ARGONNE
ATPESC2024
EXTREME-SCALE COMPUTING

Performance Analysis of GPU-accelerated Applications with HPCToolkit

John Mellor-Crummey
Professor, Rice University

Argonne NATIONAL LABORATORY

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
  - —GAMESS (OpenMP)
  - —Deepwave (Pytorch)
  - —Quicksilver (CUDA)
  - —PeleC (AMReX)
  - —LAMMPS at Exascale (Kokkos)
- Coming attractions
- Hands-on with HPCToolkit
  - Installing HPCToolkit’s hpcviewer graphical user interface on your laptop
  - Analyzing executions with HPCToolkit’s hpcviewer user interface
  - Collecting performance measurements for sample programs (or your own code)
- Troubleshooting

Linux Foundation's HPCToolkit Performance Tools

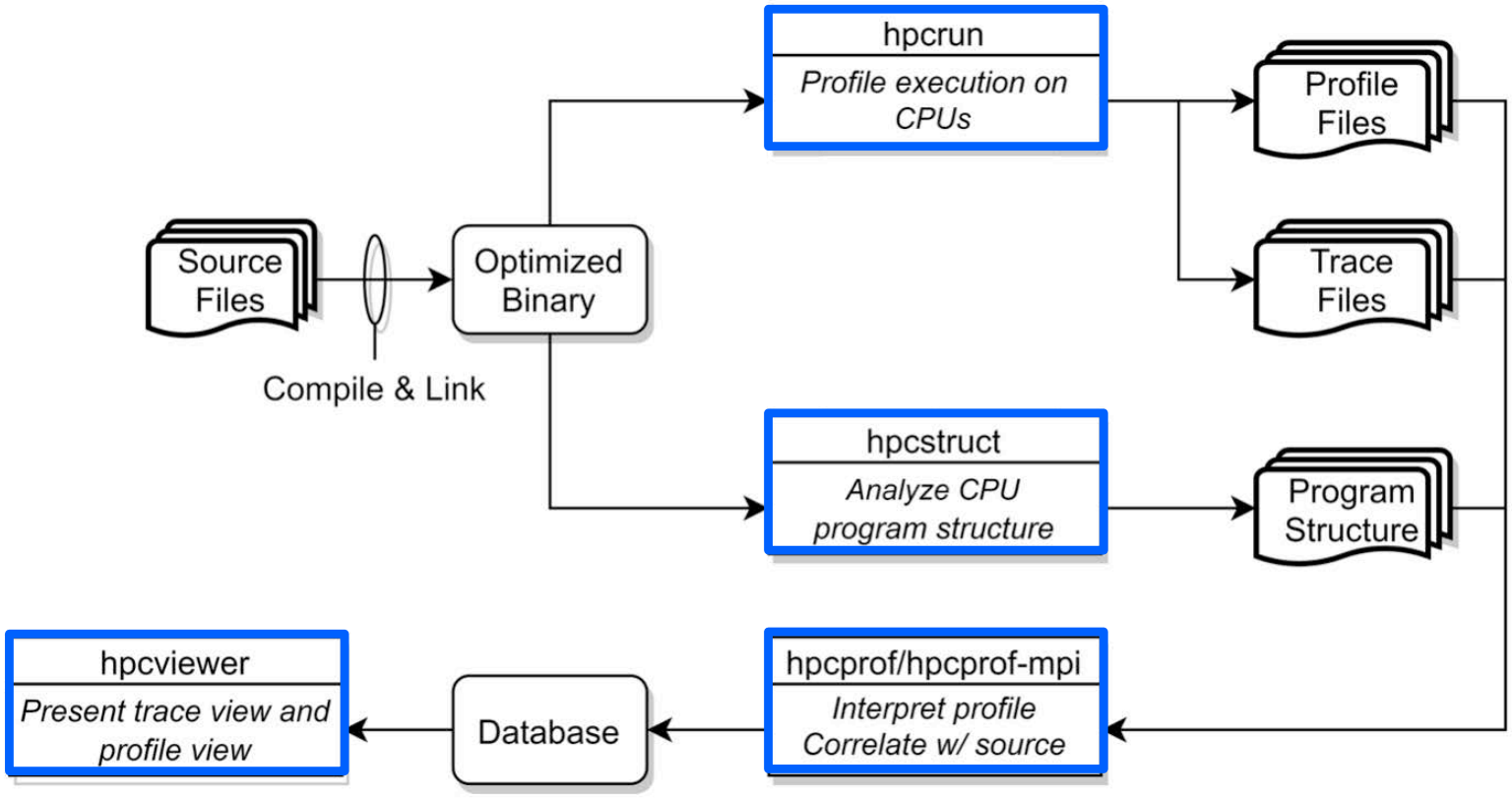
Measure and analyze performance of CPU and GPU-accelerated applications

- 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
- Measures complex programs on a broad range of platforms
  - —CPU: x86_64, Power, ARM
  - —GPU: NVIDIA, AMD, Intel

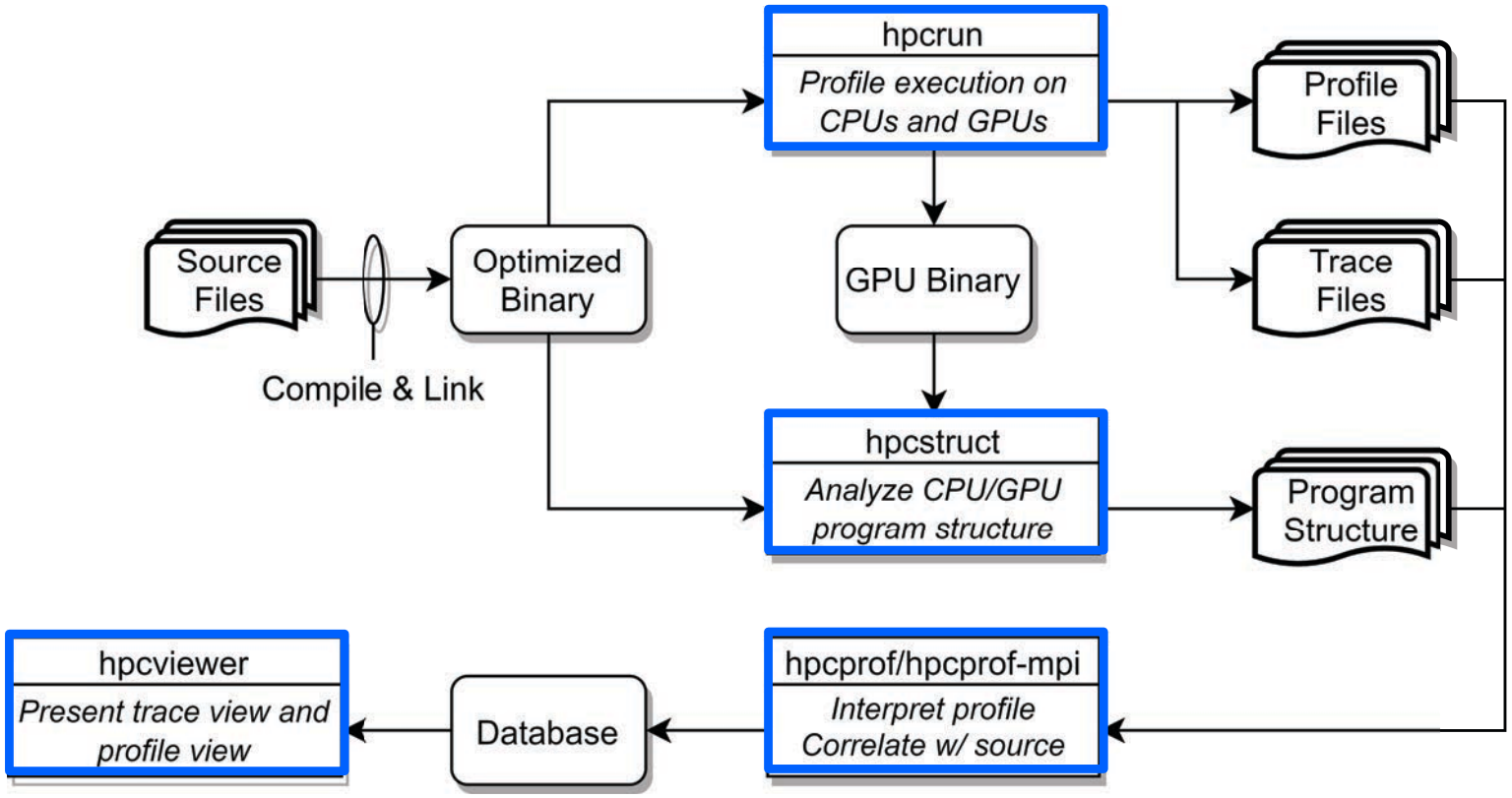
How does HPCToolkit Differ from NVIDIA's Tools?

- 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
- HPCToolkit
  - —supports more scalable tracing than Nsight Systems
    - measure exascale executions across many GPUs and nodes
  - —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 CPU Applications



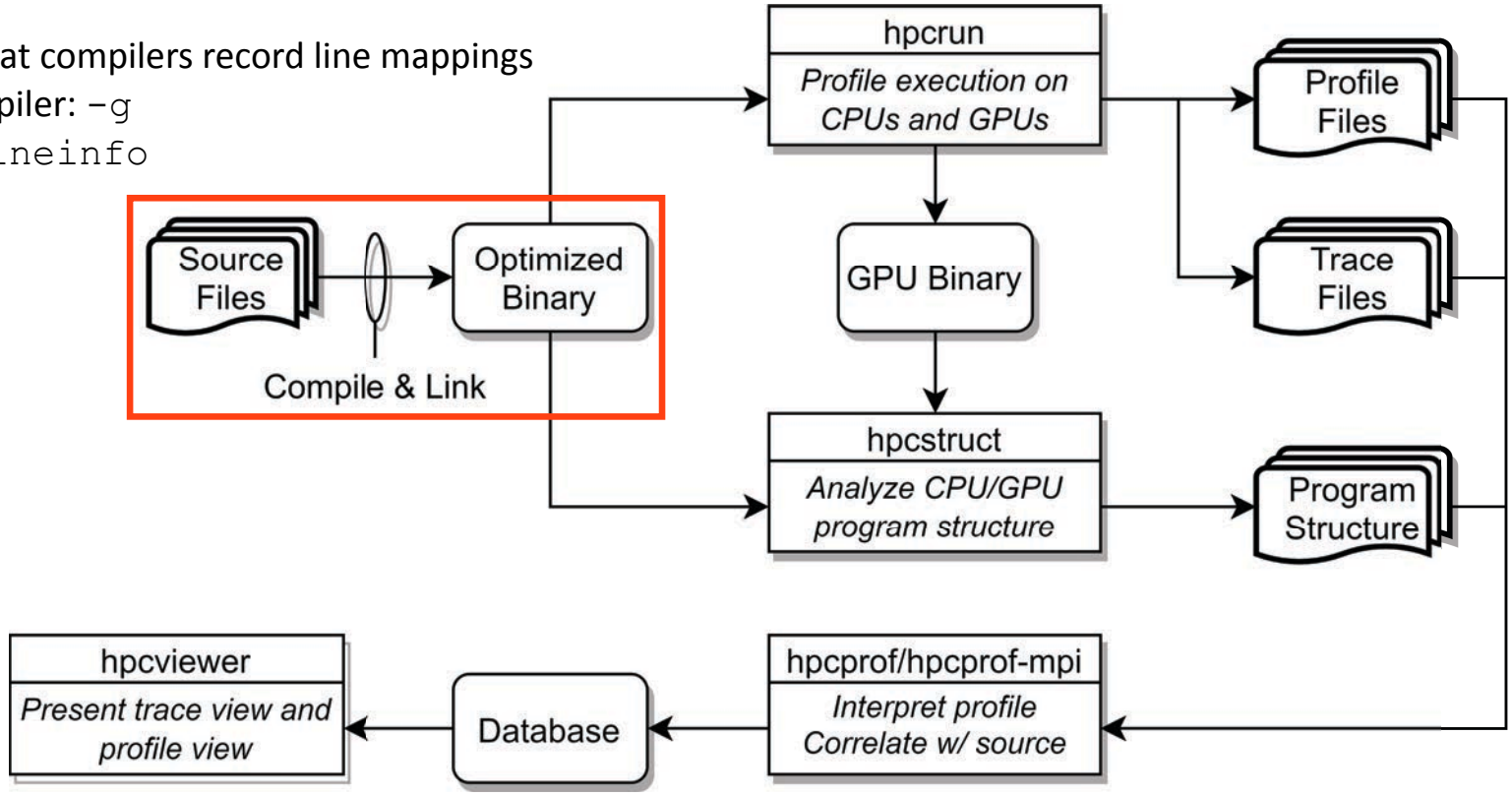
HPCToolkit's Workflow for GPU-accelerated Applications



HPCToolkit's Workflow for GPU-accelerated Applications

Step 1:

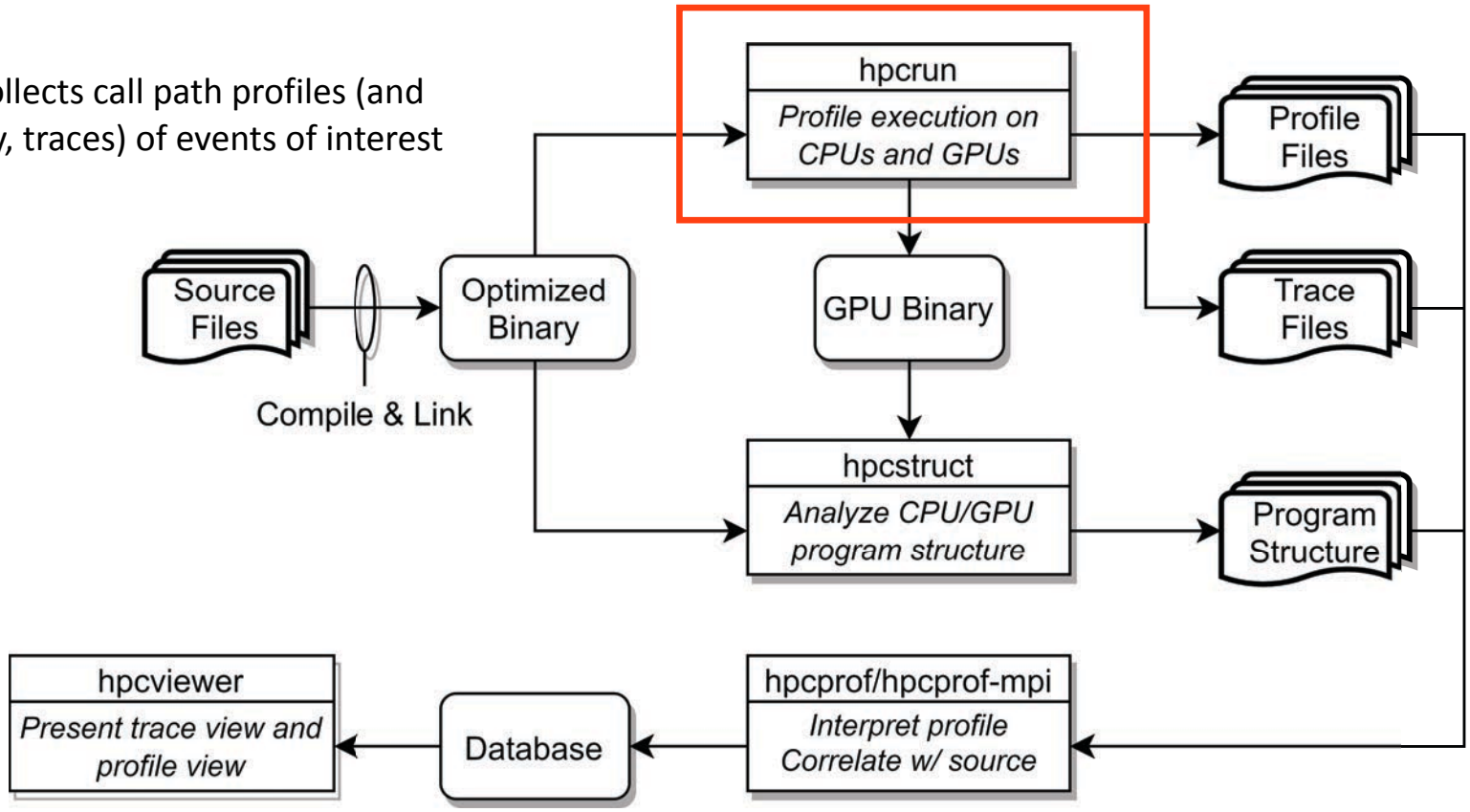
- Ensure that compilers record line mappings
- host compiler: `-g`
- nvcc: `-lineinfo`



HPCToolkit's Workflow for GPU-accelerated Applications

Step 2:

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



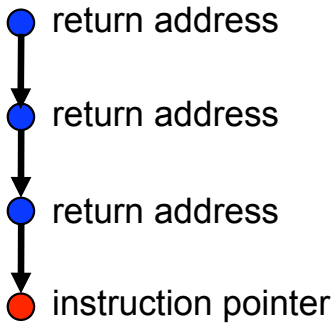
Measurement of CPU and GPU-accelerated Applications

- 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; PC Samples (NVIDIA, AMD, Intel)
- Binary instrumentation of GPU kernels on Intel GPUs for fine-grain measurement

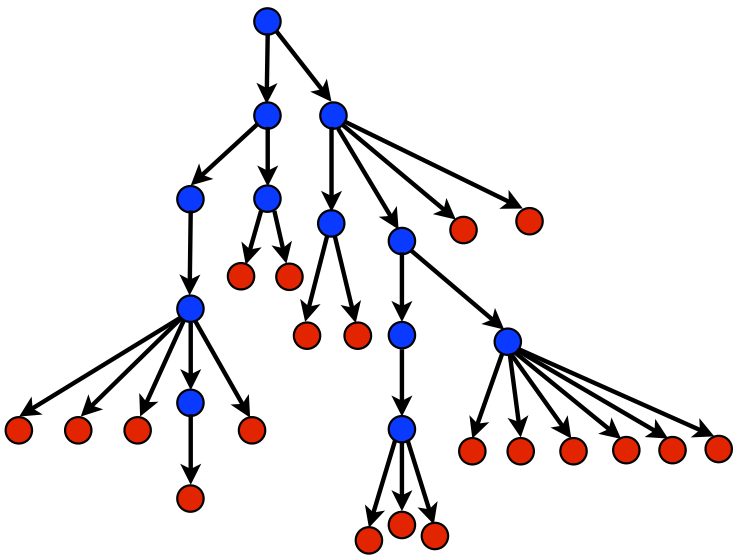
Call Stack Unwinding to Attribute Costs in Context

- 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

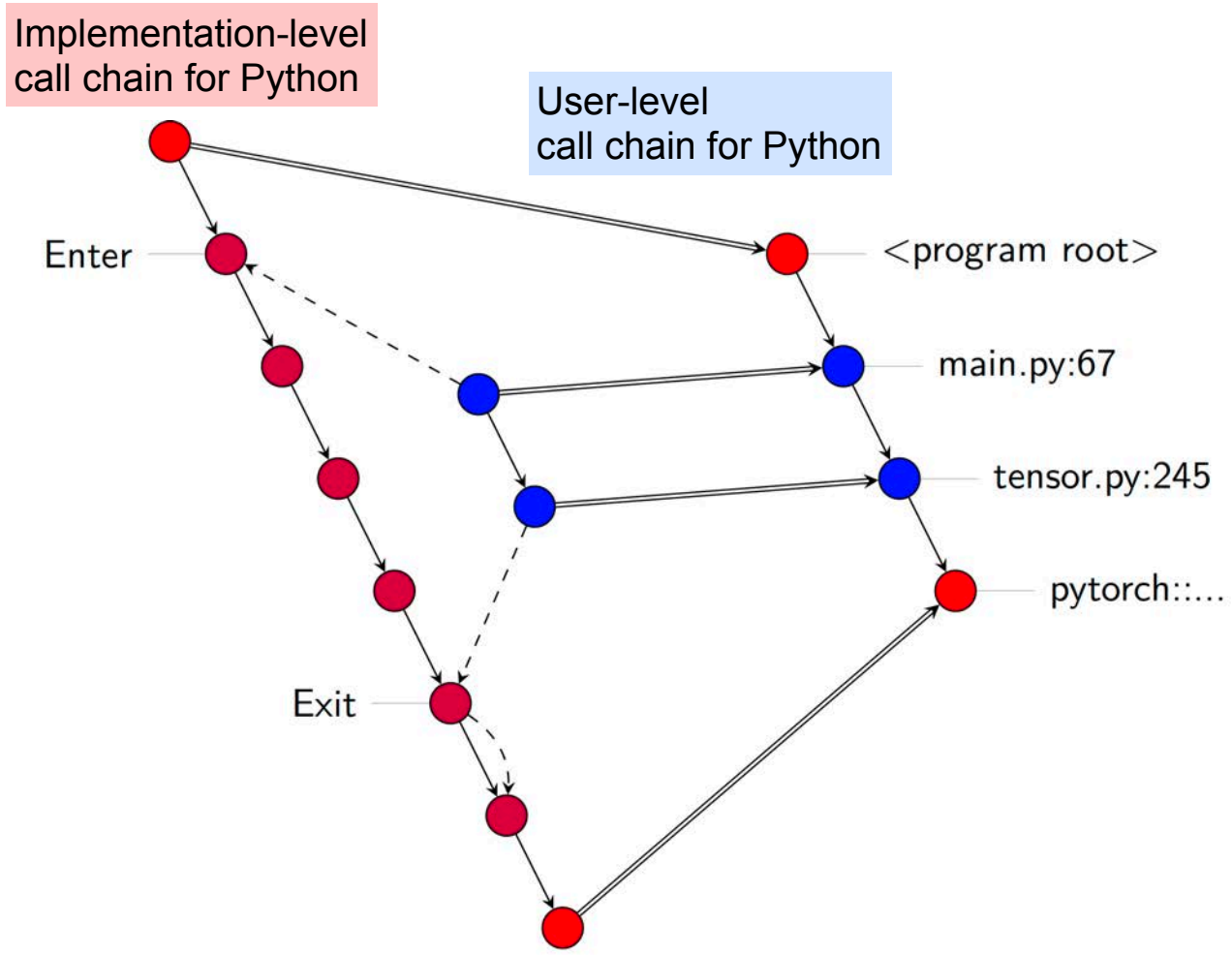
- GPU profiling
  - `hpcrun -e gpu=xxx <app> ….`

  xxx ∈ *{nvidia,amd,opencl,level0}*
- GPU PC sampling (NVIDIA GPU only)
  - `hpcrun -e gpu=nvidia,pc <app>`
- CPU and GPU Tracing (in addition to profiling)
  - `hpcrun -e CPUTIME -e gpu=xxx -t <app>`
  - `hpcrun -e CPUTIME -e gpu=xxx -tt <app>` # boosted resolution CPU traces for GPU tracing
- Use hpcrun with MPI on Polaris
  - `mpiexec -n <ranks> … hpcrun -e gpu=xxx <app>`

Measure and Attribute Performance of Python-driven Codes

Challenge: Straightforward sampling-based approach attributes metrics to Python interpreter code rather than application-level Python source code

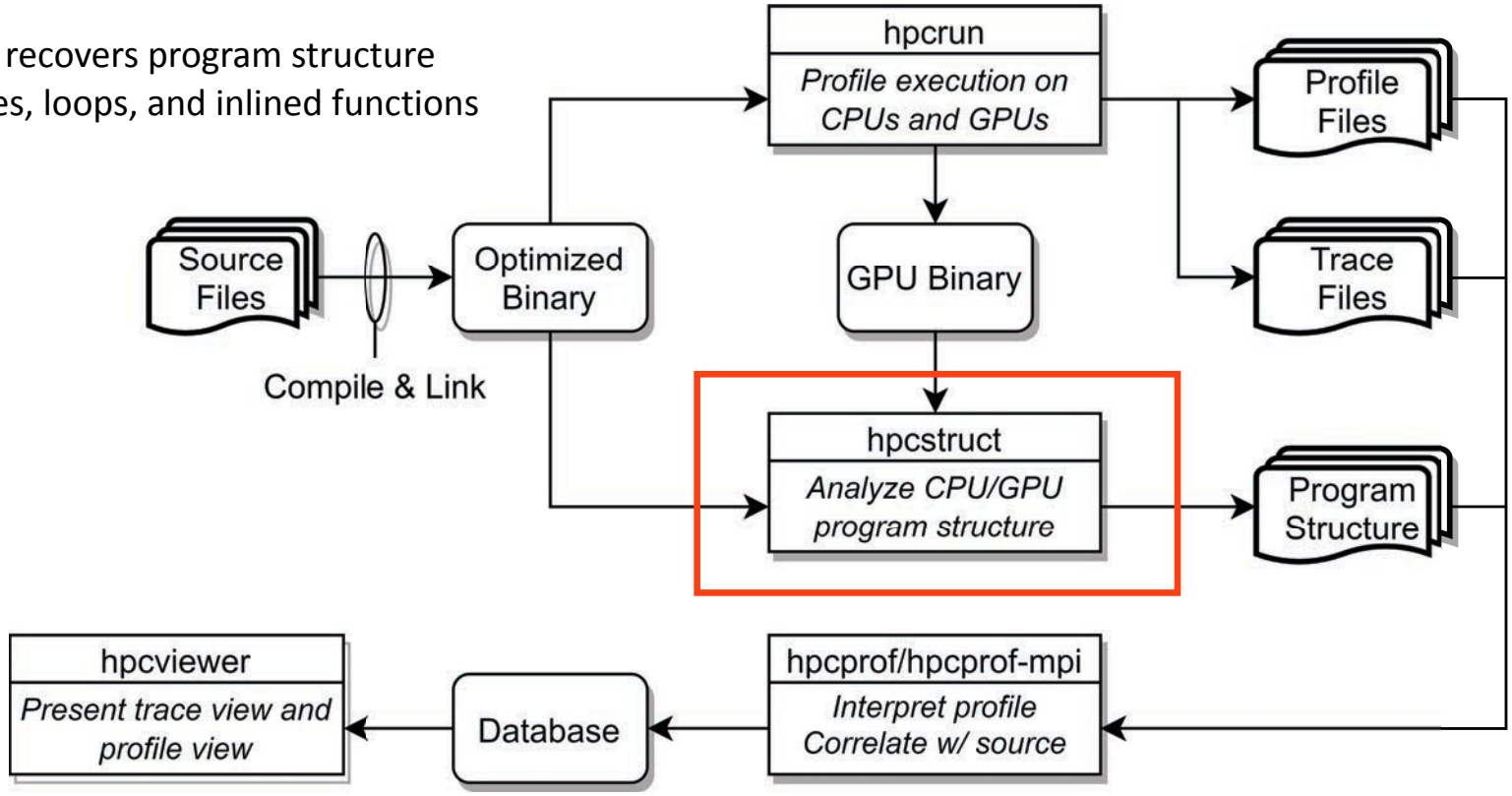
Approach: Develop approach to map implementation-level measurements back to Python source code



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>
```

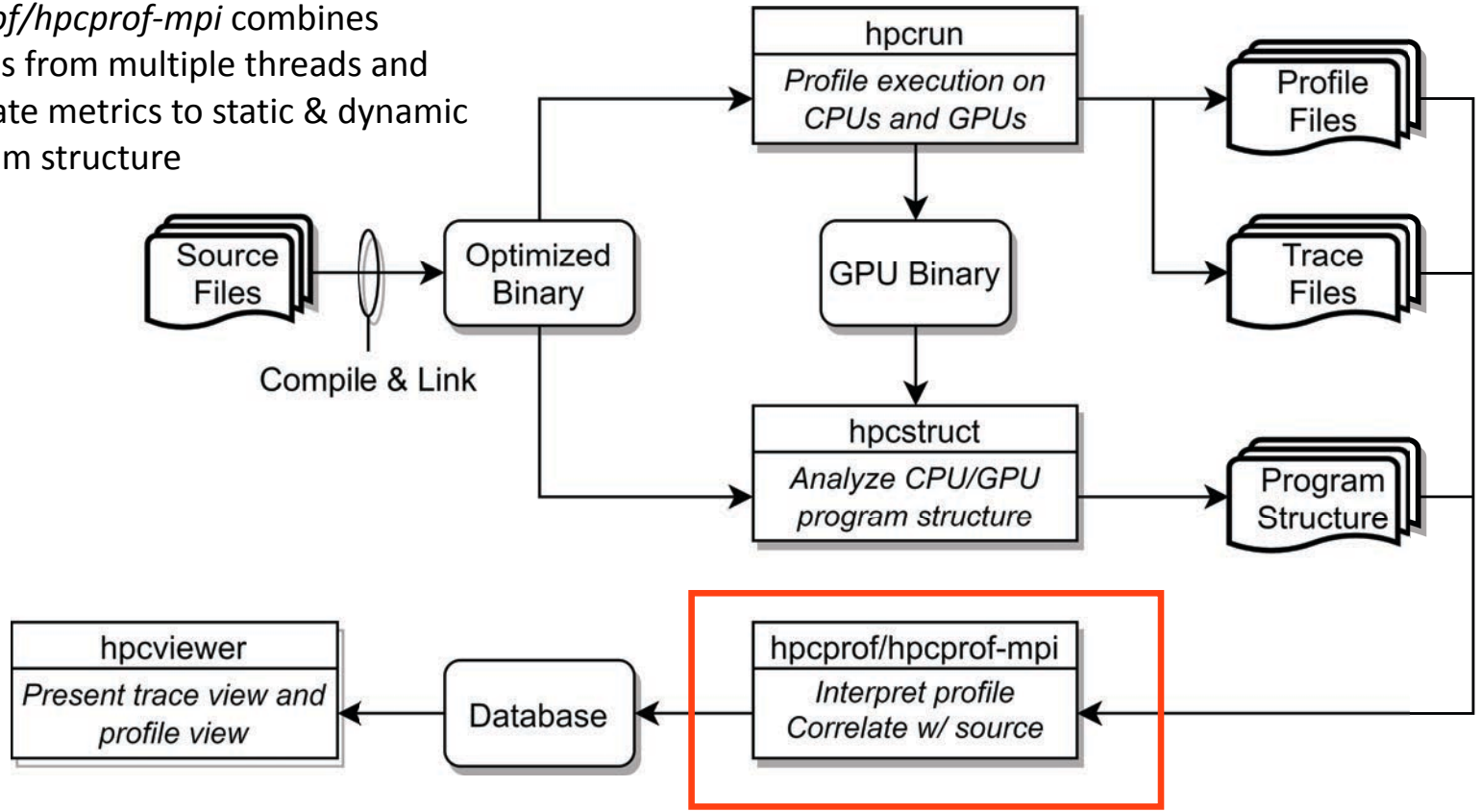
- What it does
  - Recover program structure information
    - Files, functions, inlined templates or functions, loops, source lines
  - In parallel, analyze all CPU and GPU binaries that were measured by HPCToolkit
    - —analyze large application binaries with 16 threads
    - —analyze multiple small application binaries concurrently with 2 threads each
  - 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>
```

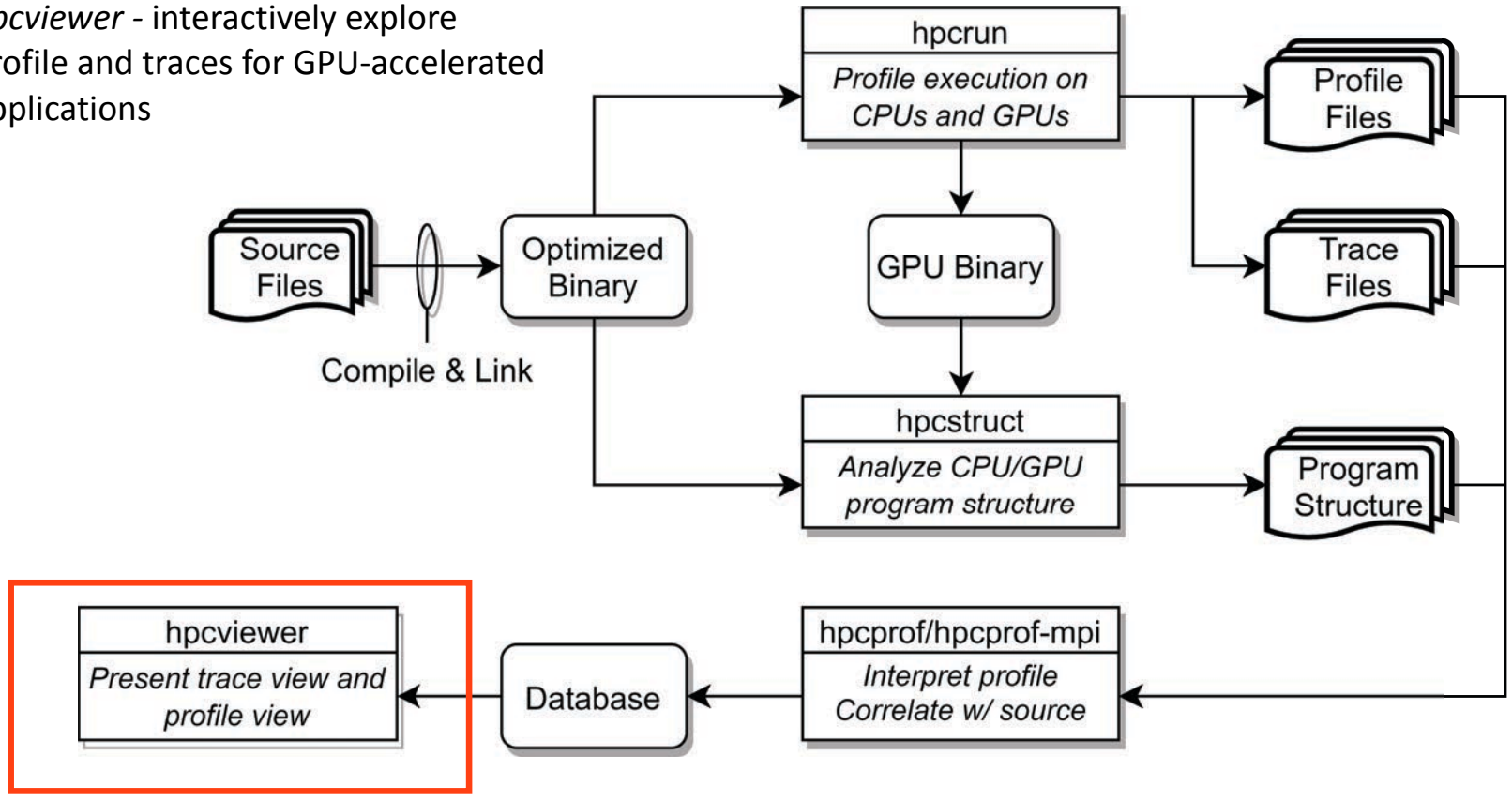
- Analyze data from large executions with distributed-memory parallelism + multithreading (large scale)

```
mpiexec -n ${NODES} --ppn 1 –depth=64 \
    hpcprof-mpi <measurement-directory>
```

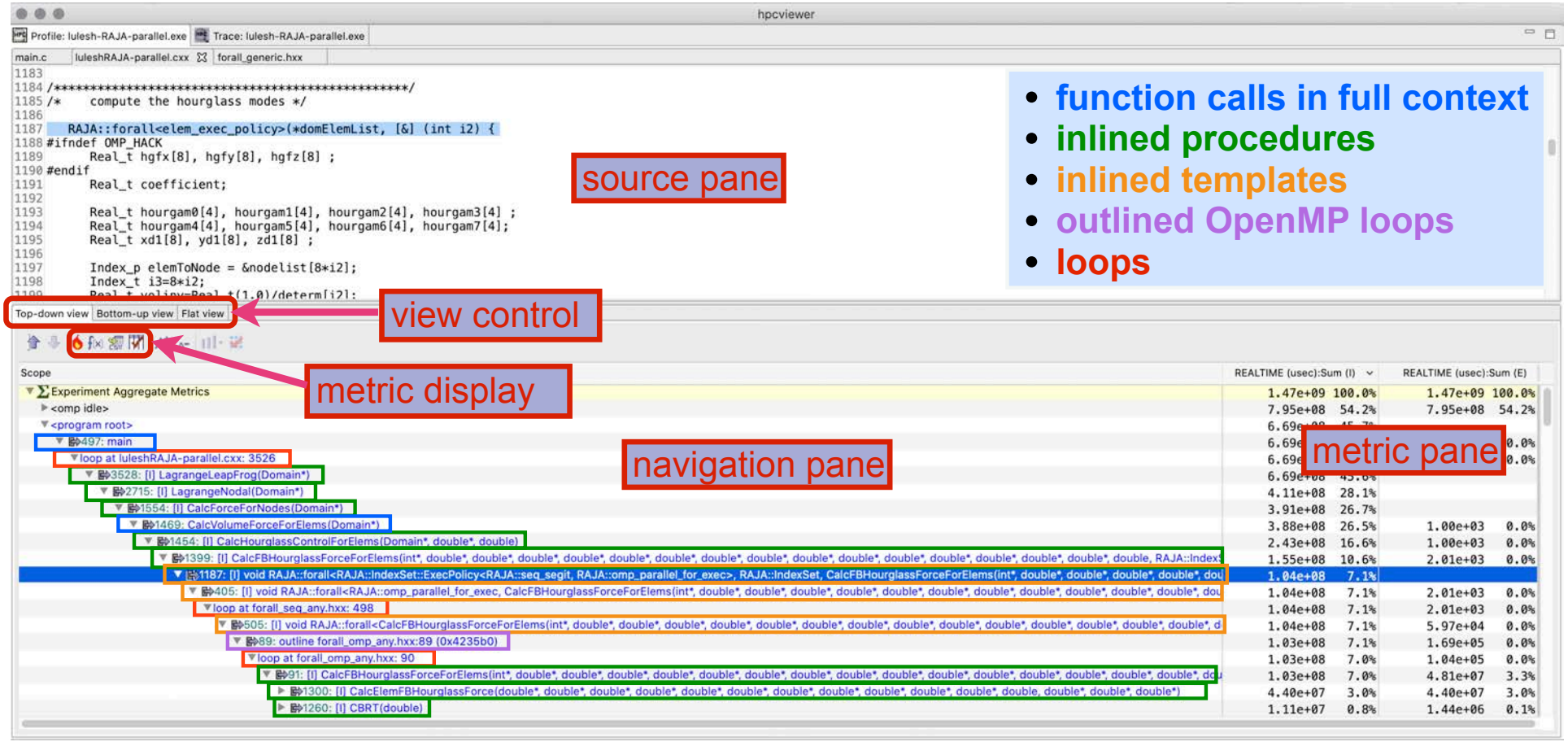
HPCToolkit's Workflow for GPU-accelerated Applications

Step 4:

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

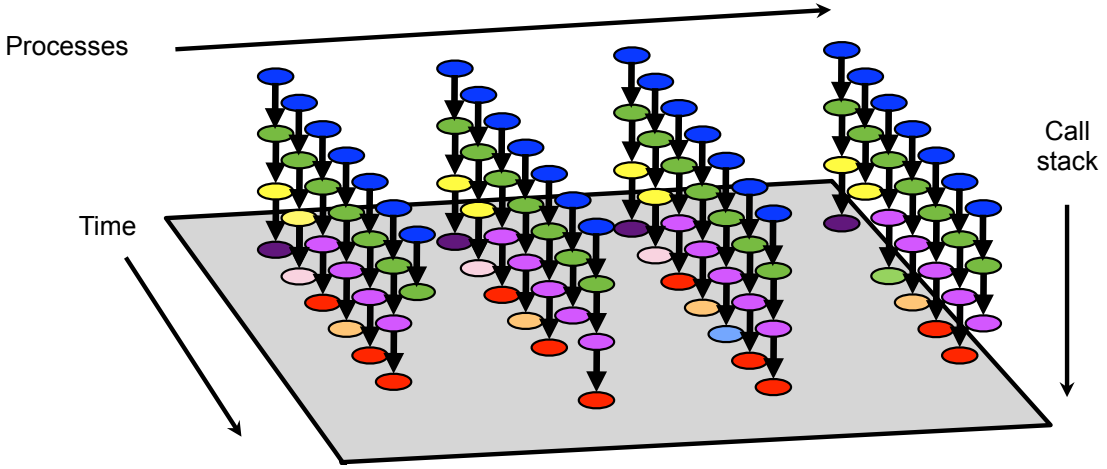


Code-centric Analysis with hpcviewer

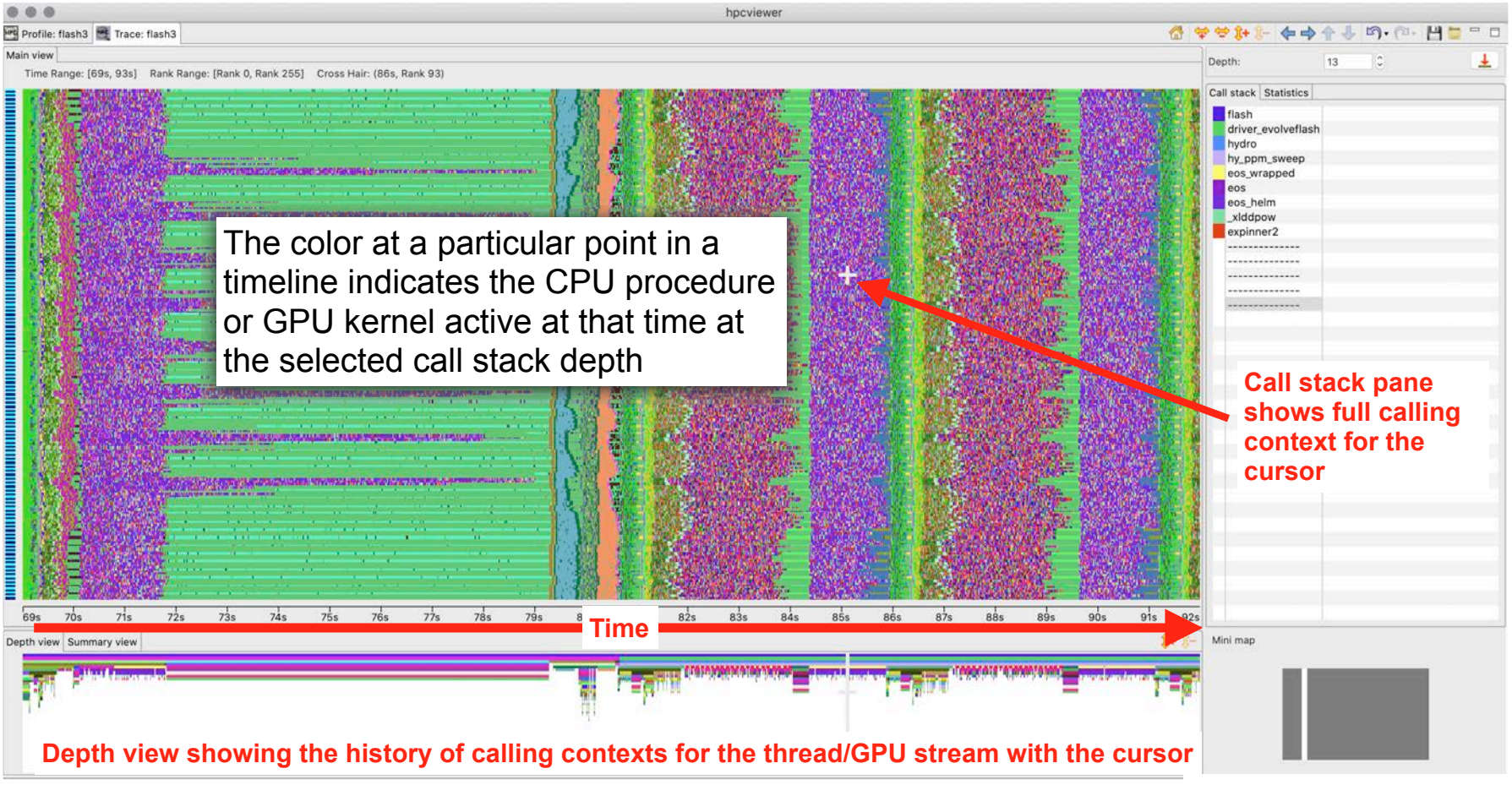


Understanding Temporal Behavior

- 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



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

What Can you Measure?

Parallel programming models

- Across nodes: MPI, SHMEM, UPC++,
- Within nodes: OpenMP, Kokkos, RAJA, HIP, DPC++, Sycl, CUDA, OpenACC

Languages

- C, C++, Fortran, Python

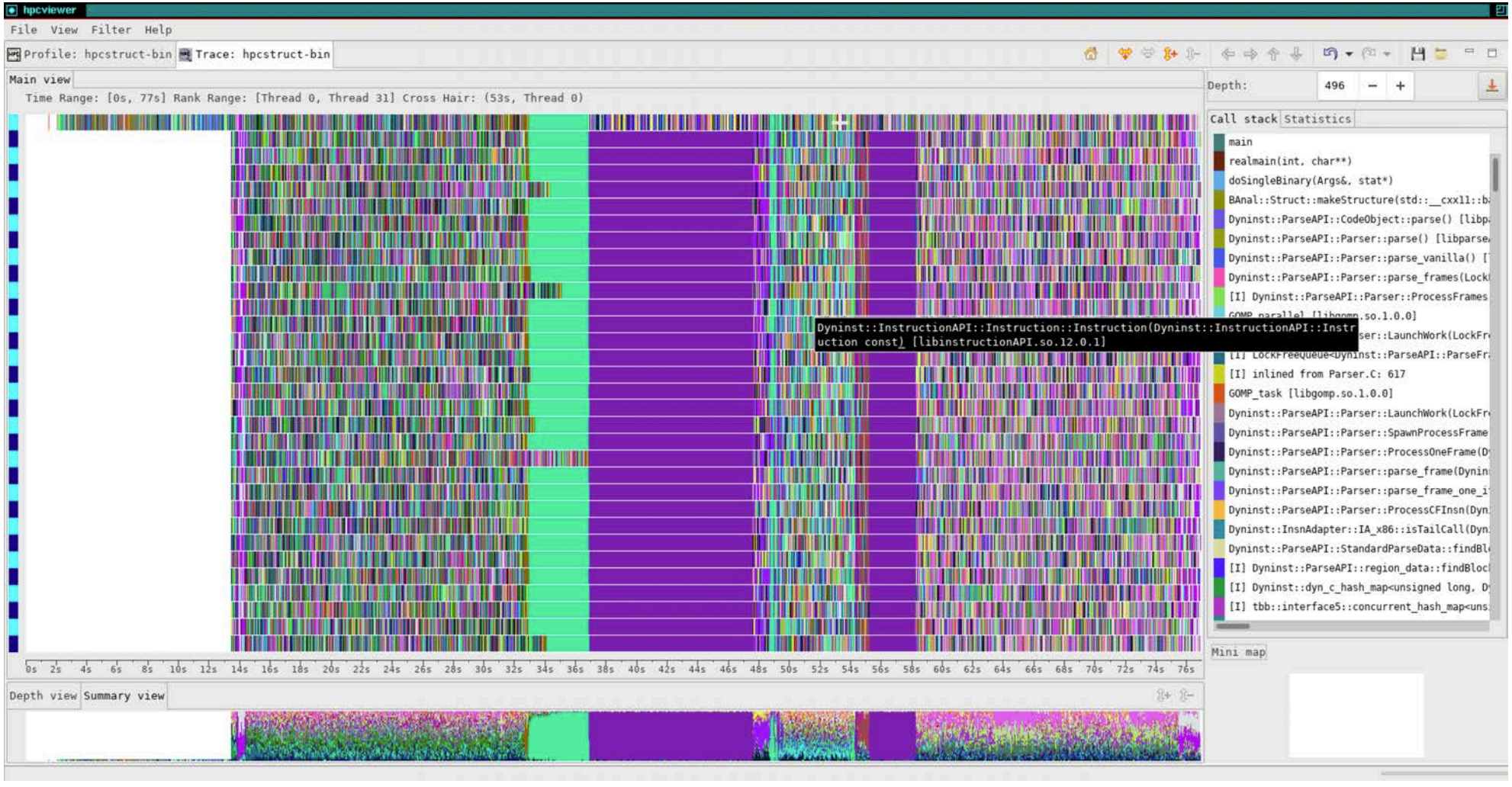
Frameworks

- Pytorch, Tensorflow (maybe)

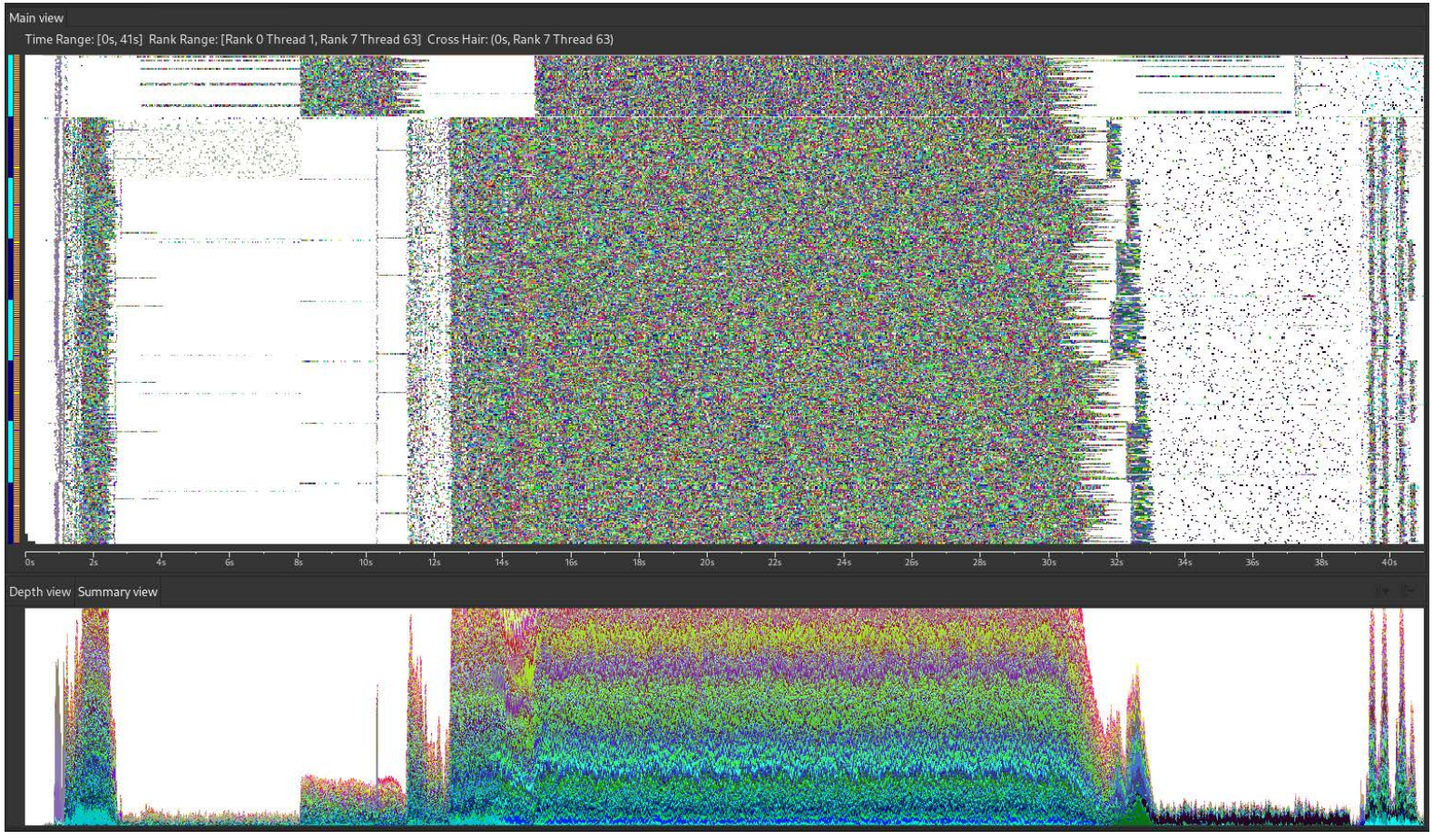
Hardware

- CPU cores and GPUs within a node
- All of the nodes in Polaris

hpcstruct Example: Analyze 7.7GB TensorFlow library (170MB text) in 77s



Analyze 38.1GB data for 2K MPI ranks + 2K GPUs using 1K threads in 41s



Case Studies

- GAMESS (OpenMP)
- Quicksilver (CUDA)
- PeleC (AMReX)
- LAMMPS (Kokkos) at exascale

Case Study: GAMESS

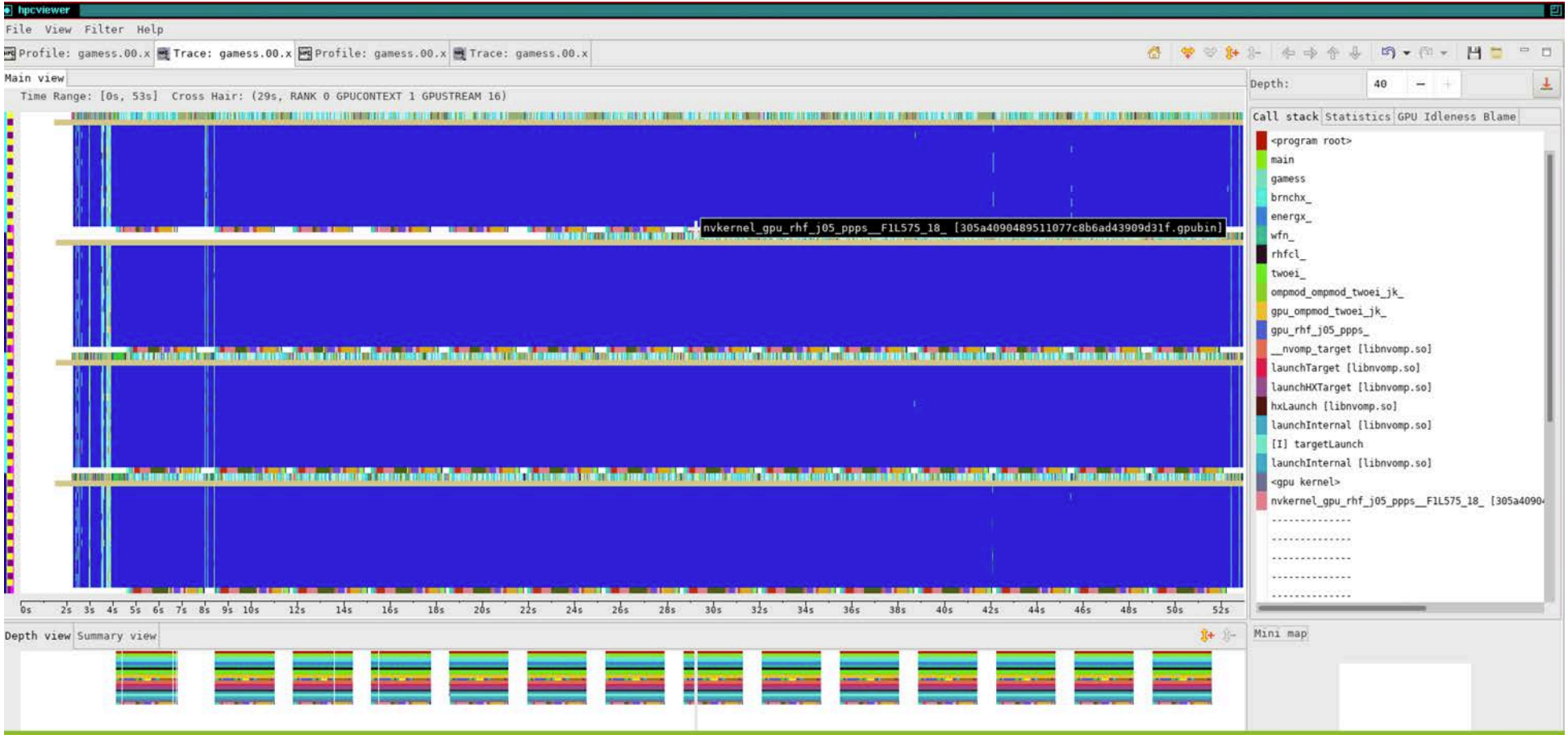
- General Atomic and Molecular Electronic Structure System (GAMESS)
  - —general *ab initio* quantum chemistry package
- Calculates the energies, structures, and properties of a wide range of chemical systems

- 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

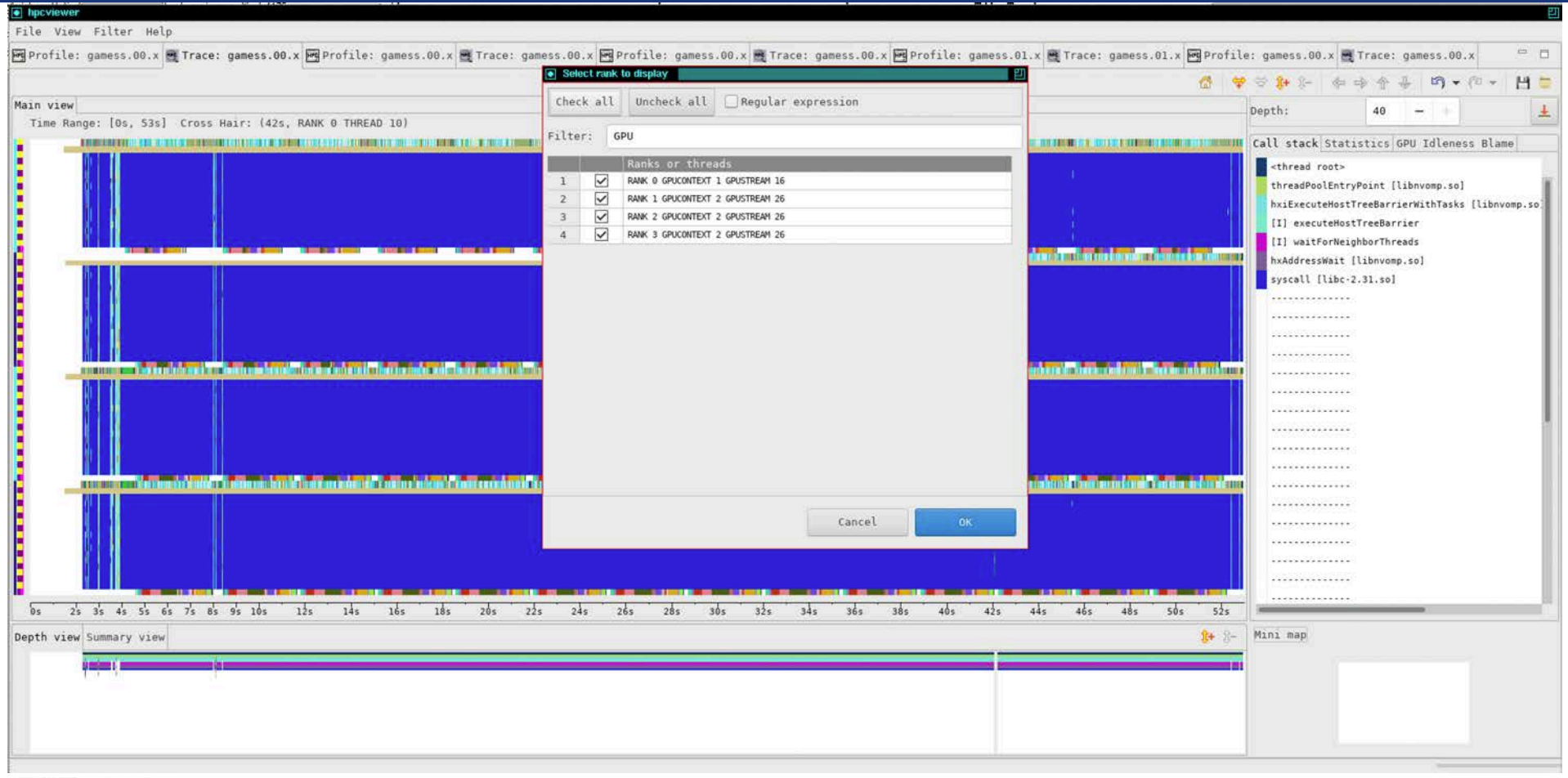
Time-centric Analysis: GAMESS 4 ranks, 4 GPUs on Perlmutter



GAMESS original

All CPU threads and GPU streams

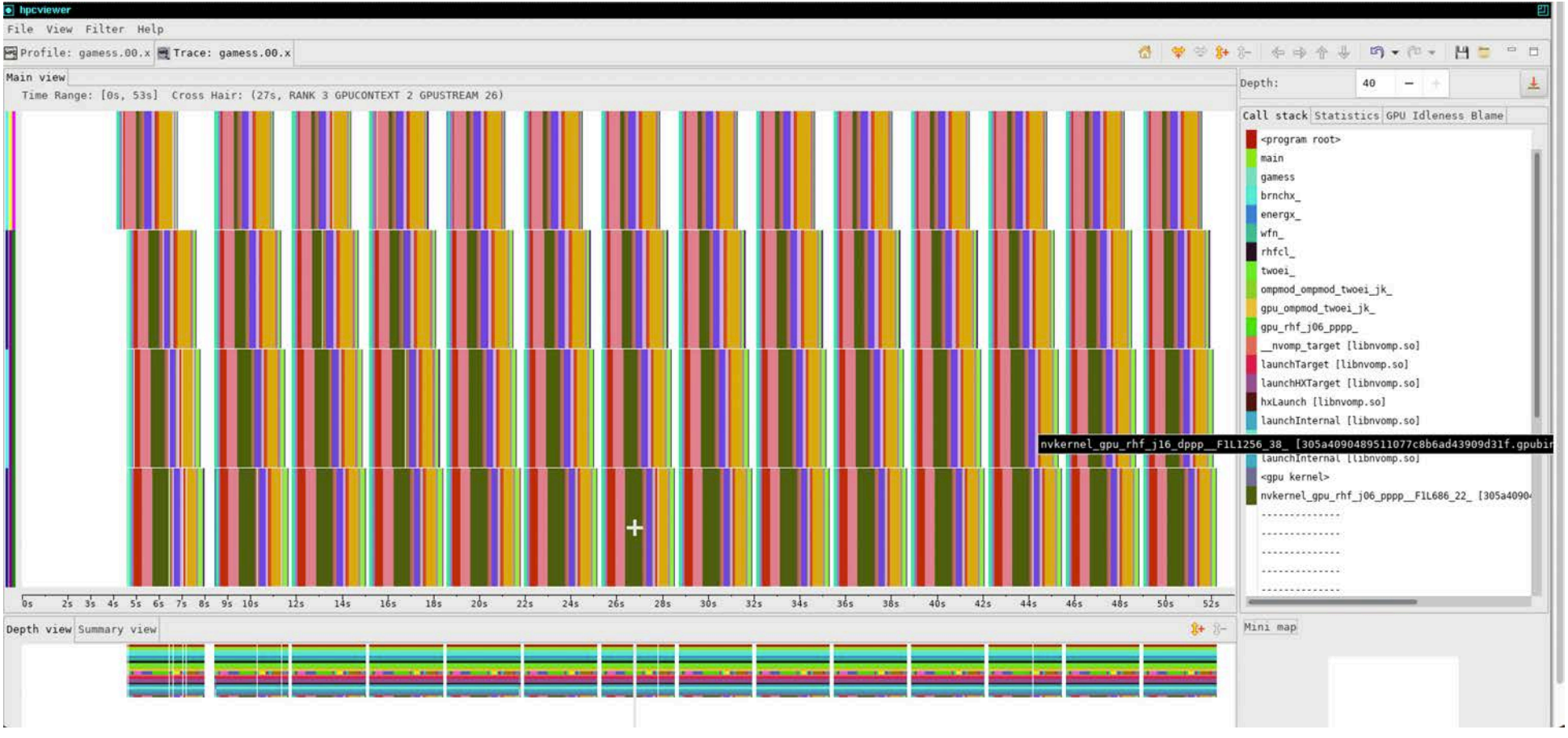
Time-centric Analysis: GAMESS 4 ranks, 4 GPUs on Perlmutter



GAMESS original

All CPU threads and GPU streams

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

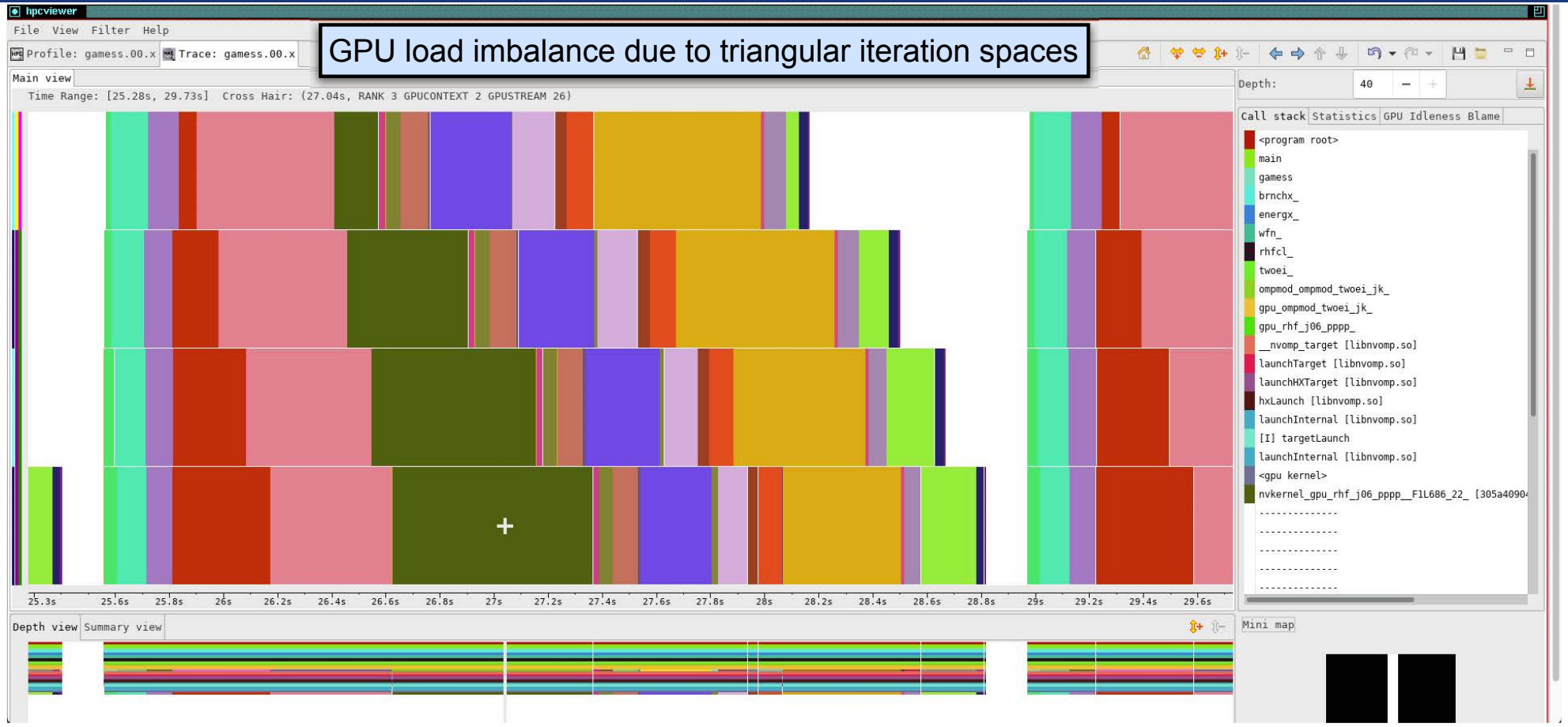


ARGONNE ATPESC2024 EXTREME-SCALE COMPUTING

GAMESS original

All GPU streams, whole execution

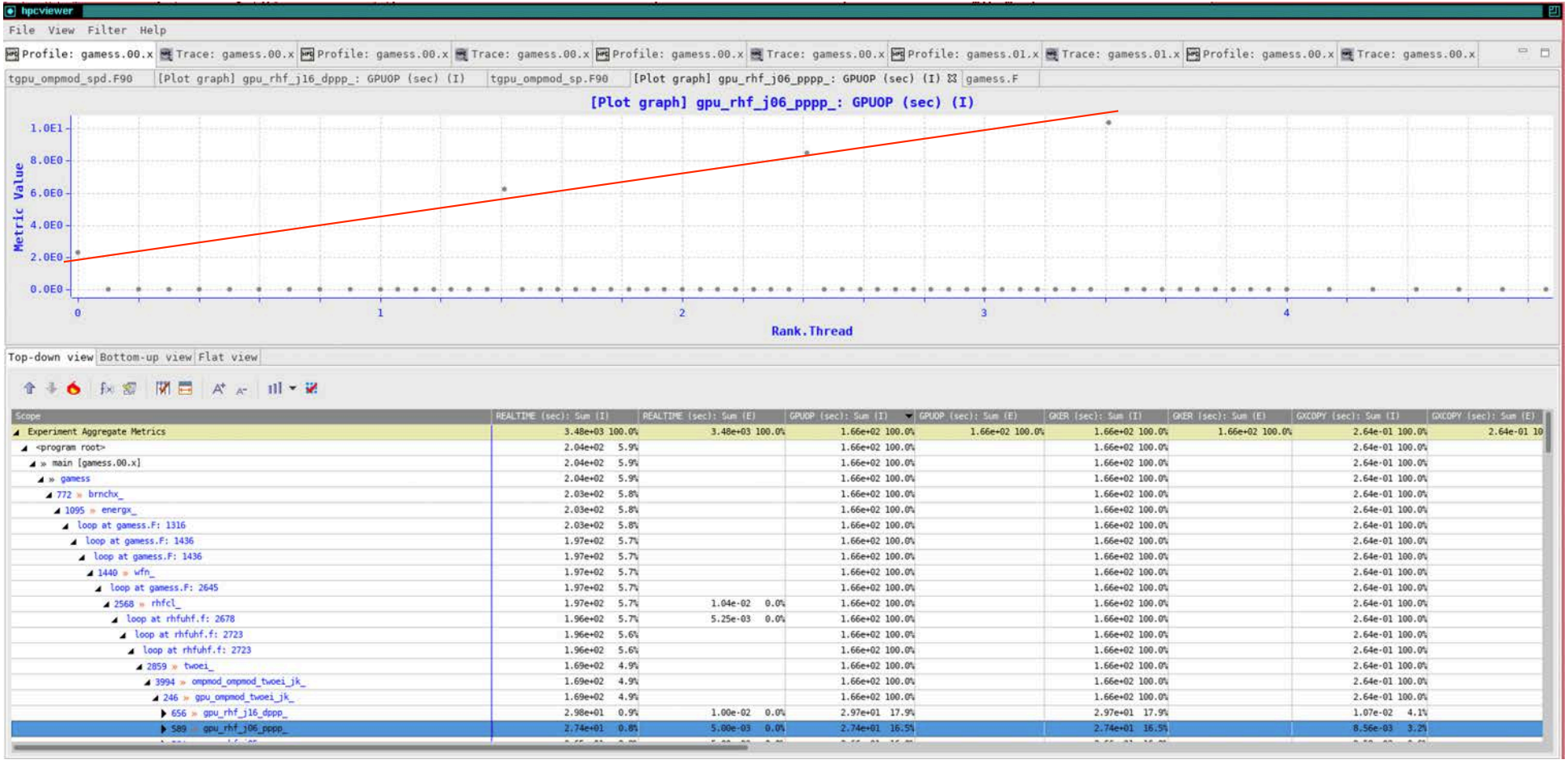
Time-centric Analysis: GAMESS 4 ranks, 4 GPUs on Perlmutter



GAMESS original

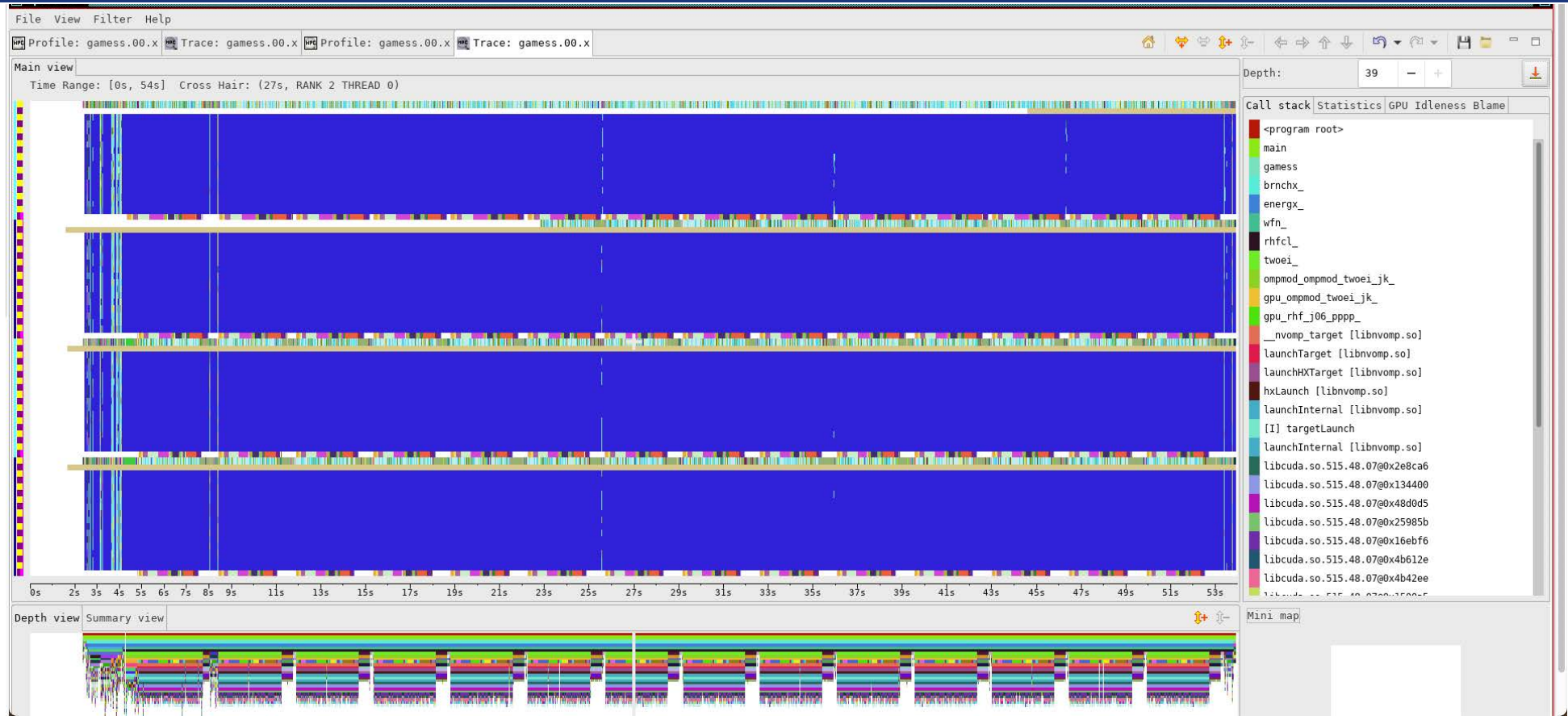
GPU streams: 1 iteration

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



GAMESS original

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

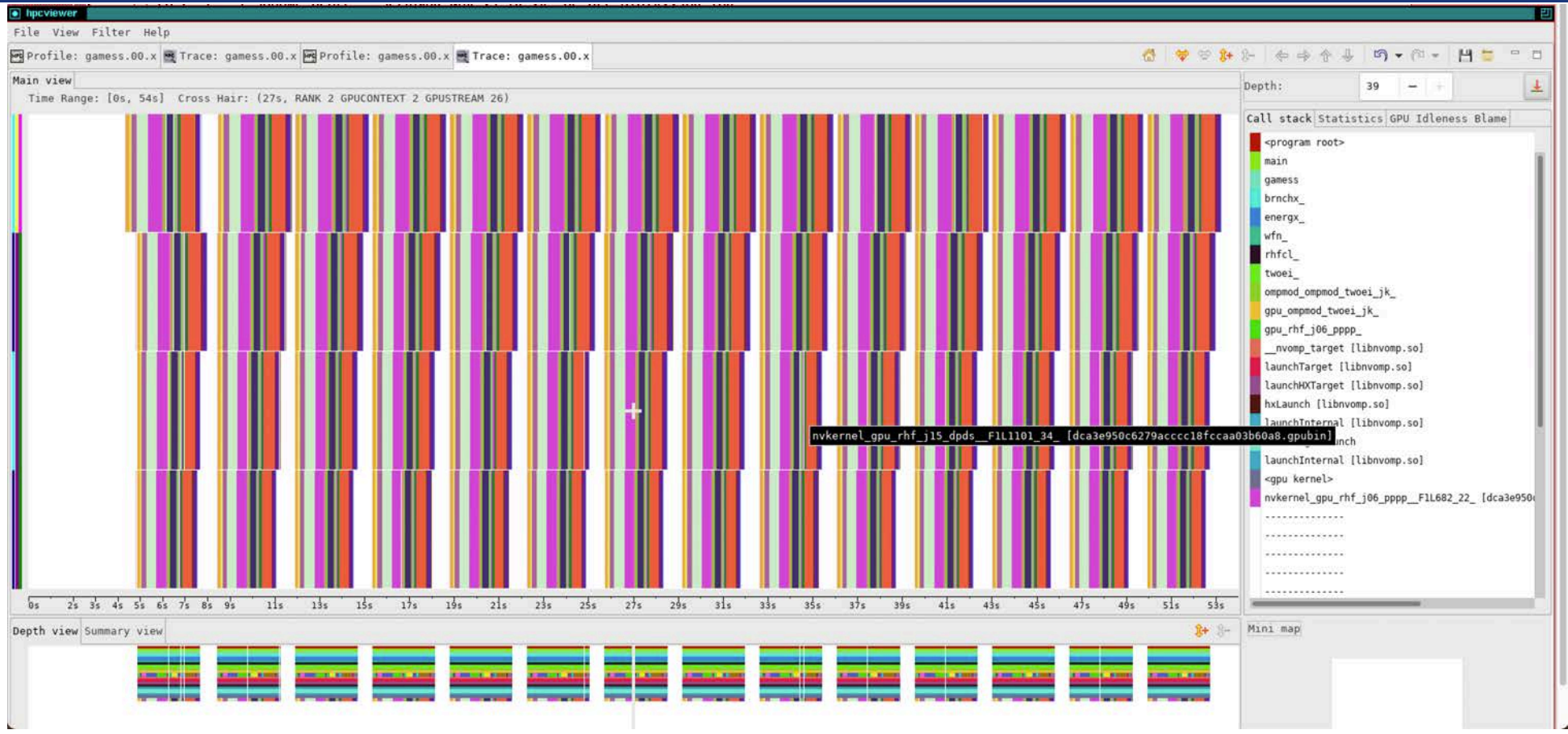


ARGONNE ATPESC2024 EXTREME-SCALE COMPUTING

GAMESS improved

All CPU threads and GPU streams

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



GAMESS improved

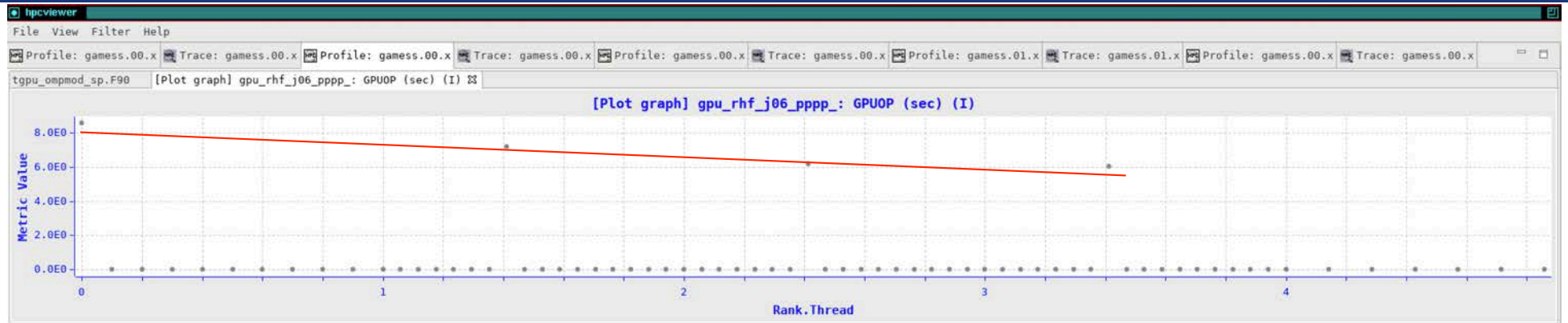
All GPU streams, whole execution

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

GAMESS improved

All GPU streams: 2 iterations

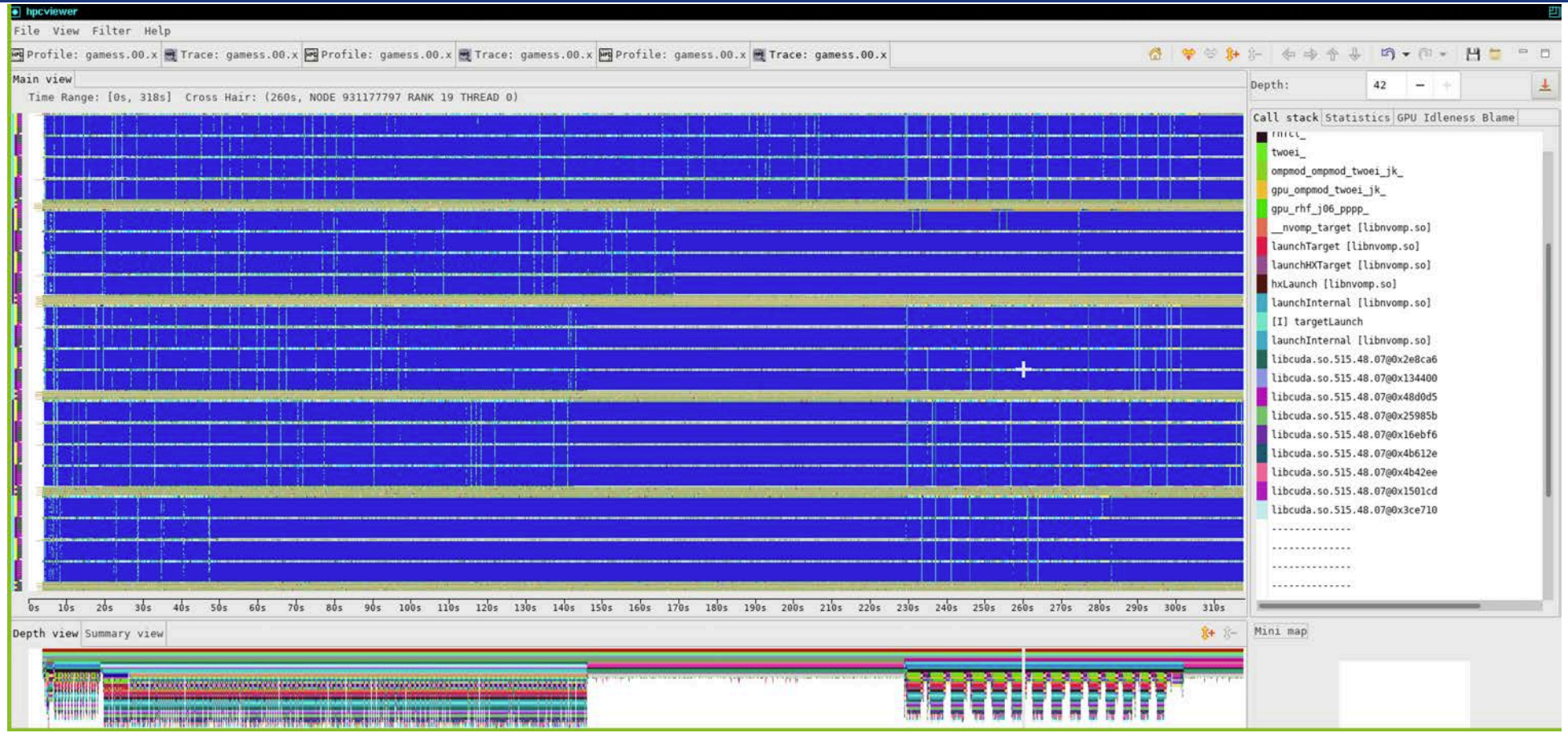
Time-centric Analysis: GAMESS 4 ranks, 4 GPUs on Perlmutter



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)
» main [gamess.00.x]	2.09e+02 5.9%		1.67e+02 100.0%		1.67e+02 100.0%		2.64e-01 100.0%	
» gamess	2.09e+02 5.9%		1.67e+02 100.0%		1.67e+02 100.0%		2.64e-01 100.0%	
772 » brnchx_	2.08e+02 5.8%		1.67e+02 100.0%		1.67e+02 100.0%		2.64e-01 100.0%	
1095 » energx_	2.08e+02 5.8%		1.67e+02 100.0%		1.67e+02 100.0%		2.64e-01 100.0%	
loop at gamess.F: 1316	2.08e+02 5.8%		1.67e+02 100.0%		1.67e+02 100.0%		2.64e-01 100.0%	
loop at gamess.F: 1436	2.01e+02 5.7%		1.67e+02 100.0%		1.67e+02 100.0%		2.64e-01 100.0%	
loop at gamess.F: 1436	2.01e+02 5.7%		1.67e+02 100.0%		1.67e+02 100.0%		2.64e-01 100.0%	
1440 » wfn_	2.01e+02 5.7%		1.67e+02 100.0%		1.67e+02 100.0%		2.64e-01 100.0%	
loop at gamess.F: 2645	2.01e+02 5.7%		1.67e+02 100.0%		1.67e+02 100.0%		2.64e-01 100.0%	
2568 » rhfcl_	2.01e+02 5.7%	2.06e-02 0.0%	1.67e+02 100.0%		1.67e+02 100.0%		2.64e-01 100.0%	
loop at rhfuhf.f: 2678	2.01e+02 5.6%	1.52e-02 0.0%	1.67e+02 100.0%		1.67e+02 100.0%		2.64e-01 100.0%	
loop at rhfuhf.f: 2723	2.01e+02 5.6%		1.67e+02 100.0%		1.67e+02 100.0%		2.64e-01 100.0%	
loop at rhfuhf.f: 2723	2.01e+02 5.6%	5.34e-03 0.0%	1.67e+02 100.0%		1.67e+02 100.0%		2.64e-01 100.0%	
2859 » twoei_	1.70e+02 4.8%		1.67e+02 100.0%		1.67e+02 100.0%		2.64e-01 100.0%	
3994 » ompmod_ompmod_twoei_jk_	1.70e+02 4.8%		1.67e+02 100.0%		1.67e+02 100.0%		2.64e-01 100.0%	
246 » gpu_ompmod_twoei_jk_	1.70e+02 4.8%		1.67e+02 100.0%		1.67e+02 100.0%		2.64e-01 100.0%	
656 » gpu_rhf_j16_dppp_	2.99e+01 0.8%		2.98e+01 17.8%		2.98e+01 17.9%		1.08e-02 4.1%	
589 » gpu_rhf_j06_pppp_	2.79e+01 0.8%	1.00e-02 0.0%	2.79e+01 16.7%		2.79e+01 16.7%		8.98e-03 3.4%	
584 » gpu_rhf_j05_ppps_	2.67e+01 0.7%	1.50e-02 0.0%	2.66e+01 15.9%		2.66e+01 16.0%		9.56e-03 3.6%	
631 » gpu_rhf_j11_dpps_	1.57e+01 0.4%	2.50e-02 0.0%	1.57e+01 9.4%		1.57e+01 9.4%		1.08e-02 4.1%	
579 » gpu_rhf_j04_psps_	1.22e+01 0.3%		1.22e+01 7.3%		1.22e+01 7.3%		9.87e-03 3.7%	
671 » gpu_rhf_j19_dpdp	7.76e+00 0.2%	5.00e-03 0.0%	7.77e+00 4.6%		7.76e+00 4.6%		9.99e-03 3.8%	

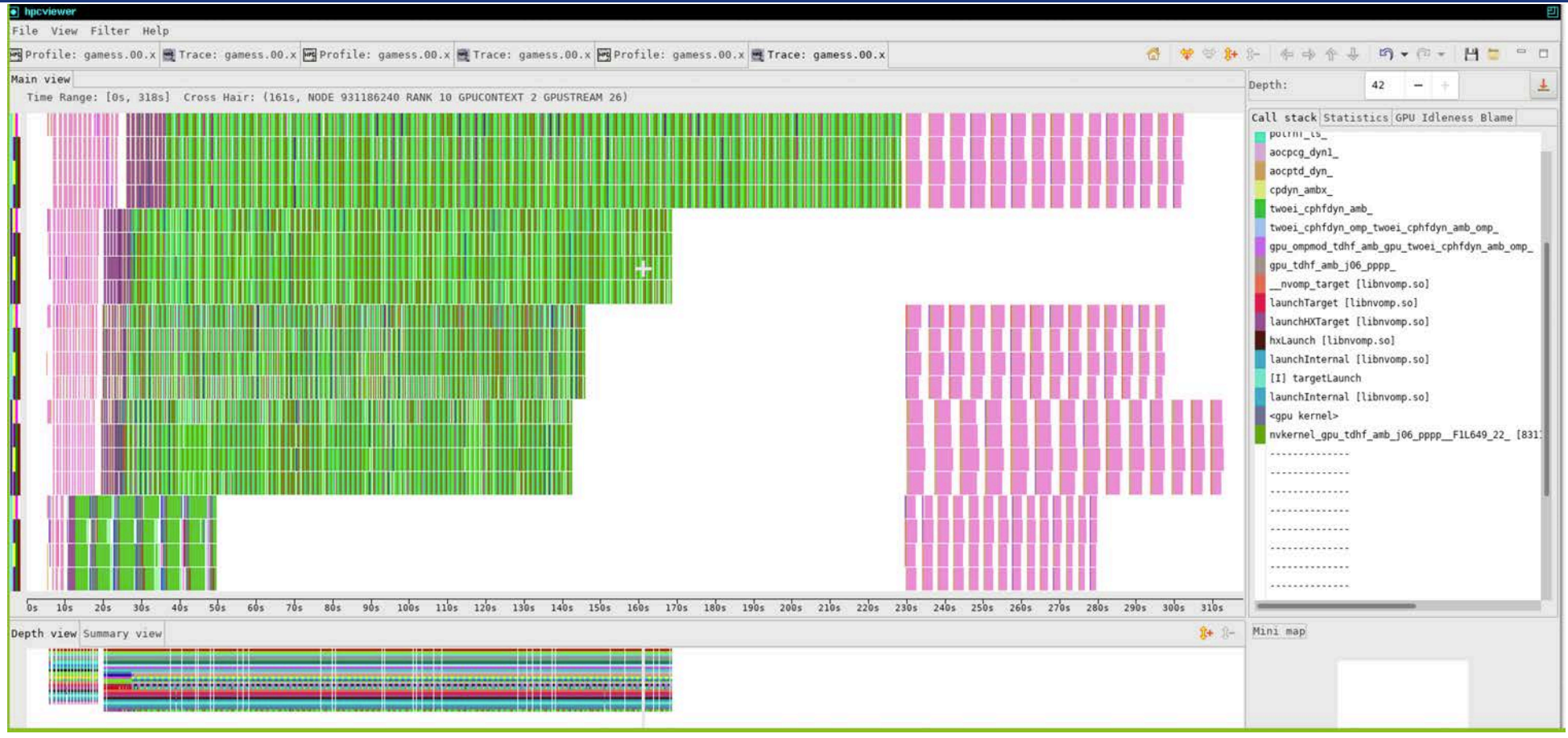
GAMESS improved

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



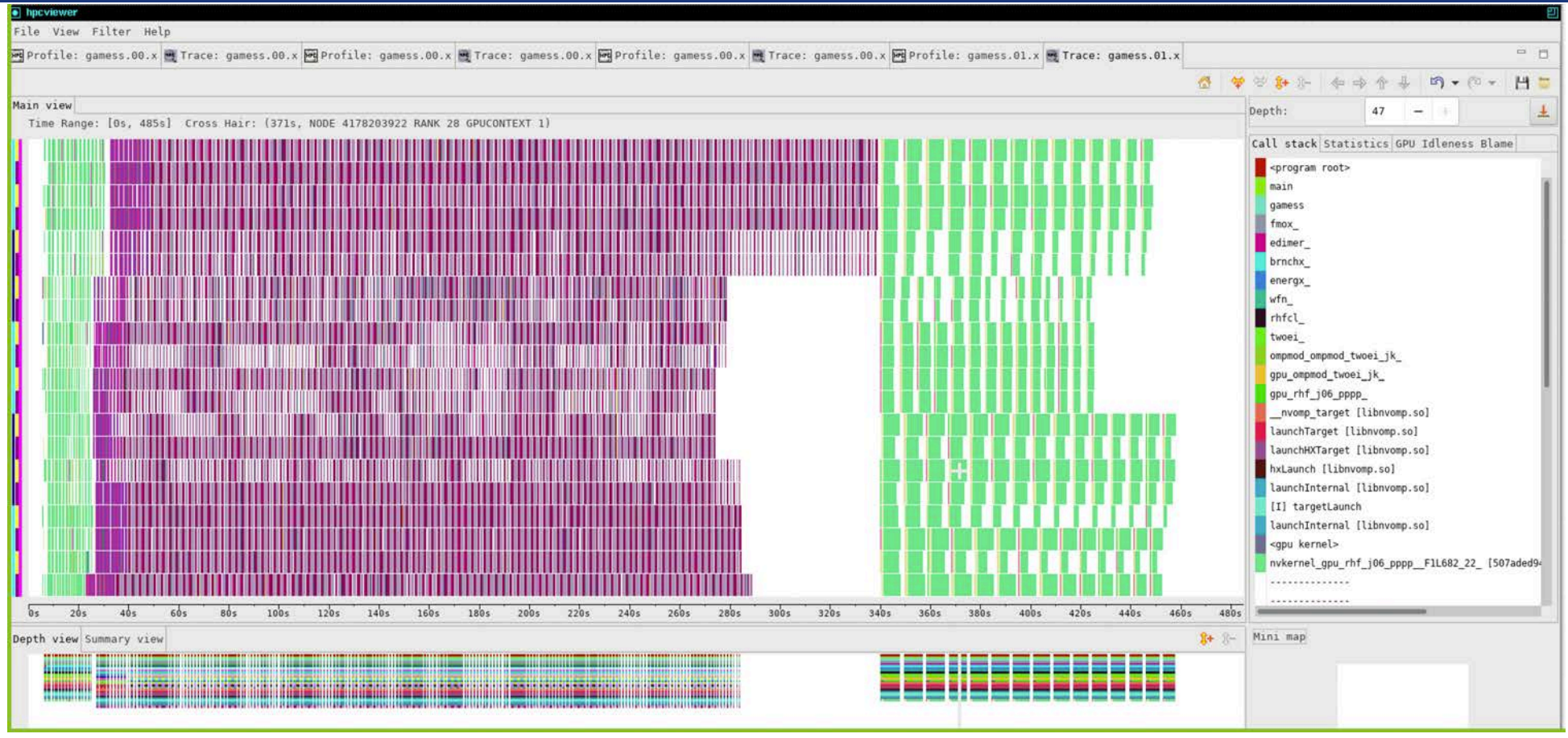
GAMESS improved CPU Threads and GPU Streams

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



GAMESS improved

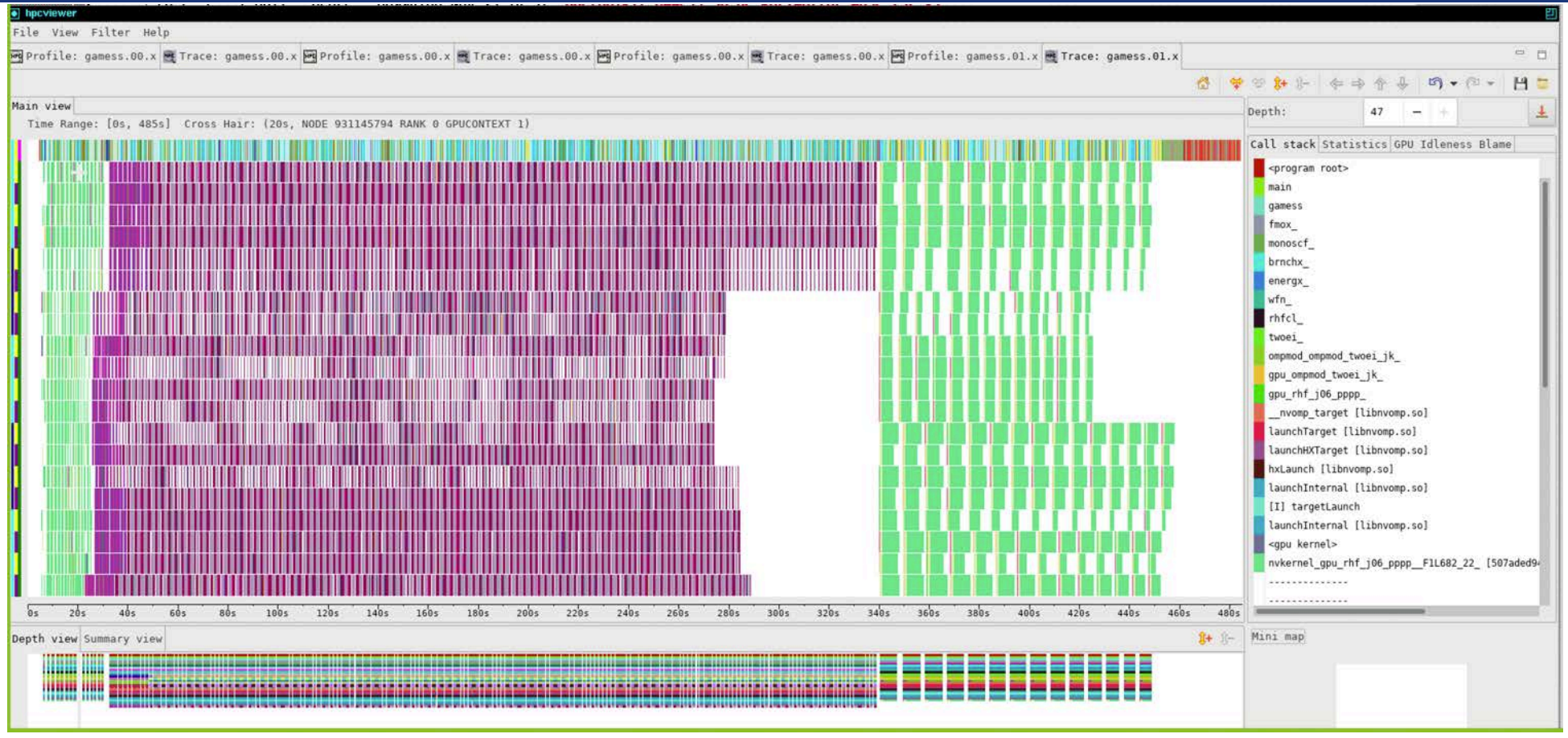
Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter



ARGONNE ATPESC2024 EXTREME-SCALE COMPUTING

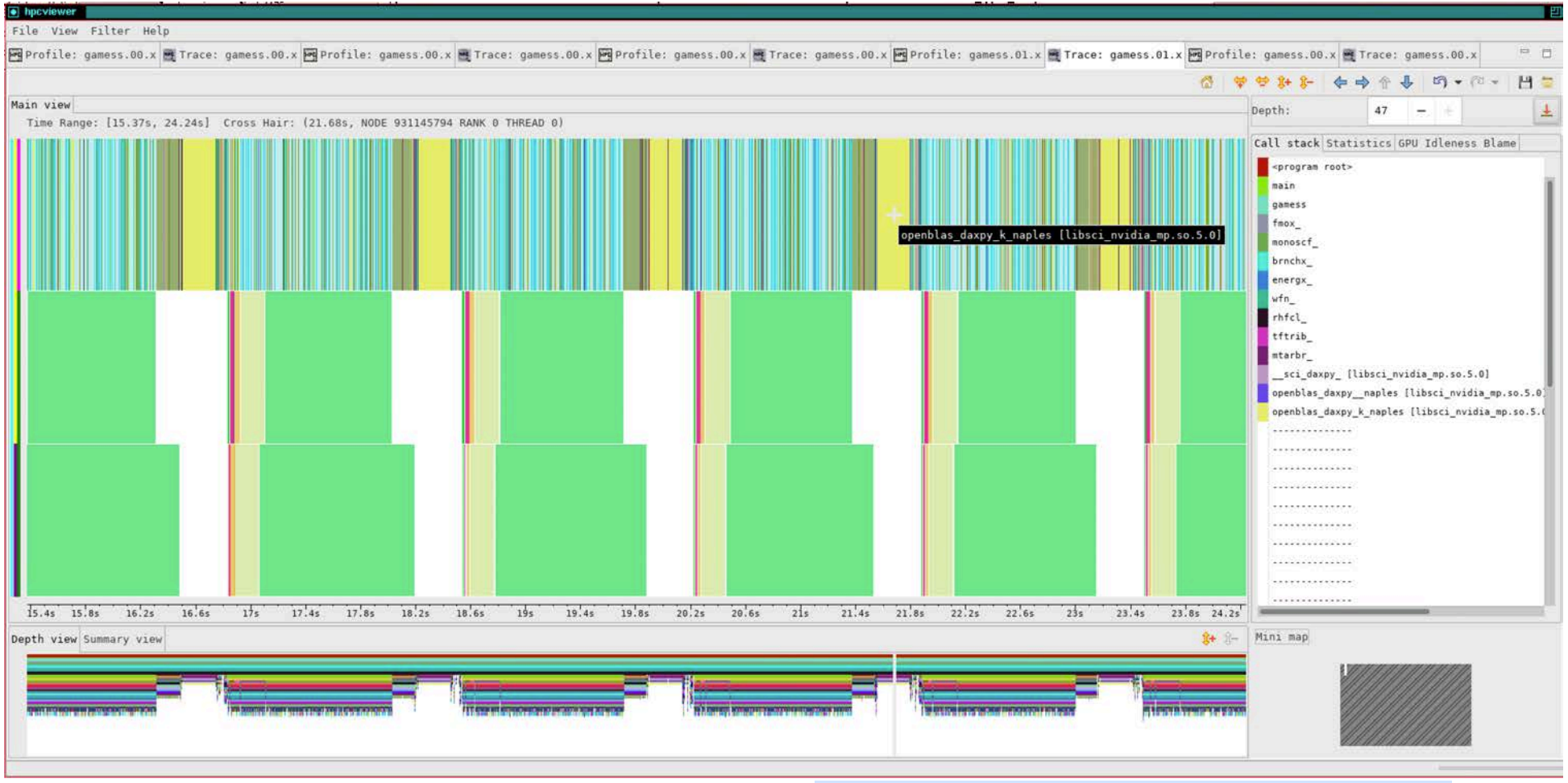
GAMESS improved + better manual distribution of work in input

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



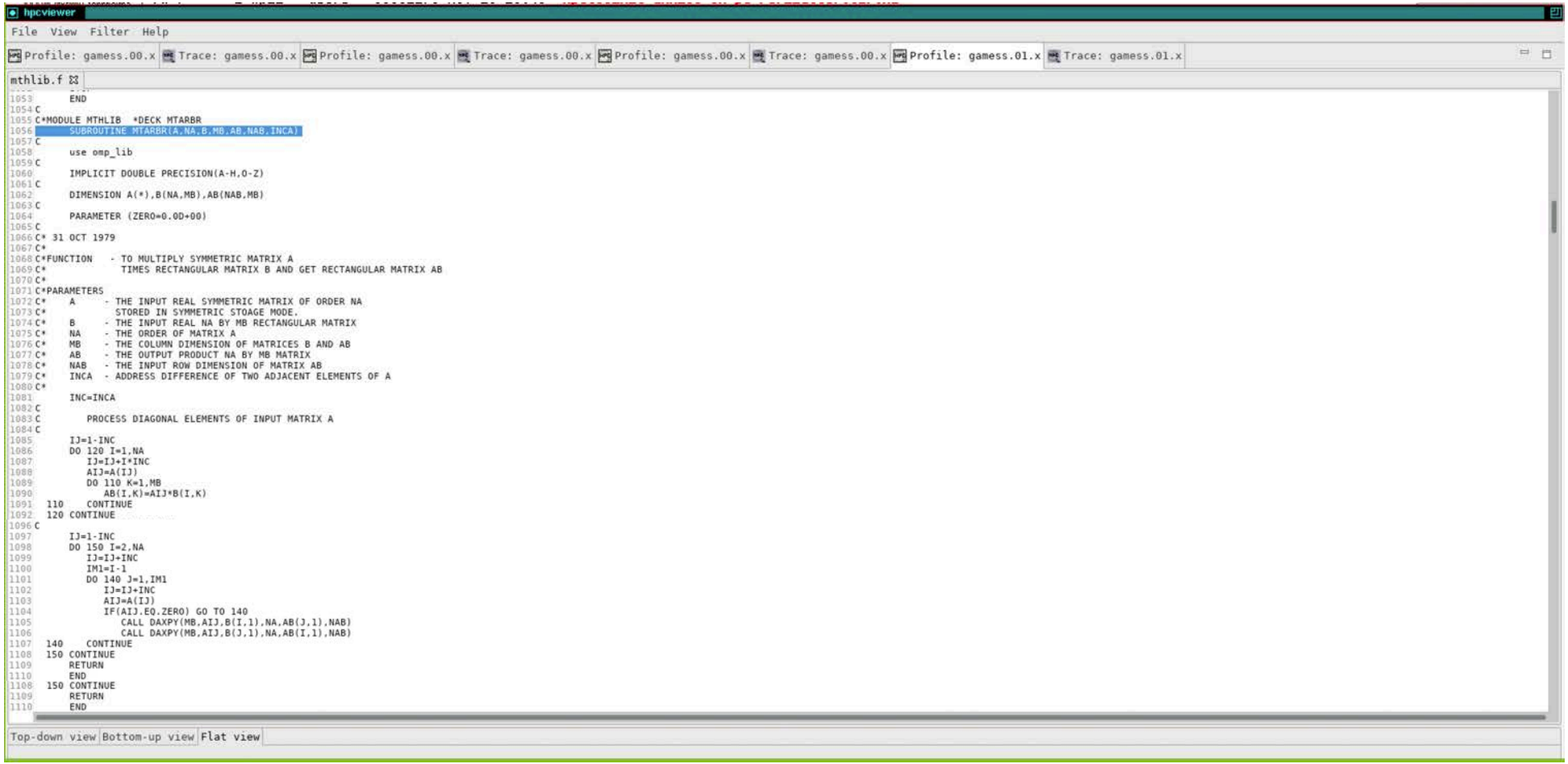
Showing Rank 0 Thread 0 in addition to GPU streams

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

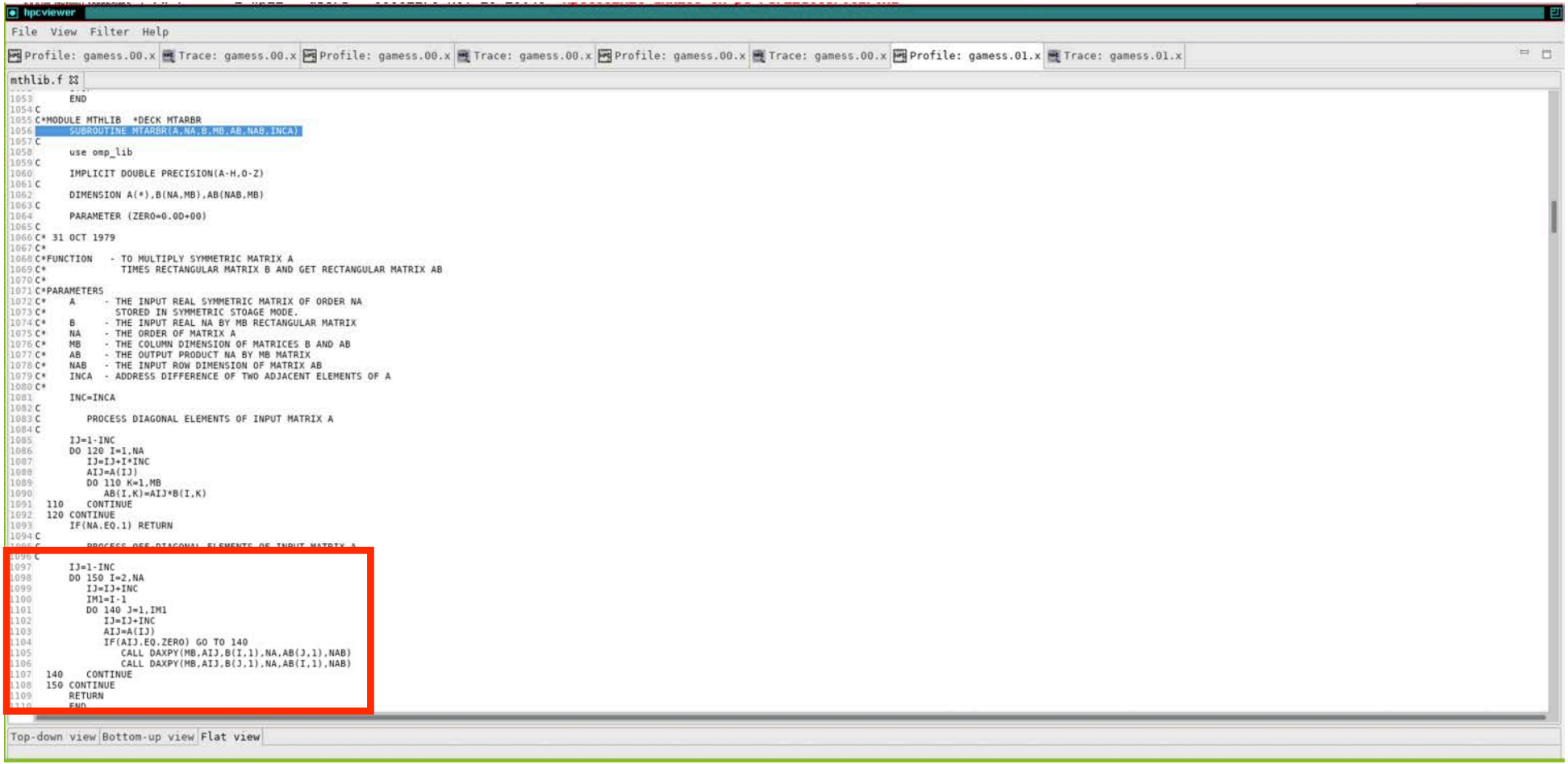


1 CPU Stream, 2 GPU Streams: 6 Iterations

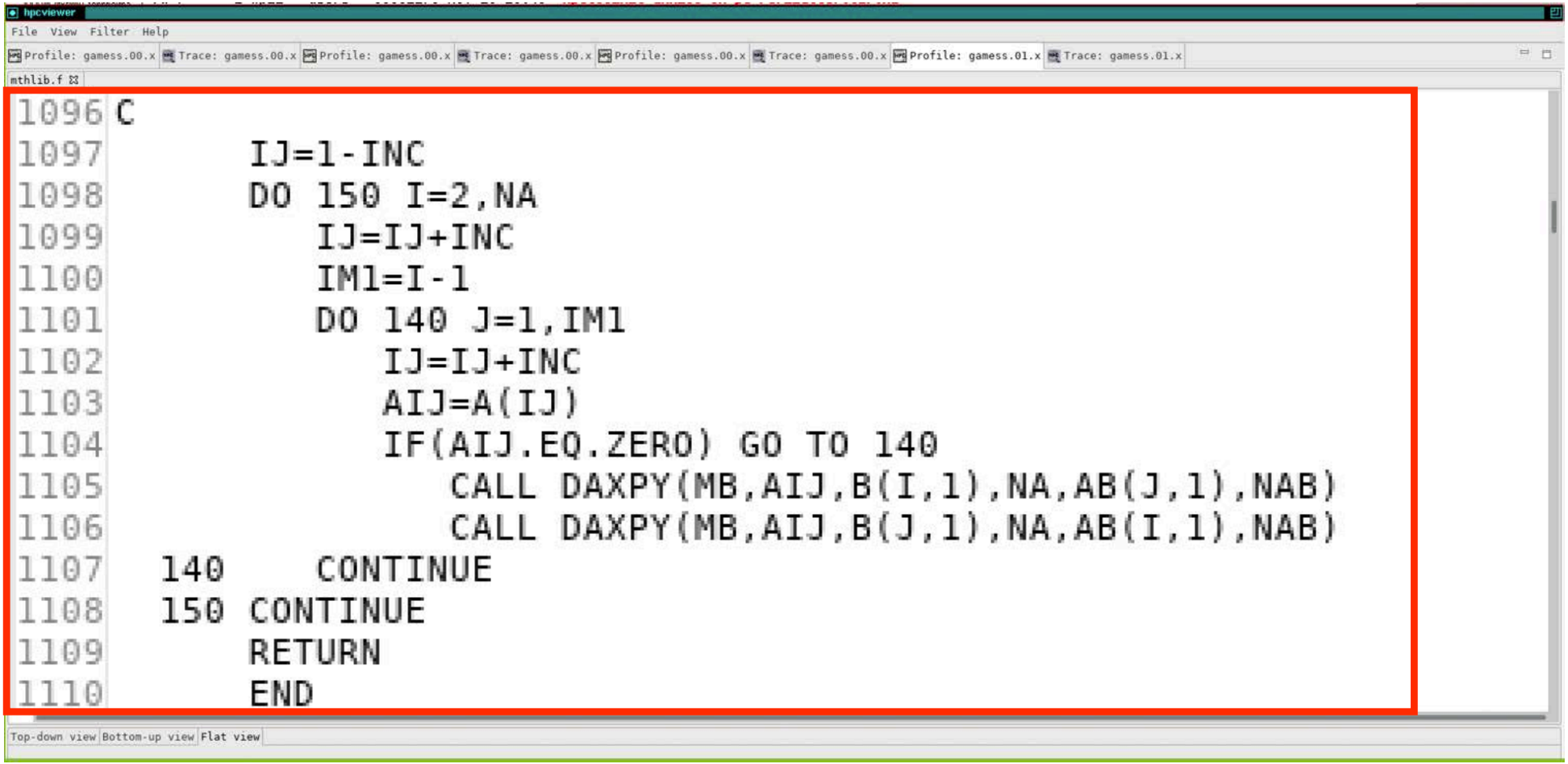
Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter



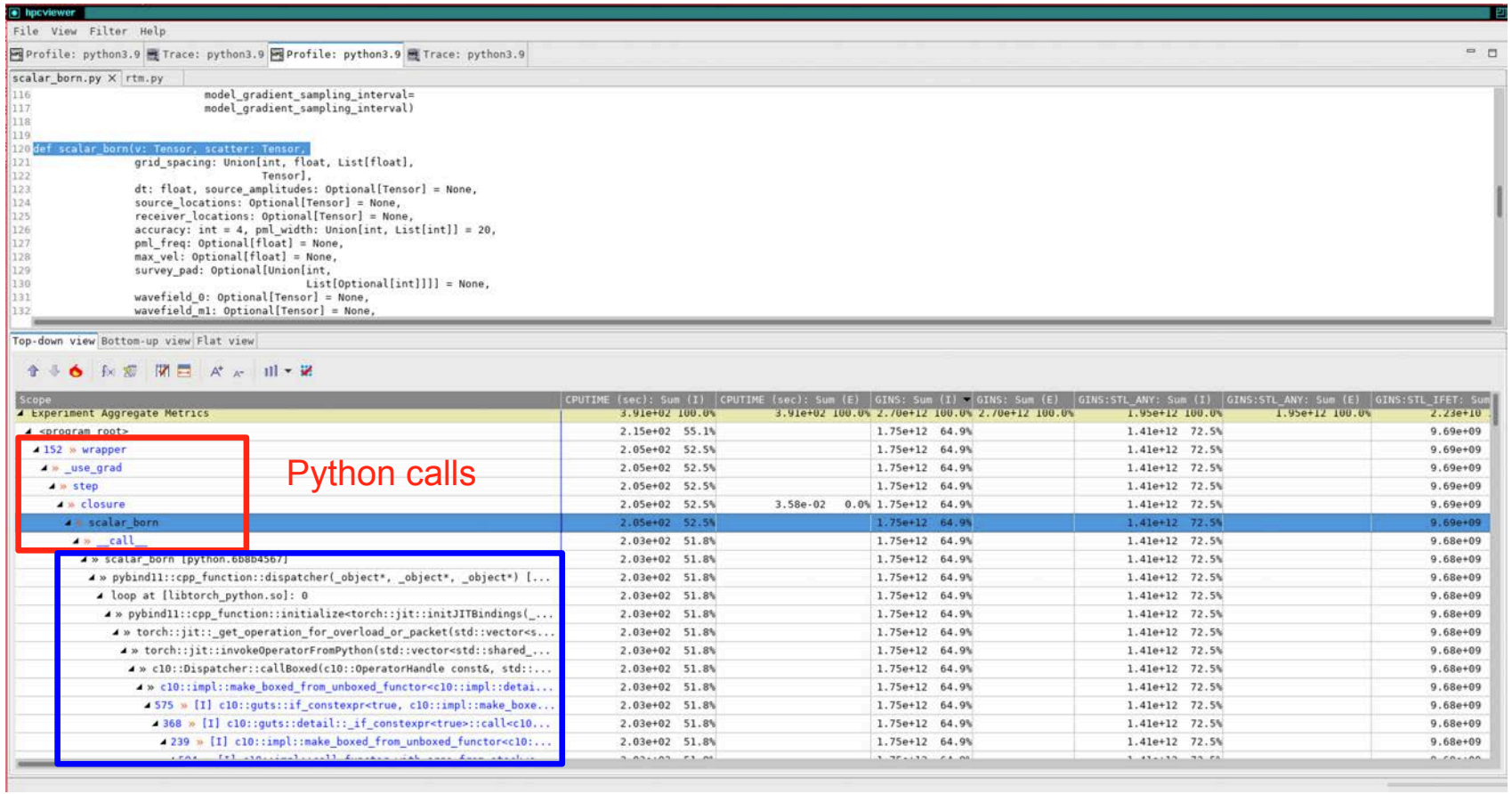
Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter



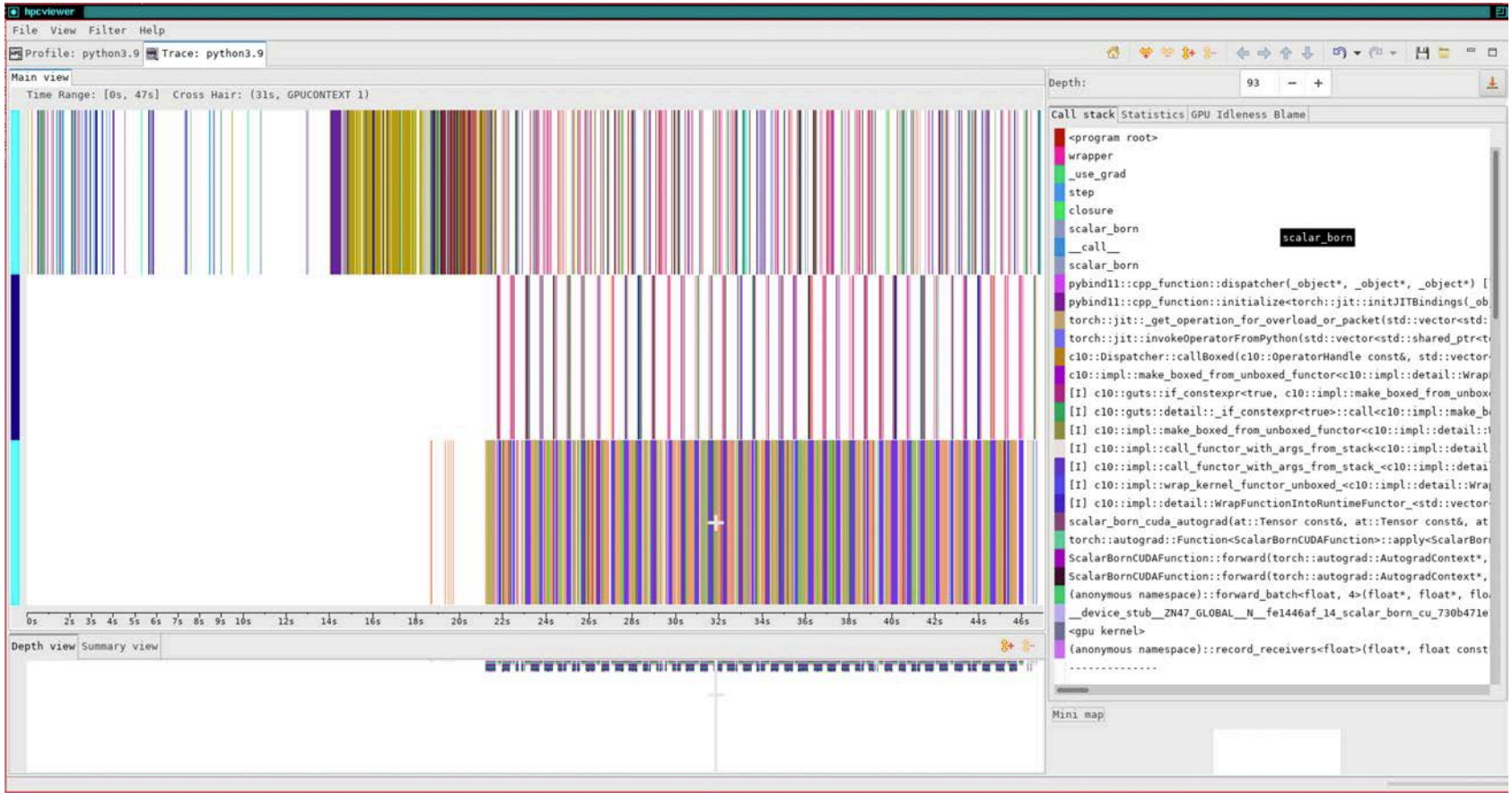
Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter



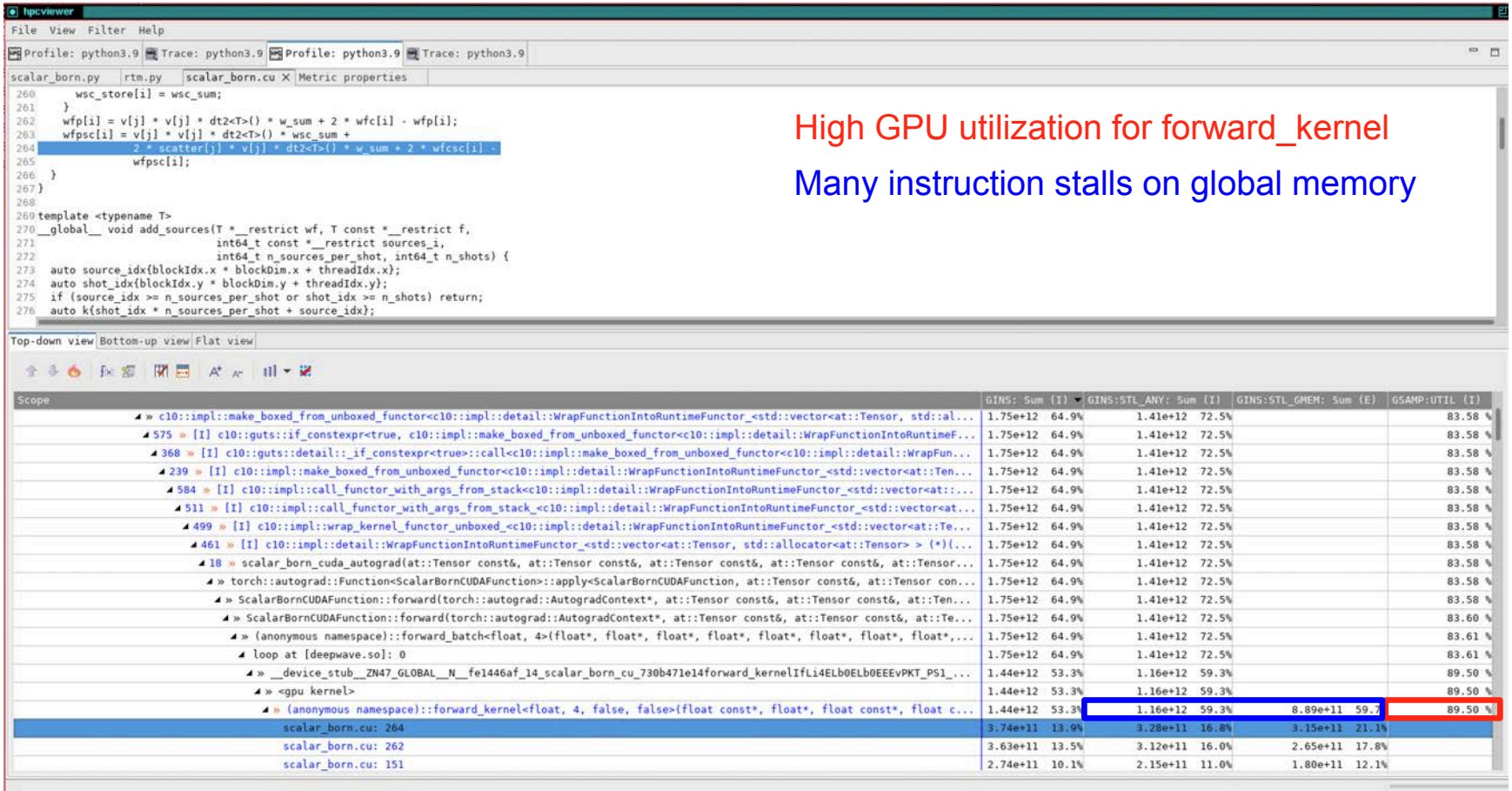
Measure Pytorch Deepwave (RTM) on CPU + NVIDIA GPU



Trace Pytorch Deepwave (RTM): CPU threads 0, 48; GPU stream



Analyze Pytorch Deepwave (RTM) Kernel using PC Sampling



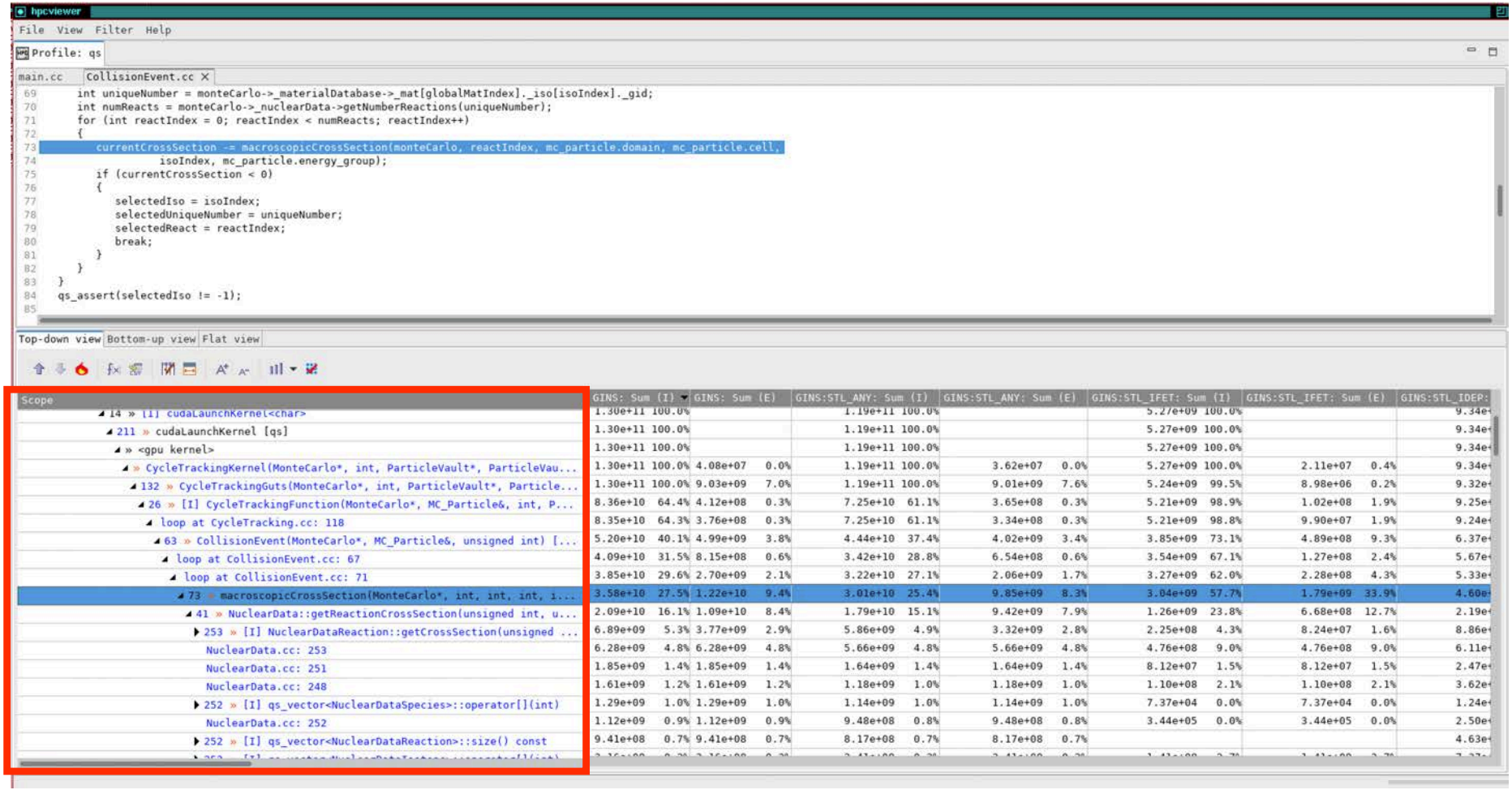
High GPU utilization for forward_kernel

Many instruction stalls on global memory

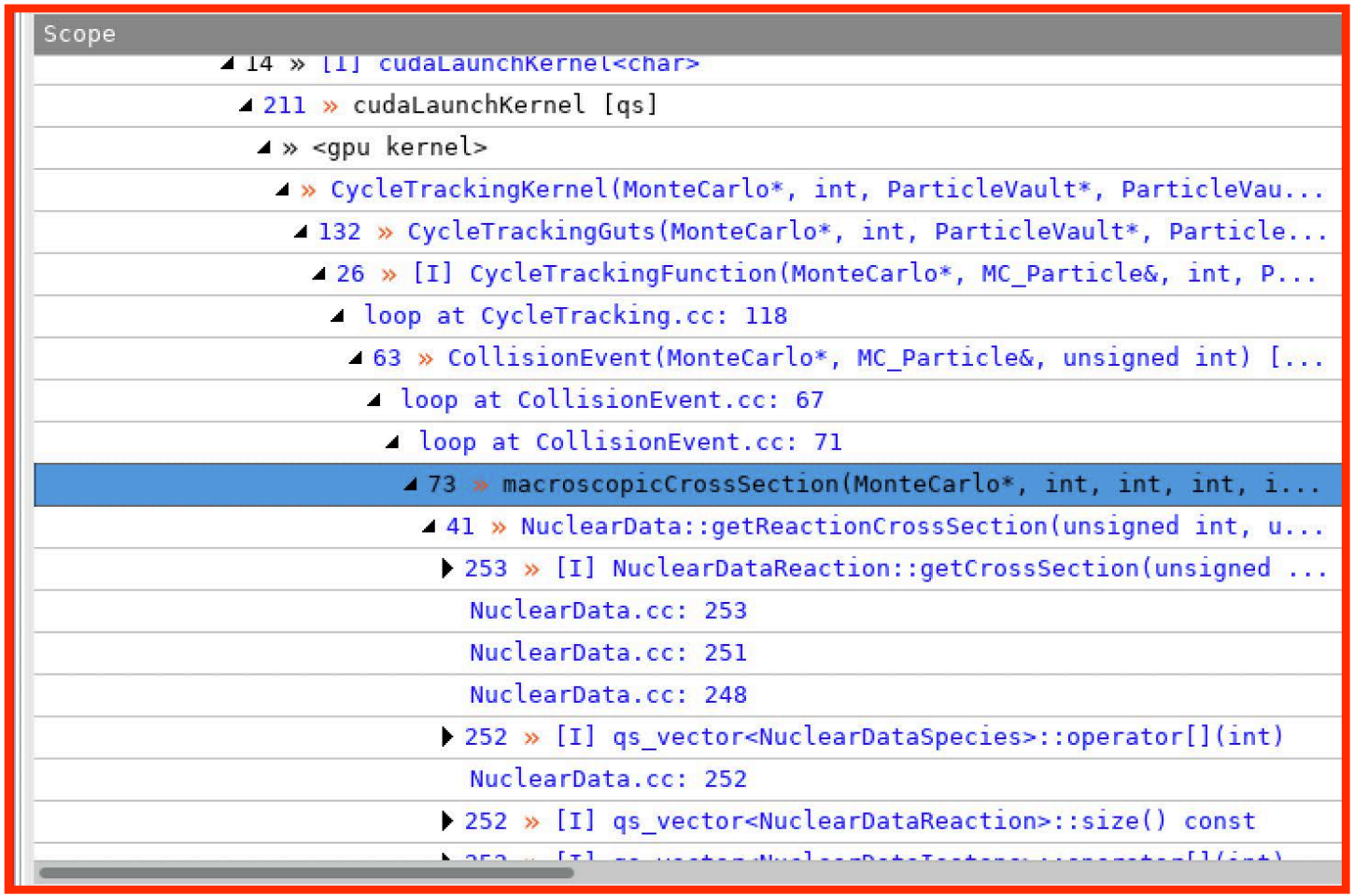
Case Study: Quicksilver

- Proxy application that solves a simplified dynamic Monte Carlo particle transport problem
  - Attempts to replicate memory access patterns, communication patterns, and branching or divergence of LLNL's 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

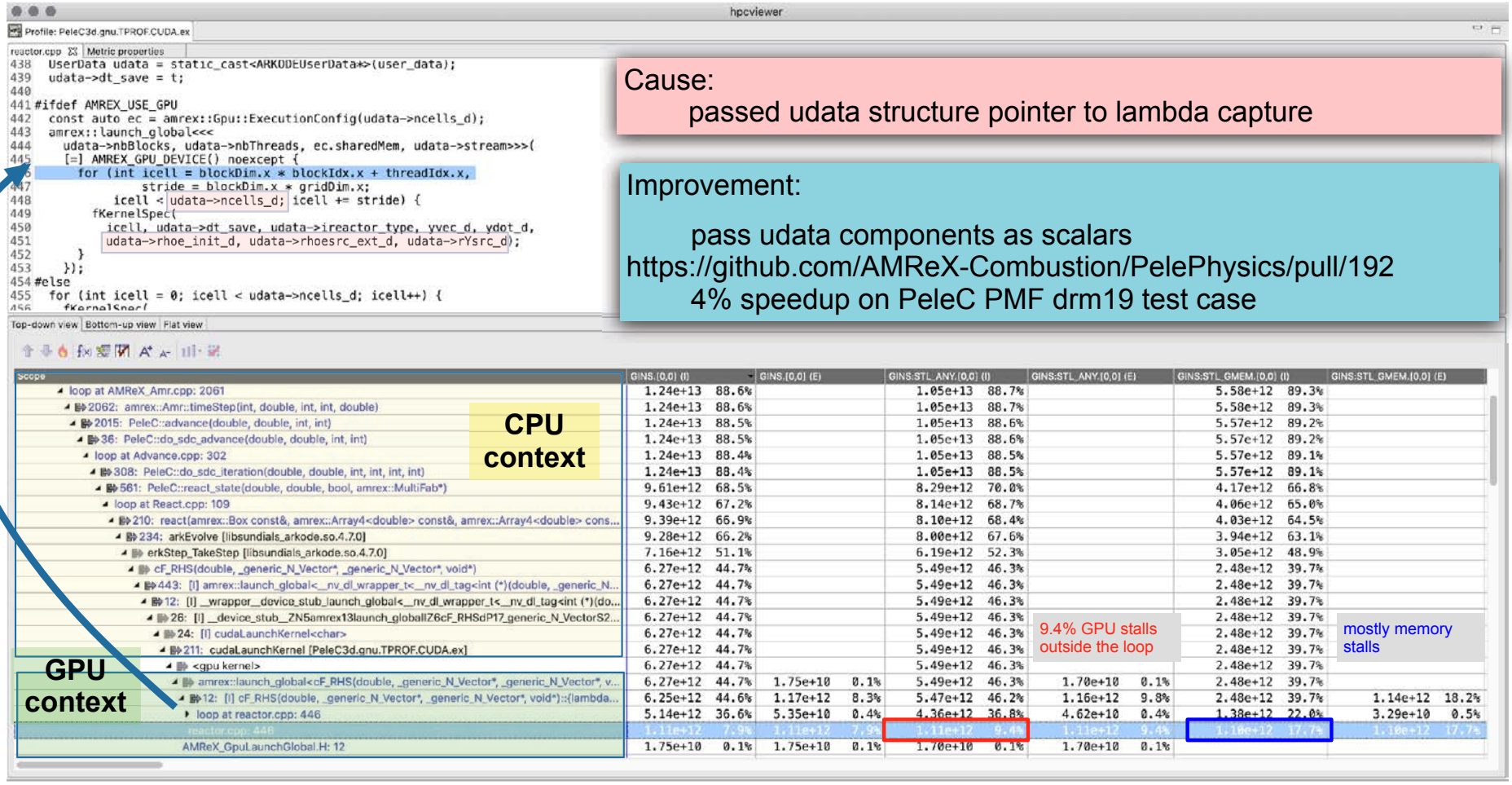
Quicksilver: Detailed Analysis within a Kernel using PC Sampling



Quicksilver: Detailed analysis within a Kernel using PC Sampling



Analysis of PeleC using PC Sampling on an NVIDIA GPU



Key Metrics for GPU Kernels

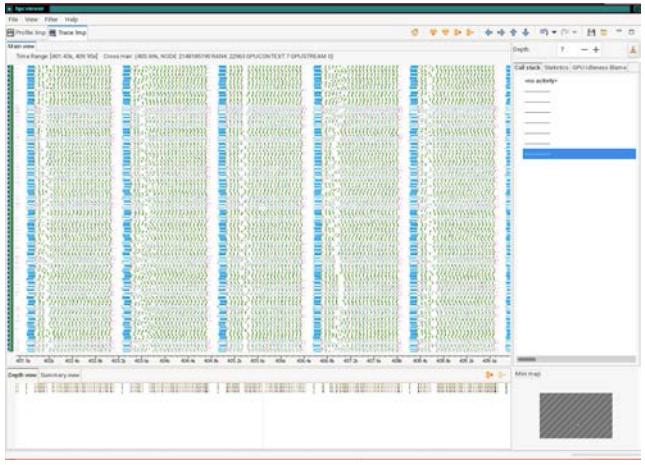
- GPUOP: GPU operation time (kernel launch, copies, etc.)
- GXCOPY:* GPU copies of various kinds
- GKER: GPU kernel time
- GKER:FGP_ACT: fine grain parallelism actual (active warps per SM)
- GKER:FGP_MAX: maximum possible fine-grain parallelism (max warps per SM)
- GKER:BLK_THR: threads per block
- GKER:BLK_SM: block shared memory
- GKER:OCC_THR: theoretical thread occupancy

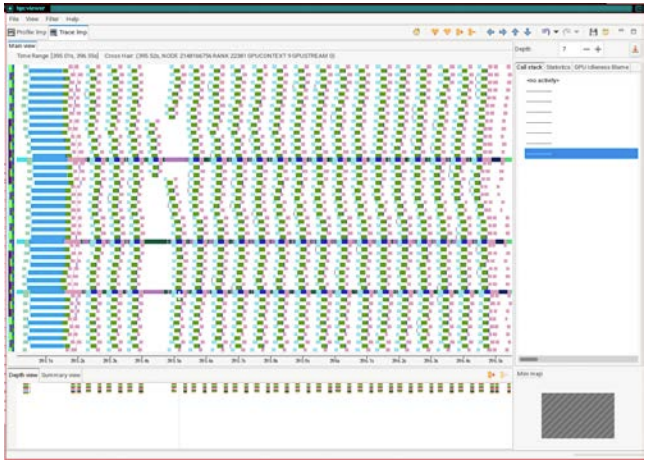
Metrics for GPU Kernels with PC Samples

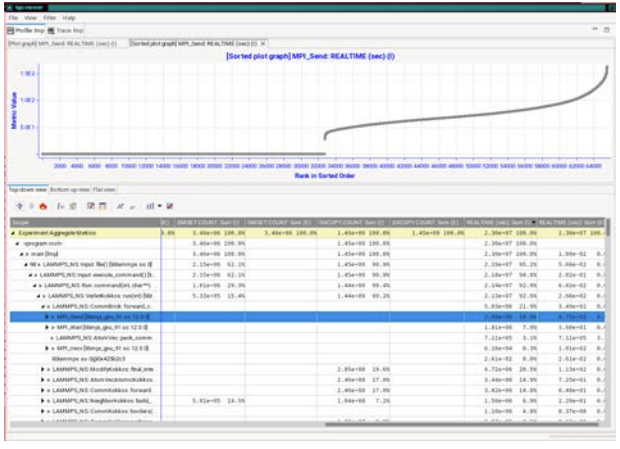
- GINS: GPU instructions
- GINS:STL_ANY: GPU instruction stalls for any reason
- GINS:STL_IFET: GPU instruction stalls for instruction fetch
- GINS:STL_GMEM: GPU instruction stalls for global memory
- GINS:STL_CMEM: GPU instruction stalls for constant memory
- GINS:STL_IDEP: GPU instruction stalls for instruction dependences
- GINS:STL_PIPE: GPU instruction pipeline stalls
- GINS:STL_MTHR: GPU instruction stalls for memory throttling

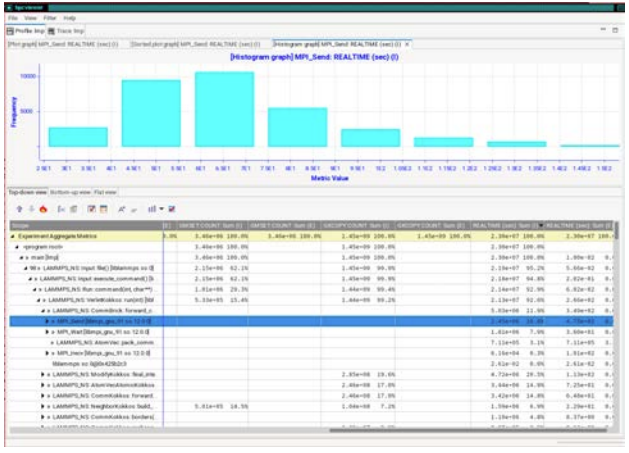
- GSAMP:EXP: expected number of samples
- GSAMP:TOT: total number of samples recorded
- GSAMP:UTIL: GPU utilization = (PC samples expected) / (PC samples total)

LAMMPS on Frontier: Executions with Kernel Duration of Milliseconds

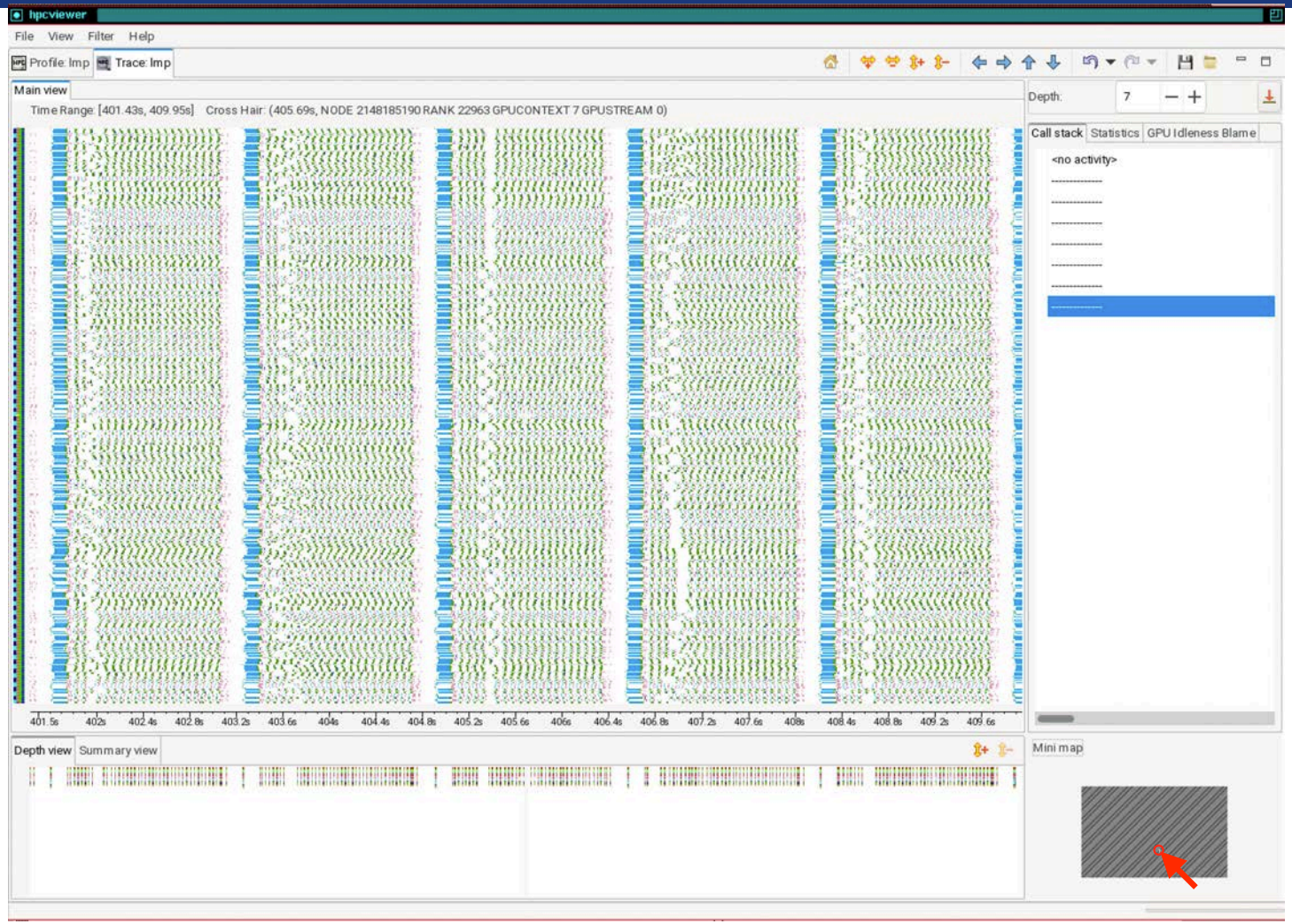




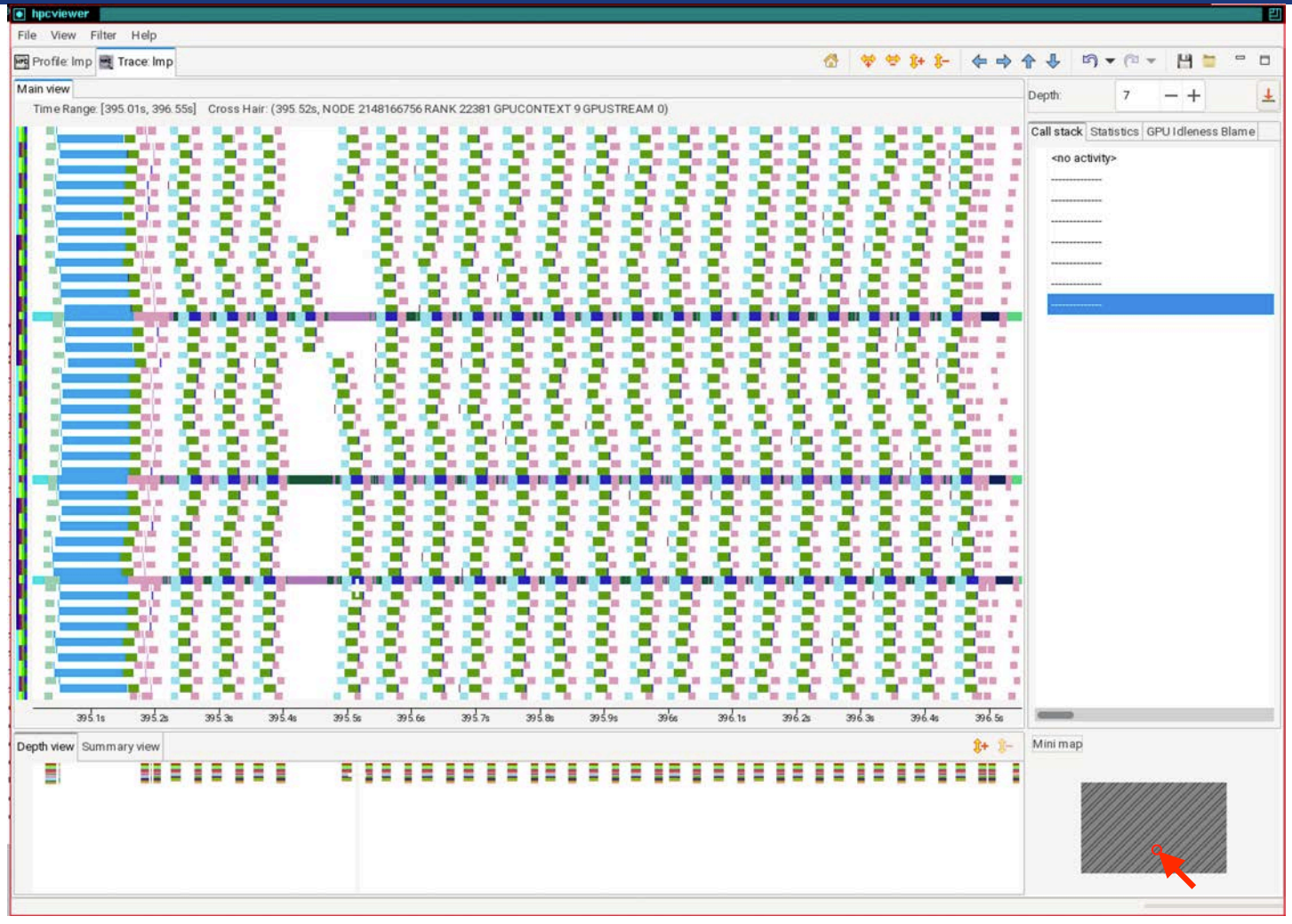




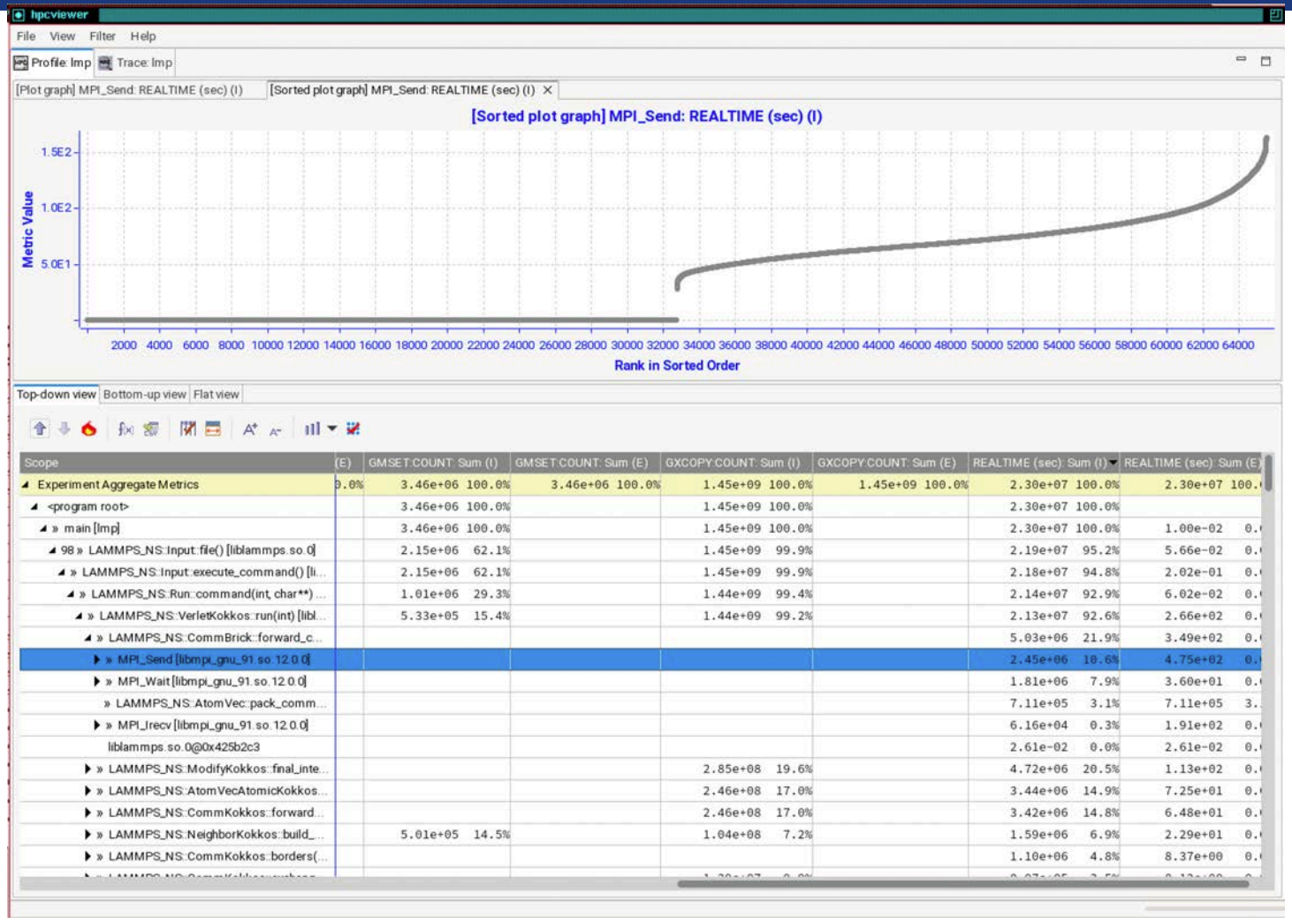
LAMMPS on Frontier: Executions with Kernel Duration of Milliseconds



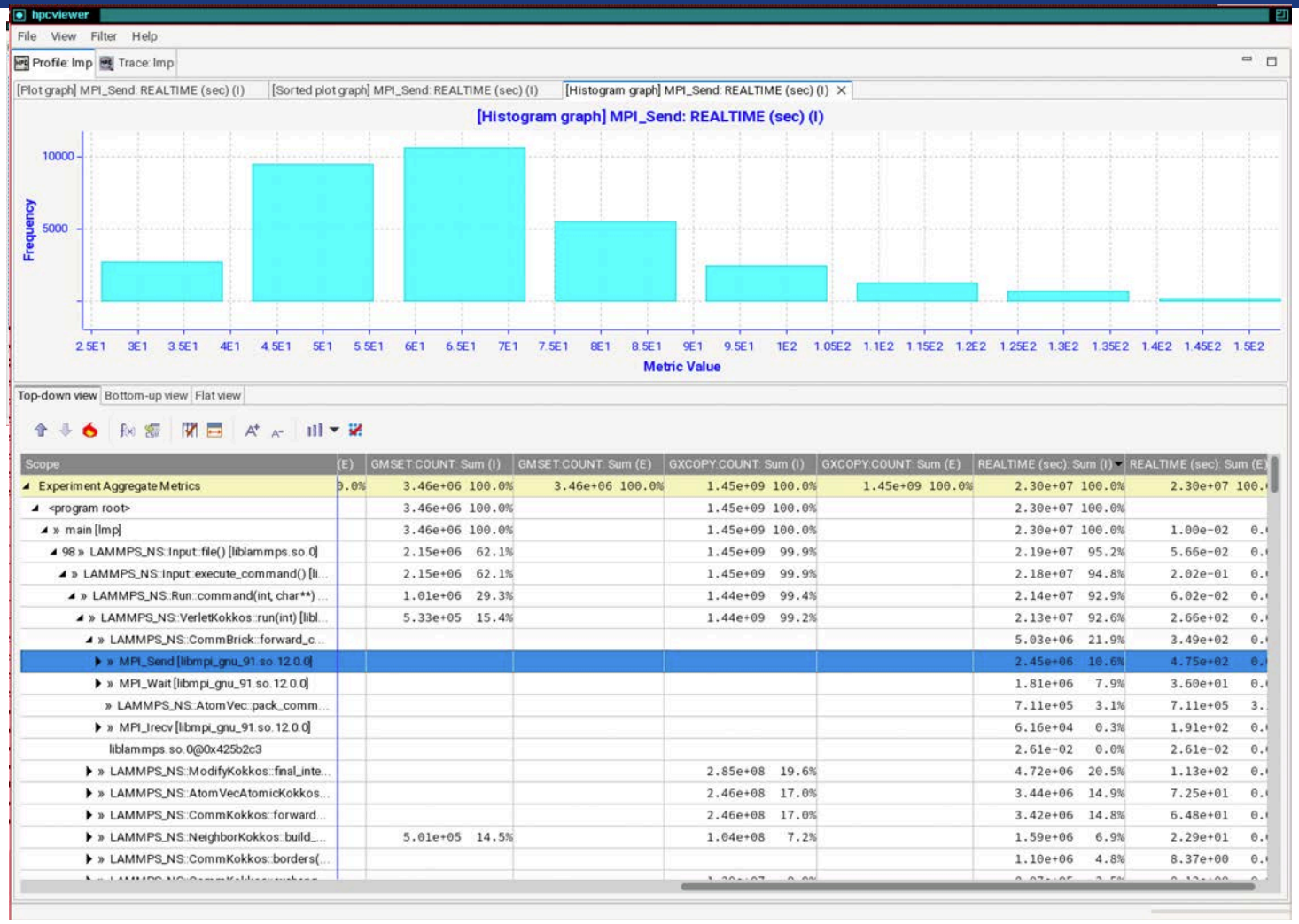
LAMMPS on Frontier: Executions with Kernel Duration of Milliseconds



LAMMPS on Frontier: Executions with Kernel Duration of Milliseconds

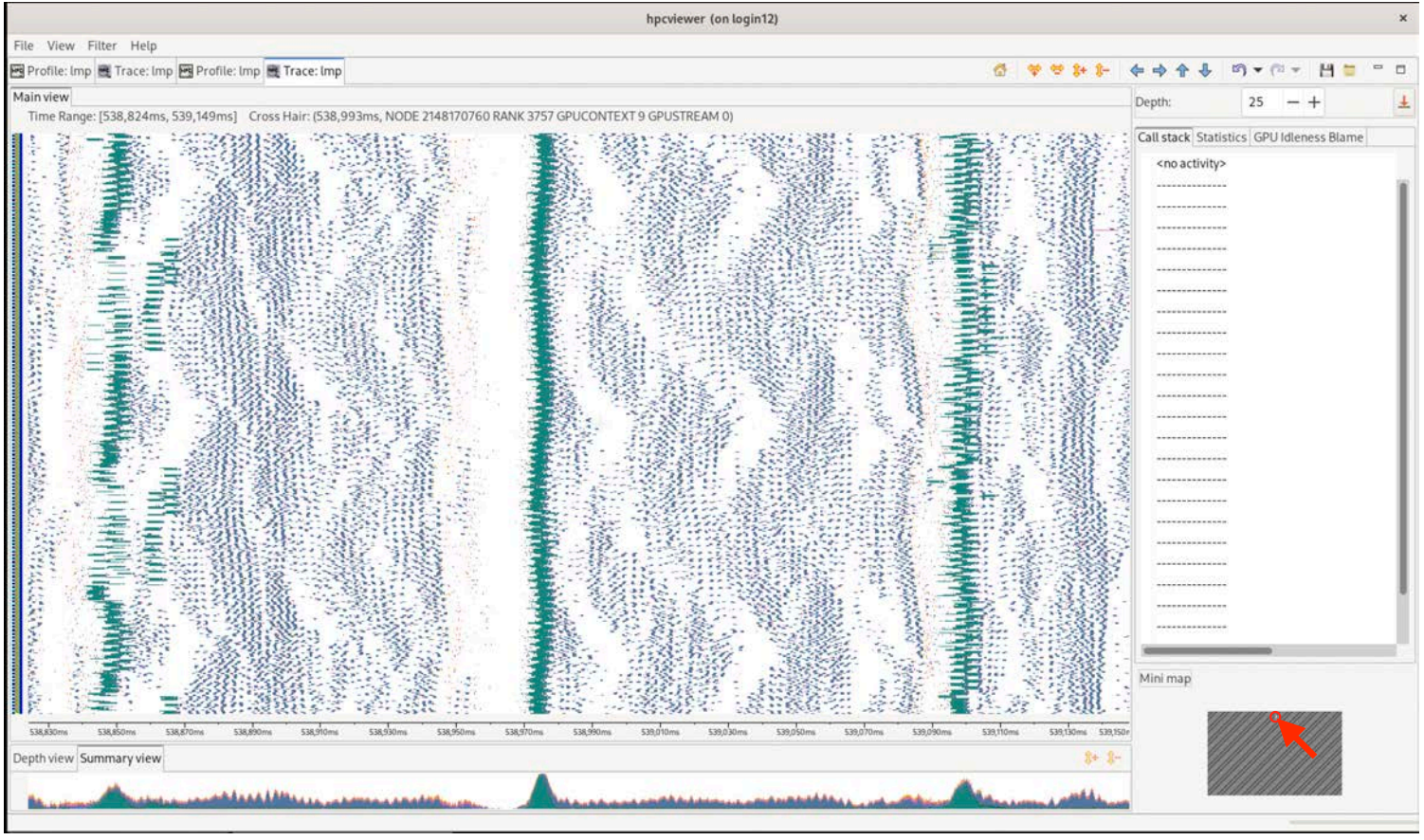


LAMMPS on Frontier: Executions with Kernel Duration of Milliseconds



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

Kernel duration of microseconds



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

Kernel duration of microseconds

Coming Attractions

- Emerging GUI support for accessing hpctoolkit databases on remote systems (try it today)
- More capabilities on AMD GPUs based on AMD's new rocprofiler-sdk (released yesterday)
  - PC sampling with precise attribution without serialization of concurrent kernels
  - improved support for hardware counters
- Integrated support for NVTX/ROCTX/Caliper/Kokkos labels
- Python-based interface for analysis of performance results
- Support for PC sampling on Intel GPUs

Quick Start Guide for HPCToolkit on Polaris

- Load HPCToolkit on Polaris

```
module use /soft/modulefiles
module load hpctoolkit
```

- Measure an execution
  - —Profile and trace CPU and GPU activity of your application in a job script

```
mpiexec -n <ranks> hpcrun -e CPUTIME —e gpu=nvidia -tt <myapp> <myapp arguments>
```

  - —Use PC sampling to collect instruction-level measurements within GPU kernels

```
mpiexec -n <ranks> hpcrun -e gpu=nvidia,pc <myapp> <myapp arguments>
```

- Analyze measurement data
  - —Analyze the CPU and GPU binaries recorded as your application was measured without PC sampling

```
hpcstruct hpctoolkit-<myapp>-measurements.<jobid>
```

  - —Analyze the CPU and GPU binaries recorded as your application was measured with PC sampling

```
hpcstruct --gpucfg yes hpctoolkit-<myapp>-measurements.<jobid>
```

- Combine measurement data with binary analysis results to produce a performance database

```
hpcprof  hpctoolkit-<myapp>-measurements.<jobid>
```

- View your performance data

```
hpcviewer hpctoolkit-<myapp>-database.<jobid>
```

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: https://bit.ly/hpcviewer-install

  Select the right one for your laptop: MacOS (Apple Silicon, Intel), Windows, Linux
- User manual for hpcviewer: https://hpctoolkit.gitlab.io/hpcviewer

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 pre-installed on Polaris
- You can install hpcserver anywhere: https://bit.ly/hpcserver-install

Configuring Hpcviewer Remote Access

Run hpcviewer

From the file menu, select “Open remote database”

Fill in the hostname/IP address: polaris.alcf.anl.gov

Fill in your username on Polaris

Fill in the remote installation directory for hpcviewer’s server: `/eagle/ATPESC2024/hpctoolkit/hpcserver`

Select the authentication method: “Use password”

Click “OK”

Authenticate using your token as you normally do

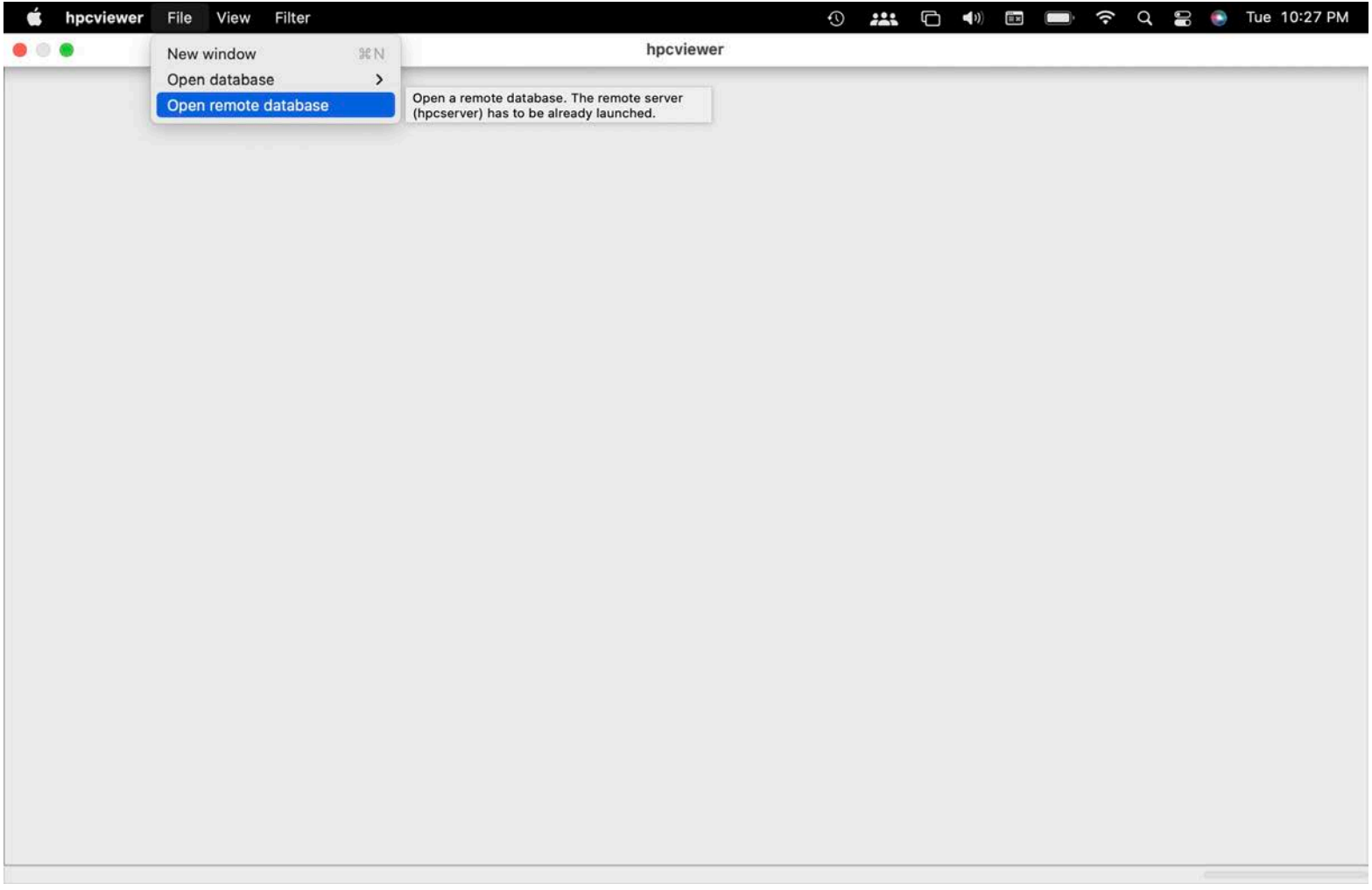
Navigate to a database with the file chooser in /eagle/APTESC/hpctoolkit/databases: quicksilver, lammps, arborx

arborx: hpctoolkit-arborx-md.d hpctoolkit-arborx-md-pc.d

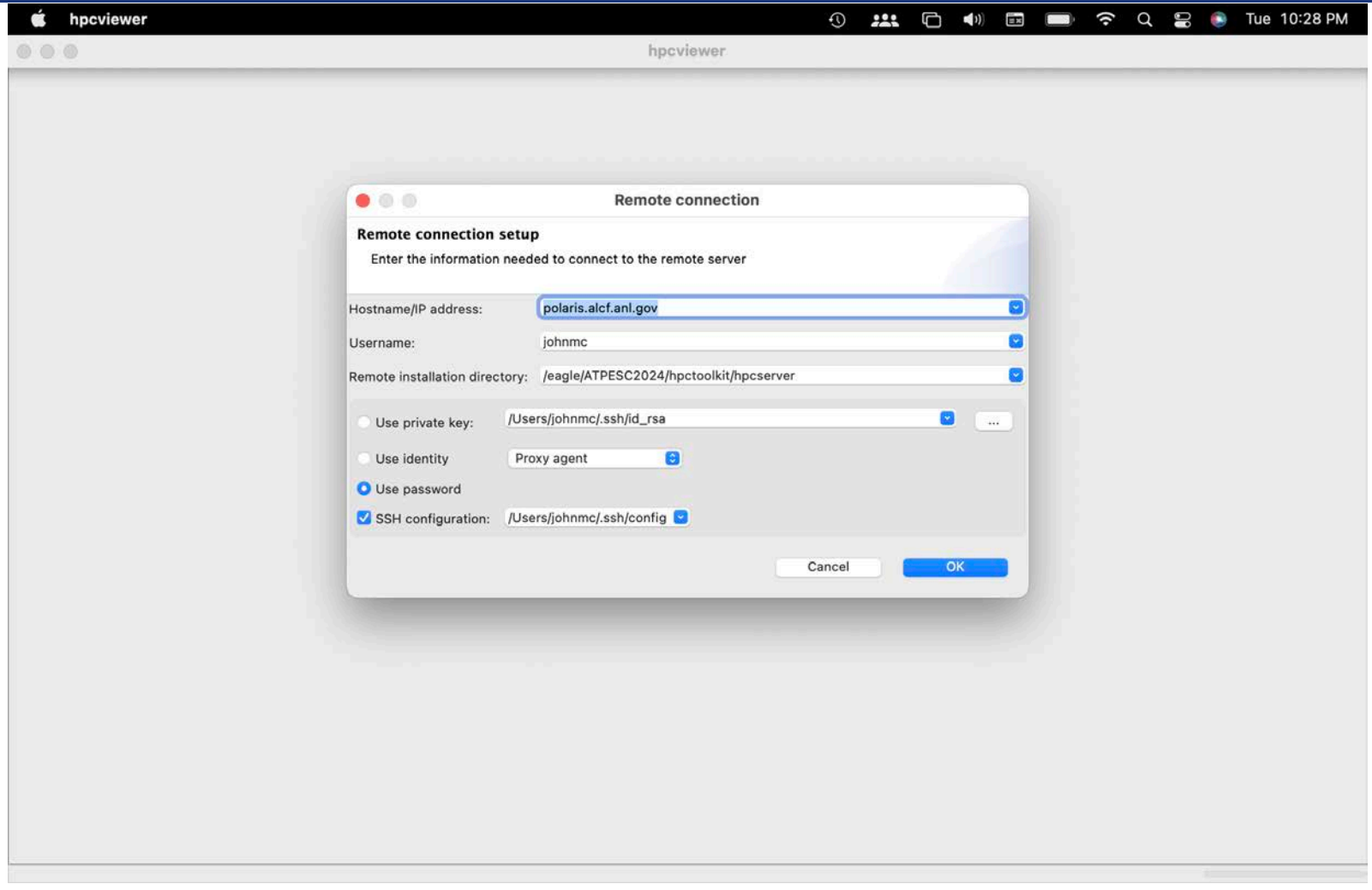
lammps: hpctoolkit-lmp.d hpctoolkit-lmp-pc.d

quicksilver: hpctoolkit-qs-gpu-cuda.d hpctoolkit-qs-gpu-cuda-pc.d

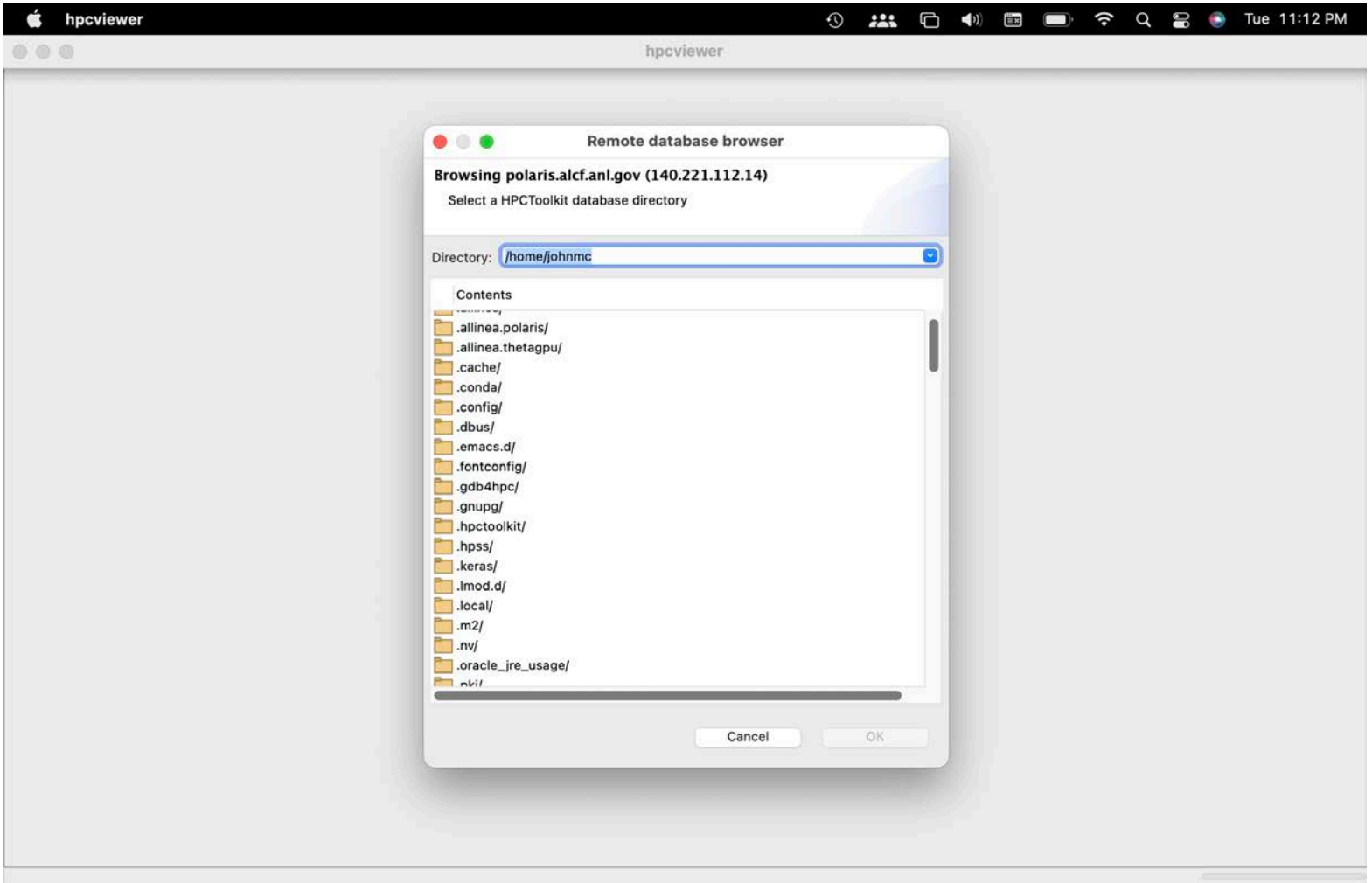
Opening a Remote Database



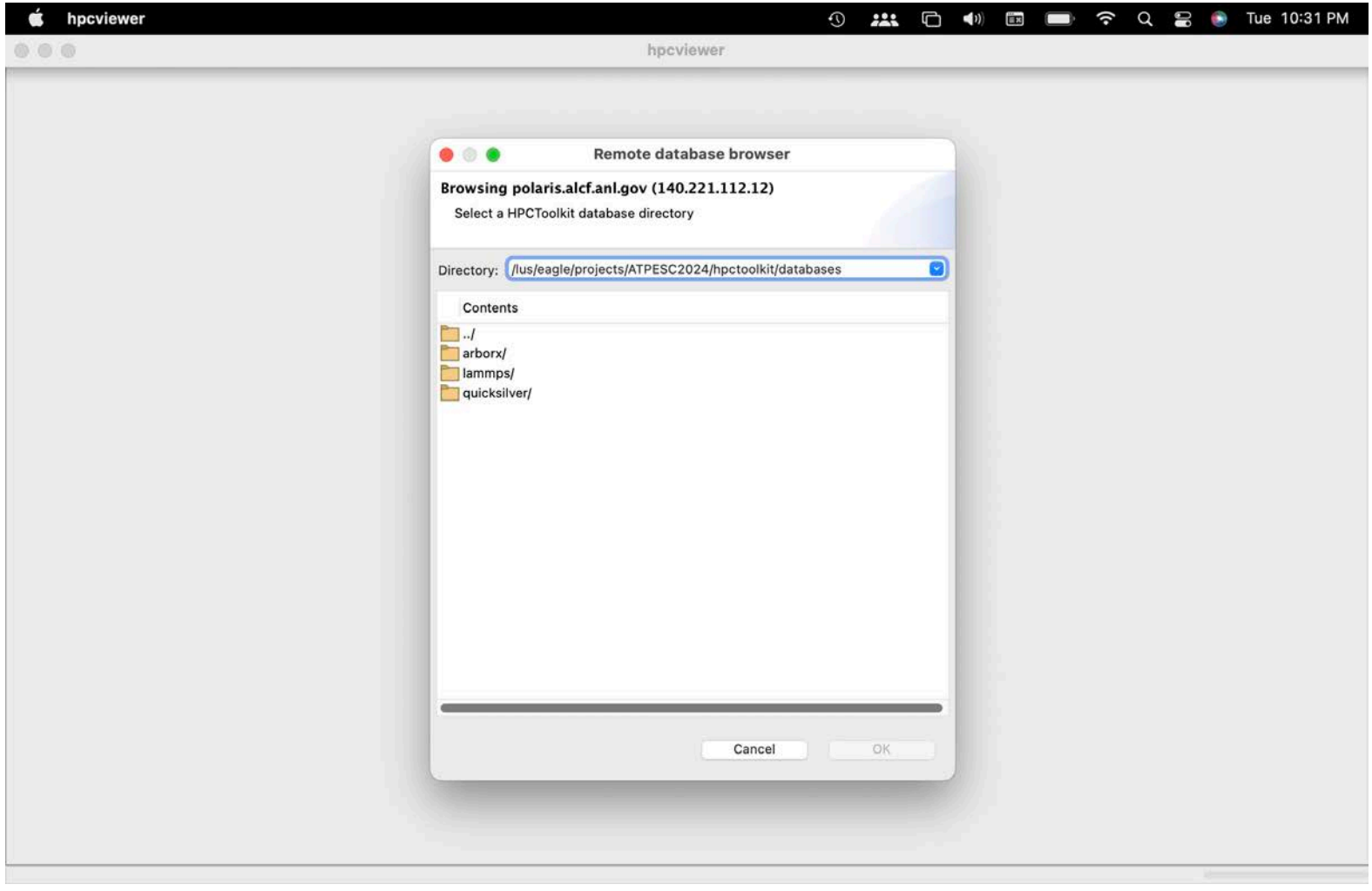
Configuring for use with Polaris



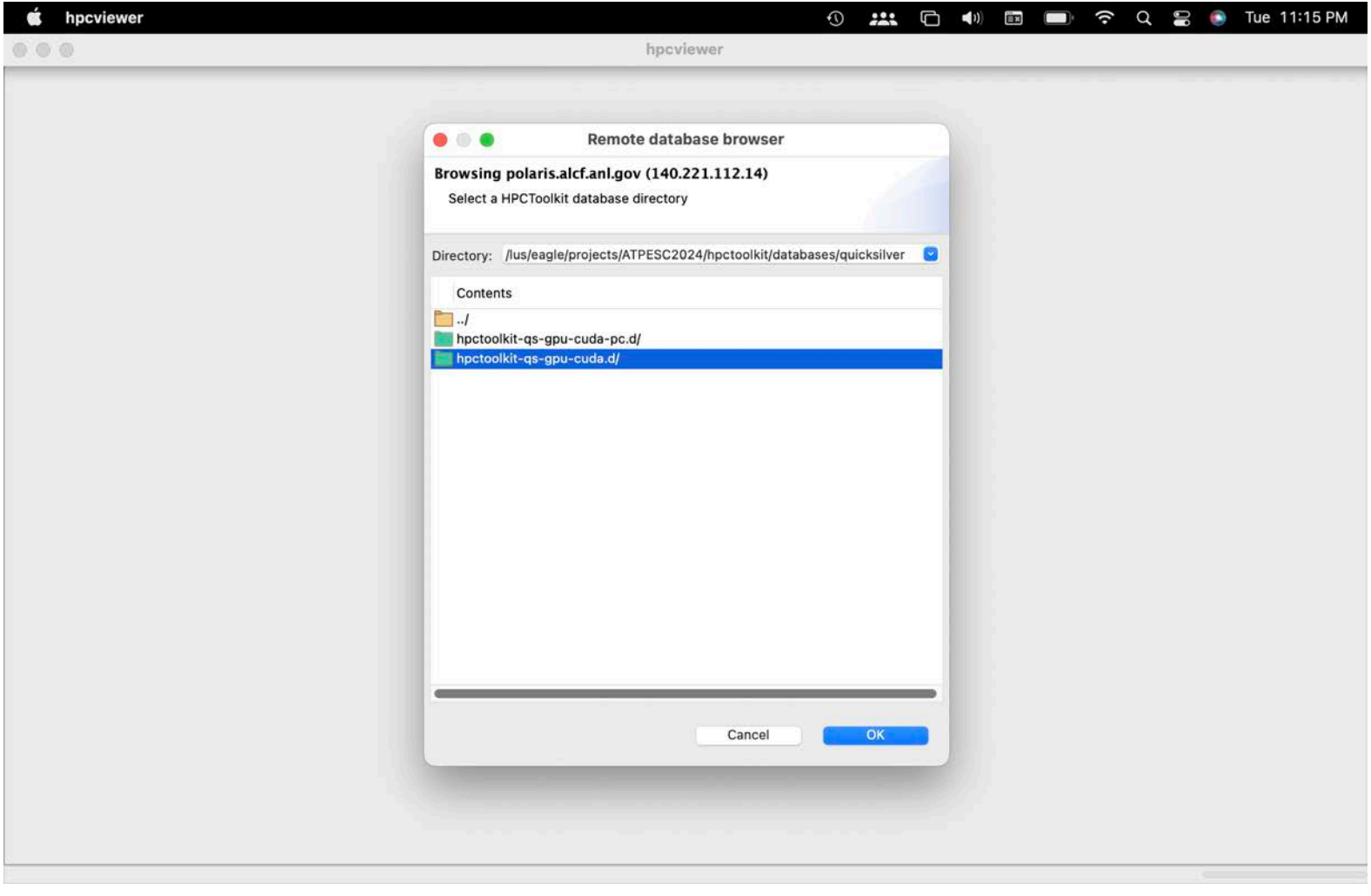
First View of Polaris: Your Home Directory



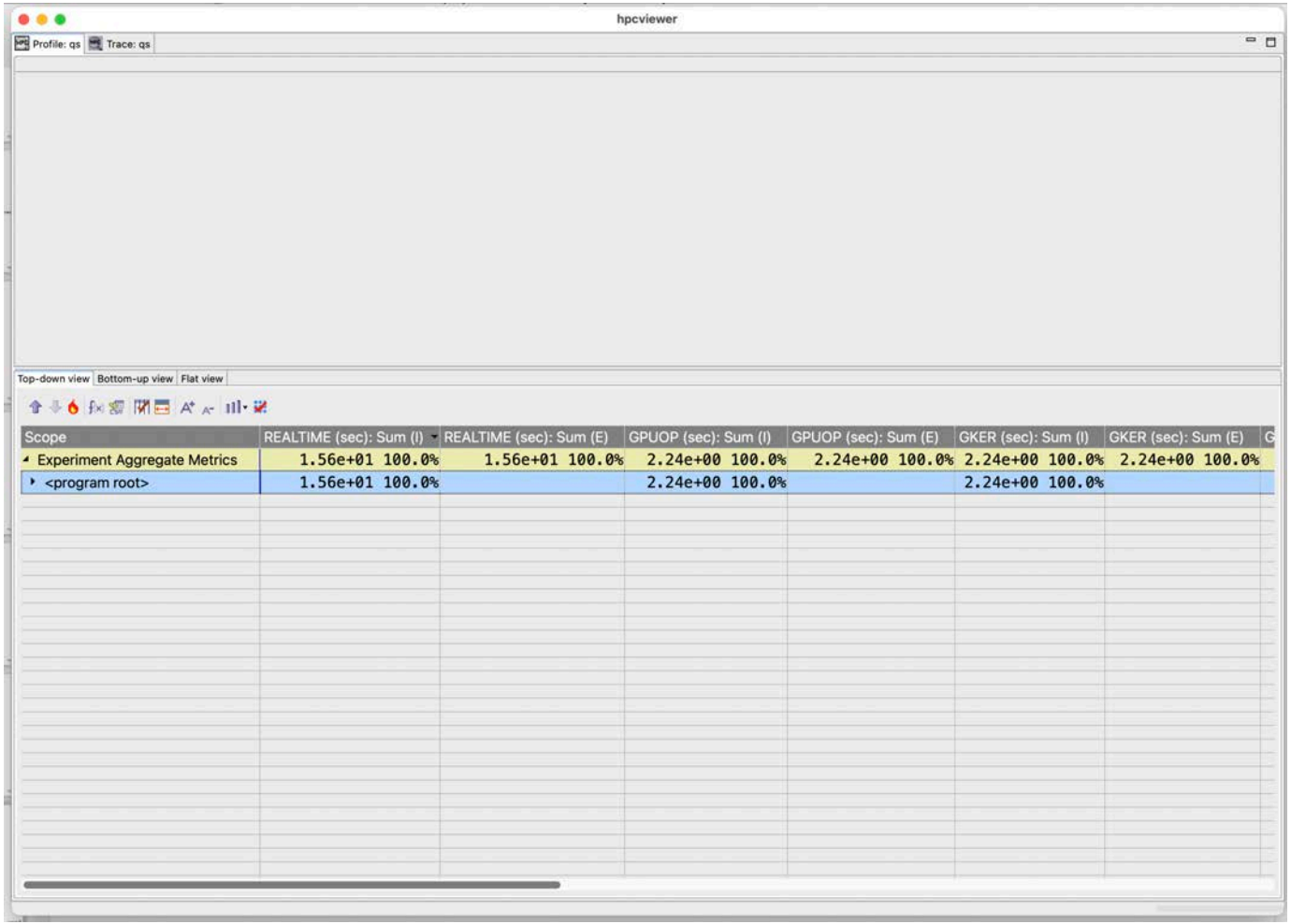
Navigate to Example Databases



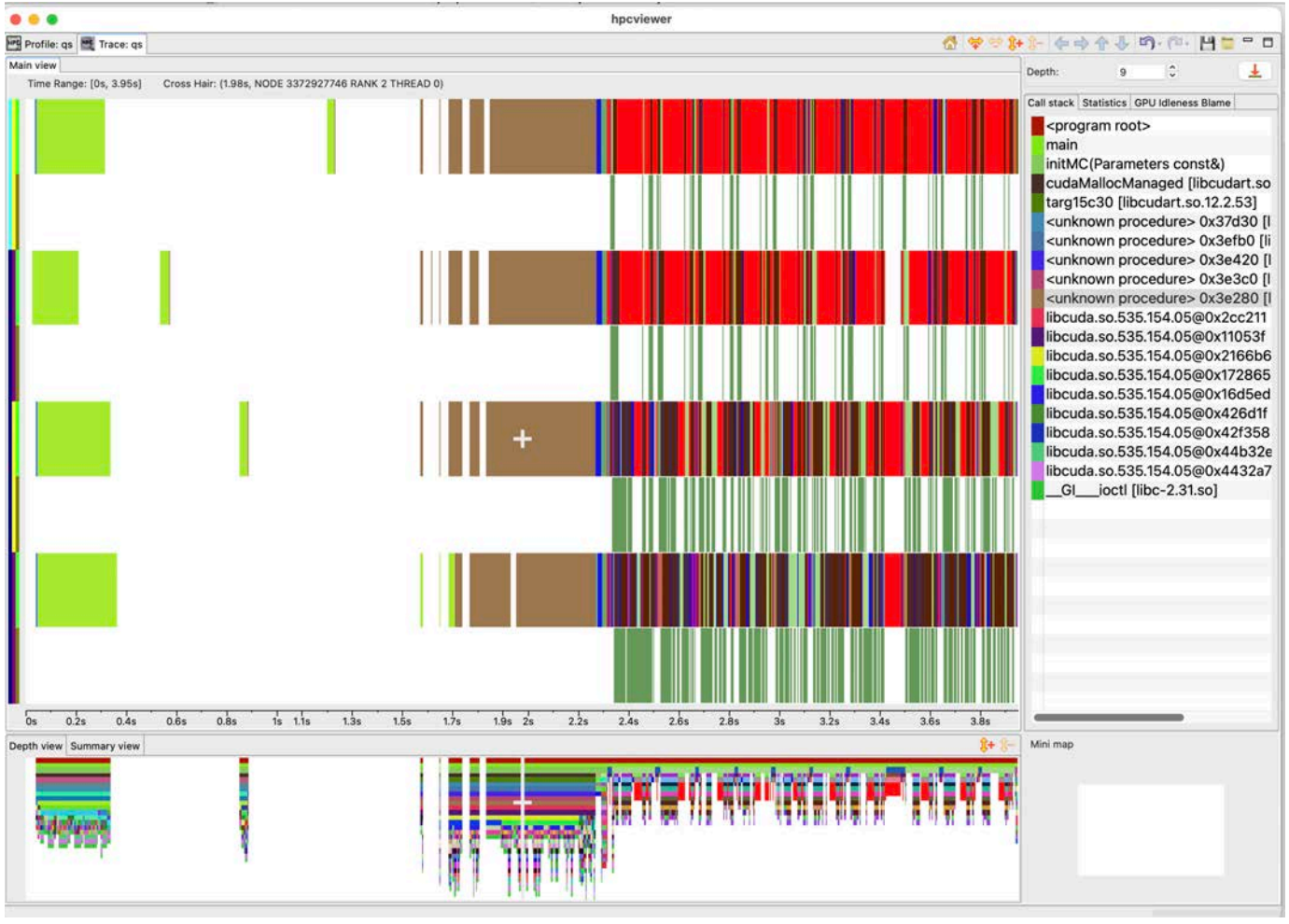
Select a Quicksilver Database with Traces



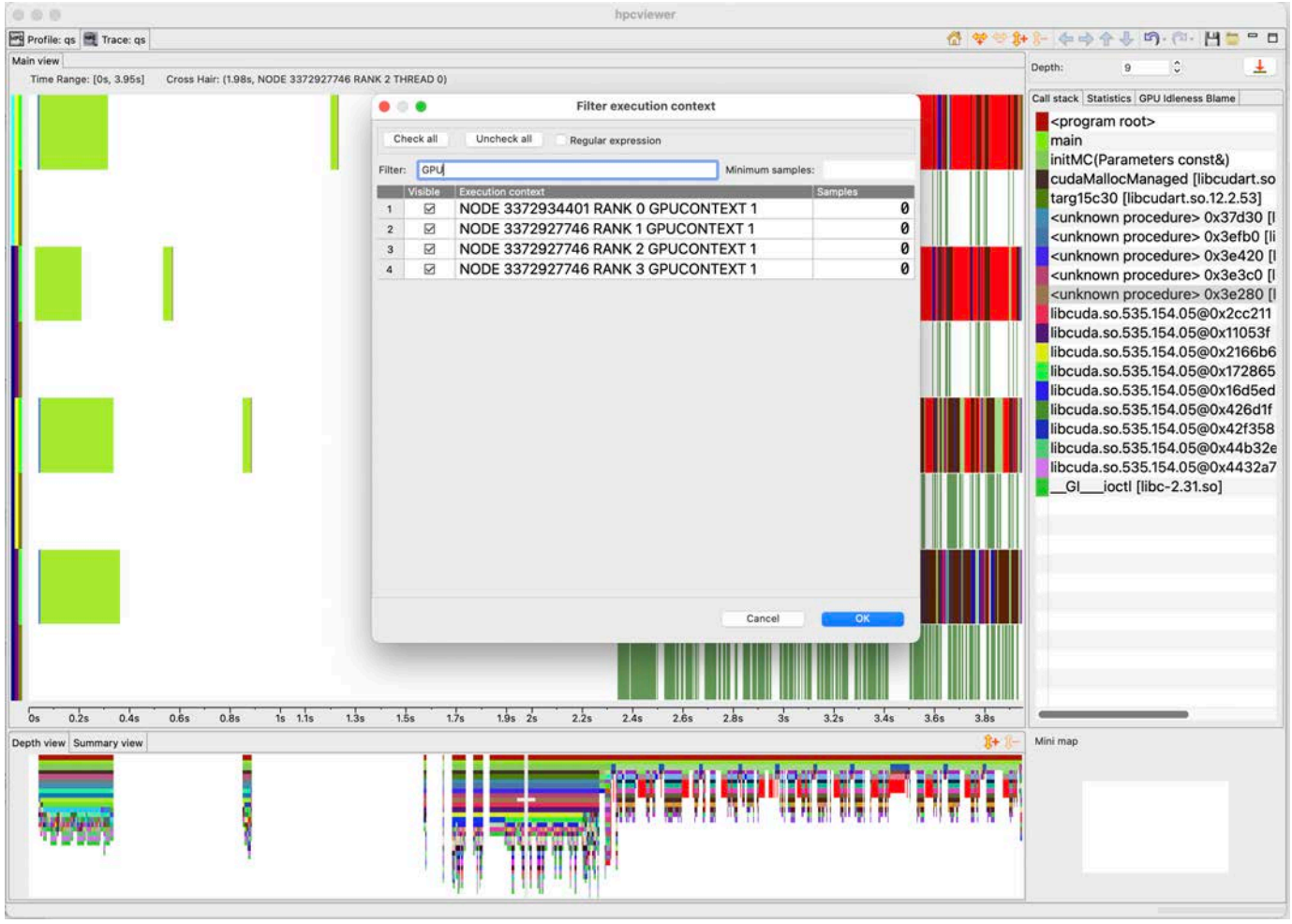
After Selecting hpctoolkit-qs-gpu-cuda.d



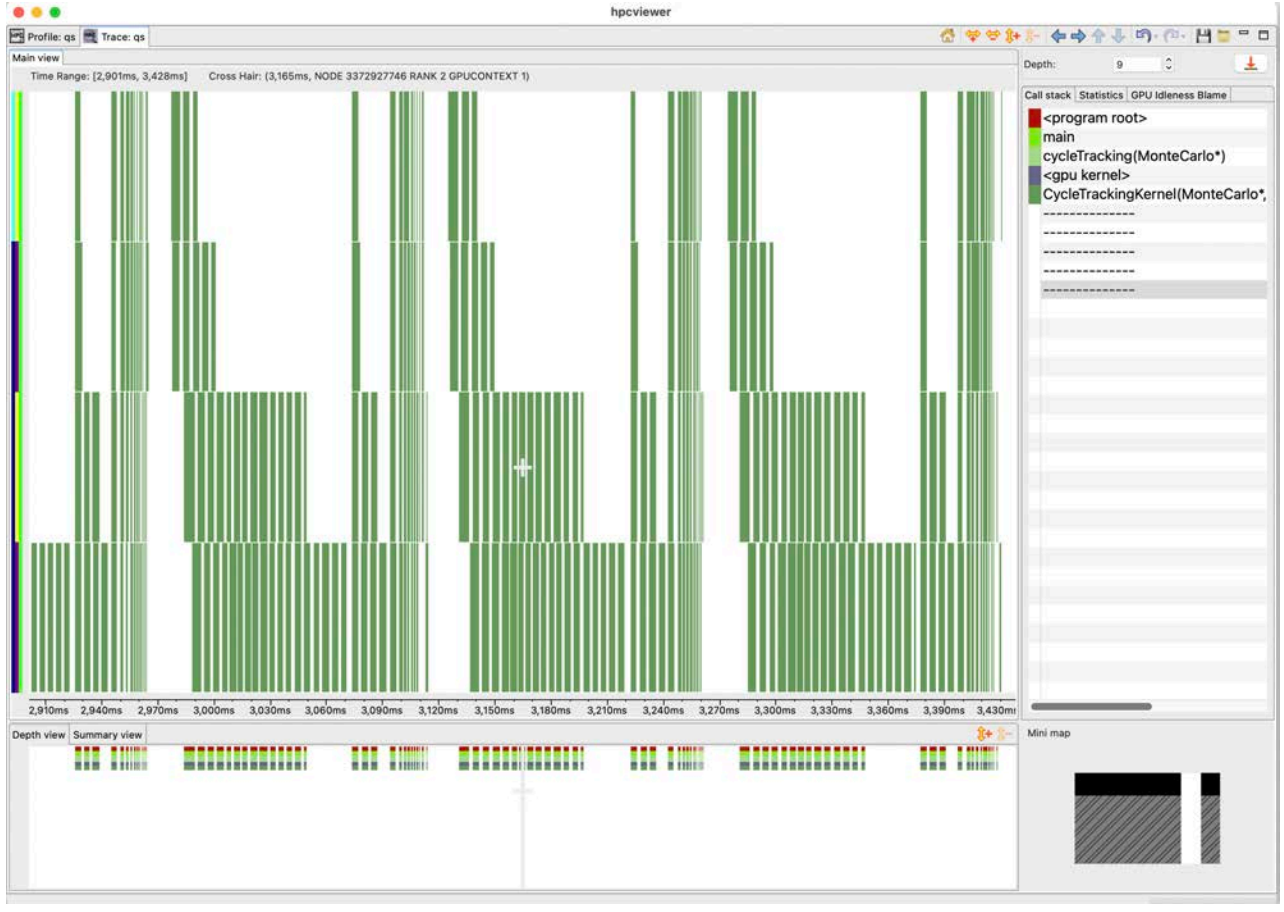
Select the Tab “Trace: qs”



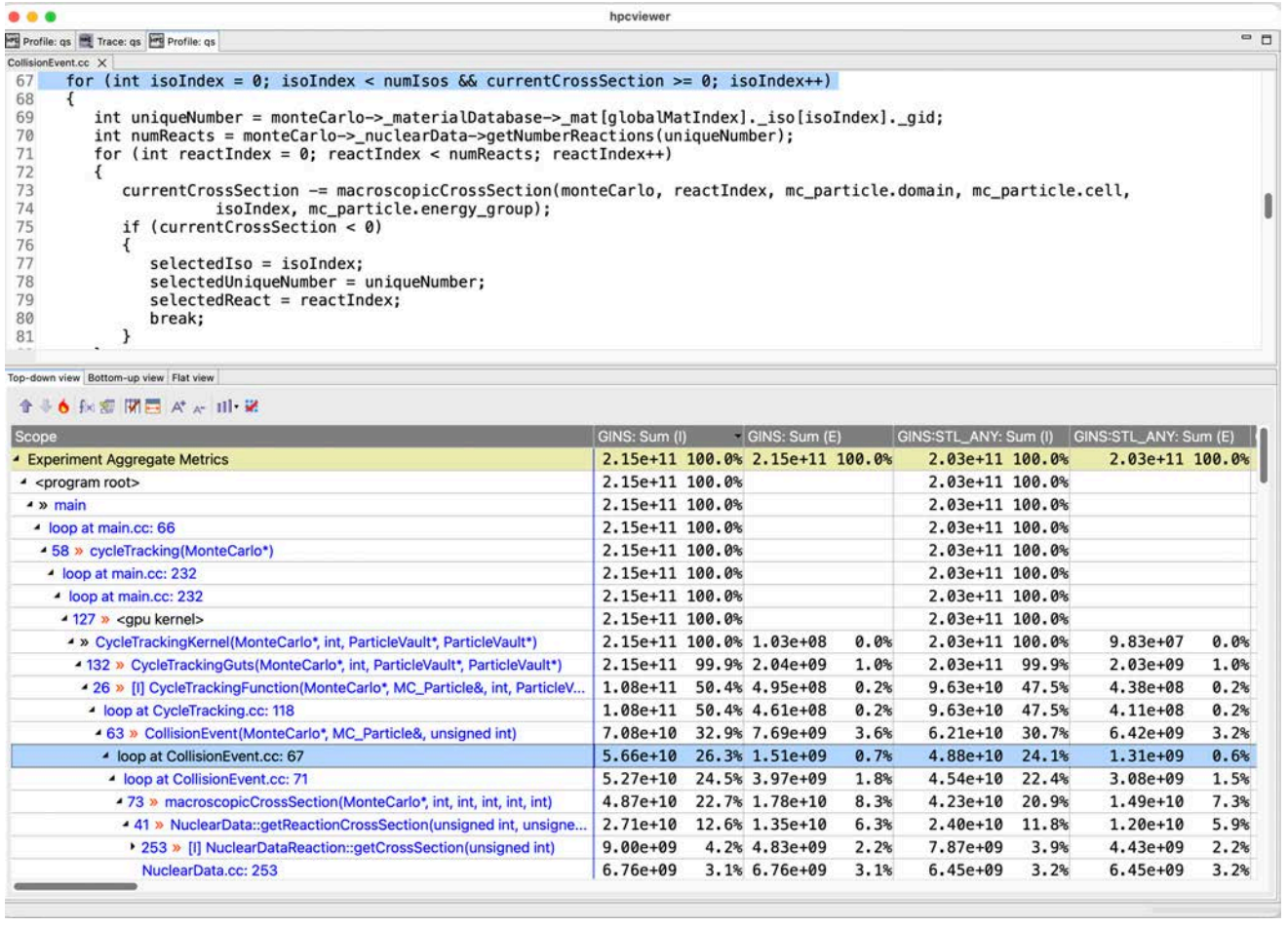
Use the Filter to “Uncheck all” and Check “GPU” streams



See Load Imbalance Across the Four GPUs



The Profile View in the other “PC Sampling” Database



Collecting Performance Data with HPCToolkit: Turnkey Examples

Hands-on Tutorial Examples

```
% git clone https://github.com/hpctoolkit/hpctoolkit-tutorial-examples
% cd hpctoolkit-tutorial-examples/gpu/nvidia
% ls
   arborx.kokkos    lammps.kokkos    quicksilver.cuda
```

A Hands-on Example: Quicksilver

A LLNL proxy application for dynamic Monte Carlo particle transport (MPI + CUDA)

```
cd hpctoolkit-tutorial-examples/gpu/nvidia/quicksilver.cuda
source setup/polaris.sh
make build
make run
make run-pc
make view
make view-pc
```

Notes

- Running “make view” or “make view-pc” requires an X11 desktop to support the GUI
- Alternatively, you can use the hpcviewer’s “open remote database” capability to view the databases
  - hpctoolkit-qs-gpu-cuda.d: profiles + traces
  - hpctoolkit-qs-gpu-cuda-pc.d: GPU PC samples

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 hid 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, but not attribute them to code

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?

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

A Hands-on Example: ArborX

Performance portable algorithms for geometric search MPI + Kokkos + OpenMP

```
cd hpctoolkit-tutorial-examples/gpu/nvidia/arborx
source setup/polaris.sh
make build
make run
make run-pc
make view
make view-pc
```

Notes

- Running “make view” or “make view-pc” requires an X11 desktop to support the GUI
- Alternatively, you can use the hpcviewer’s “open remote database” capability to view the databases
  - hpctoolkit-arborx-md.d: profiles + traces
  - hpctoolkit-arborx-md-pc.d: GPU PC samples

Analyzing ArborX Traces

Using a measurement database with profiles and traces

- Is the GPU active for most of the brief execution or not?
- Zoom in on the pair of trace lines that represents the GPU activity for a rank
    - You will see that there are two GPU trace lines per process
    - What happens on each?

A Hands-on Example: LAMMPS

A molecular dynamics code with a focus on materials modeling (Kokkos + MPI)

```
cd hpctoolkit-tutorial-examples/gpu/nvidia/lammps.cuda
source setup/polaris.sh
make build
make run
make run-pc
make view
make view-pc
```

Notes

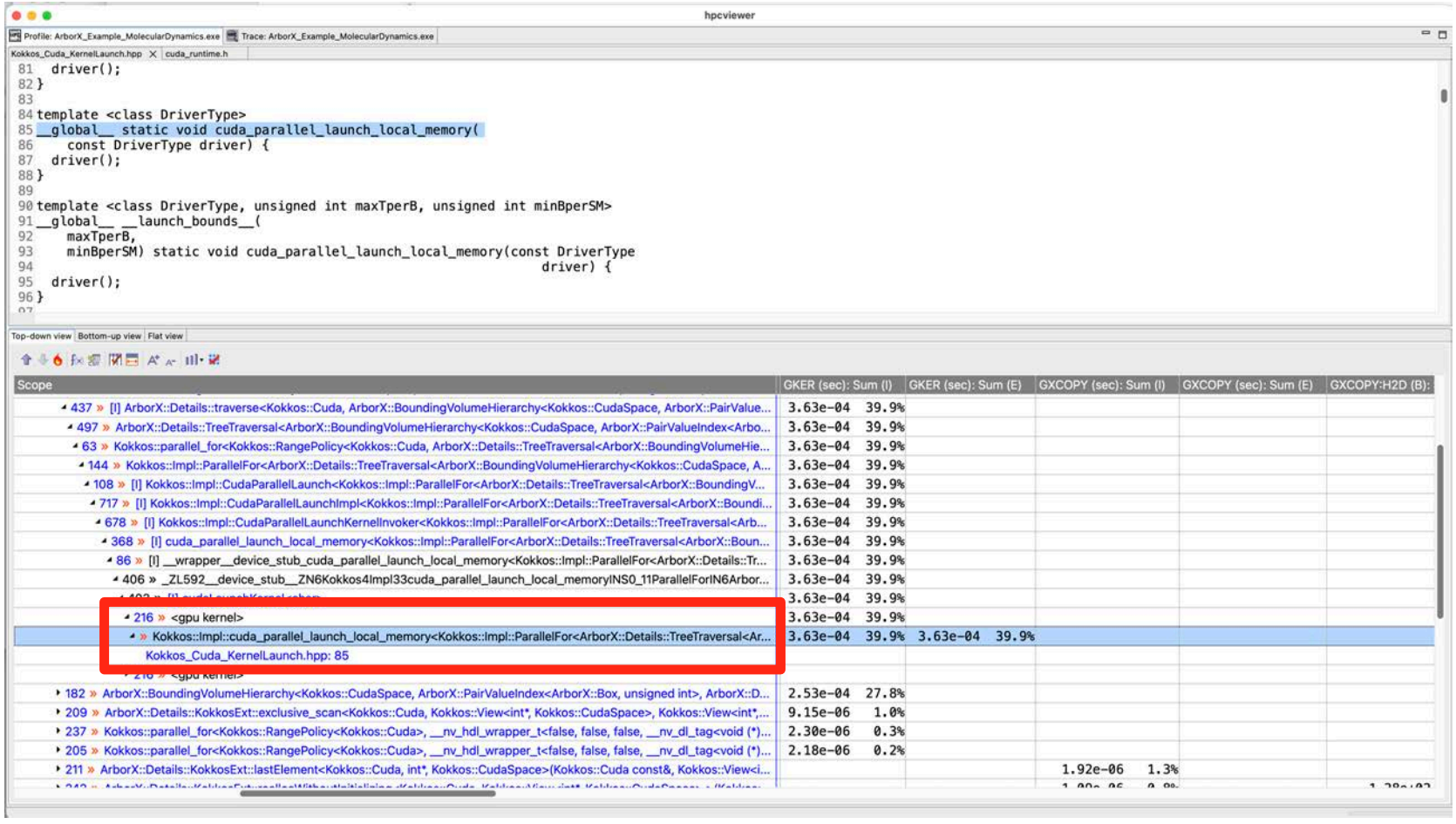
- Running “make view” or “make view-pc” requires an X11 desktop to support the GUI
- Alternatively, you can use the hpcviewer’s “open remote database” capability to view the databases
  - hpctoolkit-lmp.d: profiles and traces
  - hpctoolkit-lmp-pc.d: GPU PC samples

Analyzing LAMMPS Profiles, Traces, and PC Samples

HPCToolkit can profile, trace, and collect PC samples for codes regardless of their complexity

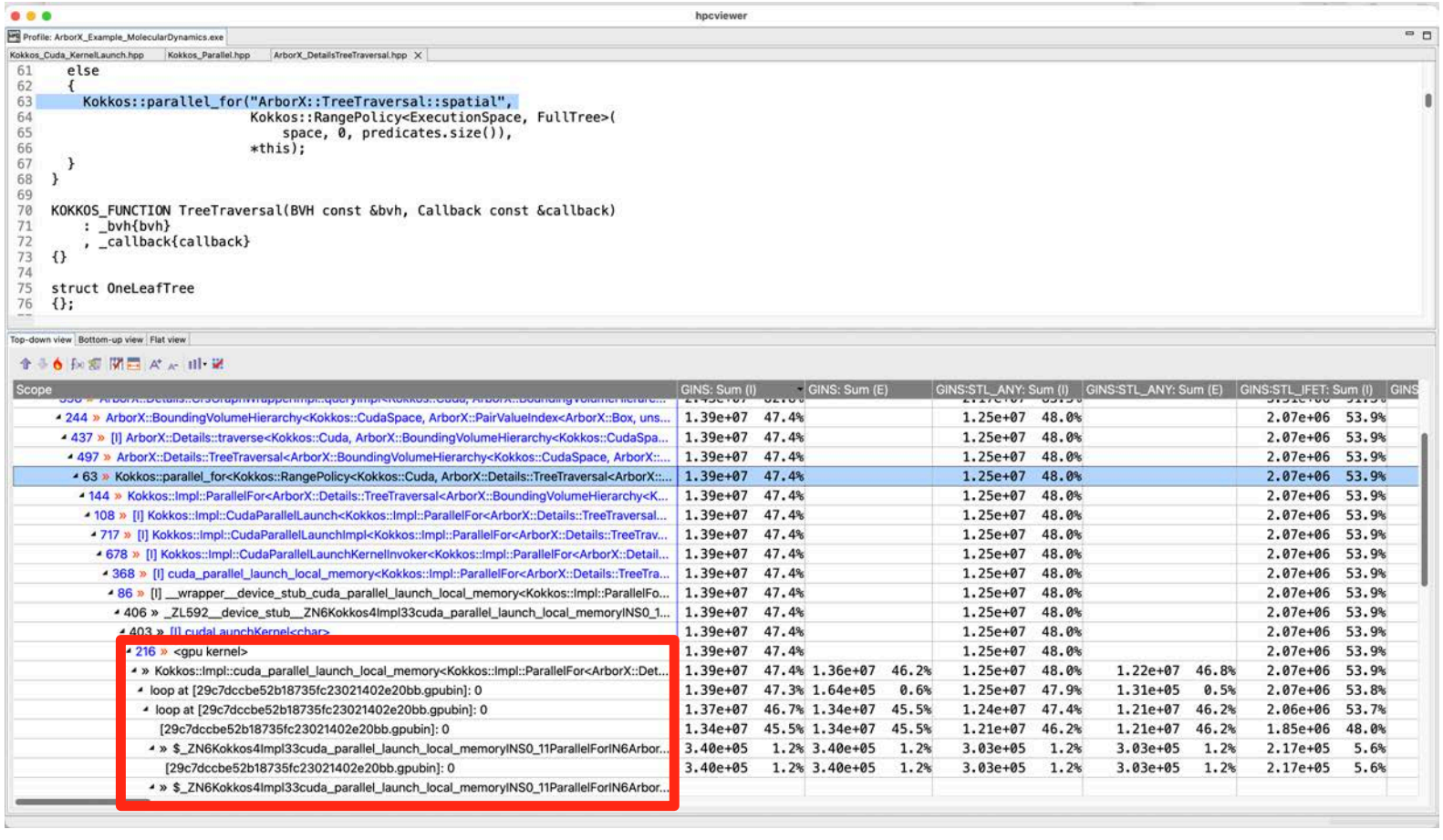
Troubleshooting: Only GPU kernel Name

- Need to measure with PC sampling to measure within GPU kernels



Troubleshooting: No GPU source code lines with PC sampling

- If you don't see source code with PC sampling on NVIDIA GPUs: compile with "-lineinfo" option



Troubleshooting: Compiling ArborX with GPU Line Map Info

- ArborX cmake isn't set up to include GPU line mappings
- Force the compiler to record GPU line mappings

- cmake -DARBORX_ENABLE_EXAMPLES=true \
  -DCMAKE_INSTALL_PREFIX=`pwd`/../install \
  -DCMAKE_CXX_COMPILER=g++ \
  -DCMAKE_BUILD_TYPE=RelWithDebInfo \
  -DCMAKE_CXX_FLAGS_RELWITHDEBINFO="-O2 -g -DNDEBUG -lineinfo"

HPCToolkit Resources

- Documentation
  - —Man pages
    - https://hpctoolkit.org/man/hpctoolkit.html
  - —User manual
    - https://hpctoolkit.org/manual/HPCToolkit-users-manual.pdf
  - —Tutorial videos
    - https://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
    - https://hpctoolkit.org/download.html
  - —Install HPCToolkit on your Linux desktop, cluster, or supercomputer using Spack
    - https://hpctoolkit.org/software-instructions.html

Current Funding for HPCToolkit

- Government
  - —Lawrence Livermore National Laboratory Subcontract B665301
  - —DOE Software Tools Ecosystem Project - UT-Battelle Subcontract CW54422
  - —Argonne National Laboratory Subcontract 4F-60094
- Corporate
  - —Advanced Micro Devices
  - —TotalEnergies EP Research & Technology USA, LLC.