



Performance Analysis of GPU-accelerated Applications with HPCToolkit

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

Linux Foundation's HPCToolkit Performance Tools

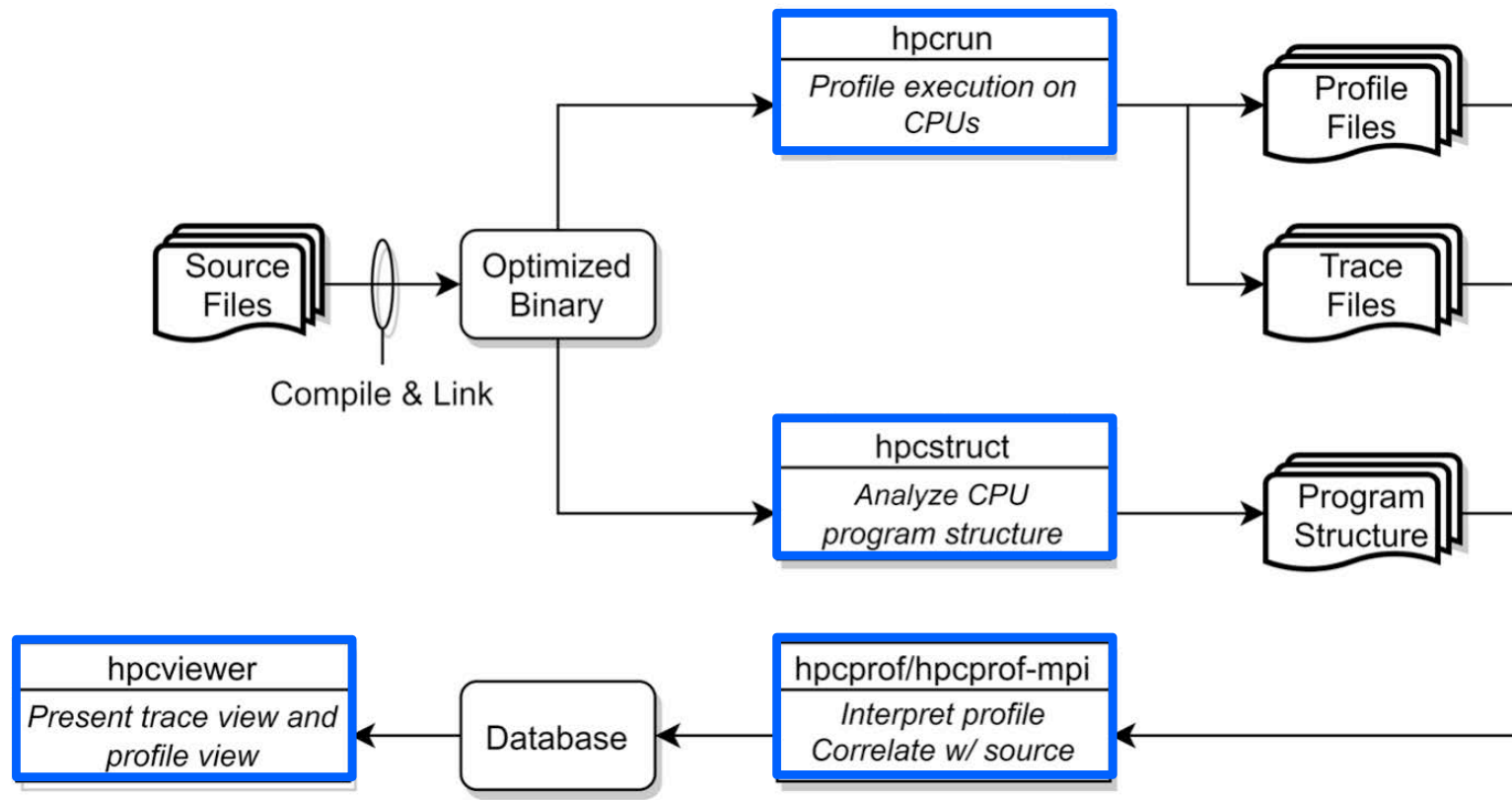
Measure and analyze performance of CPU and GPU-accelerated applications

- Easy: profile unmodified application binaries
- Fast: low-overhead measurement
- Informative: understand where an application spends its time and why
 - call path profiles associate metrics with application source code contexts
 - optional hierarchical traces to understand execution dynamics
- Broad audience
 - application developers
 - framework developers
 - runtime and tool developers
- Measures complex programs on a broad range of platforms
 - CPU: x86_64, Power, ARM
 - GPU: NVIDIA, AMD, Intel

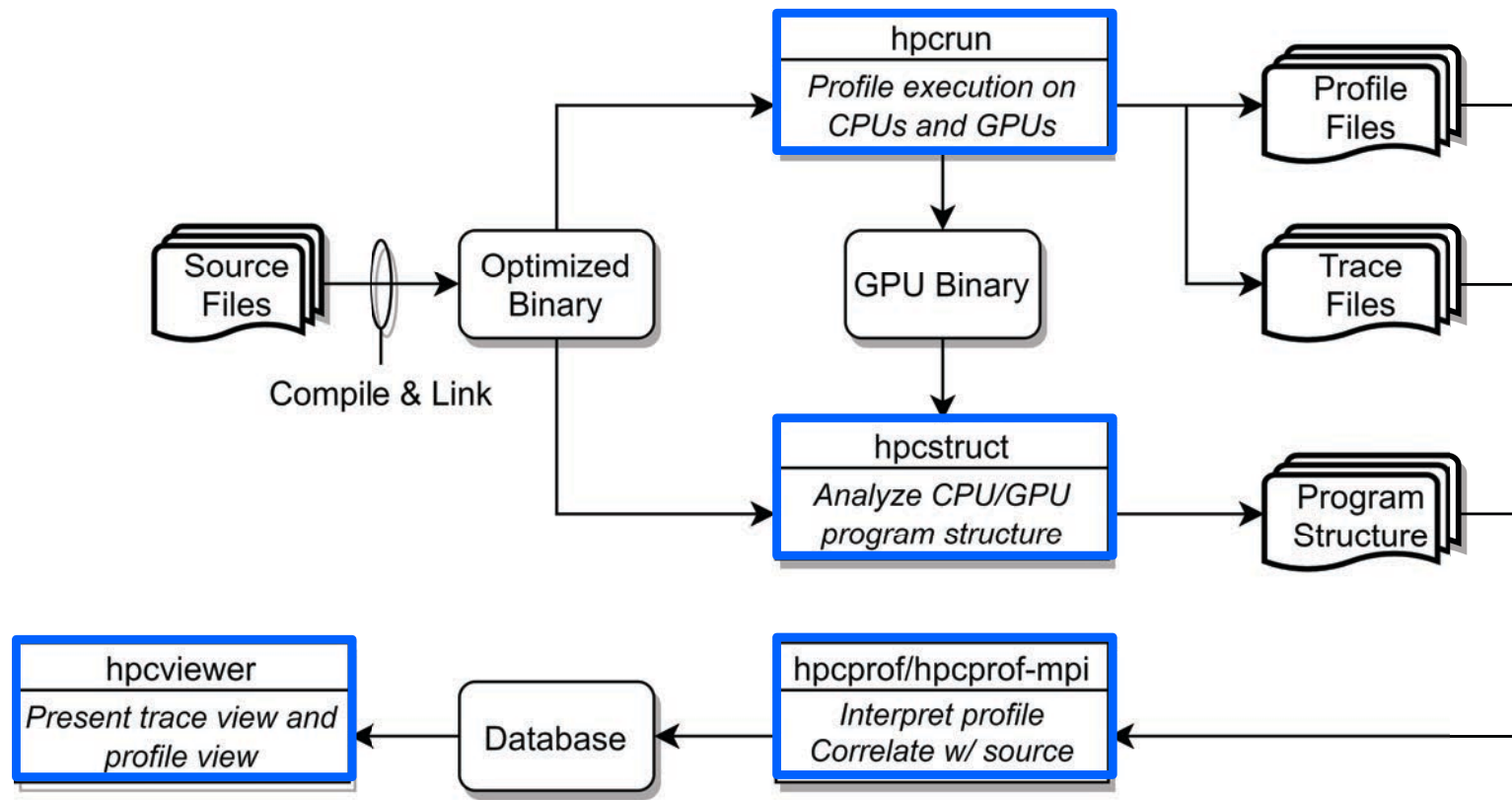
How does HPCToolkit Differ from NVIDIA's Tools?

- NVIDIA NSight Systems
 - tracing of CPU and GPU streams
 - analyze traces when you open them with the GUI
 - long running traces are huge and thus extremely slow to analyze, limiting scalability
 - designed for measurement and analysis within a node
- NVIDIA NSight Compute
 - detailed measurement of kernels with counters and execution replay
 - very slow measurement
 - flat display of measurements within GPU kernels
- HPCToolkit
 - supports more scalable tracing than Nsight Systems
 - measure exascale executions across many GPUs and nodes
 - scalable, parallel post-mortem analysis vs. non-scalable in-GUI analysis
 - detailed reconstruction of estimates for calling context profiles within GPU kernels

HPCToolkit's Workflow for CPU Applications



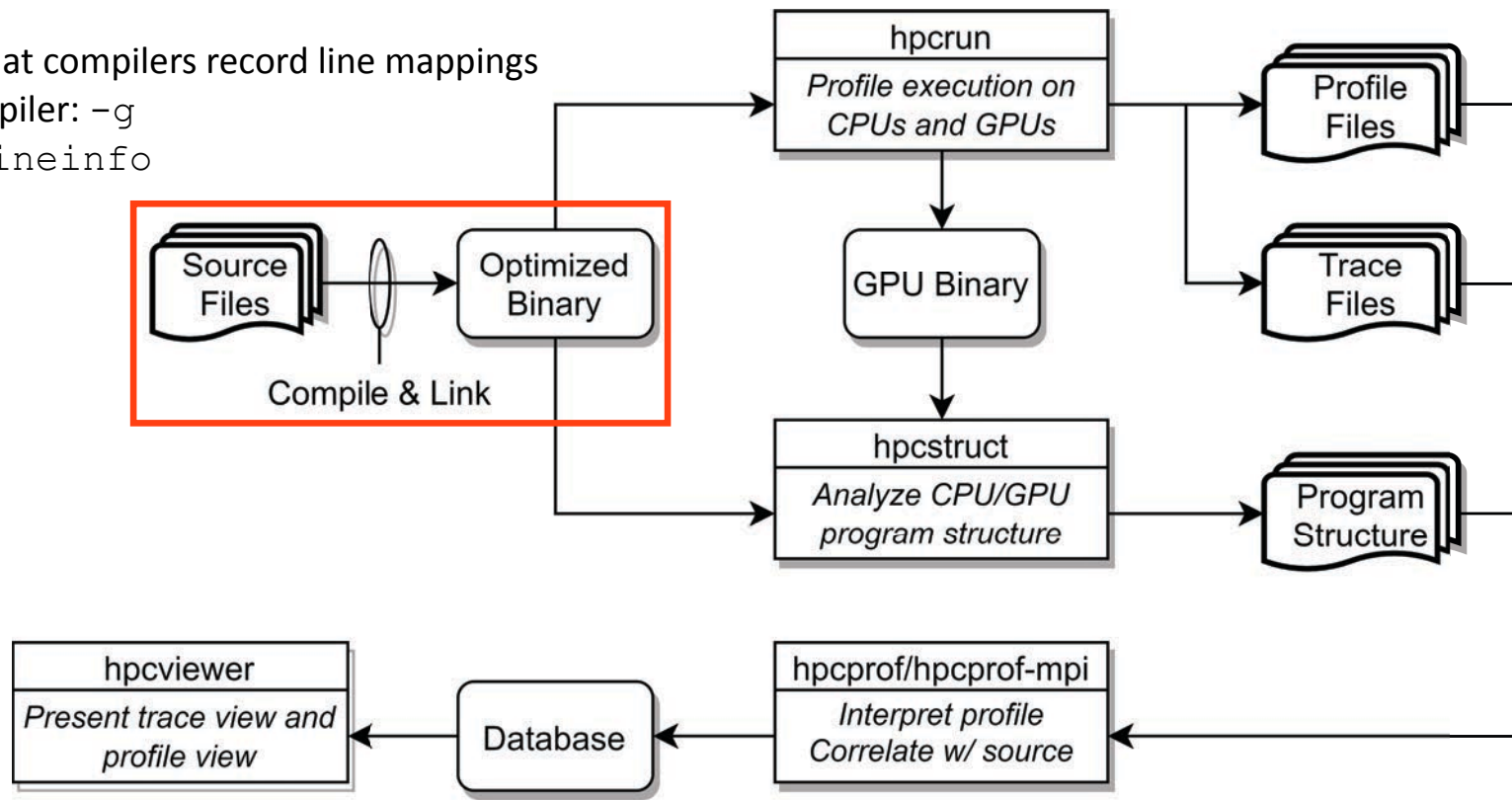
HPCToolkit's Workflow for GPU-accelerated Applications



HPCToolkit's Workflow for GPU-accelerated Applications

Step 1:

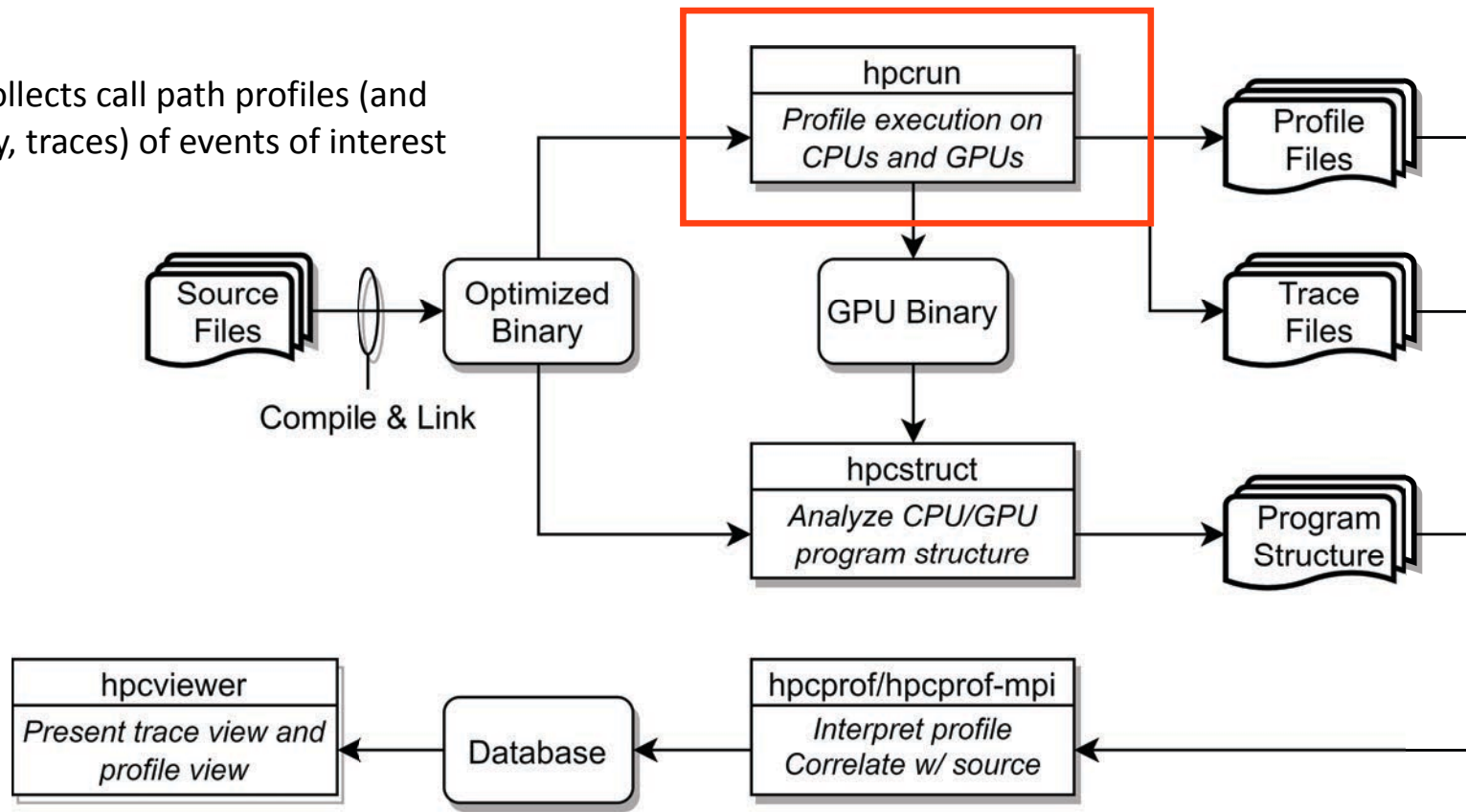
- Ensure that compilers record line mappings
- host compiler: `-g`
- `nvcc: -lineinfo`



HPCToolkit's Workflow for GPU-accelerated Applications

Step 2:

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



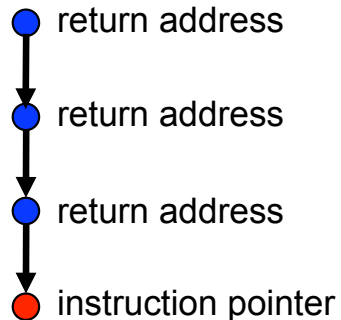
Measurement of CPU and GPU-accelerated Applications

- Sampling using Linux timers and hardware counter overflows on the CPU
- Callbacks when GPU operations are launched and (sometimes) completed
- Event stream for GPU operations; PC Samples (NVIDIA, AMD, Intel)
- Binary instrumentation of GPU kernels on Intel GPUs for fine-grain measurement

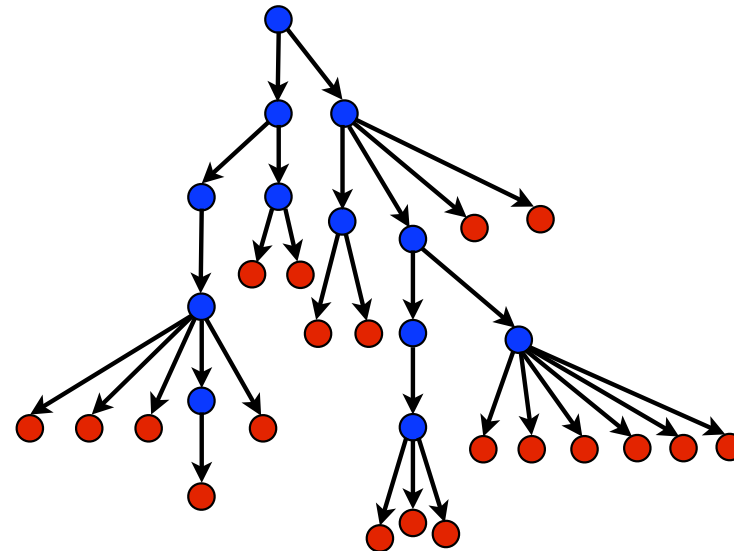
Call Stack Unwinding to Attribute Costs in Context

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

Call path sample



Calling context tree



hpcrun: Measure CPU and/or GPU activity

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

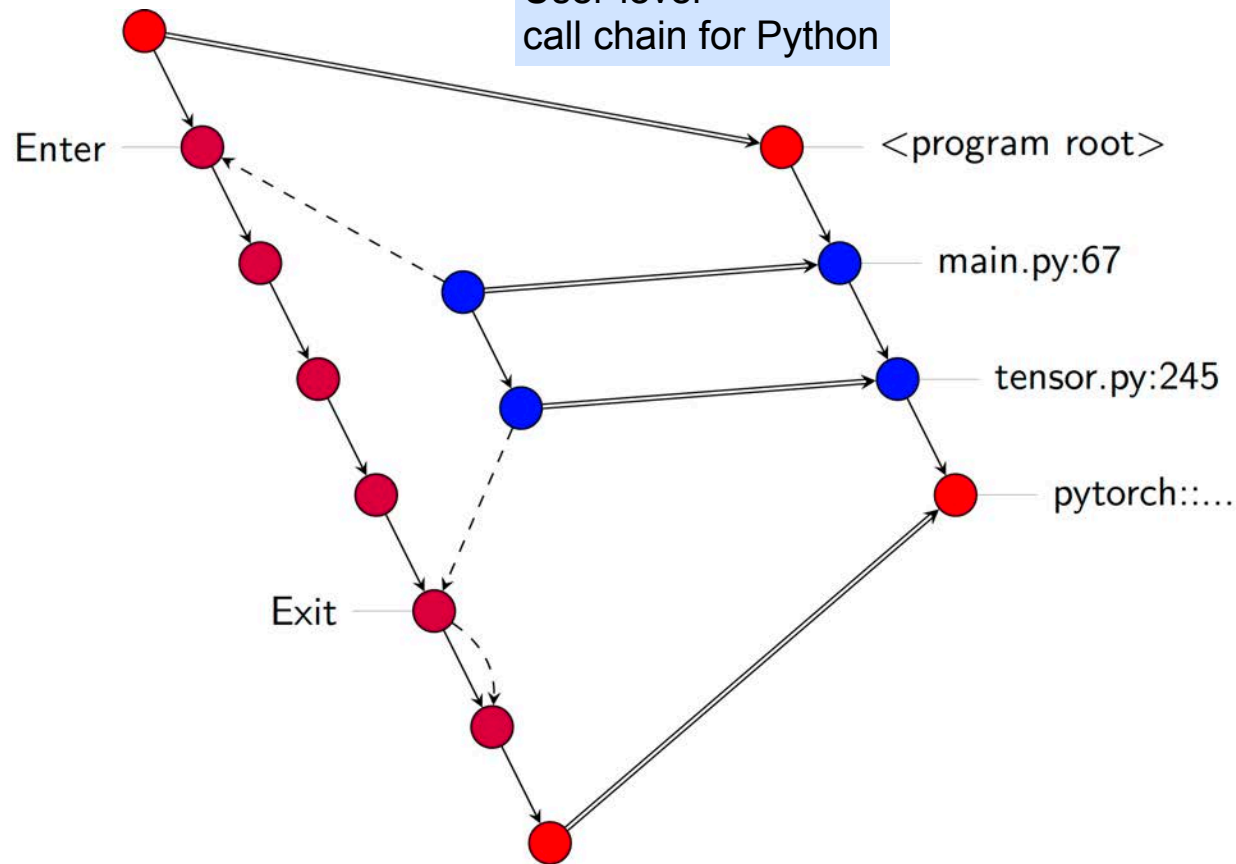
Measure and Attribute Performance of Python-driven Codes

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

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

Implementation-level
call chain for Python

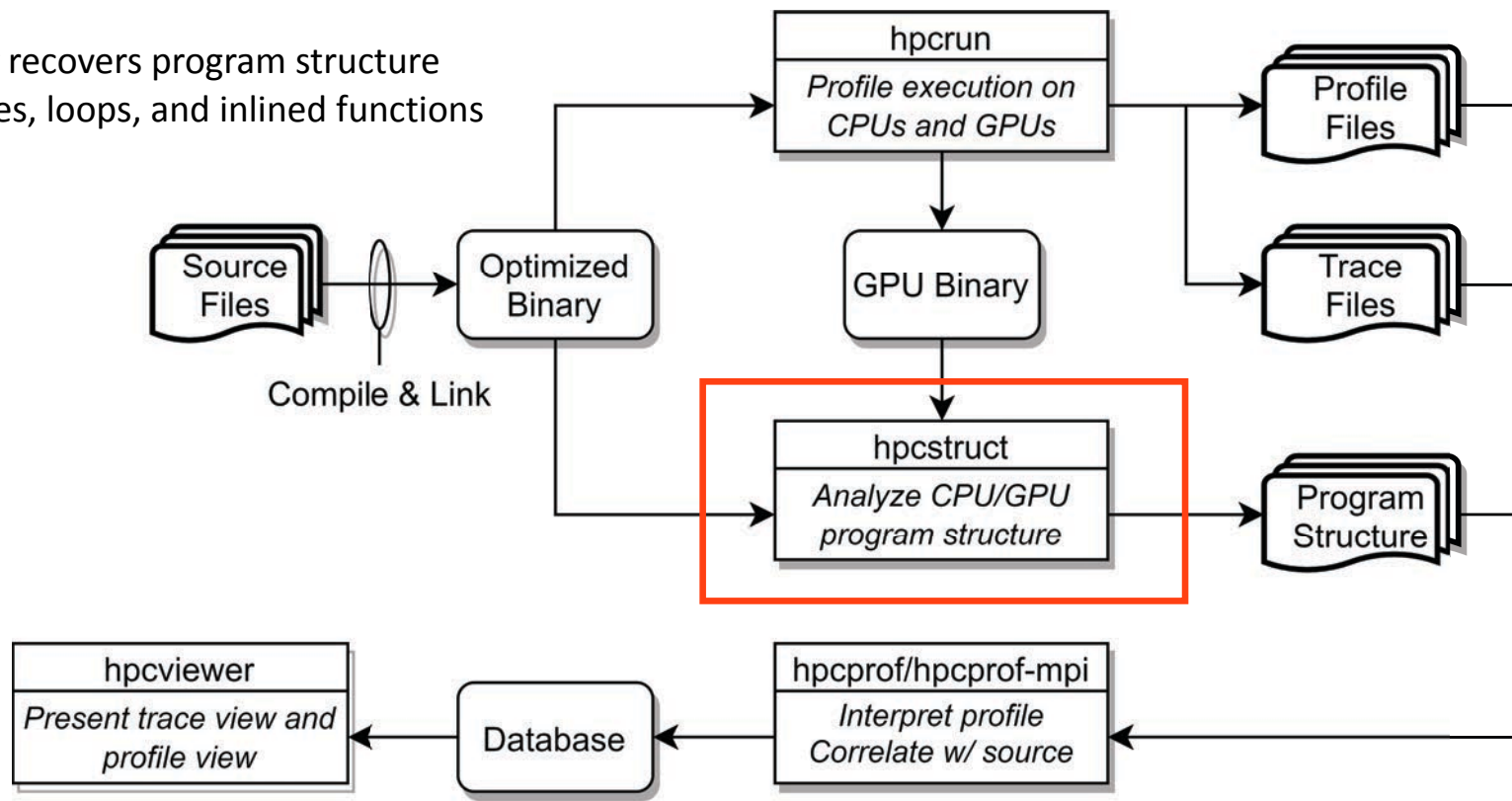
User-level
call chain for Python



HPCToolkit's Workflow for GPU-accelerated Applications

Step 3:

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



hpcstruct: Analyze CPU and GPU Binaries Using Multiple Threads

- Usage

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

- What it does

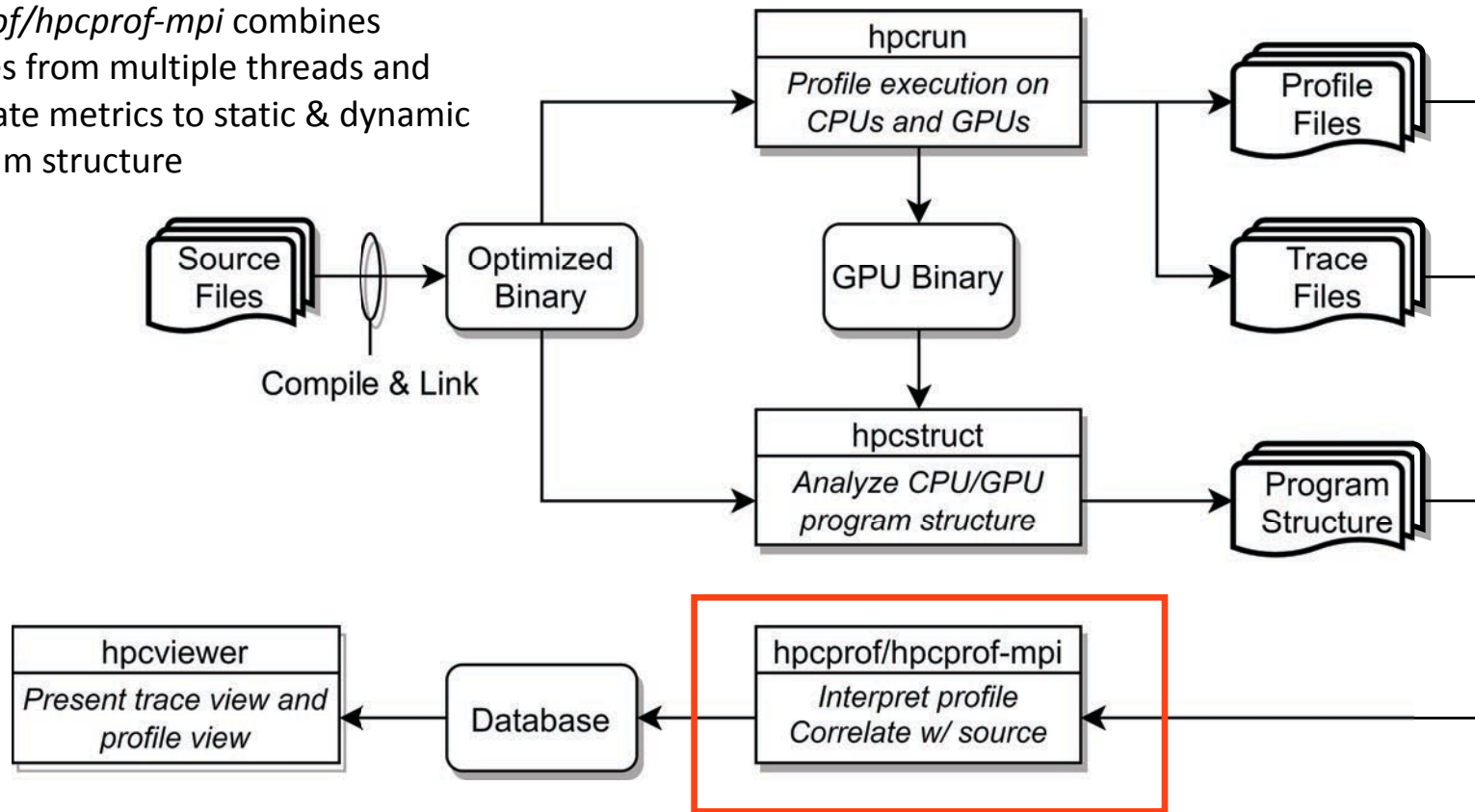
- Recover program structure information
 - Files, functions, inlined templates or functions, loops, source lines
- In parallel, analyze all CPU and GPU binaries that were measured by HPCToolkit
 - analyze large application binaries with 16 threads
 - analyze multiple small application binaries concurrently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

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

HPCToolkit's Workflow for GPU-accelerated Applications

Step 4:

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



hpcprof/hpcprof-mpi: Associate Measurements with Program Structure

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

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

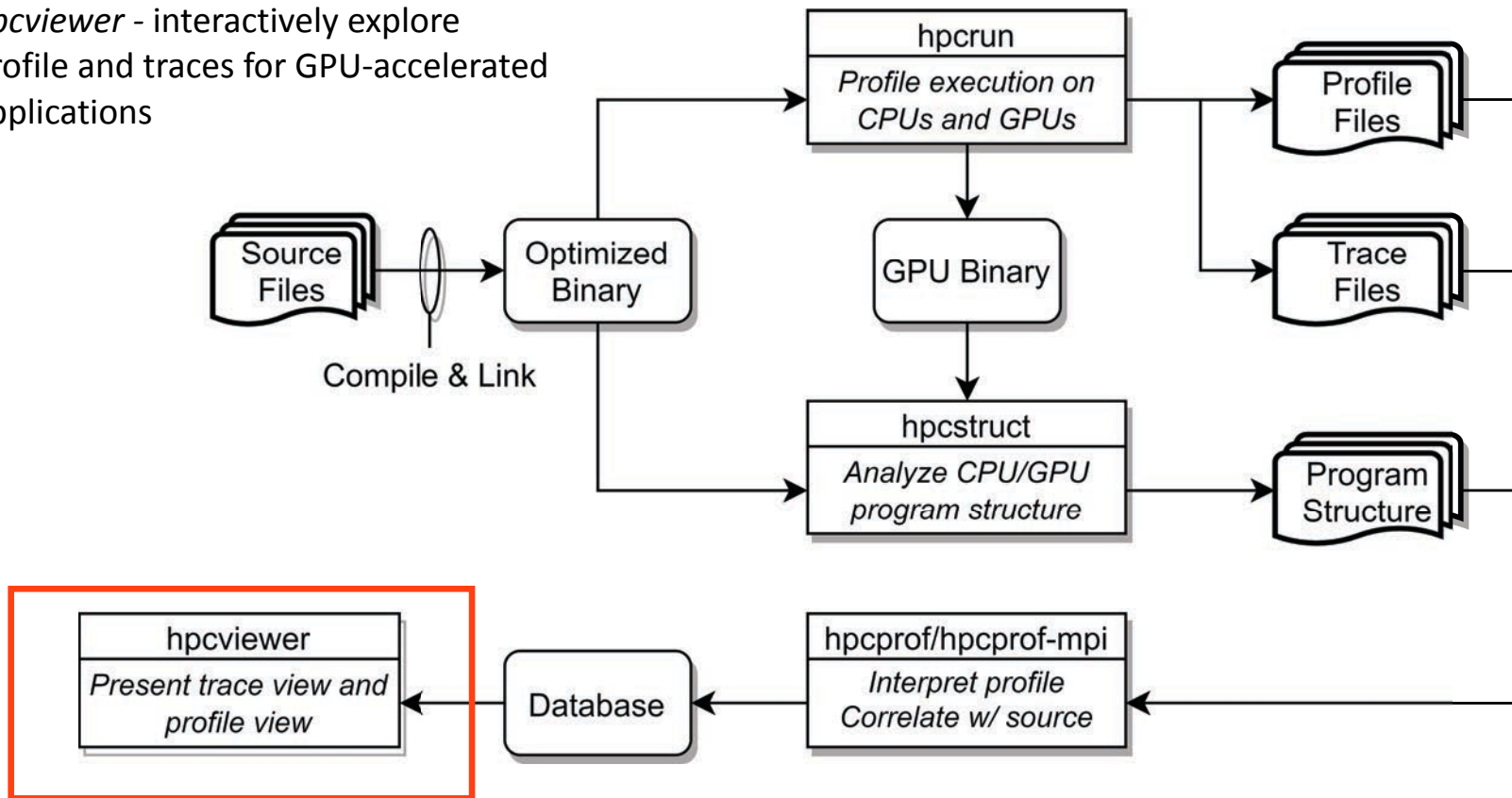
- Analyze data from large executions with distributed-memory parallelism + multithreading (large scale)

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


HPCToolkit's Workflow for GPU-accelerated Applications

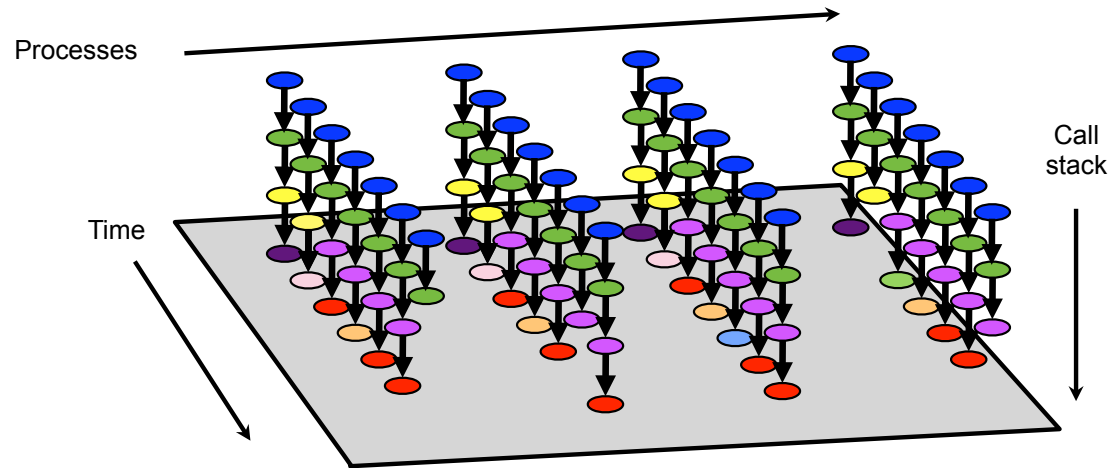
Step 4:

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



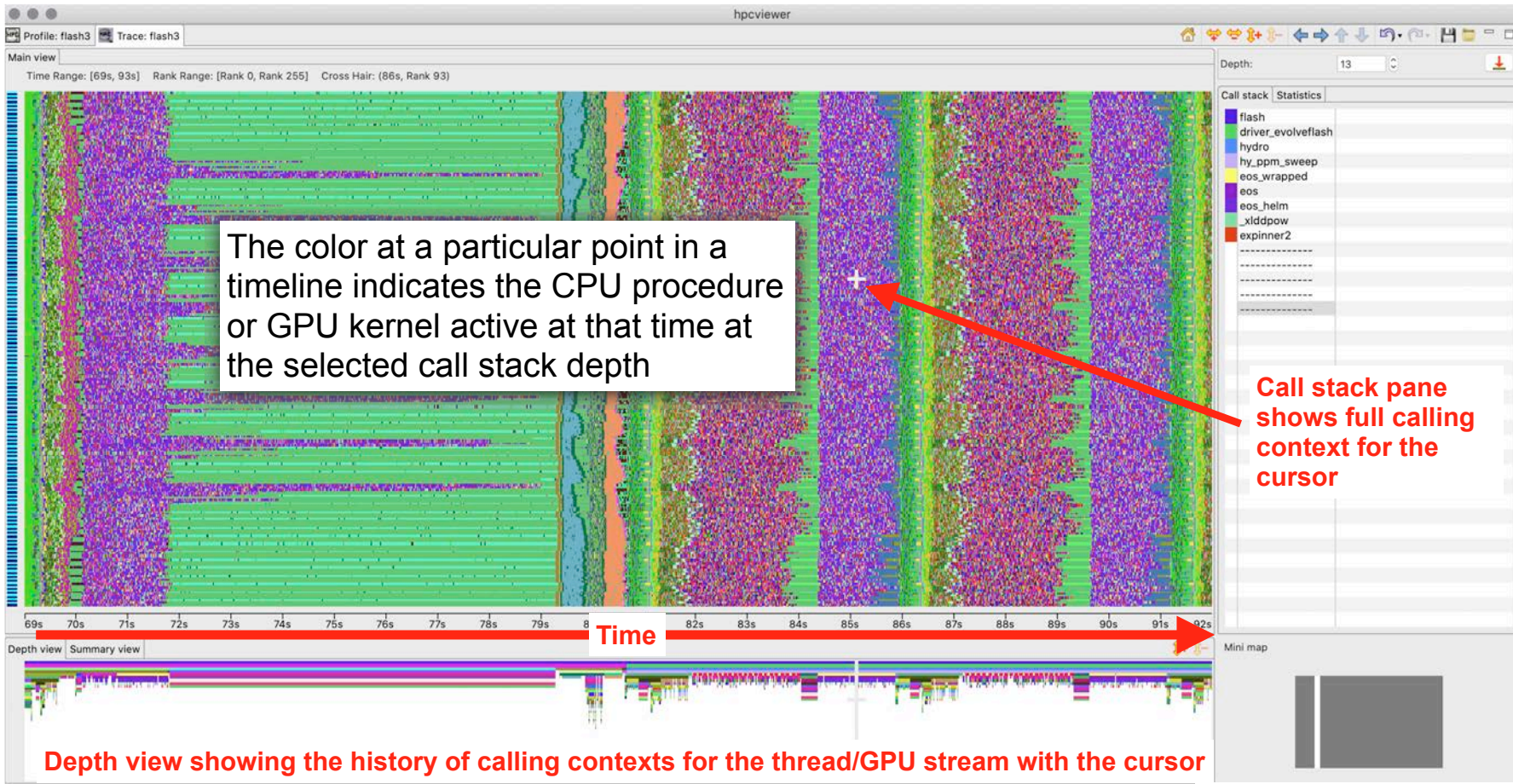
Understanding Temporal Behavior

- 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

MPI ranks, OpenMP Threads, GPU streams



What Can you Measure?

Parallel programming models

Across nodes: MPI, SHMEM, UPC++,

Within nodes: OpenMP, Kokkos, RAJA, HIP, DPC++, Sycl, CUDA, OpenACC

Languages

C, C++, Fortran, Python

Frameworks

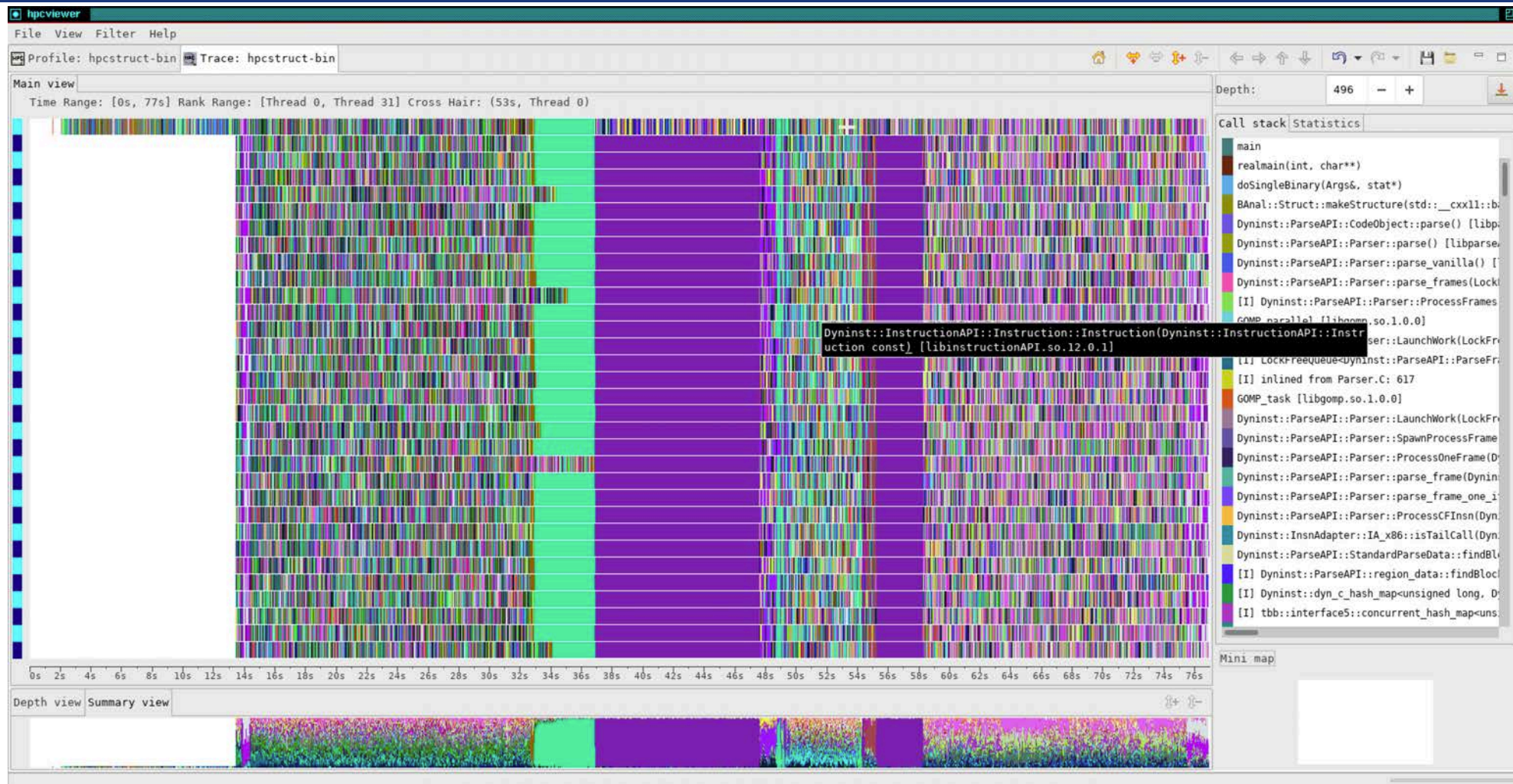
Pytorch, Tensorflow (maybe)

Hardware

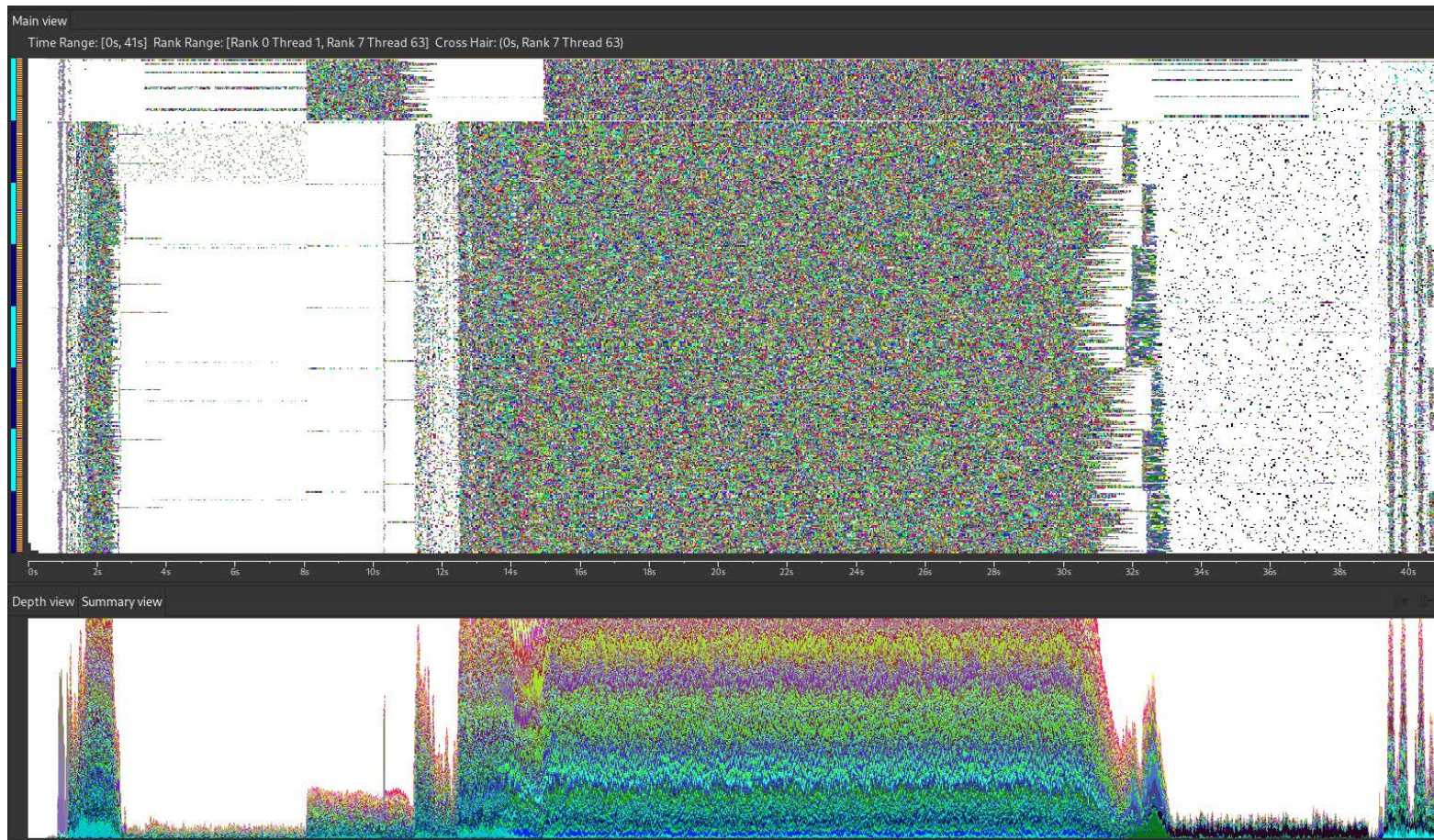
CPU cores and GPUs within a node

All of the nodes in Polaris

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



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



Case Studies

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

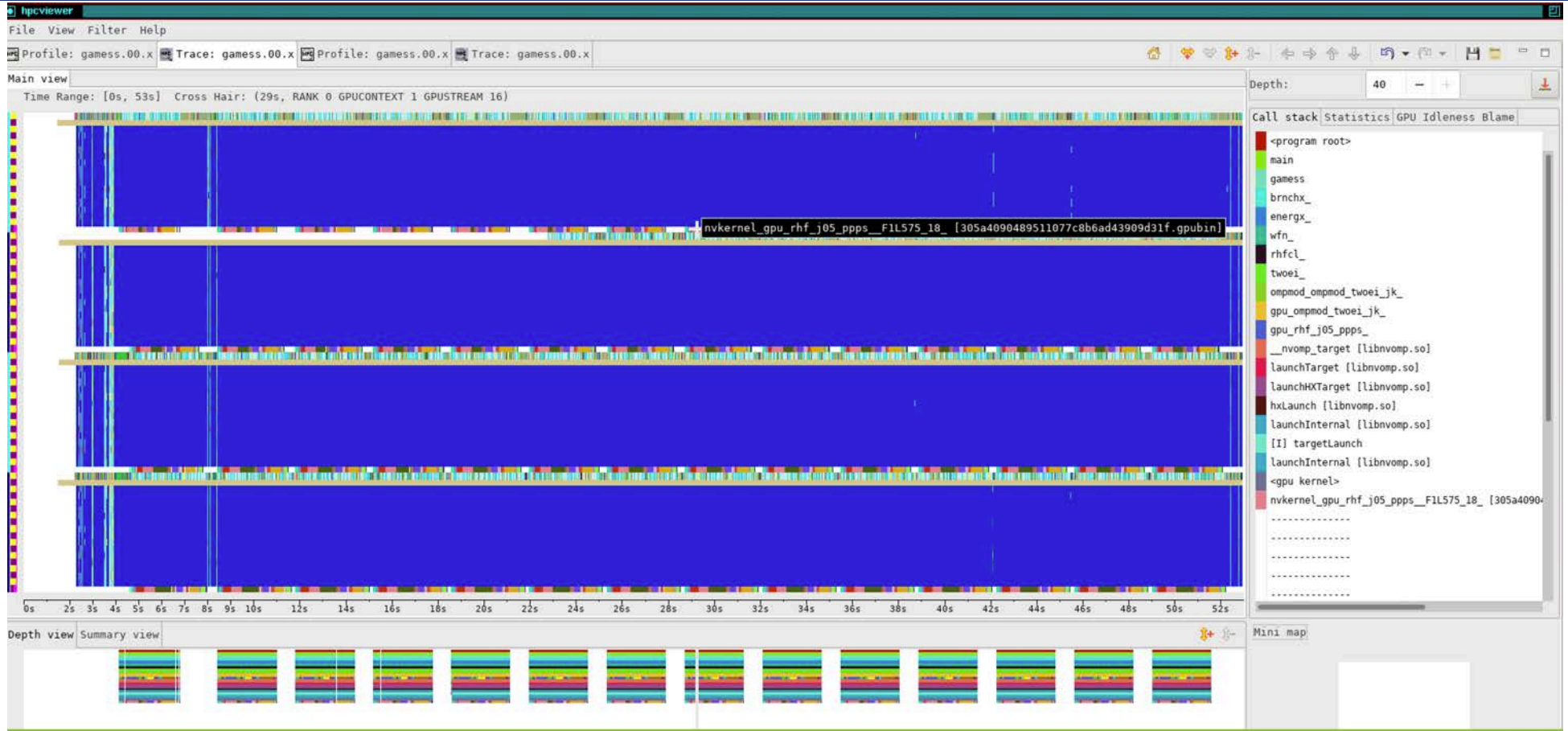
Case Study: GAMESS

- General Atomic and Molecular Electronic Structure System (GAMESS)
 - general *ab initio* quantum chemistry package
- Calculates the energies, structures, and properties of a wide range of chemical systems
- Experiments
 - GPU-accelerated nodes at a prior Perlmutter hackathon
 - Single node with 4 GPUs
 - Five nodes with 20 GPUs

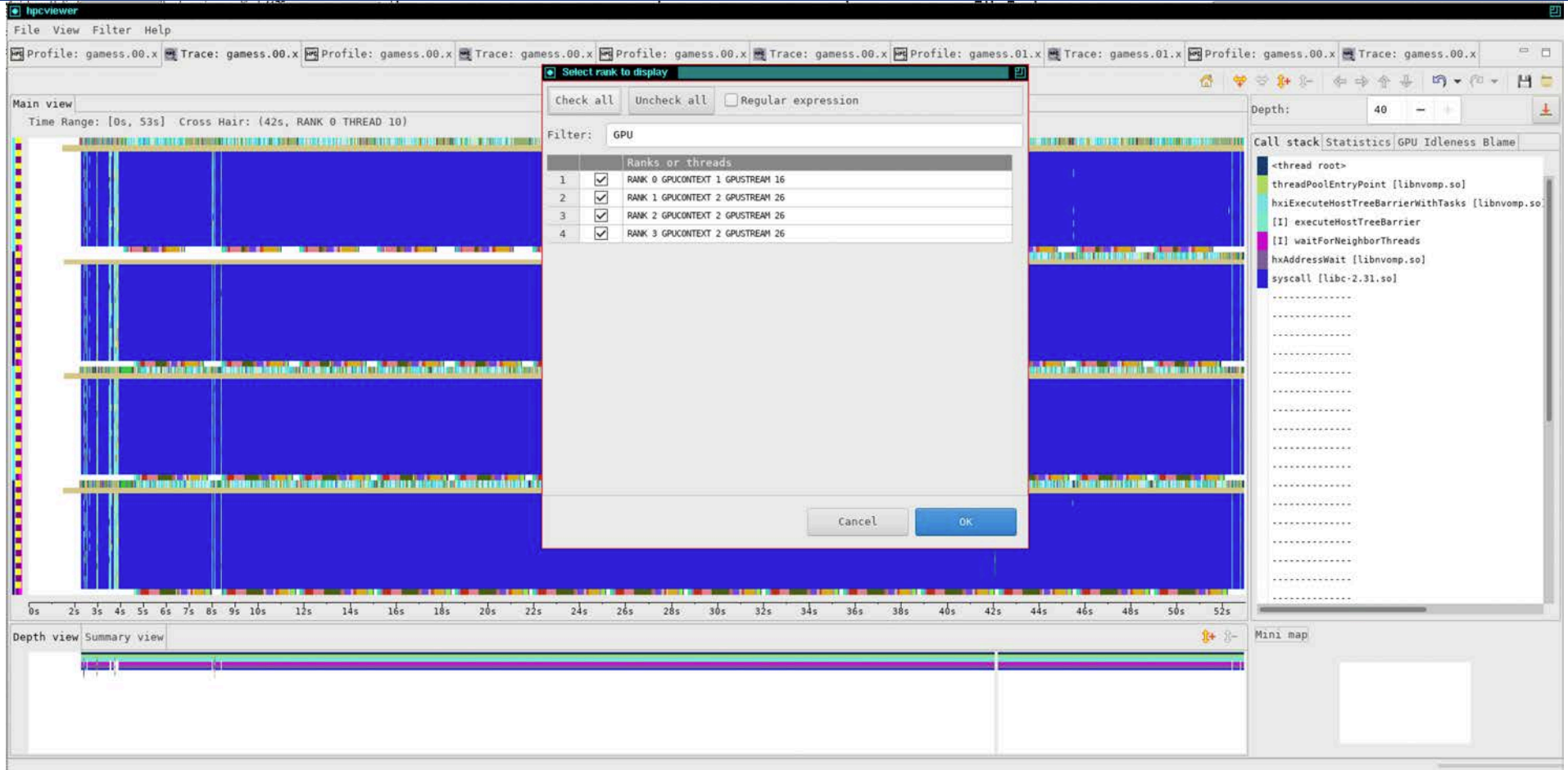
Perlmutter node at a glance

AMD Milan CPU
4 NVIDIA A100 GPUs
256 GB memory

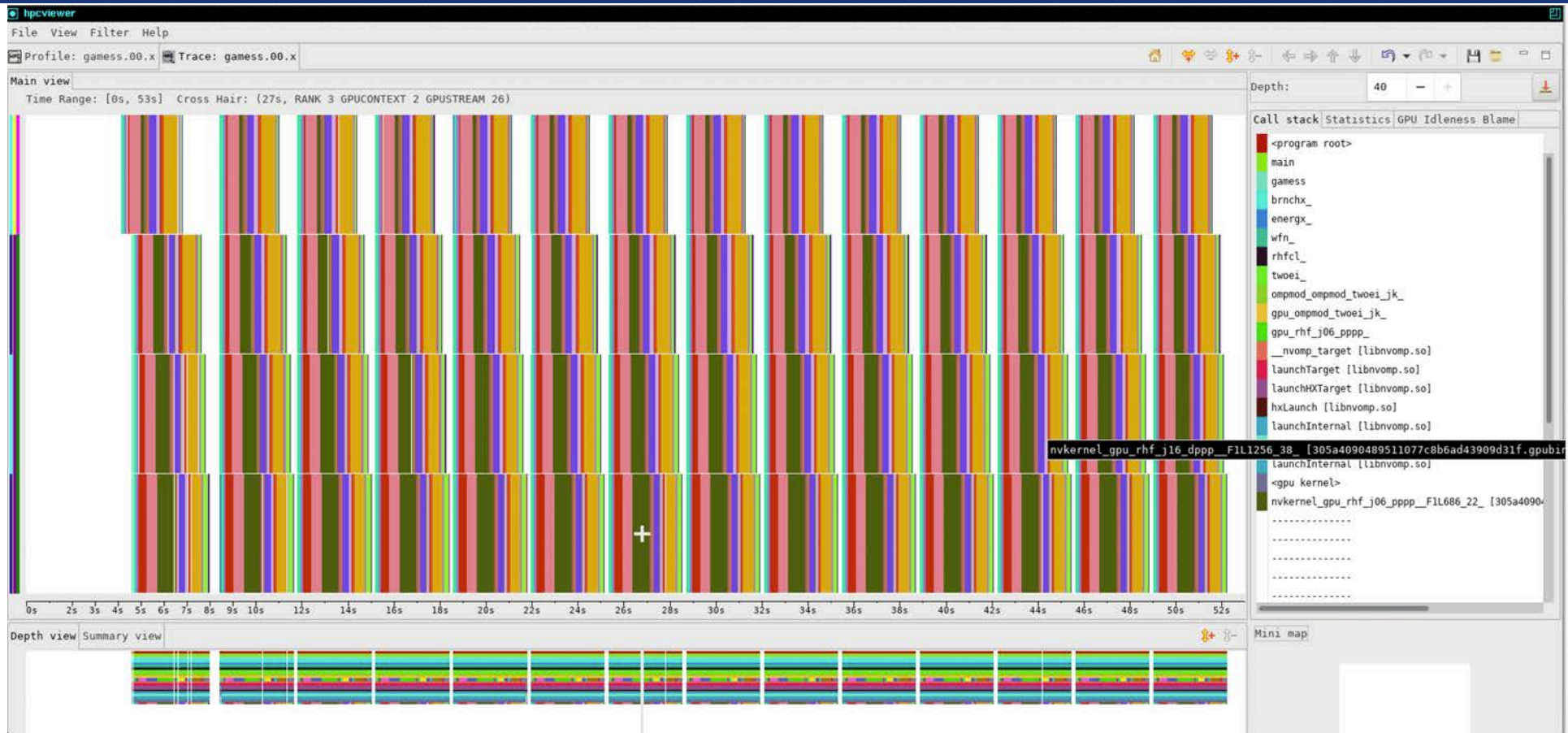
Time-centric Analysis: GAMESS 4 ranks, 4 GPUs on Perlmutter



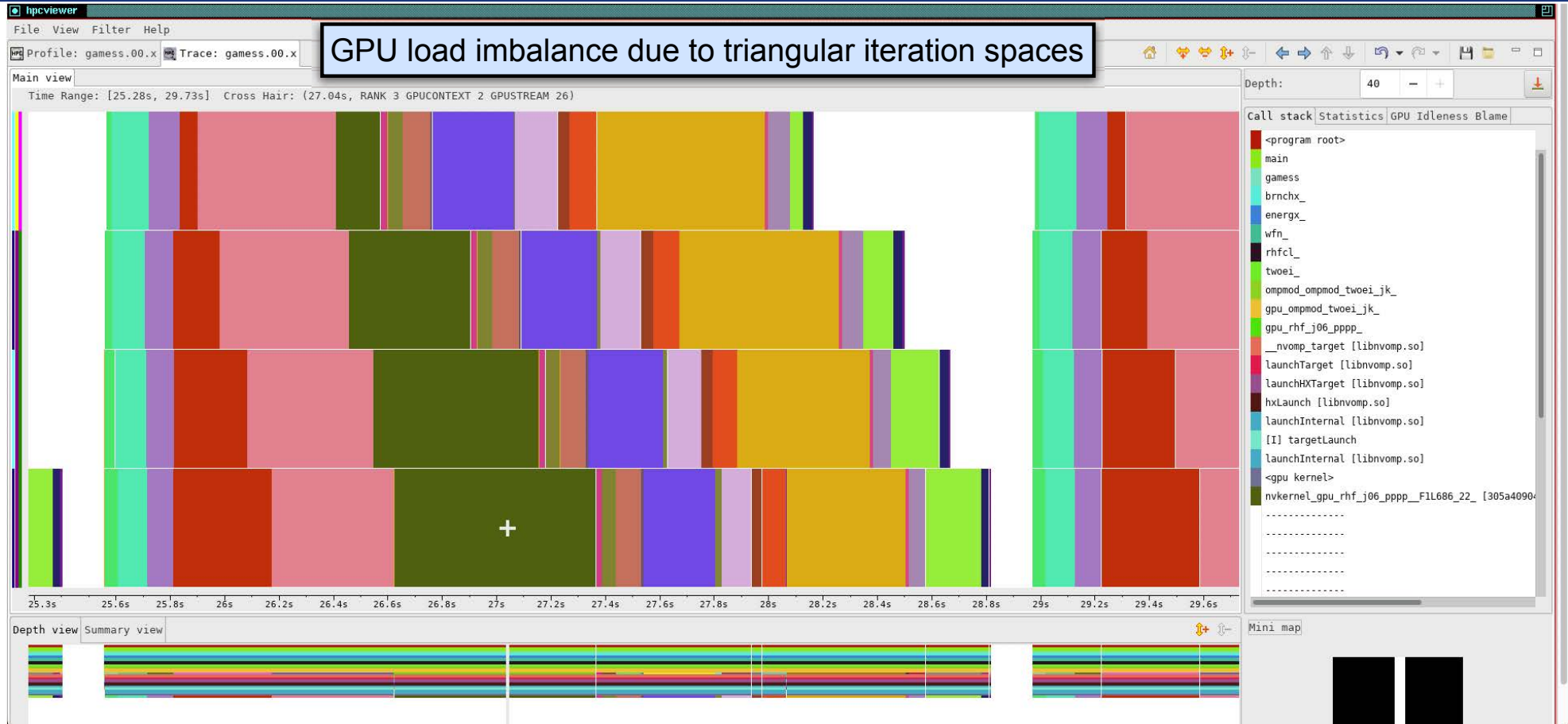
Time-centric Analysis: GAMESS 4 ranks, 4 GPUs on Perlmutter



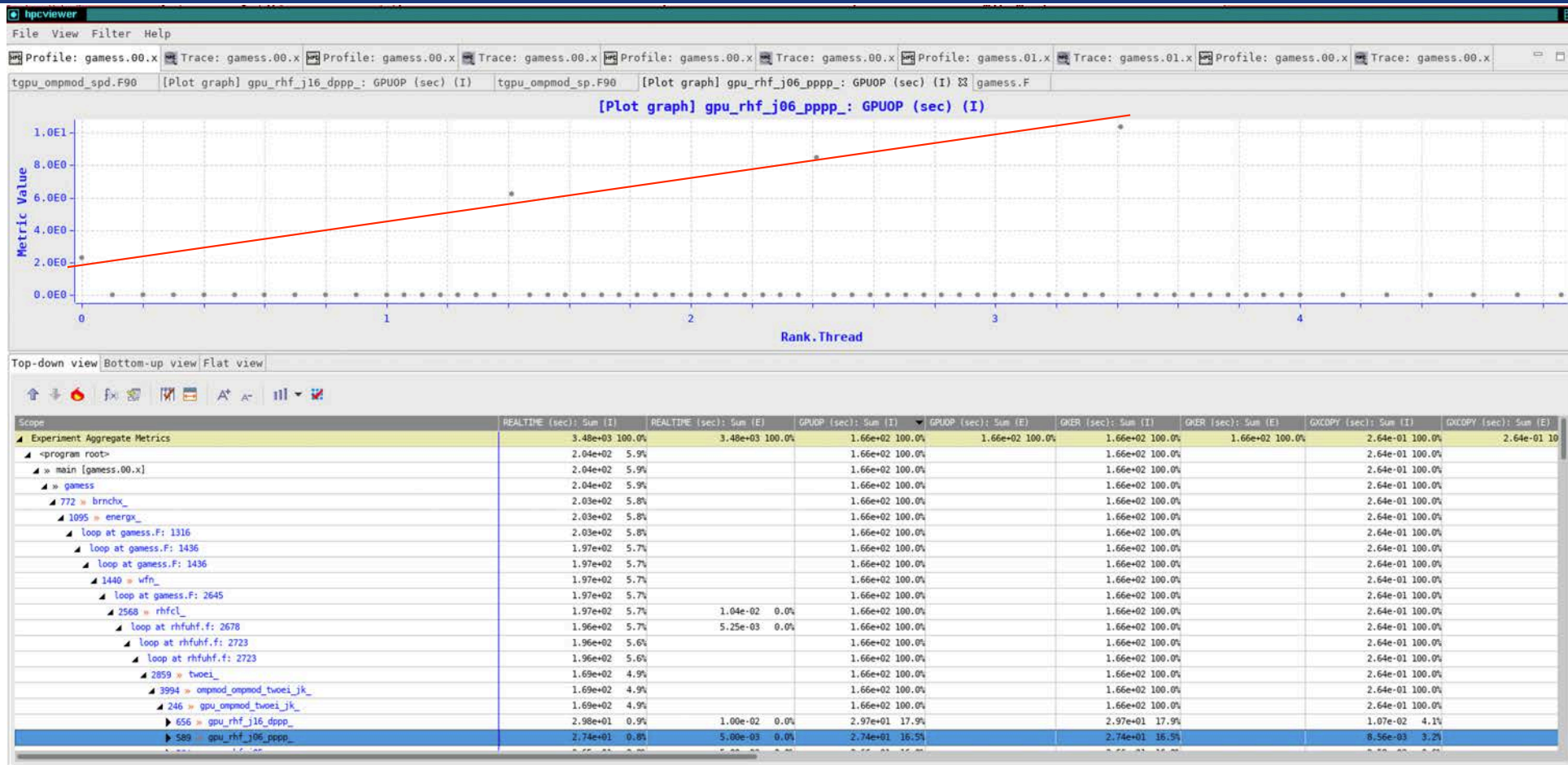
Time-centric Analysis: GAMESS 4 ranks, 4 GPUs on Perlmutter



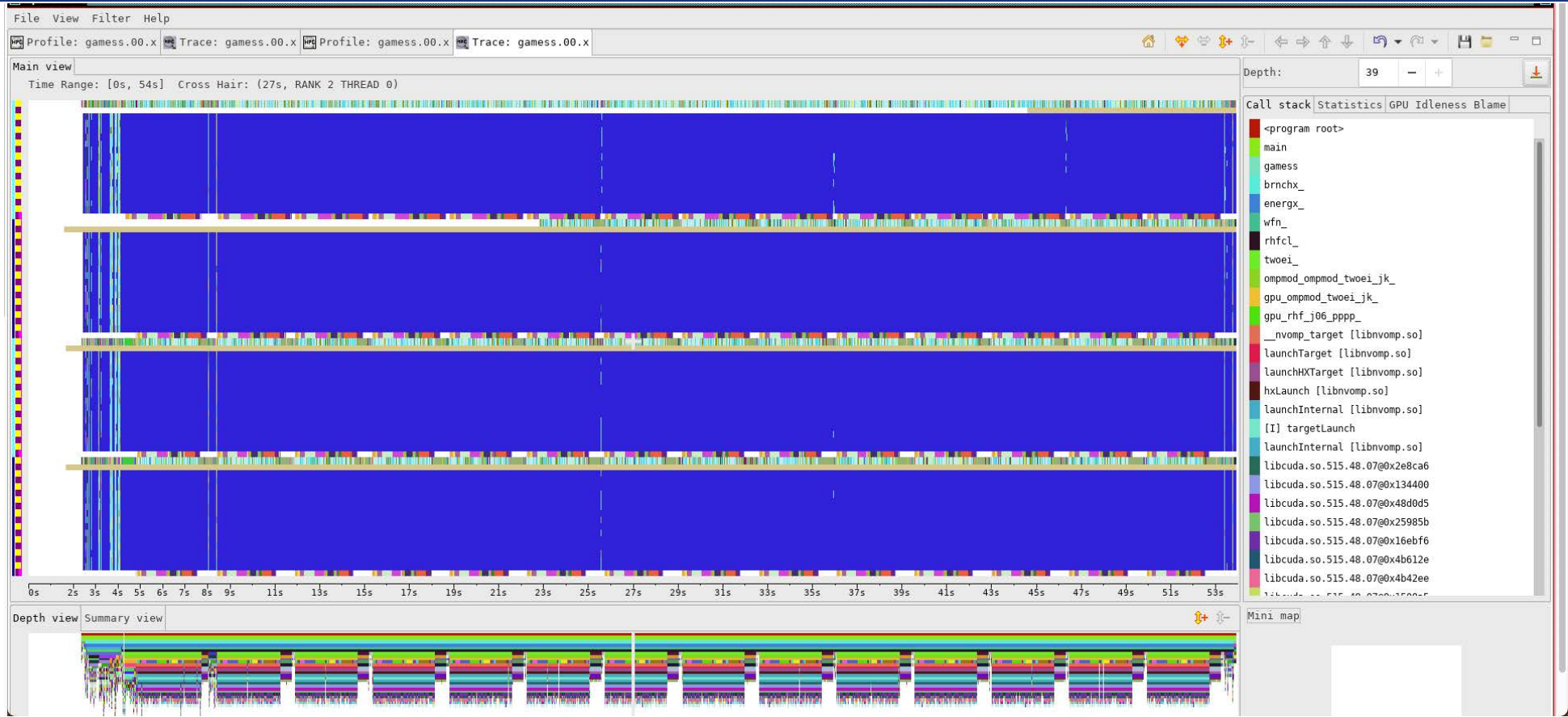
Time-centric Analysis: GAMESS 4 ranks, 4 GPUs on Perlmutter



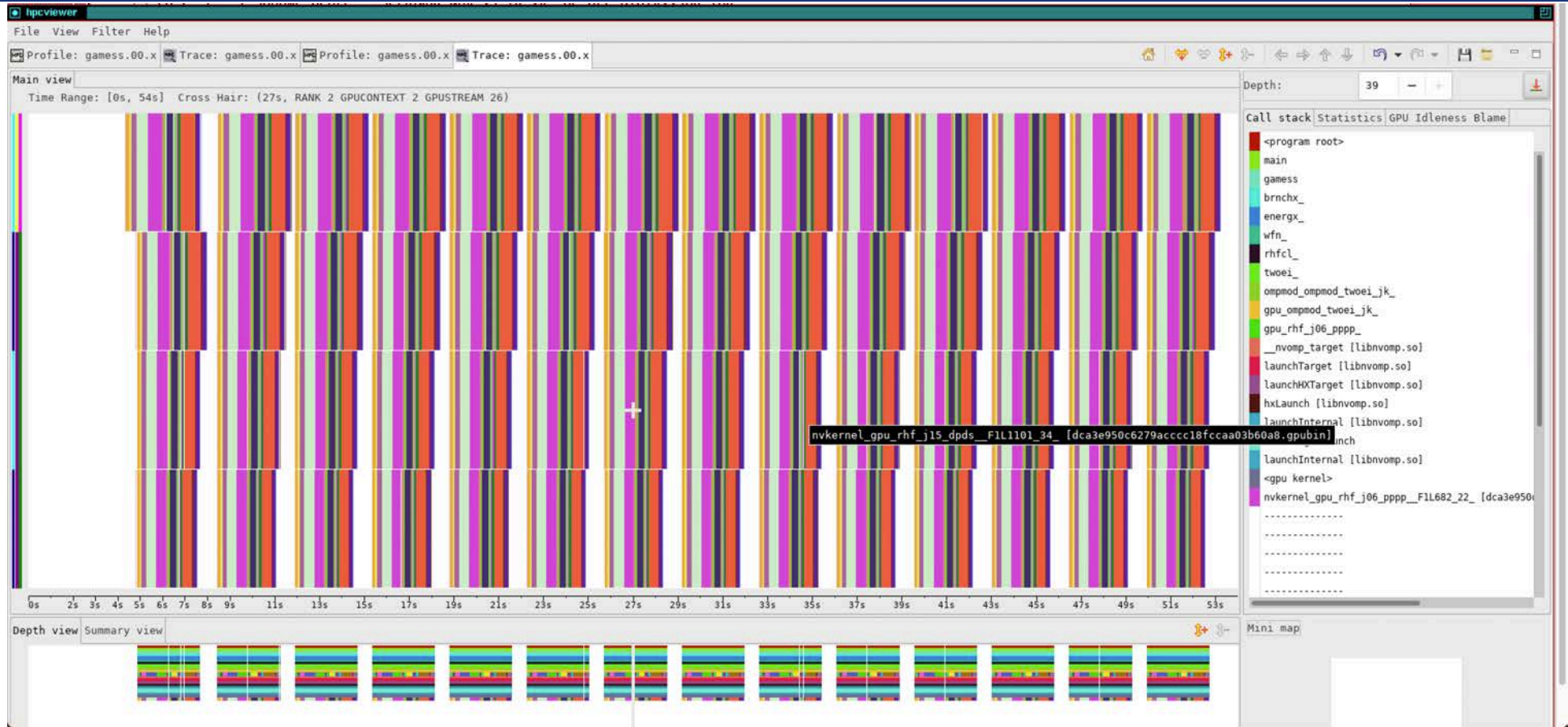
Time-centric Analysis: GAMESS 4 ranks, 4 GPUs on Perlmutter



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



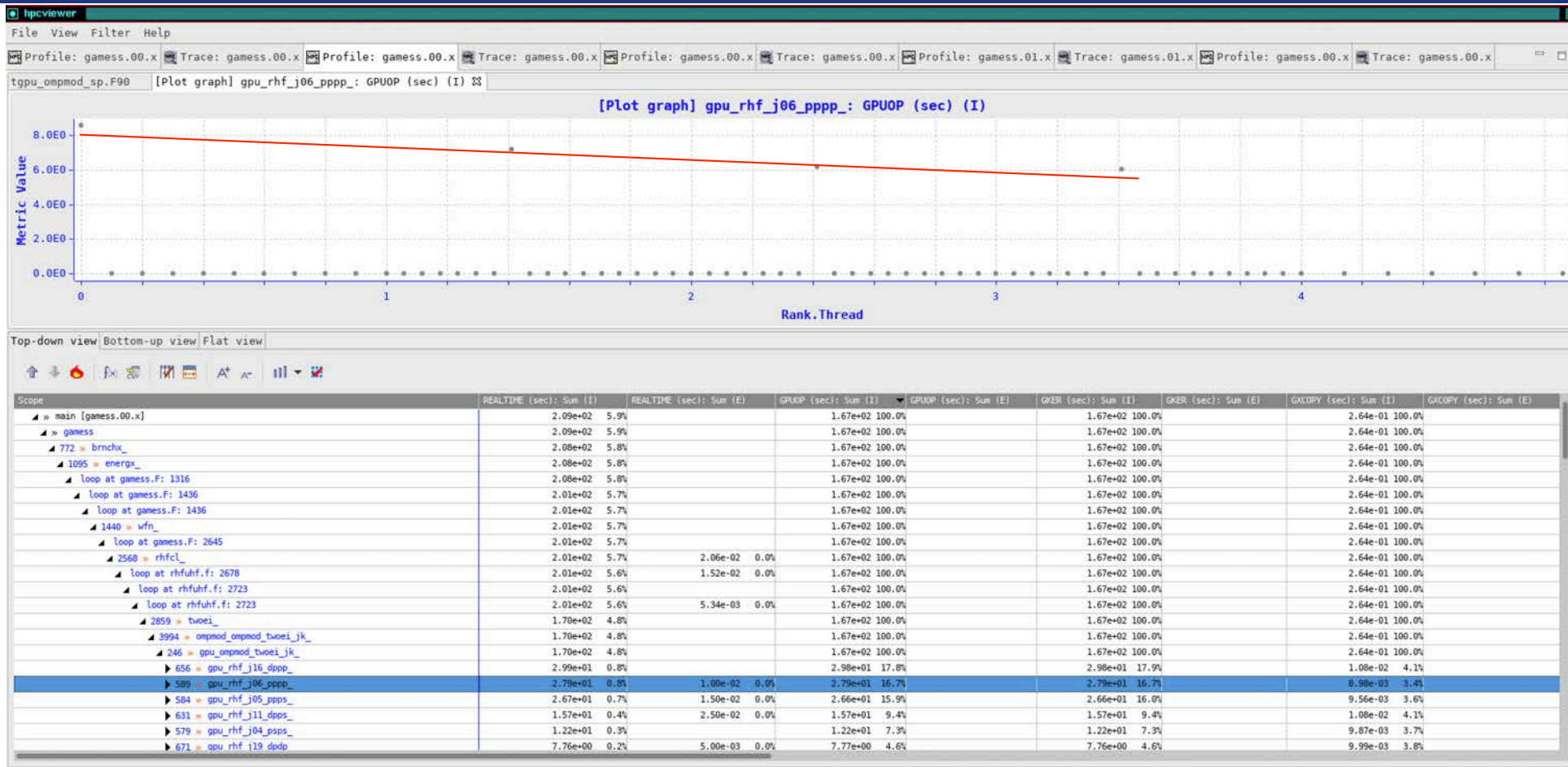
Time-centric Analysis: GAMESS 4 ranks, 4 GPUs on Perlmutter



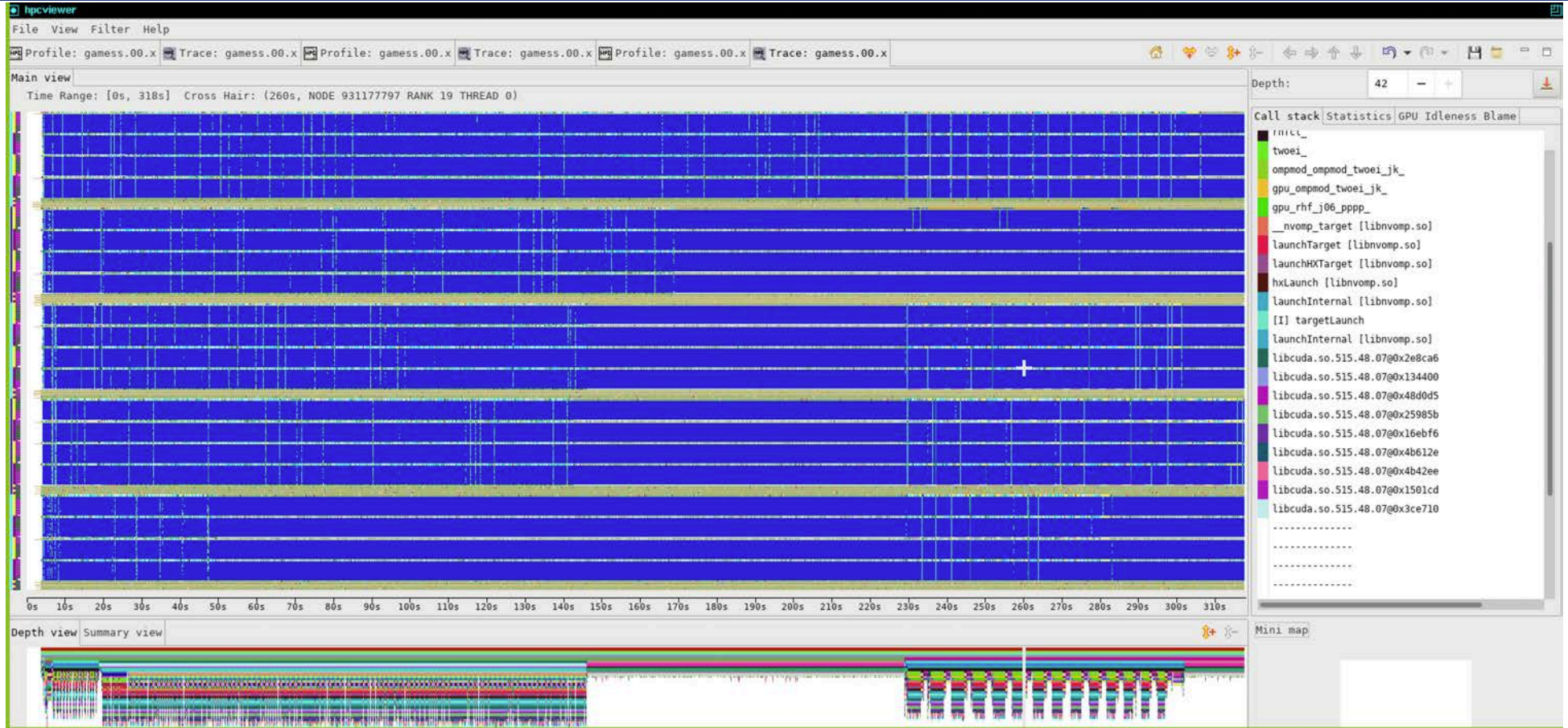
Time-centric Analysis: GAMESS 4 ranks, 4 GPUs on Perlmutter



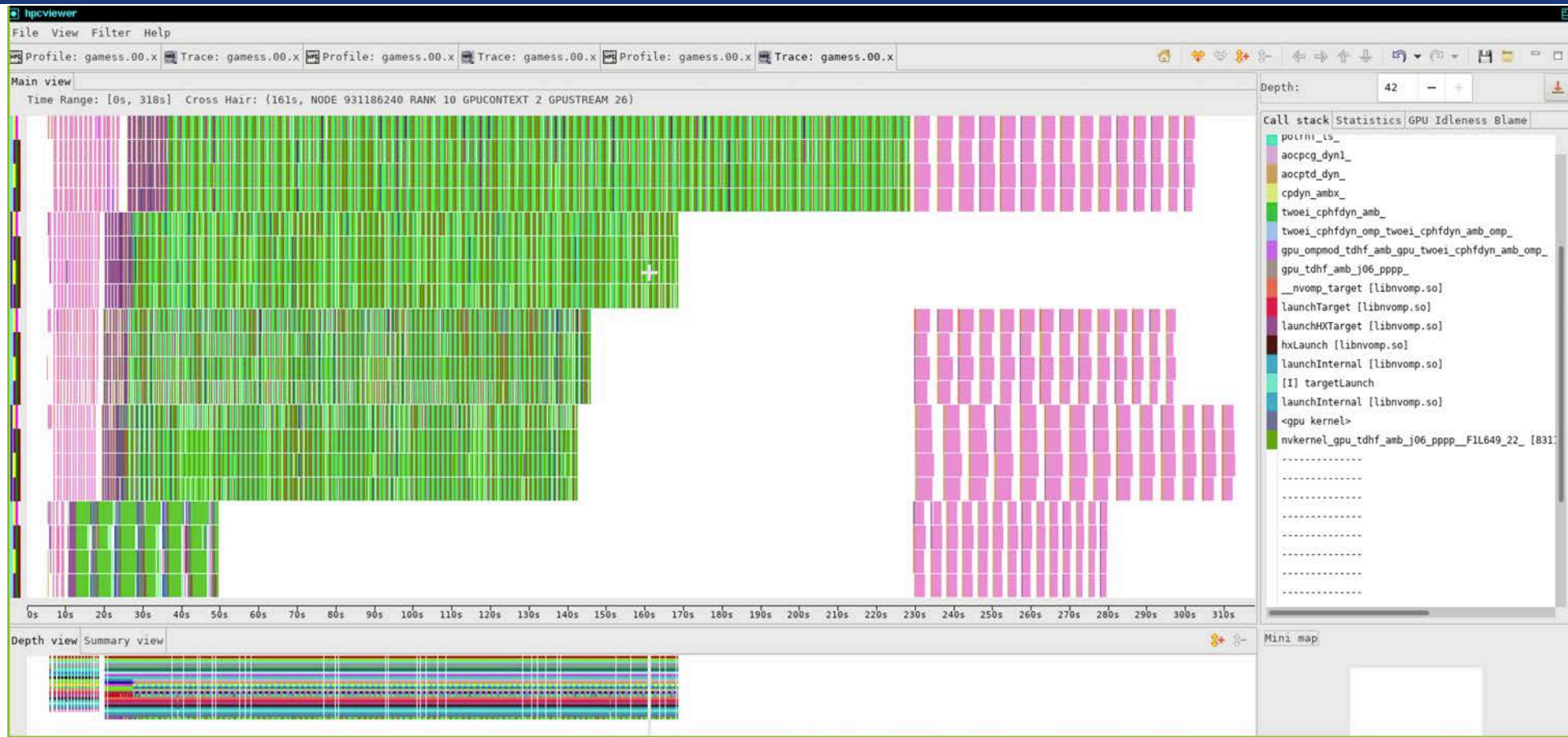
Time-centric Analysis: GAMESS 4 ranks, 4 GPUs on Perlmutter



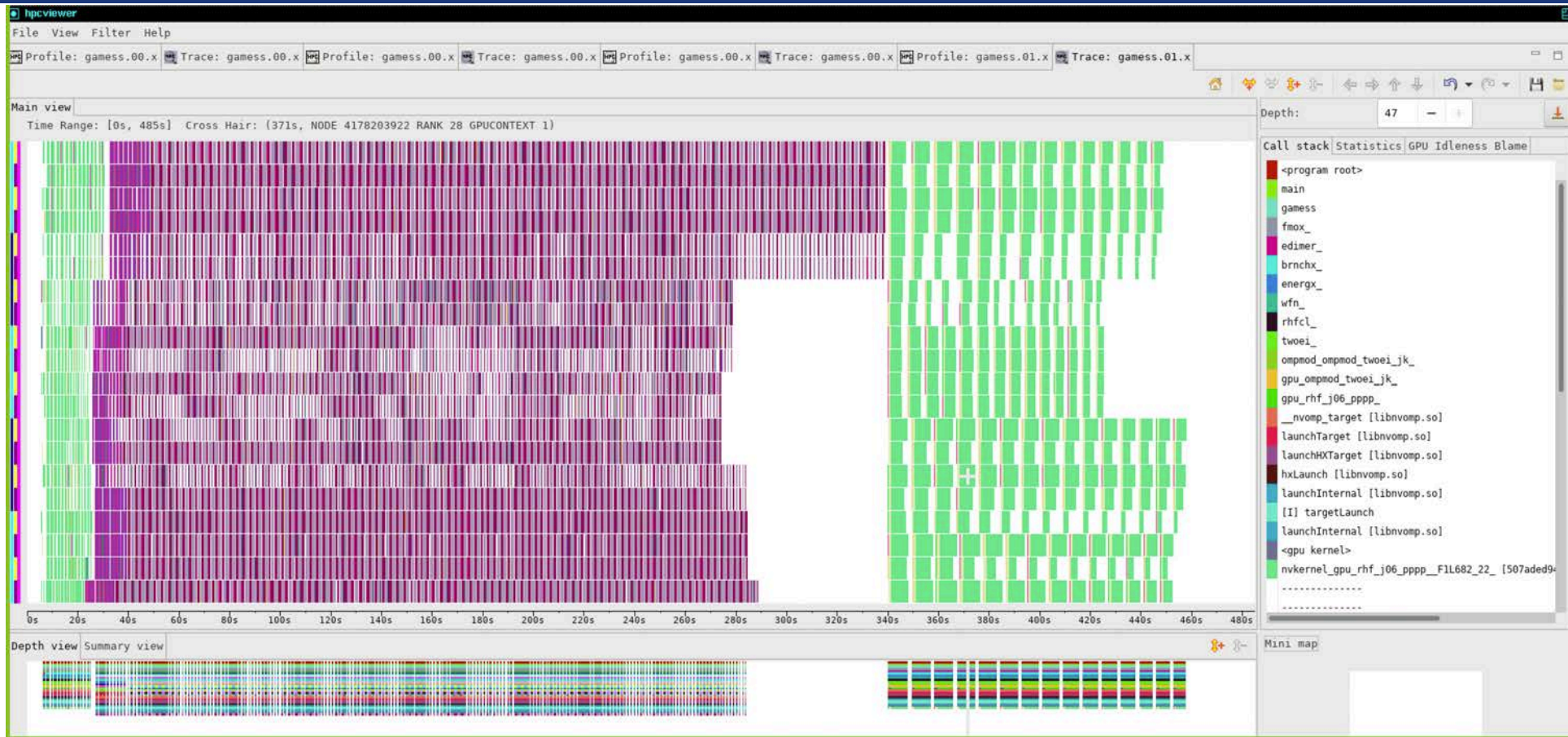
Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter



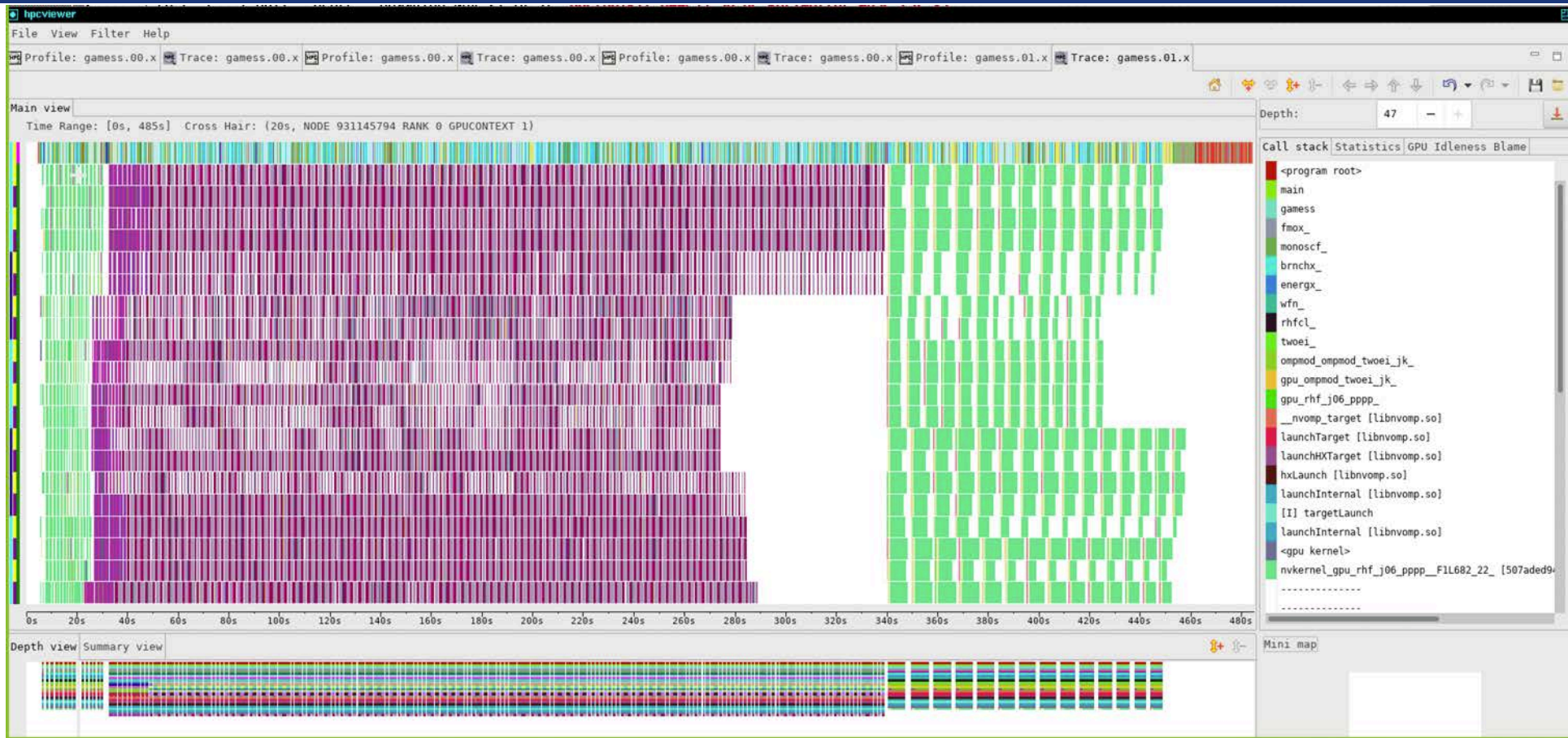
Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter



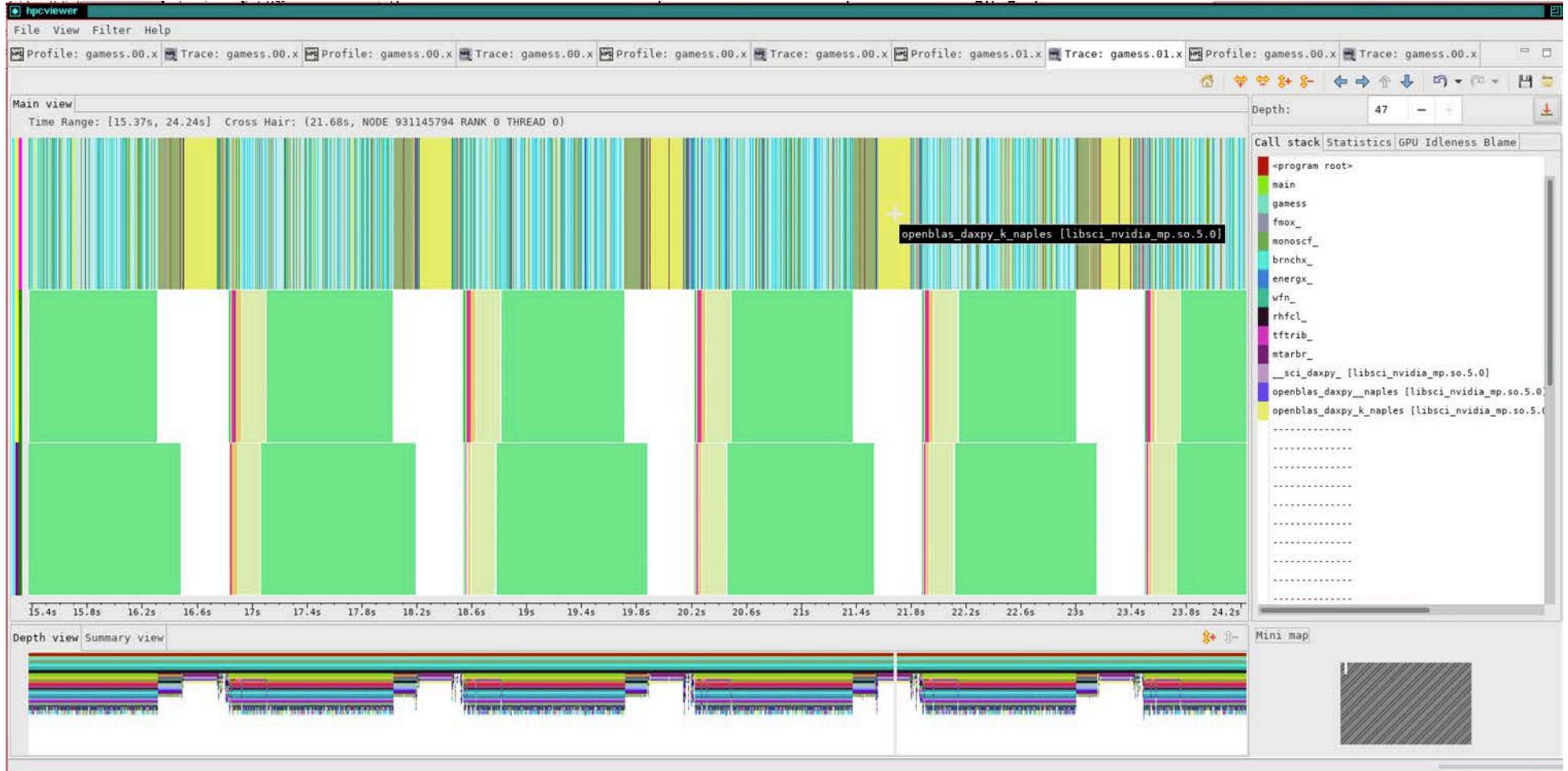
Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter



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



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

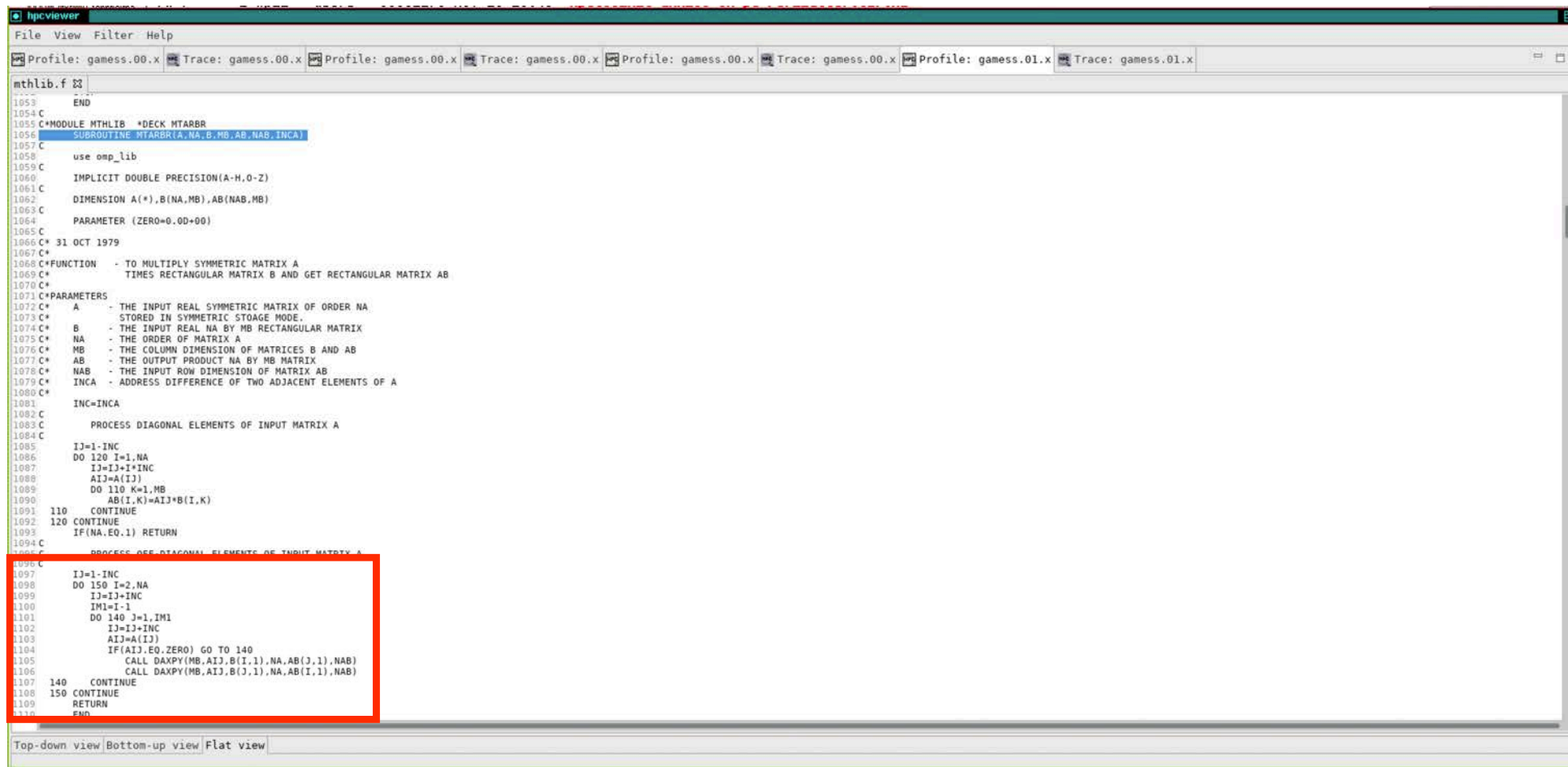


1 CPU Stream, 2 GPU Streams: 6 Iterations

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

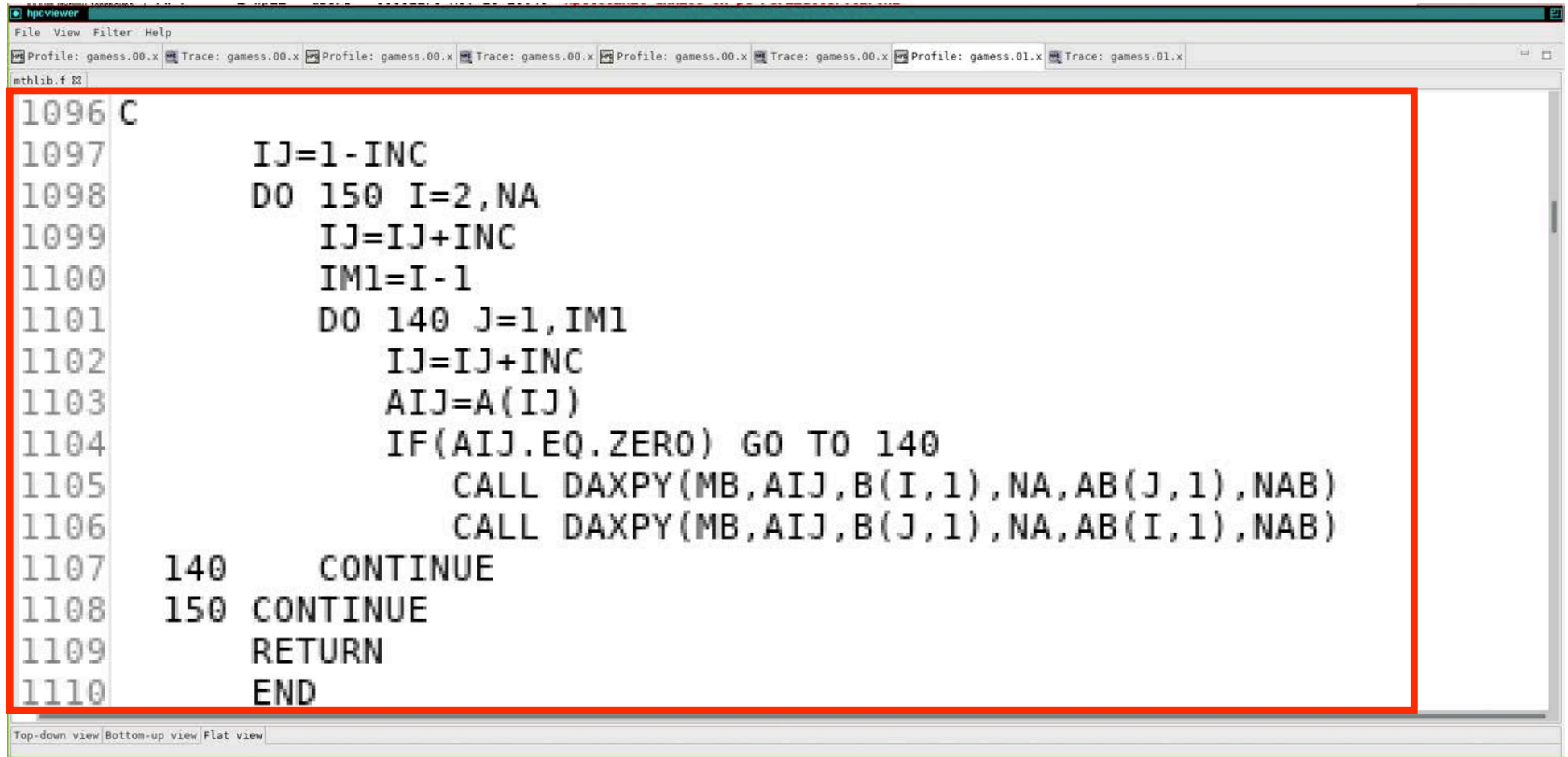
```
hpcviewer
File View Filter Help
Profile: gamess.00.x Trace: gamess.00.x Profile: gamess.00.x Trace: gamess.00.x Profile: gamess.01.x Trace: gamess.01.x
mthlib.f
1053      END
1054 C
1055 C*MODULE MTHLIB *DECK MTARBR
1056      SUBROUTINE MTARBR(A,NA,B,MB,AB,NAB,INCA)
1057 C
1058      use omp_lib
1059 C
1060      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
1061 C
1062      DIMENSION A(*),B(NA,MB),AB(NAB,MB)
1063 C
1064      PARAMETER (ZERO=0.0D+00)
1065 C
1066 C* 31 OCT 1979
1067 C*
1068 C*FUNCTION - TO MULTIPLY SYMMETRIC MATRIX A
1069 C*          TIMES RECTANGULAR MATRIX B AND GET RECTANGULAR MATRIX AB
1070 C*
1071 C*PARAMETERS
1072 C*  A - THE INPUT REAL SYMMETRIC MATRIX OF ORDER NA
1073 C*      STORED IN SYMMETRIC STORAGE MODE.
1074 C*  B - THE INPUT REAL NA BY MB RECTANGULAR MATRIX
1075 C*  NA - THE ORDER OF MATRIX A
1076 C*  MB - THE COLUMN DIMENSION OF MATRICES B AND AB
1077 C*  AB - THE OUTPUT PRODUCT NA BY MB MATRIX
1078 C*  NAB - THE INPUT ROW DIMENSION OF MATRIX AB
1079 C*  INCA - ADDRESS DIFFERENCE OF TWO ADJACENT ELEMENTS OF A
1080 C*
1081      INC=INCA
1082 C
1083 C          PROCESS DIAGONAL ELEMENTS OF INPUT MATRIX A
1084 C
1085      IJ=1-INC
1086      DO 120 I=1,NA
1087          IJ=IJ+INC
1088          AIJ=A(IJ)
1089          DO 110 K=1,MB
1090              AB(I,K)=AIJ*B(I,K)
1091      110      CONTINUE
1092      120      CONTINUE
1093 C
1094 C
1095      IJ=1-INC
1096      DO 150 I=2,NA
1097          IJ=IJ+INC
1098          IM1=I-1
1099          DO 140 J=1,IM1
1100              IJ=IJ+INC
1101              AIJ=A(IJ)
1102              IF(AIJ.EQ.ZERO) GO TO 140
1103              CALL DAXPY(MB,AIJ,B(I,1),NA,AB(J,1),NAB)
1104              CALL DAXPY(MB,AIJ,B(J,1),NA,AB(I,1),NAB)
1105      140      CONTINUE
1106      150      CONTINUE
1107      RETURN
1108      END
1109      150      CONTINUE
1110      RETURN
1111      END
Top-down view Bottom-up view Flat view
```


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.01.x Trace: gamess.01.x
mthlib.f
1053      END
1054 C
1055 C*MODULE MTHLIB *DECK MTARBR
1056      SUBROUTINE MTARBR(A,NA,B,MB,AB,NAB,INCA)
1057 C
1058      use omp_lib
1059 C
1060      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
1061 C
1062      DIMENSION A(*),B(NA,MB),AB(NAB,MB)
1063 C
1064      PARAMETER (ZERO=0.0D+00)
1065 C
1066 C* 31 OCT 1979
1067 C*
1068 C*FUNCTION - TO MULTIPLY SYMMETRIC MATRIX A
1069 C*          TIMES RECTANGULAR MATRIX B AND GET RECTANGULAR MATRIX AB
1070 C*
1071 C*PARAMETERS
1072 C*  A - THE INPUT REAL SYMMETRIC MATRIX OF ORDER NA
1073 C*    STORED IN SYMMETRIC STORAGE MODE.
1074 C*  B - THE INPUT REAL NA BY MB RECTANGULAR MATRIX
1075 C*  NA - THE ORDER OF MATRIX A
1076 C*  MB - THE COLUMN DIMENSION OF MATRICES B AND AB
1077 C*  AB - THE OUTPUT PRODUCT NA BY MB MATRIX
1078 C*  NAB - THE INPUT ROW DIMENSION OF MATRIX AB
1079 C*  INCA - ADDRESS DIFFERENCE OF TWO ADJACENT ELEMENTS OF A
1080 C*
1081      INC=INCA
1082 C
1083 C          PROCESS DIAGONAL ELEMENTS OF INPUT MATRIX A
1084 C
1085      IJ=1-INC
1086      DO 120 I=1,NA
1087          IJ=IJ+INC
1088          AIJ=A(IJ)
1089          DO 110 K=1,MB
1090              AB(I,K)=AIJ*B(I,K)
1091      110      CONTINUE
1092      120      CONTINUE
1093      IF(NA.EQ.1) RETURN
1094 C
1095 C          PROCESS OFF-DIAGONAL ELEMENTS OF INPUT MATRIX A
1096 C
1097      IJ=1-INC
1098      DO 150 I=2,NA
1099          IJ=IJ+INC
1100          IM1=I-1
1101          DO 140 J=1,IM1
1102              IJ=IJ+INC
1103              AIJ=A(IJ)
1104              IF(AIJ.EQ.ZERO) GO TO 140
1105              CALL DAXPY(MB,AIJ,B(I,1),NA,AB(J,1),NAB)
1106              CALL DAXPY(MB,AIJ,B(J,1),NA,AB(I,1),NAB)
1107      140      CONTINUE
1108      150      CONTINUE
1109      RETURN
1110      END
Top-down view Bottom-up view Flat view
```

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



The image shows a screenshot of the hpcviewer application window. The window title is 'hpcviewer' and it has a menu bar with 'File', 'View', 'Filter', and 'Help'. The window contains several tabs, including 'Profile: gamess.00.x' and 'Trace: gamess.00.x'. The main content area displays the source code for 'mthlib.f'. The code is enclosed in a red rectangular box. The code consists of several lines of Fortran, including a loop structure with nested DO loops and a RETURN statement.

```
mthlib.f
1096 C
1097     IJ=1-INC
1098     DO 150 I=2,NA
1099         IJ=IJ+INC
1100         IM1=I-1
1101         DO 140 J=1,IM1
1102             IJ=IJ+INC
1103             AIJ=A(IJ)
1104             IF(AIJ.EQ.ZERO) GO TO 140
1105                 CALL DAXPY(MB,AIJ,B(I,1),NA,AB(J,1),NAB)
1106                 CALL DAXPY(MB,AIJ,B(J,1),NA,AB(I,1),NAB)
1107     140     CONTINUE
1108     150 CONTINUE
1109         RETURN
1110     END
```

At the bottom of the window, there are view options: 'Top-down view', 'Bottom-up view', and 'Flat view'.

Measure Pytorch Deepwave (RTM) on CPU + NVIDIA GPU

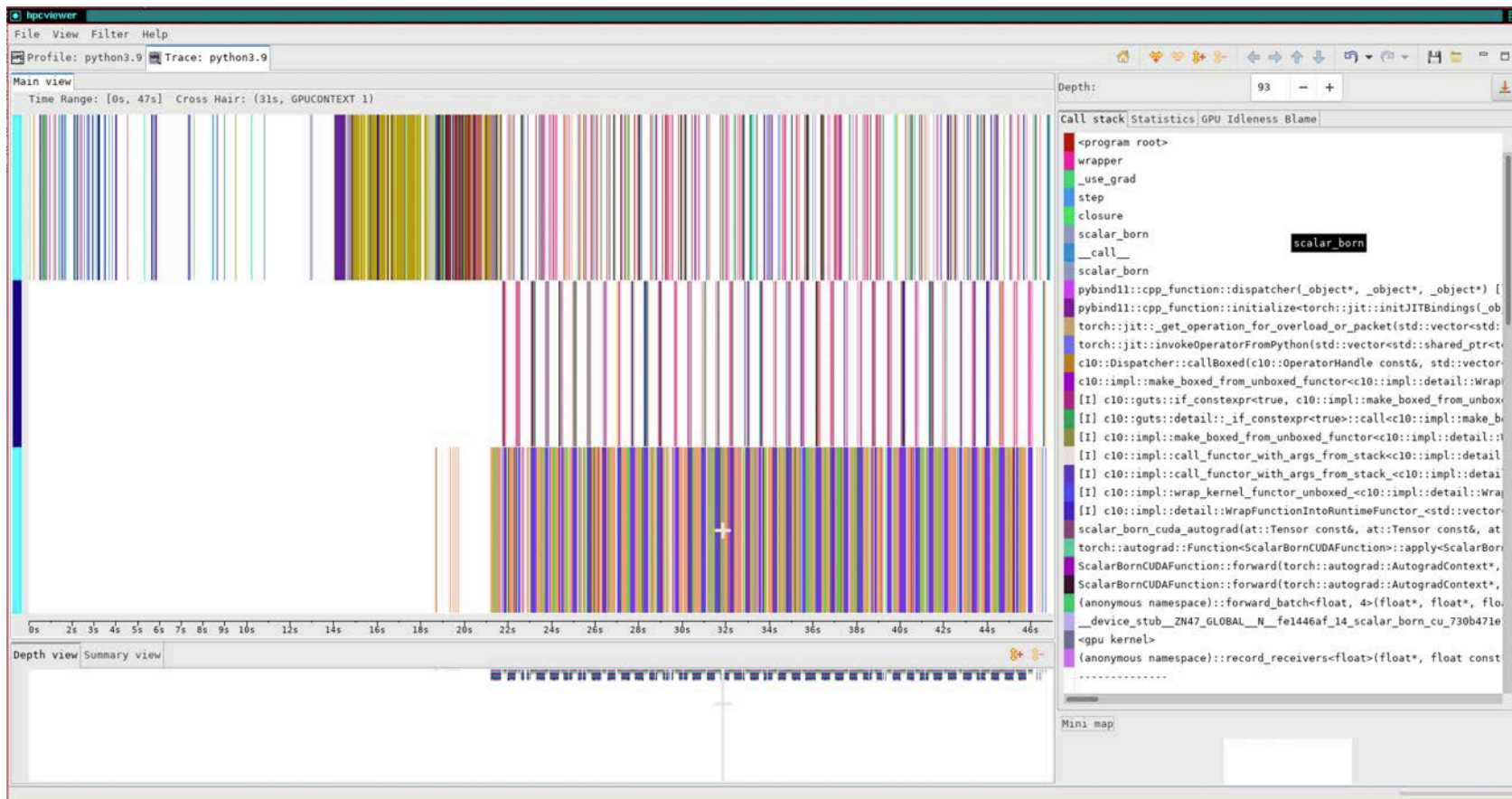
The screenshot shows the nvcviewer interface. The top pane displays the source code for 'rtm.py', with the 'scalar_born' function definition highlighted. The bottom pane shows the execution profile in 'Flat view' mode. The profile table includes columns for CPUTIME (sec), GINS, and GINS:STL_ANY. The 'scalar_born' function is highlighted in blue, and its sub-calls are enclosed in a blue box. A red box highlights the 'Python calls' section of the profile.

Scope	CPUTIME (sec): Sum (I)	CPUTIME (sec): Sum (E)	GINS: Sum (I)	GINS: Sum (E)	GINS:STL_ANY: Sum (I)	GINS:STL_ANY: Sum (E)	GINS:STL_IFET: Sum
Experiment Aggregate Metrics	3.91e+02 100.0%	3.91e+02 100.0%	2.70e+12 100.0%	2.70e+12 100.0%	1.95e+12 100.0%	1.95e+12 100.0%	2.23e+10
Program root	2.15e+02 55.1%		1.75e+12 64.9%		1.41e+12 72.5%		9.69e+09
152 » wrapper	2.05e+02 52.5%		1.75e+12 64.9%		1.41e+12 72.5%		9.69e+09
use_grad	2.05e+02 52.5%		1.75e+12 64.9%		1.41e+12 72.5%		9.69e+09
step	2.05e+02 52.5%		1.75e+12 64.9%		1.41e+12 72.5%		9.69e+09
closure	2.05e+02 52.5%	3.58e-02 0.0%	1.75e+12 64.9%		1.41e+12 72.5%		9.69e+09
scalar_born	2.05e+02 52.5%		1.75e+12 64.9%		1.41e+12 72.5%		9.69e+09
call	2.03e+02 51.8%		1.75e+12 64.9%		1.41e+12 72.5%		9.68e+09
scalar_born [python.6884567]	2.03e+02 51.8%		1.75e+12 64.9%		1.41e+12 72.5%		9.68e+09
pybind11::cpp_function::dispatcher(object*, object*, object*) [...]	2.03e+02 51.8%		1.75e+12 64.9%		1.41e+12 72.5%		9.68e+09
loop at [libtorch_python.sol]: 0	2.03e+02 51.8%		1.75e+12 64.9%		1.41e+12 72.5%		9.68e+09
pybind11::cpp_function::initialize(torch::jit::initJITBindings(...))	2.03e+02 51.8%		1.75e+12 64.9%		1.41e+12 72.5%		9.68e+09
torch::jit::get_operation_for_overload_or_packet(std::vector<...>)	2.03e+02 51.8%		1.75e+12 64.9%		1.41e+12 72.5%		9.68e+09
torch::jit::invokeOperatorFromPython(std::vector<std::shared_ptr<...>>)	2.03e+02 51.8%		1.75e+12 64.9%		1.41e+12 72.5%		9.68e+09
c10::Dispatcher::callBoxed(c10::OperatorHandle const&, std::vector<...>)	2.03e+02 51.8%		1.75e+12 64.9%		1.41e+12 72.5%		9.68e+09
c10::impl::make_boxed_from_unboxed_funcctor<c10::impl::detail::...>	2.03e+02 51.8%		1.75e+12 64.9%		1.41e+12 72.5%		9.68e+09
575 » [I] c10::guts::if_constexpr_true, c10::impl::make_boxed...	2.03e+02 51.8%		1.75e+12 64.9%		1.41e+12 72.5%		9.68e+09
368 » [I] c10::guts::detail::_if_constexpr_true::call<c10::...>	2.03e+02 51.8%		1.75e+12 64.9%		1.41e+12 72.5%		9.68e+09
239 » [I] c10::impl::make_boxed_from_unboxed_funcctor<c10::...>	2.03e+02 51.8%		1.75e+12 64.9%		1.41e+12 72.5%		9.68e+09

Python calls

Pytorch-generated code

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



Analyze Pytorch Deepwave (RTM) Kernel using PC Sampling

The screenshot displays the hpcviewer interface. The top pane shows the source code for `scalar_born.py` and `scalar_born.cu`. The bottom pane shows a performance table with columns for Scope, GINS: Sum (I), GINS:STL_ANY: Sum (I), GINS:STL_OHEM: Sum (E), and GSAMP:UTIL (I).

High GPU utilization for forward_kernel
Many instruction stalls on global memory

Scope	GINS: Sum (I)	GINS:STL_ANY: Sum (I)	GINS:STL_OHEM: Sum (E)	GSAMP:UTIL (I)
◀ c10::impl::make_boxed_from_unboxed_funcor<c10::impl::detail::WrapFunctionIntoRuntimeFuncor_<std::vector<at::Tensor, std::al...	1.75e+12 64.9%	1.41e+12 72.5%		83.58 %
◀ 575 » [I] c10::guts::if_constexpr<true, c10::impl::make_boxed_from_unboxed_funcor<c10::impl::detail::WrapFunctionIntoRuntimeF...	1.75e+12 64.9%	1.41e+12 72.5%		83.58 %
◀ 368 » [I] c10::guts::detail::if_constexpr<true>::call<c10::impl::make_boxed_from_unboxed_funcor<c10::impl::detail::WrapFun...	1.75e+12 64.9%	1.41e+12 72.5%		83.58 %
◀ 239 » [I] c10::impl::make_boxed_from_unboxed_funcor<c10::impl::detail::WrapFunctionIntoRuntimeFuncor_<std::vector<at::Ten...	1.75e+12 64.9%	1.41e+12 72.5%		83.58 %
◀ 584 » [I] c10::impl::call_funcor_with_args_from_stack<c10::impl::detail::WrapFunctionIntoRuntimeFuncor_<std::vector<at::...	1.75e+12 64.9%	1.41e+12 72.5%		83.58 %
◀ 511 » [I] c10::impl::call_funcor_with_args_from_stack<c10::impl::detail::WrapFunctionIntoRuntimeFuncor_<std::vector<at...	1.75e+12 64.9%	1.41e+12 72.5%		83.58 %
◀ 499 » [I] c10::impl::wrap_kernel_funcor_unboxed<c10::impl::detail::WrapFunctionIntoRuntimeFuncor_<std::vector<at::Te...	1.75e+12 64.9%	1.41e+12 72.5%		83.58 %
◀ 461 » [I] c10::impl::detail::WrapFunctionIntoRuntimeFuncor_<std::vector<at::Tensor, std::allocator<at::Tensor> > (*)(&...	1.75e+12 64.9%	1.41e+12 72.5%		83.58 %
◀ 18 » scalar_born_cuda_autograd(at::Tensor const&, at::Tensor const&, at::Tensor const&, at::Tensor const&, at::Tensor...	1.75e+12 64.9%	1.41e+12 72.5%		83.58 %
◀ torch::autograd::Function<ScalarBornCUDAFunction>::apply<ScalarBornCUDAFunction, at::Tensor const&, at::Tensor con...	1.75e+12 64.9%	1.41e+12 72.5%		83.58 %
◀ ScalarBornCUDAFunction::forward(torch::autograd::AutogradContext*, at::Tensor const&, at::Tensor const&, at::Ten...	1.75e+12 64.9%	1.41e+12 72.5%		83.58 %
◀ ScalarBornCUDAFunction::forward(torch::autograd::AutogradContext*, at::Tensor const&, at::Tensor const&, at::Te...	1.75e+12 64.9%	1.41e+12 72.5%		83.60 %
◀ (anonymous namespace)::forward_batch<float, 4>(float*, float*, float*, float*, float*, float*, float*, float*,...	1.75e+12 64.9%	1.41e+12 72.5%		83.61 %
◀ loop at [deepwave.so]: 0	1.75e+12 64.9%	1.41e+12 72.5%		83.61 %
◀ _device_stub_ZN47_GLOBAL_N_fe1446af_14_scalar_born_cu_730b471e14forward_kernelIfLi4ELb0LE0EEVPKT_PS1_...	1.44e+12 53.3%	1.16e+12 59.3%		89.50 %
◀ <gpu kernel>	1.44e+12 53.3%	1.16e+12 59.3%		89.50 %
◀ (anonymous namespace)::forward_kernel<float, 4, false, false>(float const*, float*, float const*, float c...	1.44e+12 53.3%	1.16e+12 59.3%	8.89e+11 59.7	89.50 %
scalar_born.cu: 264	3.74e+11 13.9%	3.28e+11 16.8%	3.15e+11 21.1%	
scalar_born.cu: 262	3.63e+11 13.5%	3.12e+11 16.0%	2.65e+11 17.8%	
scalar_born.cu: 151	2.74e+11 10.1%	2.15e+11 11.0%	1.80e+11 12.1%	

Case Study: Quicksilver

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

Quicksilver: Detailed Analysis within a Kernel using PC Sampling

The screenshot displays the hpcviewer application. The top pane shows the source code for CollisionEvent.cc, with lines 73-74 highlighted in blue. The bottom pane shows a performance table with columns for various metrics and a scope tree on the left.

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 98.8%	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, 1...	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%	3.44e+05 0.0%	3.44e+05 0.0%	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
  ▶ 252 » [I] qs_vector<NuclearDataReaction>::operator[](int)
```

Analysis of PeleC using PC Sampling on an NVIDIA GPU

The image shows a screenshot of the hpcviewer application. The top window displays the source code for reactor.cpp, with a blue arrow pointing from a 'GPU context' label to a lambda capture in a loop. The code snippet is as follows:

```
441 #ifdef AMREX_USE_GPU
442 const auto ec = amrex::Gpu::ExecutionConfig(udata->ncells_d);
443 amrex::launch_global<<<
444   udata->nbBlocks, udata->nbThreads, ec.sharedMem, udata->stream>>>{
445   [=] AMREX_GPU_DEVICE() noexcept {
446     for (int icell = blockDim.x * blockIdx.x + threadIdx.x,
447         stride = blockDim.x * gridDim.x;
448         icell < udata->ncells_d; icell += stride) {
449       fKernelSpec(
450         icell, udata->dt_save, udata->ireactor_type, yvec_d, ydot_d,
451         udata->rhoesrc_ext_d, udata->rhoesrc_ext_d, udata->rhoesrc_ext_d);
452     }
453   });
454 #else
455 for (int icell = 0; icell < udata->ncells_d; icell++) {
456   fKernelSpec(
457     icell, udata->dt_save, udata->ireactor_type, yvec_d, ydot_d,
458     udata->rhoesrc_ext_d, udata->rhoesrc_ext_d, udata->rhoesrc_ext_d);
459 }
460 #endif
```

Two callout boxes provide context:

- Cause:** passed udata structure pointer to lambda capture
- Improvement:** pass udata components as scalars
<https://github.com/AMReX-Combustion/PelePhysics/pull/192>
4% speedup on PeleC PMF drm19 test case

The bottom window shows a performance table with columns for GINS, GINS_STL_ANY, and GINS_STL_GMEM. A blue arrow points from a 'GPU context' label to the 'reactor.cpp: 446' row in the table.

Scope	GINS [0,0] (I)	GINS [0,0] (E)	GINS_STL_ANY [0,0] (I)	GINS_STL_ANY [0,0] (E)	GINS_STL_GMEM [0,0] (I)	GINS_STL_GMEM [0,0] (E)
loop at AMReX_Amr.cpp: 2061	1.24e+13 88.6%		1.05e+13 88.7%		5.58e+12 89.3%	
loop at AMReX_Amr.cpp: 2062	1.24e+13 88.6%		1.05e+13 88.7%		5.58e+12 89.3%	
loop at AMReX_Amr.cpp: 2015	1.24e+13 88.5%		1.05e+13 88.6%		5.57e+12 89.2%	
loop at AMReX_Amr.cpp: 36	1.24e+13 88.5%		1.05e+13 88.6%		5.57e+12 89.2%	
loop at AMReX_Amr.cpp: 302	1.24e+13 88.4%		1.05e+13 88.5%		5.57e+12 89.1%	
loop at AMReX_Amr.cpp: 308	1.24e+13 88.4%		1.05e+13 88.5%		5.57e+12 89.1%	
loop at AMReX_Amr.cpp: 561	9.61e+12 68.5%		8.29e+12 70.0%		4.17e+12 66.8%	
loop at AMReX_Amr.cpp: 109	9.43e+12 67.2%		8.14e+12 68.7%		4.06e+12 65.0%	
loop at AMReX_Amr.cpp: 210	9.39e+12 66.9%		8.10e+12 68.4%		4.03e+12 64.5%	
loop at AMReX_Amr.cpp: 234	9.28e+12 66.2%		8.00e+12 67.6%		3.94e+12 63.1%	
loop at AMReX_Amr.cpp: erkStep_TakeStep	7.16e+12 51.1%		6.19e+12 52.3%		3.05e+12 48.9%	
loop at AMReX_Amr.cpp: cF_RHS	6.27e+12 44.7%		5.49e+12 46.3%		2.48e+12 39.7%	
loop at AMReX_Amr.cpp: 443	6.27e+12 44.7%		5.49e+12 46.3%		2.48e+12 39.7%	
loop at AMReX_Amr.cpp: 12	6.27e+12 44.7%		5.49e+12 46.3%		2.48e+12 39.7%	
loop at AMReX_Amr.cpp: 26	6.27e+12 44.7%		5.49e+12 46.3%		2.48e+12 39.7%	
loop at AMReX_Amr.cpp: 24	6.27e+12 44.7%		5.49e+12 46.3%		2.48e+12 39.7%	
loop at AMReX_Amr.cpp: 211	6.27e+12 44.7%		5.49e+12 46.3%		2.48e+12 39.7%	
loop at AMReX_Amr.cpp: cgpu kernel	6.27e+12 44.7%		5.49e+12 46.3%		2.48e+12 39.7%	
loop at AMReX_Amr.cpp: 12	6.27e+12 44.7%	1.75e+10 0.1%	5.49e+12 46.3%	1.70e+10 0.1%	2.48e+12 39.7%	
loop at AMReX_Amr.cpp: 12	6.25e+12 44.6%	1.17e+12 8.3%	5.47e+12 46.2%	1.16e+12 9.8%	2.48e+12 39.7%	1.14e+12 18.2%
loop at reactor.cpp: 446	5.14e+12 36.6%	5.35e+10 0.4%	4.36e+12 36.8%	4.62e+10 0.4%	1.38e+12 22.0%	3.29e+10 0.5%
reactor.cpp: 446	1.11e+12 7.9%	1.11e+12 7.9%	1.11e+12 9.4%	1.11e+12 9.4%	1.18e+12 17.7%	1.18e+12 17.7%
AMReX_GpuLaunchGlobal.H: 12	1.75e+10 0.1%	1.75e+10 0.1%	1.70e+10 0.1%	1.70e+10 0.1%		

Annotations on the table:

- CPU context:** points to the top rows of the table.
- GPU context:** points to the bottom rows of the table.
- 9.4% GPU stalls outside the loop:** points to the 'reactor.cpp: 446' row.
- mostly memory stalls:** points to the 'AMReX_GpuLaunchGlobal.H: 12' row.

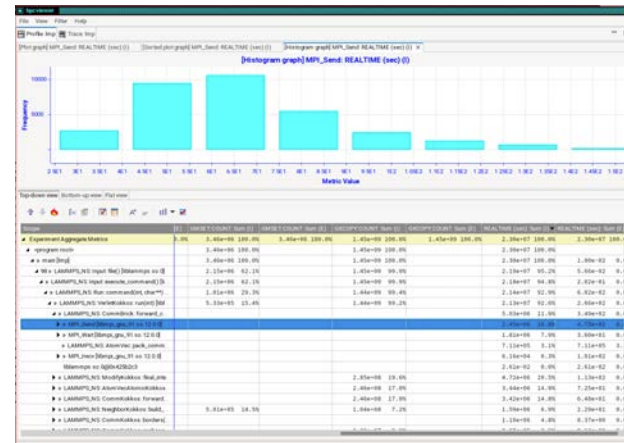
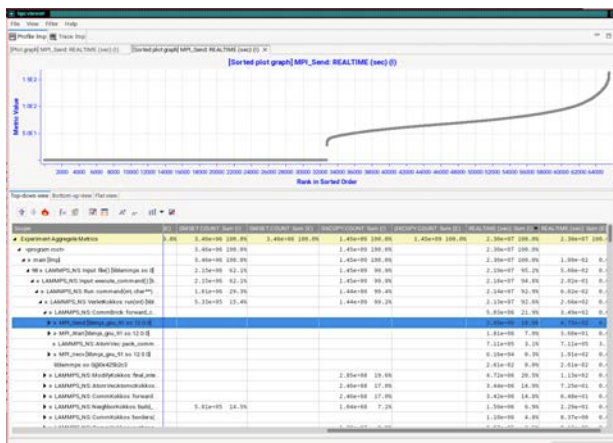
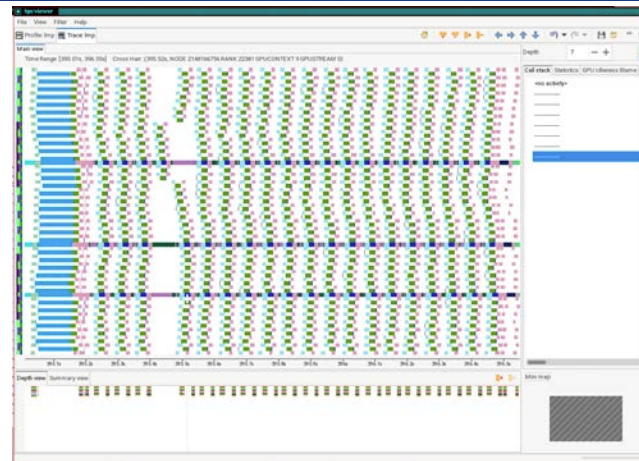
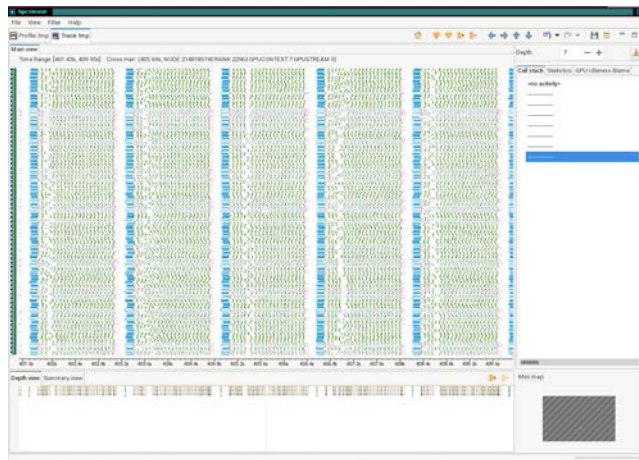
Key Metrics for GPU Kernels

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

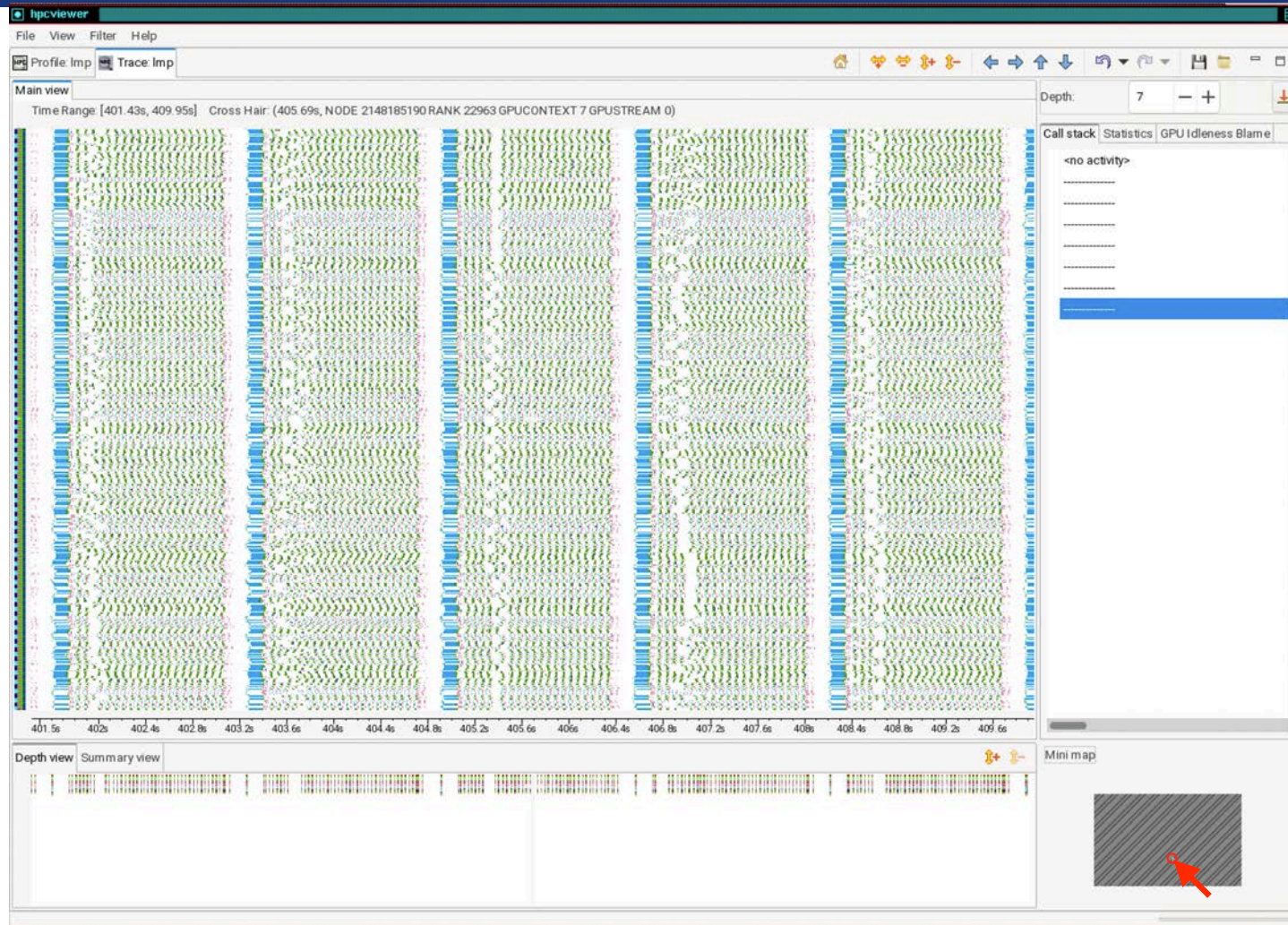
Metrics for GPU Kernels with PC Samples

- GINS: GPU instructions
 - GINS:STL_ANY: GPU instruction stalls for any reason
 - GINS:STL_IFET: GPU instruction stalls for instruction fetch
 - GINS:STL_GMEM: GPU instruction stalls for global memory
 - GINS:STL_CMEM: GPU instruction stalls for constant memory
 - GINS:STL_IDEP: GPU instruction stalls for instruction dependences
 - GINS:STL_PIPE: GPU instruction pipeline stalls
 - GINS:STL_MTHR: GPU instruction stalls for memory throttling
-
- GSAMP:EXP: expected number of samples
 - GSAMP:TOT: total number of samples recorded
 - GSAMP:UTIL: GPU utilization = (PC samples expected) / (PC samples total)

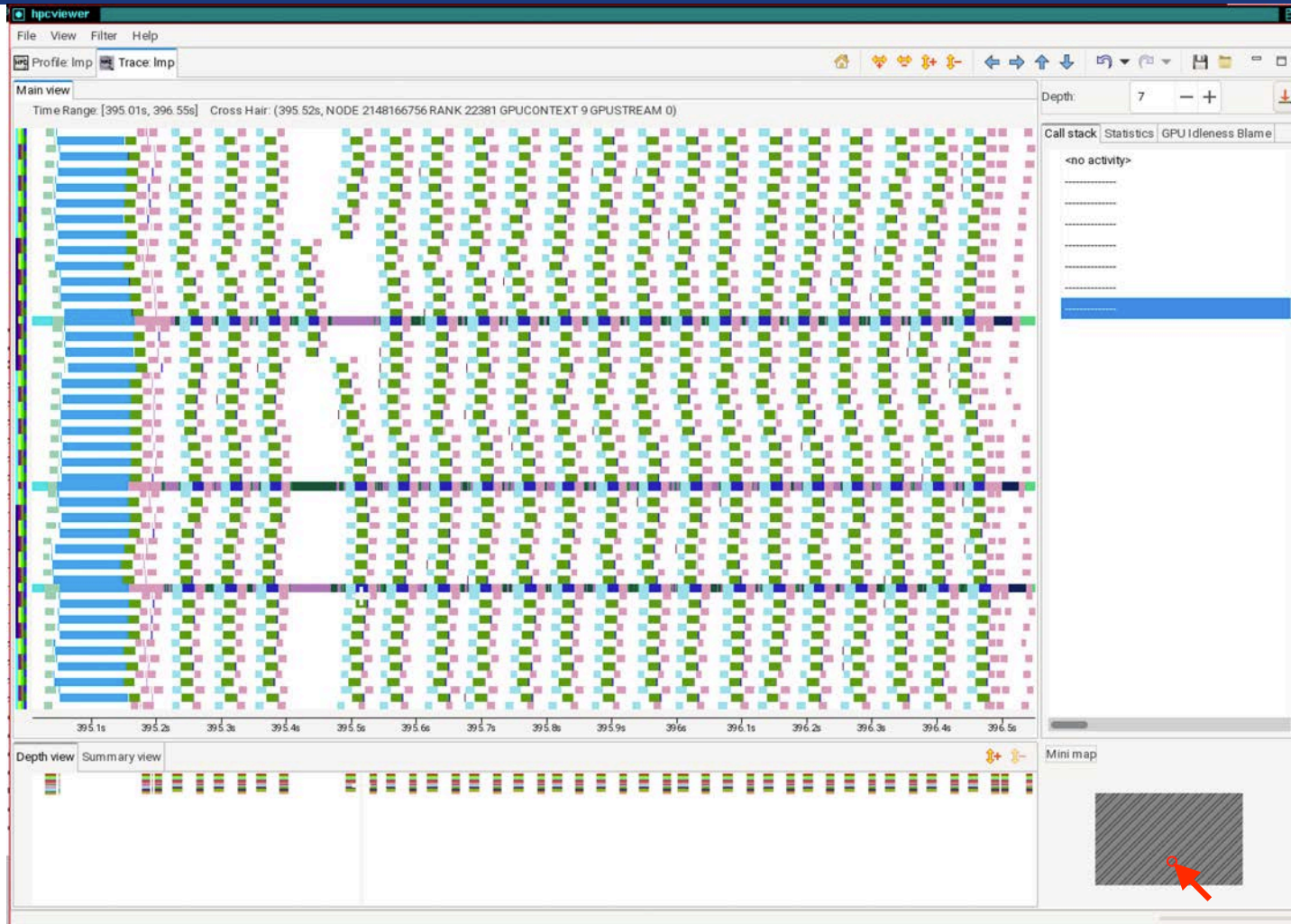
LAMMPS on Frontier: Executions with Kernel Duration of Milliseconds



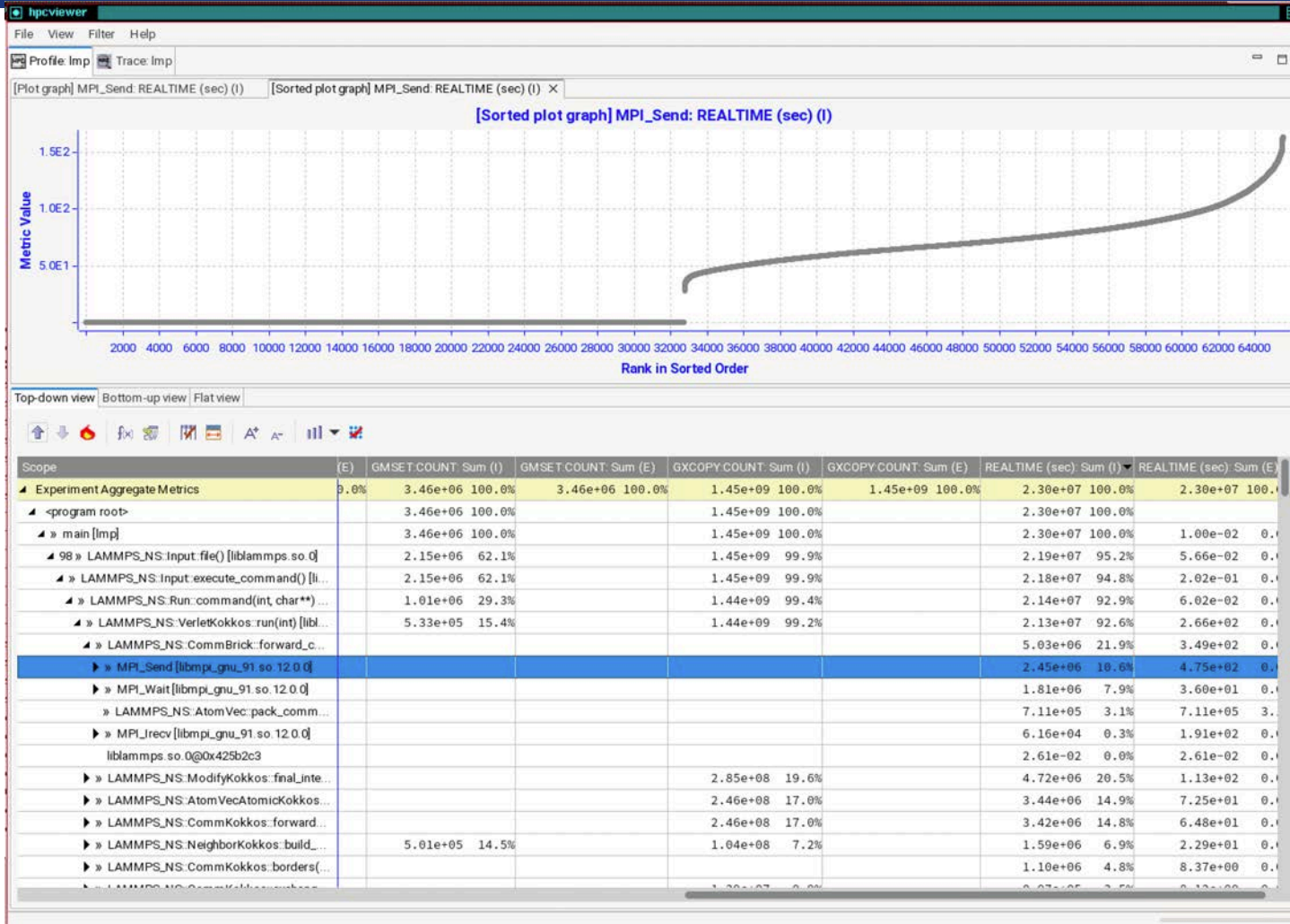
LAMMPS on Frontier: Executions with Kernel Duration of Milliseconds



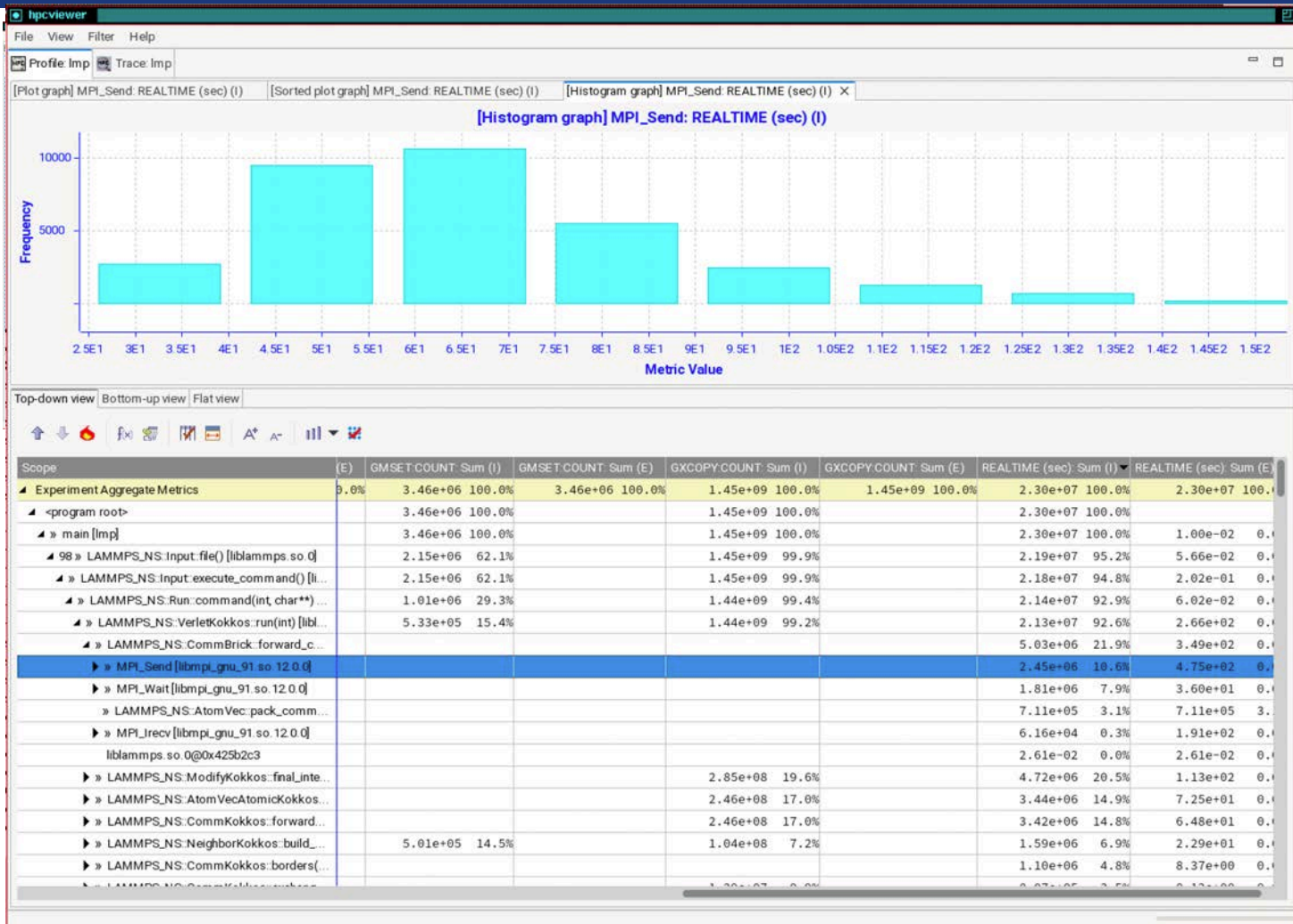
LAMMPS on Frontier: Executions with Kernel Duration of Milliseconds



LAMMPS on Frontier: Executions with Kernel Duration of Milliseconds

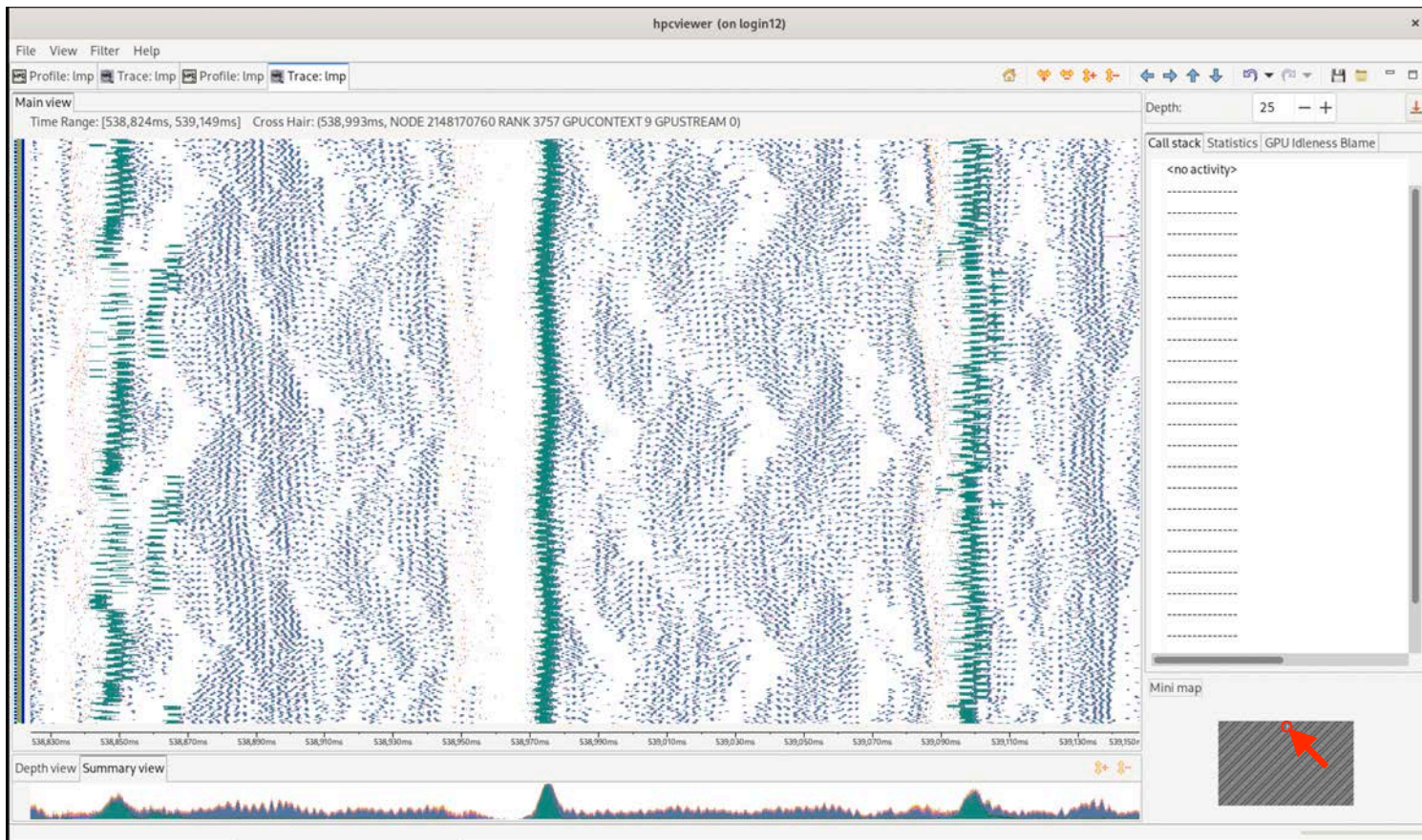


LAMMPS on Frontier: Executions with Kernel Duration of Milliseconds



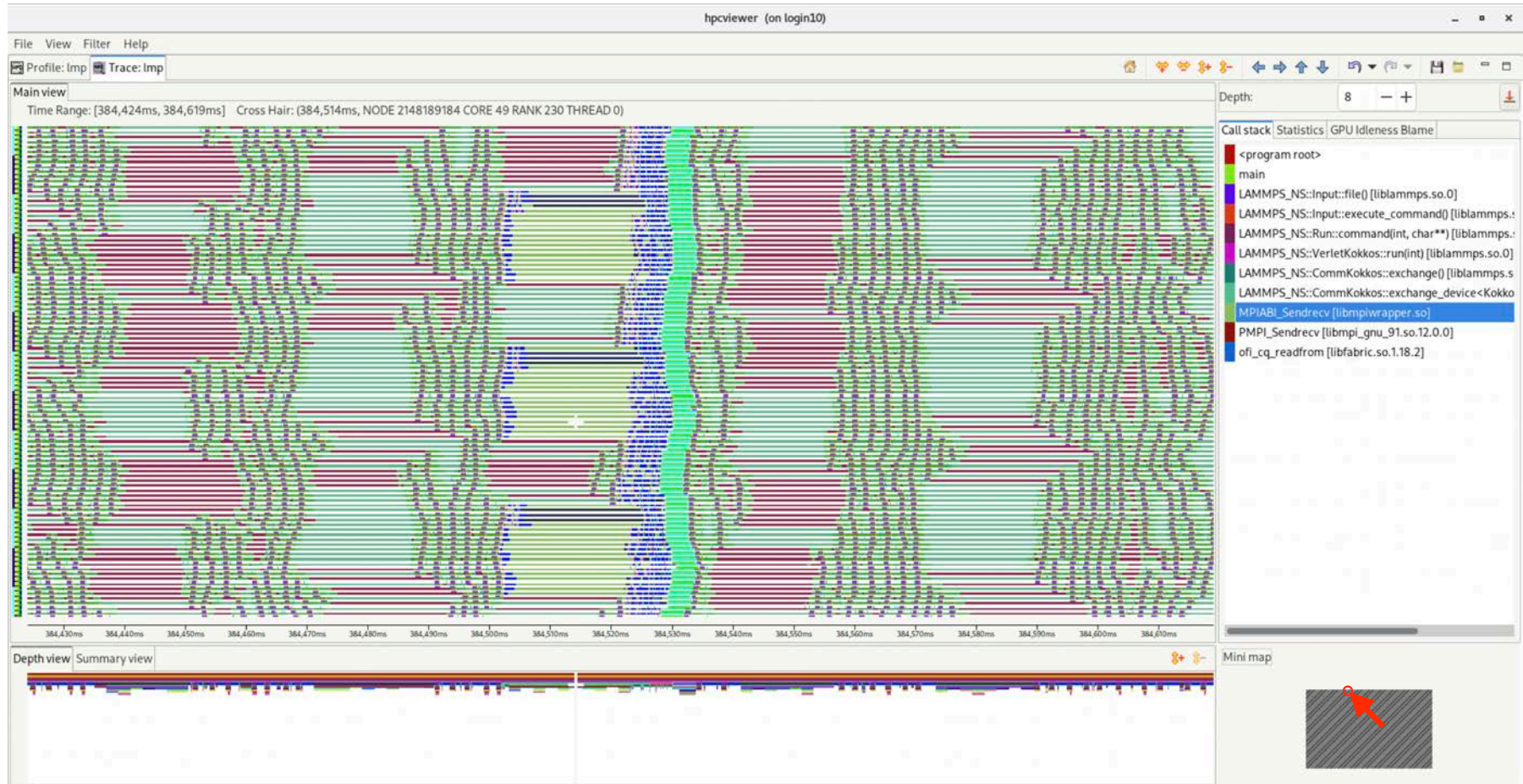
LAMMPS on Frontier: 8K nodes, 64K MPI ranks + 64K GPU tiles

Kernel duration of microseconds



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

Kernel duration of microseconds



Coming Attractions

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

Quick Start Guide for HPCToolkit on Polaris

- Load HPCToolkit on Polaris

```
module use /soft/modulefiles
```

```
module load hpctoolkit
```

- Measure an execution

—Profile and trace CPU and GPU activity of your application in a job script

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

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

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

- Analyze measurement data

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

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

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

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

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

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

- View your performance data

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


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

Hpcviewer Graphical User Interface on Your Laptop

Prepare to explore performance data on your laptop

- Download and install hpcviewer: <https://bit.ly/hpcviewer-install>

Select the right one for your laptop: MacOS (Apple Silicon, Intel), Windows, Linux

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

Viewing Performance Data

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

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

— hpcserver has been pre-installed on Polaris

— You can install hpcserver anywhere: <https://bit.ly/hpcserver-install>

Configuring Hpcviewer Remote Access

Run hpcviewer

From the file menu, select “Open remote database”

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

Fill in your username on Polaris

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

Select the authentication method: “Use password”

Click “OK”

Authenticate using your token as you normally do

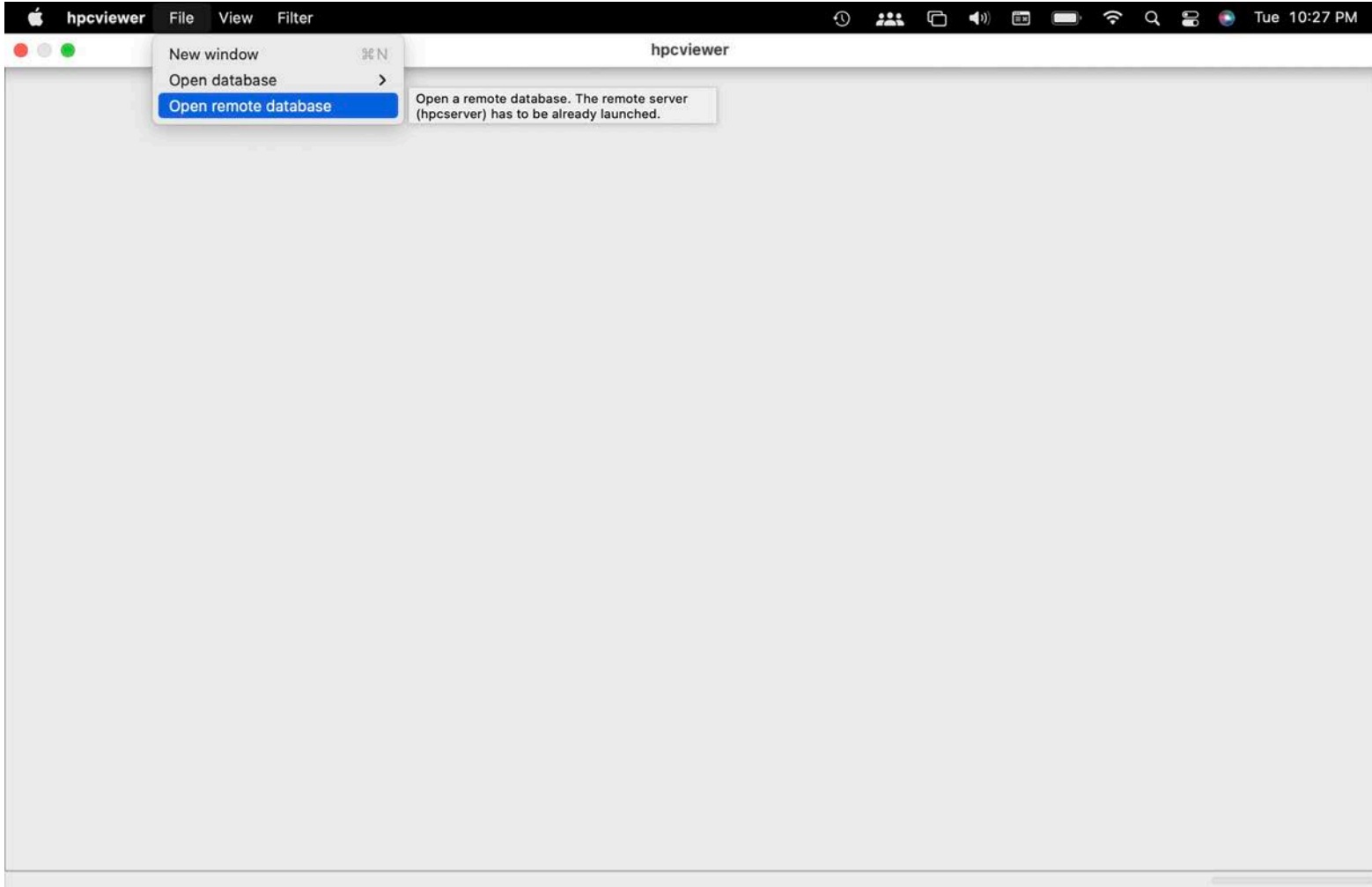
Navigate to a database with the file chooser in `/eagle/APTESC/hpctoolkit/databases`: quicksilver, lammps, arborx

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

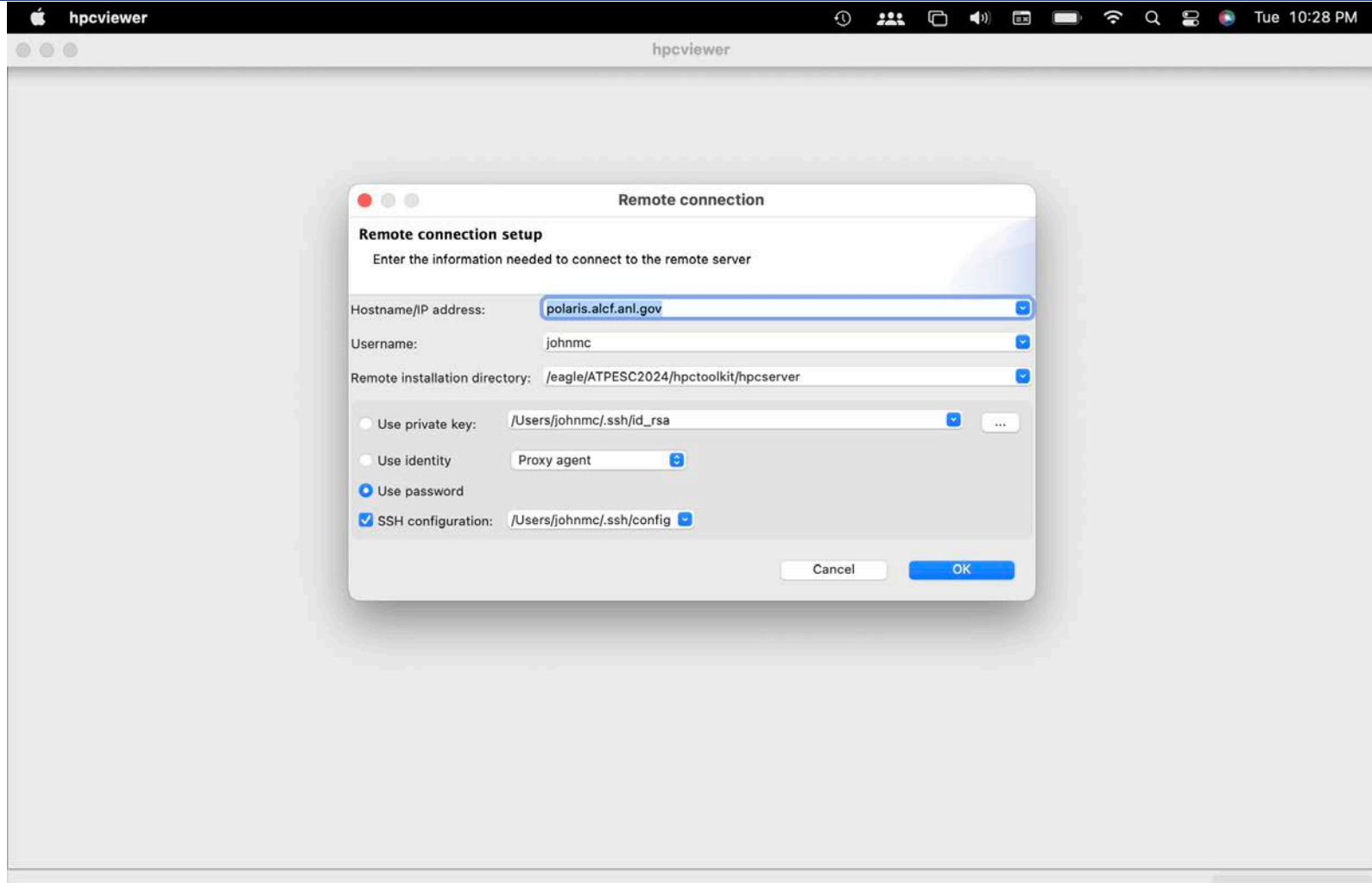
lammps: `hpctoolkit-lmp.d` `hpctoolkit-lmp-pc.d`

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

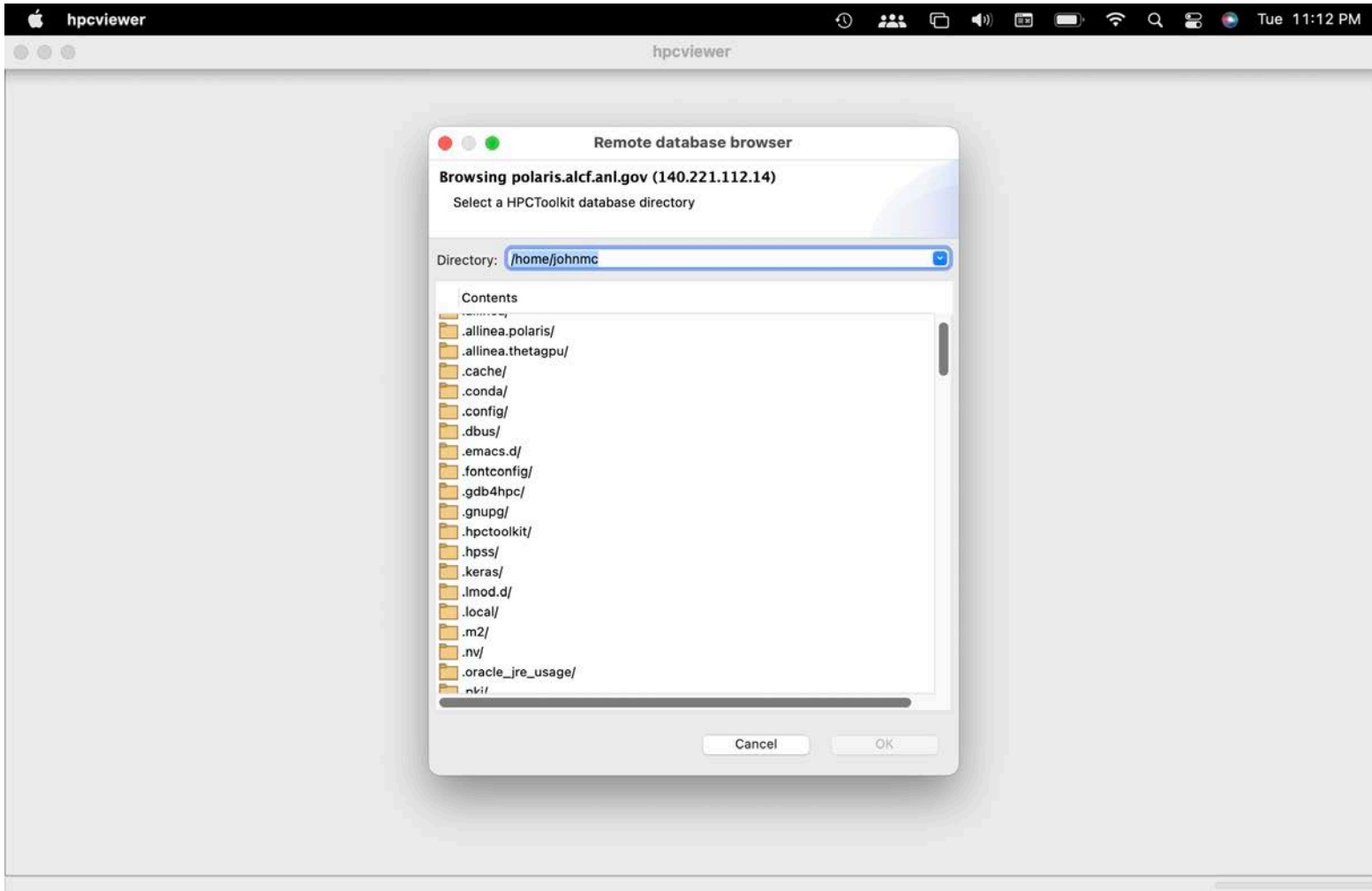
Opening a Remote Database



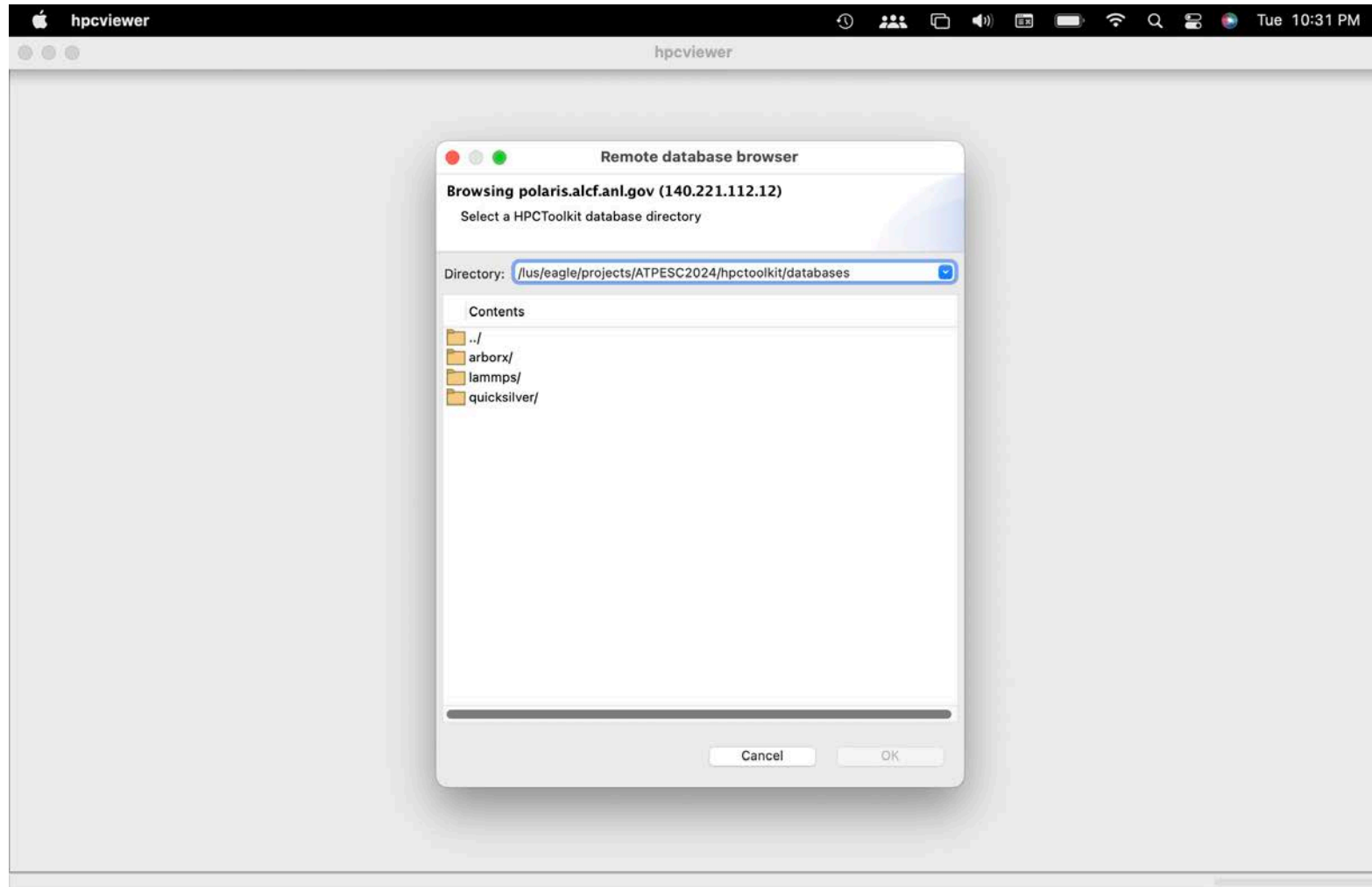
Configuring for use with Polaris



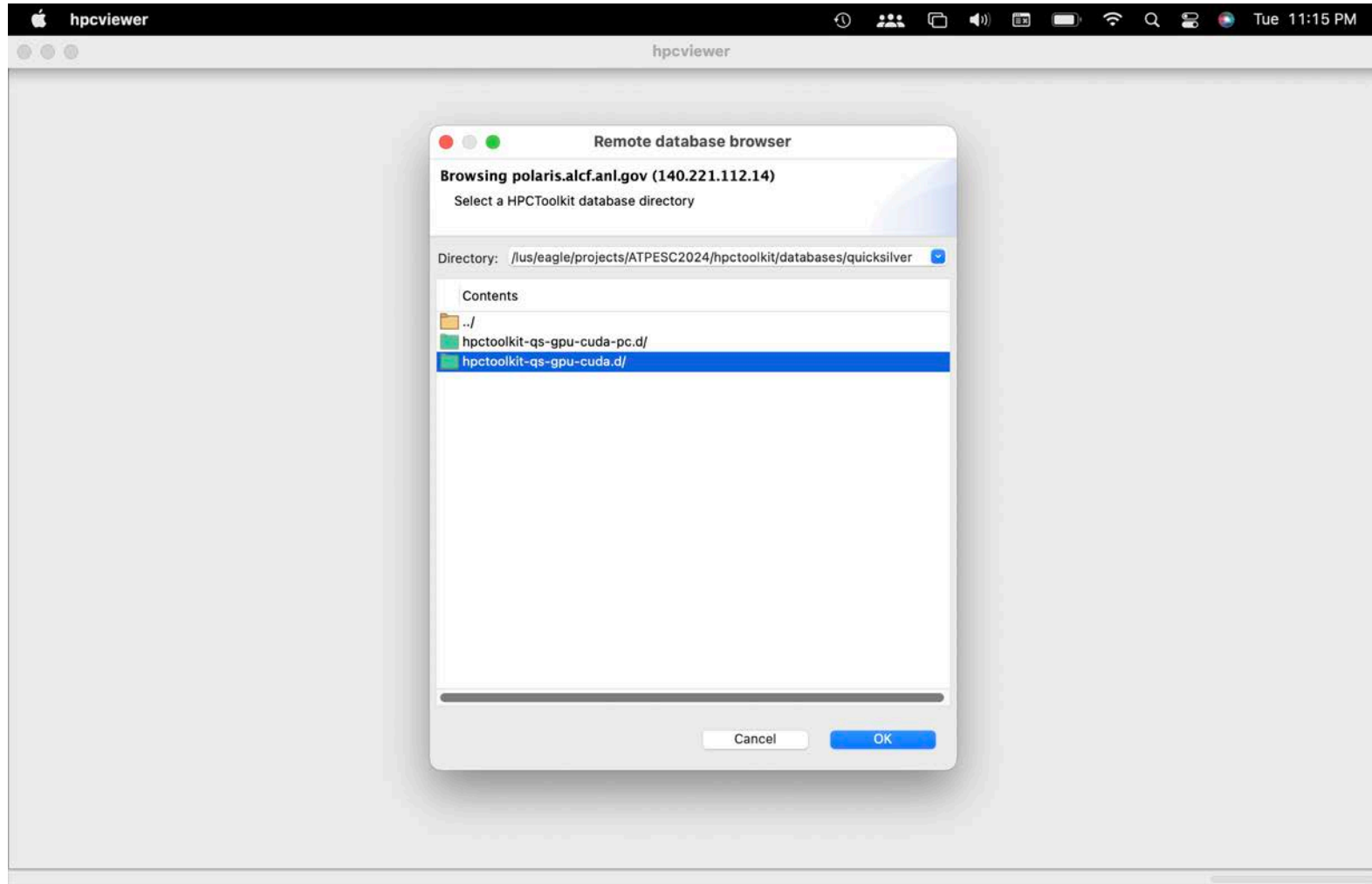
First View of Polaris: Your Home Directory



Navigate to Example Databases



Select a Quicksilver Database with Traces



After Selecting hpctoolkit-qs-gpu-cuda.d

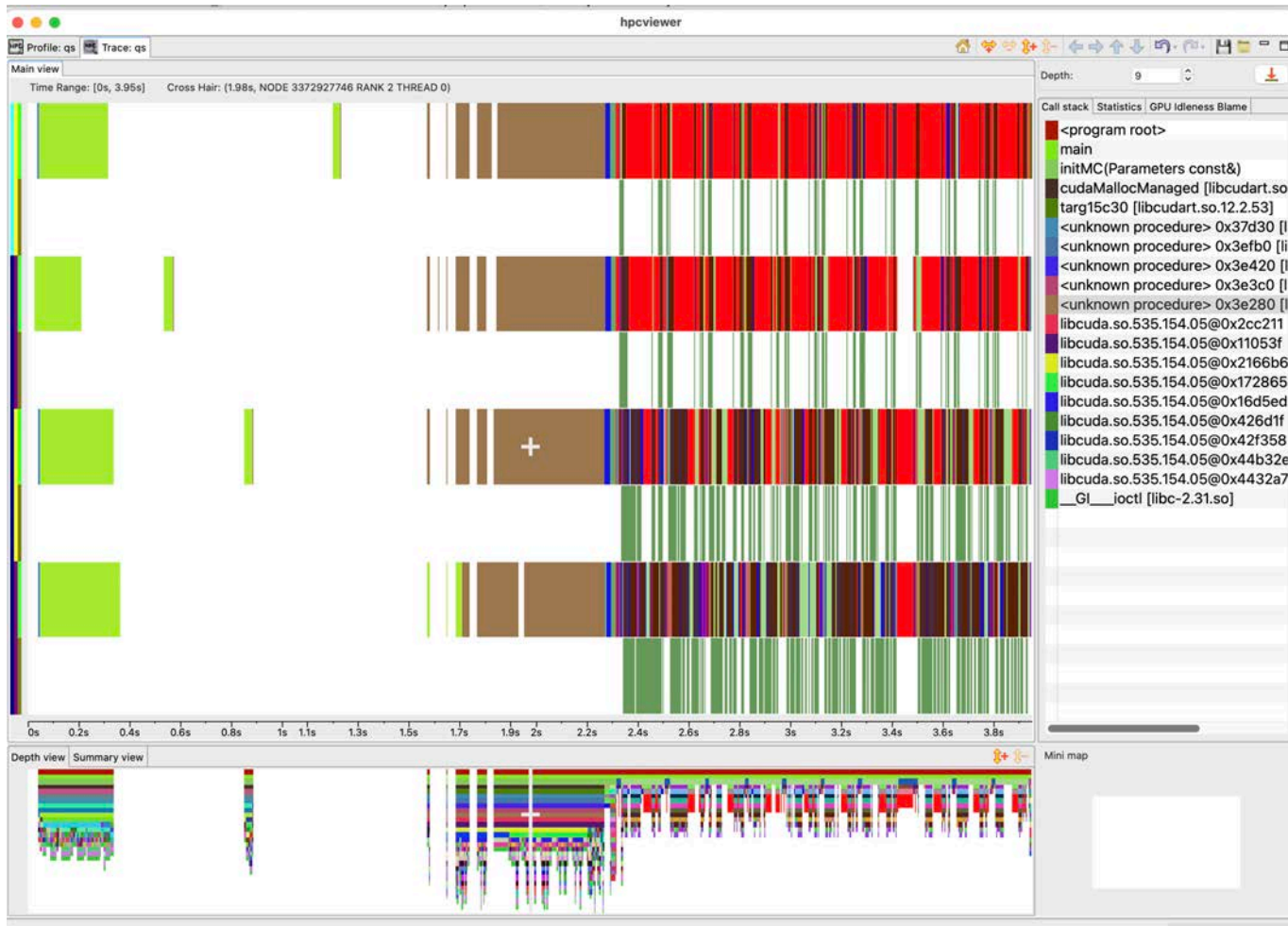
hpcviewer

Profile: qs Trace: qs

Top-down view Bottom-up view Flat view

Scope	REALTIME (sec): Sum (I)	REALTIME (sec): Sum (E)	GPUOP (sec): Sum (I)	GPUOP (sec): Sum (E)	GKER (sec): Sum (I)	GKER (sec): Sum (E)
Experiment Aggregate Metrics	1.56e+01 100.0%	1.56e+01 100.0%	2.24e+00 100.0%	2.24e+00 100.0%	2.24e+00 100.0%	2.24e+00 100.0%
<program root>	1.56e+01 100.0%		2.24e+00 100.0%		2.24e+00 100.0%	

Select the Tab “Trace: qs”



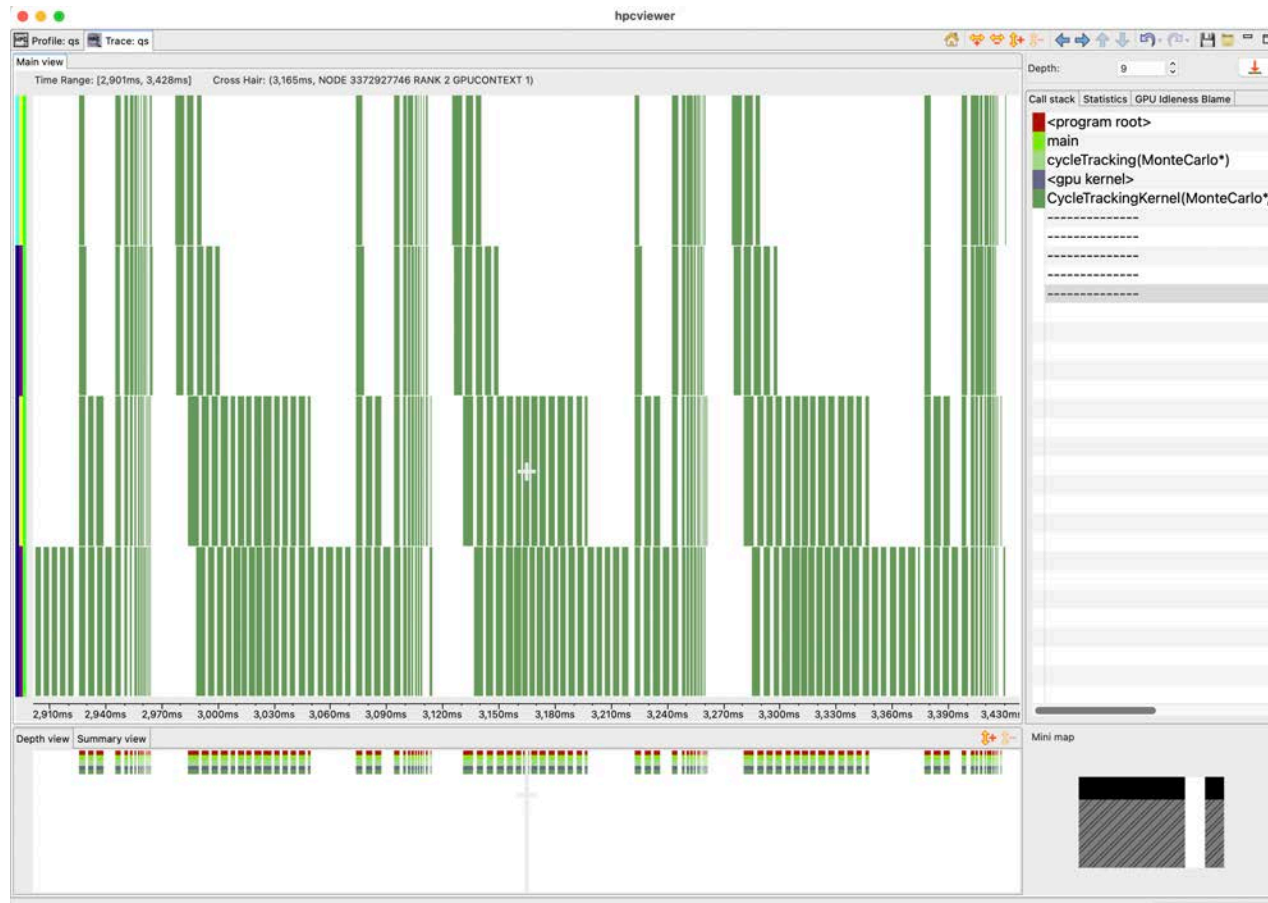
Use the Filter to “Uncheck all” and Check “GPU” streams

The screenshot shows the hpcviewer application interface. A dialog box titled "Filter execution context" is open in the center. The dialog has three radio buttons: "Check all" (selected), "Uncheck all", and "Regular expression". Below these is a text input field containing "GPU" and a "Minimum samples:" field. A table lists execution contexts with checkboxes in the "Visible" column.

Visible	Execution context	Samples
<input checked="" type="checkbox"/>	NODE 3372934401 RANK 0 GPUCONTEXT 1	0
<input checked="" type="checkbox"/>	NODE 3372927746 RANK 1 GPUCONTEXT 1	0
<input checked="" type="checkbox"/>	NODE 3372927746 RANK 2 GPUCONTEXT 1	0
<input checked="" type="checkbox"/>	NODE 3372927746 RANK 3 GPUCONTEXT 1	0

The background of the hpcviewer window shows a heatmap of execution over time, with a time range of [0s, 3.95s]. The x-axis is labeled from 0s to 3.8s in 0.2s increments. The y-axis represents different execution contexts. A cross-hair is positioned at (1.98s, NODE 3372927746 RANK 2 THREAD 0). On the right side, there is a "Call stack" panel showing a list of functions, including main, initMC, cudaMallocManaged, and various libcuda.so and libc.so paths. At the bottom, there is a "Depth view" and "Summary view" section showing a detailed view of the execution context over time.

See Load Imbalance Across the Four GPUs



The Profile View in the other “PC Sampling” Database

The screenshot displays the hpcviewer application. The top pane shows a C++ code snippet from CollisionEvent.cc, with lines 67-81 highlighted. The code is a loop over isoIndex, containing nested loops for uniqueNumber and reactIndex, and a call to macroscopicCrossSection.

The bottom pane shows a performance table with the following columns: Scope, GINS: Sum (I), GINS: Sum (E), GINS:STL_ANY: Sum (I), and GINS:STL_ANY: Sum (E). The table lists various scopes and their corresponding performance metrics.

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 » [] CycleTrackingFunction(MonteCarlo*, MC_Particle&, int, ParticleVault*)	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)	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, unsigned int)	2.71e+10 12.6%	1.35e+10 6.3%	2.40e+10 11.8%	1.20e+10 5.9%
253 » [] 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%

Collecting Performance Data with HPCToolkit: Turnkey Examples

Hands-on Tutorial Examples

```
% git clone https://github.com/hpctoolkit/hpctoolkit-tutorial-  
examples
```

```
% cd hpctoolkit-tutorial-examples/gpu/nvidia
```

```
% ls
```

```
    arborx.kokkos    lammps.kokkos    quicksilver.cuda
```


A Hands-on Example: Quicksilver

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


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

Notes

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




Analyzing Quicksilver Traces

Using a measurement database with profiles and traces

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

Analyzing Quicksilver Traces

Using a measurement database with profiles and traces

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




Analyzing Quicksilver Traces

Using a measurement database with profiles and traces

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

Analyzing the Quicksilver Summary Profile

Using a measurement database with profiles and traces

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

Analyzing the Quicksilver Summary Profile

- You will notice that for quicksilver, HPCToolkit doesn't report any data copies between the host and device
 - The quicksilver code uses “unified memory” so that all of the data movement occurs between CPU and GPU using page faults rather than explicit copies
 - Today's GPU hardware doesn't support attribution of page faults to individual instructions
 - We could profile them, but not attribute them to code

Analyzing Quicksilver PC Samples

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

Using the default top-down view of the profile

- Select the column “GINS (I)” to focus on the measurement of inclusive GPU Instructions
- Select use the flame button to look at where the instructions are executed
- In the call stack revealed, you will see a <gpu kernel> placeholder that separates CPU activity (above) from GPU kernel activity (below)
- Below the <gpu kernel> placeholder you will see the function calls, inlined functions, loops and statements in HPCToolkit’s reconstruction of calling contexts within the CycleTrackingKernel
- Using the bottom-up view of the profile
 - Select the bottom-up tab of above the control pane
 - Select the GINS STL_ANY (E) column, which will sort the functions by the exclusive GPU instruction stalls within that function
 - Scroll right to see which of the types of contributing types of stalls accounts for most of the STL_ANY amount
 - Select the function that has the most exclusive stalls
 - Select the the hot path to see where this function is called from.
 - Where do the calls to the costly function come from?
 - Does there appear to be an opportunity to reduce the number of calls to this function?

Filtering Tips to Hide Unwanted Implementation Details

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

A Hands-on Example: ArborX

Performance portable algorithms for geometric search MPI + Kokkos + OpenMP

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

Notes

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

Analyzing ArborX Traces

Using a measurement database with profiles and traces

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

A Hands-on Example: LAMMPS

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

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

Notes

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

Analyzing LAMMPS Profiles, Traces, and PC Samples

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

Troubleshooting: Only GPU kernel Name

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

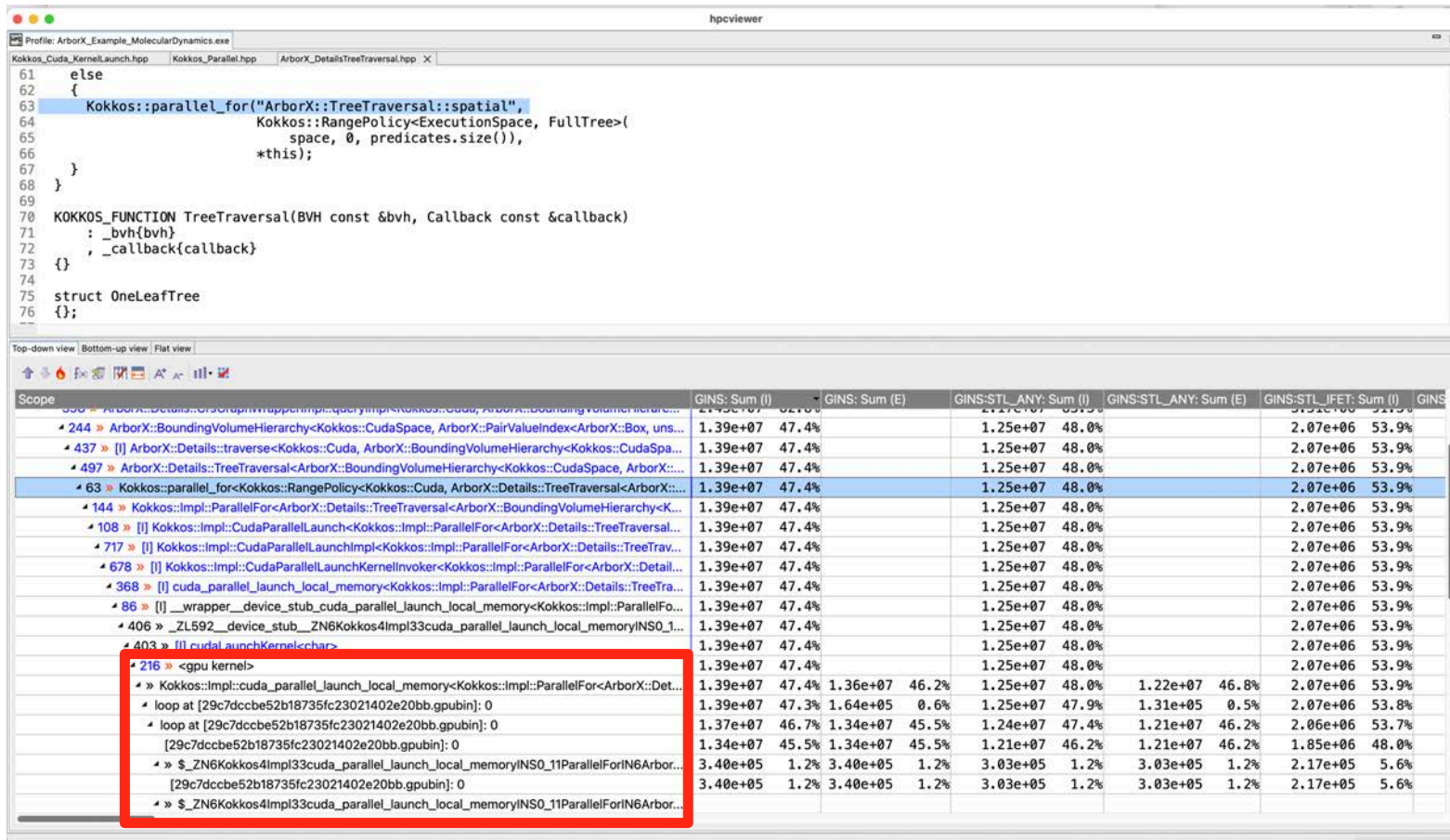
The screenshot shows the hpcviewer application. The top pane displays C++ code from `Kokkos_Cuda_KernelLaunch.hpp`. The bottom pane shows a performance table with columns for Scope, GKER (sec): Sum (I), GKER (sec): Sum (E), GXCOPY (sec): Sum (I), GXCOPY (sec): Sum (E), and GXCOPY:H2D (B). A red box highlights the entry for the GPU kernel at line 216.

```
81 driver();
82 }
83
84 template <class DriverType>
85 __global__ static void cuda_parallel_launch_local_memory(
86     const DriverType driver) {
87     driver();
88 }
89
90 template <class DriverType, unsigned int maxTperB, unsigned int minBperSM>
91 __global__ __launch_bounds__(
92     maxTperB,
93     minBperSM) static void cuda_parallel_launch_local_memory(const DriverType
94     driver) {
95     driver();
96 }
97
```

Scope	GKER (sec): Sum (I)	GKER (sec): Sum (E)	GXCOPY (sec): Sum (I)	GXCOPY (sec): Sum (E)	GXCOPY:H2D (B)
437 » [I] ArborX::Details::traverse<Kokkos::Cuda, ArborX::BoundingVolumeHierarchy<Kokkos::CudaSpace, ArborX::PairValue...	3.63e-04	39.9%			
497 » ArborX::Details::TreeTraversal<ArborX::BoundingVolumeHierarchy<Kokkos::CudaSpace, ArborX::PairValueIndex<Arbo...	3.63e-04	39.9%			
63 » Kokkos::parallel_for<Kokkos::RangePolicy<Kokkos::Cuda, ArborX::Details::TreeTraversal<ArborX::BoundingVolumeHie...	3.63e-04	39.9%			
144 » Kokkos::Impl::ParallelFor<ArborX::Details::TreeTraversal<ArborX::BoundingVolumeHierarchy<Kokkos::CudaSpace, A...	3.63e-04	39.9%			
108 » [I] Kokkos::Impl::CudaParallelLaunch<Kokkos::Impl::ParallelFor<ArborX::Details::TreeTraversal<ArborX::BoundingV...	3.63e-04	39.9%			
717 » [I] Kokkos::Impl::CudaParallelLaunchImpl<Kokkos::Impl::ParallelFor<ArborX::Details::TreeTraversal<ArborX::Bound...	3.63e-04	39.9%			
678 » [I] Kokkos::Impl::CudaParallelLaunchKernelInvoker<Kokkos::Impl::ParallelFor<ArborX::Details::TreeTraversal<Arb...	3.63e-04	39.9%			
368 » [I] cuda_parallel_launch_local_memory<Kokkos::Impl::ParallelFor<ArborX::Details::TreeTraversal<ArborX::Boun...	3.63e-04	39.9%			
86 » [I] __wrapper__device_stub_cuda_parallel_launch_local_memory<Kokkos::Impl::ParallelFor<ArborX::Details::Tr...	3.63e-04	39.9%			
406 » __ZL592_device_stub__ZN6Kokkos4Impl33cuda_parallel_launch_local_memoryINS0_11ParallelForIN6Arbor...	3.63e-04	39.9%			
402 » [I] cuda_parallel_launch_kernel...	3.63e-04	39.9%			
216 » <gpu kernel>	3.63e-04	39.9%			
» Kokkos::Impl::cuda_parallel_launch_local_memory<Kokkos::Impl::ParallelFor<ArborX::Details::TreeTraversal<Ar...	3.63e-04	39.9%	3.63e-04	39.9%	
Kokkos_Cuda_KernelLaunch.hpp: 85					
216 » <gpu kernel>					
182 » ArborX::BoundingVolumeHierarchy<Kokkos::CudaSpace, ArborX::PairValueIndex<ArborX::Box, unsigned int>, ArborX::D...	2.53e-04	27.8%			
209 » ArborX::Details::KokkosExt::exclusive_scan<Kokkos::Cuda, Kokkos::View<int*, Kokkos::CudaSpace>, Kokkos::View<int*,...	9.15e-06	1.0%			
237 » Kokkos::parallel_for<Kokkos::RangePolicy<Kokkos::Cuda>, __nv_hdl_wrapper_t<false, false, false, __nv_dl_tag<void (*)...	2.30e-06	0.3%			
205 » Kokkos::parallel_for<Kokkos::RangePolicy<Kokkos::Cuda>, __nv_hdl_wrapper_t<false, false, false, __nv_dl_tag<void (*)...	2.18e-06	0.2%			
211 » ArborX::Details::KokkosExt::lastElement<Kokkos::Cuda, int*, Kokkos::CudaSpace><Kokkos::Cuda const&, Kokkos::View<...			1.92e-06	1.3%	
242 » ArborX::Details::KokkosExt::lastElement<Kokkos::Cuda, Kokkos::View<int*, Kokkos::CudaSpace>, Kokkos::View<int*,...			1.92e-06	1.3%	

Troubleshooting: No GPU source code lines with PC sampling

- If you don't see source code with GPU sampling on NVIDIA GPUs: compile with “-lineinfo” option



The screenshot shows the hpcviewer application. The top pane displays C++ source code for a tree traversal function. The bottom pane shows a performance table with columns for GINS and GINS-STL_ANY metrics. A red box highlights the 'gpu kernel' section of the table, which includes several entries for Kokkos::Impl::CudaParallelLaunchKernelInvoker and Kokkos::Impl::ParallelForArborX::Details::TreeTraversal.

```
61     else
62     {
63         Kokkos::parallel_for("ArborX::TreeTraversal::spatial",
64                             Kokkos::RangePolicy<ExecutionSpace, FullTree>(
65                                 space, 0, predicates.size()),
66                             *this);
67     }
68 }
69
70 KOKKOS_FUNCTION TreeTraversal(BVH const &bvh, Callback const &callback)
71     : _bvh{bvh}
72     , _callback{callback}
73 {}
74
75 struct OneLeafTree
76 {};
```

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)
244 » ArborX::BoundingVolumeHierarchy<Kokkos::CudaSpace, ArborX::PairValueIndex<ArborX::Box, uns...	1.39e+07	47.4%	1.25e+07	48.0%	2.07e+06	53.9%
437 » [I] ArborX::Details::traverse<Kokkos::Cuda, ArborX::BoundingVolumeHierarchy<Kokkos::CudaSpa...	1.39e+07	47.4%	1.25e+07	48.0%	2.07e+06	53.9%
497 » ArborX::Details::TreeTraversal<ArborX::BoundingVolumeHierarchy<Kokkos::CudaSpace, ArborX::...	1.39e+07	47.4%	1.25e+07	48.0%	2.07e+06	53.9%
63 » Kokkos::parallel_for<Kokkos::RangePolicy<Kokkos::Cuda, ArborX::Details::TreeTraversal<ArborX::...	1.39e+07	47.4%	1.25e+07	48.0%	2.07e+06	53.9%
144 » Kokkos::Impl::ParallelFor<ArborX::Details::TreeTraversal<ArborX::BoundingVolumeHierarchy<K...	1.39e+07	47.4%	1.25e+07	48.0%	2.07e+06	53.9%
108 » [I] Kokkos::Impl::CudaParallelLaunchKernelInvoker<Kokkos::Impl::ParallelFor<ArborX::Details::TreeTraversa...	1.39e+07	47.4%	1.25e+07	48.0%	2.07e+06	53.9%
717 » [I] Kokkos::Impl::CudaParallelLaunchKernelInvoker<Kokkos::Impl::ParallelFor<ArborX::Details::TreeTrav...	1.39e+07	47.4%	1.25e+07	48.0%	2.07e+06	53.9%
678 » [I] Kokkos::Impl::CudaParallelLaunchKernelInvoker<Kokkos::Impl::ParallelFor<ArborX::Detail...	1.39e+07	47.4%	1.25e+07	48.0%	2.07e+06	53.9%
368 » [I] cuda_parallel_launch_local_memory<Kokkos::Impl::ParallelFor<ArborX::Details::TreeTra...	1.39e+07	47.4%	1.25e+07	48.0%	2.07e+06	53.9%
86 » [I] __wrapper__device_stub_cuda_parallel_launch_local_memory<Kokkos::Impl::ParallelFo...	1.39e+07	47.4%	1.25e+07	48.0%	2.07e+06	53.9%
406 » _ZL592__device_stub__ZN6Kokkos4Impl33cuda_parallel_launch_local_memoryINS0_1...	1.39e+07	47.4%	1.25e+07	48.0%	2.07e+06	53.9%
403 » [I] cuda_launchKernelschar>	1.39e+07	47.4%	1.25e+07	48.0%	2.07e+06	53.9%
216 » <gpu kernel>	1.39e+07	47.4%	1.25e+07	48.0%	2.07e+06	53.9%
» Kokkos::Impl::cuda_parallel_launch_local_memory<Kokkos::Impl::ParallelFor<ArborX::Det...	1.39e+07	47.4%	1.36e+07	46.2%	1.22e+07	46.8%
» loop at [29c7dccbe52b18735fc23021402e20bb.gpubin]: 0	1.39e+07	47.3%	1.64e+05	0.6%	1.25e+07	47.9%
» loop at [29c7dccbe52b18735fc23021402e20bb.gpubin]: 0	1.37e+07	46.7%	1.34e+07	45.5%	1.24e+07	47.4%
» [29c7dccbe52b18735fc23021402e20bb.gpubin]: 0	1.34e+07	45.5%	1.34e+07	45.5%	1.21e+07	46.2%
» \$ _ZN6Kokkos4Impl33cuda_parallel_launch_local_memoryINS0_11ParallelForIN6Arbor...	3.40e+05	1.2%	3.40e+05	1.2%	3.03e+05	1.2%
» [29c7dccbe52b18735fc23021402e20bb.gpubin]: 0	3.40e+05	1.2%	3.40e+05	1.2%	3.03e+05	1.2%
» \$ _ZN6Kokkos4Impl33cuda_parallel_launch_local_memoryINS0_11ParallelForIN6Arbor...	3.40e+05	1.2%	3.40e+05	1.2%	3.03e+05	1.2%

Troubleshooting: Compiling ArborX with GPU Line Map Info

- ArborX cmake isn't set up to include GPU line mappings
- Force the compiler to record GPU line mappings
- `cmake -DARBORX_ENABLE_EXAMPLES=true \
-DCMAKE_INSTALL_PREFIX=`pwd`/../install \
-DCMAKE_CXX_COMPILER=g++ \
-DCMAKE_BUILD_TYPE=RelWithDebInfo \
-DCMAKE_CXX_FLAGS_RELWITHDEBINFO="-O2 -g -DNDEBUG -lineinfo"`

HPCToolkit Resources

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

Current Funding for HPCToolkit

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