Measurement and Analysis of GPU-accelerated Applications with HPCToolkit

Jonathon Anderson, Yuning Xia, John Mellor-Crummey
Rice University

Consortium for the Advancement of Scientific Software
Birds of a Feather Days
February 11, 2026

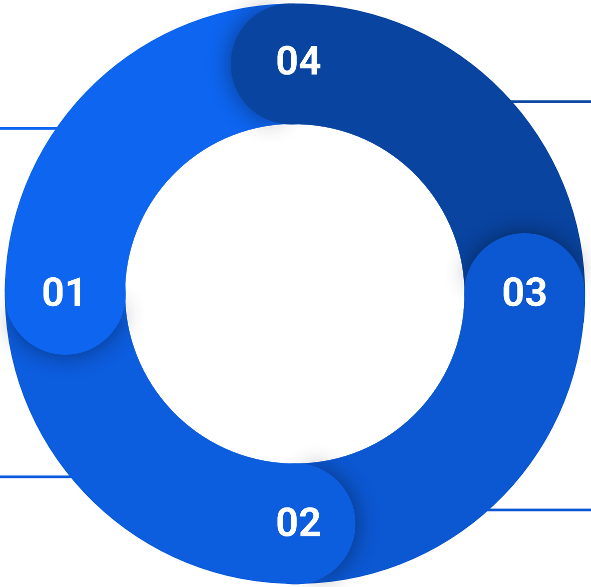
DOE's GPU-Accelerated Exascale Platforms

Argonne NATIONAL LABORATORY
U.S. DEPARTMENT OF ENERGY
intel
Hewlett Packard Enterprise
Aurora

U.S. DEPARTMENT OF ENERGY
NNSA
Lawrence Livermore National Laboratory
AMD
CRAY
EL CAPITAN

- Frontier compute nodes (OLCF)
  - 1 AMD EPYC “Trento” CPU
  - 4 MI250X AMD Radeon Instinct GPUs
  - 4 Slingshot 11 endpoints
  - Unified memory architecture
- Aurora compute nodes (ALCF)
  - 2 Intel Xeon “Sapphire Rapids” processors
  - 6 Intel Data Center GPU Max 1500
  - 8 Slingshot 11 endpoints
  - Unified memory architecture
- El Capitan compute nodes (LLNL)
  - 4 AMD MI300A APU
  - 4 Slingshot 11 endpoints
  - Unified memory architecture

Tuning HPC Applications



Profile the Application

Collect detailed performance data on an application execution

Analyze Performance Data

Aggregate performance data from all application threads and generate a database of “analysis results”

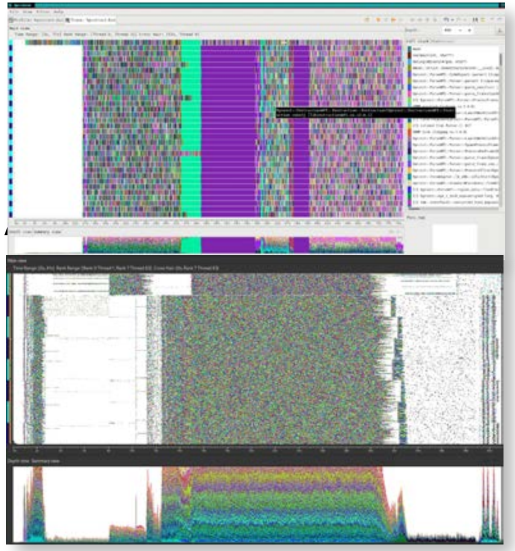
Inspect and Identify

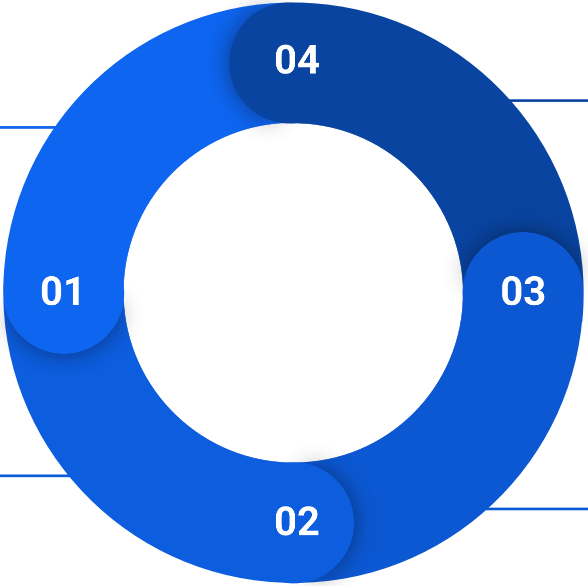
Inspect locations with poor performance and identify root causes

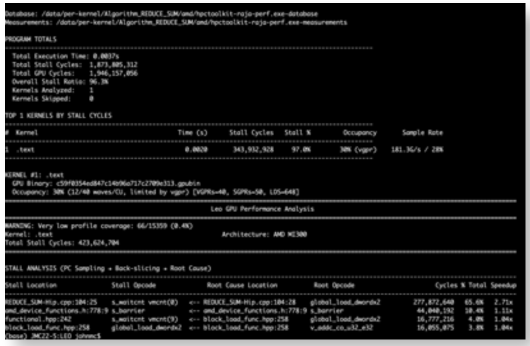
Optimize and Improve

Remove performance bottlenecks and further improve the application

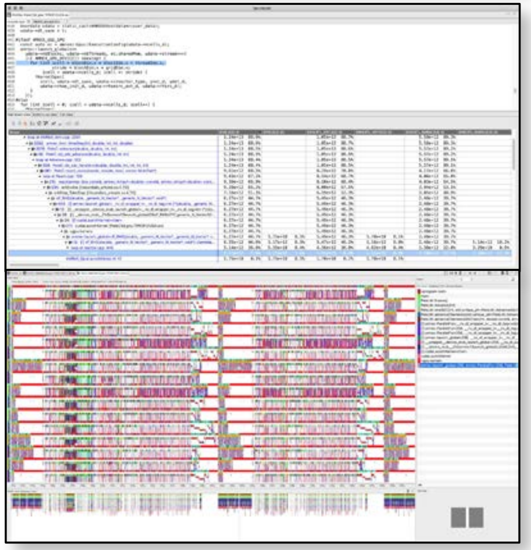
Tuning HPC Applications







```
Database: /data/per-kernel/Algorithm_REDUCE_SUM/amd/hpctoolkit-raja-perf.exe-database
Measurements: /data/per-kernel/Algorithm_REDUCE_SUM/amd/hpctoolkit-raja-perf.exe-measurements

PROGRAM TOTALS
--------------------------------------------------------------------------------
  Total Execution Time: 0.0037s
  Total Stall Cycles:  1,873,805,312
  Total GPU Cycles:    1,946,157,056
  Overall Stall Ratio: 96.3%
  Kernels Analyzed:    1
  Kernels Skipped:     0

TOP 1 KERNELS BY STALL CYCLES
--------------------------------------------------------------------------------
#  Kernel                                 Time (s)     Stall Cycles   Stall %       Occupancy      Sample Rate
--------------------------------------------------------------------------------
1  .text                                  0.0020       343,932,928    97.0%        30% (vgpr)    181.3G/s / 28%
--------------------------------------------------------------------------------

KERNEL #1: .text
  GPU Binary: c59f0354ed847c14b96a717c2709e313.gpubin
  Occupancy: 30% (12/40 waves/CU, limited by vgpr) [VGPRs=40, SGPRs=50, LDS=648]
================================================================================================
                                         Leo GPU Performance Analysis
================================================================================================
WARNING: Very low profile coverage: 66/15359 (0.4%)
Kernel: .text                                         Architecture: AMD MI300
Total Stall Cycles: 423,624,704
================================================================================================

STALL ANALYSIS (PC Sampling + Back-slicing + Root Cause)
------------------------------------------------------------------------------------------------
Stall Location              Stall Opcode         Root Cause Location          Root Opcode                   Cycles % Total  Speedup
------------------------------------------------------------------------------------------------
REDUCE_SUM-Hip.cpp:104:25   s_waitcnt vmcnt(0)   <-- REDUCE_SUM-Hip.cpp:104:28  global_load_dwordx2      277,872,640   65.6%   2.71x
amd_device_functions.h:778:9 s_barrier           <-- amd_device_functions.h:778:9 s_barrier                44,040,192   10.4%   1.11x
functional.hpp:242          s_waitcnt vmcnt(9)   <-- block_load_func.hpp:258    global_load_dwordx2       16,777,216    4.0%   1.04x
block_load_func.hpp:258     global_load_dwordx2  <-- block_load_func.hpp:258    v_addc_co_u32_e32         16,055,075    3.8%   1.04x
(base) JMC22-5:LEO johnmc$
```



Why HPCToolkit?

- **Widely applicable**: many parallel programming models within & across nodes
- **Easy**: profiles unmodified application binaries
- **Fast:** low-overhead measurement; parallel analysis of large performance data
- **Scalable**: measure and analyze GPU-accelerated executions at large scale
- **Informative**: learn where an application spends its time and why
  - call path profiles associate metrics with application source code contexts
  - hierarchical traces reveal execution dynamics
  - LEO identifies and quantifies root causes of GPU performance losses
- **Broad audience**: developers of applications, frameworks, runtimes & tools
- **Multiplatform**: unlike vendor tools, works with a wide range of CPUs and GPUs

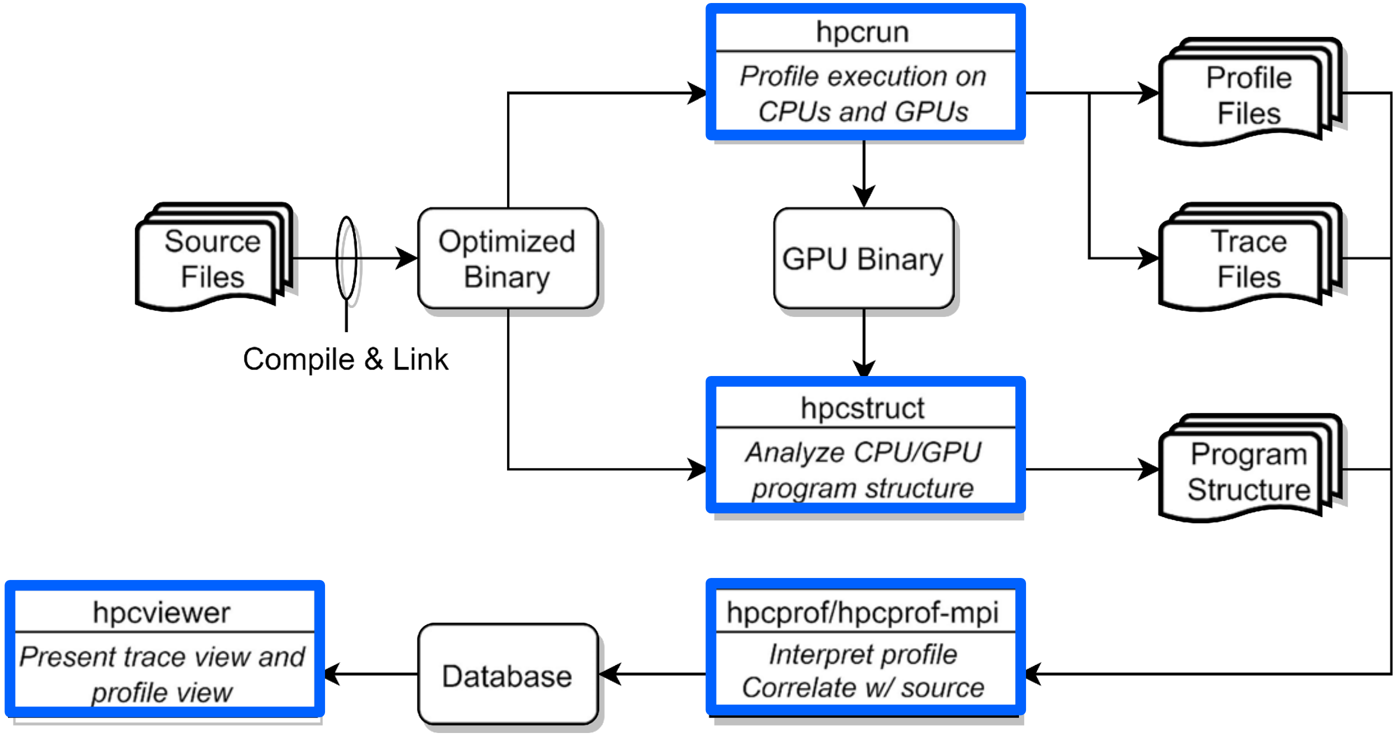
How is HPCToolkit Different from Vendor Tools?

- More scalable tracing than vendor tools
  - measure exascale executions across many nodes and GPUs
  - GUI supports interactive exploration of TB of performance data
- Scalable, parallel post-mortem analysis vs. non-scalable in-GUI analysis
- Detailed reconstruction of calling context profiles within GPU kernels
- Identifies and quantifies root causes of GPU performance losses

Today’s Agenda

- Introduce HPCToolkit tools and workflow
- Illustrate HPCToolkit’s use with some case studies
- Live demos
  - profiling and tracing
  - instruction-based performance metrics
- Instruction-level performance monitoring
  - explaining instruction-level performance
  - improving analysis and attribution of PC samples
  - automating analysis of GPU bottlenecks
- Discussion about needs, problems, and suggestions

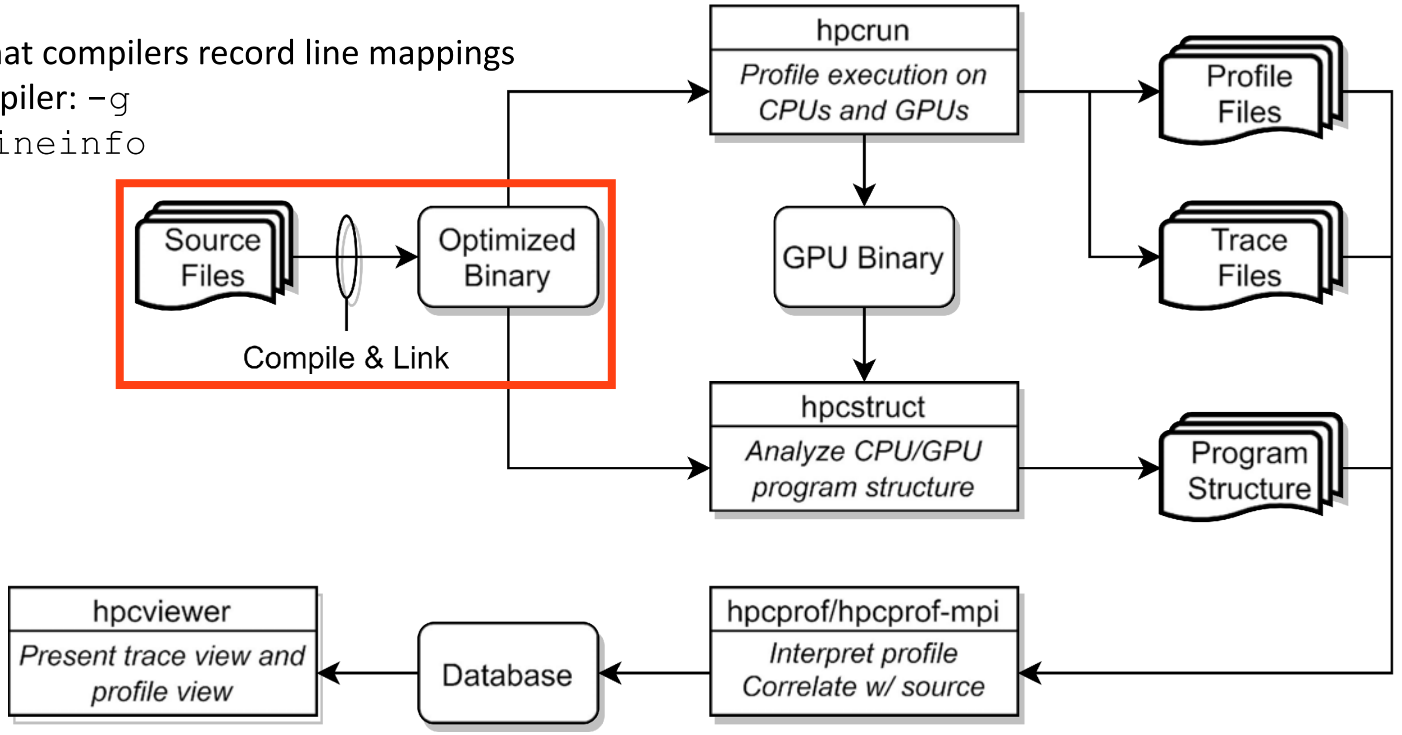
HPCToolkit's Workflow for GPU-accelerated Applications



HPCToolkit's Workflow for GPU-accelerated Applications

Step 1:

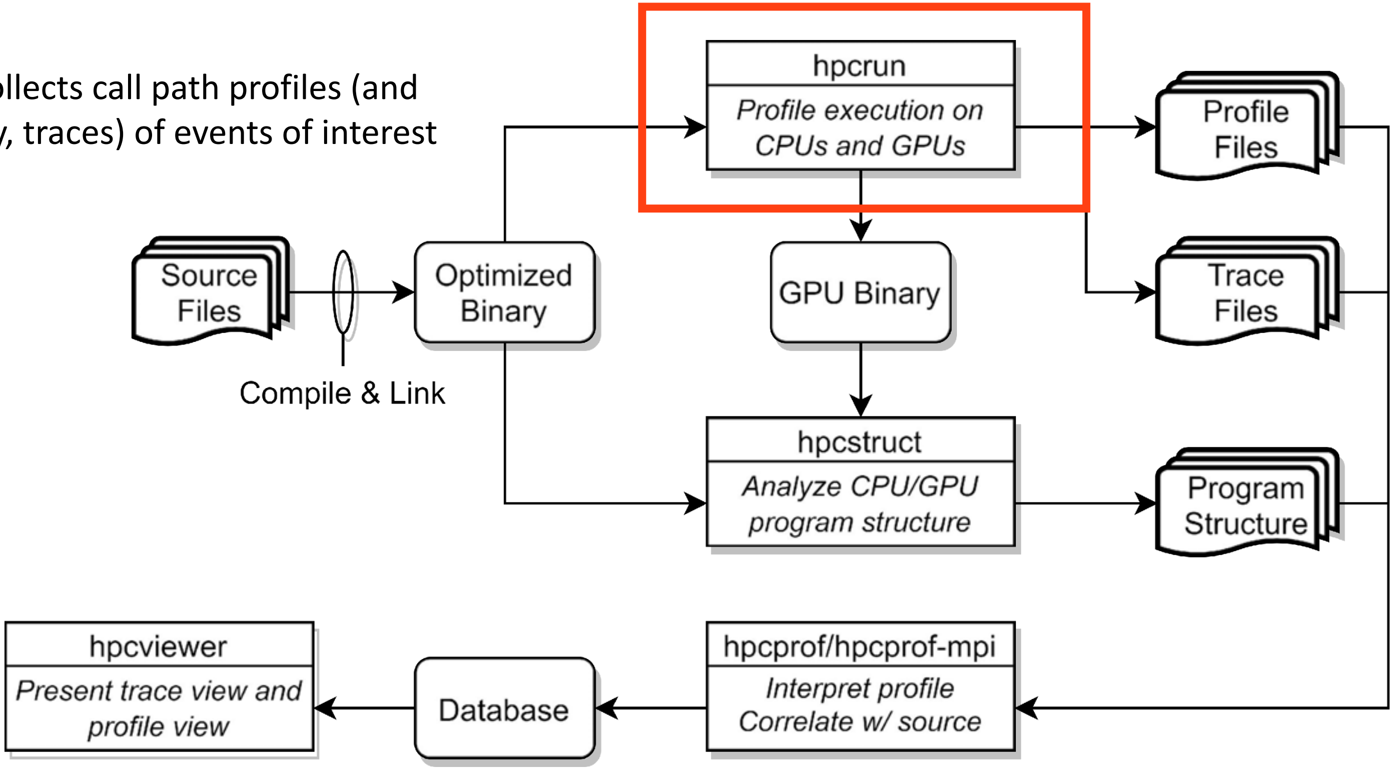
- Ensure that compilers record line mappings
- host compiler: `-g`
- nvcc: `-lineinfo`



HPCToolkit’s Workflow for GPU-accelerated Applications

Step 2:

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



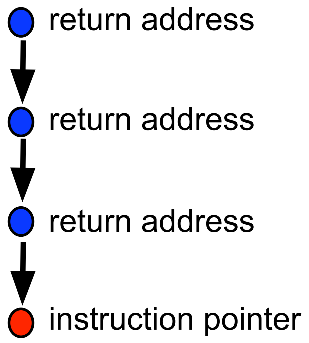
Measurement of CPU and GPU-accelerated Applications

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

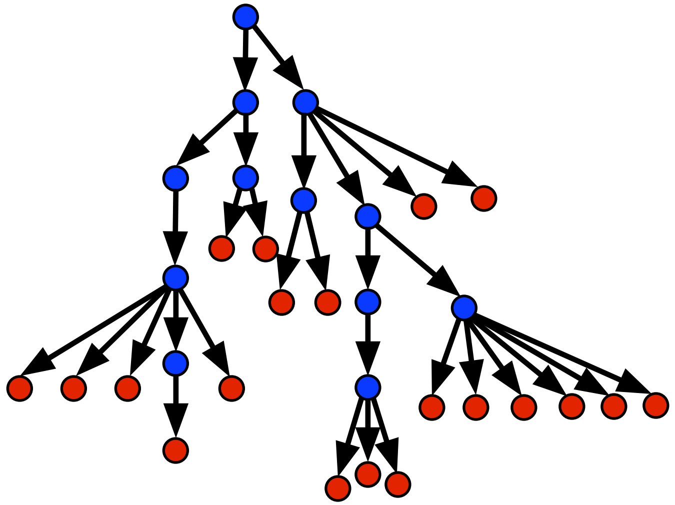
Call Stack Unwinding to Attribute Costs in Context

- Unwind when timer or hardware counter overflows
  - —measurement overhead ∝ 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

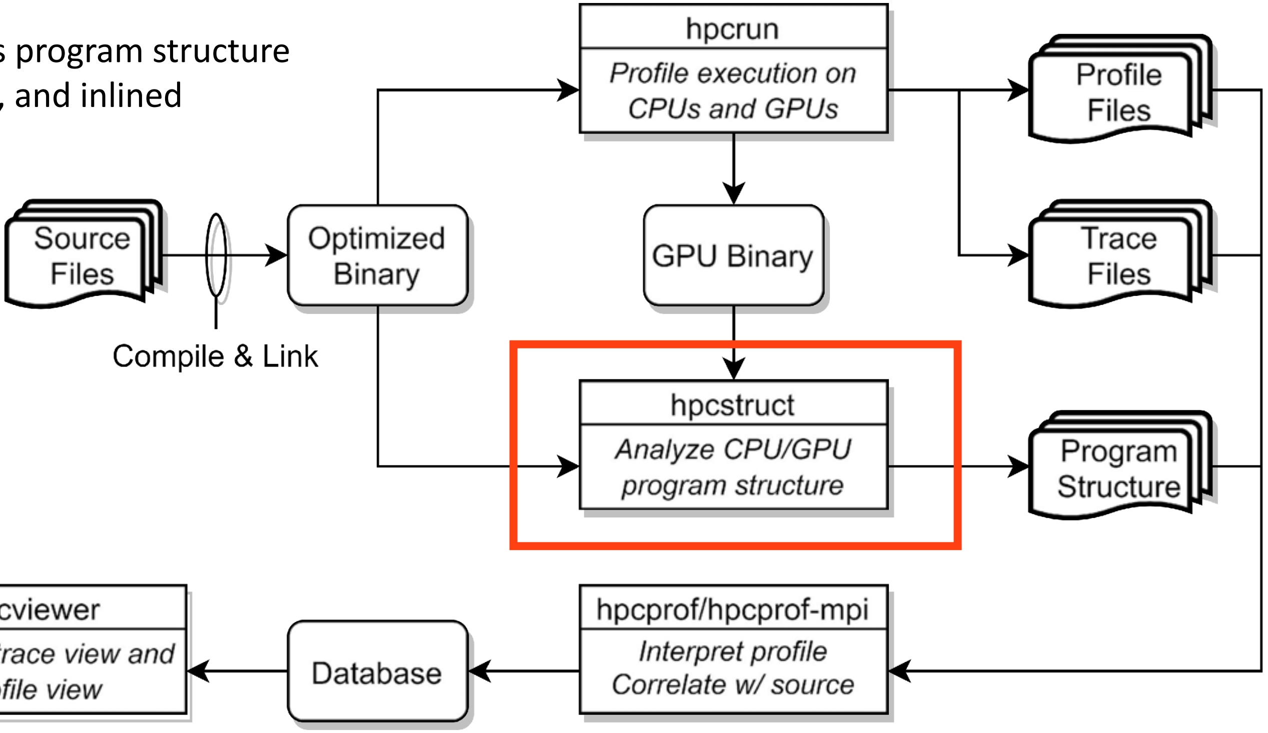
- CPU and GPU profiling
  - `hpcrun -e CPUTIME -e gpu=XXX <app> …`
- CPU and GPU profiling and tracing
  - `hpcrun -e CPUTIME -e gpu=xxx -tt <app>`
- GPU PC sampling
  - `hpcrun -e gpu=yyy,pc <app>`
- Measuring MPI programs
  - `srun -n <ranks> … hpcrun -e CPUTIME -e gpu=XXX <app>`

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

HPCToolkit’s Workflow for GPU-accelerated Applications

Step 3:

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



hpcstruct: Analyze CPU and GPU Binaries Using Multiple Threads

- Usage

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

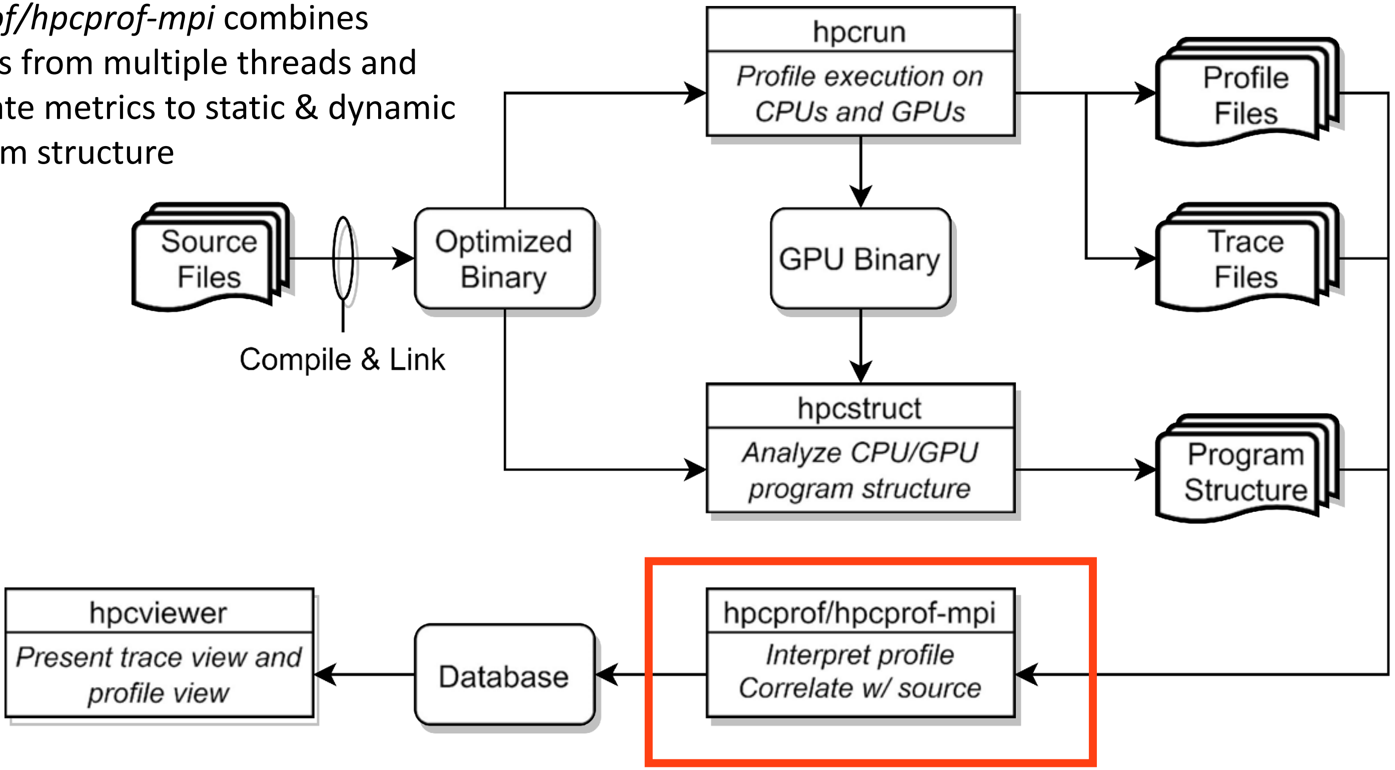
- What it does
  - Recover program structure information
    - Files, functions, inlined templates or functions, loops, source lines
  - In parallel, analyze all CPU and GPU binaries that were measured by HPCToolkit
    — typically analyze large application binaries with 16 threads
    — typically analyze 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 detailed analysis of 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-scale executions with multithreading

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

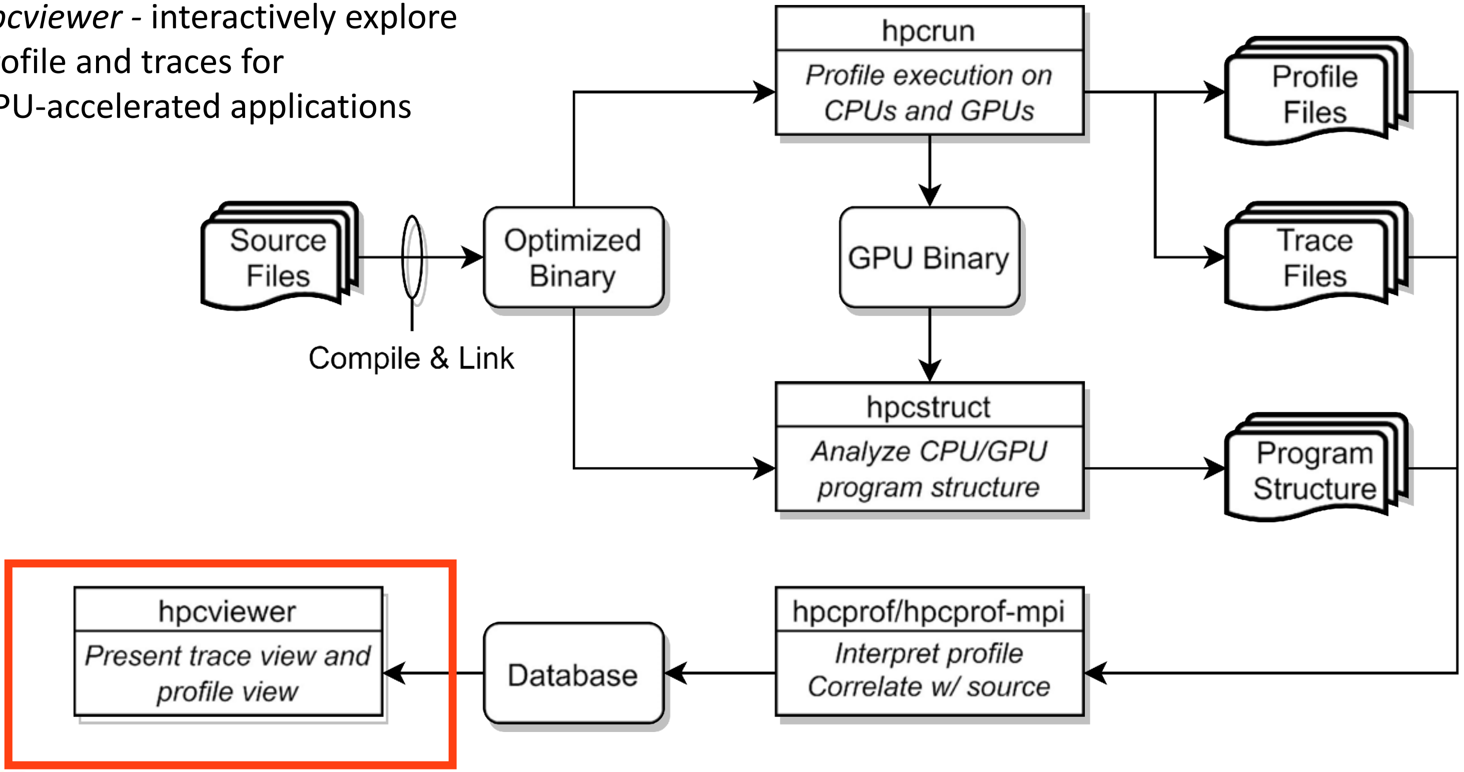
- Analyze data from large-scale executions with distributed-memory parallelism + multithreading

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

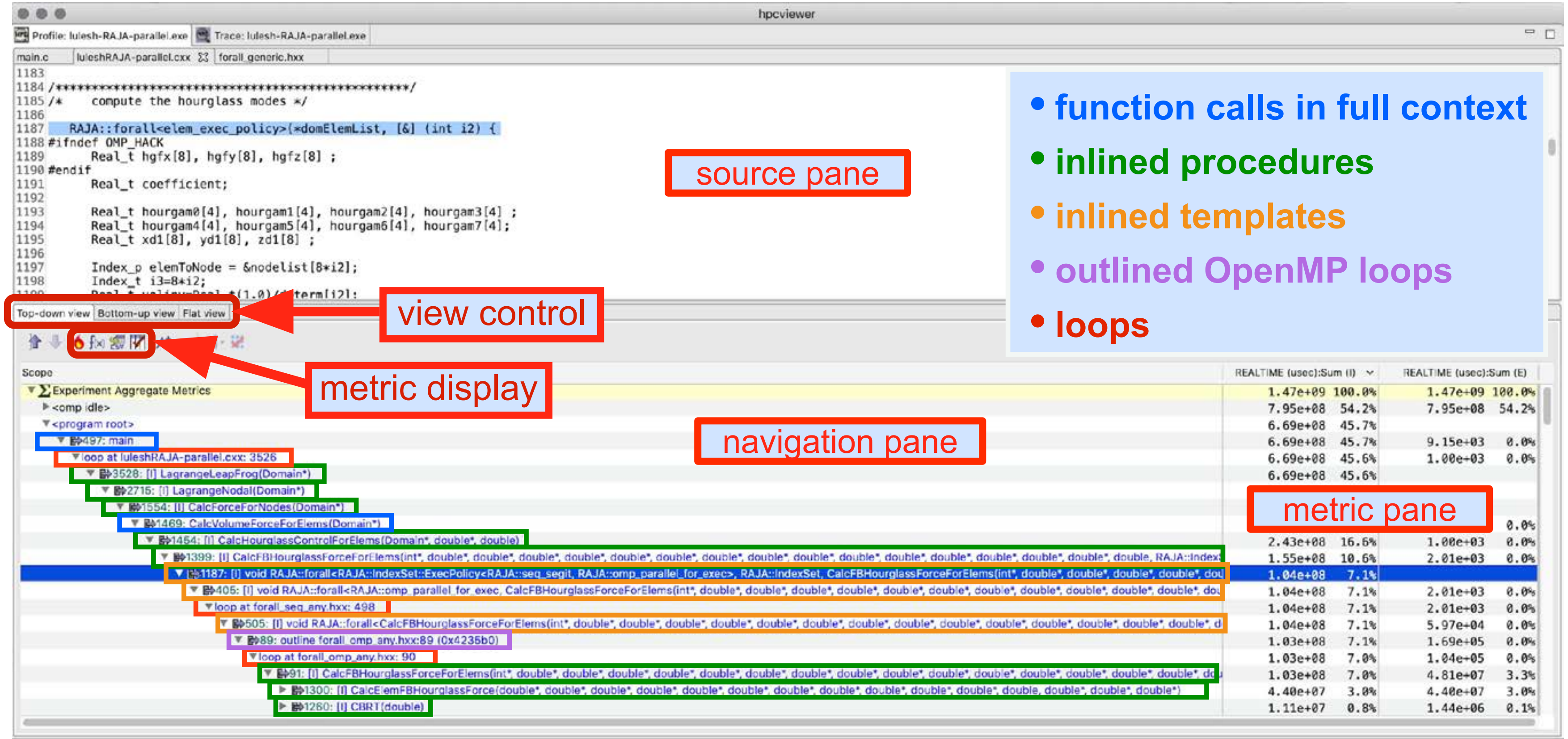
HPCToolkit’s Workflow for GPU-accelerated Applications

Step 4:

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



Code-centric Analysis with hpcviewer



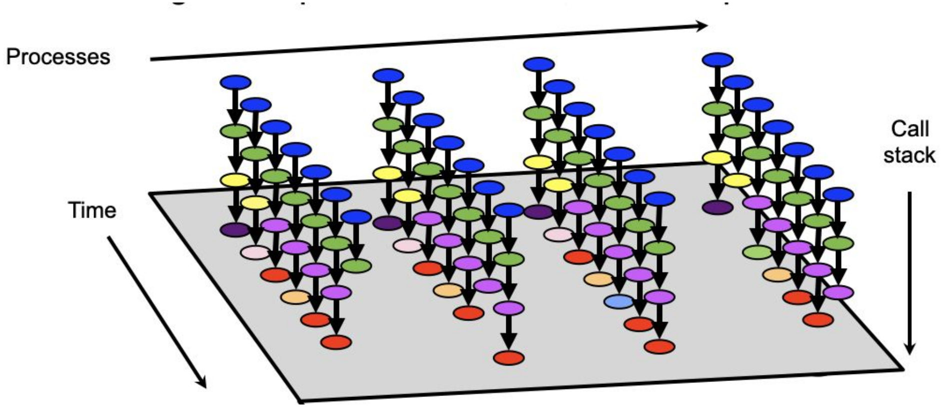
```
/*****************************************************/
/*    compute the hourglass modes */

   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];
```

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

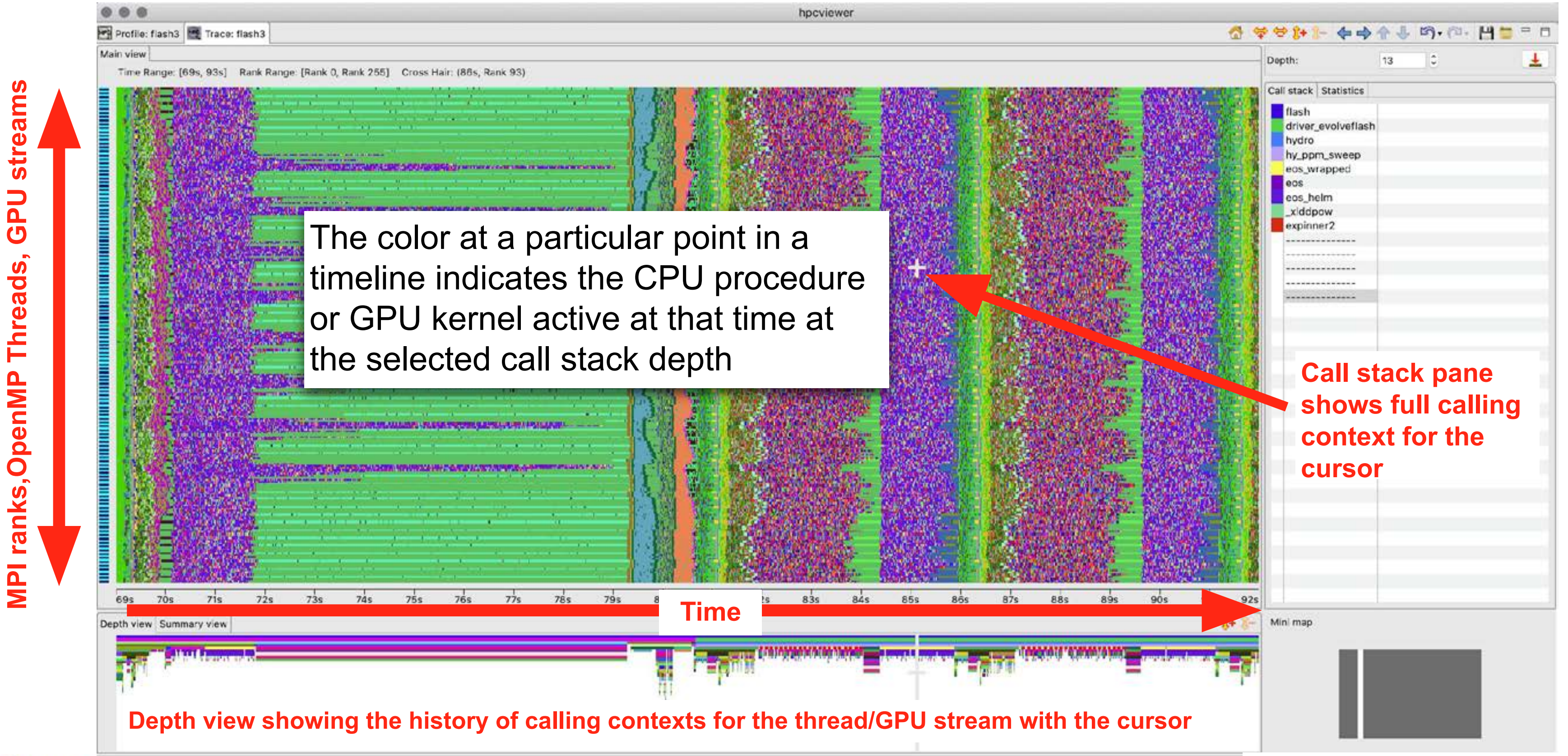
```
Scope                                                                  REALTIME (usec):Sum (I)   REALTIME (usec):Sum (E)
▼ Σ Experiment Aggregate Metrics                                          1.47e+09 100.0%           1.47e+09 100.0%
  ▶ <omp idle>                                                            7.95e+08  54.2%           7.95e+08  54.2%
  ▼ <program root>                                                        6.69e+08  45.7%
    ▼ 497: main                                                           6.69e+08  45.7%           9.15e+03   0.0%
      ▼ loop at luleshRAJA-parallel.cxx: 3526                             6.69e+08  45.6%           1.00e+03   0.0%
        ▼ 3528: [I] LagrangeLeapFrog(Domain*)                             6.69e+08  45.6%
          ▼ 2715: [I] LagrangeNodal(Domain*)
            ▼ 1554: [I] CalcForceForNodes(Domain*)
              ▼ 1469: CalcVolumeForceForElems(Domain*)                                                       0.0%
                ▼ 1464: [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::Index   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*, 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*, double*, dou   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*, 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



Understanding hpcviewer's Trace View



Time Range: [69s, 93s]   Rank Range: [Rank 0, Rank 255]   Cross Hair: (86s, Rank 93)

Depth: 13

🟦	flash
🟩	driver_evolveflash
🟦	hydro
🟪	hy_ppm_sweep
🟨	eos_wrapped
🟪	eos
🟪	eos_helm
🟩	_xlddpow
🟥	expinner2

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

Today's Agenda

- Introduce HPCToolkit tools and workflow
- Illustrate HPCToolkit's use with some case studies
- Live demos
  - profiling and tracing
  - instruction-based performance metrics
- Instruction-level performance monitoring
  - explaining instruction-level performance
  - improving analysis and attribution of PC samples
  - automating analysis of GPU bottlenecks
- Discussion about needs, problems, and suggestions

Case Studies

- ExaWind (Nalu-Wind + AMRWind) - Wind turbine and wind farm simulation
- PeleLMeX - Adaptive mesh hydrodynamics code for low mach number reacting flows
- GAMESS (OpenMP) - general ab initio quantum chemistry package
- LAMMPS (Kokkos) - classical molecular dynamics code with a focus on materials modeling

At Exascale!

ExaWind: Modeling Turbine Wake Formation

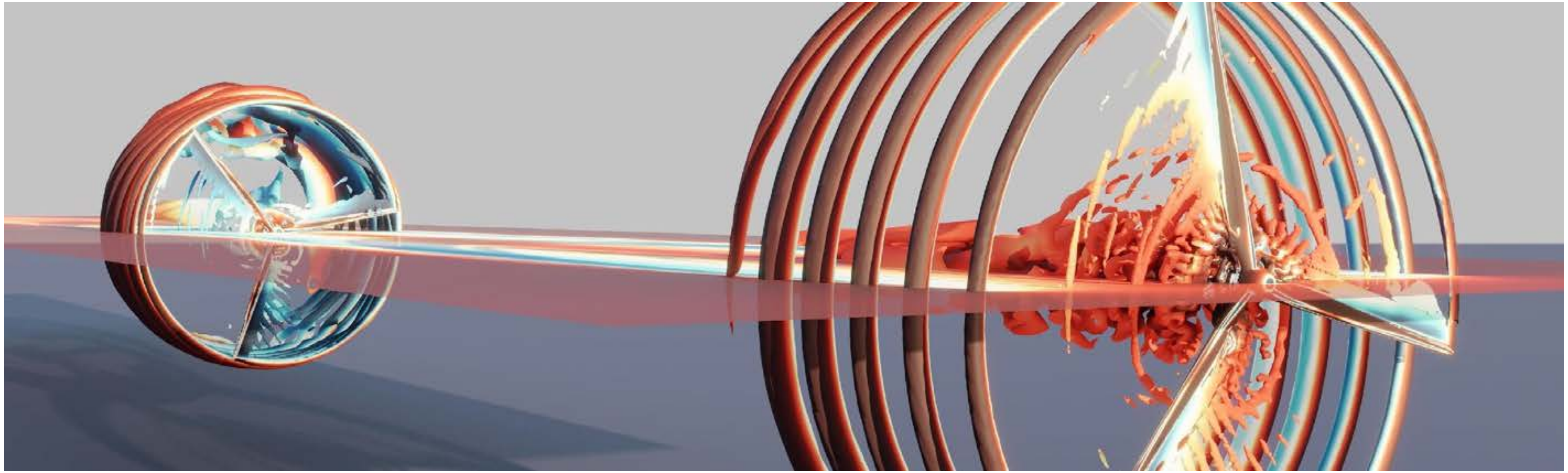


Figure credit: Jon Rood, NREL

ExaWind: Wakes from Three Turbines over Time

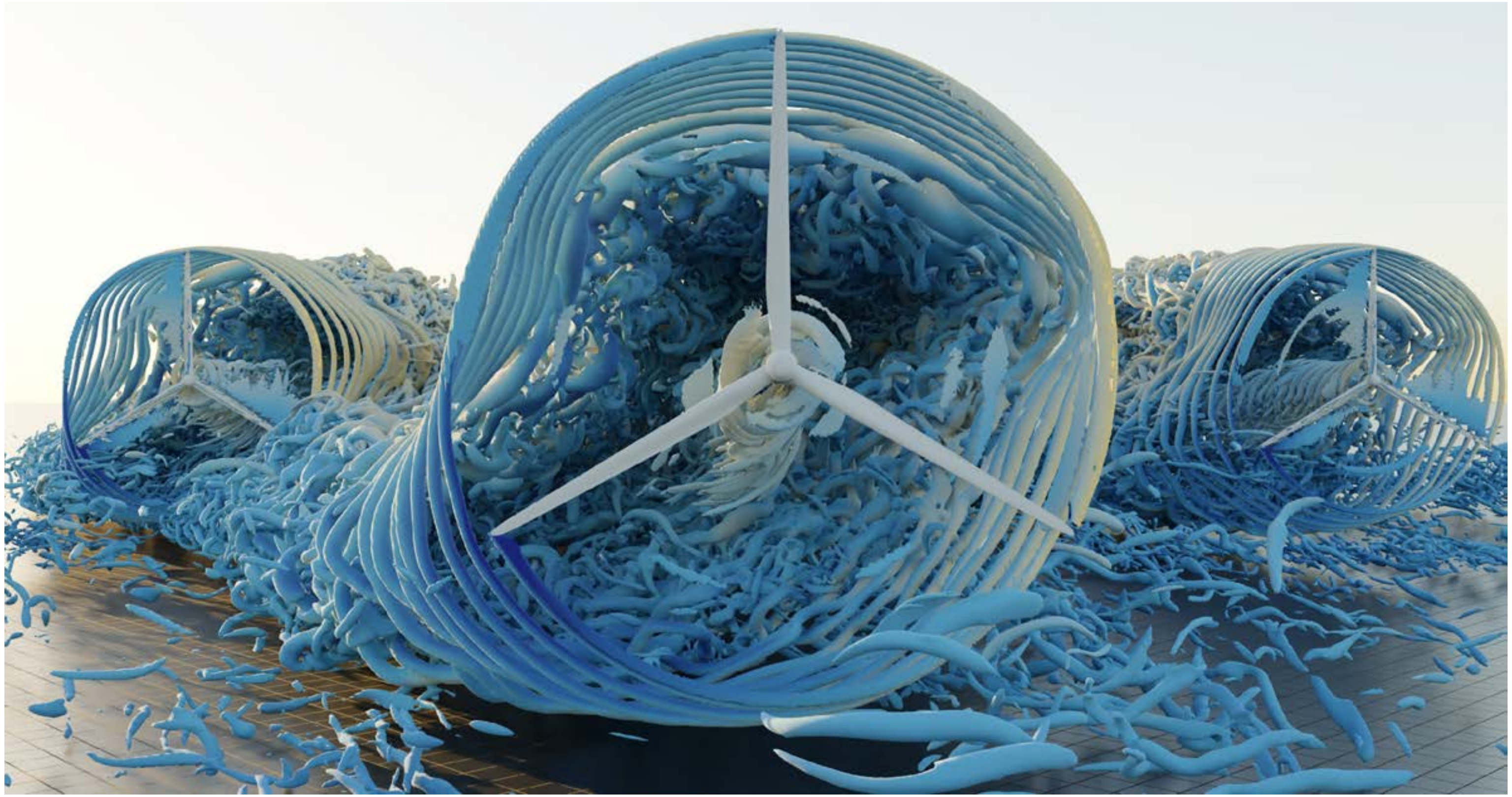


Figure credit: Jon Rood, NREL

ExaWind: Visualization of a Wind Farm Simulation

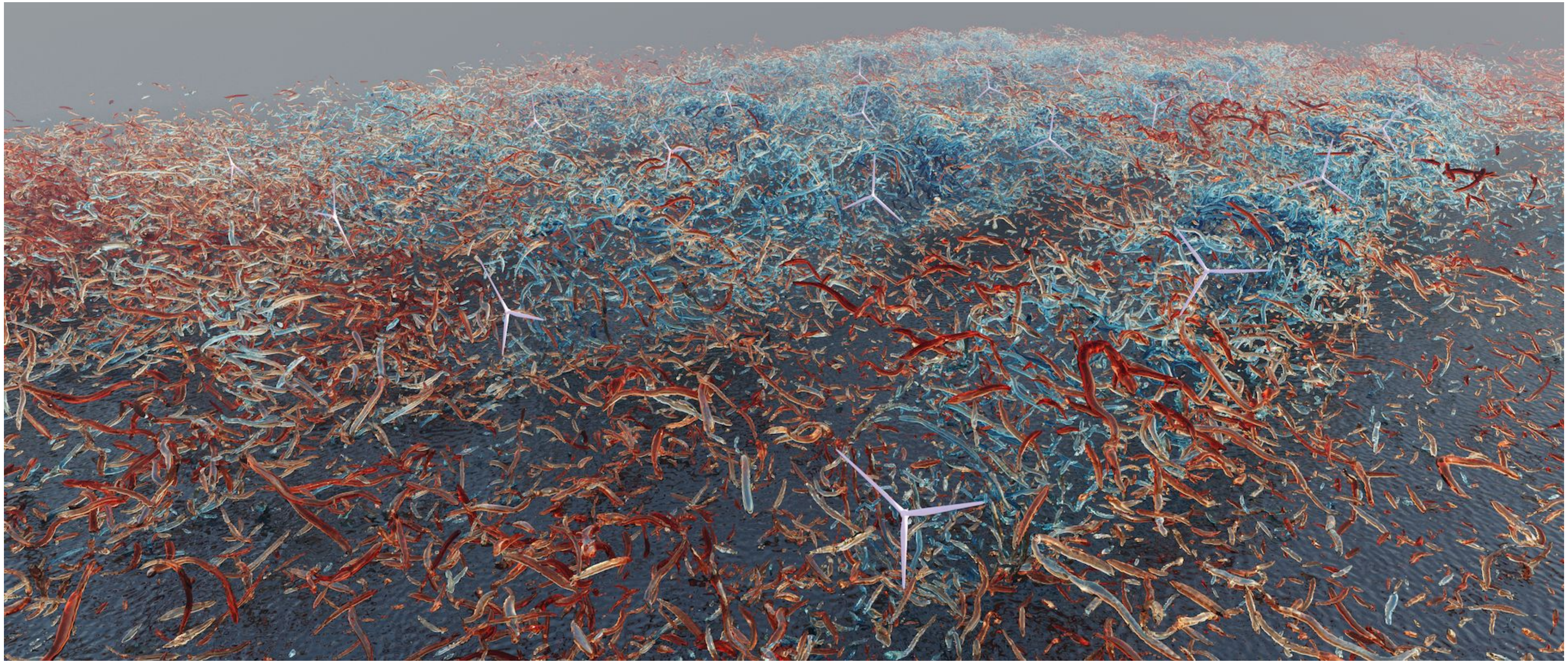
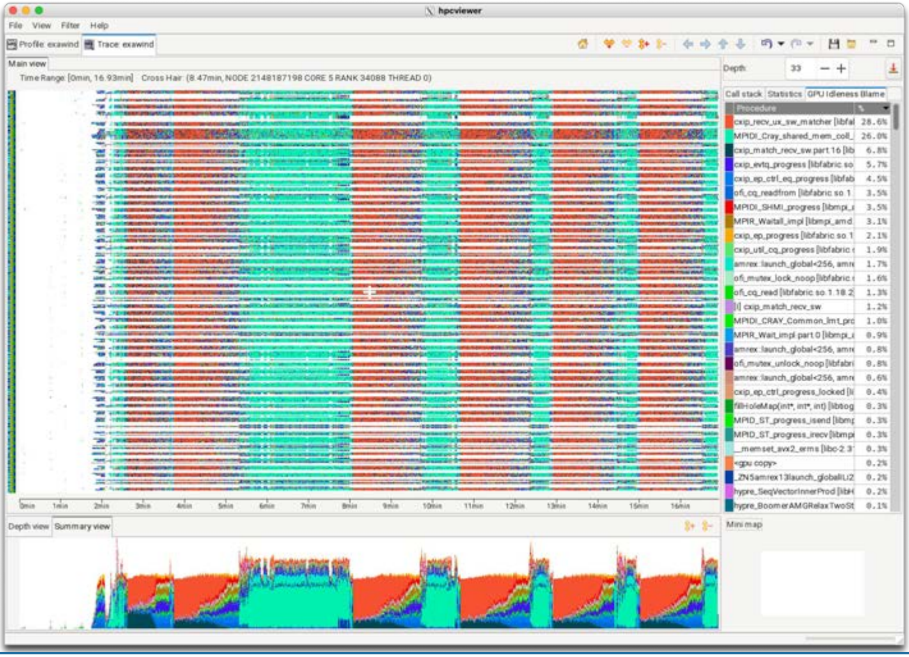


Figure credit: Jon Rood, NREL

ExaWind: Execution Traces on Frontier Collected with HPCToolkit

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

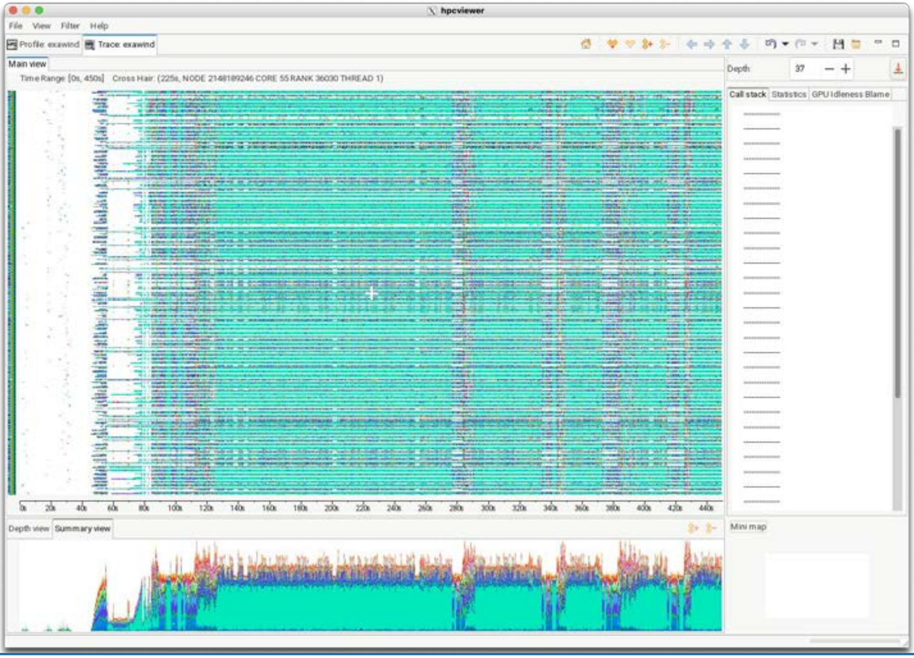
Before: MPI waiting (bad), shown in red



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

	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, amr	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, amr	0.8%
🟪	ofi_mutex_unlock_noop [libfabri	0.8%
🟫	amrex::launch_global<256, amr	0.6%
🟧	cxip_ep_ctrl_progress_locked [li	0.4%
🟩	fillHoleMap(int*, int*, int) [libtog	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%

After: MPI overhead negligible*



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

Figure credits: Jon Rood, NREL

*replaced non-blocking send/recv with ialltoallv

ExaWind Testimonials for HPCToolkit

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

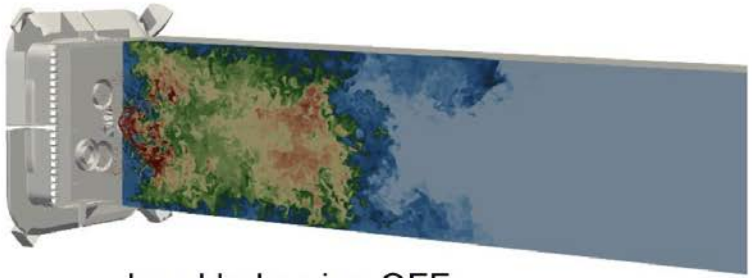
- Jon Rood NREL (5/31/2024)

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

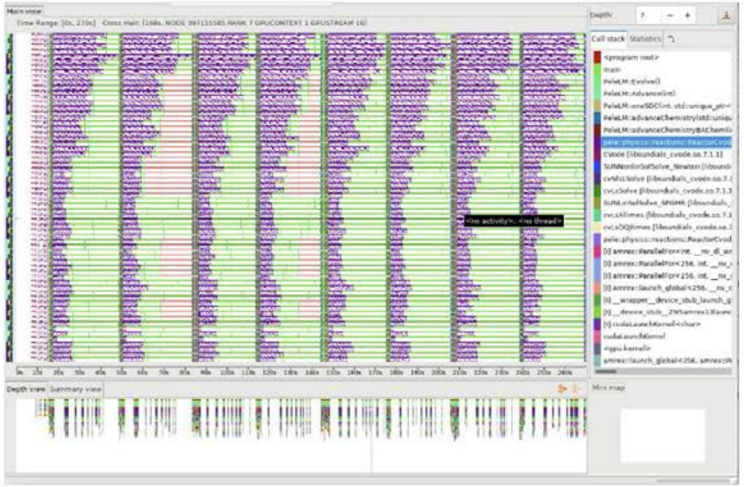
- Jon Rood NREL (6/3/2024)

Full SAF Case: Load Balancing Issues

Confirmed on Frontier production runs: 60% speedup



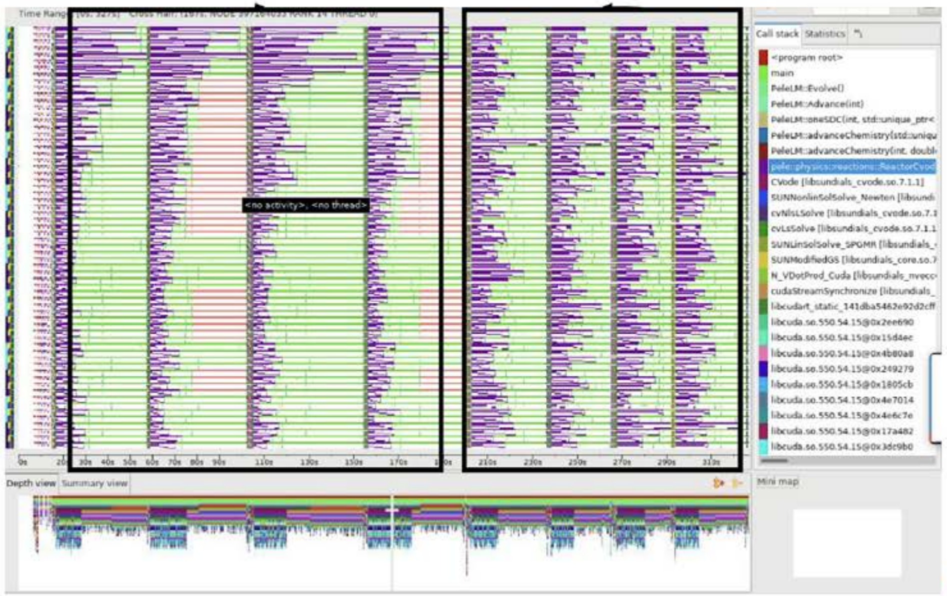
Load balancing OFF
(pink-> Cvode calls)



Load balancing ON

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

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

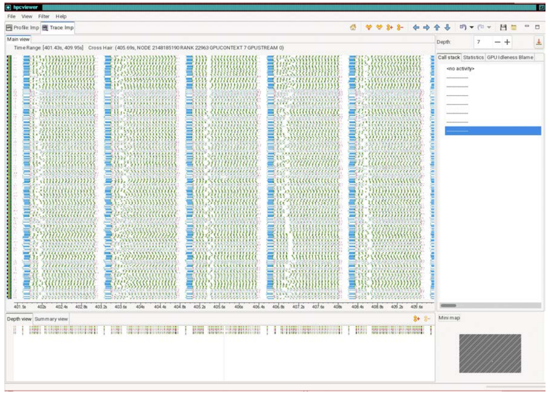


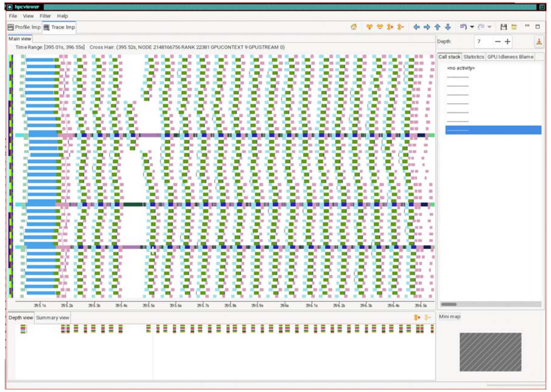
```
<program root>
main
PeleLM::Evolve()
PeleLM::Advance(int)
PeleLM::oneSDC(int, std::unique_ptr<
PeleLM::advanceChemistry(std::uniqu
PeleLM::advanceChemistry(int, doubl
pele::physics::reactions::ReactorCvod
CVode [libsundials_cvode.so.7.1.1]
SUNNonlinSolSolve_Newton [libsundi
cvNlsLSolve [libsundials_cvode.so.7.1
cvLsSolve [libsundials_cvode.so.7.1.1
SUNLinSolSolve_SPGMR [libsundials_
SUNModifiedGS [libsundials_core.so.7
N_VDotProd_Cuda [libsundials_nvecc
cudaStreamSynchronize [libsundials_
libcudart_static_141dba5462e92d2cff
libcuda.so.550.54.15@0x2ee690
libcuda.so.550.54.15@0x15d4ec
libcuda.so.550.54.15@0x4b80a8
libcuda.so.550.54.15@0x249279
libcuda.so.550.54.15@0x1805cb
libcuda.so.550.54.15@0x4e7014
libcuda.so.550.54.15@0x4e6c7e
libcuda.so.550.54.15@0x17a482
libcuda.so.550.54.15@0x3dc9b0
```

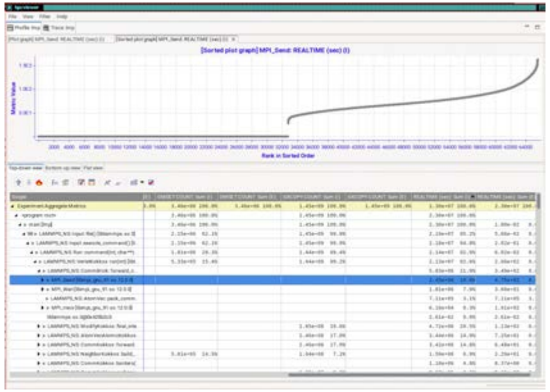
OpenACC   OPEN HACKATHONS

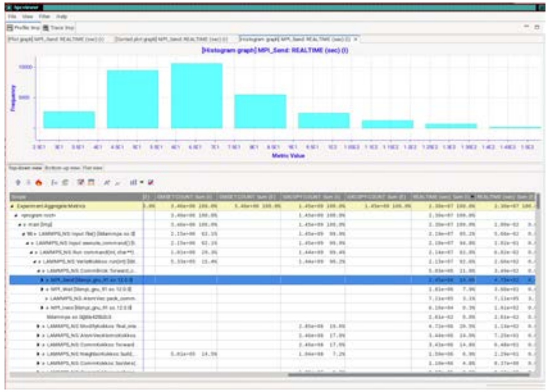
Figure credit: PeleLMEX Team, NREL Hackathon, February 2025

LAMMPS on Frontier: Executions with Kernel Duration of Milliseconds



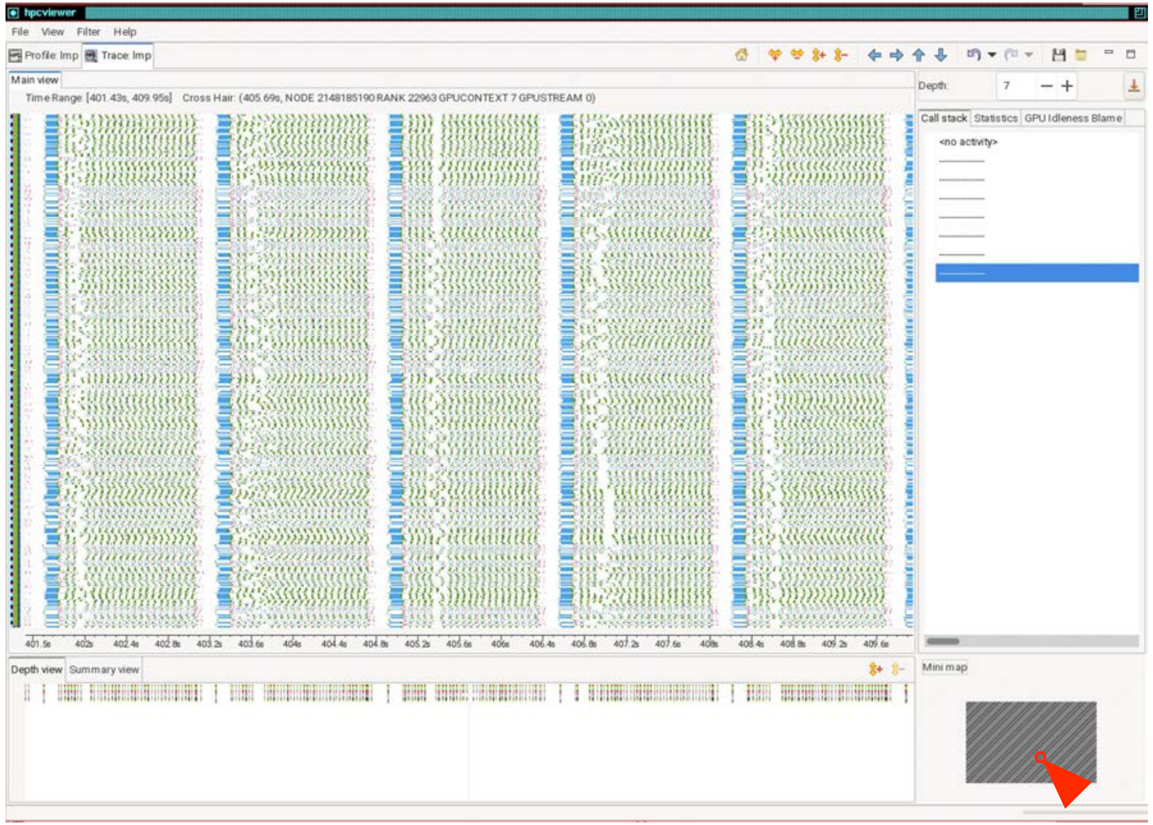






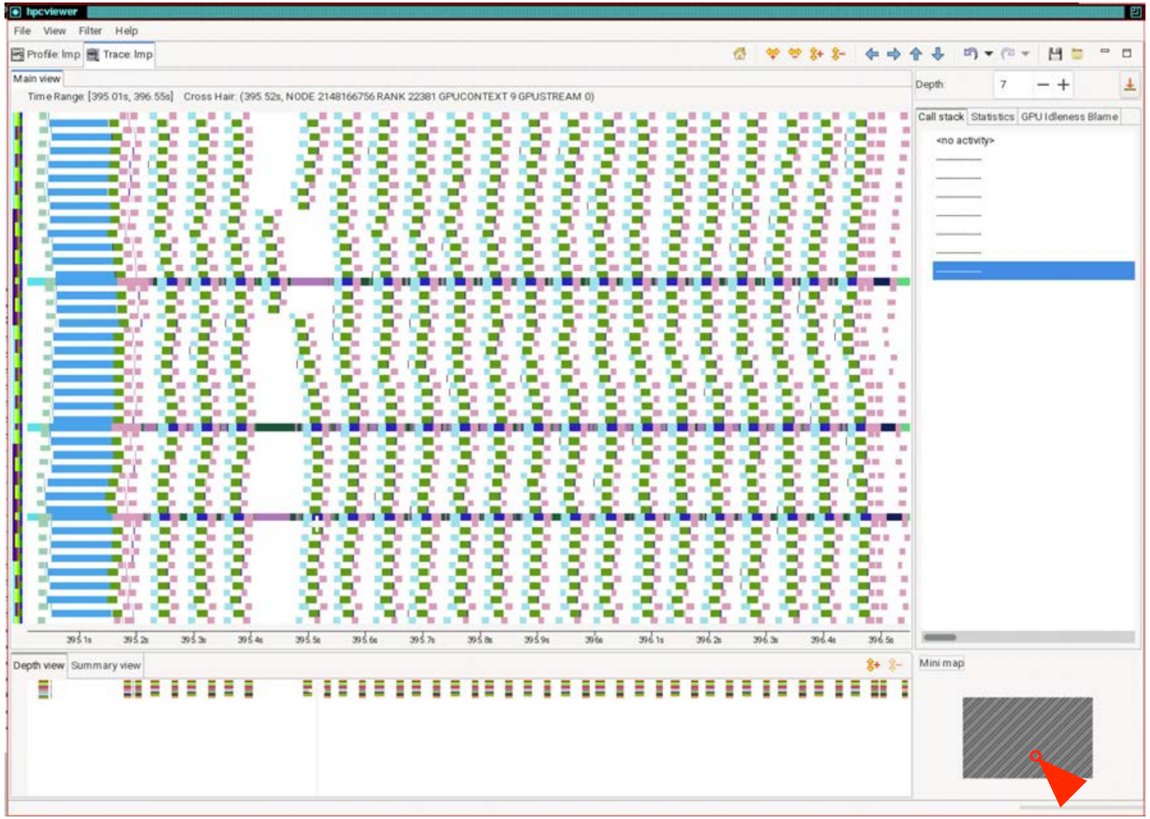
8K nodes, 64K MPI ranks + 64K GPU tiles

LAMMPS on Frontier: Executions with Kernel Duration of Milliseconds



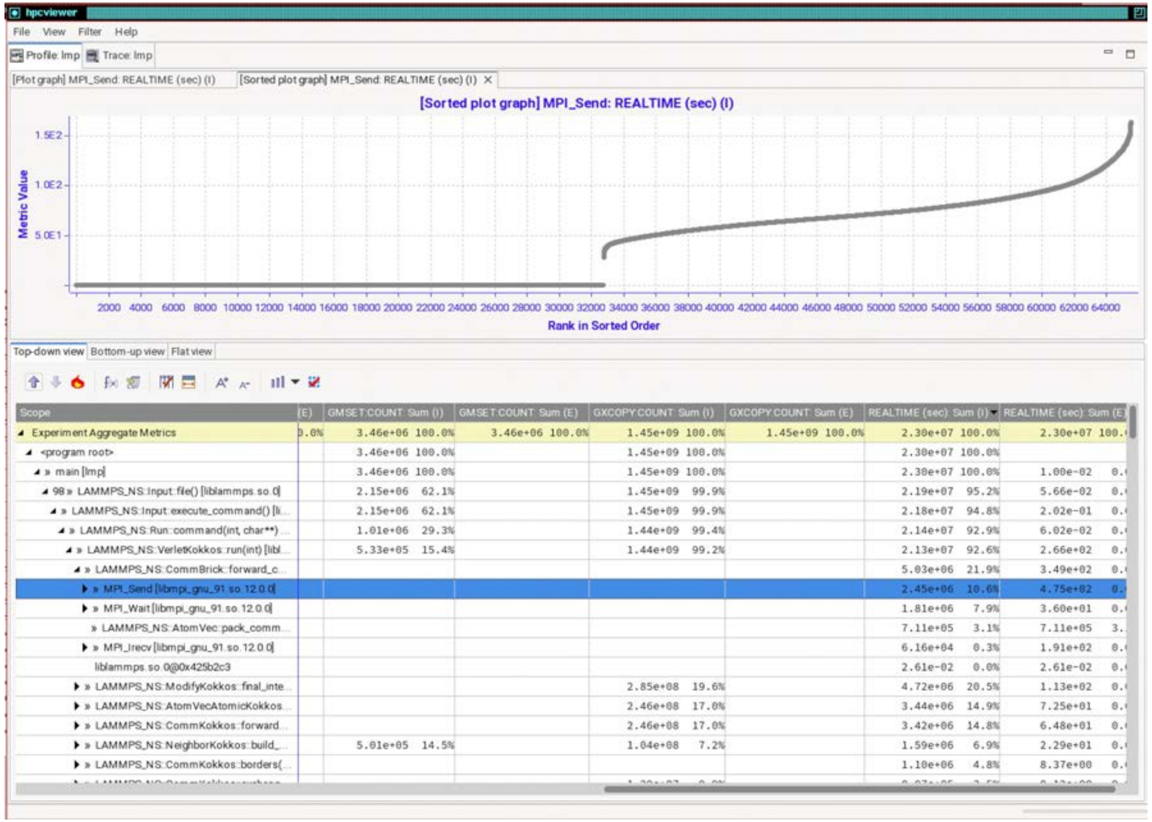
Time Range: [401.43s, 409.95s] Cross Hair: (405.69s, NODE 2148185190 RANK 22963 GPUCONTEXT 7 GPUSTREAM 0)

LAMMPS on Frontier: Executions with Kernel Duration of Milliseconds



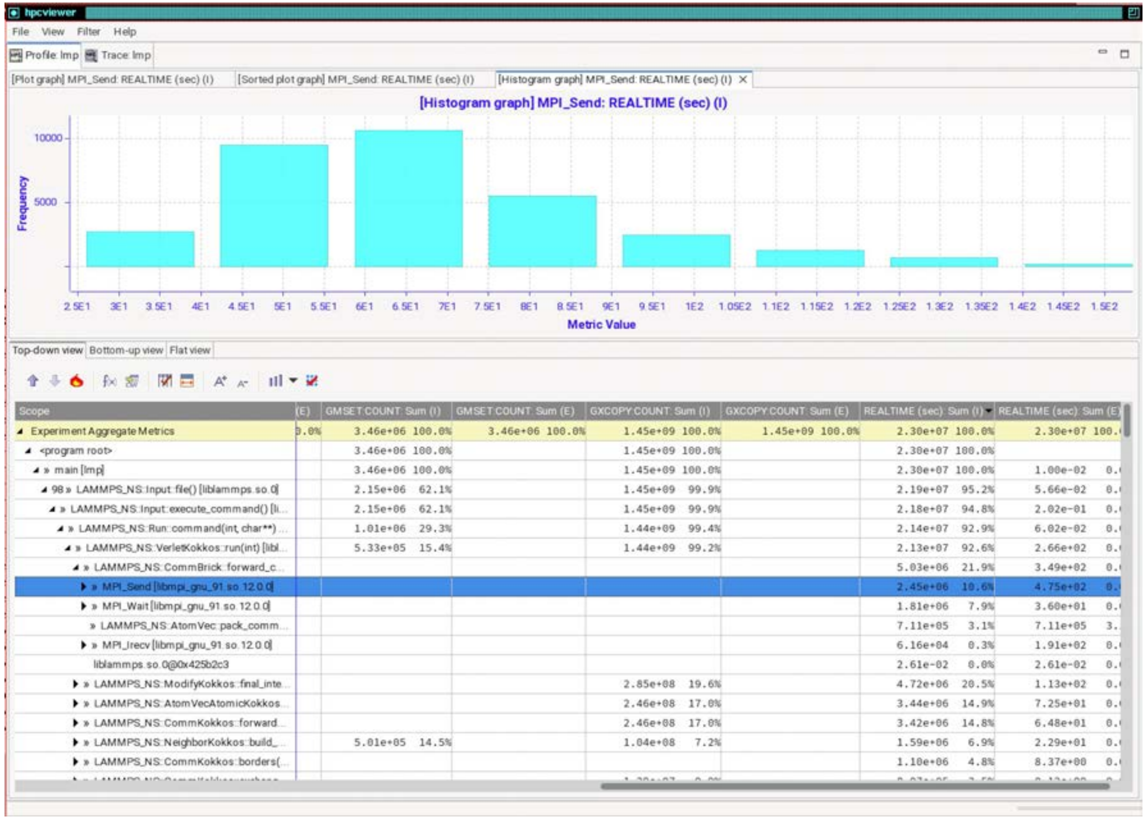
Time Range: [395.01s, 396.55s]   Cross Hair: (395.52s, NODE 2148166756 RANK 22381 GPUCONTEXT 9 GPUSTREAM 0)

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%	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.

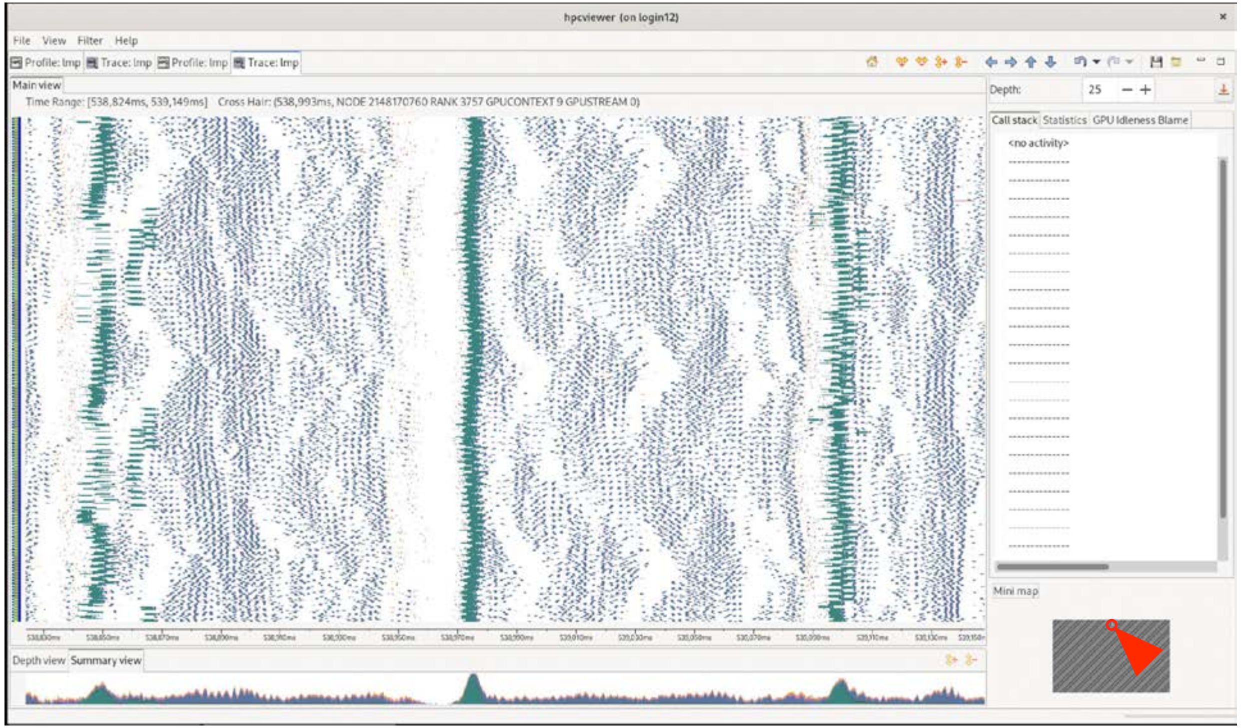
LAMMPS on Frontier: Executions with Kernel Duration of Milliseconds


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%	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)

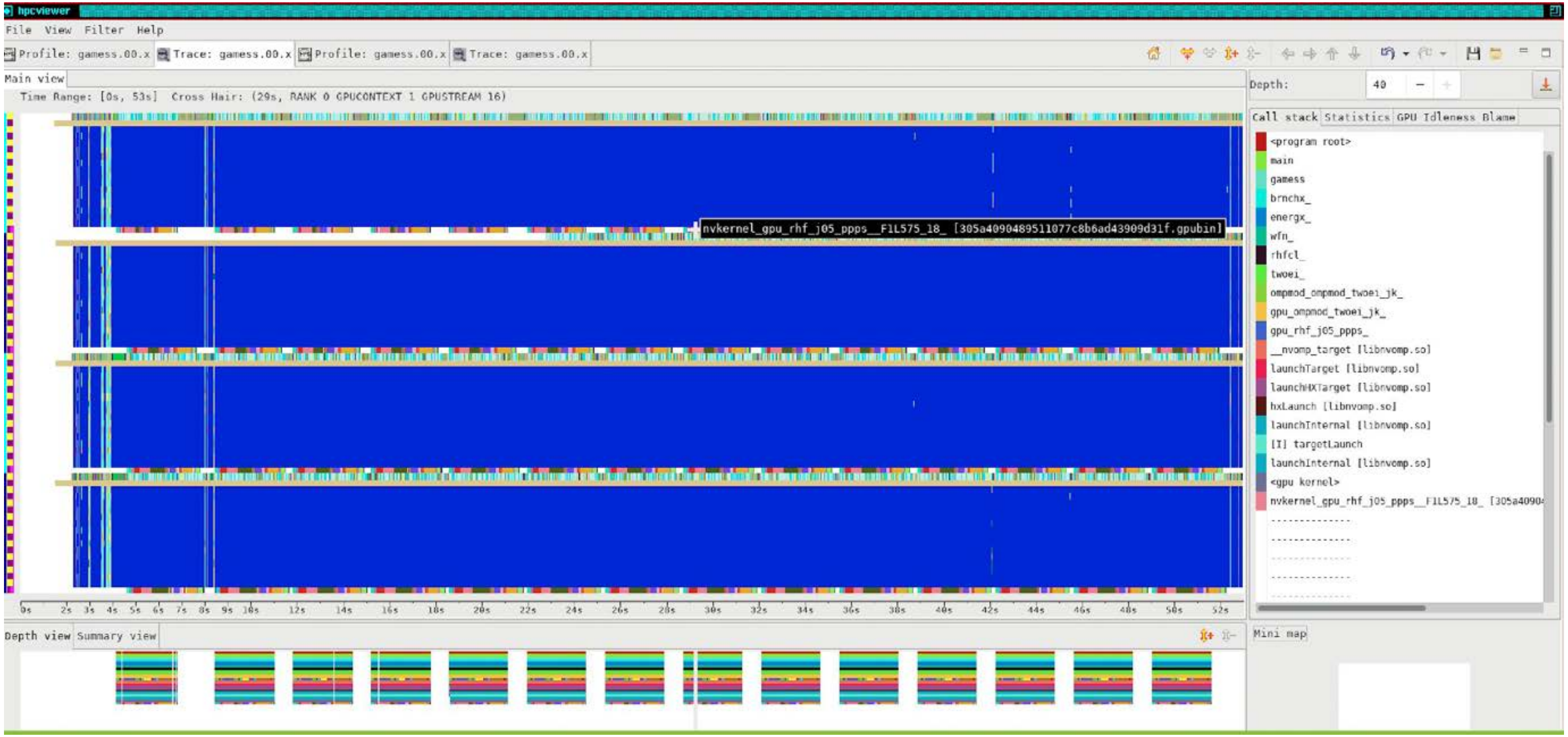
Case Study: GAMESS

- 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]
```

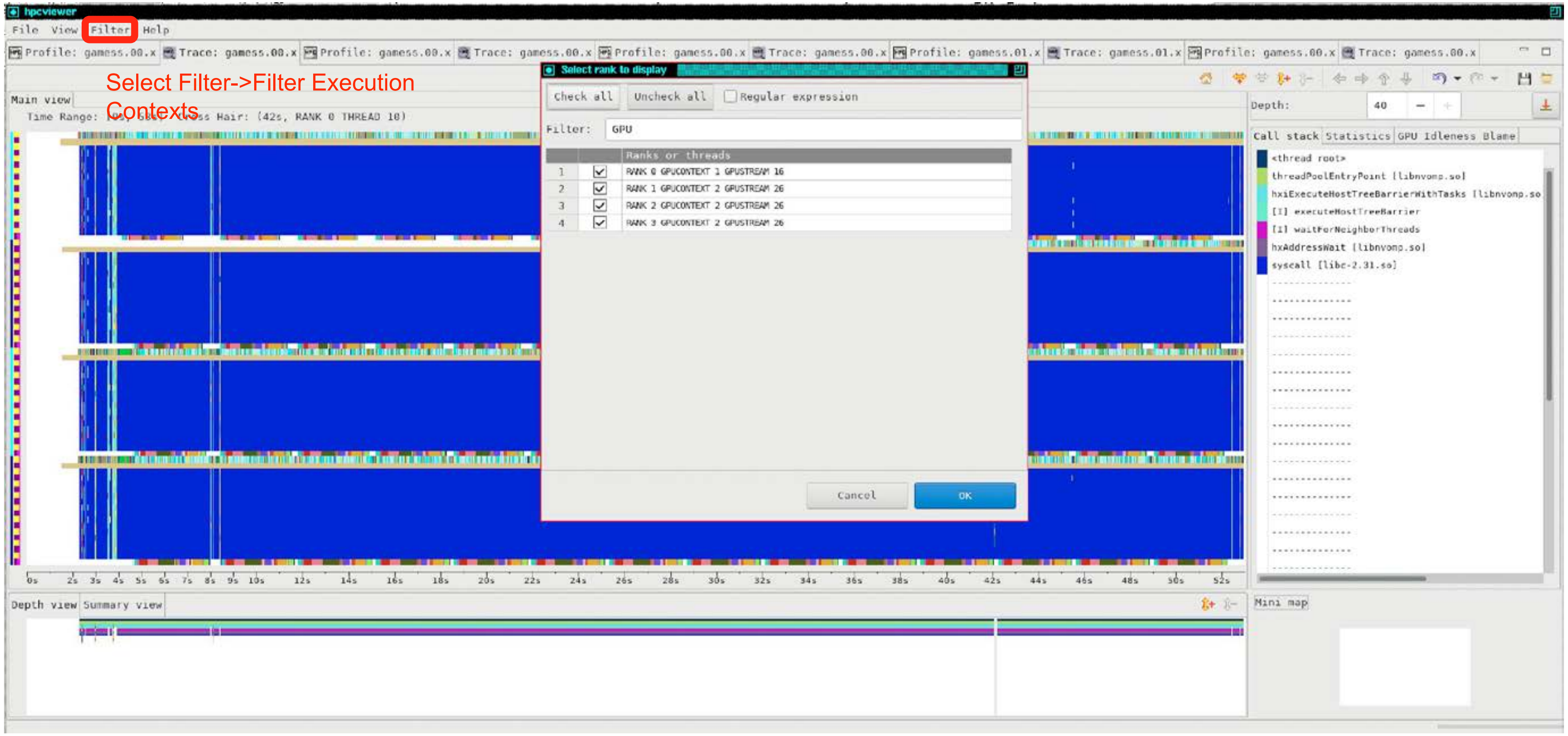
Call stack:

```
<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



Select Filter->Filter Execution Contexts

Time Range: [illegible] Cross Hair: (42s, RANK 0 THREAD 10)

Select rank to display

Check all | Uncheck all | Regular expression

Filter: GPU

		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

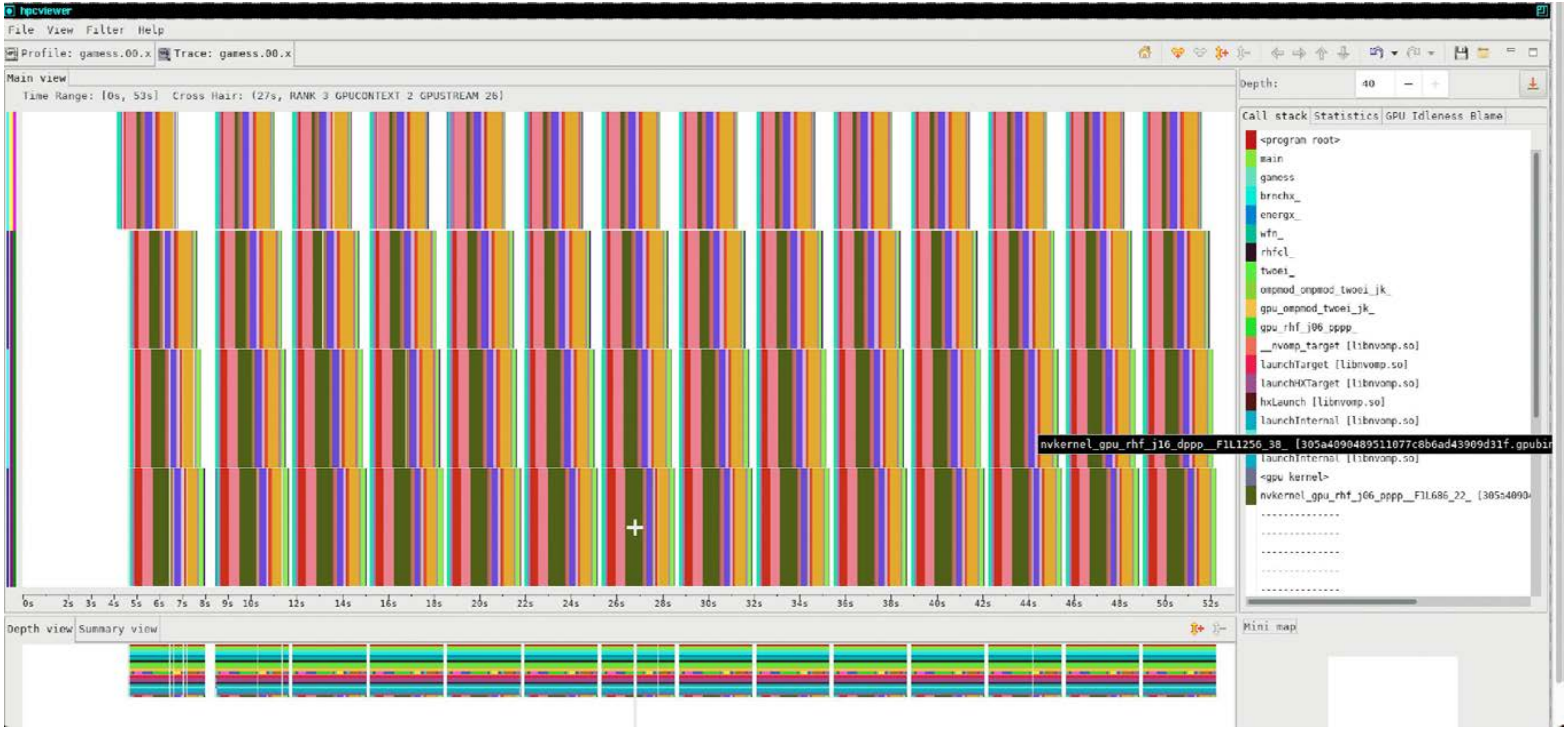
Call stack:

```
<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)
```

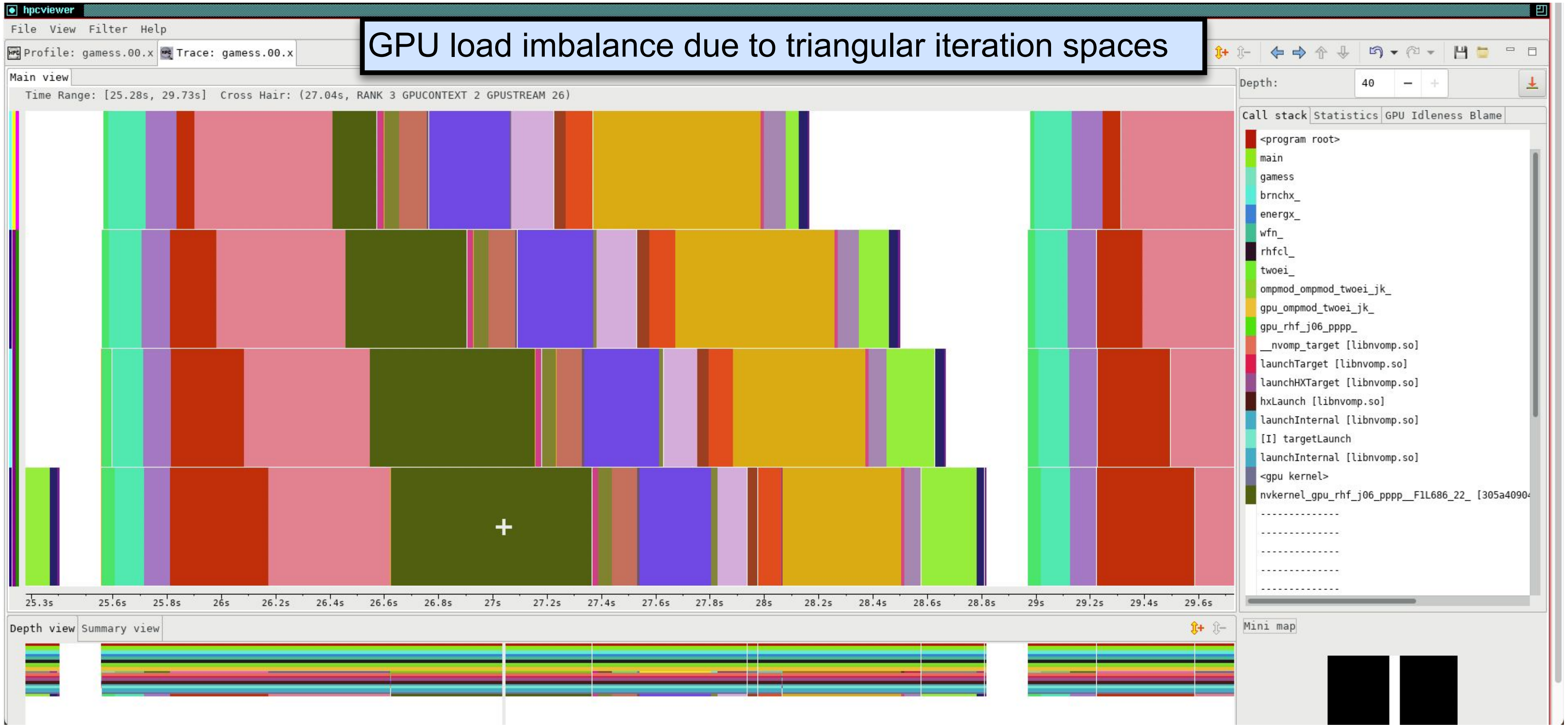
```
<program root>
main
gamess
brnchx_
energx_
wfn_
rhfcl_
twoei_
ompmod_ompmod_twoei_jk_
gpu_ompmod_twoei_jk_
gpu_rhf_j06_pppp_
__nvomp_target [libnvomp.so]
launchTarget [libnvomp.so]
launchHXTarget [libnvomp.so]
hxLaunch [libnvomp.so]
launchInternal [libnvomp.so]
launchInternal [libnvomp.so]
<gpu kernel>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

```
nvkernel_gpu_rhf_j16_dppp__F1L1256_38_ [305a4090489511077c8b6ad43909d31f.gpubin
```

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

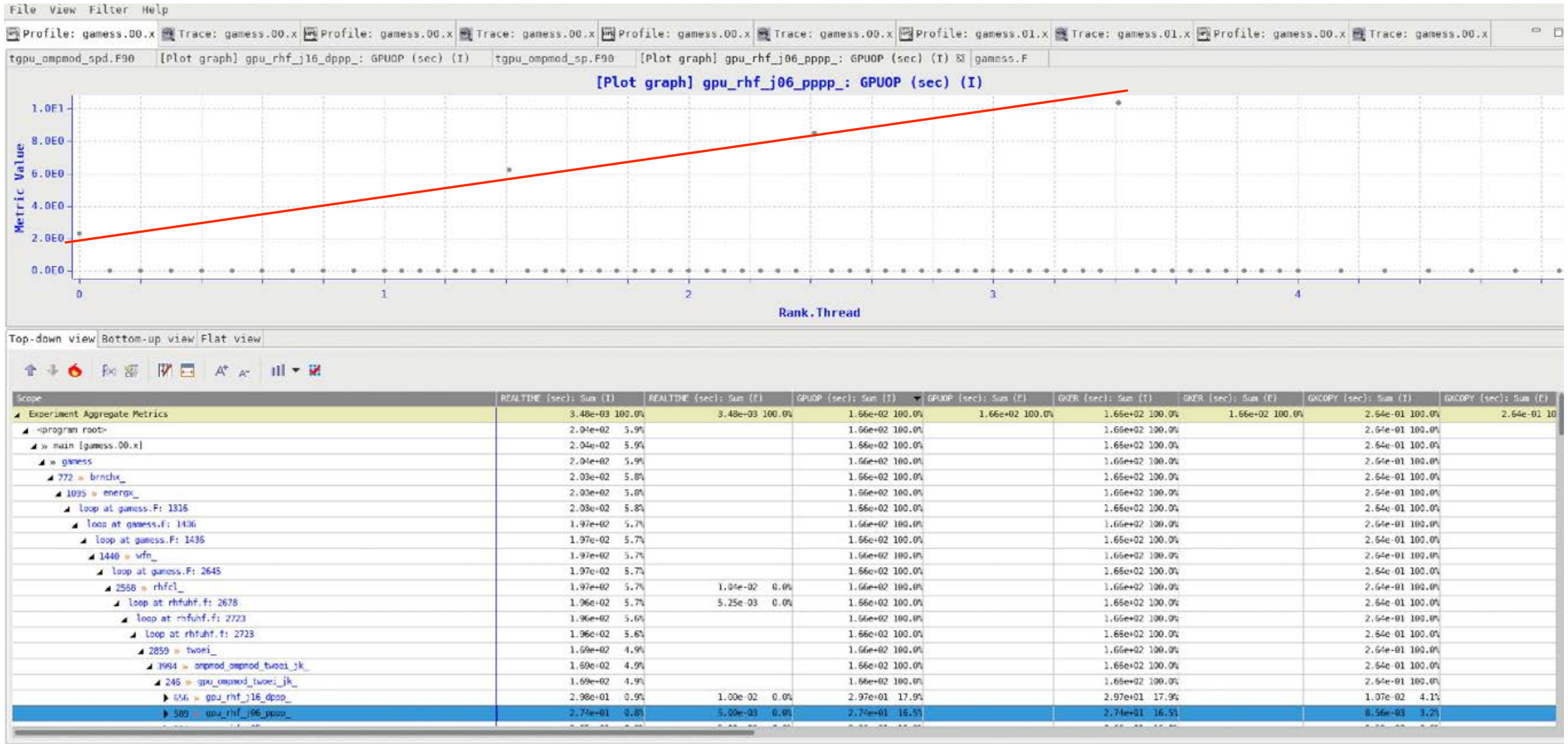
Time Range: [25.28s, 29.73s] Cross Hair: (27.04s, RANK 3 GPUCONTEXT 2 GPUSTREAM 26)

```
<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

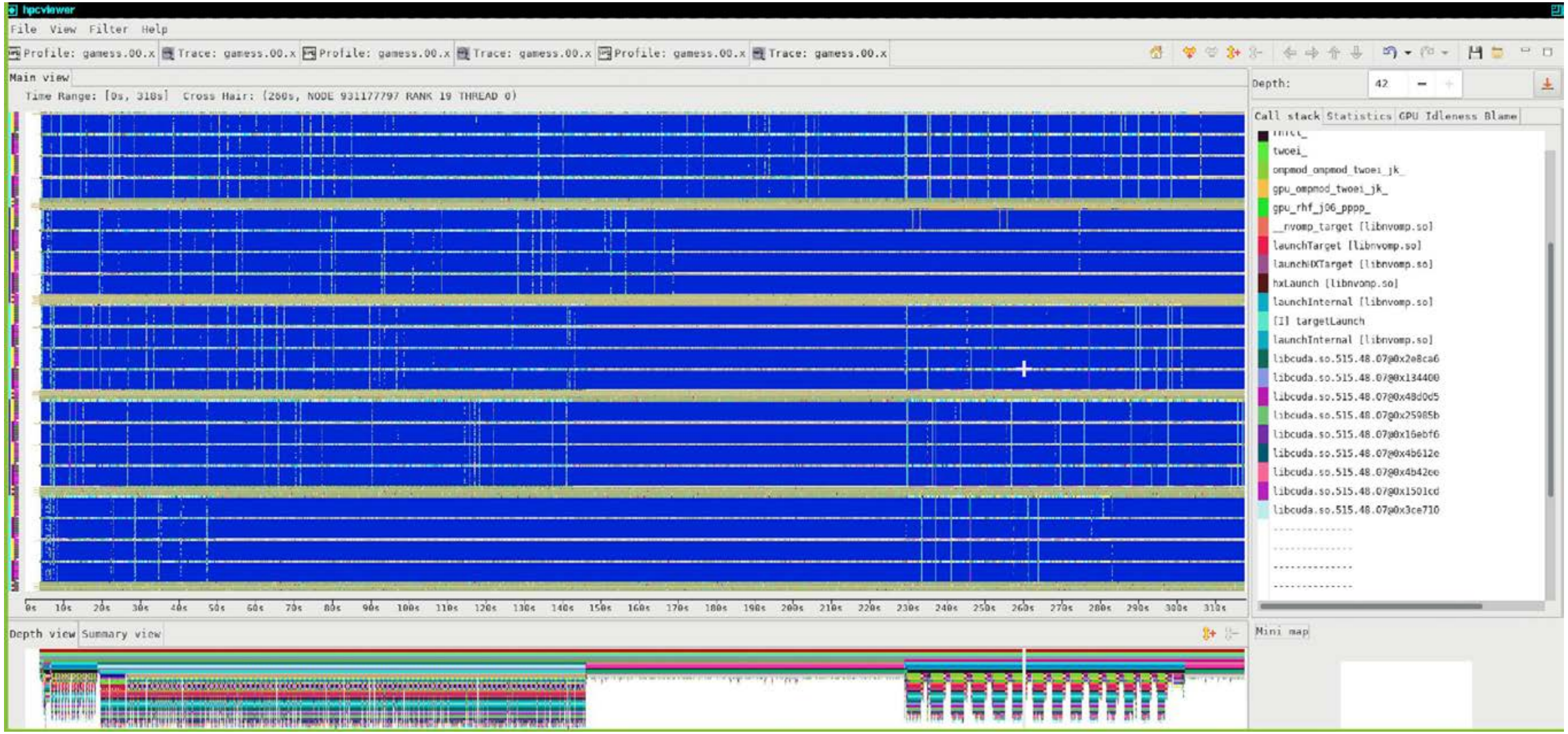
Time-centric Analysis: GAMESS 4 ranks, 4 GPUs on Perlmutter



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%		1.65e+02 100.0%		2.64e-01 100.0%	
» main [gamess.00.x]	2.04e+02 5.9%		1.66e+02 100.0%		1.65e+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%		1.65e+02 100.0%		2.64e-01 100.0%	
1035 » energx_	2.03e+02 5.8%		1.66e+02 100.0%		1.65e+02 100.0%		2.64e-01 100.0%	
loop at gamess.F: 1316	2.03e+02 5.8%		1.66e+02 100.0%		1.65e+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%		1.65e+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%		1.65e+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%		1.65e+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%		1.65e+02 100.0%		2.64e-01 100.0%	
246 » gpu_ompmod_twoei_jk_	1.69e+02 4.9%		1.66e+02 100.0%		1.65e+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

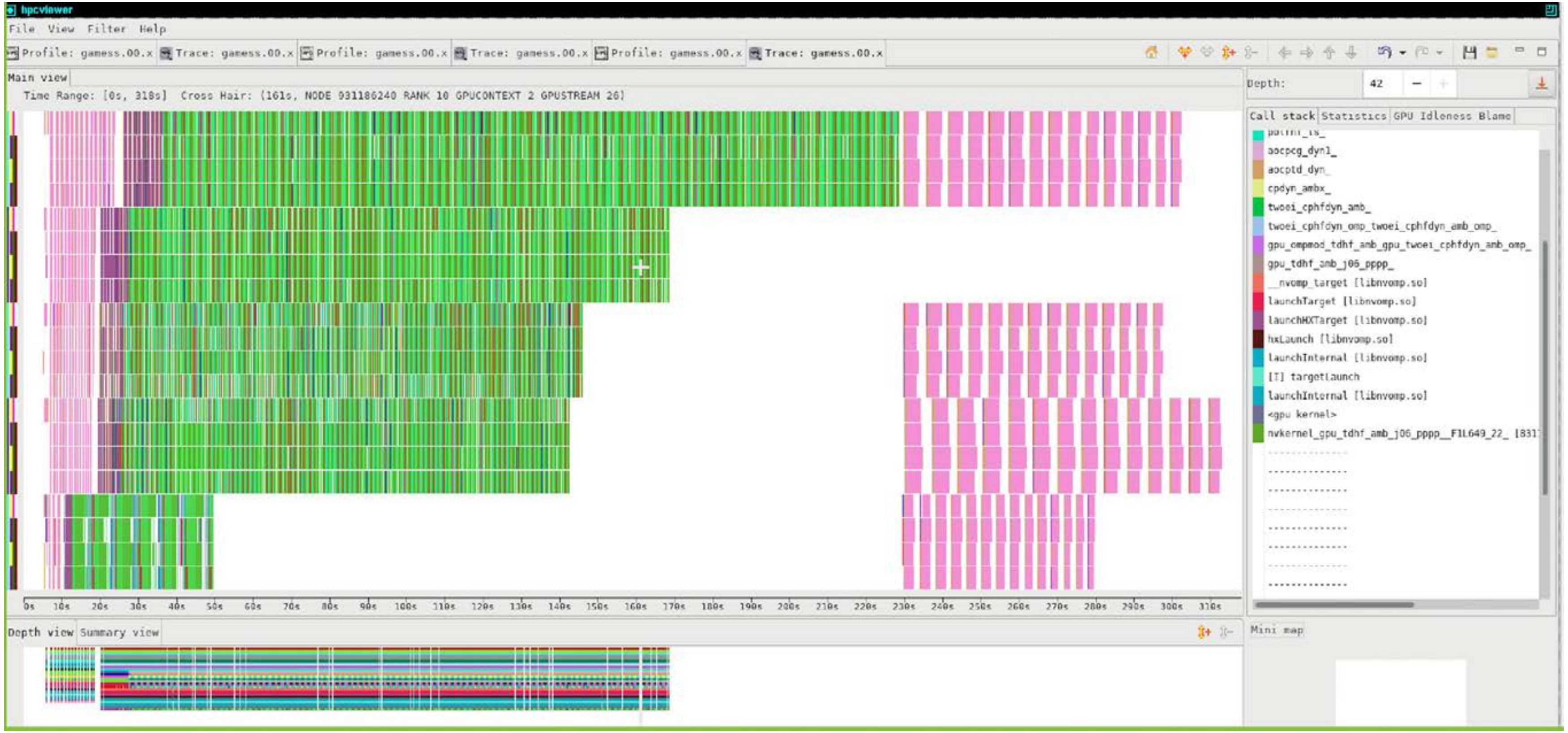


```
Time Range: [0s, 318s]  Cross Hair: (260s, NODE 931177797 RANK 19 THREAD 0)
```

```
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
--------------
--------------
--------------
--------------
```

CPU Threads and GPU Streams

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


hpcviewer

File View Filter Help

Profile: gamess.00.x | Trace: gamess.00.x | Profile: gamess.00.x | Trace: gamess.00.x | Profile: gamess.00.x | Trace: gamess.00.x

Main view

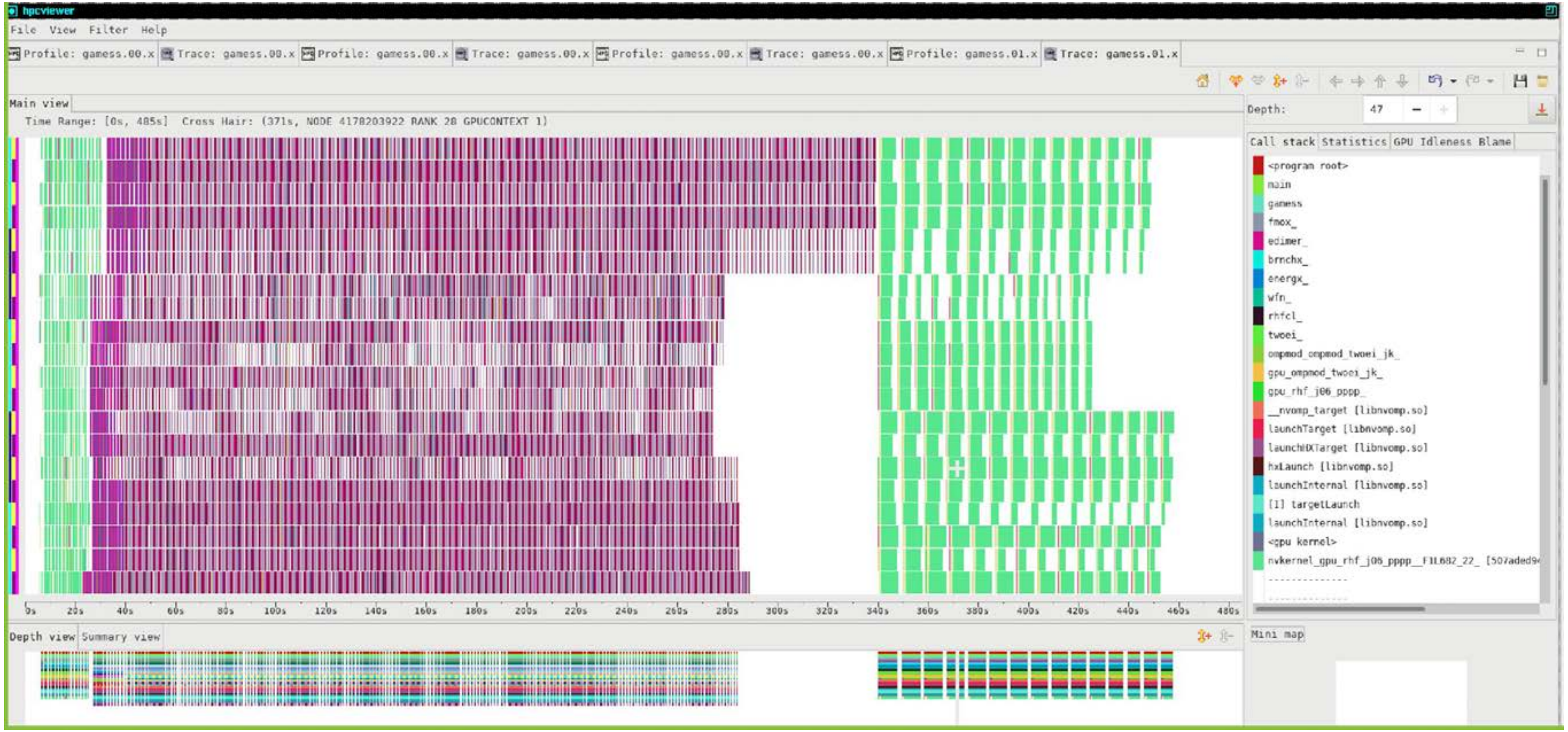
Time Range: [0s, 318s] Cross Hair: (161s, NODE 931186240 RANK 10 GPUCONTEXT 2 GPUSTREAM 26)

Depth: 42

Call stack | Statistics | GPU Idleness Blame

🟩	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

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



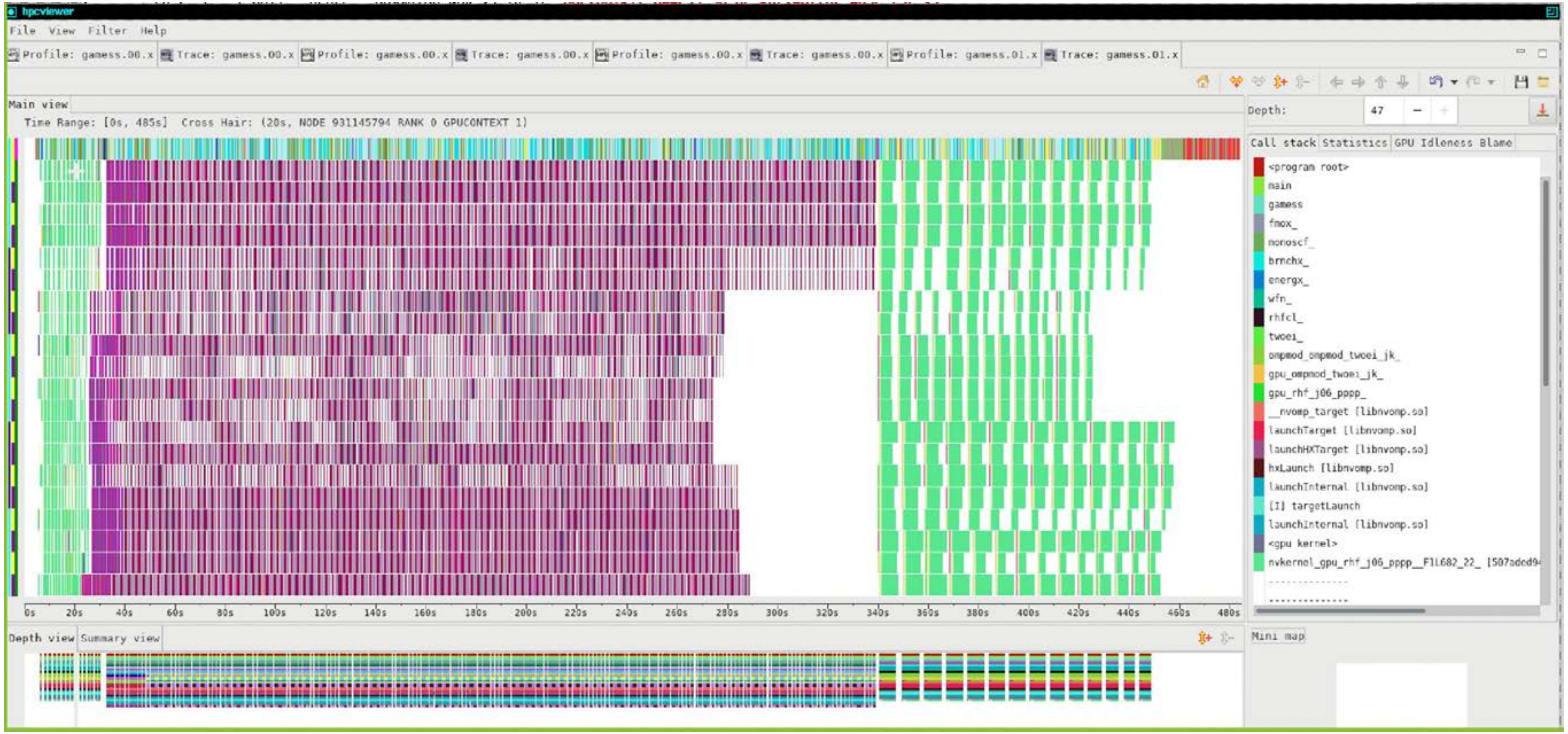
Time Range: [0s, 485s] Cross Hair: (371s, NODE 4178203922 RANK 28 GPUCONTEXT 1)

Depth: 47

```
<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

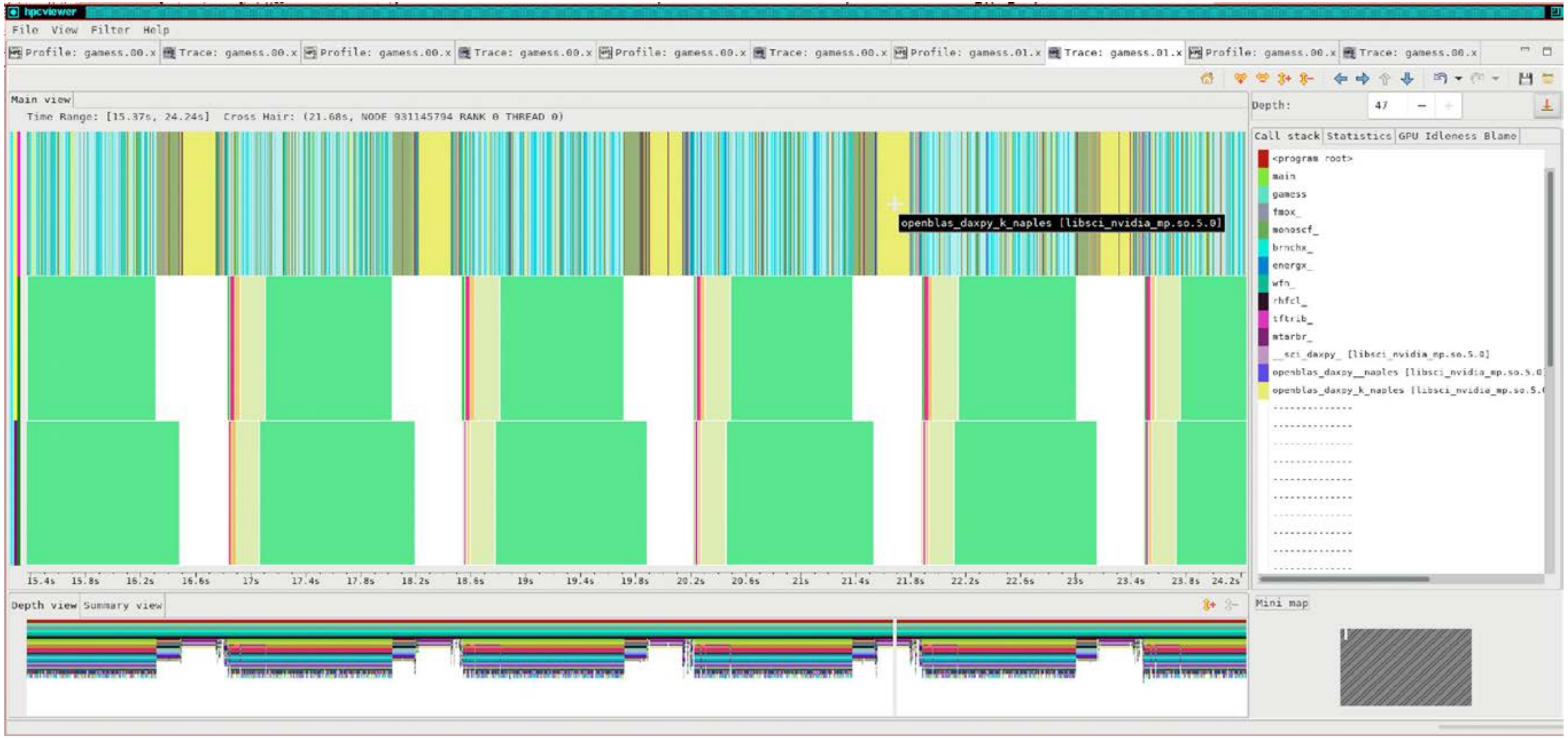


Time Range: [0s, 485s] Cross Hair: (20s, NODE 931145794 RANK 0 GPUCONTEXT 1)

```
<program root>
main
gamess
fmox_
nonoscf_
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



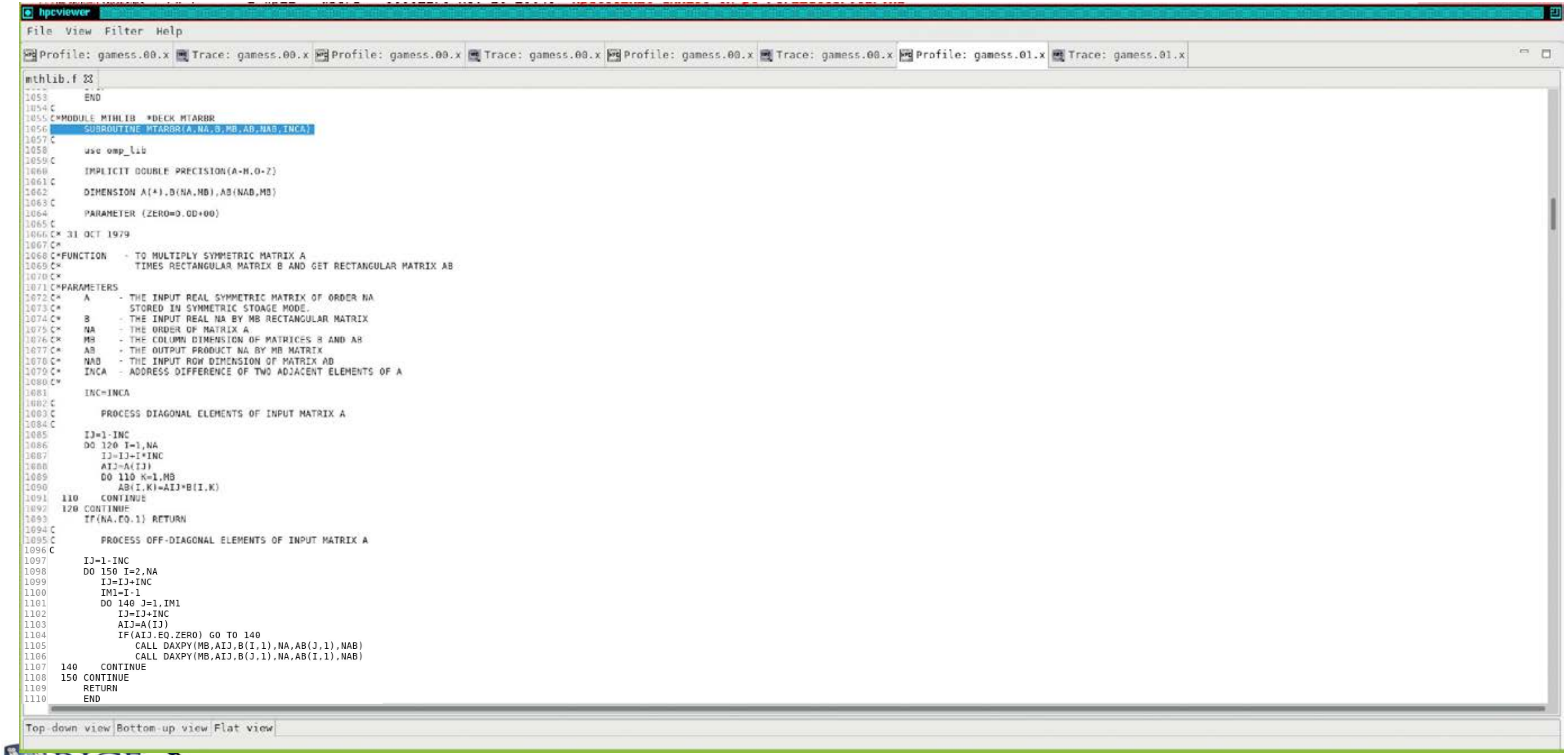
Time Range: [15.37s, 24.24s] Cross Hair: (21.68s, NODE 931145794 RANK 0 THREAD 0)

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.5.0]

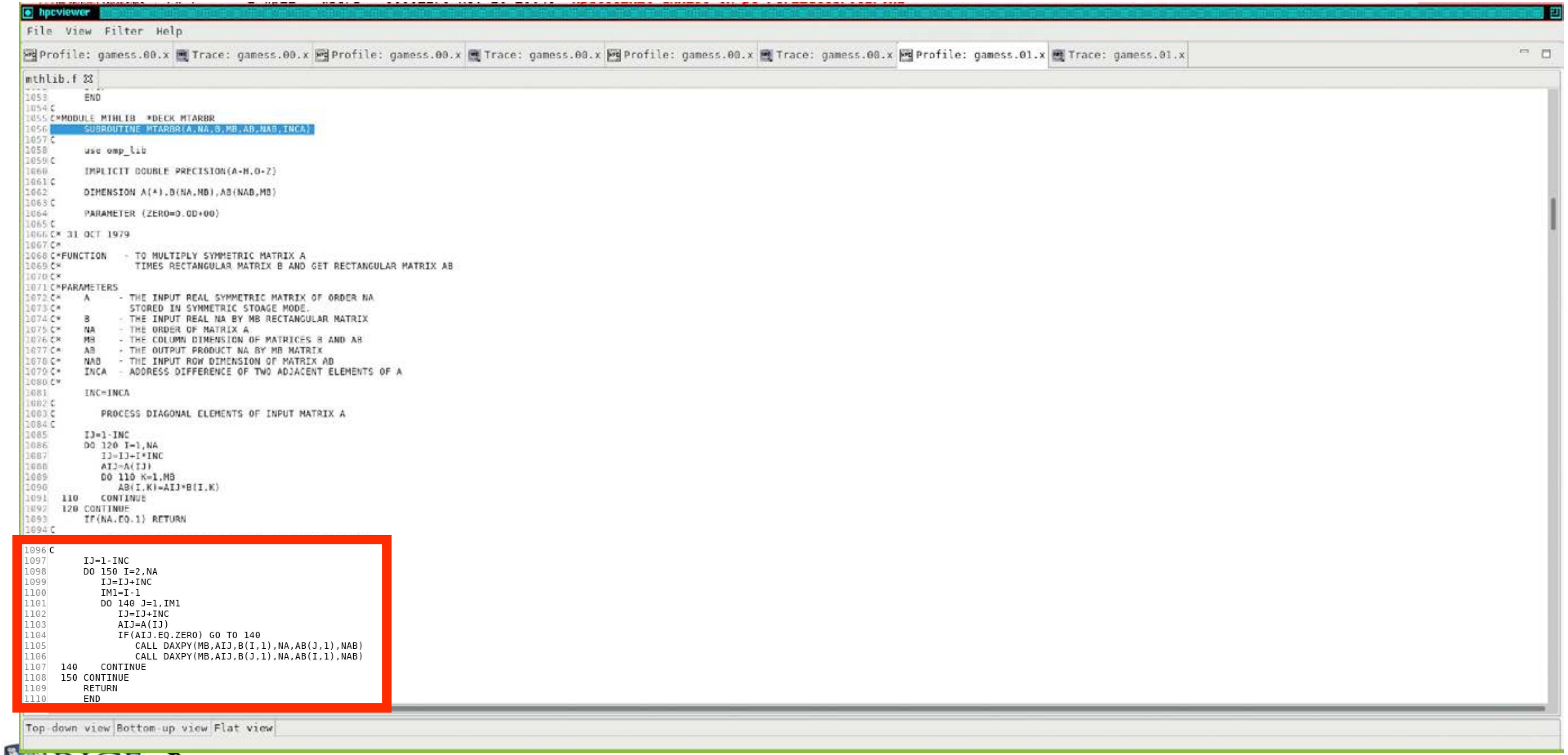
1 CPU Stream, 2 GPU Streams: 6 Iterations

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



```fortran
      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
```

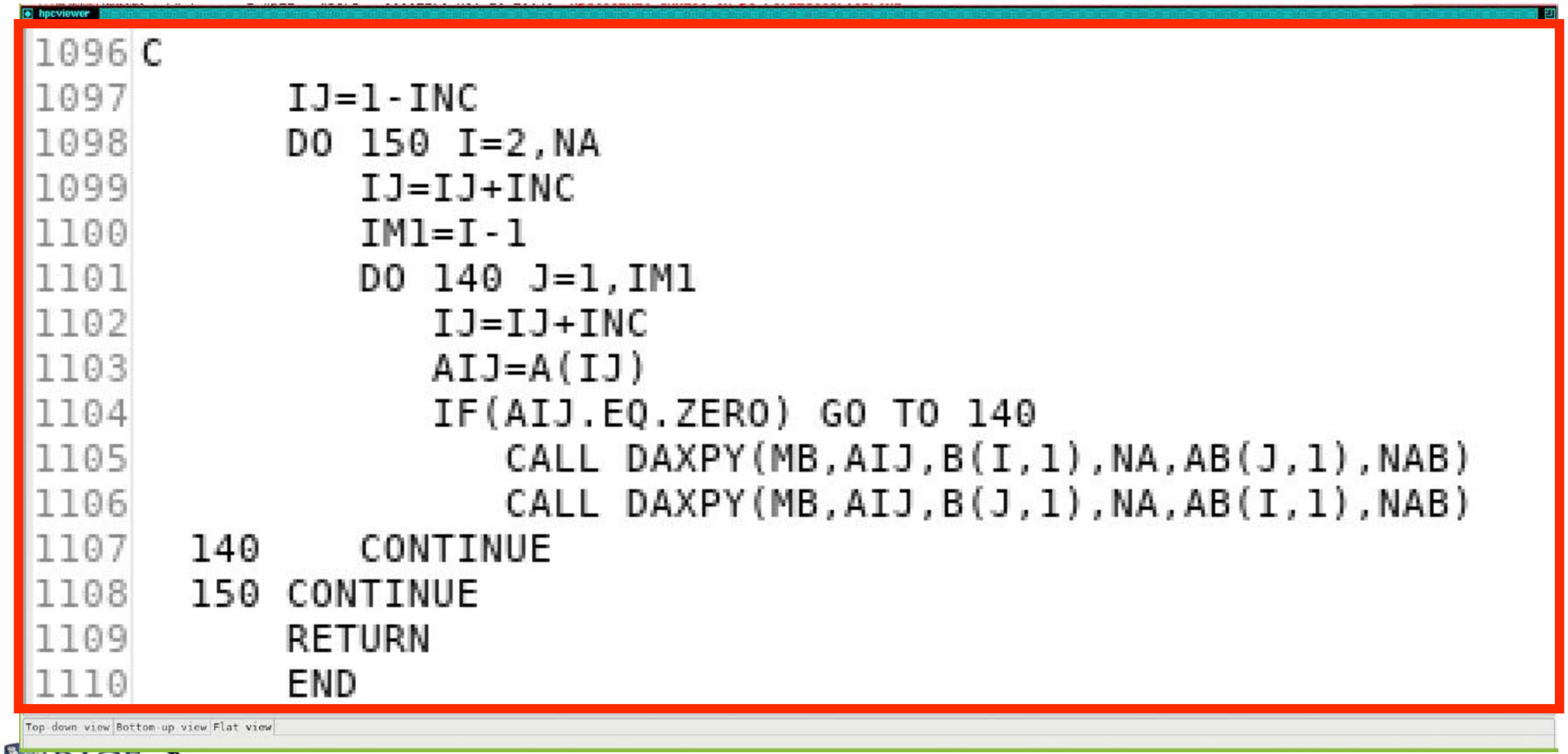
Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter



```
      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
      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
```

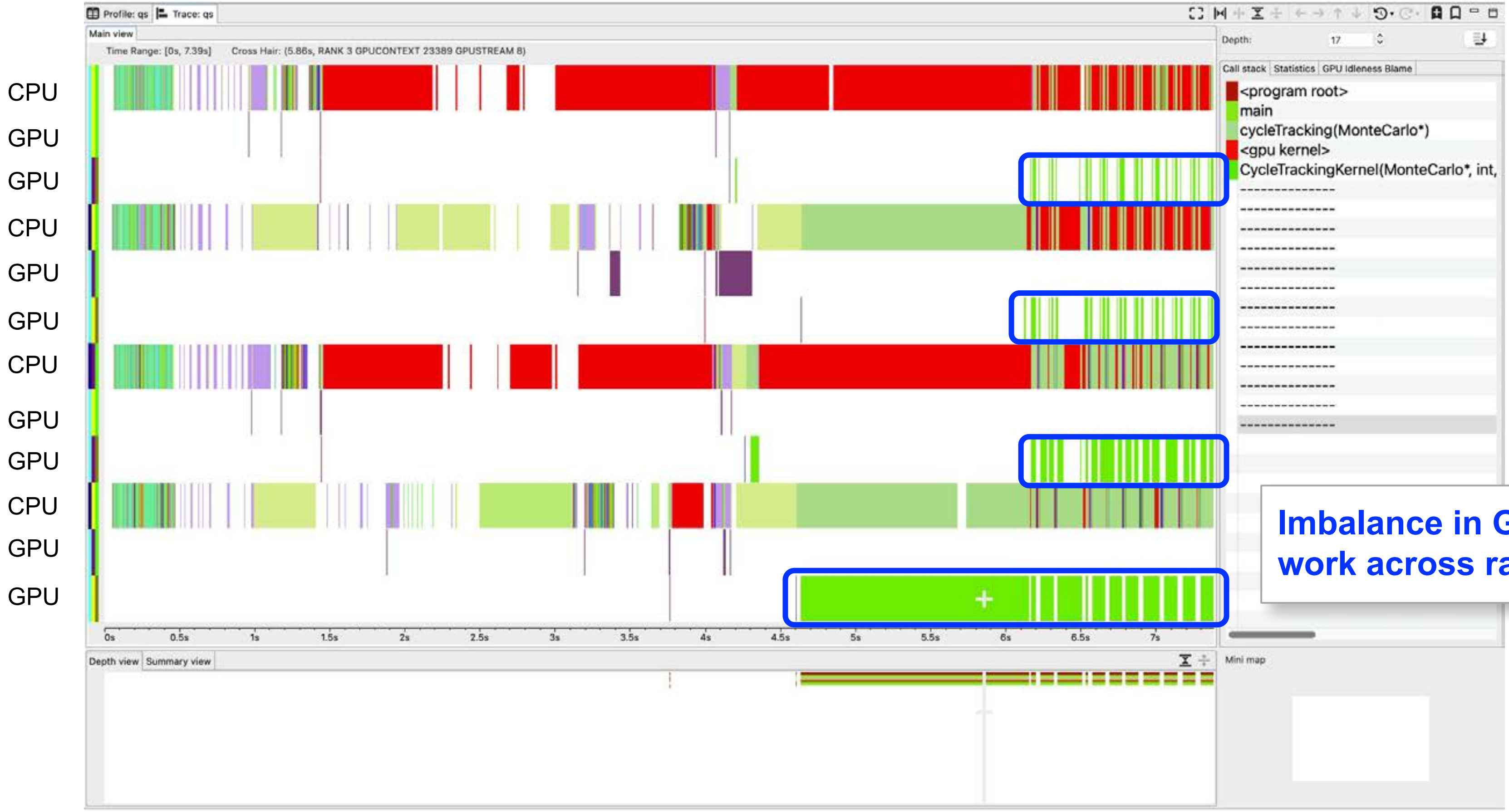
Today’s Agenda

- Introduce HPCToolkit tools and workflow
- Illustrate HPCToolkit’s use with some case studies
- Live demos
  - profiling and tracing
  - instruction-based performance metrics
- Instruction-level performance monitoring
  - explaining instruction-level performance
  - improving analysis and attribution of PC samples
  - automating analysis of GPU bottlenecks
- Discussion about needs, problems, and suggestions

Case Study: Quicksilver

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

Quicksilver: Trace view

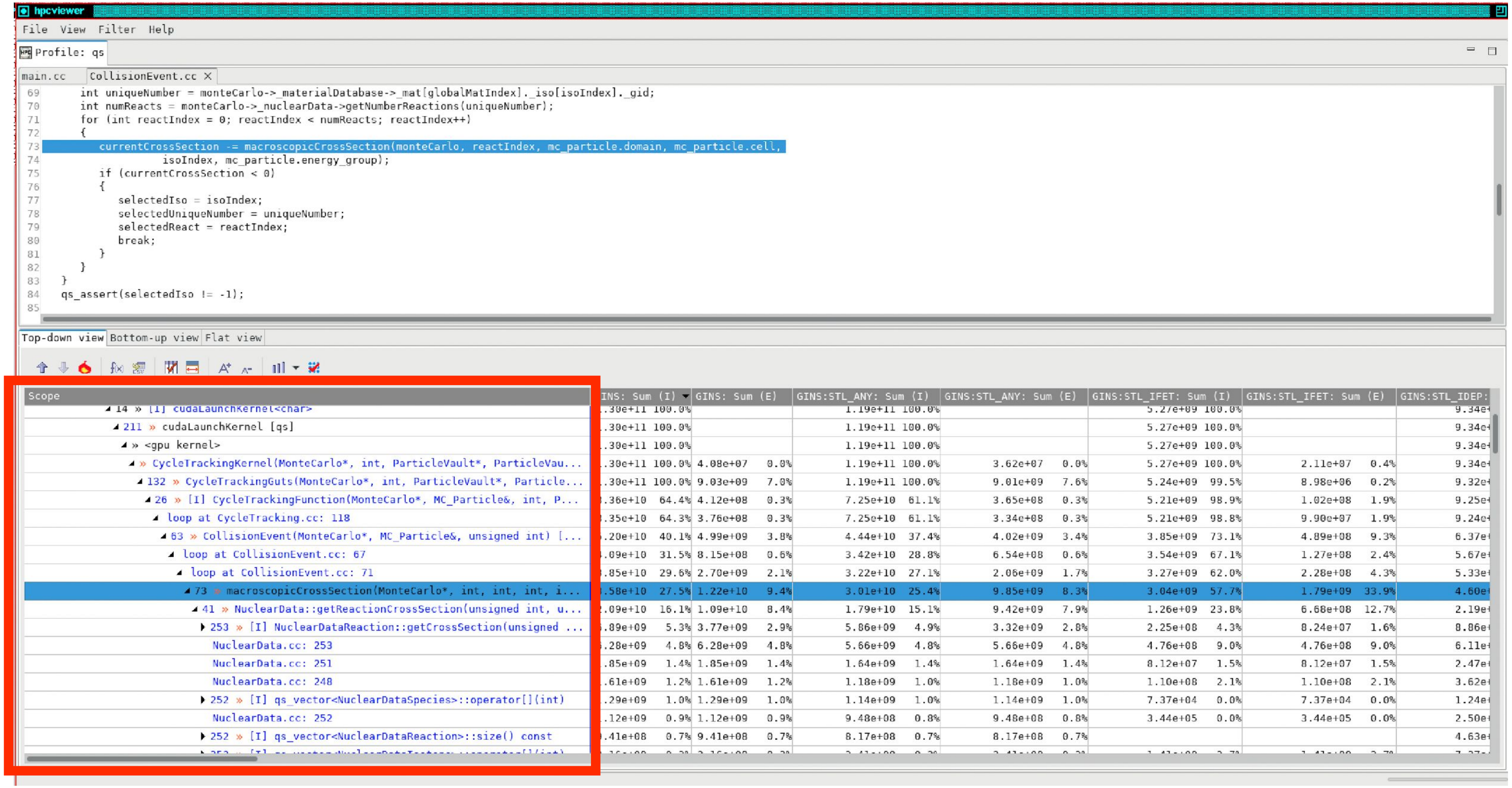


Time Range: [0s, 7.39s]   Cross Hair: (5.86s, RANK 3 GPUCONTEXT 23389 GPUSTREAM 8)

<program root>
main
cycleTracking(MonteCarlo*)
<gpu kernel>
CycleTrackingKernel(MonteCarlo*, int,

Imbalance in GPU work across ranks

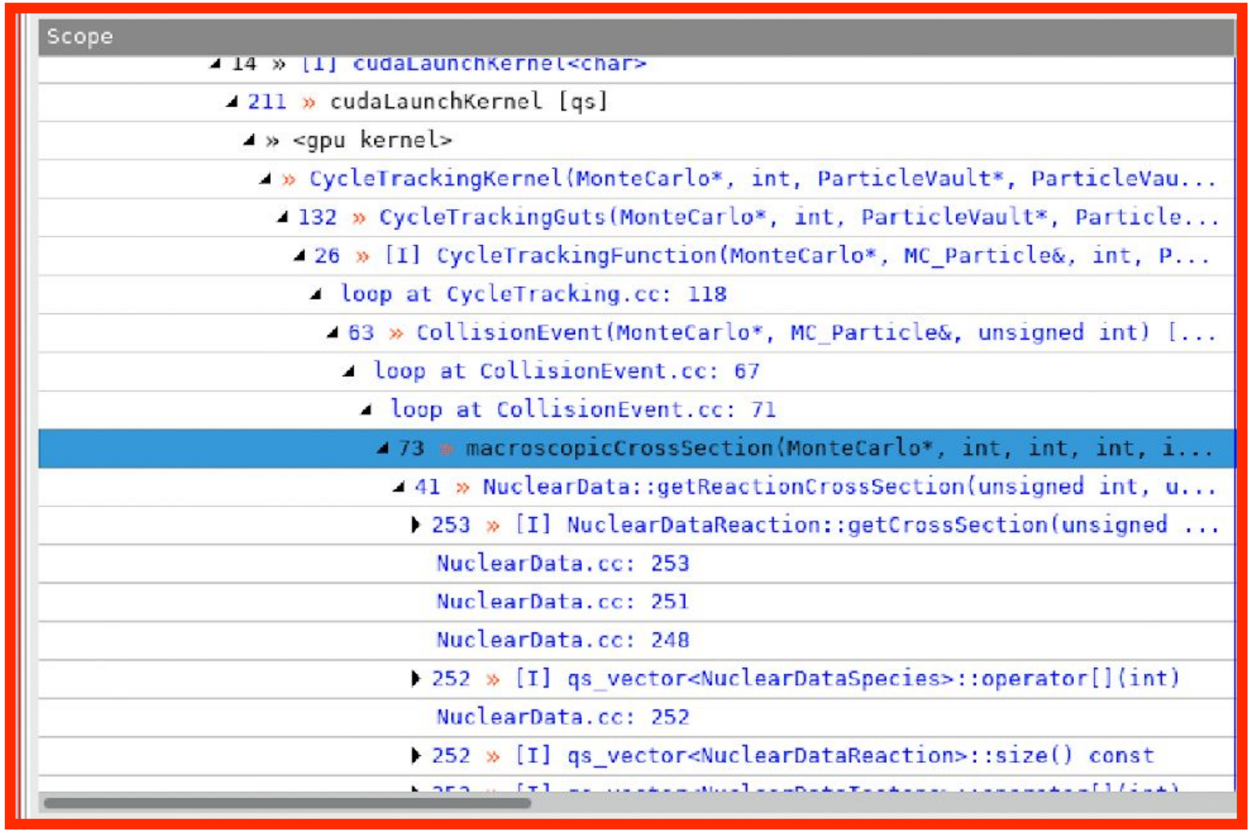
Quicksilver: Detailed analysis within a Kernel using PC Sampling



```
    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	INS: 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>	.30e+11	100.0%			1.19e+11	100.0%			5.27e+09	100.0%			9.34e
211 » cudaLaunchKernel [qs]	.30e+11	100.0%			1.19e+11	100.0%			5.27e+09	100.0%			9.34e
» <gpu kernel>	.30e+11	100.0%			1.19e+11	100.0%			5.27e+09	100.0%			9.34e
» CycleTrackingKernel(MonteCarlo*, int, ParticleVault*, ParticleVau...	.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...	.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...	.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	.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) [...	.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	.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	.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...	.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...	.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 ...	.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	.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	.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	.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)	.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	.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	.41e+08	0.7%	9.41e+08	0.7%	8.17e+08	0.7%	8.17e+08	0.7%					4.63e

Quicksilver: Attribution to Code within a Kernel



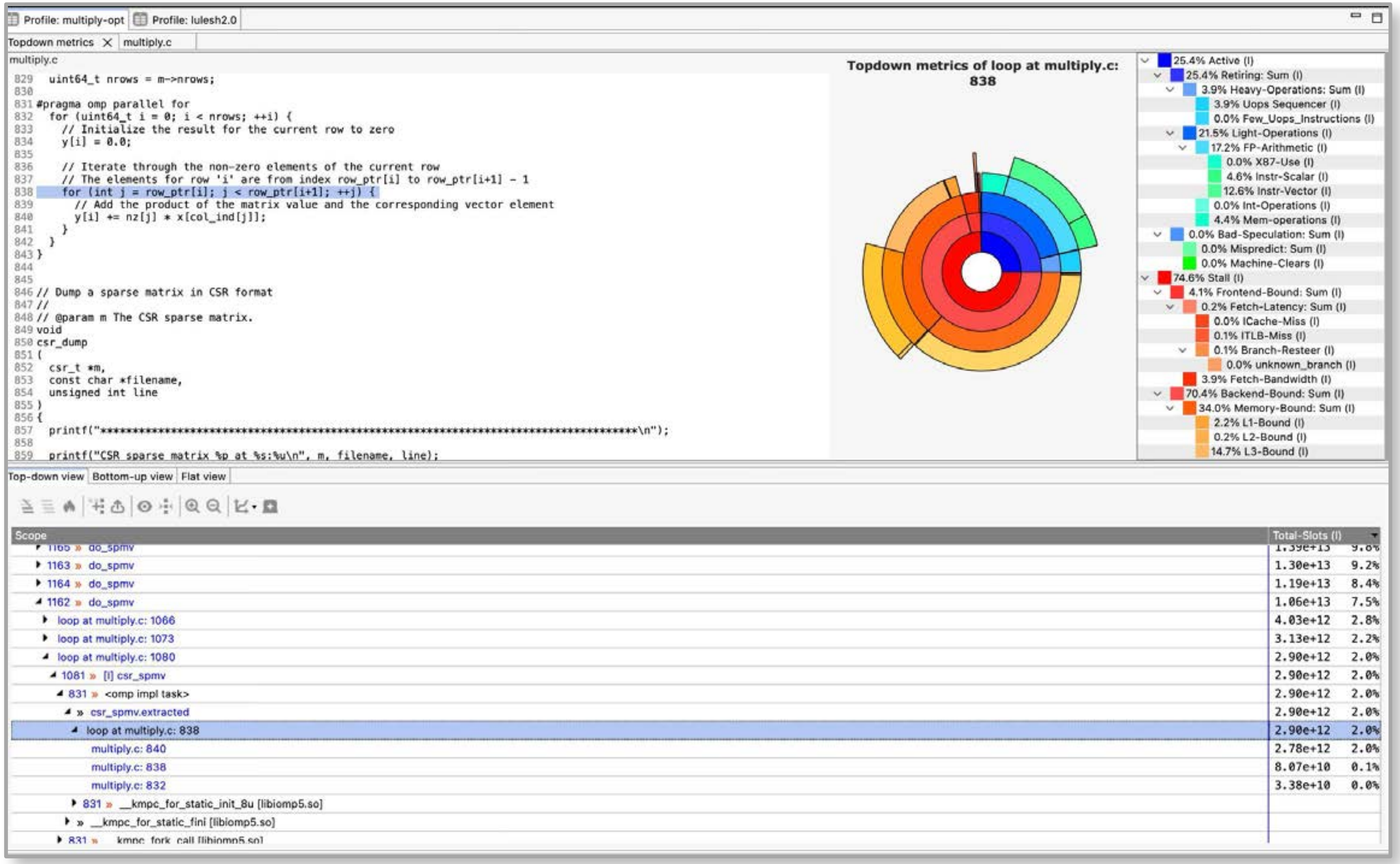
```
Scope
◢ 14 » [I] cudaLaunchKernel<char>
  ◢ 211 » cudaLaunchKernel [qs]
    ◢ » <gpu kernel>
      ◢ » CycleTrackingKernel(MonteCarlo*, int, ParticleVault*, ParticleVau...
        ◢ 132 » CycleTrackingGuts(MonteCarlo*, int, ParticleVault*, Particle...
          ◢ 26 » [I] CycleTrackingFunction(MonteCarlo*, 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
```

Today's Agenda

- Introduce HPCToolkit tools and workflow
- Illustrate HPCToolkit's use with some case studies
- Live demos
  - profiling and tracing
  - instruction-based performance metrics
- Instruction-level performance monitoring
  - explaining instruction-level performance
  - improving analysis and attribution of PC samples
  - automating analysis of GPU bottlenecks
- Discussion about needs, problems, and suggestions

Work in Progress: Present GPU Metrics using a Donut Graph

- Figure shows forthcoming top-down display of CPU metrics
- Plan similar top-down display of GPU metrics
  - Issues
  - Exposed stalls
    - Memory
    - Pipeline
    - Ifetch
    - …
  - Hidden stalls


Profile: multiply-opt | Profile: lulesh2.0

Topdown metrics | multiply.c

multiply.c

```
829   uint64_t nrows = m->nrows;
830
831 #pragma omp parallel for
832   for (uint64_t i = 0; i < nrows; ++i) {
833     // Initialize the result for the current row to zero
834     y[i] = 0.0;
835
836     // Iterate through the non-zero elements of the current row
837     // The elements for row 'i' are from index row_ptr[i] to row_ptr[i+1] - 1
838     for (int j = row_ptr[i]; j < row_ptr[i+1]; ++j) {
839       // Add the product of the matrix value and the corresponding vector element
840       y[i] += nz[j] * x[col_ind[j]];
841     }
842   }
843 }
844
845
846 // Dump a sparse matrix in CSR format
847 //
848 // @param m The CSR sparse matrix.
849 void
850 csr_dump
851 (
852   csr_t *m,
853   const char *filename,
854   unsigned int line
855 )
856 {
857   printf("*************************************************************************************\n");
858
859   printf("CSR sparse matrix %p at %s:%u\n", m, filename, line);
```

Topdown metrics of loop at multiply.c: 838

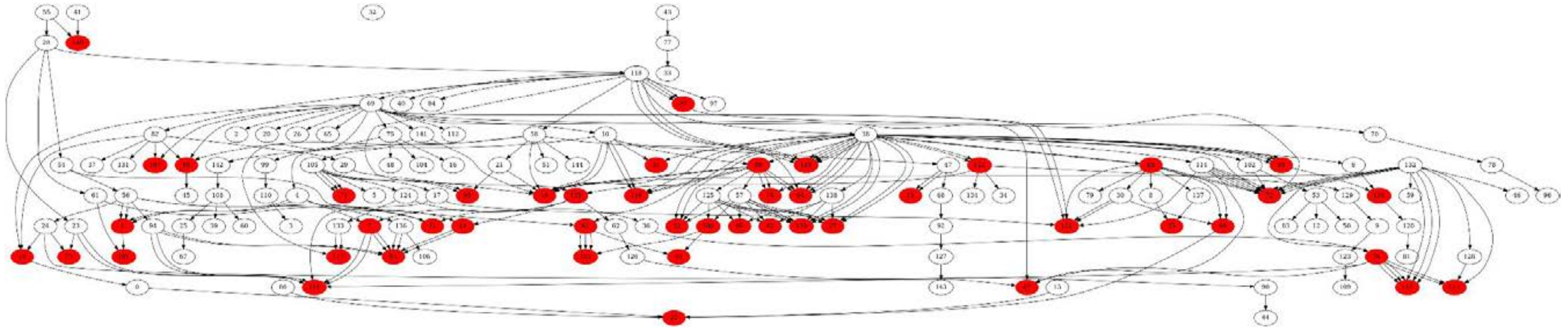
- 25.4% Active (I)
  - 25.4% Retiring: Sum (I)
    - 3.9% Heavy-Operations: Sum (I)
      - 3.9% Uops Sequencer (I)
      - 0.0% Few_Uops_Instructions (I)
    - 21.5% Light-Operations (I)
      - 17.2% FP-Arithmetic (I)
        - 0.0% X87-Use (I)
        - 4.6% Instr-Scalar (I)
        - 12.6% Instr-Vector (I)
      - 0.0% Int-Operations (I)
      - 4.4% Mem-operations (I)
  - 0.0% Bad-Speculation: Sum (I)
    - 0.0% Mispredict: Sum (I)
    - 0.0% Machine-Clears (I)
- 74.6% Stall (I)
  - 4.1% Frontend-Bound: Sum (I)
    - 0.2% Fetch-Latency: Sum (I)
      - 0.0% ICache-Miss (I)
      - 0.1% ITLB-Miss (I)
      - 0.1% Branch-Resteer (I)
        - 0.0% unknown_branch (I)
    - 3.9% Fetch-Bandwidth (I)
  - 70.4% Backend-Bound: Sum (I)
    - 34.0% Memory-Bound: Sum (I)
      - 2.2% L1-Bound (I)
      - 0.2% L2-Bound (I)
      - 14.7% L3-Bound (I)

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

Scope	Total-Slots (I)	
1165 » do_spmv	1.39e+13	9.8%
1163 » do_spmv	1.30e+13	9.2%
1164 » do_spmv	1.19e+13	8.4%
1162 » do_spmv	1.06e+13	7.5%
loop at multiply.c: 1066	4.03e+12	2.8%
loop at multiply.c: 1073	3.13e+12	2.2%
loop at multiply.c: 1080	2.90e+12	2.0%
1081 » [I] csr_spmv	2.90e+12	2.0%
831 » <omp impl task>	2.90e+12	2.0%
» csr_spmv.extracted	2.90e+12	2.0%
loop at multiply.c: 838	2.90e+12	2.0%
multiply.c: 840	2.78e+12	2.0%
multiply.c: 838	8.07e+10	0.1%
multiply.c: 832	3.38e+10	0.0%
831 » __kmpc_for_static_init_8u [libiomp5.so]		
» __kmpc_for_static_fini [libiomp5.so]		
831 » __kmpc_fork_call [libiomp5.so]		

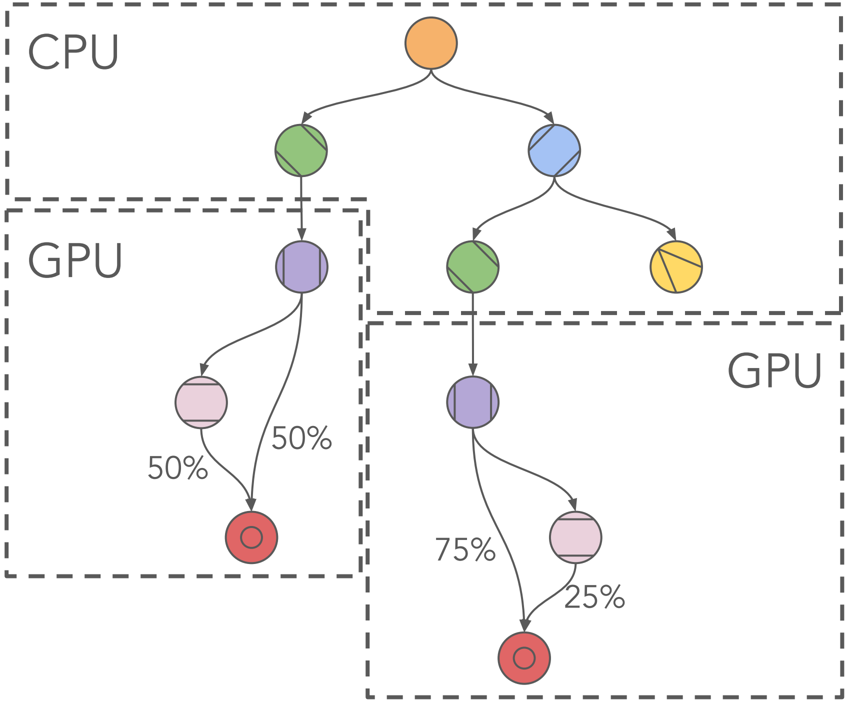
WIP: Enabling Instruction-level Metrics for Complex Kernels

- Calling context is great for visualization, but slow to analyze for complex GPU kernels!
  - Every plausible calling context must be expanded to attribute performance
  - Example: Quicksilver has one kernel with 145 separate GPU functions
    - Call graph below, **red** nodes have multiple plausible calling contexts
  - Sometimes >100,000 calling contexts for a single function, e.g. cuda_div



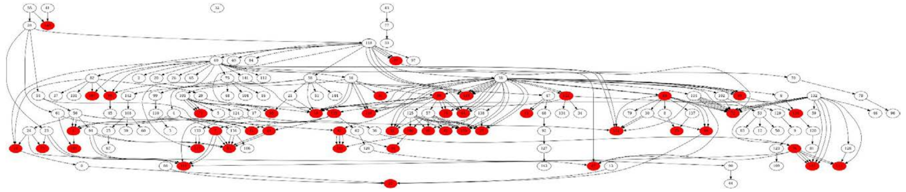
WIP: Enabling Instruction-level Metrics for Complex Kernels

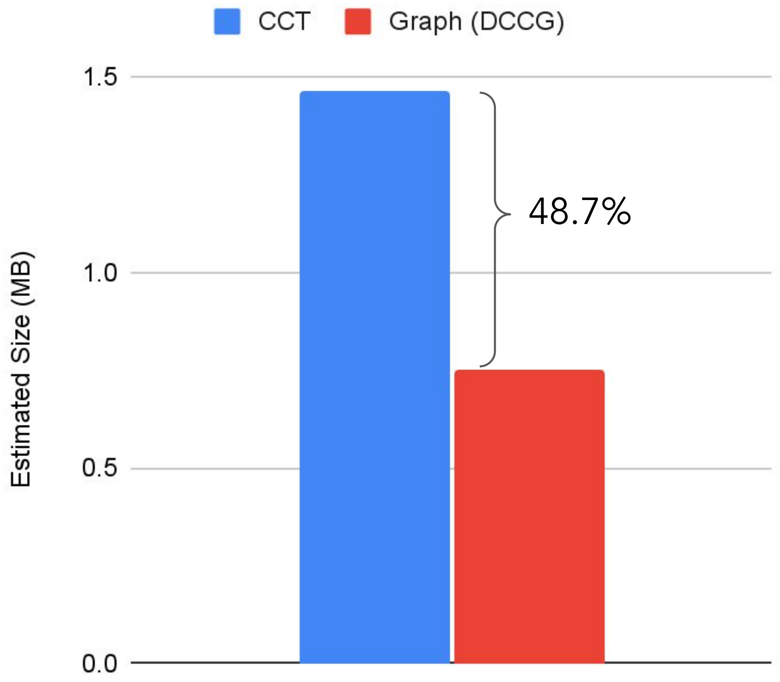
- Next generation: Graph-based calling contexts
  - Performance is attributed to nodes
  - Distribution factors recorded on edges
- Efficient representation for GPU code
  - Avoid reconstruction of duplicate contexts
  - Match GPU measurement capabilities
- Clean model for both CPU and GPU performance
  - Not a hybrid model, one graph handles all



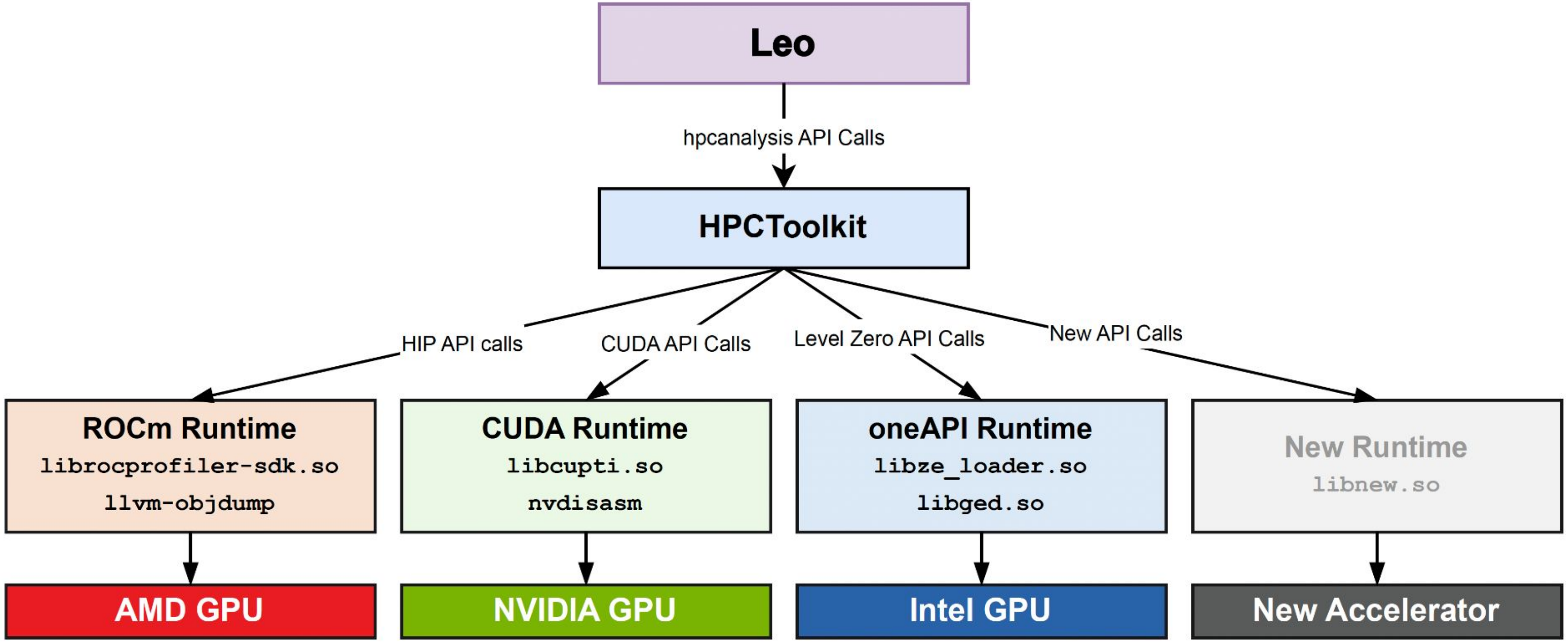
WIP: Enabling Instruction-level Metrics for Complex Kernels

- Case study: Quicksilver
  - 409 unique calling contexts
  - 145 graph nodes (1 per GPU function)
- Preliminary results
  - 48% estimated size reduction
  - 1.3x potential speed up





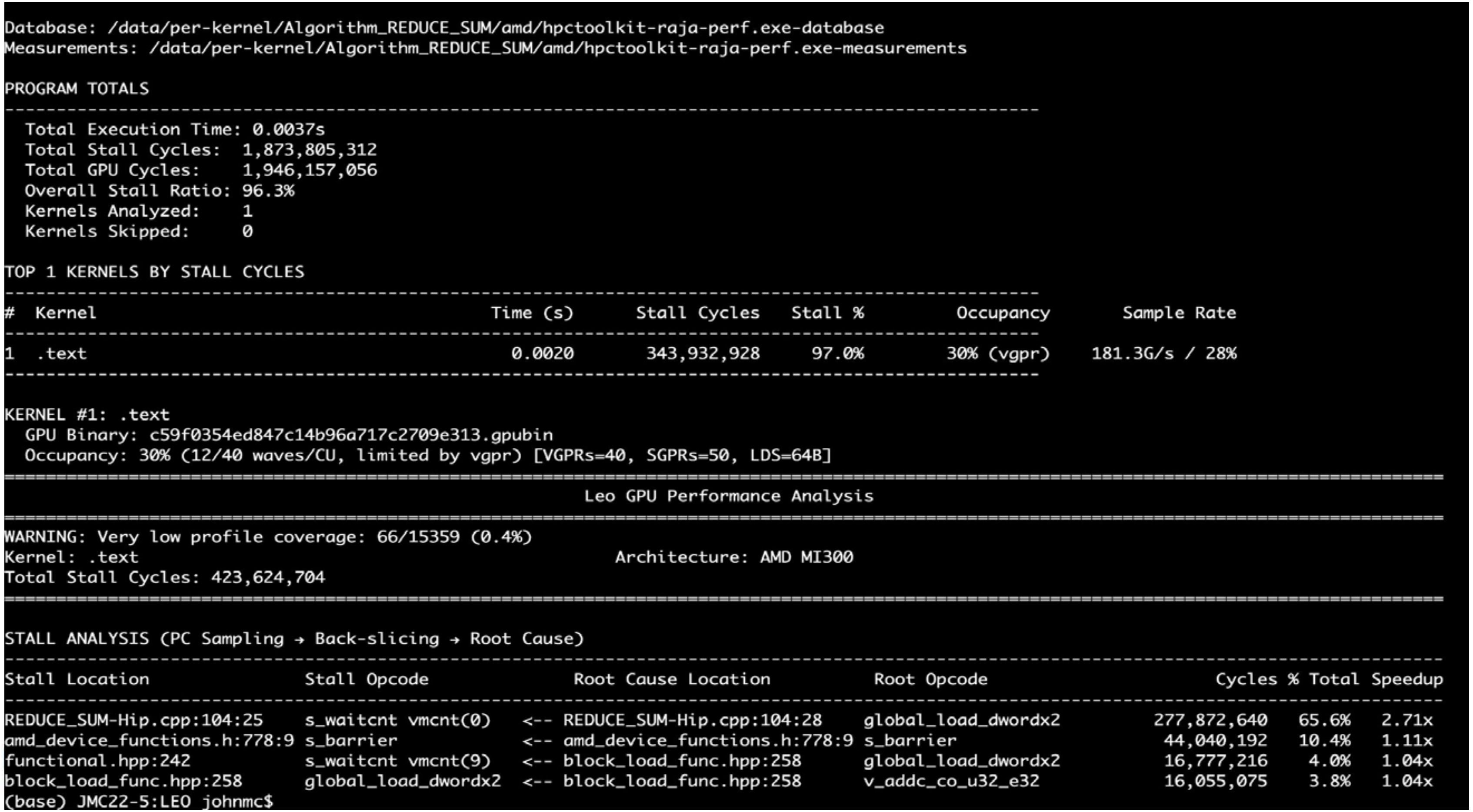
WIP: Leo - Platform-independent analysis with PC Sampling



WIP: Leo - Platform-independent analysis with PC Sampling

- GPU PC sampling collects a wealth of information
  - Stall reasons (AMD, Intel, NVIDIA)
  - Hidden stalls (NVIDIA, AMD)
  - Register utilization (NVIDIA, AMD)
  - Compute unit utilization (AMD)
    - Pipeline utilization
    - Wavefront occupancy
    - Thread utilization (divergence)
- Leo pinpoints and quantifies root causes of GPU stalls
  - Ranks code locations suffering from GPU instruction stalls
  - Dumps machine code into assembly code
  - Performs dataflow analysis to trace stalls back to potential root causes

WIP: Leo - Sample Report: Analysis of RajaPerfSuite



```
Database: /data/per-kernel/Algorithm_REDUCE_SUM/amd/hpctoolkit-raja-perf.exe-database
Measurements: /data/per-kernel/Algorithm_REDUCE_SUM/amd/hpctoolkit-raja-perf.exe-measurements

PROGRAM TOTALS
----------------------------------------------------------------------------------------------------
  Total Execution Time: 0.0037s
  Total Stall Cycles:  1,873,805,312
  Total GPU Cycles:    1,946,157,056
  Overall Stall Ratio: 96.3%
  Kernels Analyzed:    1
  Kernels Skipped:     0

TOP 1 KERNELS BY STALL CYCLES
----------------------------------------------------------------------------------------------------
#  Kernel                                       Time (s)      Stall Cycles   Stall %         Occupancy       Sample Rate
----------------------------------------------------------------------------------------------------
1  .text                                          0.0020       343,932,928     97.0%        30% (vgpr)    181.3G/s / 28%
----------------------------------------------------------------------------------------------------

KERNEL #1: .text
  GPU Binary: c59f0354ed847c14b96a717c2709e313.gpubin
  Occupancy: 30% (12/40 waves/CU, limited by vgpr) [VGPRs=40, SGPRs=50, LDS=64B]
====================================================================================================================================================
                                                            Leo GPU Performance Analysis
====================================================================================================================================================
WARNING: Very low profile coverage: 66/15359 (0.4%)
Kernel: .text                                                  Architecture: AMD MI300
Total Stall Cycles: 423,624,704
====================================================================================================================================================

STALL ANALYSIS (PC Sampling → Back-slicing → Root Cause)
----------------------------------------------------------------------------------------------------------------------------------------------------
Stall Location             Stall Opcode              Root Cause Location          Root Opcode                        Cycles % Total  Speedup
----------------------------------------------------------------------------------------------------------------------------------------------------
REDUCE_SUM-Hip.cpp:104:25  s_waitcnt vmcnt(0)     <-- REDUCE_SUM-Hip.cpp:104:28   global_load_dwordx2           277,872,640   65.6%   2.71x
amd_device_functions.h:778:9 s_barrier             <-- amd_device_functions.h:778:9 s_barrier                      44,040,192   10.4%   1.11x
functional.hpp:242         s_waitcnt vmcnt(9)     <-- block_load_func.hpp:258     global_load_dwordx2            16,777,216    4.0%   1.04x
block_load_func.hpp:258    global_load_dwordx2    <-- block_load_func.hpp:258     v_addc_co_u32_e32              16,055,075    3.8%   1.04x
(base) JMC22-5:LEO johnmc$
```

Discussion

- What performance issues do you need help with?
- What would help you investigate these issues?
- Are there any features that you would like added?

HPCToolkit Resources

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