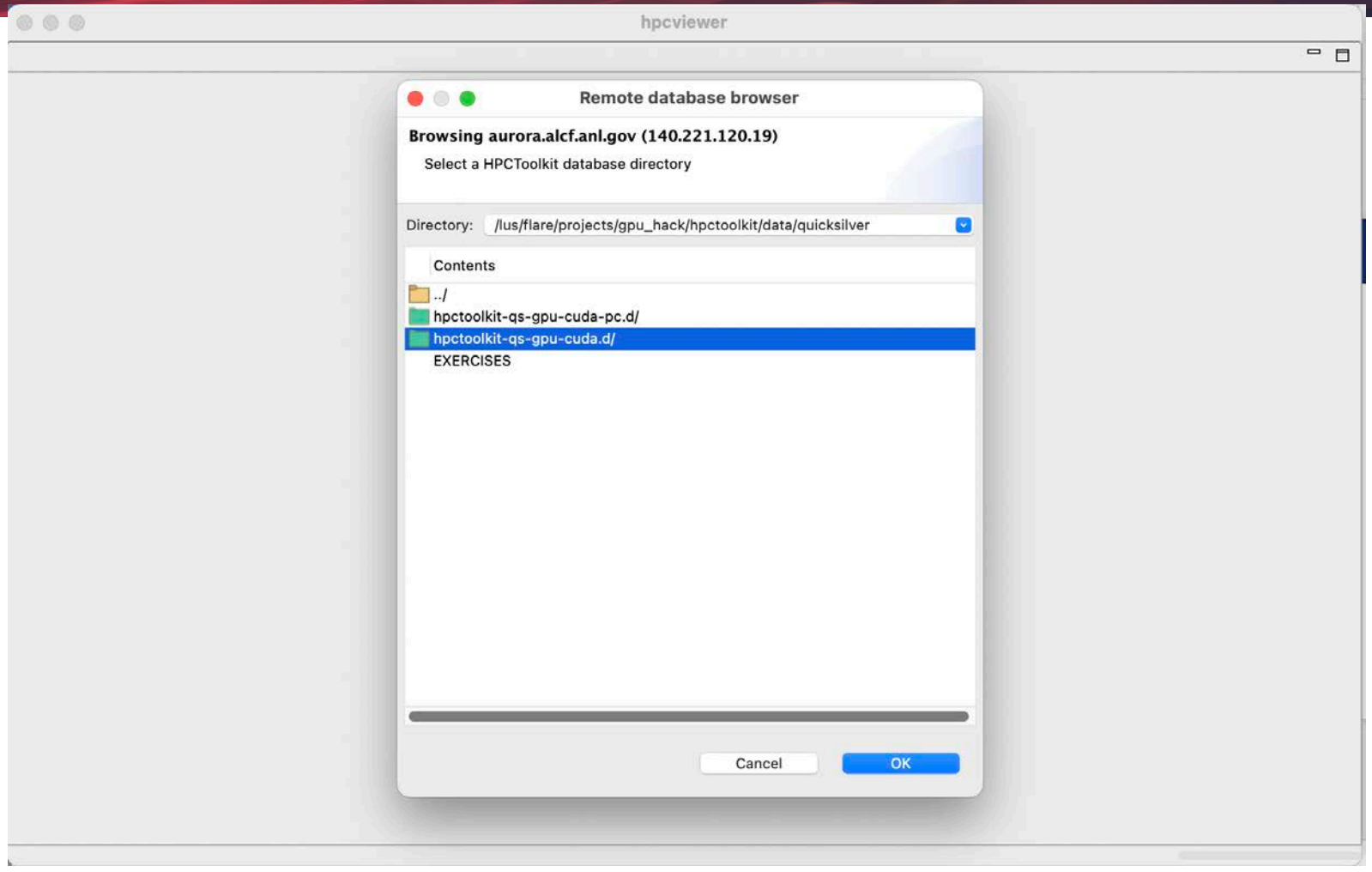
First View of Aurora: Your Home Directory



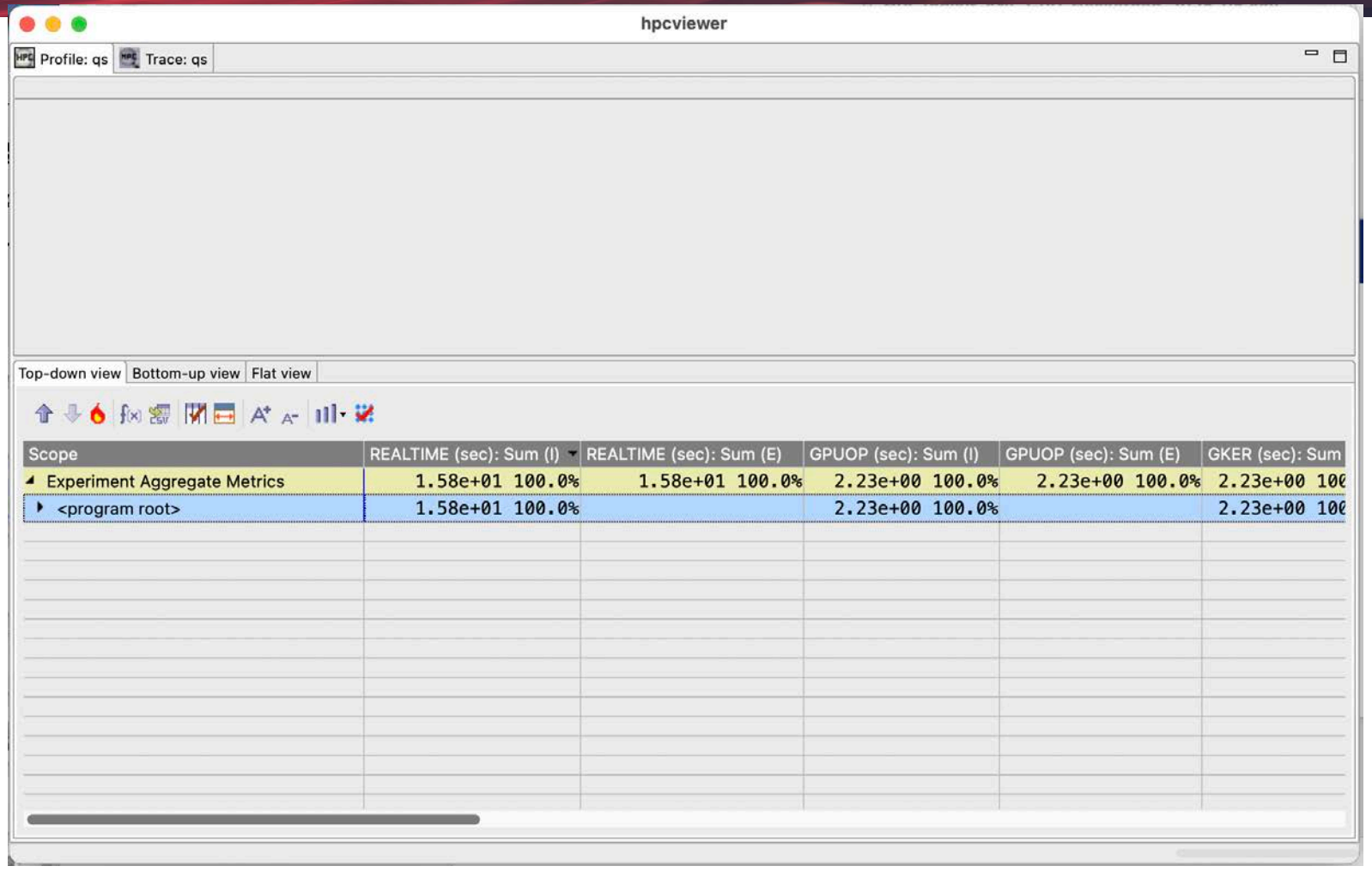
Navigate to Example Databases



Select a Quicksilver Database with Traces



After Selecting hpctoolkit-qs-gpu-cuda.d



Troubleshooting Tips

Why can’t I see my Source Code?

- To relate performance measurements to your application source code, the code must be compiled with a “-g” option in addition to your optimization flags. Otherwise, there is no information for any tool to map performance to anything finer grain than at the procedure level
  - For instance, if you are building with make, you will want to build **RelWithDebInfo** rather than **Release** for the best experience

ARGONNE TRAINING PROGRAM ON EXTREME-SCALE COMPUTING

Produced by Argonne National Laboratory, a U.S. Department of Energy Laboratory managed by UChicagoArgonne, LLC under contract DE-AC02-06CH11357.

Special thanks to the National Energy Research Scientific Computing Center (NERSC) and Oak Ridge Leadership Computing Facility (OLCF) for the use of their resources during the training event.