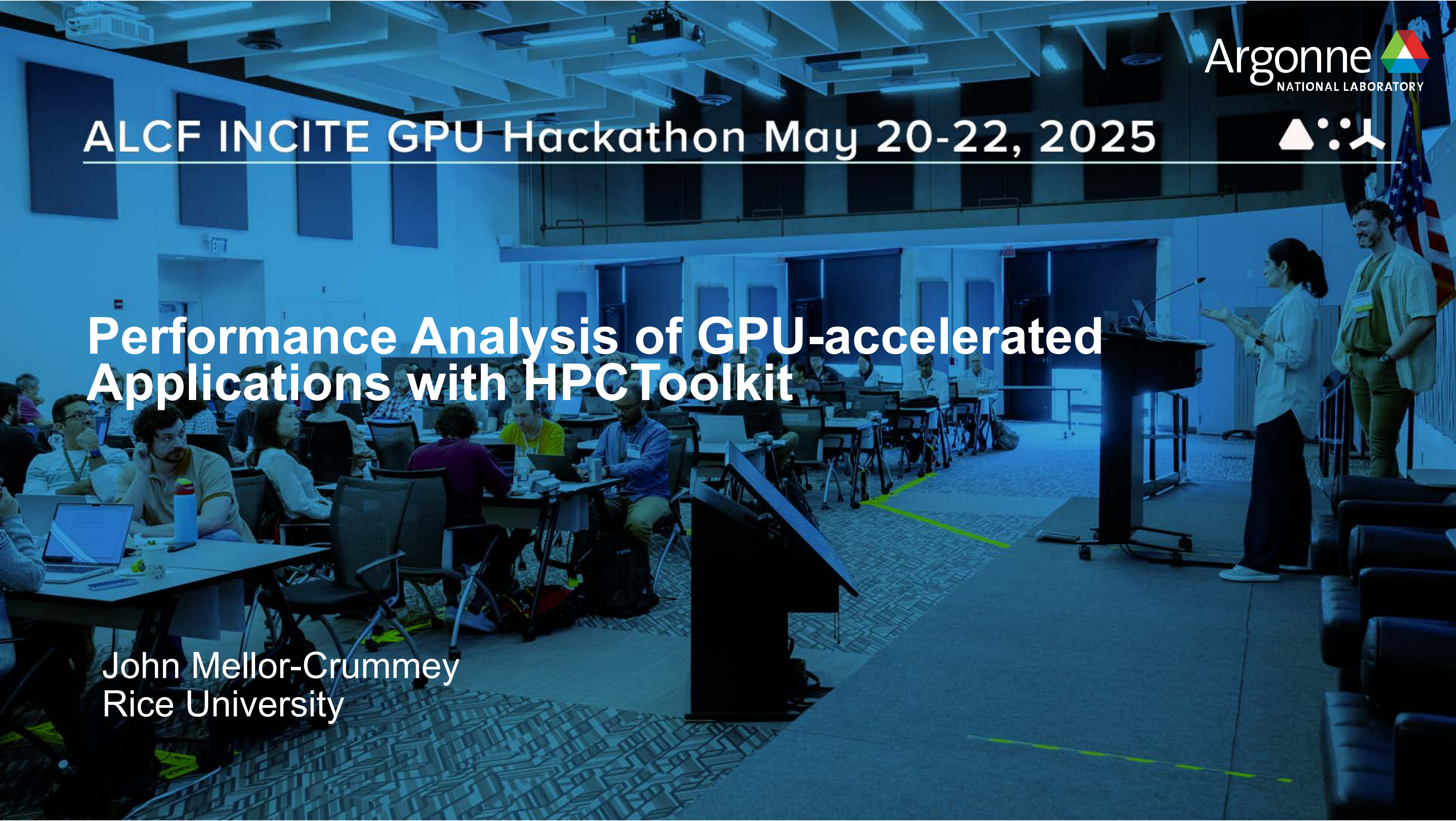


ALCF INCITE GPU Hackathon May 20-22, 2025

Performance Analysis of GPU-accelerated Applications with HPCToolkit

John Mellor-Crummey
Rice University

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:
    - —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?

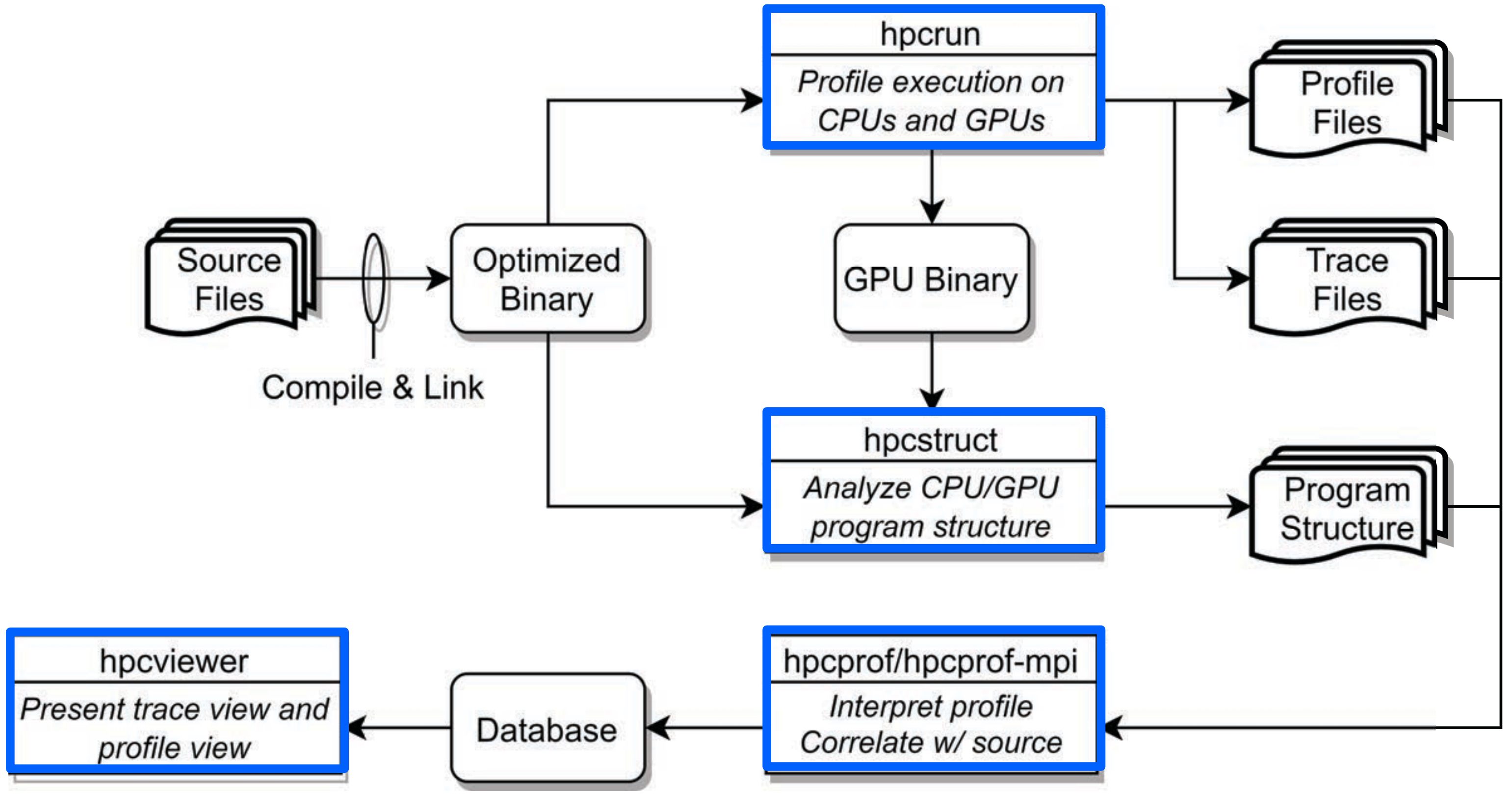
- Measure and analyze performance of CPU and GPU-accelerated applications
- Easy: profile unmodified application binaries
- Fast: low-overhead measurement
- Informative: understand where an application spends its time and why
  - —call path profiles associate metrics with application source code contexts
  - —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?

- 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 Omtitrace: designed for analysis of performance on a single node
- HPCToolkit
  - —supports more scalable tracing than vendor tools
    - ▪ 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

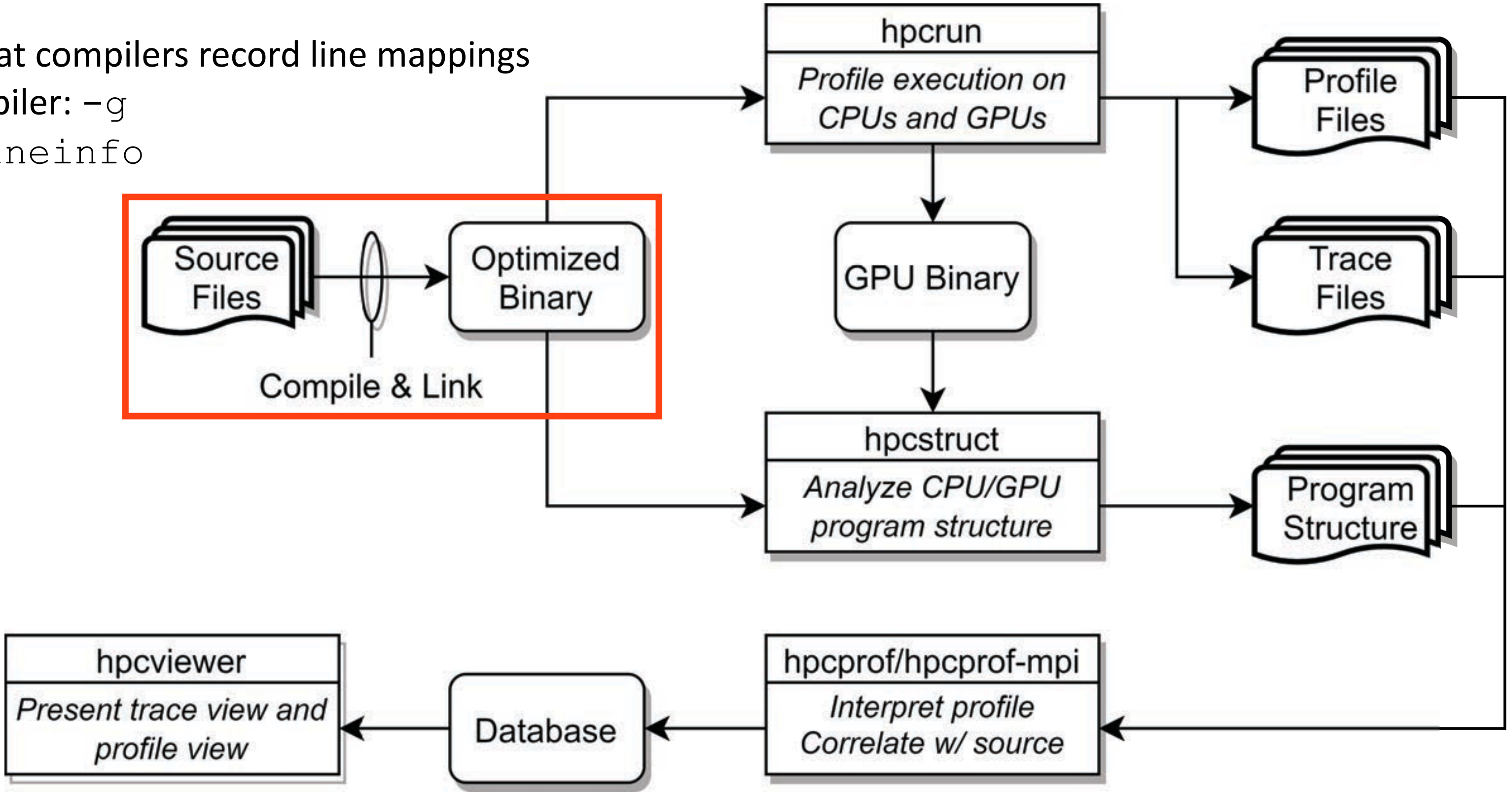
HPCToolkit’s Workflow for GPU-accelerated Applications



HPCToolkit's Workflow for GPU-accelerated Applications

Step 1:

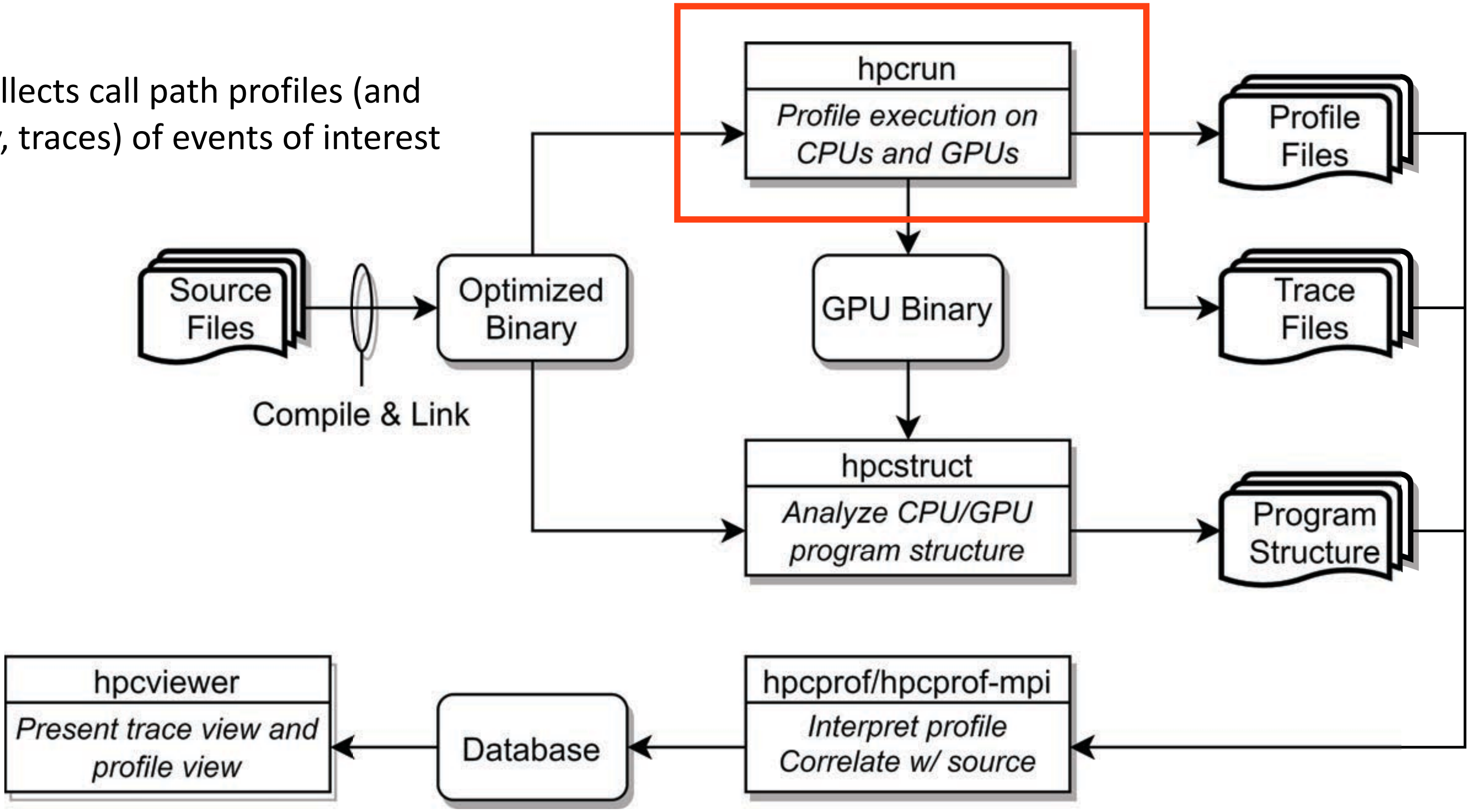
- Ensure that compilers record line mappings
- host compiler: `-g`
- nvcc: `-lineinfo`



HPCToolkit’s Workflow for GPU-accelerated Applications

Step 2:

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



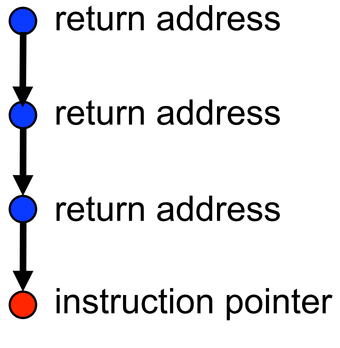
Measurement of CPU and GPU-accelerated Applications

- 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 (in progress: AMD, Intel)
- Binary instrumentation of GPU kernels on Intel GPUs for fine-grain measurement

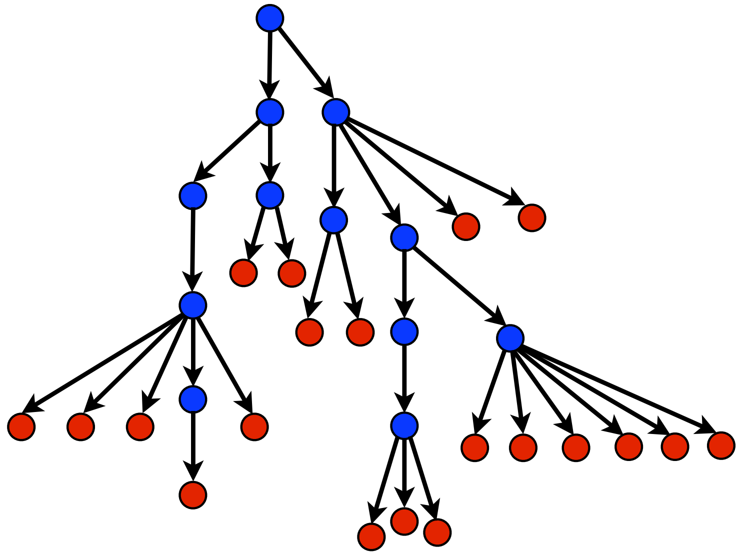
Call Stack Unwinding to Attribute Costs in Context

- 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



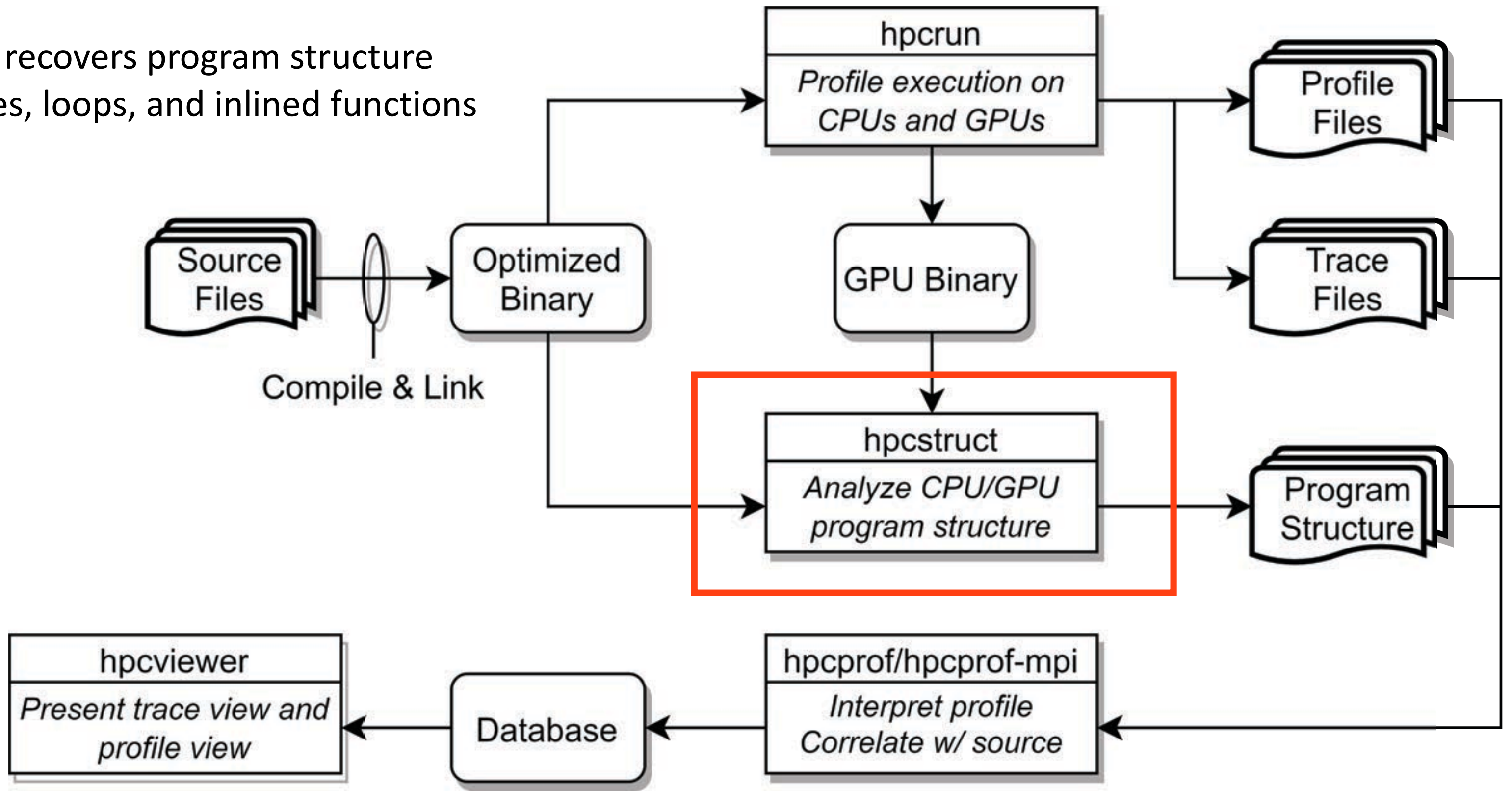
hpcrun: Measure CPU and/or GPU activity

- 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 -tt <app>`
- Use hpcrun with MPI on Polaris or Aurora
  - `mpiexec -n <ranks> … hpcrun -e gpu=xxx <app>`

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

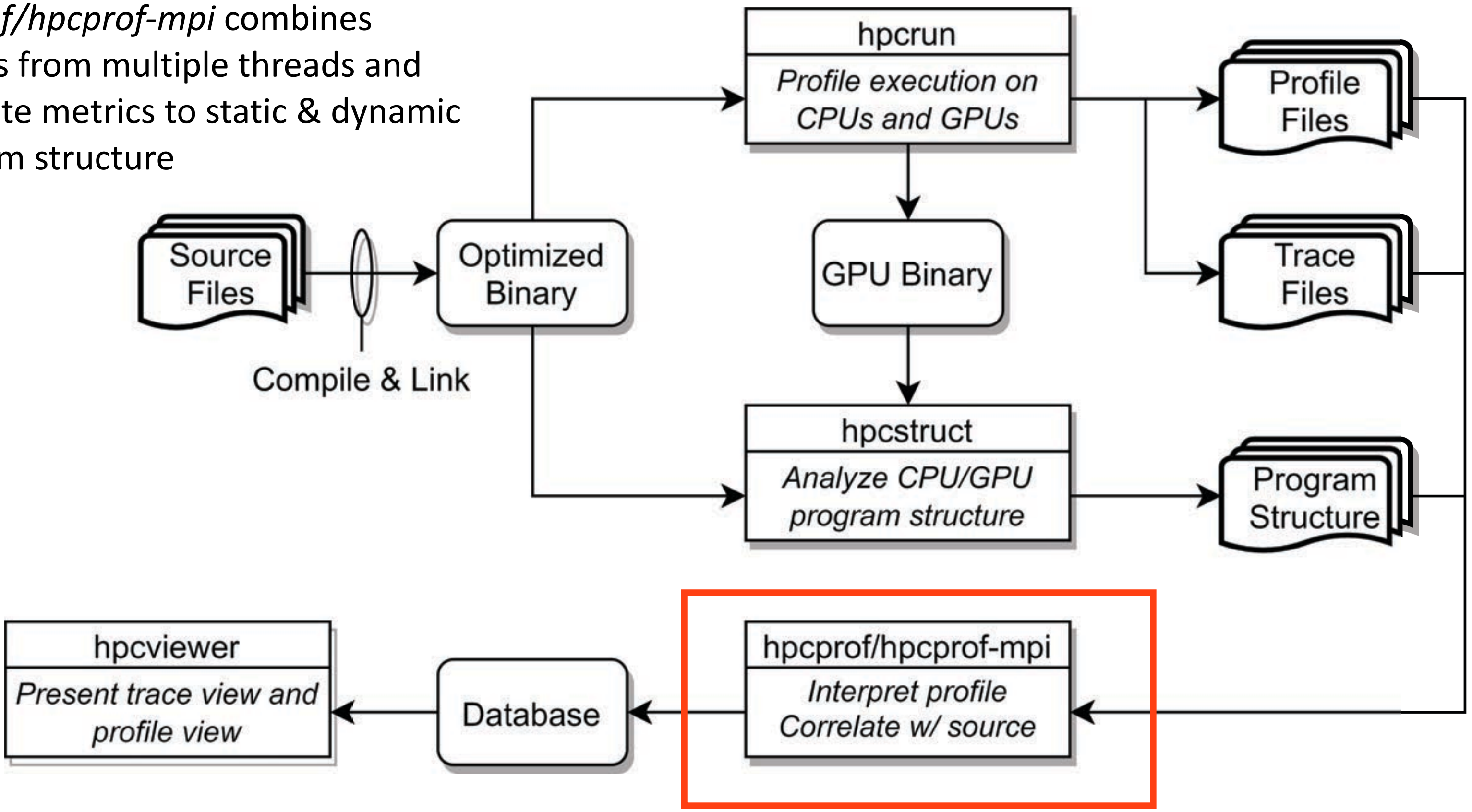
    —typically analyze multiple small application binaries concurrently with 2 threads each
  - Cache binary analysis results for reuse when analyzing other executions

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

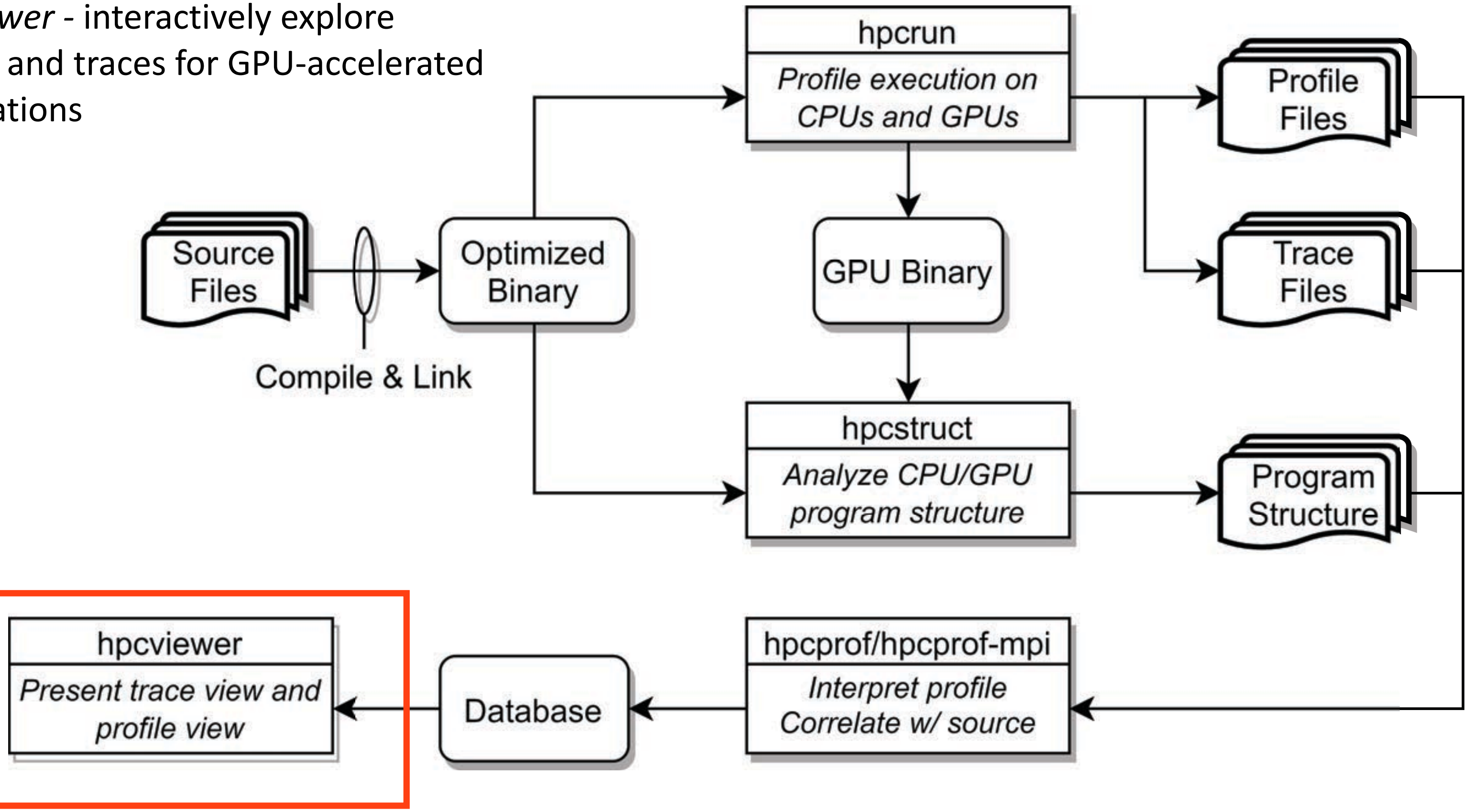
- Analyze data from large executions with distributed-memory parallelism + multithreading (large scale)

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

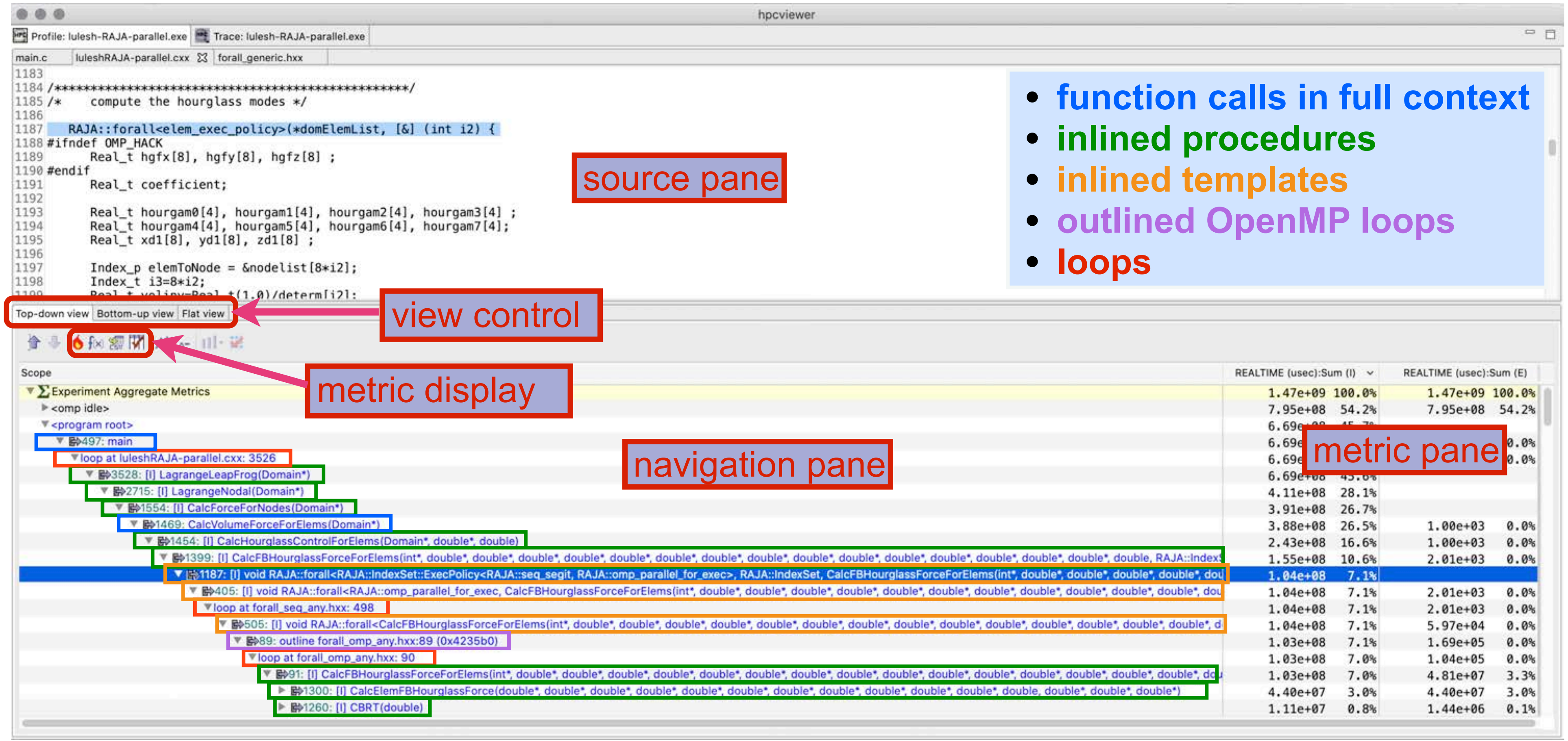
HPCToolkit’s Workflow for GPU-accelerated Applications

Step 4:

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



Code-centric Analysis with hpcviewer



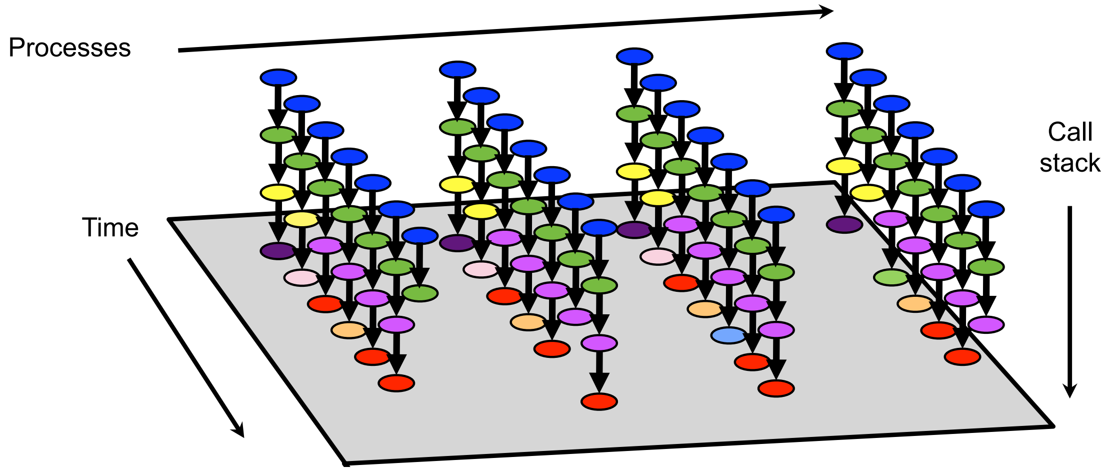
- function calls in full context
- inlined procedures
- inlined templates
- outlined OpenMP loops
- loops

```
/********************************************************/
/*    compute the hourglass modes */

   RAJA::forall<elem_exec_policy>(*domElemList, [&] (int i2) {
#ifndef OMP_HACK
        Real_t hgfx[8], hgfy[8], hgfz[8] ;
#endif
        Real_t coefficient;

        Real_t hourgam0[4], hourgam1[4], hourgam2[4], hourgam3[4] ;
        Real_t hourgam4[4], hourgam5[4], hourgam6[4], hourgam7[4];
        Real_t xd1[8], yd1[8], zd1[8] ;

        Index_p elemToNode = &nodelist[8*i2];
        Index_t i3=8*i2;
        Real_t volinv=Real_t(1.0)/determ[i2];
```

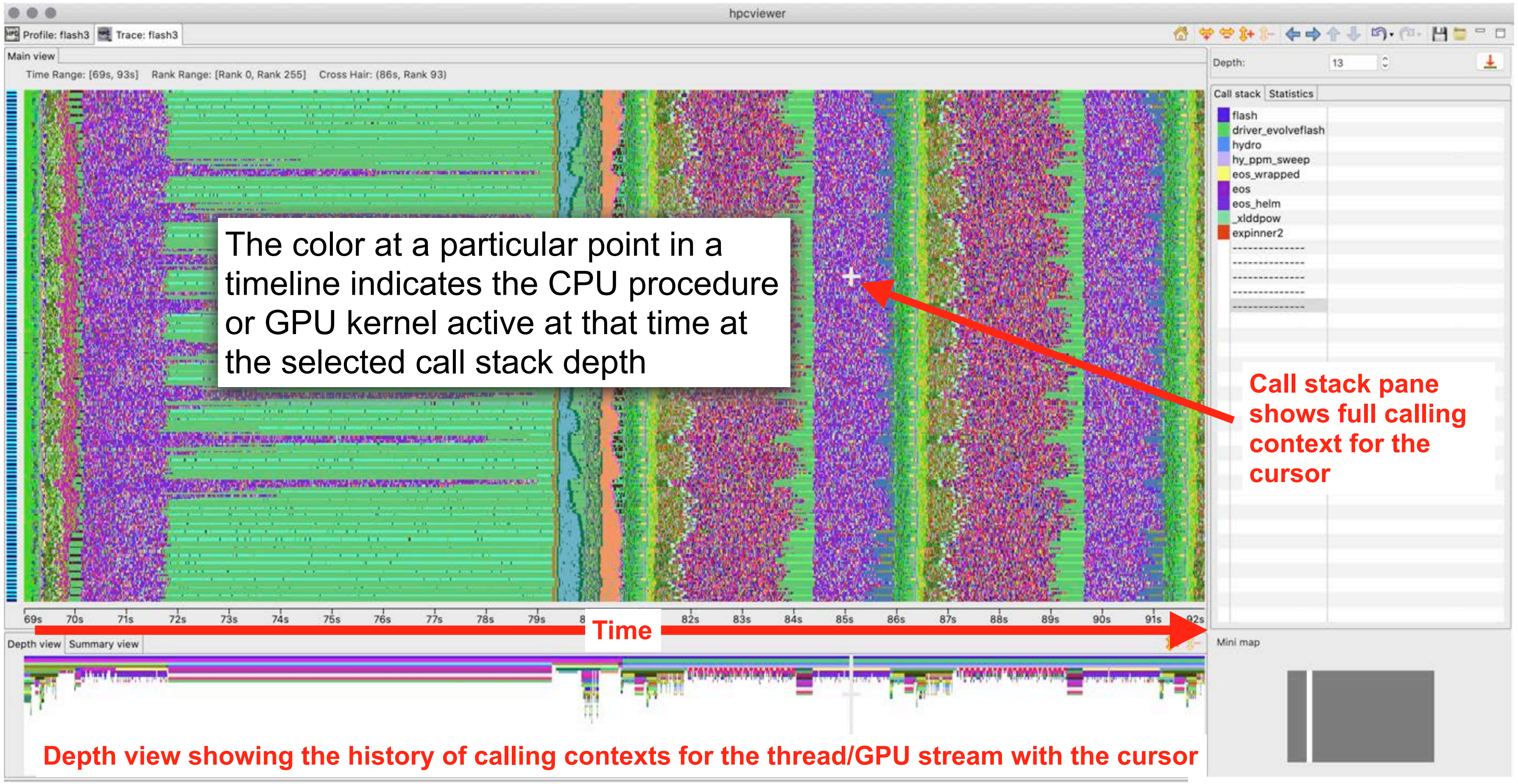
```
Scope                                                                                REALTIME (usec):Sum (I)   REALTIME (usec):Sum (E)
▼ Σ Experiment Aggregate Metrics                                                      1.47e+09 100.0%          1.47e+09 100.0%
  ▶ <omp idle>                                                                         7.95e+08  54.2%          7.95e+08  54.2%
  ▼ <program root>                                                                     6.69e
    ▼ 497: main                                                                        6.69e                              0.0%
      ▼ loop at luleshRAJA-parallel.cxx: 3526                                          6.69e                              0.0%
        ▼ 3528: [I] LagrangeLeapFrog(Domain*)                                          6.69e+08
          ▼ 2715: [I] LagrangeNodal(Domain*)                                           4.11e+08  28.1%
            ▼ 1554: [I] CalcForceForNodes(Domain*)                                     3.91e+08  26.7%
              ▼ 1469: CalcVolumeForceForElems(Domain*)                                 3.88e+08  26.5%          1.00e+03   0.0%
                ▼ 1454: [I] CalcHourglassControlForElems(Domain*, double*, double)     2.43e+08  16.6%          1.00e+03   0.0%
                  ▼ 1399: [I] CalcFBHourglassForceForElems(int*, double*, double*, double*, double*, double*, double*, double*, double*, double*, double*, double*, double*, double*, double*, double*, double, RAJA::IndexS   1.55e+08  10.6%   2.01e+03   0.0%
                    ▼ 1187: [I] void RAJA::forall<RAJA::IndexSet::ExecPolicy<RAJA::seq_segit, RAJA::omp_parallel_for_exec>, RAJA::IndexSet, CalcFBHourglassForceForElems(int*, double*, double*, double*, double*, dou   1.04e+08   7.1%
                      ▼ 405: [I] void RAJA::forall<RAJA::omp_parallel_for_exec, CalcFBHourglassForceForElems(int*, double*, double*, double*, double*, double*, double*, double*, double*, double*, double*, dou   1.04e+08   7.1%   2.01e+03   0.0%
                        ▼ loop at forall_seq_any.hxx: 498                              1.04e+08   7.1%          2.01e+03   0.0%
                          ▼ 505: [I] void RAJA::forall<CalcFBHourglassForceForElems(int*, double*, double*, double*, double*, double*, double*, double*, double*, double*, double*, double*, double*, double*, d   1.04e+08   7.1%   5.97e+04   0.0%
                            ▼ 89: outline forall_omp_any.hxx:89 (0x4235b0)             1.03e+08   7.1%          1.69e+05   0.0%
                              ▼ loop at forall_omp_any.hxx: 90                         1.03e+08   7.0%          1.04e+05   0.0%
                                ▼ 91: [I] CalcFBHourglassForceForElems(int*, double*, double*, double*, double*, double*, double*, double*, double*, double*, double*, double*, double*, double*, double*, do   1.03e+08   7.0%   4.81e+07   3.3%
                                  ▶ 1300: [I] CalcElemFBHourglassForce(double*, double*, double*, double*, double*, double*, double*, double*, double*, double*, double, double*, double*, double*)   4.40e+07   3.0%   4.40e+07   3.0%
                                  ▶ 1260: [I] CBRT(double)                             1.11e+07   0.8%          1.44e+06   0.1%
```

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



The color at a particular point in a timeline indicates the CPU procedure or GPU kernel active at that time at the selected call stack depth

Call stack pane shows full calling context for the cursor

Depth view showing the history of calling contexts for the thread/GPU stream with the cursor

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

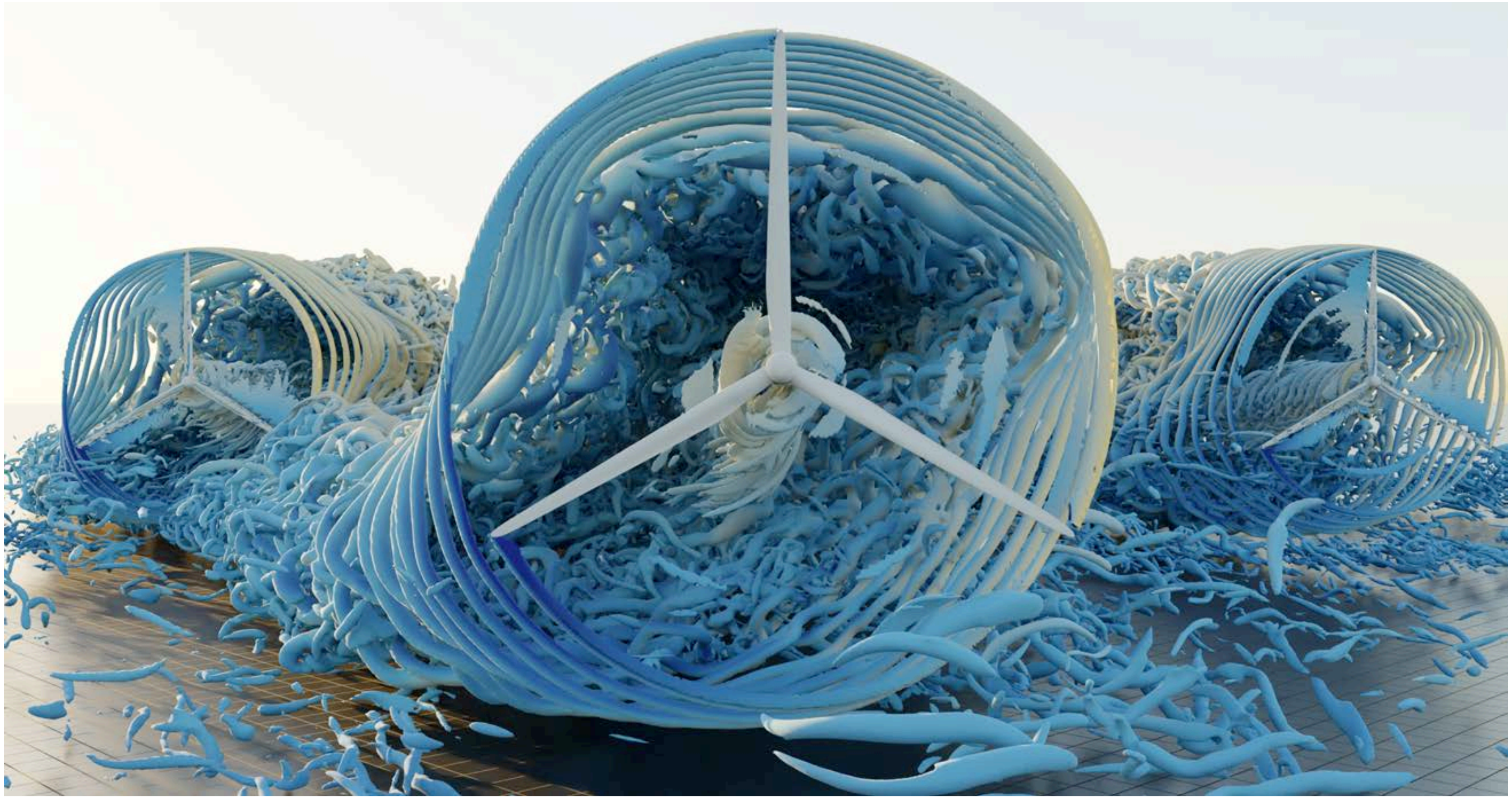


Figure credit: Jon Rood, NREL

ExaWind: Visualization of a Wind Farm Simulation

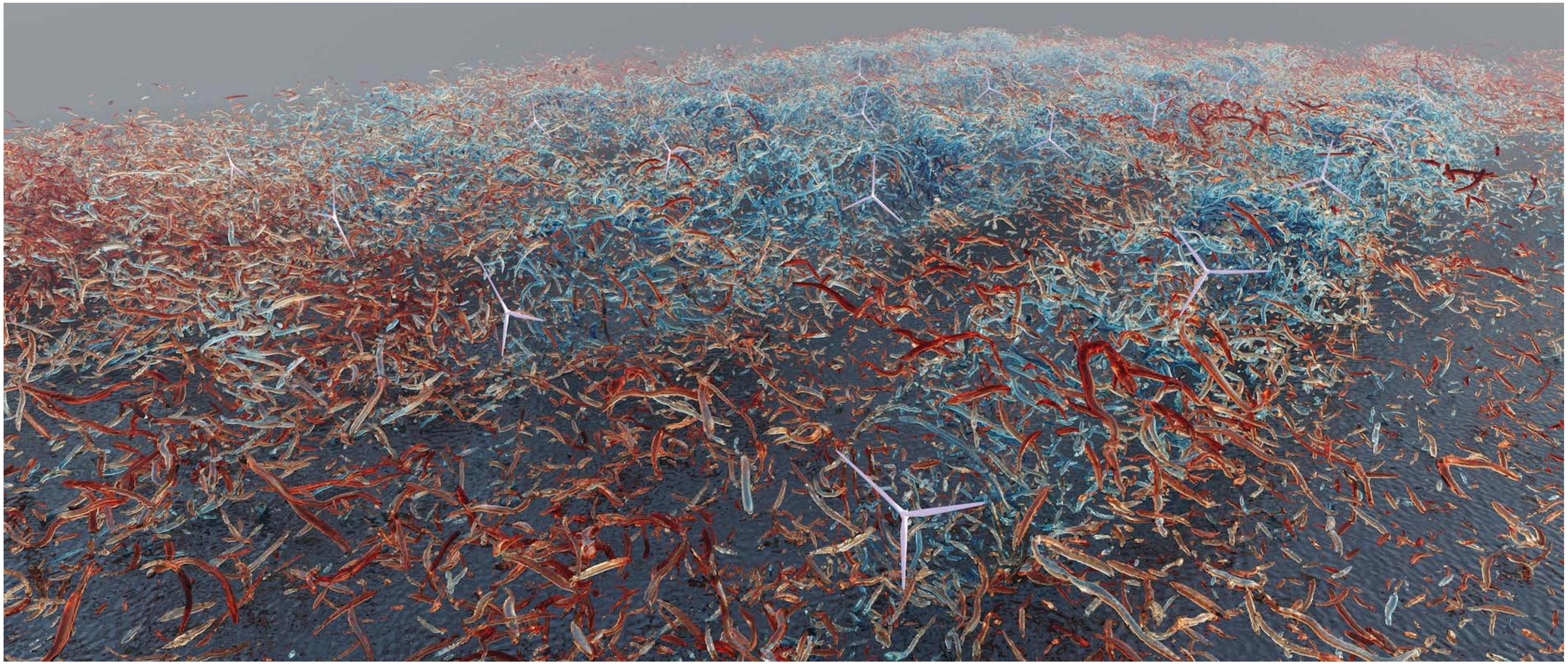


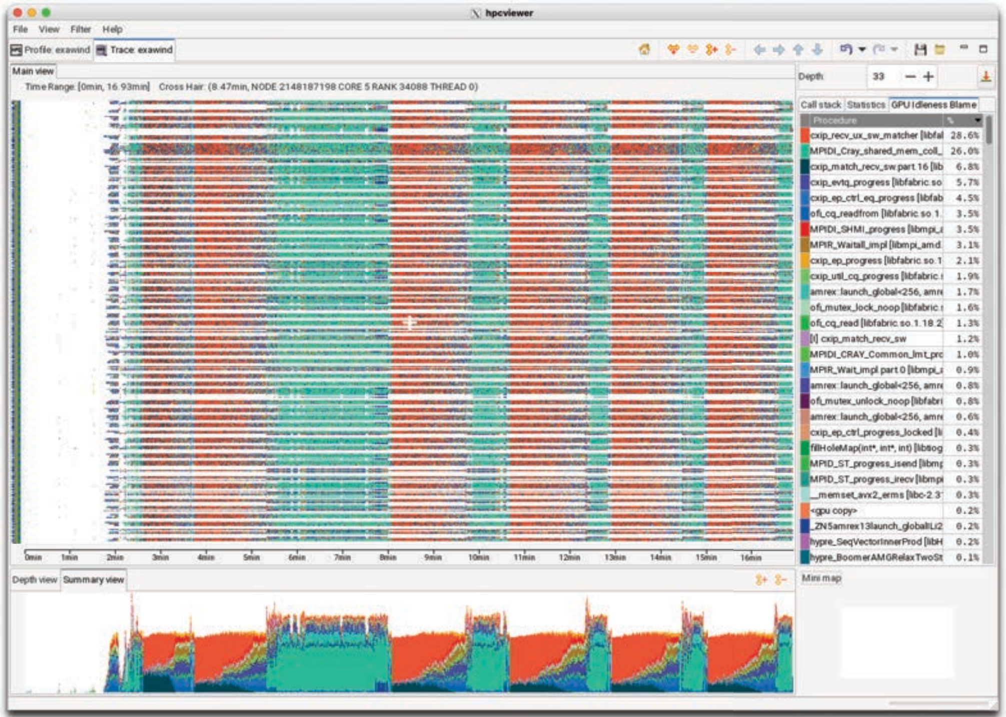
Figure credit: Jon Rood, NREL

ExaWind: Execution Traces on Frontier Collected with HPCToolkit

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

Before: MPI waiting (bad), shown in red

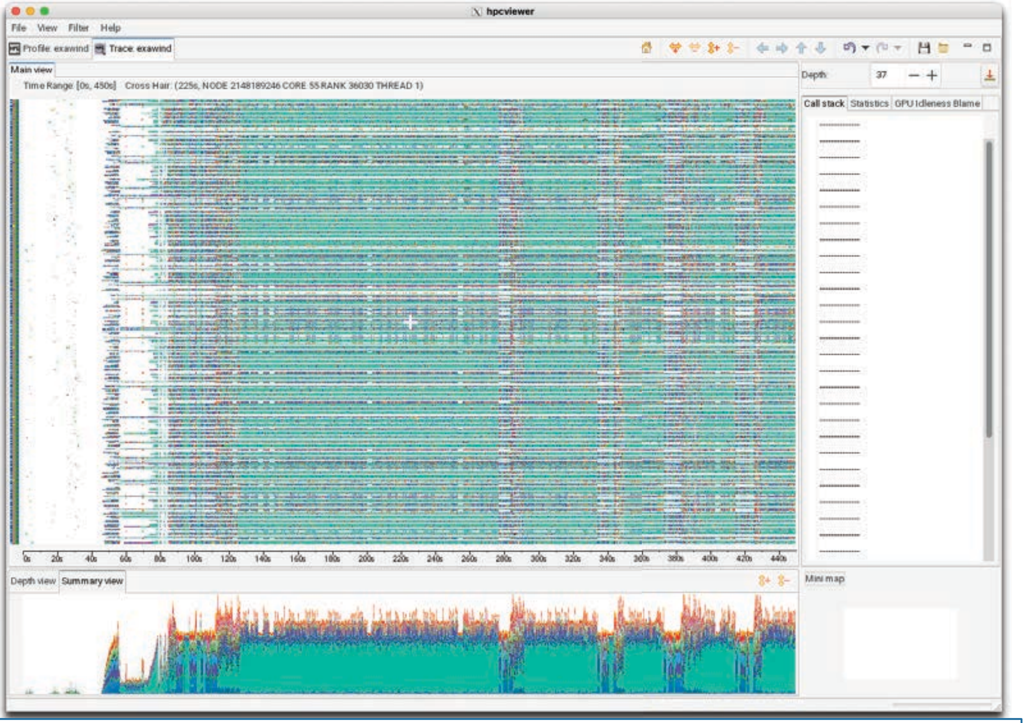
After: MPI overhead negligible*



Time Range: [0min, 16.93min] Cross Hair: (8.47min, NODE 2148187198 CORE 5 RANK 34088 THREAD 0)

Depth: 33

	Procedure	%
🟥	cxip_recv_ux_sw_matcher [libfal	28.6%
🟩	MPIDI_Cray_shared_mem_coll_	26.0%
■	cxip_match_recv_sw.part.16 [lib	6.8%
🟦	cxip_evtq_progress [libfabric.so	5.7%
🟦	cxip_ep_ctrl_eq_progress [libfab	4.5%
🟦	ofi_cq_readfrom [libfabric.so.1	3.5%
🟥	MPIDI_SHMI_progress [libmpi_	3.5%
🟫	MPIR_Waitall_impl [libmpi_amd	3.1%
🟧	cxip_ep_progress [libfabric.so.1	2.1%
🟩	cxip_util_cq_progress [libfabric.	1.9%
🟩	amrex::launch_global<256, amre	1.7%
🟩	ofi_mutex_lock_noop [libfabric.	1.6%
🟩	ofi_cq_read [libfabric.so.1.18.2	1.3%
🟪	[I] cxip_match_recv_sw	1.2%
🟩	MPIDI_CRAY_Common_lmt_pro	1.0%
🟦	MPIR_Wait_impl.part.0 [libmpi_	0.9%
🟪	amrex::launch_global<256, amre	0.8%
🟪	ofi_mutex_unlock_noop [libfabri	0.8%
🟫	amrex::launch_global<256, amre	0.6%
🟧	cxip_ep_ctrl_progress_locked [li	0.4%
🟩	fillHoleMap(int*, int*, int) [libtiog	0.3%
🟩	MPID_ST_progress_isend [libmp	0.3%
🟩	MPID_ST_progress_irecv [libmpi	0.3%
🟩	__memset_avx2_erms [libc-2.3	0.3%
🟧	<gpu copy>	0.2%
🟦	_ZN5amrex13launch_globalILi2	0.2%
🟪	hypre_SeqVectorInnerProd [libH	0.2%
🟦	hypre_BoomerAMGRelaxTwoSt	0.1%



Time Range: [0s, 450s] Cross Hair: (225s, NODE 2148189246 CORE 55 RANK 36030 THREAD 1)

Depth: 37

Figure credits: Jon Rood, NREL

*replaced non-blocking send/recv with ialltoallv

ExaWind Testimonials for HPCToolkit

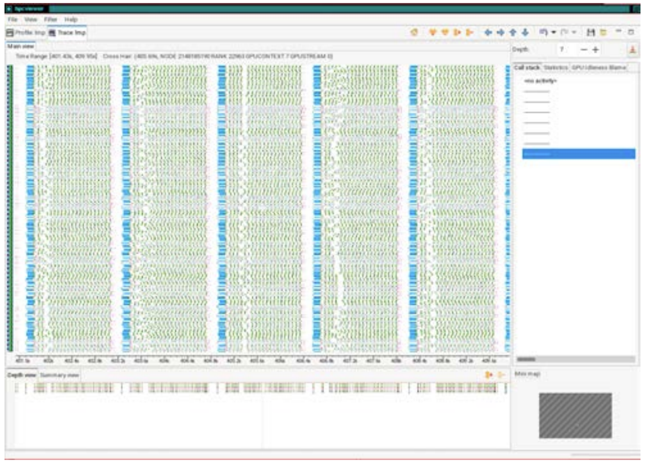
I just wanted to mention we've been using HPCToolkit a lot for our ExaWind application on Frontier, which is a hugely complicated code, and your profiler is one of the only ones we've found that really lets us easily instrument and then browse what our application is doing at runtime including GPUs.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

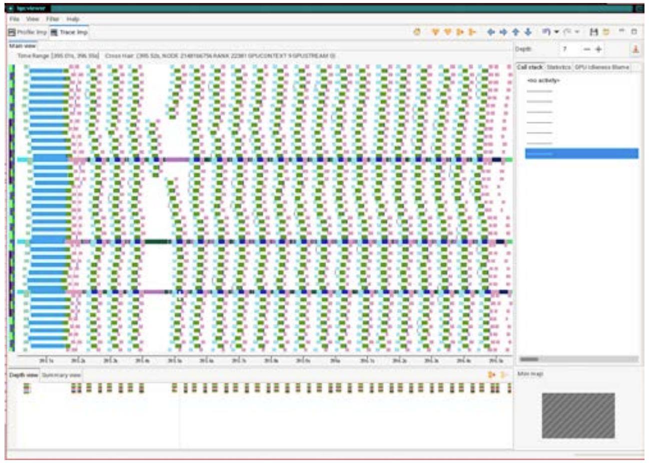
- Jon Rood NREL (5/31/2024)

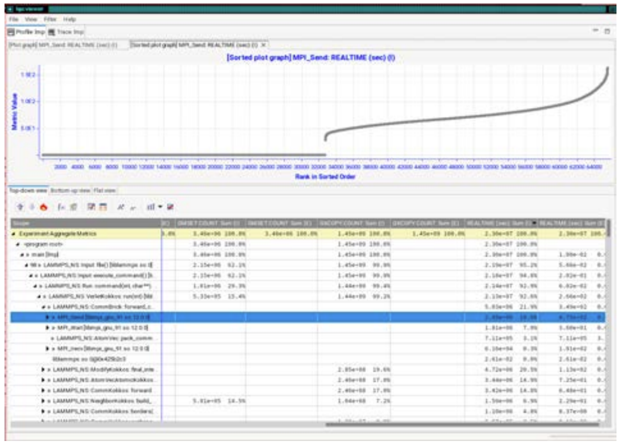
One big thing for us is that we can't overstate how complicated ExaWind is in general, and how complicated it is to build, so finding out that HPCToolkit could easily profile our entire application without a ton of instrumentation during the build process, and be able to profile it on a huge amount of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

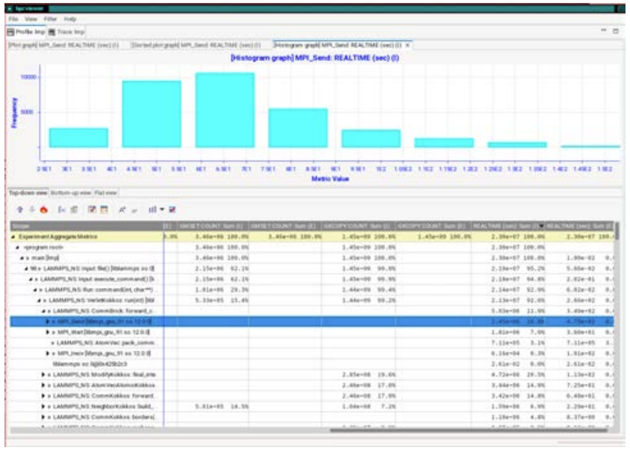
- Jon Rood NREL (6/3/2024)

LAMMPS on Frontier: Executions with Kernel Duration of Milliseconds

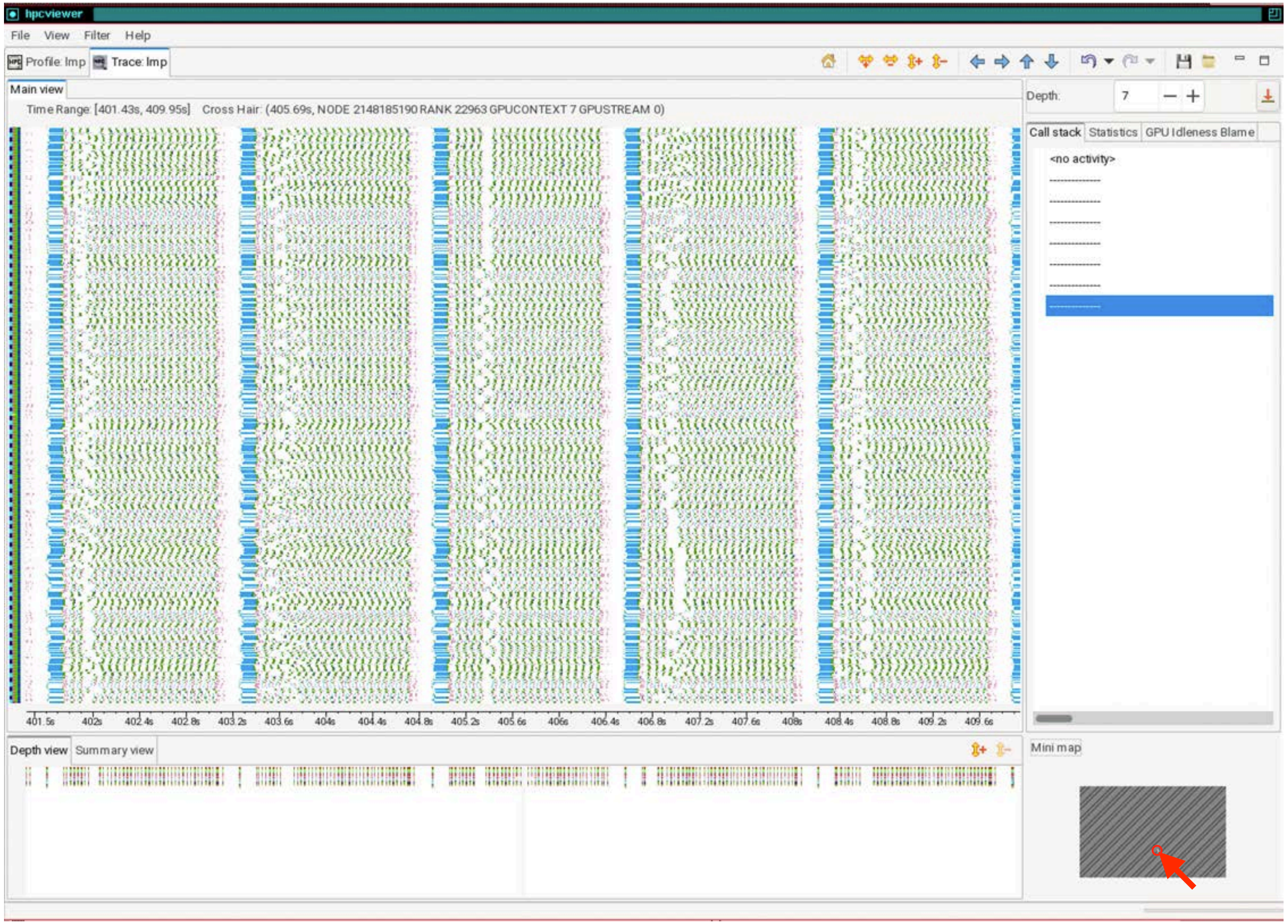






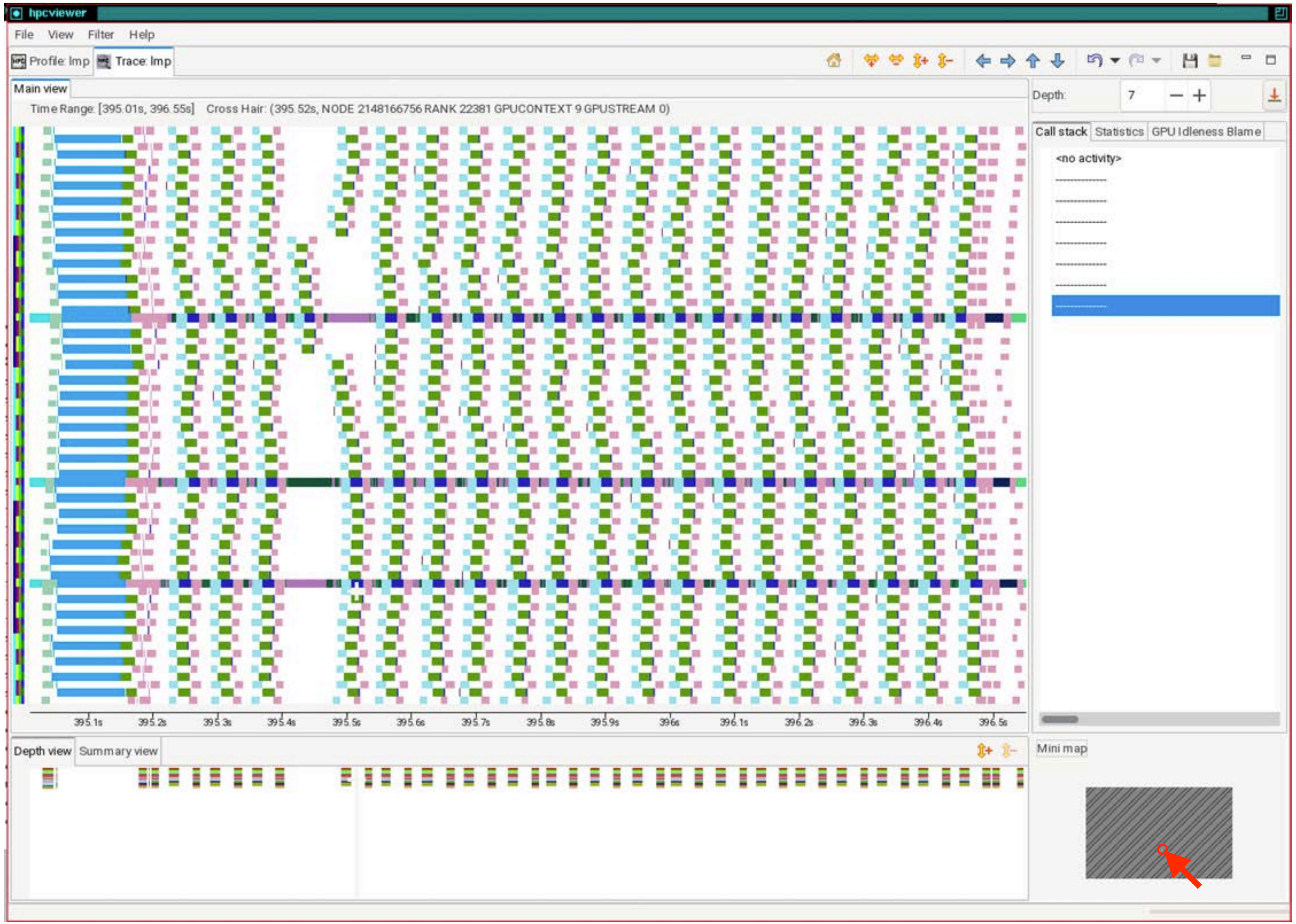


LAMMPS on Frontier: Executions with Kernel Duration of Milliseconds



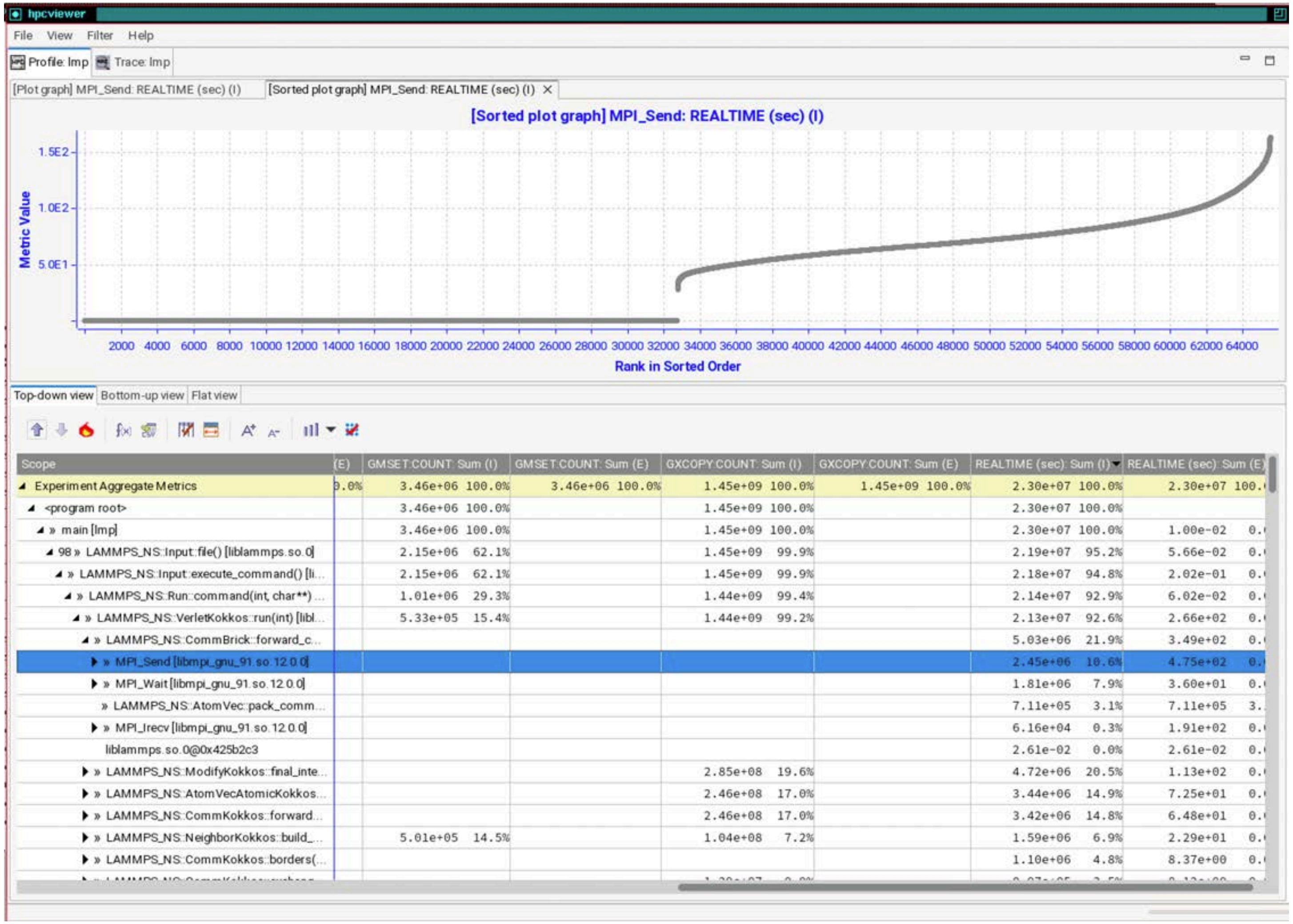
Time Range: [401.43s, 409.95s] Cross Hair: (405.69s, NODE 2148185190 RANK 22963 GPUCONTEXT 7 GPUSTREAM 0)

LAMMPS on Frontier: Executions with Kernel Duration of Milliseconds



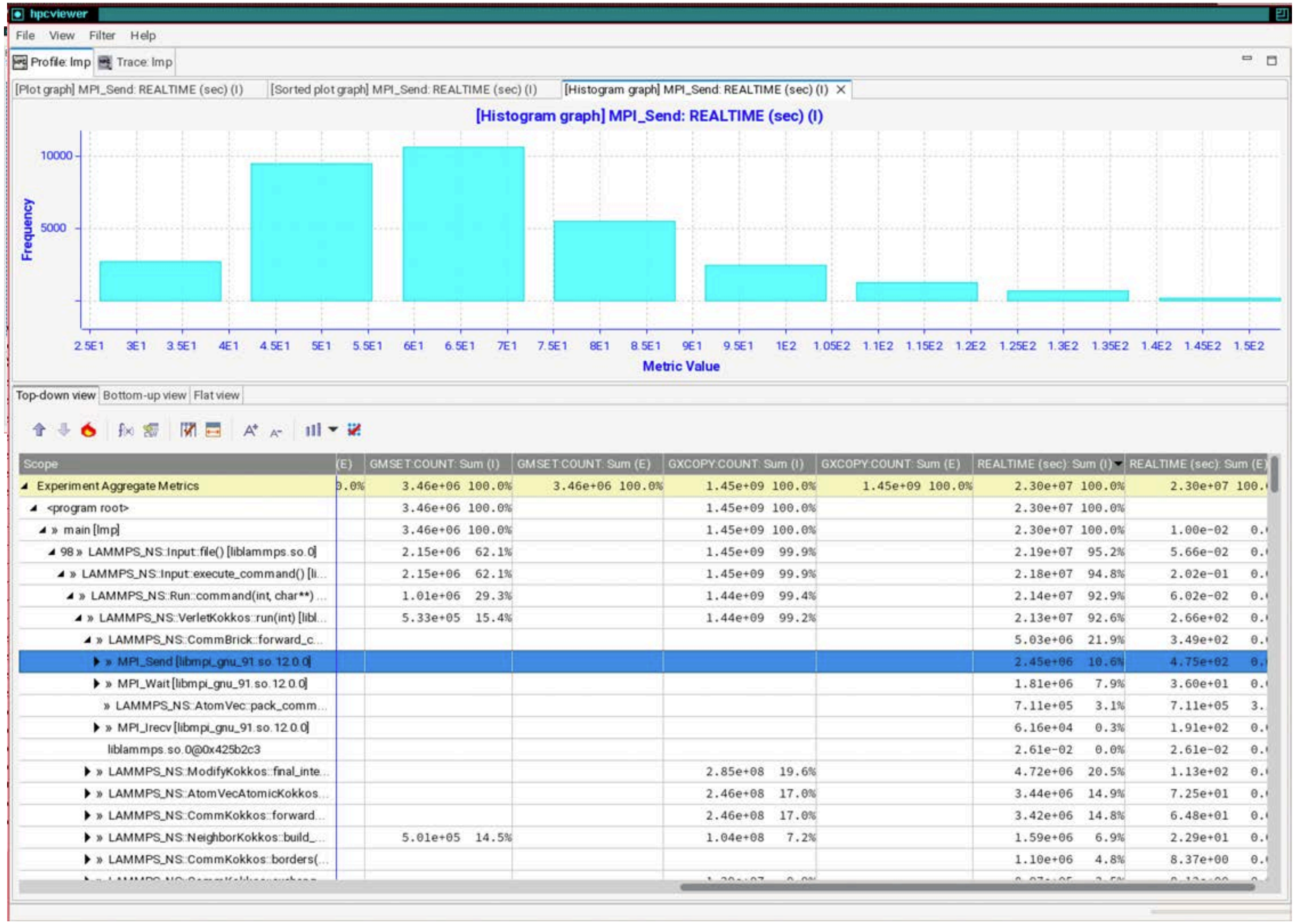
Time Range: [395.01s, 396.55s] Cross Hair: (395.52s, NODE 2148166756 RANK 22381 GPUCONTEXT 9 GPUSTREAM 0)

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.
<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.
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.
» 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.
» 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.
» 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.
» LAMMPS_NS::CommBrick::forward_c...						5.03e+06 21.9%	3.49e+02 0.
» MPI_Send [libmpi_gnu_91.so.12.0.0]						2.45e+06 10.6%	4.75e+02 0.
» MPI_Wait [libmpi_gnu_91.so.12.0.0]						1.81e+06 7.9%	3.60e+01 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.
liblammps.so.0@0x425b2c3						2.61e-02 0.0%	2.61e-02 0.
» LAMMPS_NS::ModifyKokkos::final_inte...				2.85e+08 19.6%		4.72e+06 20.5%	1.13e+02 0.
» LAMMPS_NS::AtomVecAtomicKokkos...				2.46e+08 17.0%		3.44e+06 14.9%	7.25e+01 0.
» LAMMPS_NS::CommKokkos::forward...				2.46e+08 17.0%		3.42e+06 14.8%	6.48e+01 0.
» LAMMPS_NS::NeighborKokkos::build_...		5.01e+05 14.5%		1.04e+08 7.2%		1.59e+06 6.9%	2.29e+01 0.
» LAMMPS_NS::CommKokkos::borders(...						1.10e+06 4.8%	8.37e+00 0.

LAMMPS on Frontier: Executions with Kernel Duration of Milliseconds

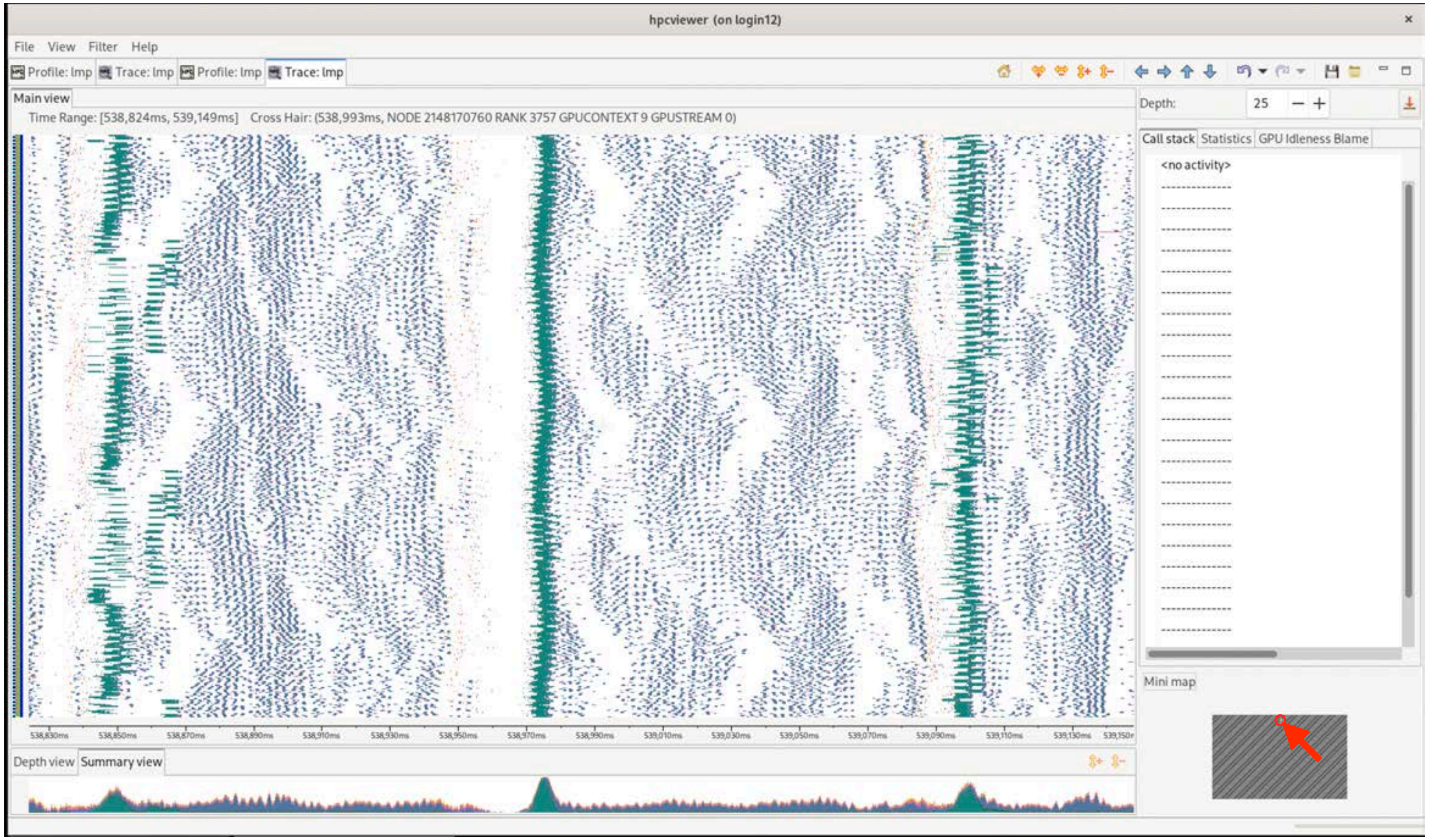


hpcviewer
File View Filter Help
Profile: lmp
Trace: lmp
[Plot graph] MPI_Send: REALTIME (sec) (I)
[Sorted plot graph] MPI_Send: REALTIME (sec) (I)
[Histogram graph] MPI_Send: REALTIME (sec) (I)
[Histogram graph] MPI_Send: REALTIME (sec) (I)
Frequency
10000
5000
2.5E1 3E1 3.5E1 4E1 4.5E1 5E1 5.5E1 6E1 6.5E1 7E1 7.5E1 8E1 8.5E1 9E1 9.5E1 1E2 1.05E2 1.1E2 1.15E2 1.2E2 1.25E2 1.3E2 1.35E2 1.4E2 1.45E2 1.5E2
Metric Value
Top-down view
Bottom-up view
Flat view

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.
<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.
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.
» 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.
» 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.
» 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.
» LAMMPS_NS::CommBrick::forward_c...						5.03e+06 21.9%	3.49e+02 0.
» MPI_Send [libmpi_gnu_91.so.12.0.0]						2.45e+06 10.6%	4.75e+02 0.
» MPI_Wait [libmpi_gnu_91.so.12.0.0]						1.81e+06 7.9%	3.60e+01 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.
liblammps.so.0@0x425b2c3						2.61e-02 0.0%	2.61e-02 0.
» LAMMPS_NS::ModifyKokkos::final_inte...				2.85e+08 19.6%		4.72e+06 20.5%	1.13e+02 0.
» LAMMPS_NS::AtomVecAtomicKokkos...				2.46e+08 17.0%		3.44e+06 14.9%	7.25e+01 0.
» LAMMPS_NS::CommKokkos::forward...				2.46e+08 17.0%		3.42e+06 14.8%	6.48e+01 0.
» LAMMPS_NS::NeighborKokkos::build_...		5.01e+05 14.5%		1.04e+08 7.2%		1.59e+06 6.9%	2.29e+01 0.
» LAMMPS_NS::CommKokkos::borders(...						1.10e+06 4.8%	8.37e+00 0.

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

Kernel duration of microseconds



Time Range: [538,824ms, 539,149ms]   Cross Hair: (538,993ms, NODE 2148170760 RANK 3757 GPUCONTEXT 9 GPUSTREAM 0)

Case Study: GAMESS

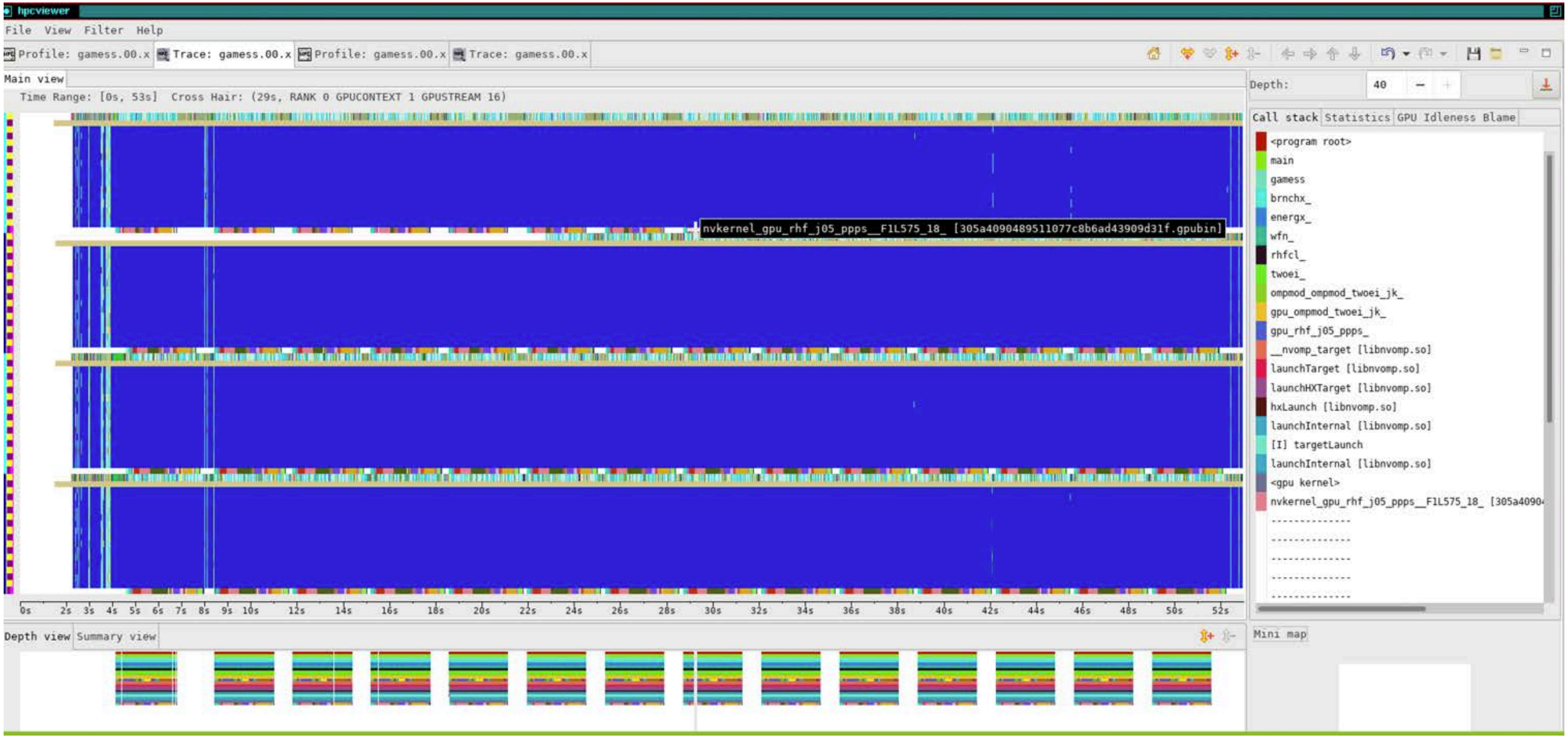
- General Atomic and Molecular Electronic Structure System (GAMESS)
  —general *ab initio* quantum chemistry package
- Calculates the energies, structures, and properties of a wide range of chemical systems

- 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



Time Range: [0s, 53s] Cross Hair: (29s, RANK 0 GPUCONTEXT 1 GPUSTREAM 16)

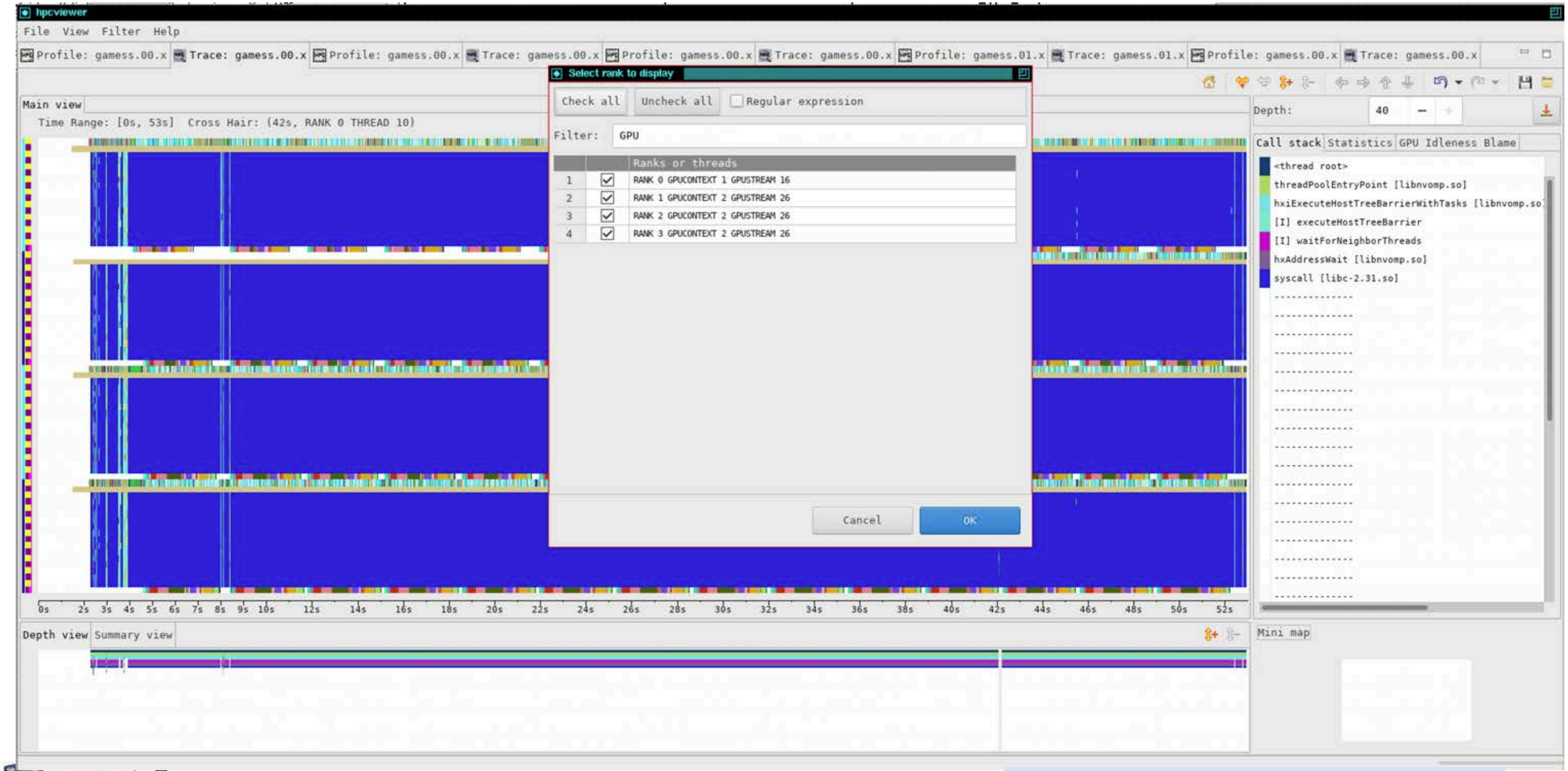
nvkernel_gpu_rhf_j05_ppps__F1L575_18_ [305a4090489511077c8b6ad43909d31f.gpubin]

```
<program root>
main
gamess
brnchx_
energx_
wfn_
rhfcl_
twoei_
ompmod_ompmod_twoei_jk_
gpu_ompmod_twoei_jk_
gpu_rhf_j05_ppps_
__nvomp_target [libnvomp.so]
launchTarget [libnvomp.so]
launchHXTarget [libnvomp.so]
hxLaunch [libnvomp.so]
launchInternal [libnvomp.so]
[I] targetLaunch
launchInternal [libnvomp.so]
<gpu kernel>
nvkernel_gpu_rhf_j05_ppps__F1L575_18_ [305a4090
```

GAMESS original

All CPU threads and GPU streams

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



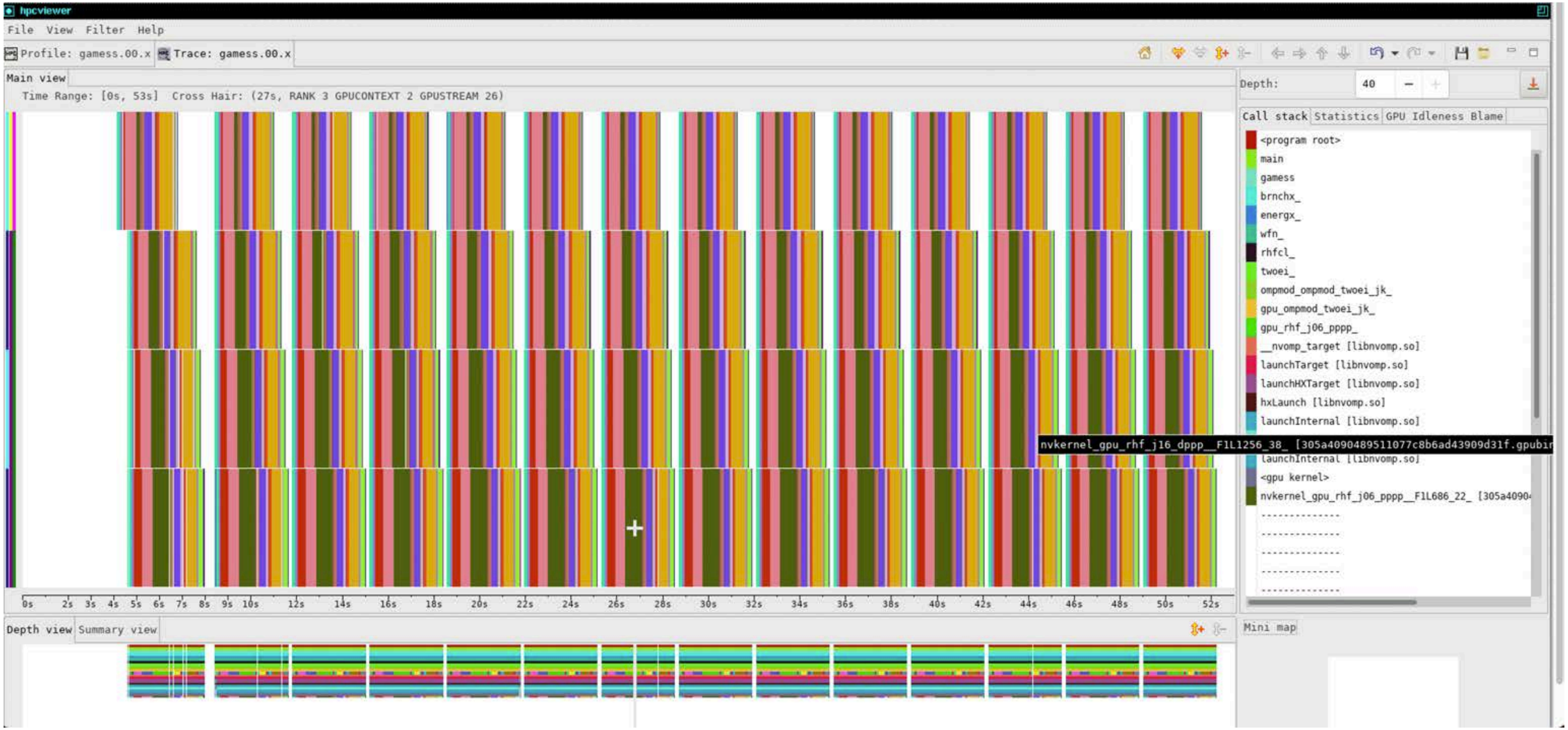
		Ranks or threads
1	✓	RANK 0 GPUCONTEXT 1 GPUSTREAM 16
2	✓	RANK 1 GPUCONTEXT 2 GPUSTREAM 26
3	✓	RANK 2 GPUCONTEXT 2 GPUSTREAM 26
4	✓	RANK 3 GPUCONTEXT 2 GPUSTREAM 26

```
<thread root>
threadPoolEntryPoint [libnvomp.so]
hxiExecuteHostTreeBarrierWithTasks [libnvomp.so]
[I] executeHostTreeBarrier
[I] waitForNeighborThreads
hxAddressWait [libnvomp.so]
syscall [libc-2.31.so]
```

GAMESS original

All CPU threads and GPU streams

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



```
Time Range: [0s, 53s]  Cross Hair: (27s, RANK 3 GPUCONTEXT 2 GPUSTREAM 26)
```

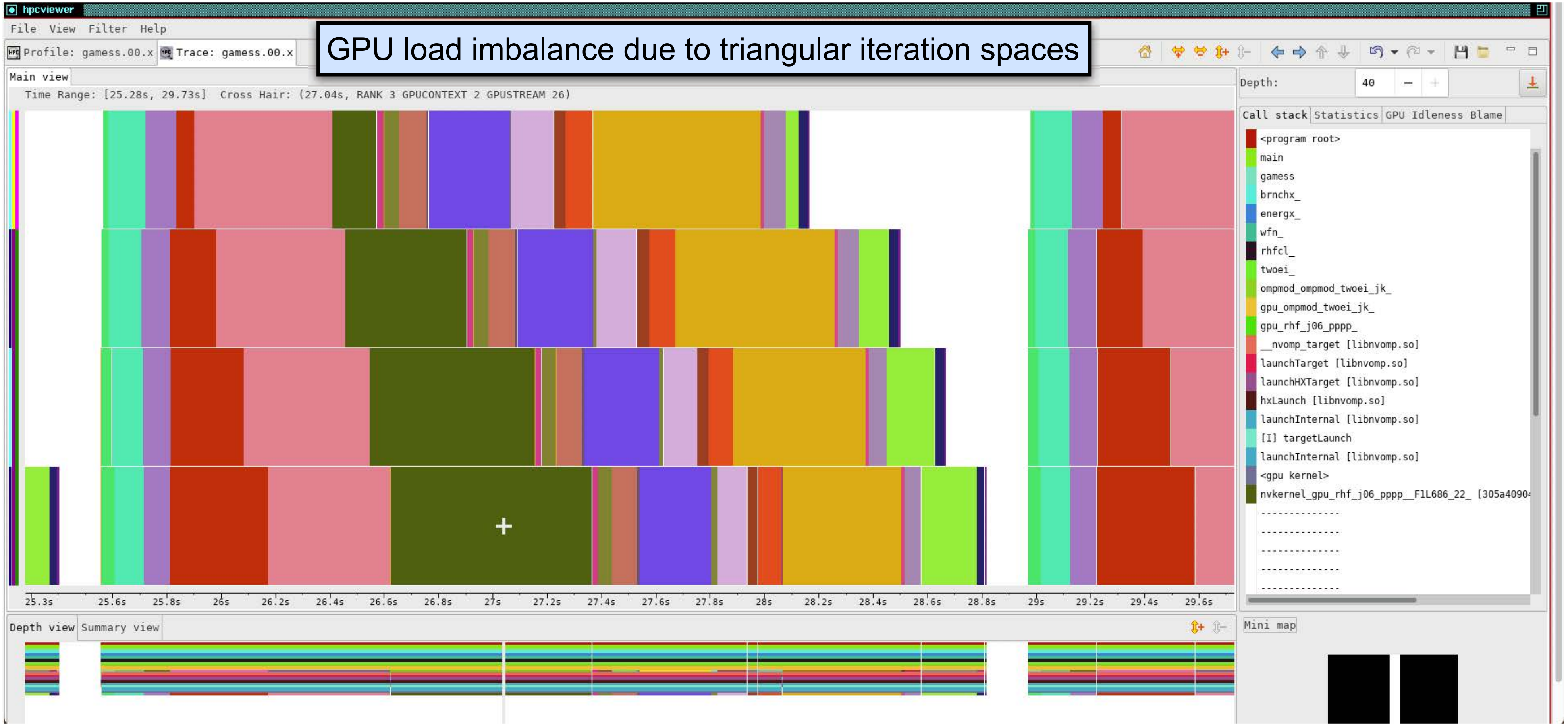
Depth: 40

```
<program root>
main
gamess
brnchx_
energx_
wfn_
rhfcl_
twoei_
ompmod_ompmod_twoei_jk_
gpu_ompmod_twoei_jk_
gpu_rhf_j06_pppp_
__nvomp_target [libnvomp.so]
launchTarget [libnvomp.so]
launchHXTarget [libnvomp.so]
hxLaunch [libnvomp.so]
launchInternal [libnvomp.so]
nvkernel_gpu_rhf_j16_dppp__F1L1256_38_ [305a4090489511077c8b6ad43909d31f.gpubin
launchInternal [libnvomp.so]
<gpu kernel>
nvkernel_gpu_rhf_j06_pppp__F1L686_22_ [305a4090
-------------
-------------
-------------
-------------
-------------
```

GAMESS original

All GPU streams, whole execution

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



GPU load imbalance due to triangular iteration spaces

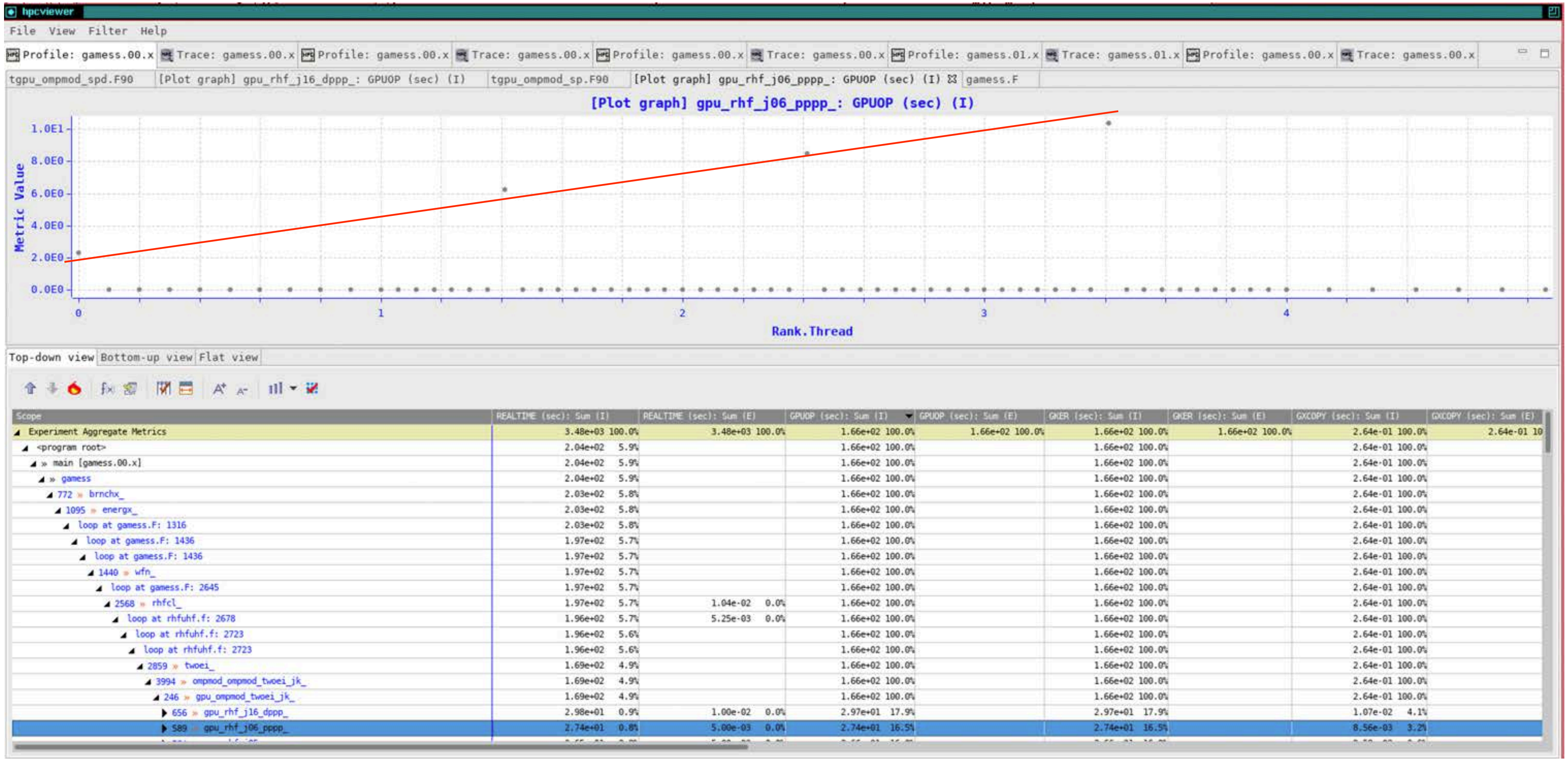
Time Range: [25.28s, 29.73s] Cross Hair: (27.04s, RANK 3 GPUCONTEXT 2 GPUSTREAM 26)

```
<program root>
main
gamess
brnchx_
energx_
wfn_
rhfcl_
twoei_
ompmod_ompmod_twoei_jk_
gpu_ompmod_twoei_jk_
gpu_rhf_j06_pppp_
__nvomp_target [libnvomp.so]
launchTarget [libnvomp.so]
launchHXTarget [libnvomp.so]
hxLaunch [libnvomp.so]
launchInternal [libnvomp.so]
[I] targetLaunch
launchInternal [libnvomp.so]
<gpu kernel>
nvkernel_gpu_rhf_j06_pppp__F1L686_22_ [305a4090
-------------
-------------
-------------
-------------
-------------
```

GAMESS original

GPU streams: 1 iteration

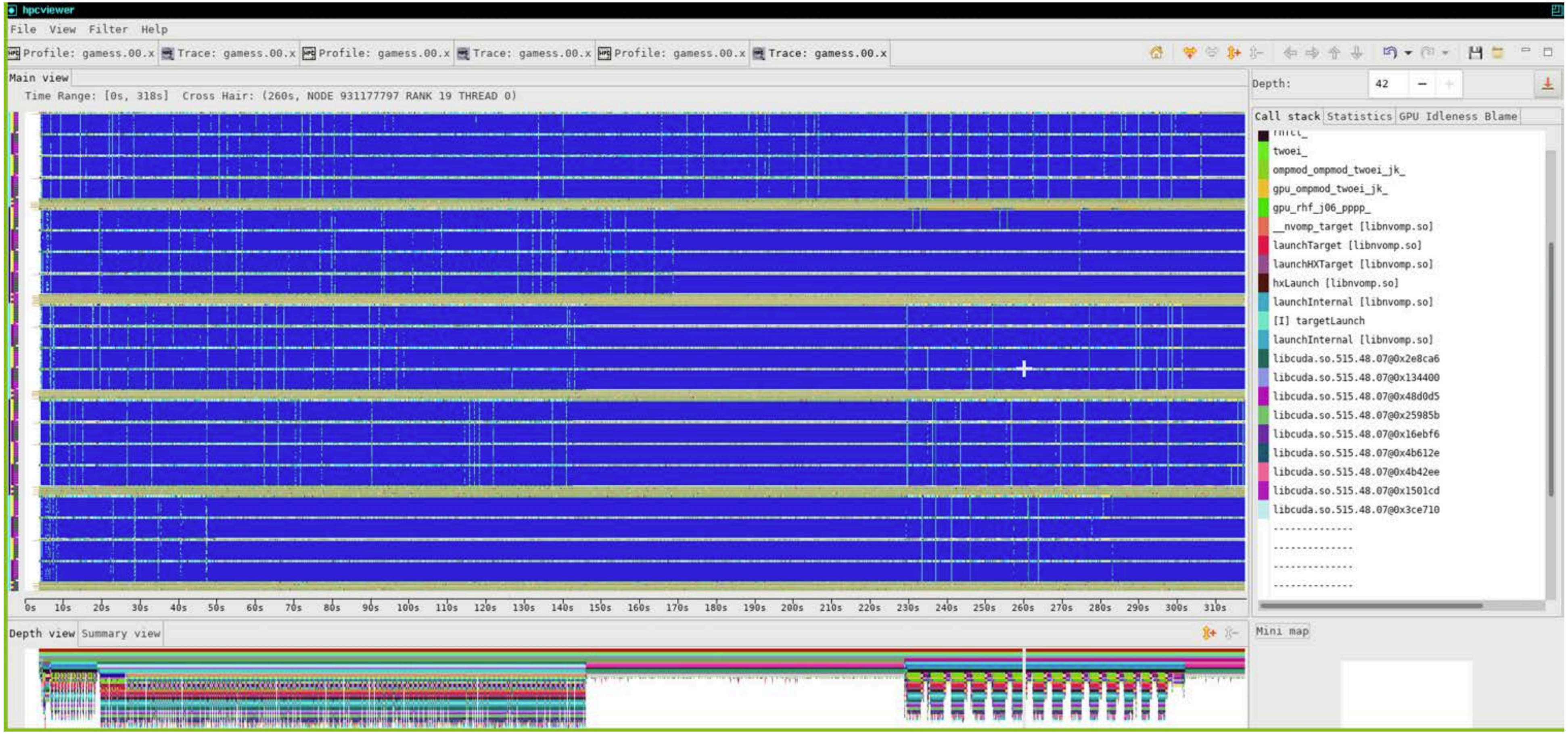
Time-centric Analysis: GAMESS 4 ranks, 4 GPUs on Perlmutter



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)
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



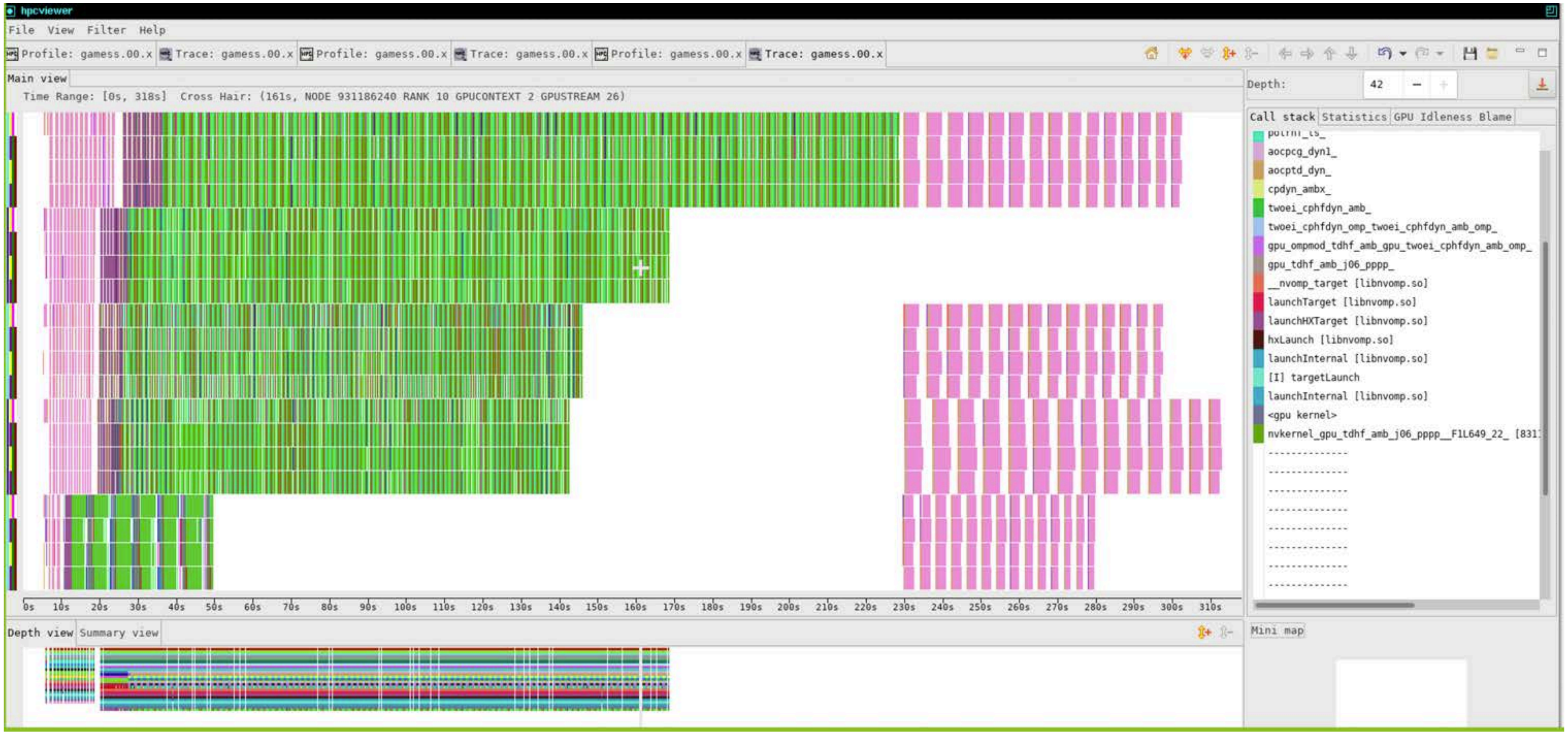
Time Range: [0s, 318s] Cross Hair: (260s, NODE 931177797 RANK 19 THREAD 0)

```
twoei_
ompmod_ompmod_twoei_jk_
gpu_ompmod_twoei_jk_
gpu_rhf_j06_pppp_
__nvomp_target [libnvomp.so]
launchTarget [libnvomp.so]
launchHXTarget [libnvomp.so]
hxLaunch [libnvomp.so]
launchInternal [libnvomp.so]
[I] targetLaunch
launchInternal [libnvomp.so]
libcuda.so.515.48.07@0x2e8ca6
libcuda.so.515.48.07@0x134400
libcuda.so.515.48.07@0x48d0d5
libcuda.so.515.48.07@0x25985b
libcuda.so.515.48.07@0x16ebf6
libcuda.so.515.48.07@0x4b612e
libcuda.so.515.48.07@0x4b42ee
libcuda.so.515.48.07@0x1501cd
libcuda.so.515.48.07@0x3ce710
-------------
-------------
-------------
-------------
```

GAMESS improved

CPU Threads and GPU Streams

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

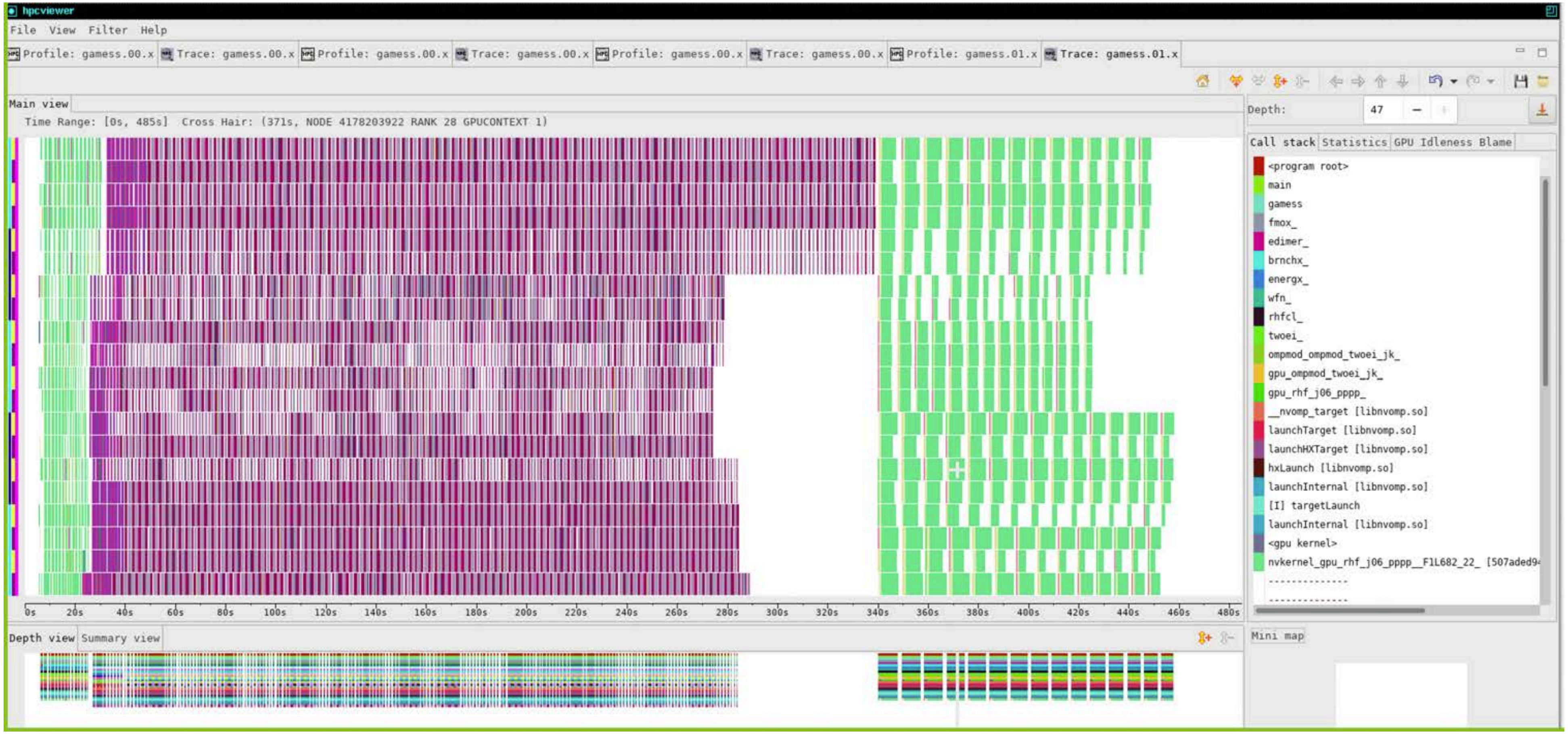


```
Time Range: [0s, 318s]  Cross Hair: (161s, NODE 931186240 RANK 10 GPUCONTEXT 2 GPUSTREAM 26)
```

```
aocpcg_dyn1_
aocptd_dyn_
cpdyn_ambx_
twoei_cphfdyn_amb_
twoei_cphfdyn_omp_twoei_cphfdyn_amb_omp_
gpu_ompmod_tdhf_amb_gpu_twoei_cphfdyn_amb_omp_
gpu_tdhf_amb_j06_pppp_
__nvomp_target [libnvomp.so]
launchTarget [libnvomp.so]
launchHXTarget [libnvomp.so]
hxLaunch [libnvomp.so]
launchInternal [libnvomp.so]
[I] targetLaunch
launchInternal [libnvomp.so]
<gpu kernel>
nvkernel_gpu_tdhf_amb_j06_pppp__F1L649_22_ [831
-------------
-------------
-------------
-------------
-------------
-------------
-------------
-------------
```

GAMESS improved

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

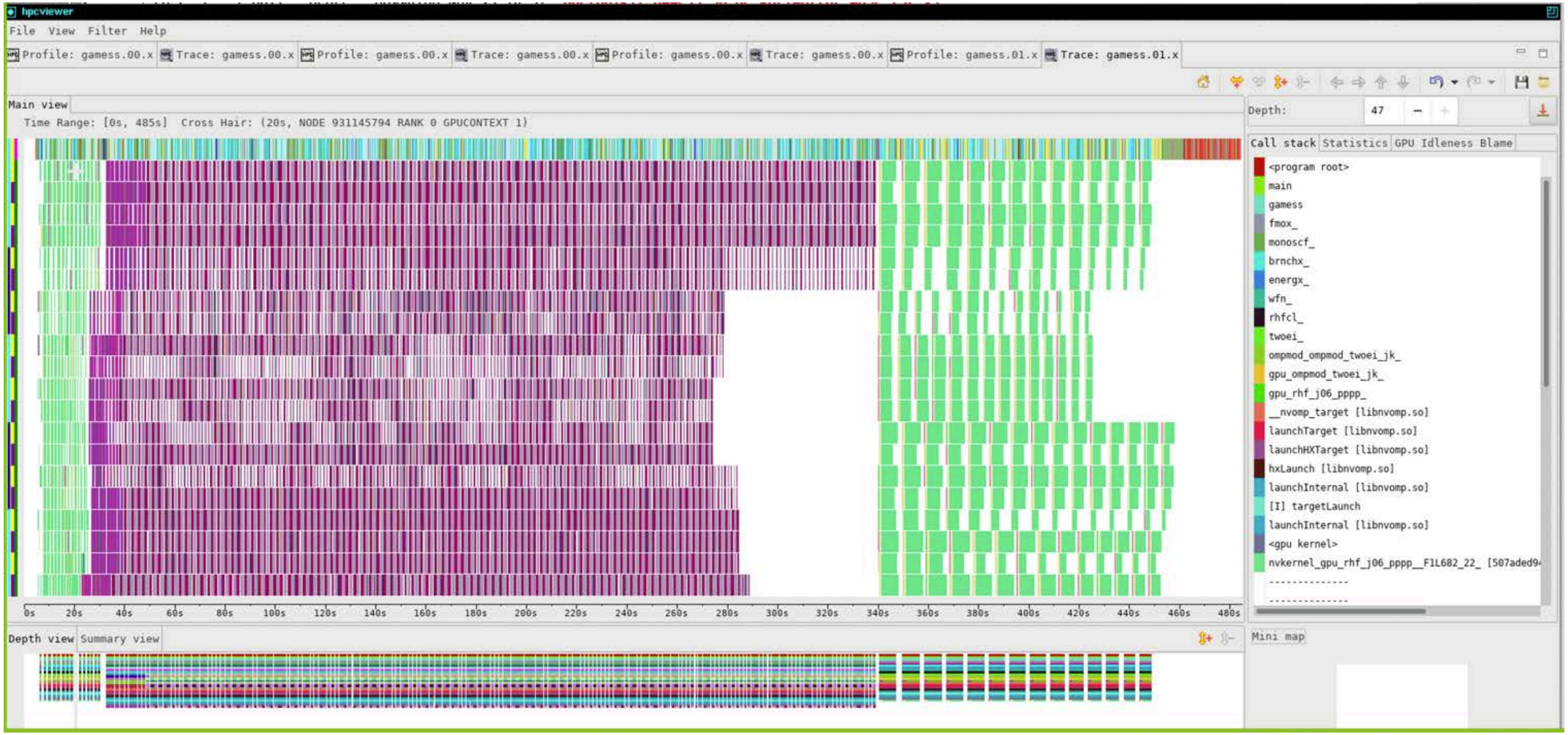


Time Range: [0s, 485s] Cross Hair: (371s, NODE 4178203922 RANK 28 GPUCONTEXT 1)

```
<program root>
main
gamess
fmox_
edimer_
brnchx_
energx_
wfn_
rhfcl_
twoei_
ompmod_ompmod_twoei_jk_
gpu_ompmod_twoei_jk_
gpu_rhf_j06_pppp_
__nvomp_target [libnvomp.so]
launchTarget [libnvomp.so]
launchHXTarget [libnvomp.so]
hxLaunch [libnvomp.so]
launchInternal [libnvomp.so]
[I] targetLaunch
launchInternal [libnvomp.so]
<gpu kernel>
nvkernel_gpu_rhf_j06_pppp__F1L682_22_ [507aded9
-------------
-------------
```

GAMESS improved with better manual distribution of work in input

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

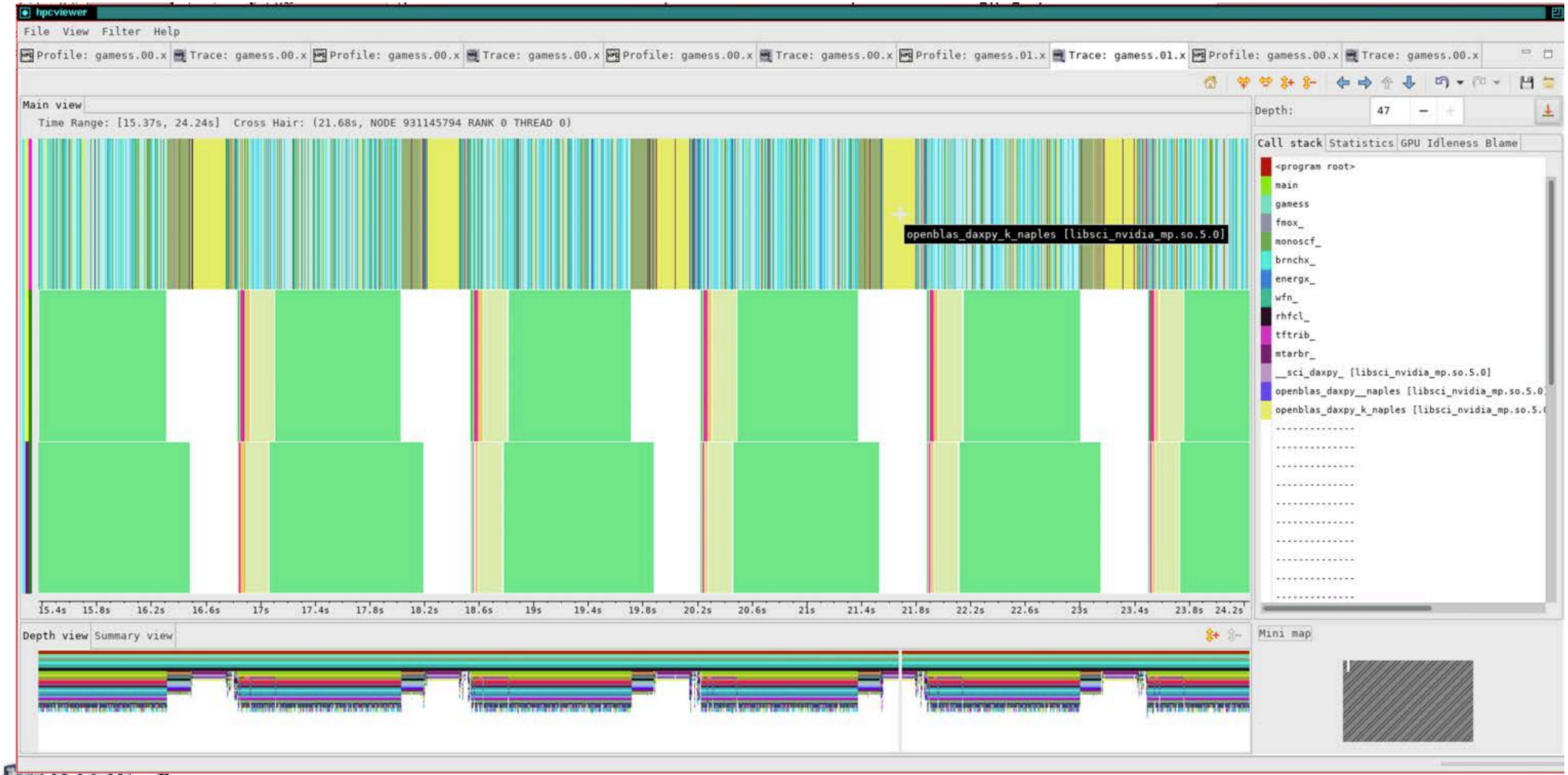


```
Time Range: [0s, 485s]  Cross Hair: (20s, NODE 931145794 RANK 0 GPUCONTEXT 1)
Depth: 47
```

```
<program root>
main
gamess
fmox_
monoscf_
brnchx_
energx_
wfn_
rhfcl_
twoei_
ompmod_ompmod_twoei_jk_
gpu_ompmod_twoei_jk_
gpu_rhf_j06_pppp_
__nvomp_target [libnvomp.so]
launchTarget [libnvomp.so]
launchHXTarget [libnvomp.so]
hxLaunch [libnvomp.so]
launchInternal [libnvomp.so]
[I] targetLaunch
launchInternal [libnvomp.so]
<gpu kernel>
nvkernel_gpu_rhf_j06_pppp__F1L682_22_ [507aded9
--------------
--------------
```

GAMESS improved adding Rank 0 Thread 0 to GPU streams

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



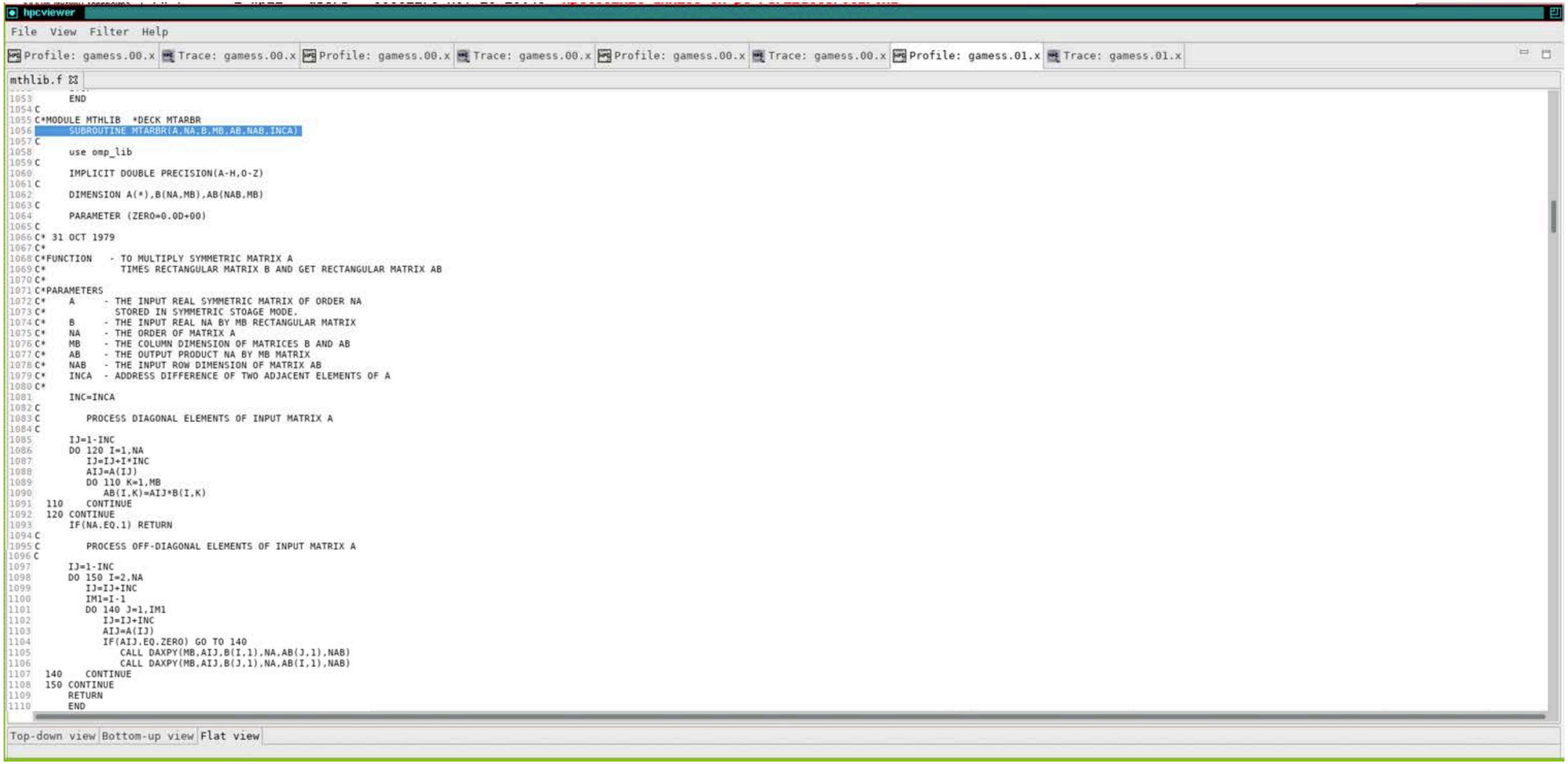
Time Range: [15.37s, 24.24s]   Cross Hair: (21.68s, NODE 931145794 RANK 0 THREAD 0)

openblas_daxpy_k_naples [libsci_nvidia_mp.so.5.0]

- <program root>
- main
- gamess
- fmox_
- monoscf_
- brnchx_
- energx_
- wfn_
- rhfcl_
- tftrib_
- mtarbr_
- __sci_daxpy_ [libsci_nvidia_mp.so.5.0]
- openblas_daxpy__naples [libsci_nvidia_mp.so.5.0]
- openblas_daxpy_k_naples [libsci_nvidia_mp.so.5.(

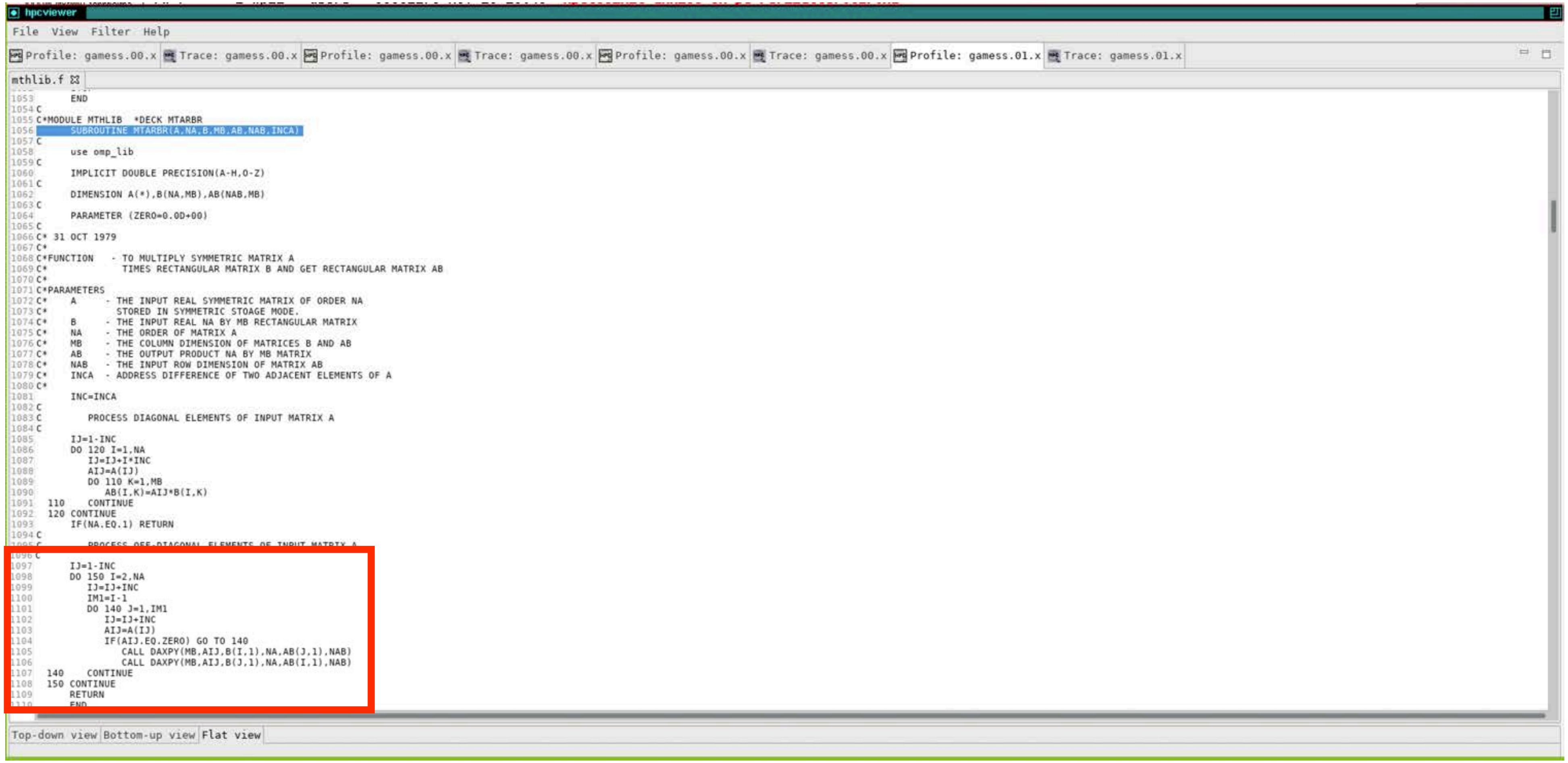
1 CPU Stream, 2 GPU Streams: 6 Iterations

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



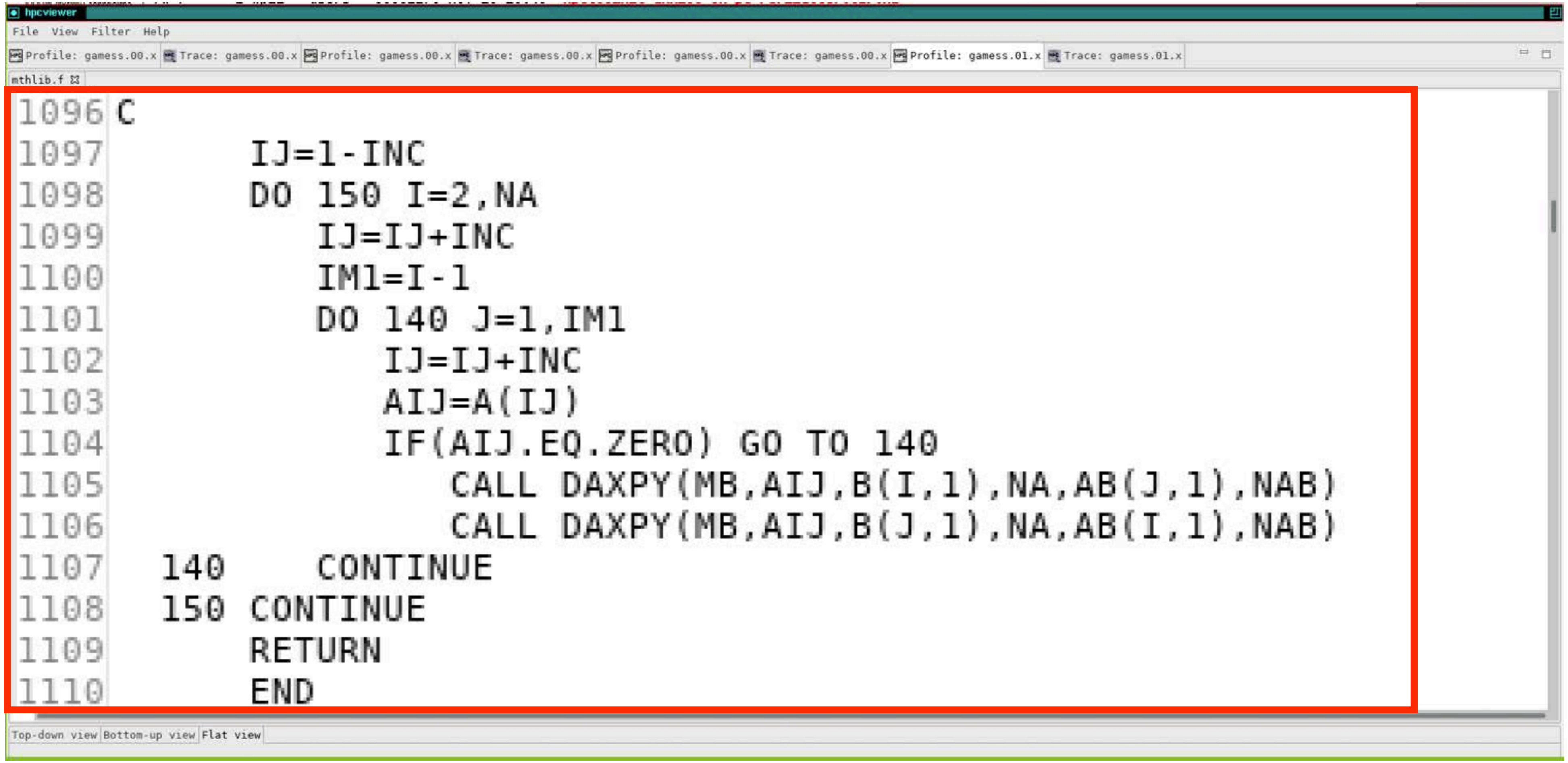
```
      END
C
C*MODULE MTHLIB  *DECK MTARBR
      SUBROUTINE MTARBR(A,NA,B,MB,AB,NAB,INCA)
C
      use omp_lib
C
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
C
      DIMENSION A(*),B(NA,MB),AB(NAB,MB)
C
      PARAMETER (ZERO=0.0D+00)
C
C* 31 OCT 1979
C*
C*FUNCTION   - TO MULTIPLY SYMMETRIC MATRIX A
C*              TIMES RECTANGULAR MATRIX B AND GET RECTANGULAR MATRIX AB
C*
C*PARAMETERS
C*     A      - THE INPUT REAL SYMMETRIC MATRIX OF ORDER NA
C*              STORED IN SYMMETRIC STOAGE MODE.
C*     B      - THE INPUT REAL NA BY MB RECTANGULAR MATRIX
C*     NA     - THE ORDER OF MATRIX A
C*     MB     - THE COLUMN DIMENSION OF MATRICES B AND AB
C*     AB     - THE OUTPUT PRODUCT NA BY MB MATRIX
C*     NAB    - THE INPUT ROW DIMENSION OF MATRIX AB
C*     INCA   - ADDRESS DIFFERENCE OF TWO ADJACENT ELEMENTS OF A
C*
      INC=INCA
C
C        PROCESS DIAGONAL ELEMENTS OF INPUT MATRIX A
C
      IJ=1-INC
      DO 120 I=1,NA
         IJ=IJ+I*INC
         AIJ=A(IJ)
         DO 110 K=1,MB
            AB(I,K)=AIJ*B(I,K)
  110    CONTINUE
  120 CONTINUE
      IF(NA.EQ.1) RETURN
C
C        PROCESS OFF-DIAGONAL ELEMENTS OF INPUT MATRIX A
C
      IJ=1-INC
      DO 150 I=2,NA
         IJ=IJ+INC
         IM1=I-1
         DO 140 J=1,IM1
            IJ=IJ+INC
            AIJ=A(IJ)
            IF(AIJ.EQ.ZERO) GO TO 140
               CALL DAXPY(MB,AIJ,B(I,1),NA,AB(J,1),NAB)
               CALL DAXPY(MB,AIJ,B(J,1),NA,AB(I,1),NAB)
  140    CONTINUE
  150 CONTINUE
      RETURN
      END
```

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



```
      END
C
C*MODULE MTHLIB  *DECK MTARBR
      SUBROUTINE MTARBR(A,NA,B,MB,AB,NAB,INCA)
C
      use omp_lib
C
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
C
      DIMENSION A(*),B(NA,MB),AB(NAB,MB)
C
      PARAMETER (ZERO=0.0D+00)
C
C* 31 OCT 1979
C*
C*FUNCTION   - TO MULTIPLY SYMMETRIC MATRIX A
C*              TIMES RECTANGULAR MATRIX B AND GET RECTANGULAR MATRIX AB
C*
C*PARAMETERS
C*    A      - THE INPUT REAL SYMMETRIC MATRIX OF ORDER NA
C*             STORED IN SYMMETRIC STOAGE MODE.
C*    B      - THE INPUT REAL NA BY MB RECTANGULAR MATRIX
C*    NA     - THE ORDER OF MATRIX A
C*    MB     - THE COLUMN DIMENSION OF MATRICES B AND AB
C*    AB     - THE OUTPUT PRODUCT NA BY MB MATRIX
C*    NAB    - THE INPUT ROW DIMENSION OF MATRIX AB
C*    INCA   - ADDRESS DIFFERENCE OF TWO ADJACENT ELEMENTS OF A
C*
      INC=INCA
C
C        PROCESS DIAGONAL ELEMENTS OF INPUT MATRIX A
C
      IJ=1-INC
      DO 120 I=1,NA
         IJ=IJ+I*INC
         AIJ=A(IJ)
         DO 110 K=1,MB
            AB(I,K)=AIJ*B(I,K)
  110    CONTINUE
  120 CONTINUE
      IF(NA.EQ.1) RETURN
C
C        PROCESS OFF-DIAGONAL ELEMENTS OF INPUT MATRIX A
C
      IJ=1-INC
      DO 150 I=2,NA
         IJ=IJ+INC
         IM1=I-1
         DO 140 J=1,IM1
            IJ=IJ+INC
            AIJ=A(IJ)
            IF(AIJ.EQ.ZERO) GO TO 140
               CALL DAXPY(MB,AIJ,B(I,1),NA,AB(J,1),NAB)
               CALL DAXPY(MB,AIJ,B(J,1),NA,AB(I,1),NAB)
  140    CONTINUE
  150 CONTINUE
      RETURN
      END
```

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

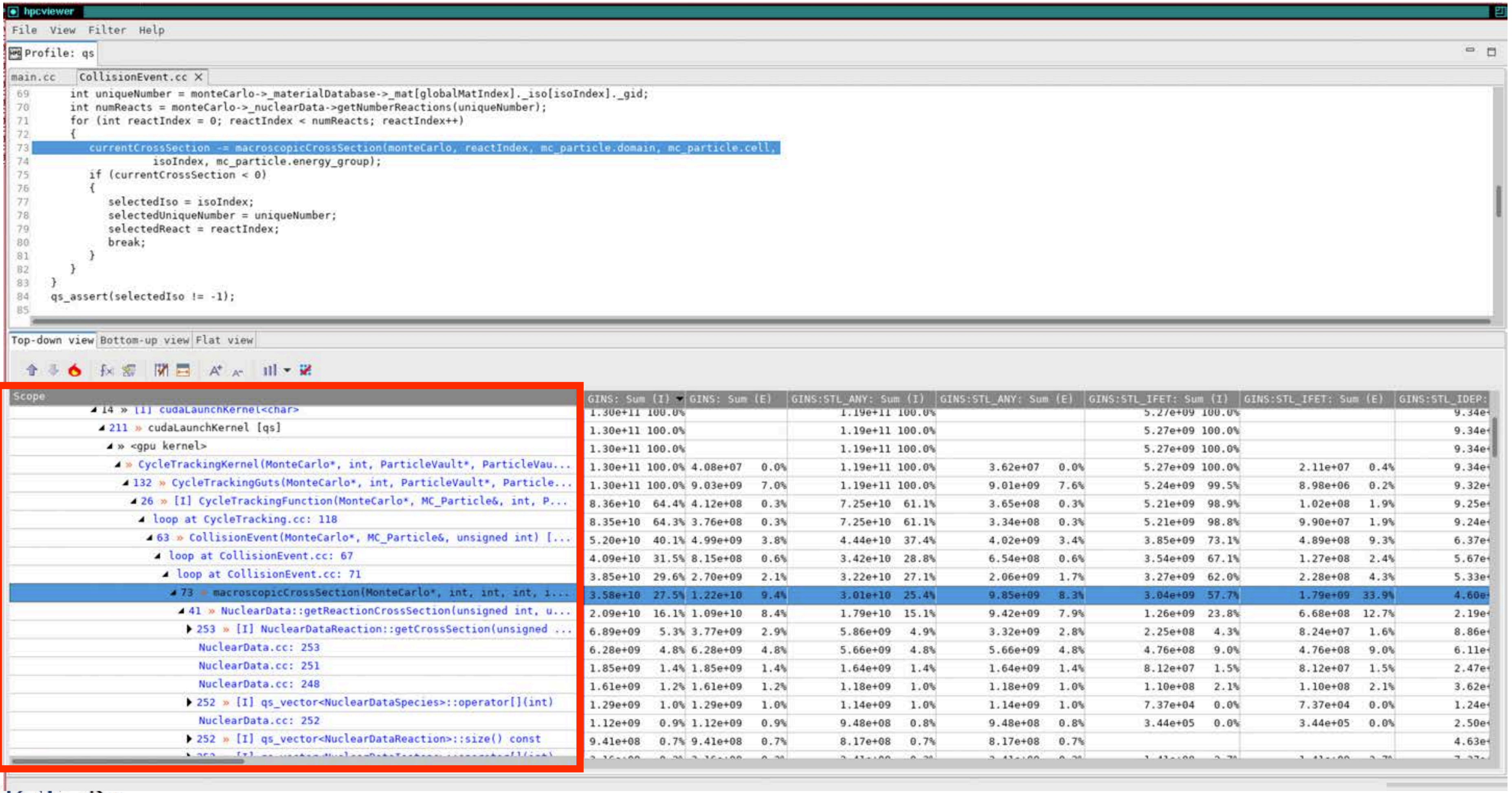


```
C
      IJ=1-INC
      DO 150 I=2,NA
         IJ=IJ+INC
         IM1=I-1
         DO 140 J=1,IM1
            IJ=IJ+INC
            AIJ=A(IJ)
            IF(AIJ.EQ.ZERO) GO TO 140
               CALL DAXPY(MB,AIJ,B(I,1),NA,AB(J,1),NAB)
               CALL DAXPY(MB,AIJ,B(J,1),NA,AB(I,1),NAB)
  140    CONTINUE
  150 CONTINUE
      RETURN
      END
```

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

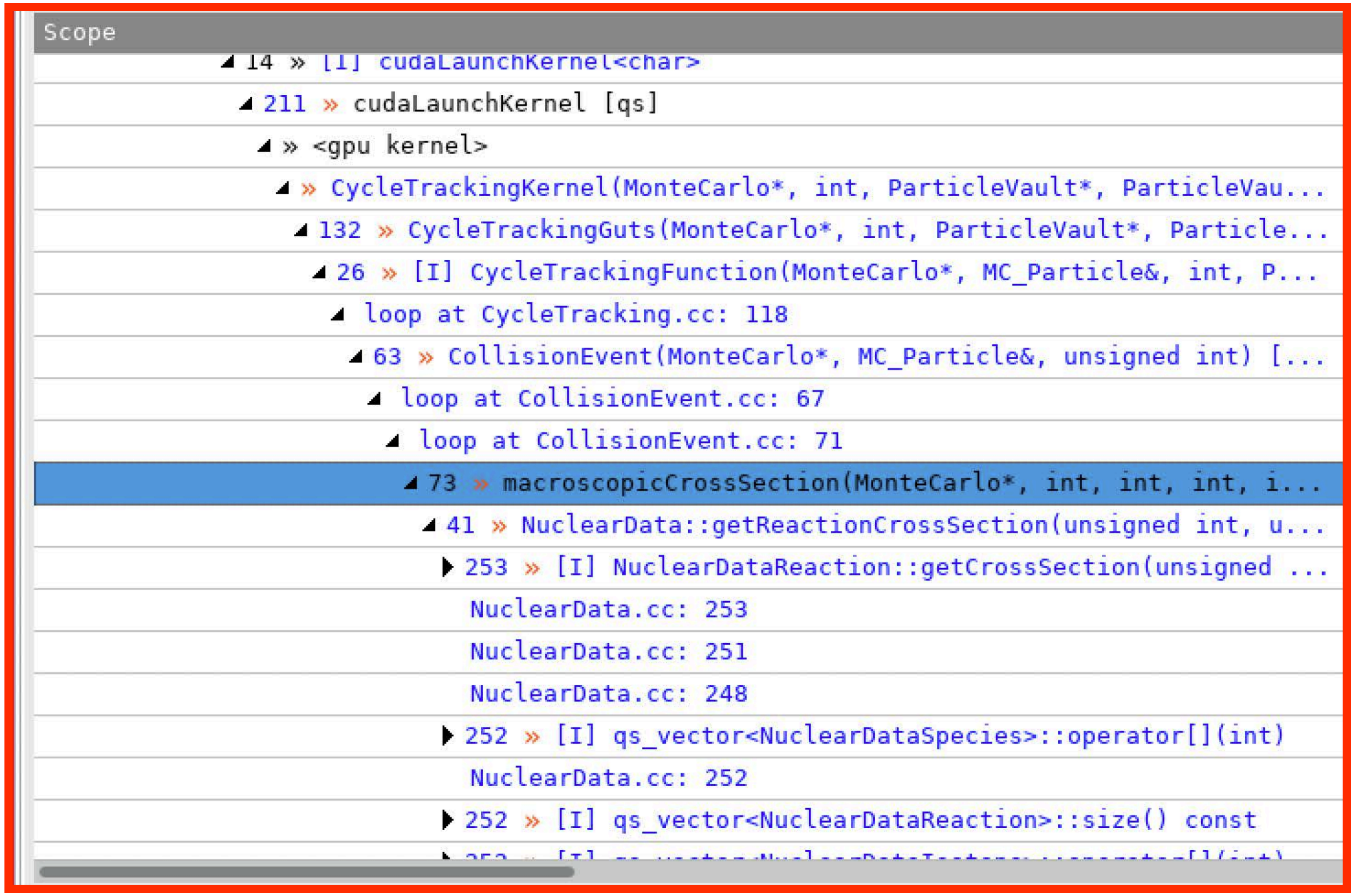
Quicksilver: Detailed analysis within a Kernel using PC Sampling



```
      int uniqueNumber = monteCarlo->_materialDatabase->_mat[globalMatIndex]._iso[isoIndex]._gid;
      int numReacts = monteCarlo->_nuclearData->getNumberReactions(uniqueNumber);
      for (int reactIndex = 0; reactIndex < numReacts; reactIndex++)
      {
         currentCrossSection -= macroscopicCrossSection(monteCarlo, reactIndex, mc_particle.domain, mc_particle.cell,
                    isoIndex, mc_particle.energy_group);
         if (currentCrossSection < 0)
         {
            selectedIso = isoIndex;
            selectedUniqueNumber = uniqueNumber;
            selectedReact = reactIndex;
            break;
         }
      }
   }
   qs_assert(selectedIso != -1);
```

Scope	GINS: Sum (I)	GINS: Sum (E)	GINS:STL_ANY: Sum (I)	GINS:STL_ANY: Sum (E)	GINS:STL_IFET: Sum (I)	GINS:STL_IFET: Sum (E)	GINS:STL_IDEP:
14 » [I] cudaLaunchKernel<char>	1.30e+11 100.0%		1.19e+11 100.0%		5.27e+09 100.0%		9.34e
211 » cudaLaunchKernel [qs]	1.30e+11 100.0%		1.19e+11 100.0%		5.27e+09 100.0%		9.34e
» <gpu kernel>	1.30e+11 100.0%		1.19e+11 100.0%		5.27e+09 100.0%		9.34e
» CycleTrackingKernel(MonteCarlo*, int, ParticleVault*, ParticleVau...	1.30e+11 100.0%	4.08e+07 0.0%	1.19e+11 100.0%	3.62e+07 0.0%	5.27e+09 100.0%	2.11e+07 0.4%	9.34e
132 » CycleTrackingGuts(MonteCarlo*, int, ParticleVault*, Particle...	1.30e+11 100.0%	9.03e+09 7.0%	1.19e+11 100.0%	9.01e+09 7.6%	5.24e+09 99.5%	8.98e+06 0.2%	9.32e
26 » [I] CycleTrackingFunction(MonteCarlo*, MC_Particle&, int, P...	8.36e+10 64.4%	4.12e+08 0.3%	7.25e+10 61.1%	3.65e+08 0.3%	5.21e+09 98.9%	1.02e+08 1.9%	9.25e
loop at CycleTracking.cc: 118	8.35e+10 64.3%	3.76e+08 0.3%	7.25e+10 61.1%	3.34e+08 0.3%	5.21e+09 98.8%	9.90e+07 1.9%	9.24e
63 » CollisionEvent(MonteCarlo*, MC_Particle&, unsigned int) [...	5.20e+10 40.1%	4.99e+09 3.8%	4.44e+10 37.4%	4.02e+09 3.4%	3.85e+09 73.1%	4.89e+08 9.3%	6.37e
loop at CollisionEvent.cc: 67	4.09e+10 31.5%	8.15e+08 0.6%	3.42e+10 28.8%	6.54e+08 0.6%	3.54e+09 67.1%	1.27e+08 2.4%	5.67e
loop at CollisionEvent.cc: 71	3.85e+10 29.6%	2.70e+09 2.1%	3.22e+10 27.1%	2.06e+09 1.7%	3.27e+09 62.0%	2.28e+08 4.3%	5.33e
73 » macroscopicCrossSection(MonteCarlo*, int, int, int, i...	3.58e+10 27.5%	1.22e+10 9.4%	3.01e+10 25.4%	9.85e+09 8.3%	3.04e+09 57.7%	1.79e+09 33.9%	4.60e
41 » NuclearData::getReactionCrossSection(unsigned int, u...	2.09e+10 16.1%	1.09e+10 8.4%	1.79e+10 15.1%	9.42e+09 7.9%	1.26e+09 23.8%	6.68e+08 12.7%	2.19e
253 » [I] NuclearDataReaction::getCrossSection(unsigned ...	6.89e+09 5.3%	3.77e+09 2.9%	5.86e+09 4.9%	3.32e+09 2.8%	2.25e+08 4.3%	8.24e+07 1.6%	8.86e
NuclearData.cc: 253	6.28e+09 4.8%	6.28e+09 4.8%	5.66e+09 4.8%	5.66e+09 4.8%	4.76e+08 9.0%	4.76e+08 9.0%	6.11e
NuclearData.cc: 251	1.85e+09 1.4%	1.85e+09 1.4%	1.64e+09 1.4%	1.64e+09 1.4%	8.12e+07 1.5%	8.12e+07 1.5%	2.47e
NuclearData.cc: 248	1.61e+09 1.2%	1.61e+09 1.2%	1.18e+09 1.0%	1.18e+09 1.0%	1.10e+08 2.1%	1.10e+08 2.1%	3.62e
252 » [I] qs_vector<NuclearDataSpecies>::operator[](int)	1.29e+09 1.0%	1.29e+09 1.0%	1.14e+09 1.0%	1.14e+09 1.0%	7.37e+04 0.0%	7.37e+04 0.0%	1.24e
NuclearData.cc: 252	1.12e+09 0.9%	1.12e+09 0.9%	9.48e+08 0.8%	9.48e+08 0.8%	3.44e+05 0.0%	3.44e+05 0.0%	2.50e
252 » [I] qs_vector<NuclearDataReaction>::size() const	9.41e+08 0.7%	9.41e+08 0.7%	8.17e+08 0.7%	8.17e+08 0.7%			4.63e

Quicksilver: Detailed analysis within a Kernel using PC Sampling



```
Scope
◢ 14 » [I] cudaLaunchKernel<char>
  ◢ 211 » cudaLaunchKernel [qs]
    ◢ » <gpu kernel>
      ◢ » CycleTrackingKernel(MonteCarlo*, int, ParticleVault*, ParticleVau...
        ◢ 132 » CycleTrackingGuts(MonteCarlo*, int, ParticleVault*, Particle...
          ◢ 26 » [I] CycleTrackingFunction(MonteCarlo*, MC_Particle&, int, P...
            ◢ loop at CycleTracking.cc: 118
              ◢ 63 » CollisionEvent(MonteCarlo*, MC_Particle&, unsigned int) [...
                ◢ loop at CollisionEvent.cc: 67
                  ◢ loop at CollisionEvent.cc: 71
                    ◢ 73 » macroscopicCrossSection(MonteCarlo*, int, int, int, i...
                      ◢ 41 » NuclearData::getReactionCrossSection(unsigned int, u...
                        ▶ 253 » [I] NuclearDataReaction::getCrossSection(unsigned ...
                          NuclearData.cc: 253
                          NuclearData.cc: 251
                          NuclearData.cc: 248
                        ▶ 252 » [I] qs_vector<NuclearDataSpecies>::operator[](int)
                          NuclearData.cc: 252
                        ▶ 252 » [I] qs_vector<NuclearDataReaction>::size() const
```

HPCToolkit Resources

- Documentation
  - —User manual for HPCToolkit: http://hpctoolkit.org/manual/HPCToolkit-users-manual.pdf
  - —Cheat sheet: https://gitlab.com/hpctoolkit/hpctoolkit/-/wikis/HPCToolkit-cheat-sheet
  - —User manual for hpcviewer: https://hpctoolkit.gitlab.io/hpcviewer
  - —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-instructions.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/gpu_hack/hpctoolkit/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

```
% git clone https://gitlab.com/hpctoolkit/tutorial-examples
% cd hpctoolkit-tutorial-examples/gpu/intel
% ls
   arborx.kokkos.sycl   minitest.sycl   minitest.omp
```

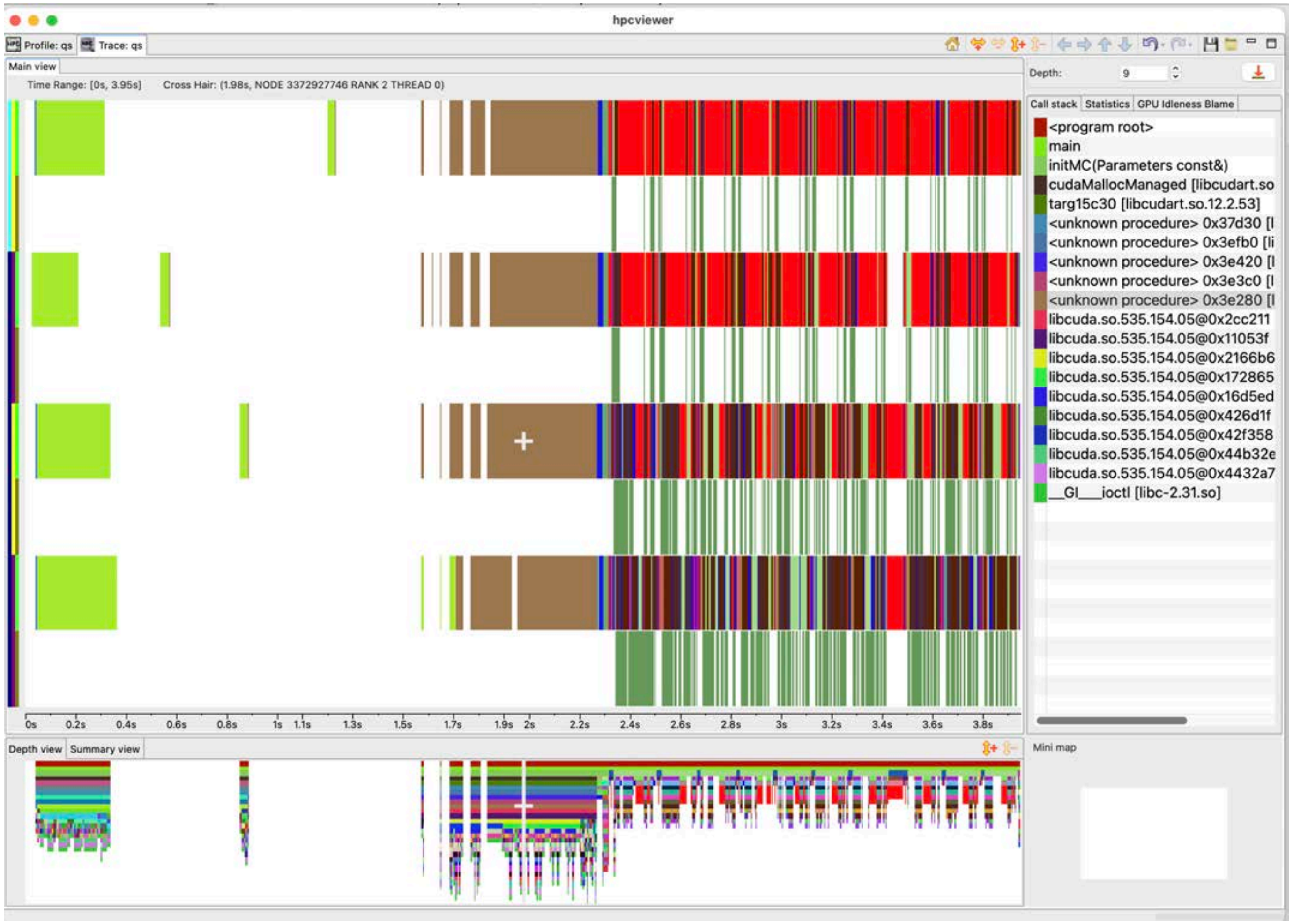
```
 For your chosen example
 1. cd to the example directory
 2. source setup-env/aurora.sh  # custom for each example
 3. make build # build the code
 4. make run # submit to the batch queue
     wait until the hpctoolkit database ending in .d appears and you see the file
     log.run.done appear
 5. make view # launch hpcviewer to explore the performance data collected
```

Hands-on Tutorial Example on Polaris: Quicksilver

```
% git clone https://gitlab.com/hpctoolkit/tutorial-examples
% cd hpctoolkit-tutorial-examples/gpu/nvidia/quicksilver.cuda
 1. source setup-env/polaris.sh  # custom for each example
 2. make build # build the code
 3. make run # submit to the batch queue
    wait until the hpctoolkit database hpctoolkit-qs.d appears and you see
    the file log.run.done appear
 4. make view # launch hpcviewer to explore the profiles and traces
 5. make run-pc # collect instruction-level kernel measurements
    wait until the hpctoolkit database hpctoolkit-qs-pc.d appears and you
    see the file log.run-pc.done appear
 6. make view-pc # launch hpcviewer to explore the pc sampling data
```

Inspecting the Pre-collected Quicksilver Data

Select the Tab “Trace: qs”

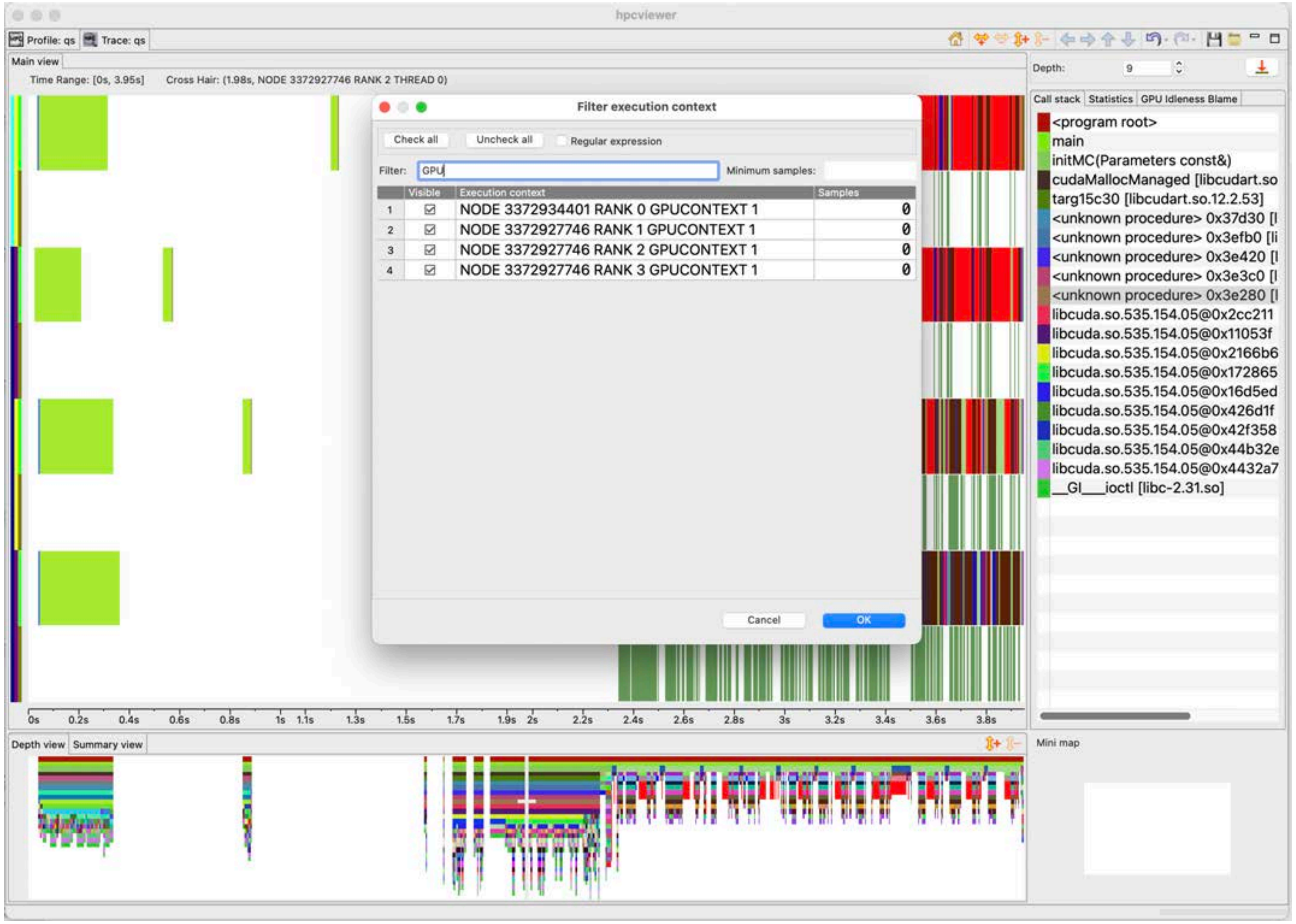


Time Range: [0s, 3.95s]   Cross Hair: (1.98s, NODE 3372927746 RANK 2 THREAD 0)

Depth: 9

```
<program root>
main
initMC(Parameters const&)
cudaMallocManaged [libcudart.so
targ15c30 [libcudart.so.12.2.53]
<unknown procedure> 0x37d30 [l
<unknown procedure> 0x3efb0 [li
<unknown procedure> 0x3e420 [l
<unknown procedure> 0x3e3c0 [l
<unknown procedure> 0x3e280 [l
libcuda.so.535.154.05@0x2cc211
libcuda.so.535.154.05@0x11053f
libcuda.so.535.154.05@0x2166b6
libcuda.so.535.154.05@0x172865
libcuda.so.535.154.05@0x16d5ed
libcuda.so.535.154.05@0x426d1f
libcuda.so.535.154.05@0x42f358
libcuda.so.535.154.05@0x44b32e
libcuda.so.535.154.05@0x4432a7
__GI___ioctl [libc-2.31.so]
```

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



Time Range: [0s, 3.95s]   Cross Hair: (1.98s, NODE 3372927746 RANK 2 THREAD 0)

Filter execution context

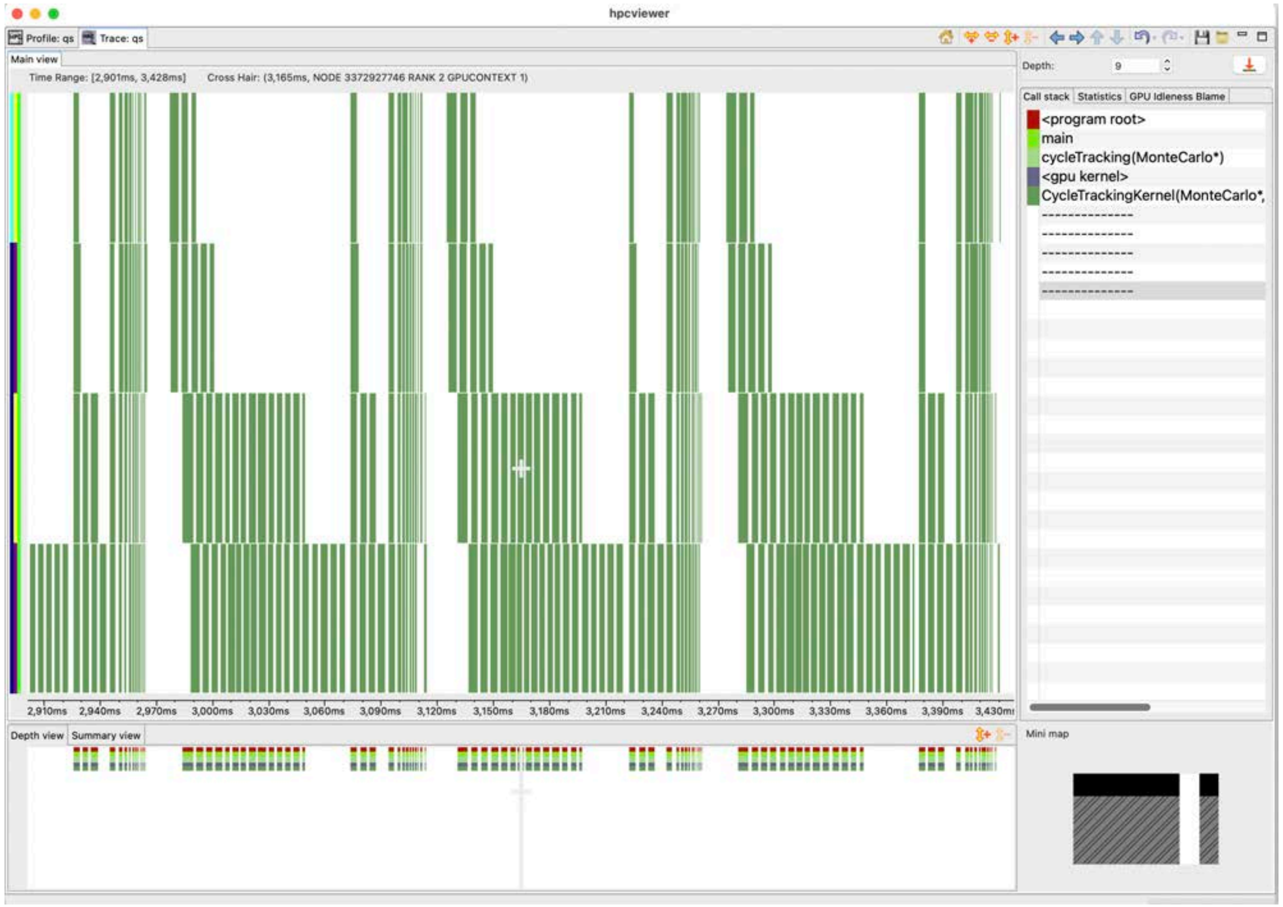
Check all   Uncheck all   Regular expression

Filter: GPU   Minimum samples:

	Visible	Execution context	Samples
1	☑	NODE 3372934401 RANK 0 GPUCONTEXT 1	0
2	☑	NODE 3372927746 RANK 1 GPUCONTEXT 1	0
3	☑	NODE 3372927746 RANK 2 GPUCONTEXT 1	0
4	☑	NODE 3372927746 RANK 3 GPUCONTEXT 1	0

<program root>
main
initMC(Parameters const&)
cudaMallocManaged [libcudart.so
targ15c30 [libcudart.so.12.2.53]
<unknown procedure> 0x37d30 [l
<unknown procedure> 0x3efb0 [li
<unknown procedure> 0x3e420 [l
<unknown procedure> 0x3e3c0 [l
<unknown procedure> 0x3e280 [l
libcuda.so.535.154.05@0x2cc211
libcuda.so.535.154.05@0x11053f
libcuda.so.535.154.05@0x2166b6
libcuda.so.535.154.05@0x172865
libcuda.so.535.154.05@0x16d5ed
libcuda.so.535.154.05@0x426d1f
libcuda.so.535.154.05@0x42f358
libcuda.so.535.154.05@0x44b32e
libcuda.so.535.154.05@0x4432a7
__GI___ioctl [libc-2.31.so]

See Load Imbalance Across the Four GPUs



Time Range: [2,901ms, 3,428ms]   Cross Hair: (3,165ms, NODE 3372927746 RANK 2 GPUCONTEXT 1)

- 🟥 <program root>
- 🟩 main
- 🟩 cycleTracking(MonteCarlo*)
- 🟪 <gpu kernel>
- 🟩 CycleTrackingKernel(MonteCarlo*,

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

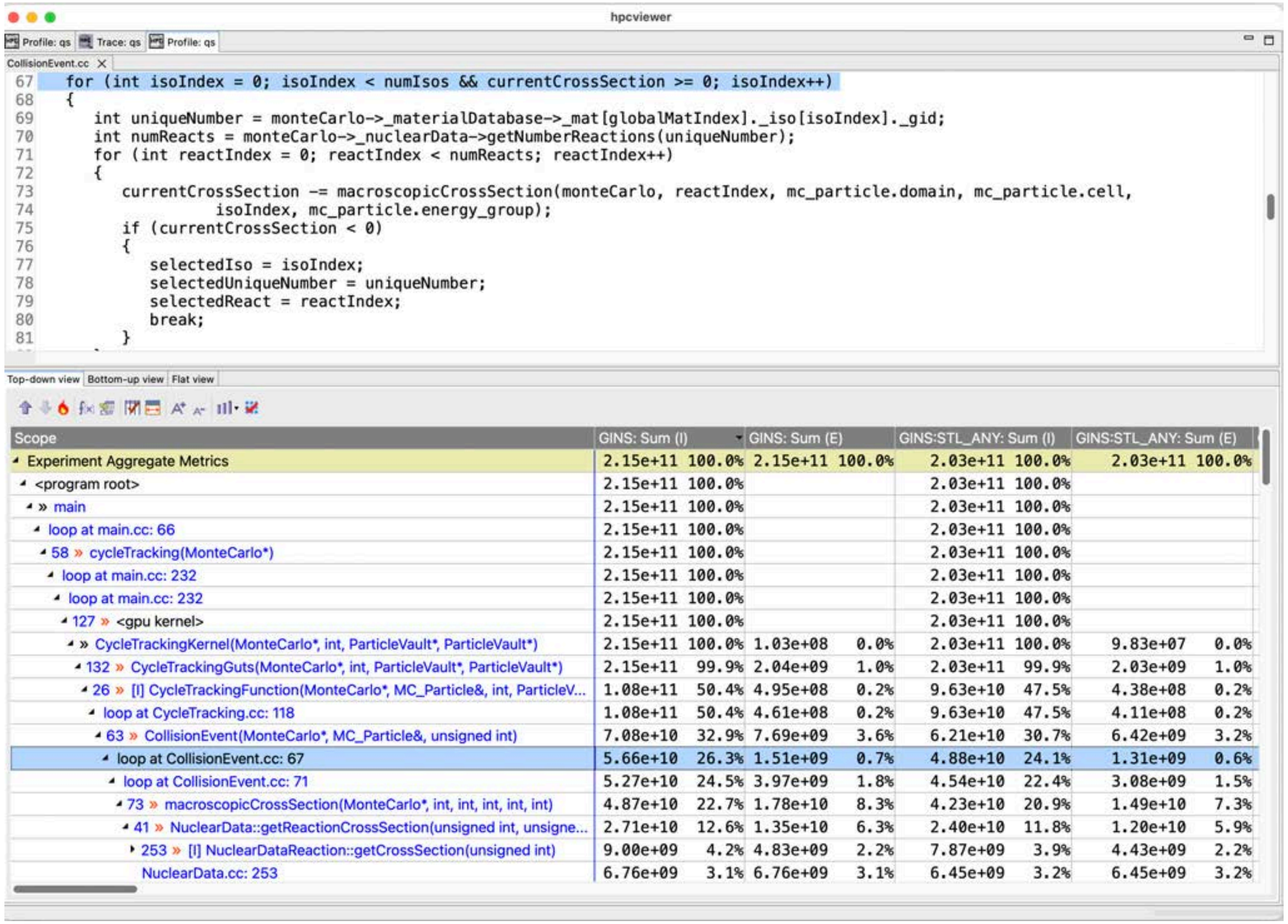
Using a measurement database with profiles and traces

- 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, but not attribute them to code

The Profile View in the other “PC Sampling” Database



```
67    for (int isoIndex = 0; isoIndex < numIsos && currentCrossSection >= 0; isoIndex++)
68    {
69       int uniqueNumber = monteCarlo->_materialDatabase->_mat[globalMatIndex]._iso[isoIndex]._gid;
70       int numReacts = monteCarlo->_nuclearData->getNumberReactions(uniqueNumber);
71       for (int reactIndex = 0; reactIndex < numReacts; reactIndex++)
72       {
73          currentCrossSection -= macroscopicCrossSection(monteCarlo, reactIndex, mc_particle.domain, mc_particle.cell,
74                    isoIndex, mc_particle.energy_group);
75          if (currentCrossSection < 0)
76          {
77             selectedIso = isoIndex;
78             selectedUniqueNumber = uniqueNumber;
79             selectedReact = reactIndex;
80             break;
81          }
```

Scope	GINS: Sum (I)		GINS: Sum (E)		GINS:STL_ANY: Sum (I)		GINS:STL_ANY: Sum (E)	
Experiment Aggregate Metrics	2.15e+11	100.0%	2.15e+11	100.0%	2.03e+11	100.0%	2.03e+11	100.0%
<program root>	2.15e+11	100.0%			2.03e+11	100.0%		
» main	2.15e+11	100.0%			2.03e+11	100.0%		
loop at main.cc: 66	2.15e+11	100.0%			2.03e+11	100.0%		
58 » cycleTracking(MonteCarlo*)	2.15e+11	100.0%			2.03e+11	100.0%		
loop at main.cc: 232	2.15e+11	100.0%			2.03e+11	100.0%		
loop at main.cc: 232	2.15e+11	100.0%			2.03e+11	100.0%		
127 » <gpu kernel>	2.15e+11	100.0%			2.03e+11	100.0%		
» CycleTrackingKernel(MonteCarlo*, int, ParticleVault*, ParticleVault*)	2.15e+11	100.0%	1.03e+08	0.0%	2.03e+11	100.0%	9.83e+07	0.0%
132 » CycleTrackingGuts(MonteCarlo*, int, ParticleVault*, ParticleVault*)	2.15e+11	99.9%	2.04e+09	1.0%	2.03e+11	99.9%	2.03e+09	1.0%
26 » [I] CycleTrackingFunction(MonteCarlo*, MC_Particle&, int, ParticleV...	1.08e+11	50.4%	4.95e+08	0.2%	9.63e+10	47.5%	4.38e+08	0.2%
loop at CycleTracking.cc: 118	1.08e+11	50.4%	4.61e+08	0.2%	9.63e+10	47.5%	4.11e+08	0.2%
63 » CollisionEvent(MonteCarlo*, MC_Particle&, unsigned int)	7.08e+10	32.9%	7.69e+09	3.6%	6.21e+10	30.7%	6.42e+09	3.2%
loop at CollisionEvent.cc: 67	5.66e+10	26.3%	1.51e+09	0.7%	4.88e+10	24.1%	1.31e+09	0.6%
loop at CollisionEvent.cc: 71	5.27e+10	24.5%	3.97e+09	1.8%	4.54e+10	22.4%	3.08e+09	1.5%
73 » macroscopicCrossSection(MonteCarlo*, int, int, int, int, int)	4.87e+10	22.7%	1.78e+10	8.3%	4.23e+10	20.9%	1.49e+10	7.3%
41 » NuclearData::getReactionCrossSection(unsigned int, unsigne...	2.71e+10	12.6%	1.35e+10	6.3%	2.40e+10	11.8%	1.20e+10	5.9%
253 » [I] NuclearDataReaction::getCrossSection(unsigned int)	9.00e+09	4.2%	4.83e+09	2.2%	7.87e+09	3.9%	4.43e+09	2.2%
NuclearData.cc: 253	6.76e+09	3.1%	6.76e+09	3.1%	6.45e+09	3.2%	6.45e+09	3.2%

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

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

    Install a recent Java (17 or 21) if you don’t have one, using the directions on the page
- 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 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`

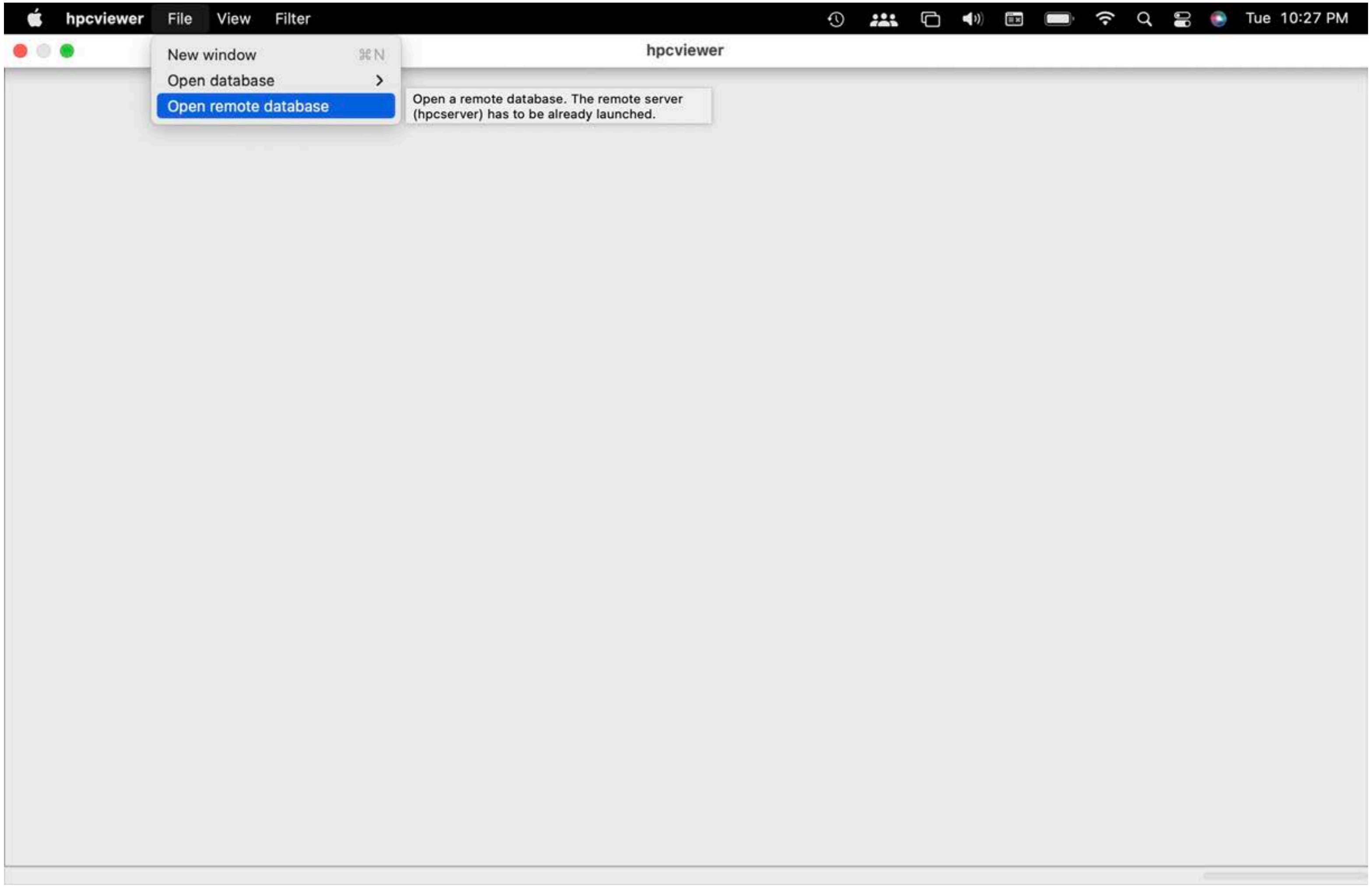
Select the authentication method: “Use password”

Click “OK”

Authenticate using your token as you normally do

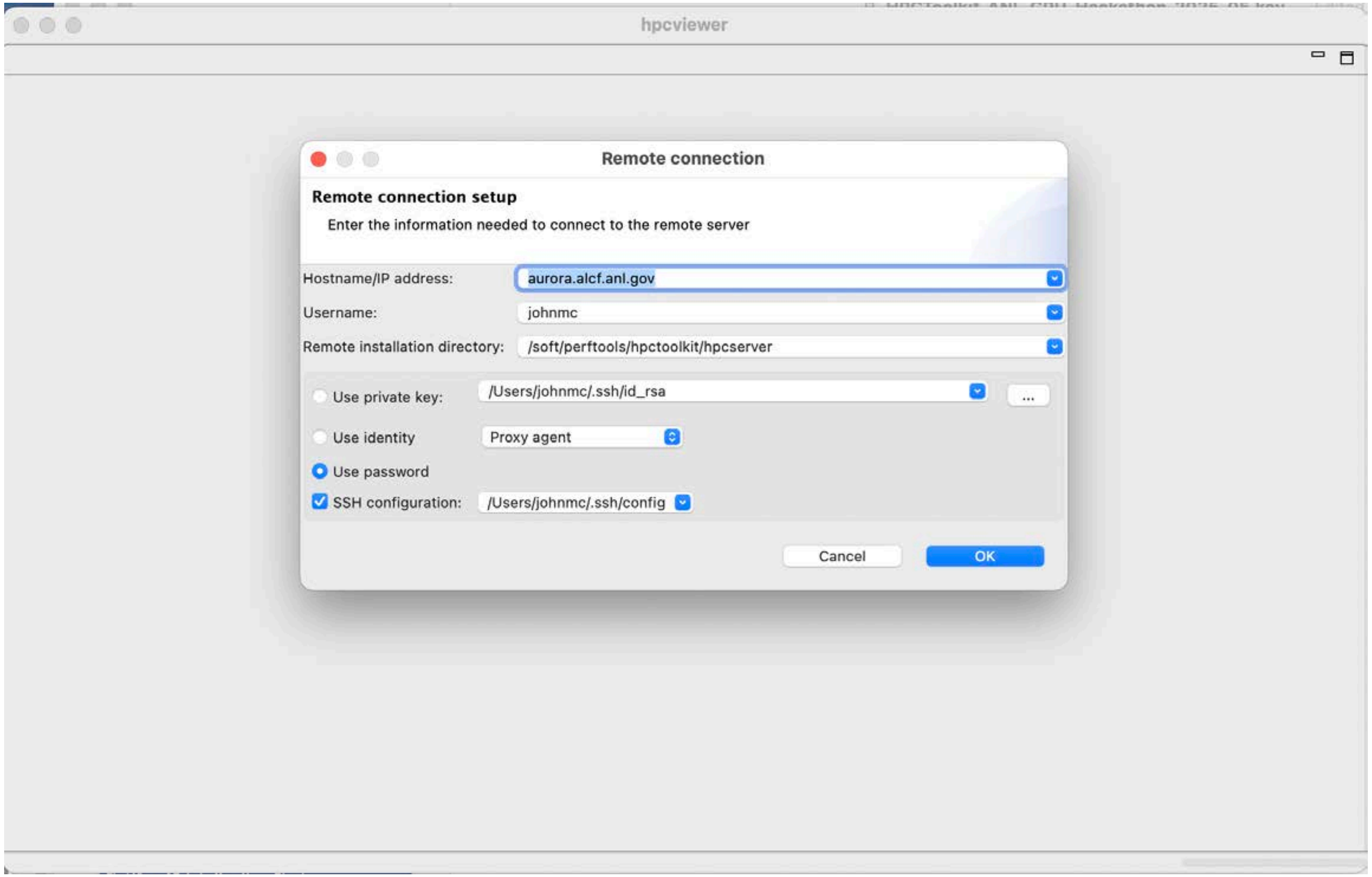
Navigate to a database with the file chooser in /flare/gpu_hack/hpctoolkit/data

Opening a Remote Database



Open a remote database. The remote server (hpcserver) has to be already launched.

Configuring for remote access to Aurora using hpcserver



Remote connection setup

Enter the information needed to connect to the remote server

Hostname/IP address: aurora.alcf.anl.gov

Username: johnmc

Remote installation directory: /soft/perftools/hpctoolkit/hpcserver

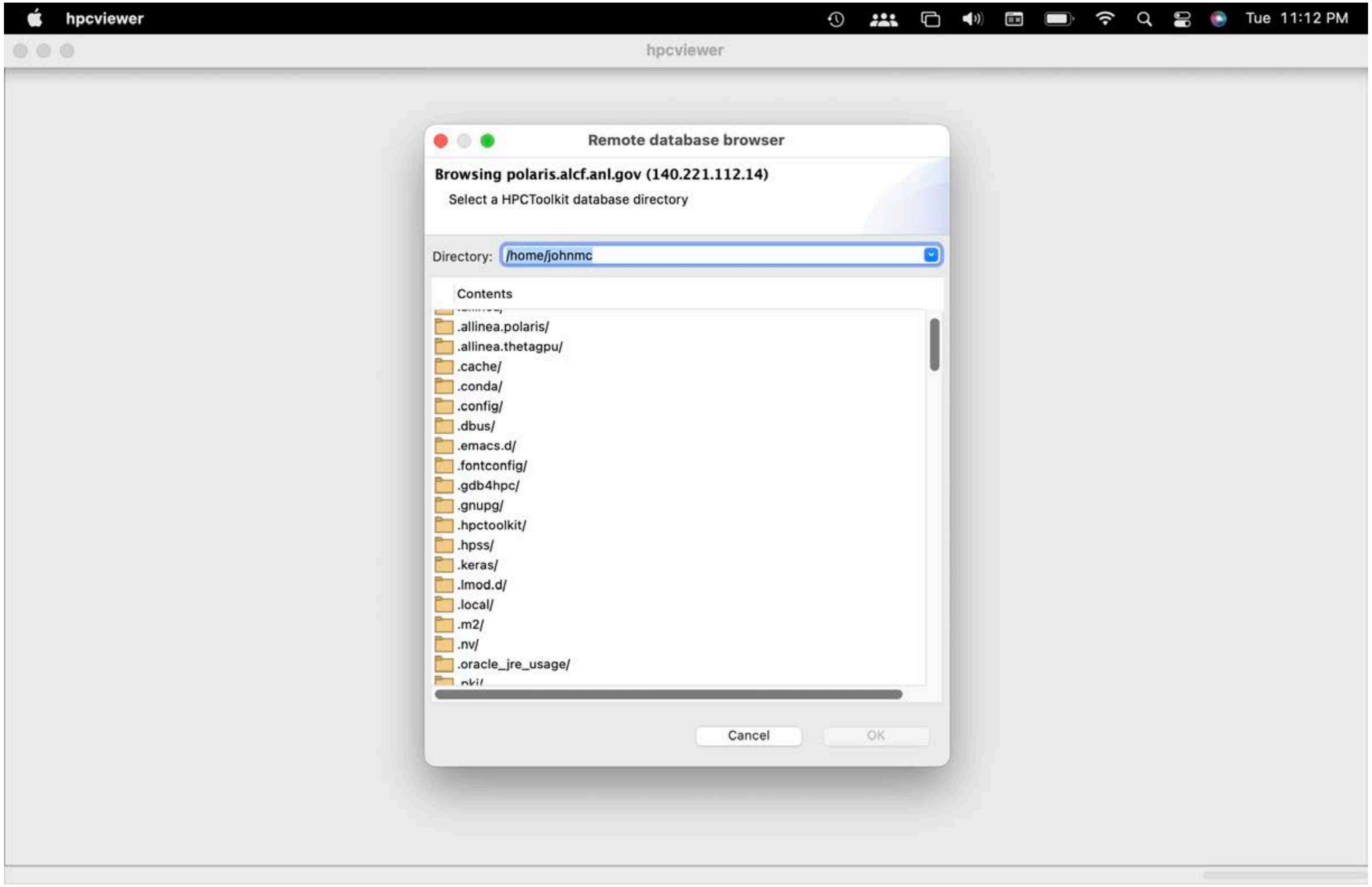
Use private key: /Users/johnmc/.ssh/id_rsa

Use identity: Proxy agent

[✓ Use password]

[✓ SSH configuration:] /Users/johnmc/.ssh/config

First View of Aurora: Your Home Directory



Browsing polaris.alcf.anl.gov (140.221.112.14)

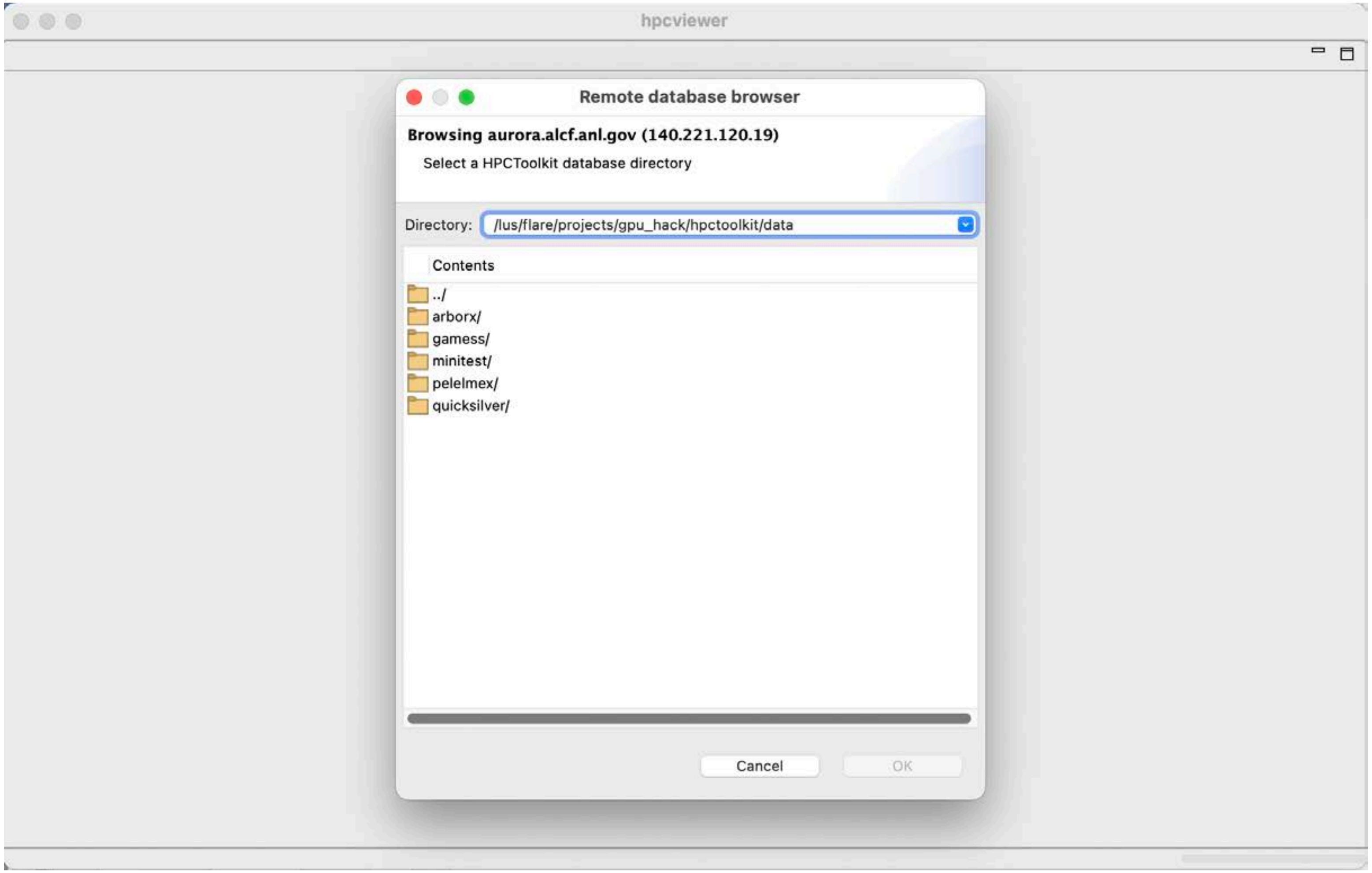
Select a HPCToolkit database directory

Directory: /home/johnmc

Contents

```
.allinea.polaris/
.allinea.thetagpu/
.cache/
.conda/
.config/
.dbus/
.emacs.d/
.fontconfig/
.gdb4hpc/
.gnupg/
.hpctoolkit/
.hpss/
.keras/
.lmod.d/
.local/
.m2/
.nv/
.oracle_jre_usage/
```

Navigate to Example Databases



Browsing aurora.alcf.anl.gov (140.221.120.19)

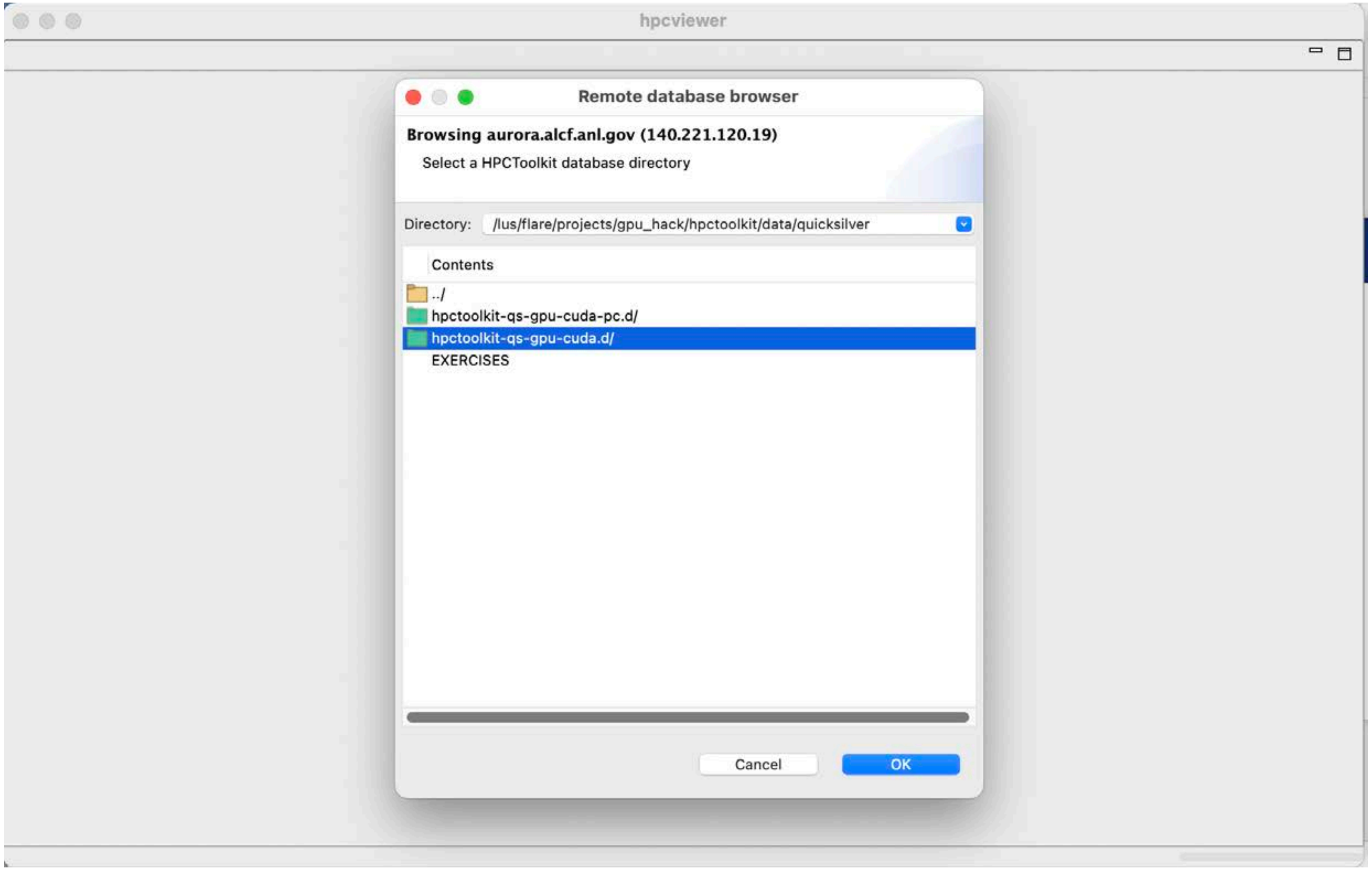
Select a HPCToolkit database directory

Directory: /lus/flare/projects/gpu_hack/hpctoolkit/data

Contents

../
arborx/
gamess/
minitest/
pelelmex/
quicksilver/

Select a Quicksilver Database with Traces



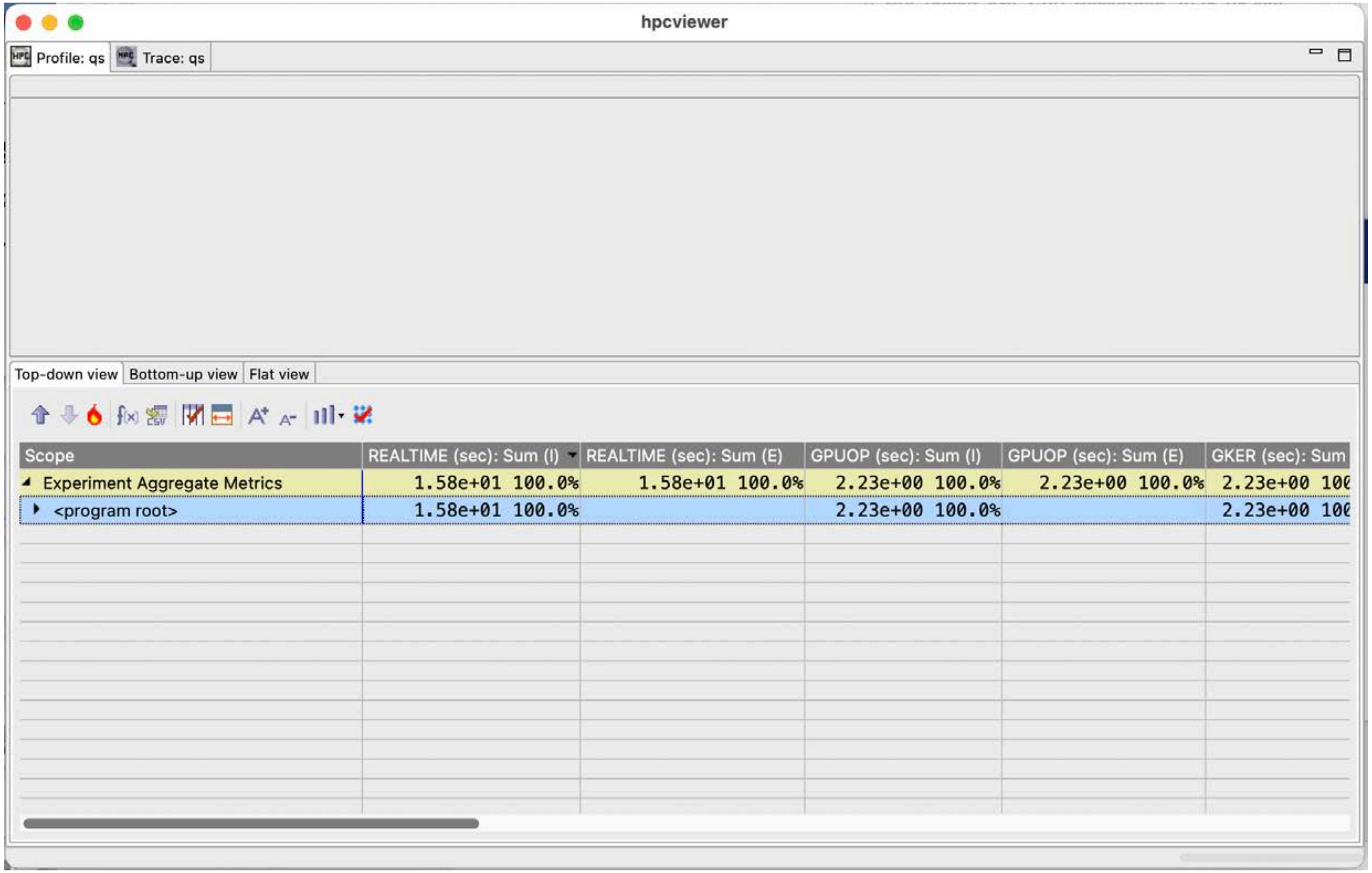
Browsing aurora.alcf.anl.gov (140.221.120.19)

Select a HPCToolkit database directory

Directory: /lus/flare/projects/gpu_hack/hpctoolkit/data/quicksilver

Contents
../
hpctoolkit-qs-gpu-cuda-pc.d/
hpctoolkit-qs-gpu-cuda.d/
EXERCISES

After Selecting hpctoolkit-qs-gpu-cuda.d



Scope	REALTIME (sec): Sum (I)	REALTIME (sec): Sum (E)	GPUOP (sec): Sum (I)	GPUOP (sec): Sum (E)	GKER (sec): Sum
Experiment Aggregate Metrics	1.58e+01 100.0%	1.58e+01 100.0%	2.23e+00 100.0%	2.23e+00 100.0%	2.23e+00 100
<program root>	1.58e+01 100.0%		2.23e+00 100.0%		2.23e+00 100

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