

# Performance Analysis of GPU-accelerated Applications with HPCToolkit

John Mellor-Crummey  
Rice University

# Linux Foundation's HPCToolkit Performance Tools

Collect **profiles and traces of unmodified parallel CPU and GPU-accelerated applications**

Understand where an application spends its time and why

- call path profiles **associate metrics with application source code** contexts

- analyze instruction-level performance within GPU kernels** and attribute it to your source code

- hierarchical **traces to understand execution dynamics**

Parallel programming models

- across nodes: MPI, SHMEM, UPC++, ...

- within nodes: OpenMP, Kokkos, RAJA, HIP, DPC++, Sycl, CUDA, OpenACC, ...

Languages

- C, C++, Fortran, Python, ...

Hardware

- CPU cores and GPUs within a node

- CPU: x86\_64, Power, ARM

- GPU: NVIDIA, AMD, Intel

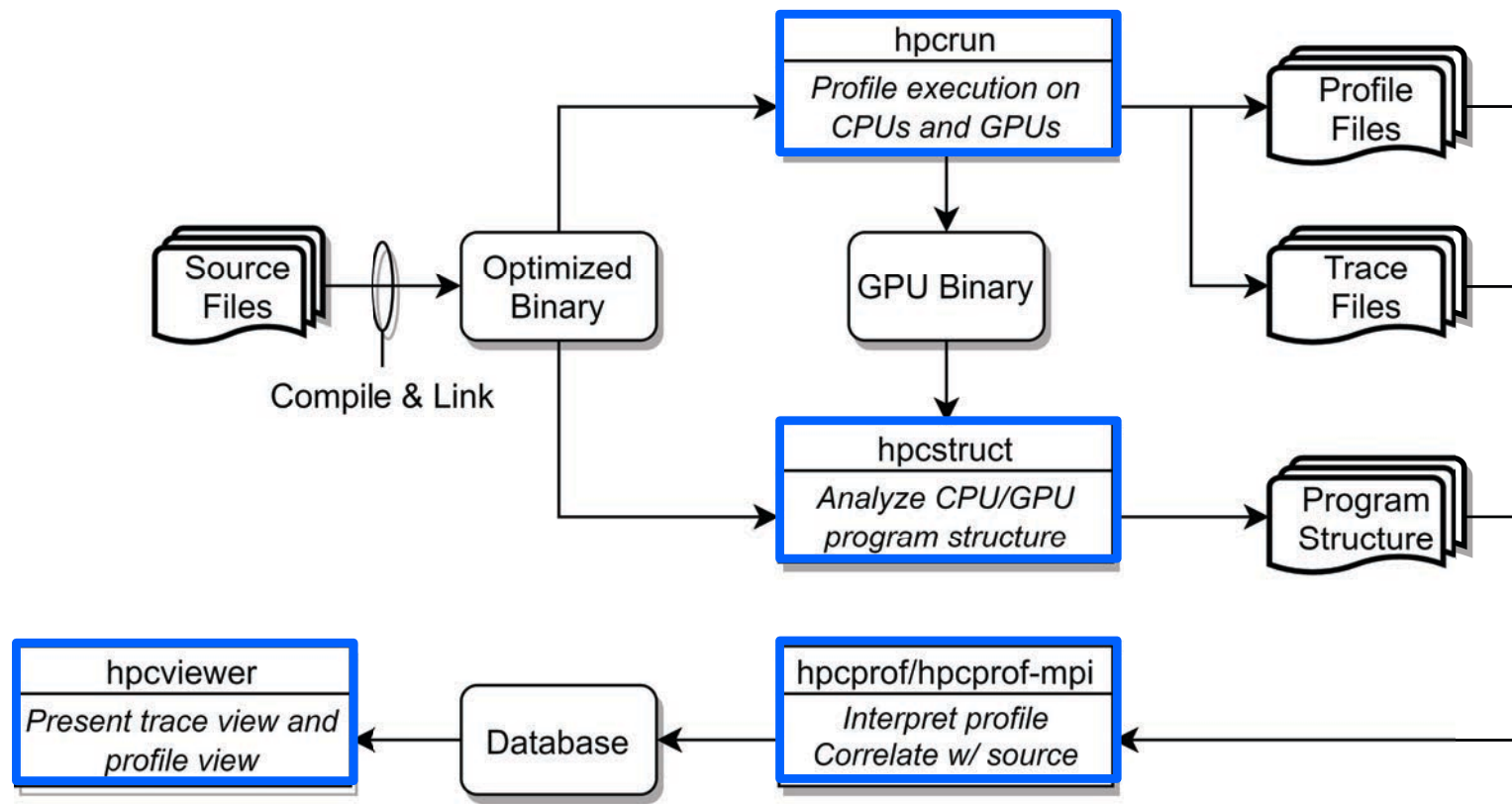
# Why HPCToolkit?

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

# How does HPCToolkit Differ from Vendor Tools?

- NVIDIA NSight Systems
  - tracing of CPU and GPU streams
  - analyze traces when you open them with the GUI
    - long running traces are huge and thus extremely slow to analyze, limiting scalability
  - designed for measurement and analysis within a node
- NVIDIA NSight Compute
  - detailed measurement of kernels with counters and execution replay
  - very slow measurement
  - flat display of measurements within GPU kernels
- Intel VTune: designed for analysis of performance on a single node
- AMD Omnitrace: designed for analysis of performance on a single node
- HPCToolkit
  - more scalable tracing than vendor tools
    - measure exascale executions across many nodes and GPUs
    - GUI can render trace data measured in TB
  - scalable, parallel post-mortem analysis vs. non-scalable in-GUI analysis
  - detailed reconstruction of estimates for calling context profiles within GPU kernels

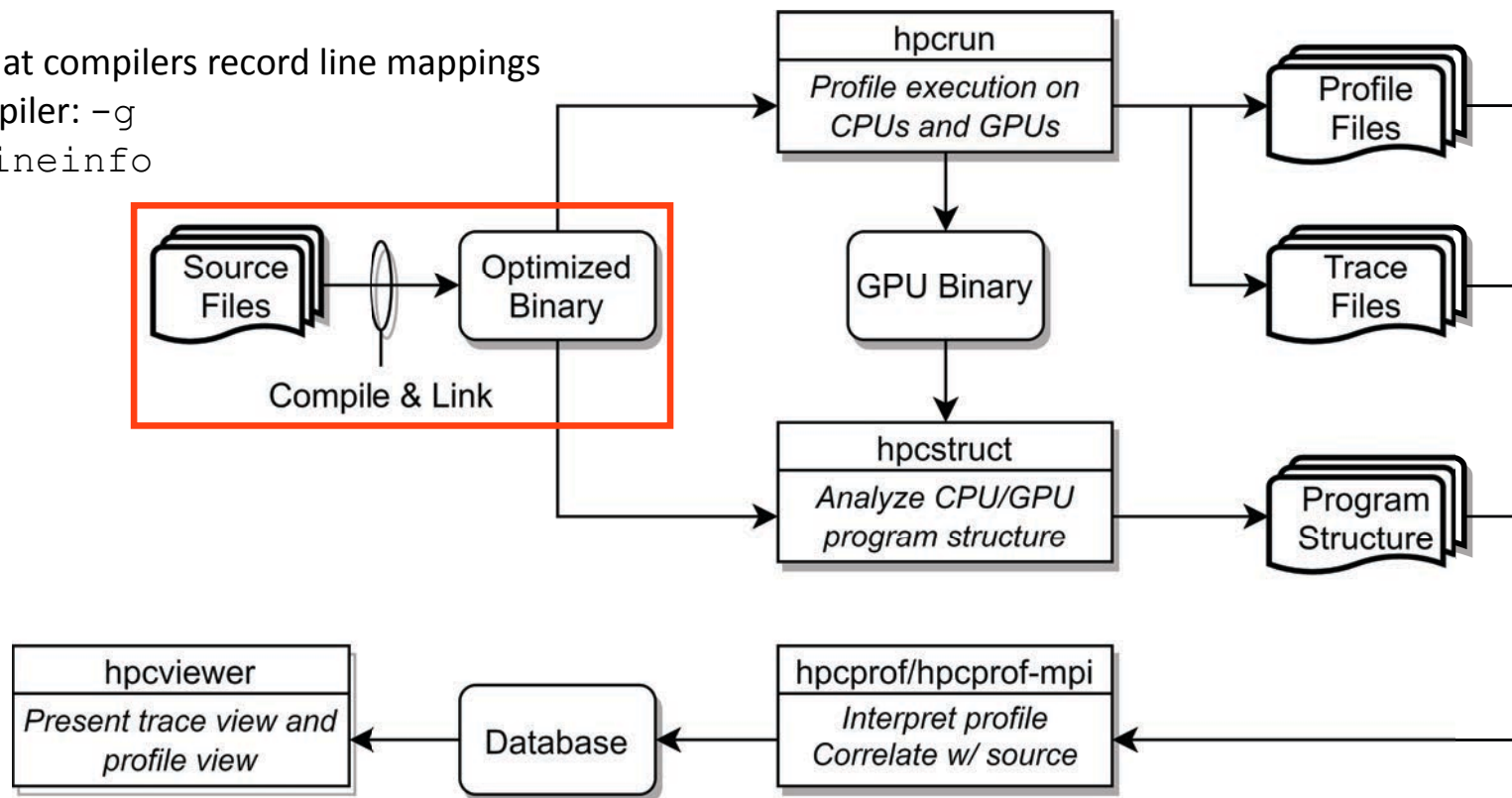
# HPCToolkit's Workflow for GPU-accelerated Applications



# HPCToolkit's Workflow for GPU-accelerated Applications

## Step 1:

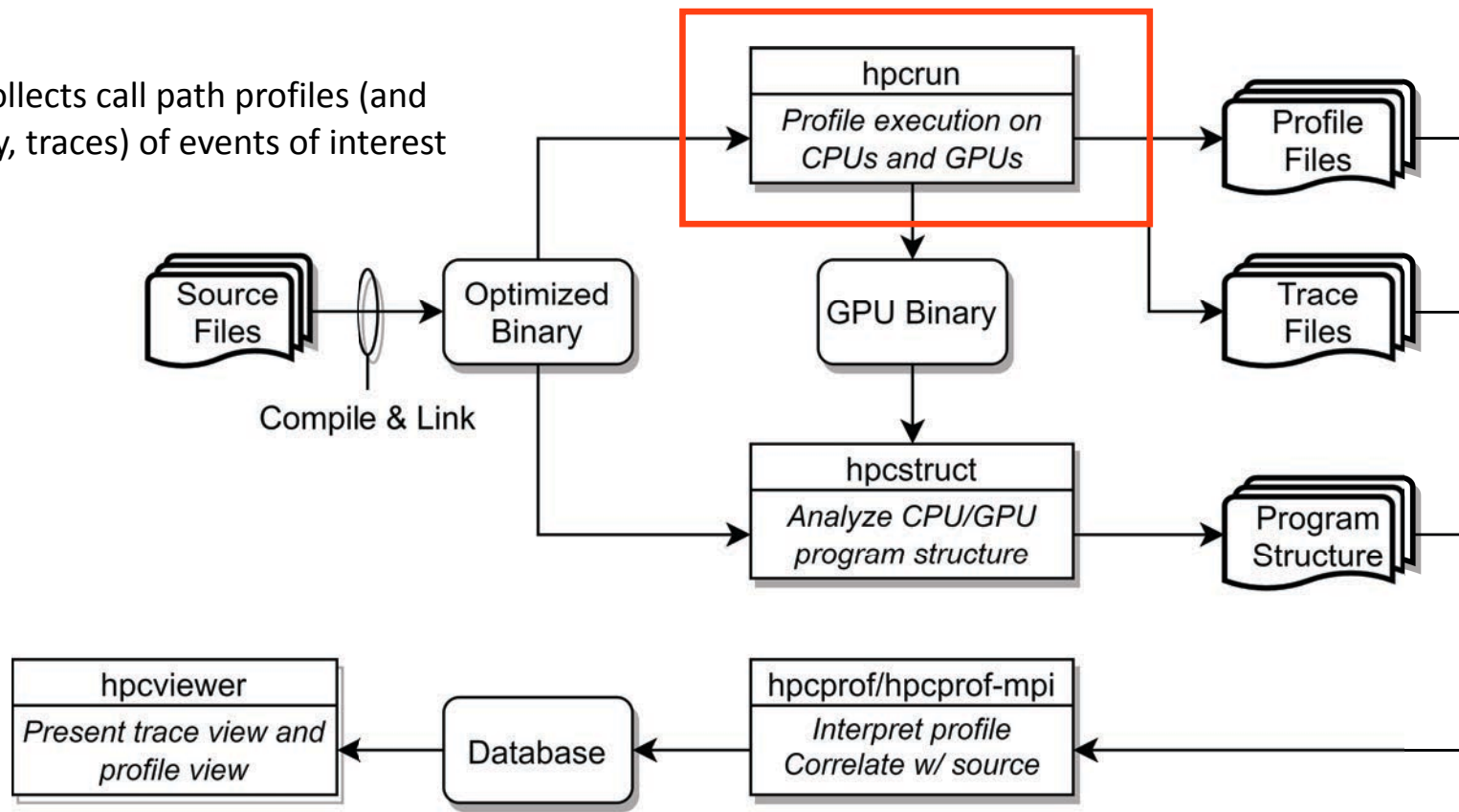
- Ensure that compilers record line mappings
- host compiler: `-g`
- `nvcc: -lineinfo`



# HPCToolkit's Workflow for GPU-accelerated Applications

Step 2:

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



# Measurement of CPU and GPU-accelerated Applications

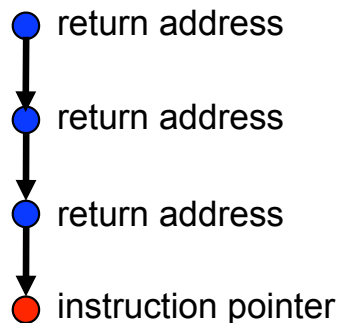
- Sampling using Linux timers and hardware counter overflows on the CPU
- Callbacks when GPU operations are launched and (sometimes) completed
- Event stream for GPU operations
- PC Samples: NVIDIA (in progress: AMD, Intel)
- Binary instrumentation of GPU kernels on Intel GPUs for fine-grain measurement



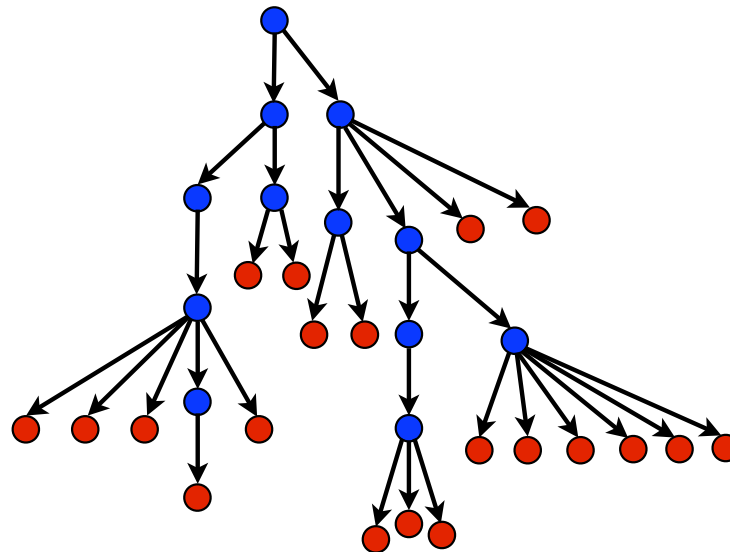
# Call Stack Unwinding to Attribute Costs in Context

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

## Call path sample



## Calling context tree



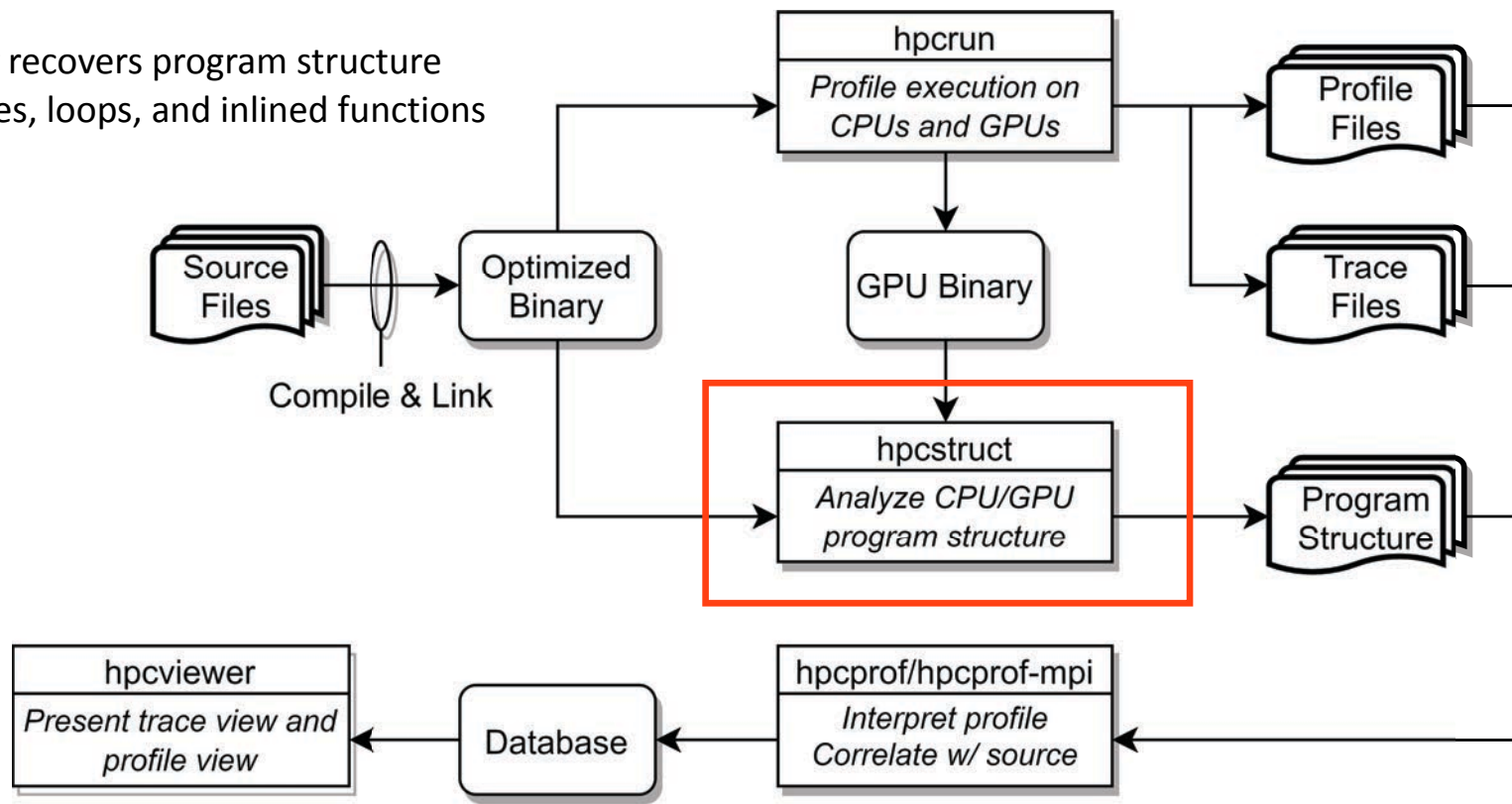
# hpcrun: Measure CPU and/or GPU activity

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

# HPCToolkit's Workflow for GPU-accelerated Applications

Step 3:

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



# hpcstruct: Analyze CPU and GPU Binaries Using Multiple Threads

- Usage

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

- What it does

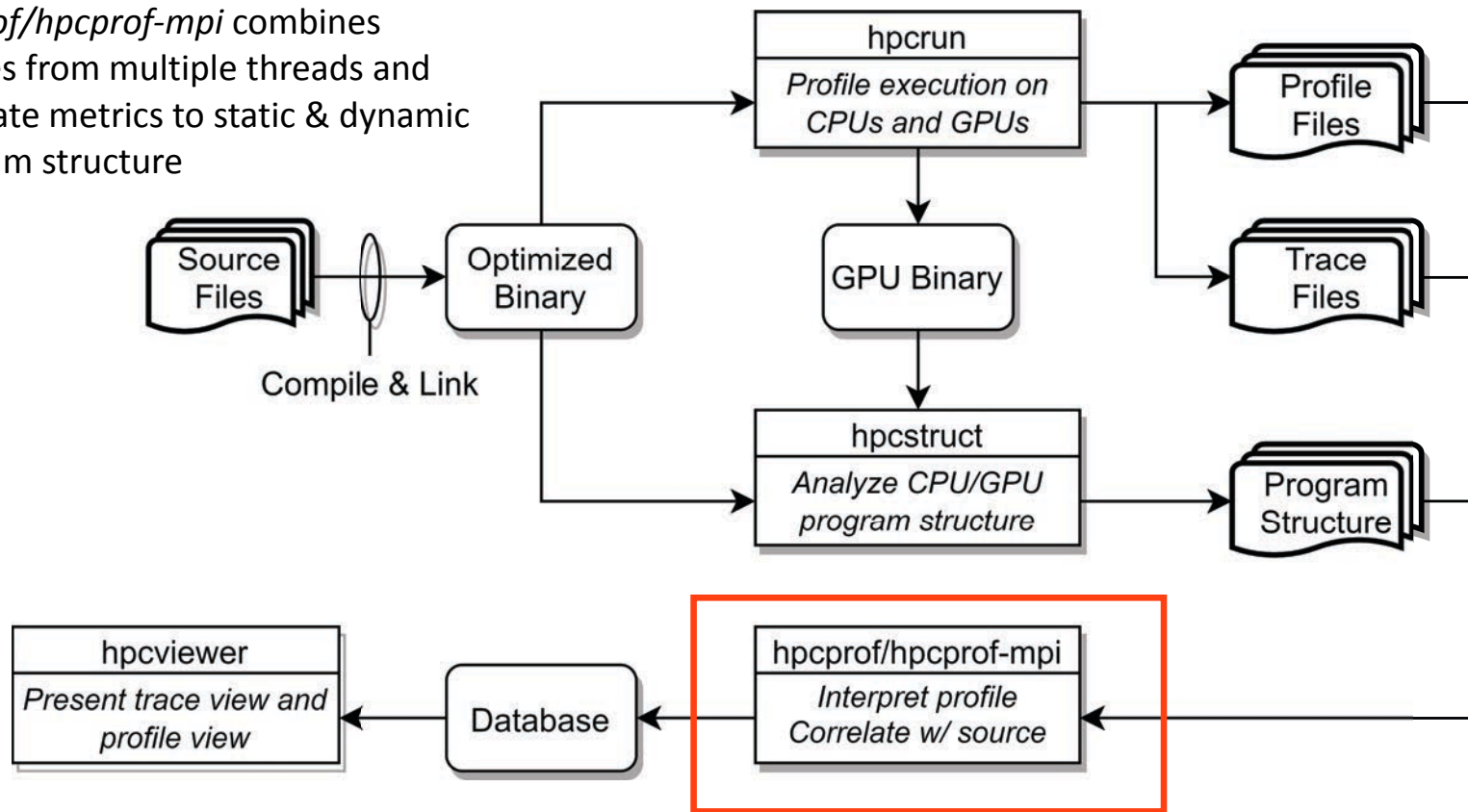
- Recover program structure information
  - Files, functions, inlined templates or functions, loops, source lines
- In parallel, analyze all CPU and GPU binaries that were measured by HPCToolkit
  - typically analyze large application binaries with 16 threads
  - typically analyze multiple small application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

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

# HPCToolkit's Workflow for GPU-accelerated Applications

Step 4:

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



# hpcprof/hpcprof-mpi: Associate Measurements with Program Structure

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

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

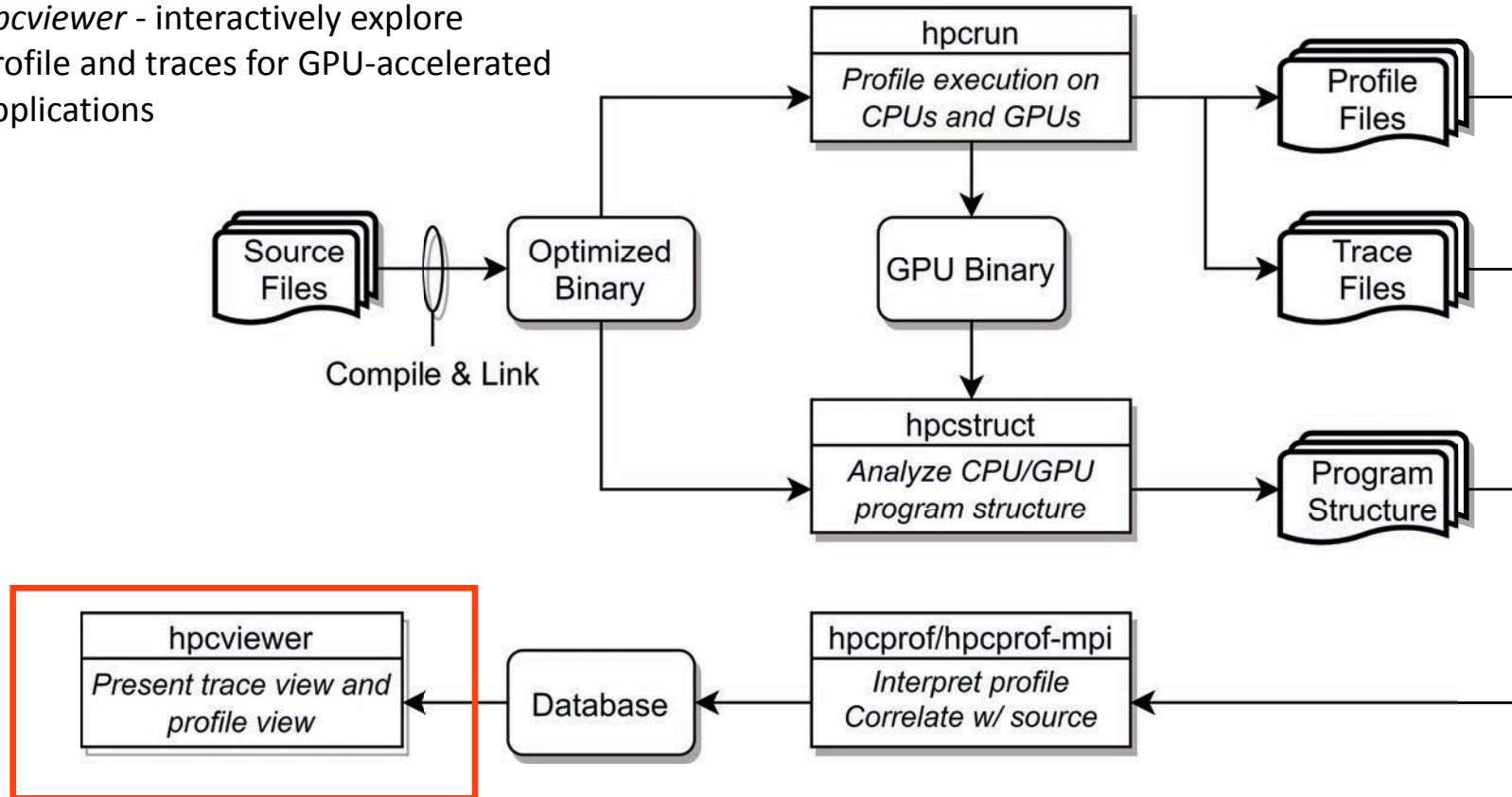
- Analyze data from large executions with distributed-memory parallelism + multithreading (large scale)

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

# HPCToolkit's Workflow for GPU-accelerated Applications

Step 4:

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



# Code-centric Analysis with hpcviewer

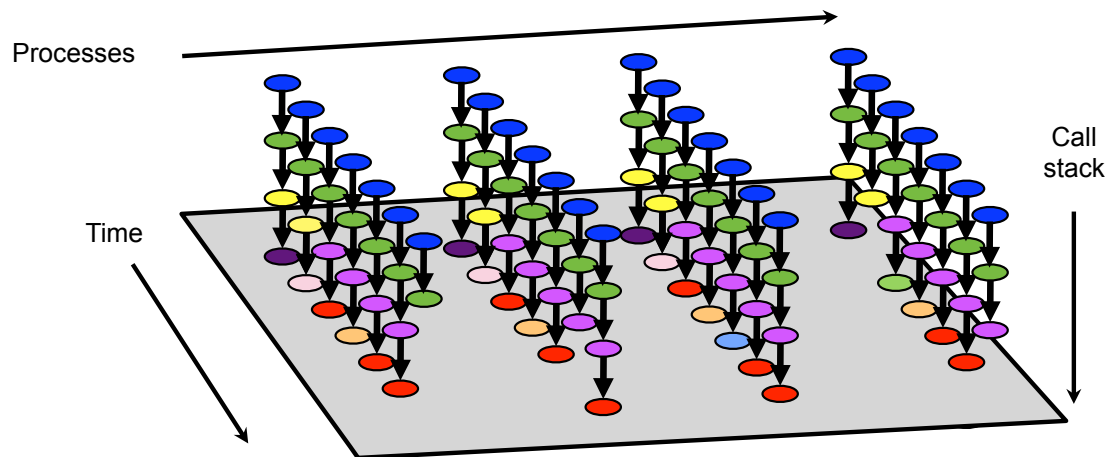
The screenshot displays the hpcviewer application interface. At the top, the 'Profile' and 'Trace' tabs are set to 'lulesh-RAJA-parallel.exe'. The 'main.c' file is open, showing C++ code for the RAJA library. A red box labeled 'source pane' highlights the code. Below the code, a red box labeled 'view control' points to the 'Top-down view', 'Bottom-up view', and 'Flat view' buttons. To the right, a red box labeled 'metric display' points to the 'Scope' tree, which shows a hierarchical view of the execution metrics. A red box labeled 'navigation pane' points to the 'Scope' tree. On the far right, a red box labeled 'metric pane' points to the 'REALTIME (usec):Sum (I)' and 'REALTIME (usec):Sum (E)' columns, which display performance data for various execution units.

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



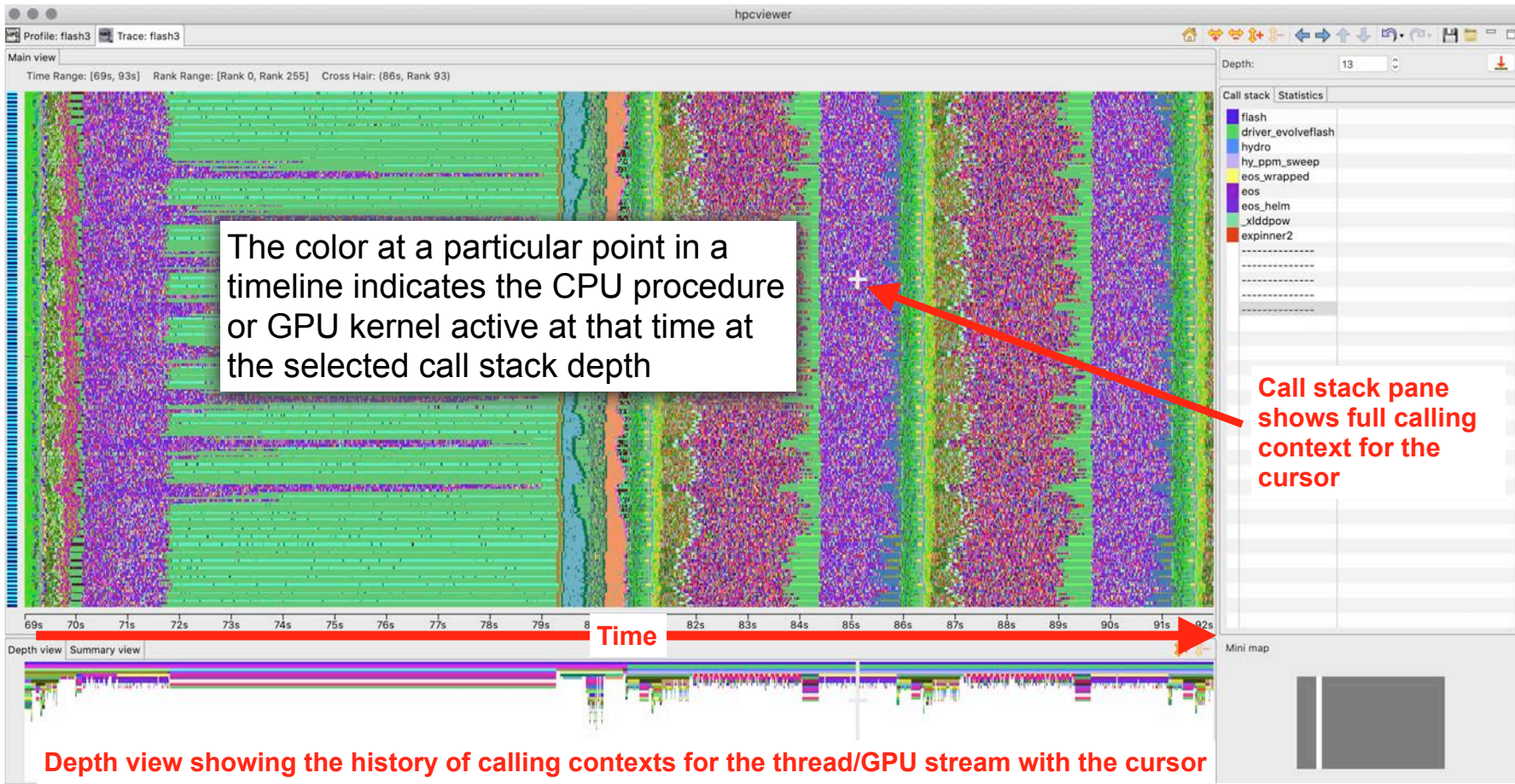
# Understanding Temporal Behavior

- 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

MPI ranks, OpenMP Threads, GPU streams

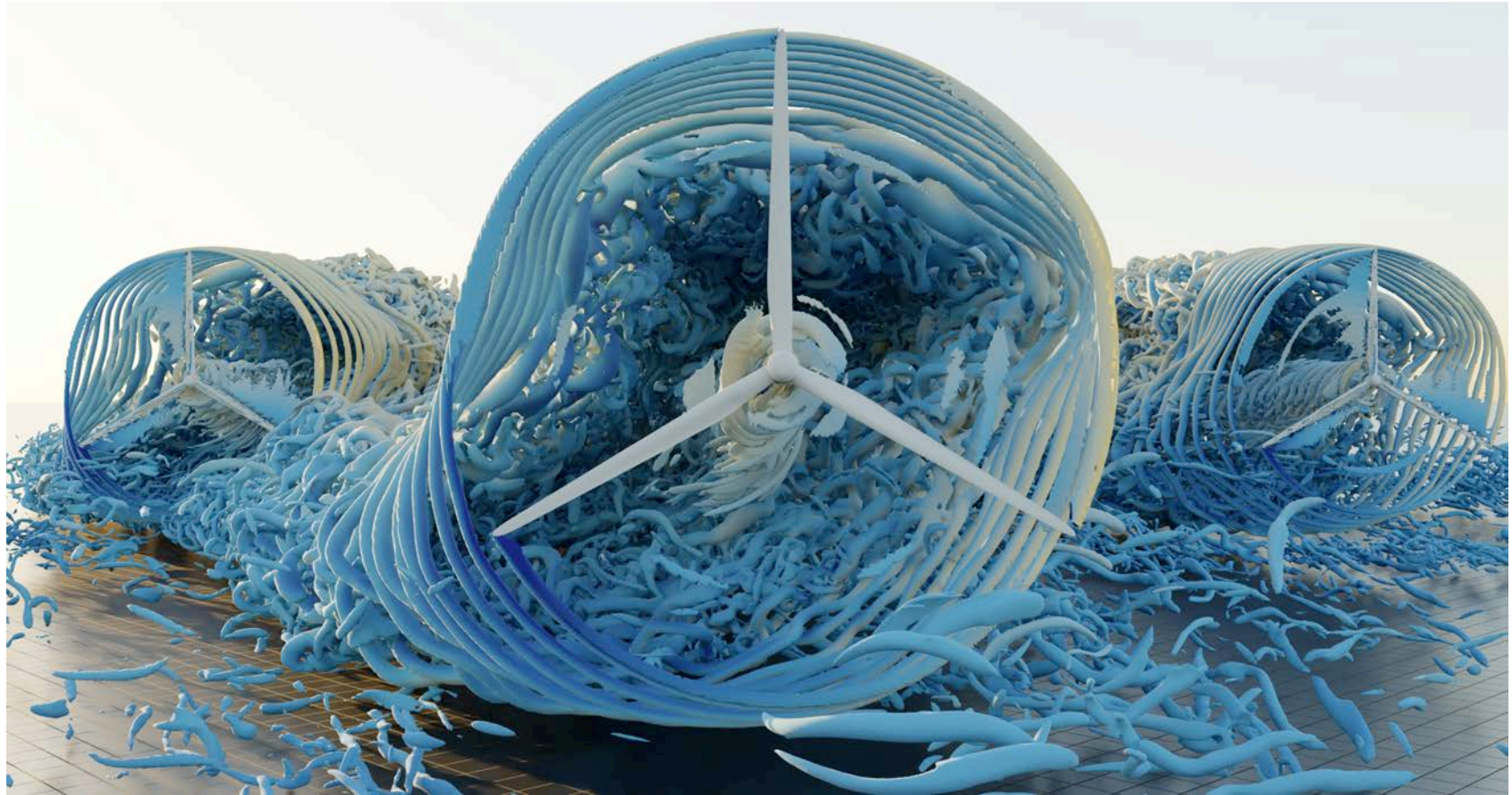


# Case Studies

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

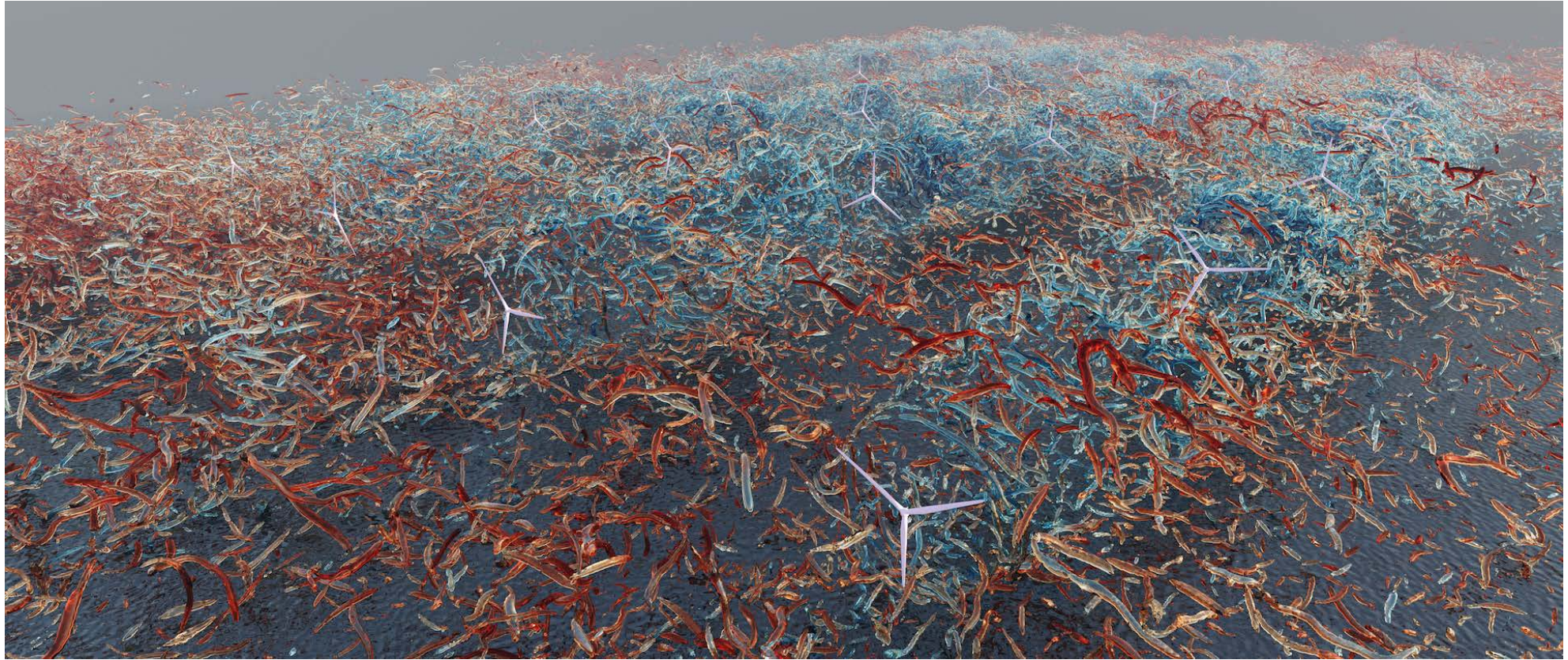


# ExaWind: Wakes from Three Turbines over Time





# ExaWind: Visualization of a Wind Farm Simulation





# 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

After: MPI overhead negligible\*

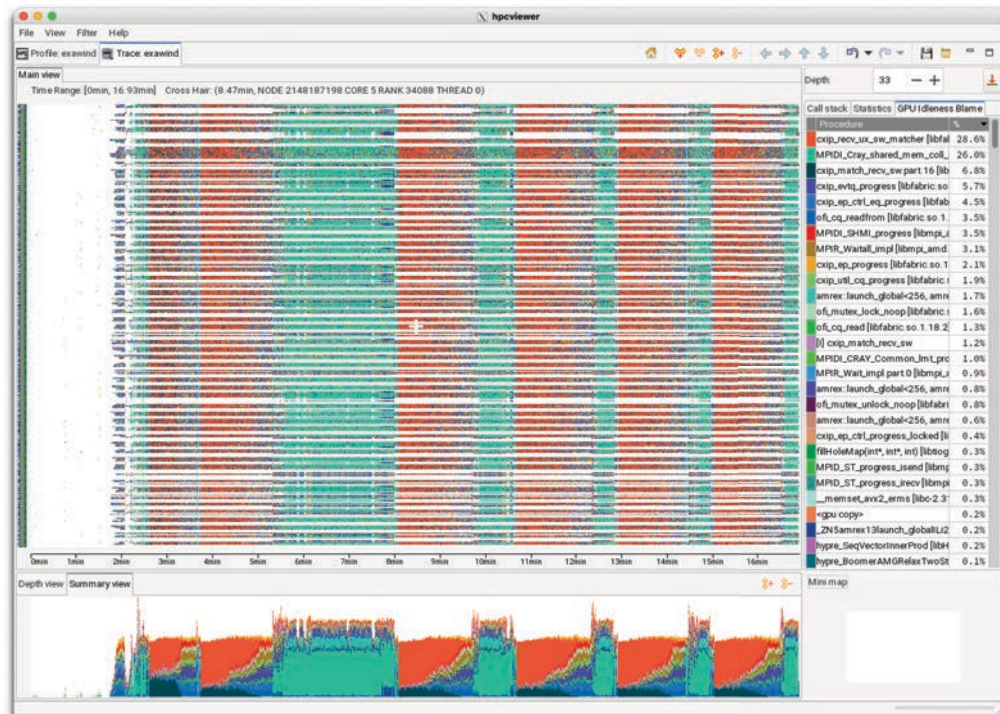


Figure credits: Jon Rood, NREL

\*replaced non-blocking send/recv with ialltoallv

# ExaWind Testimonials for HPCToolkit

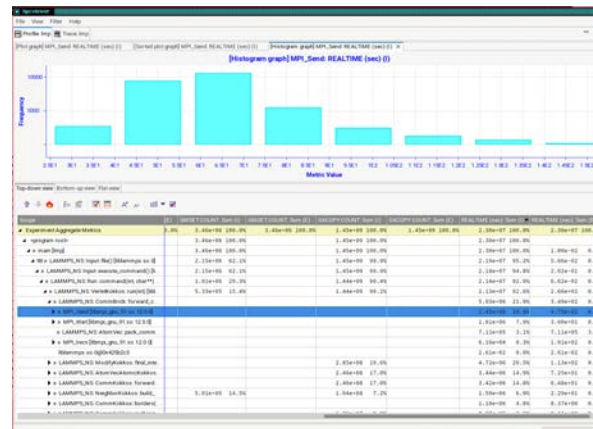
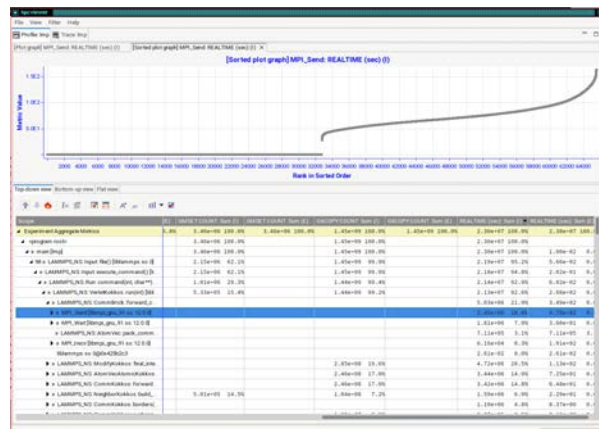
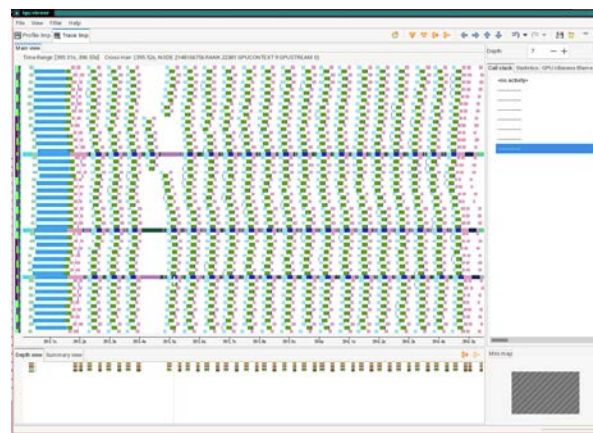
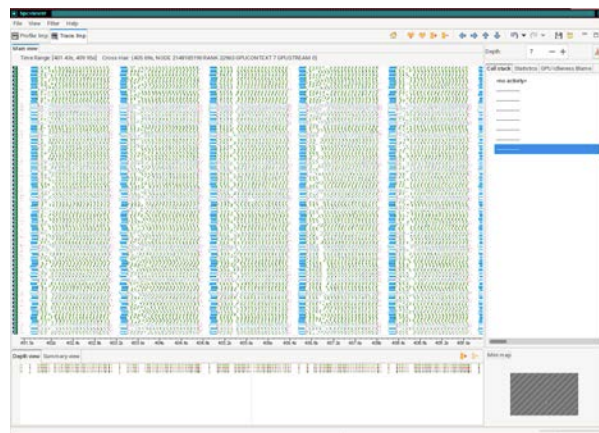
*I just wanted to mention we've been using HPCToolkit a lot for our ExaWind application on Frontier, which is **a hugely complicated code**, and **your profiler is one of the only ones we've found that really lets us easily instrument and then browse what our application is doing at runtime including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

*- Jon Rood NREL (5/31/2024)*

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

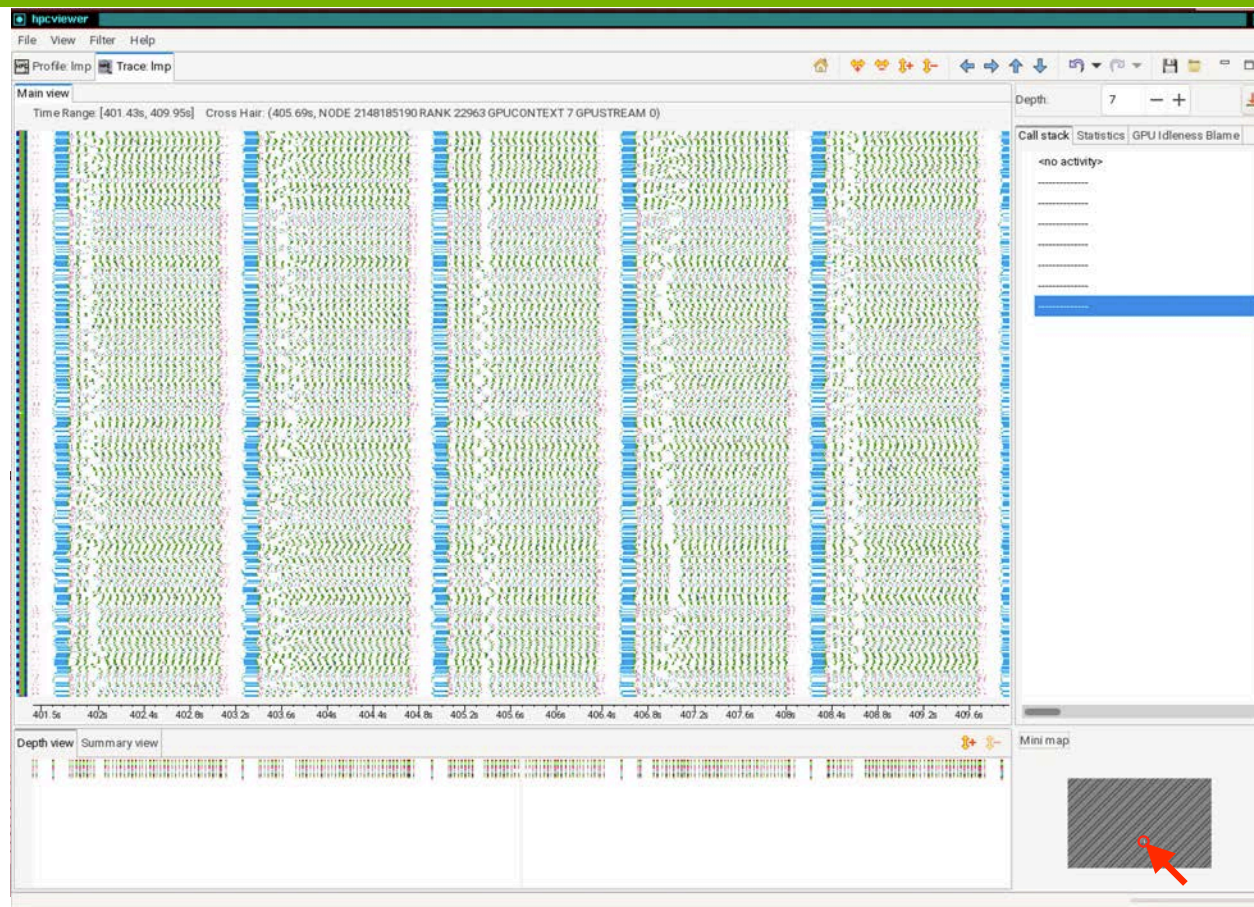
*- Jon Rood NREL (6/3/2024)*

# LAMMPS on Frontier: Executions with Kernel Duration of Milliseconds

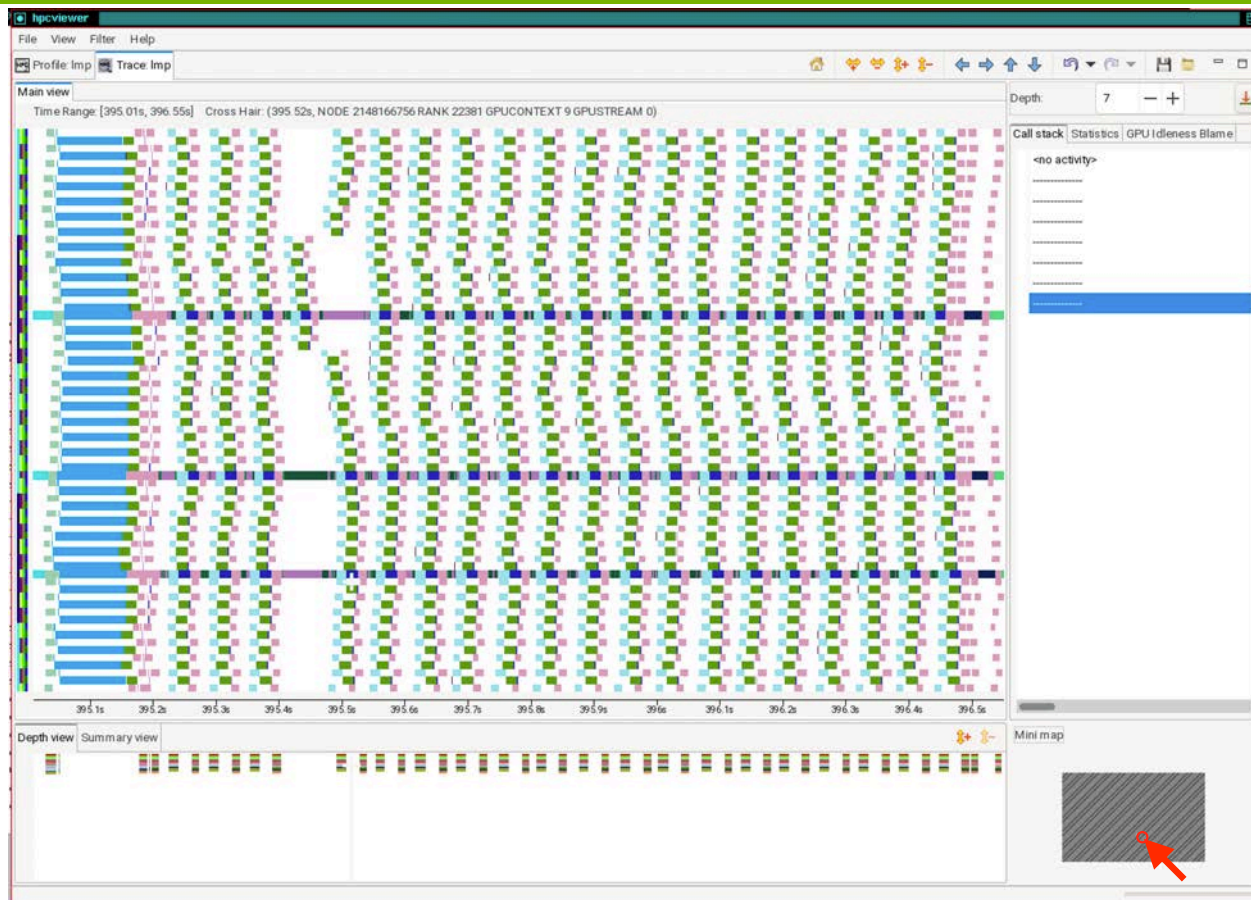




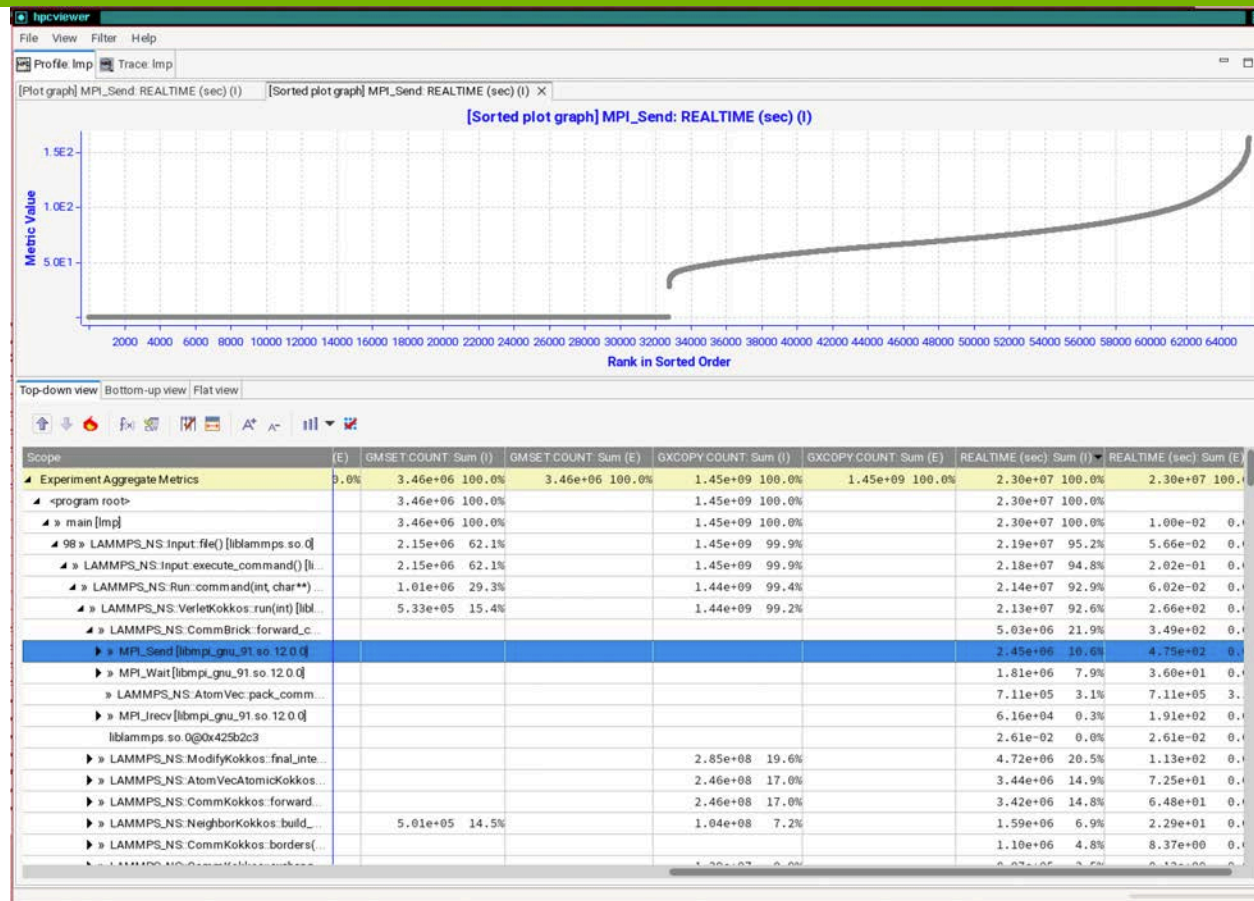
# LAMMPS on Frontier: Executions with Kernel Duration of Milliseconds



# LAMMPS on Frontier: Executions with Kernel Duration of Milliseconds

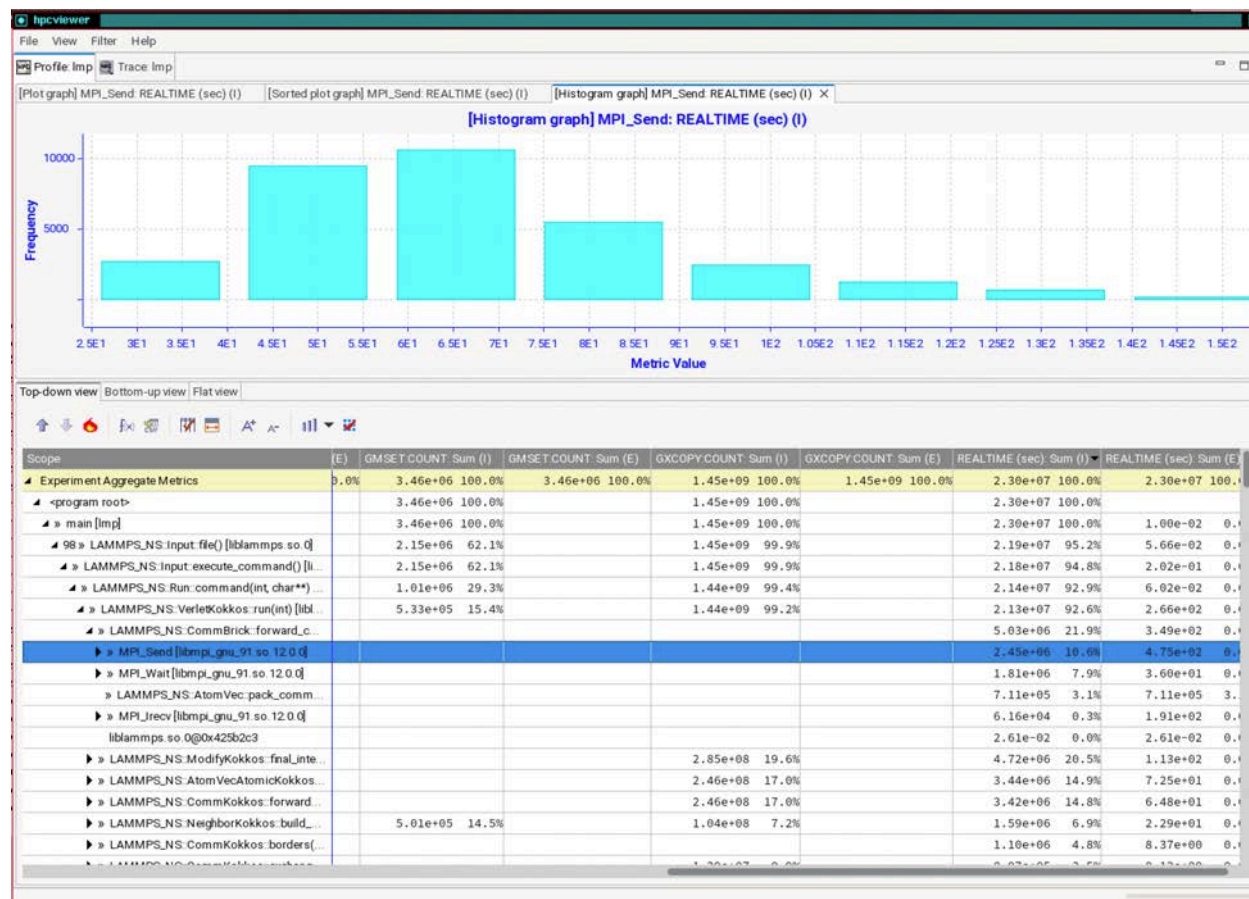


# LAMMPS on Frontier: Executions with Kernel Duration of Milliseconds



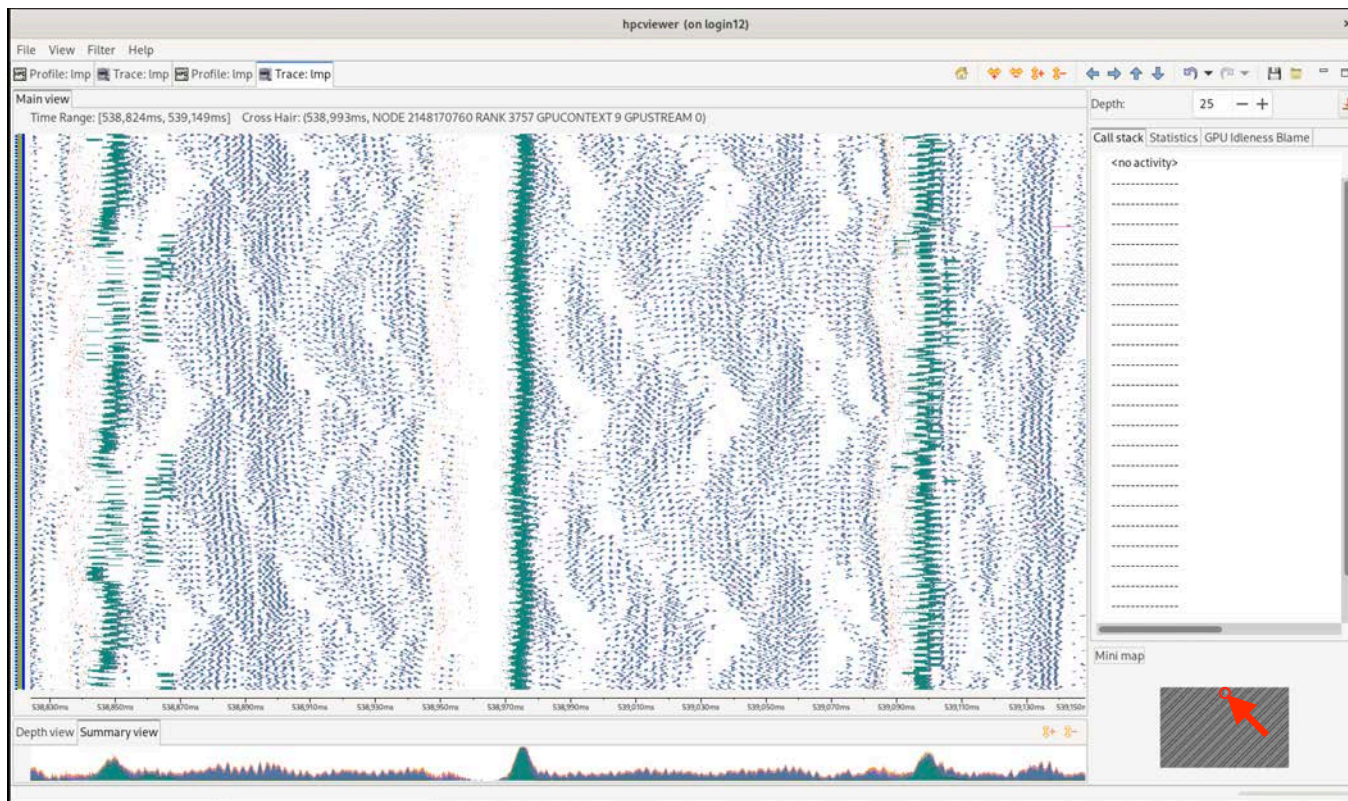


# LAMMPS on Frontier: Executions with Kernel Duration of Milliseconds



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

Kernel duration of microseconds



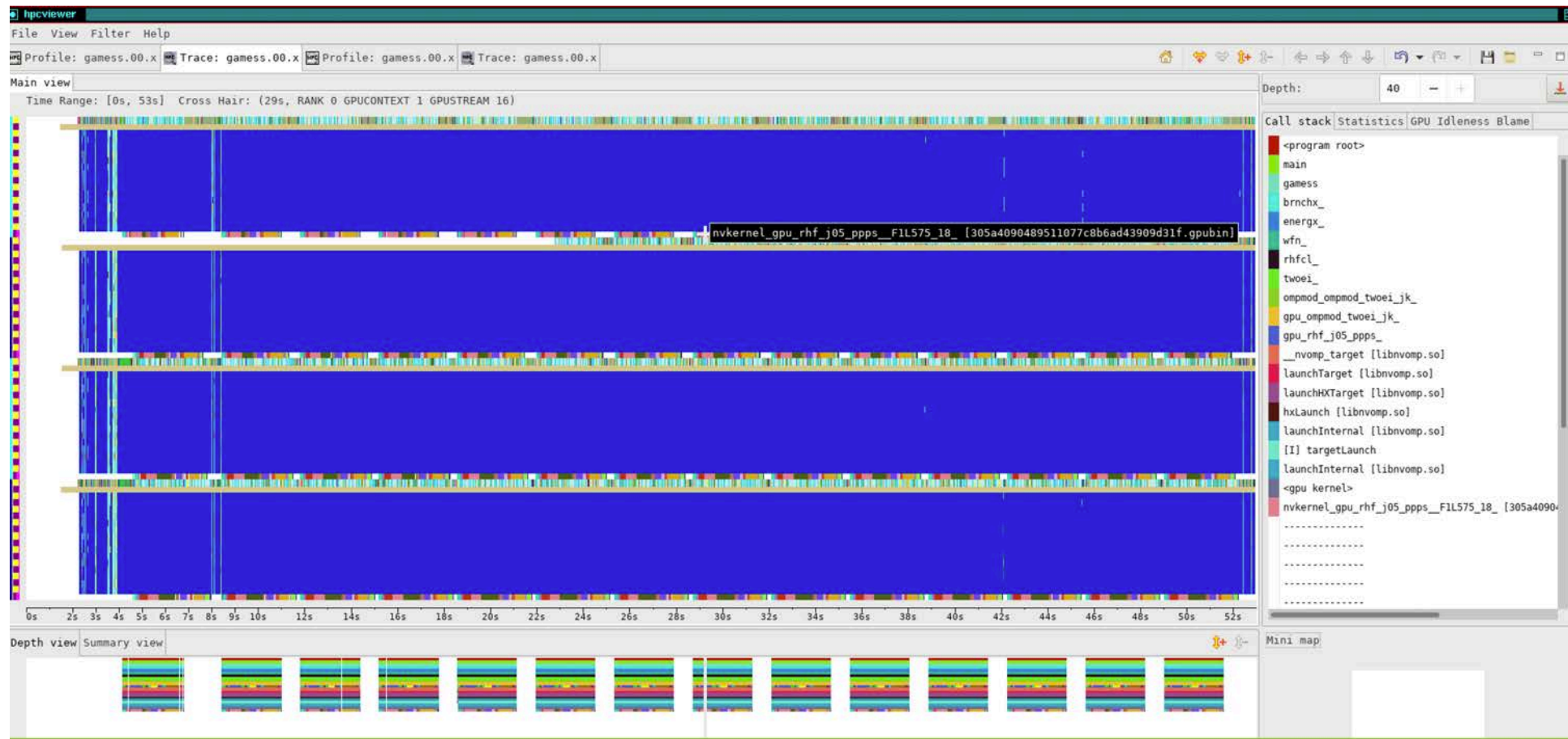
# Case Study: GAMESS

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

## Perlmutter node at a glance

AMD Milan CPU  
4 NVIDIA A100 GPUs  
256 GB memory

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

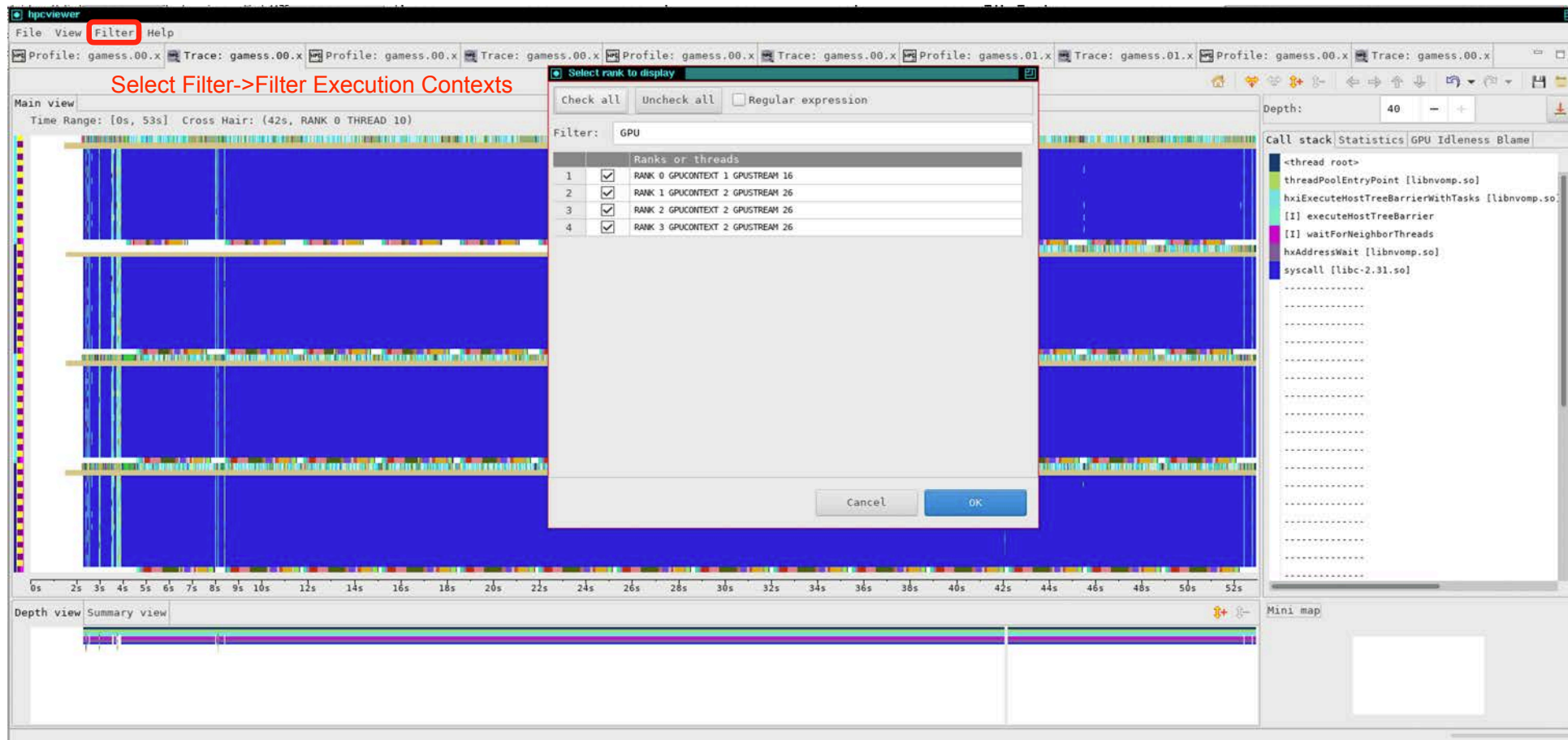


GAMESS original

All CPU threads and GPU streams



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

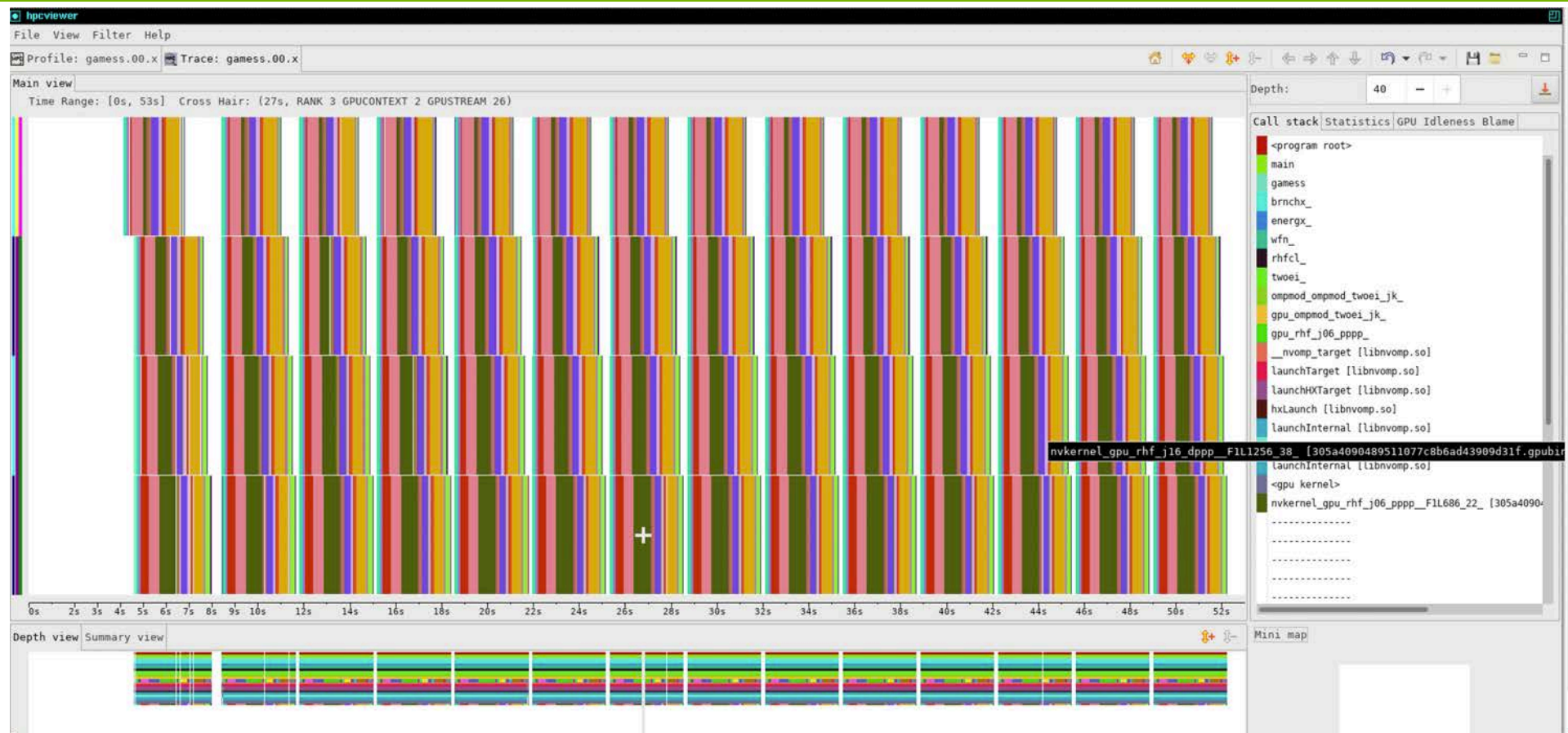


GAMESS original

All CPU threads and GPU streams



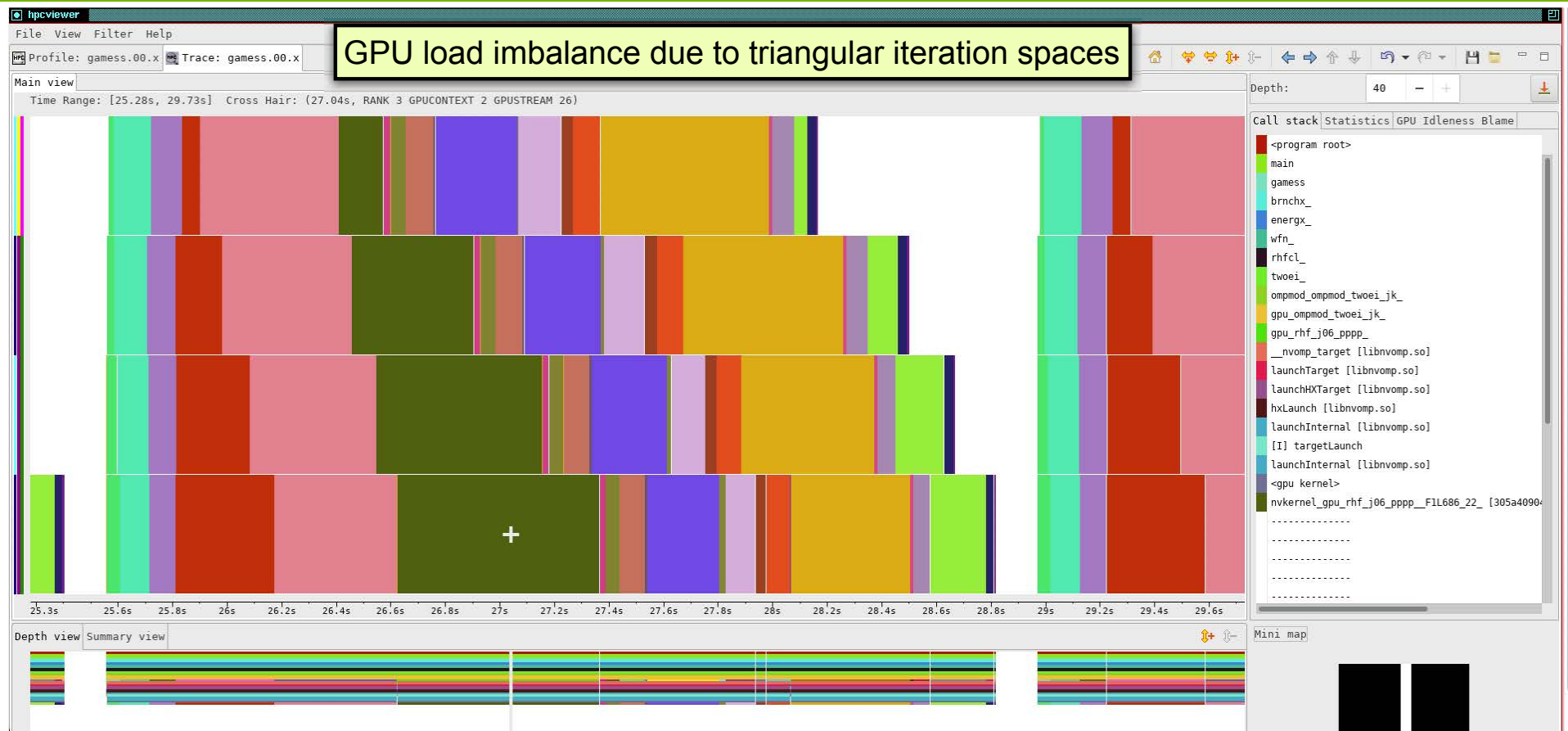
# Time-centric Analysis: GAMESS 4 ranks, 4 GPUs on Perlmutter



GAMESS original

All GPU streams; whole execution

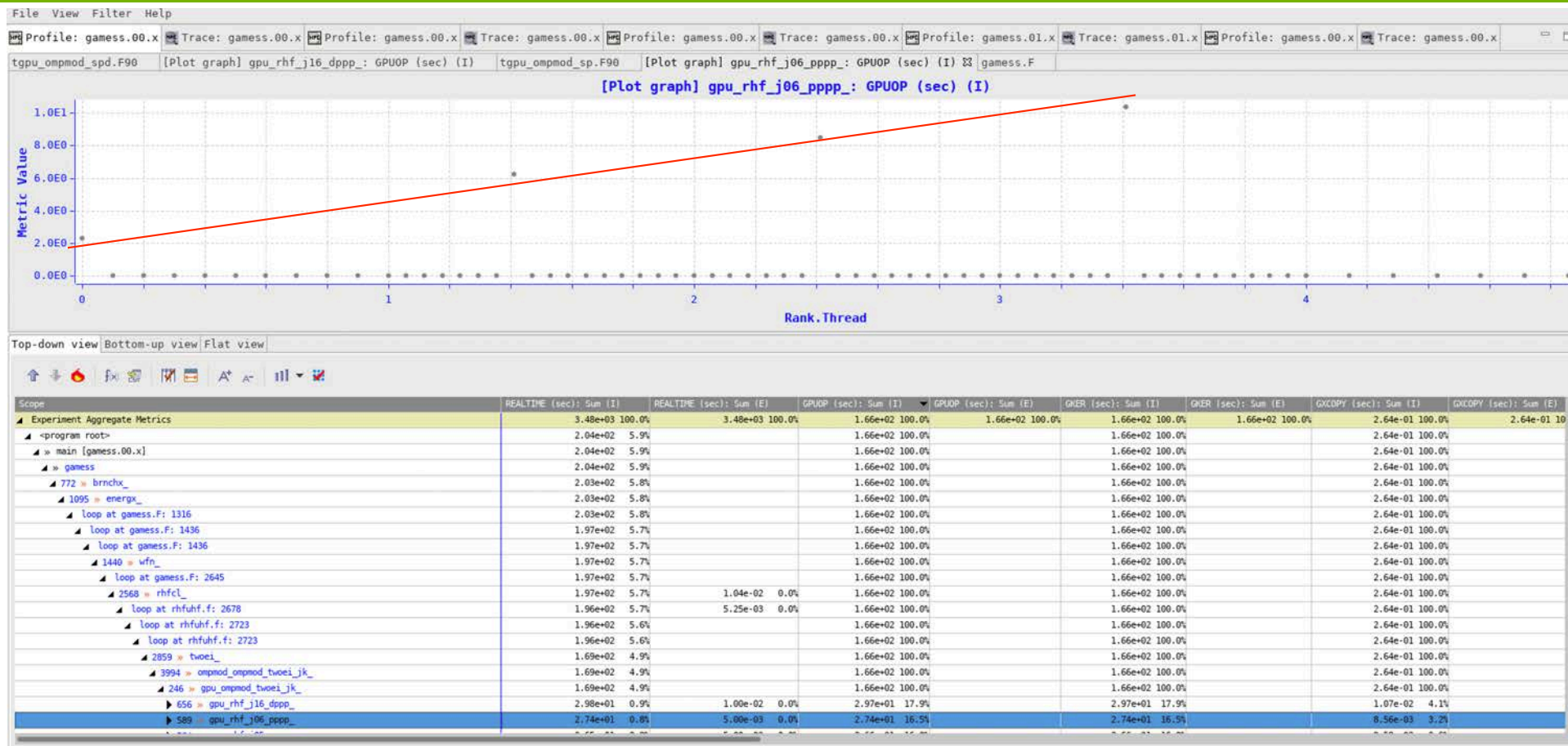
# Time-centric Analysis: GAMESS 4 ranks, 4 GPUs on Perlmutter



GAMESS original

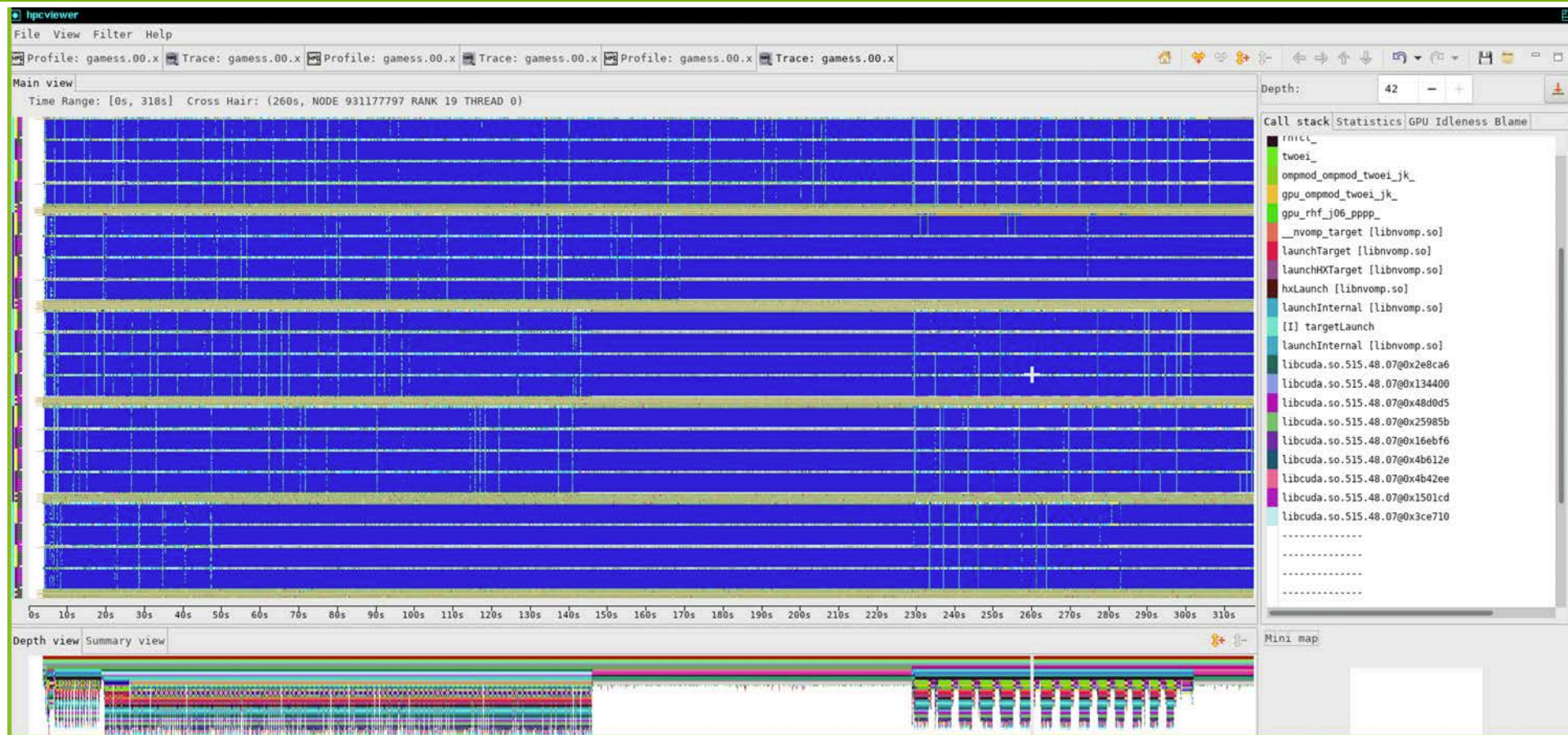
GPU streams: 1 iteration

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



GAMESS original

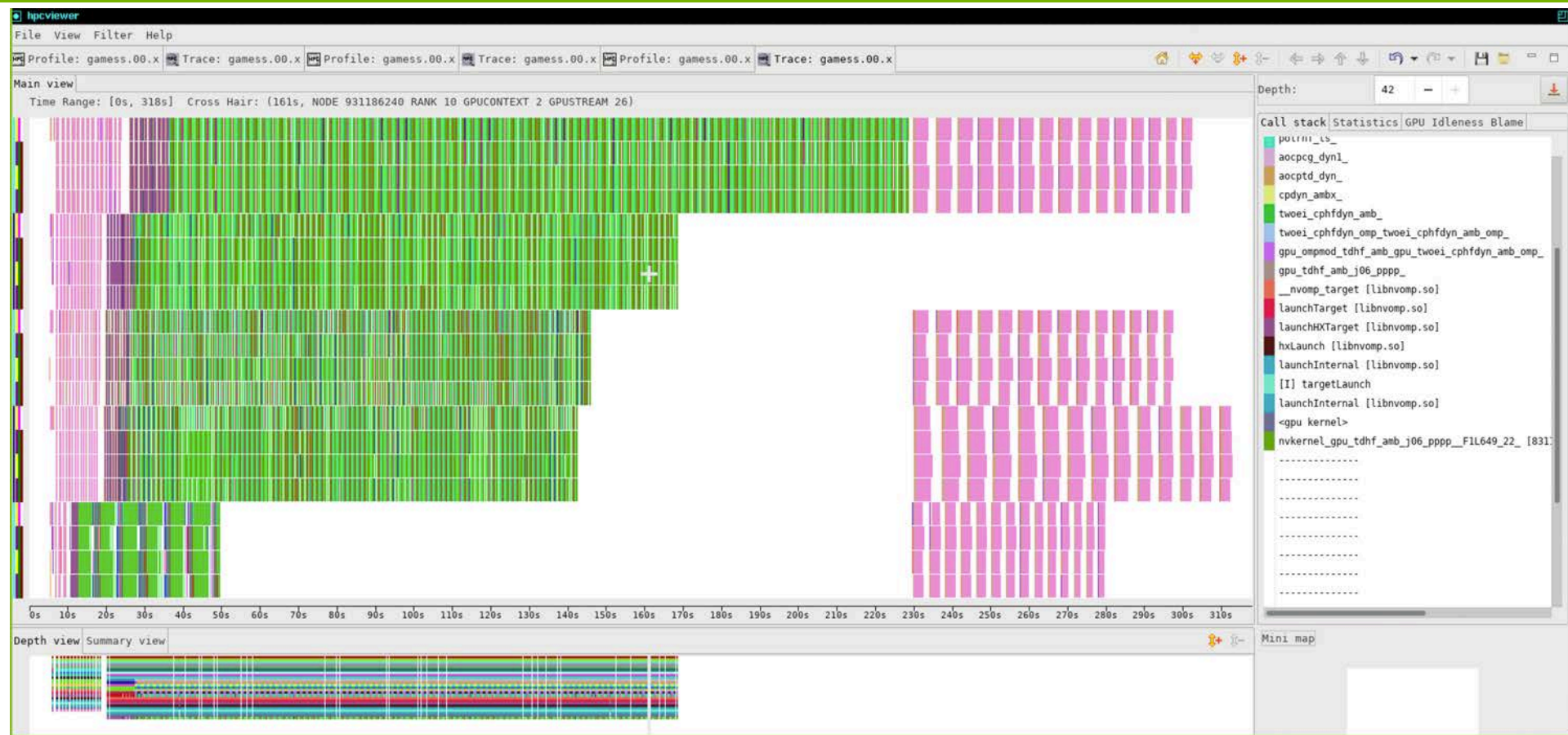
# Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter



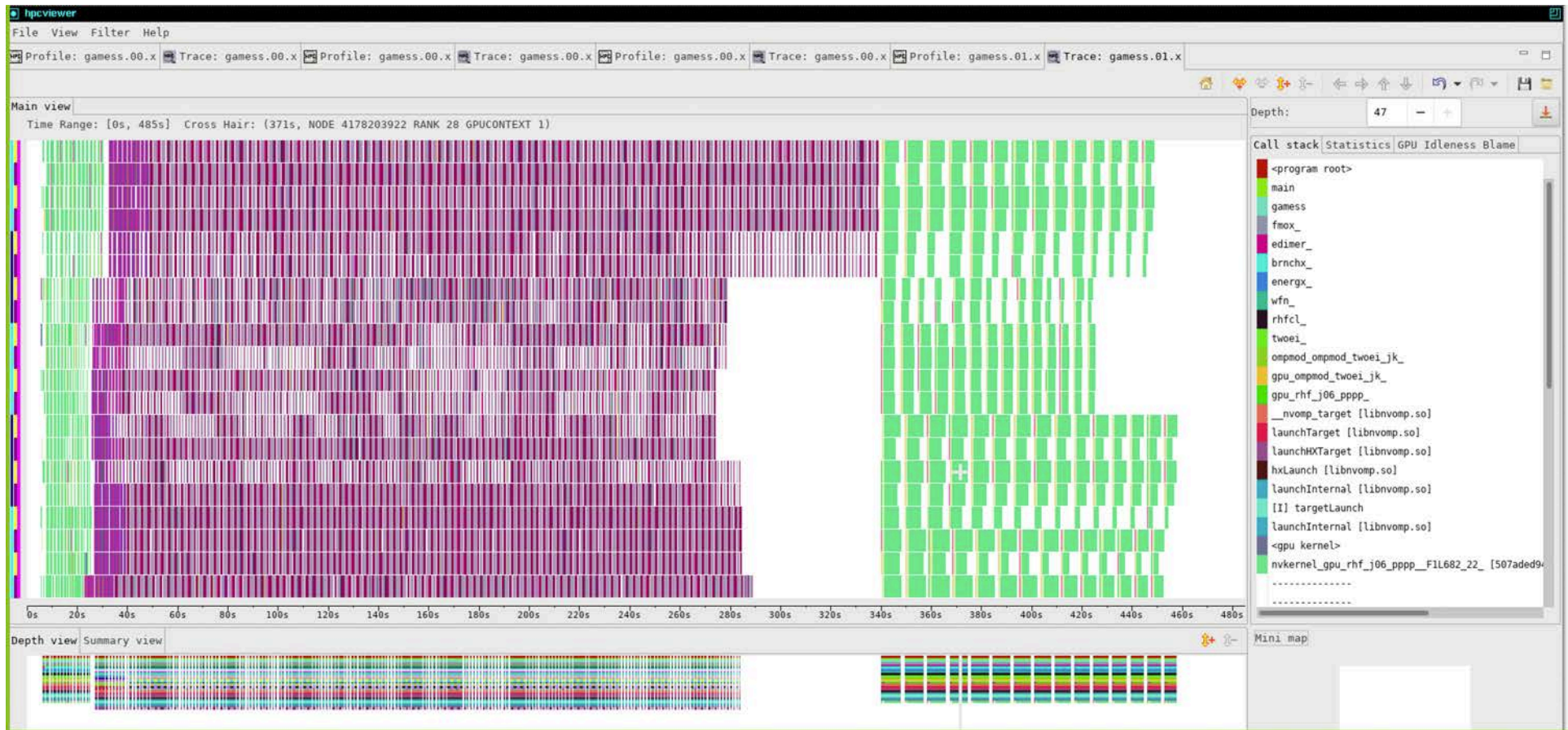
CPU Threads and GPU Streams



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



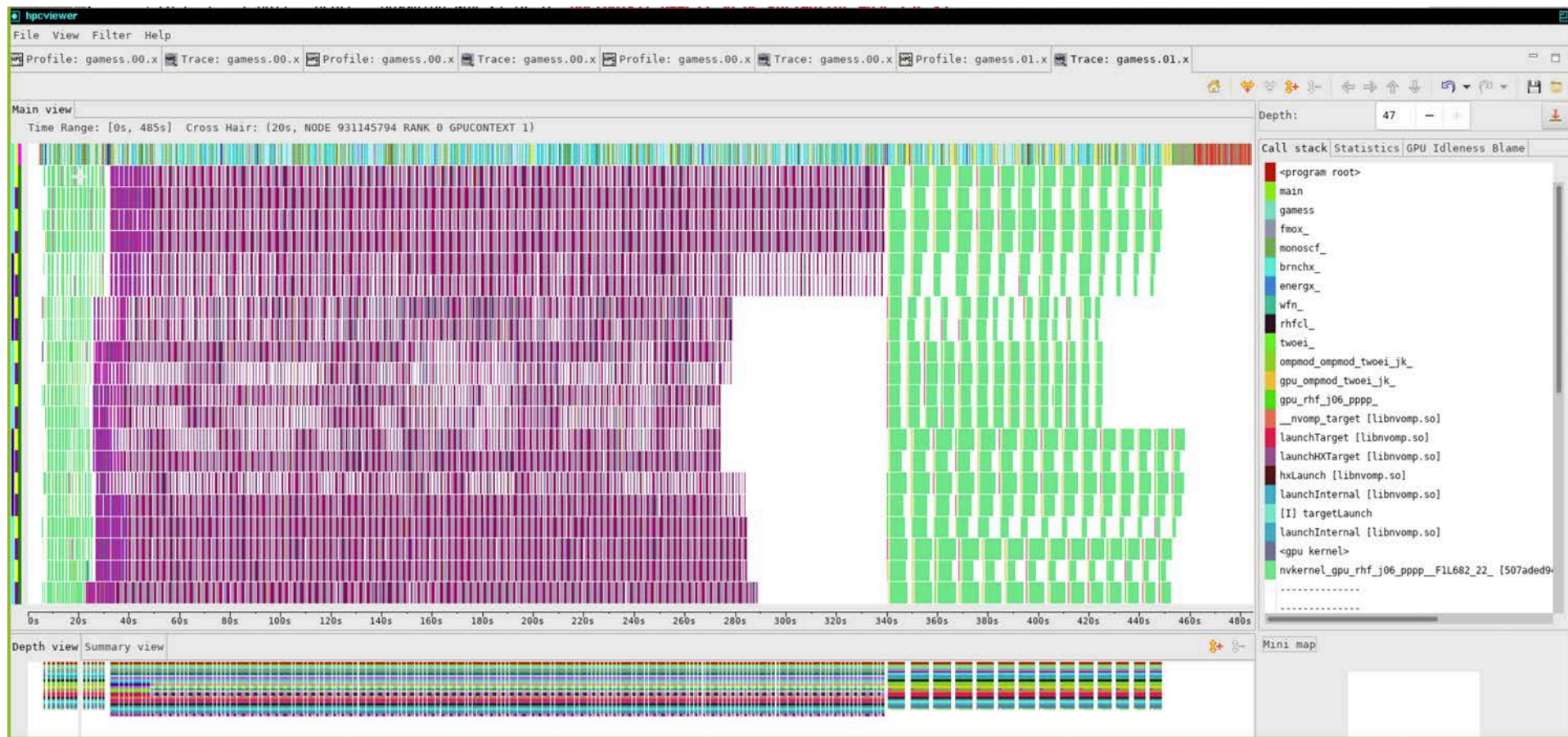
# Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter



GAMESS improved with better manual distribution of work in input

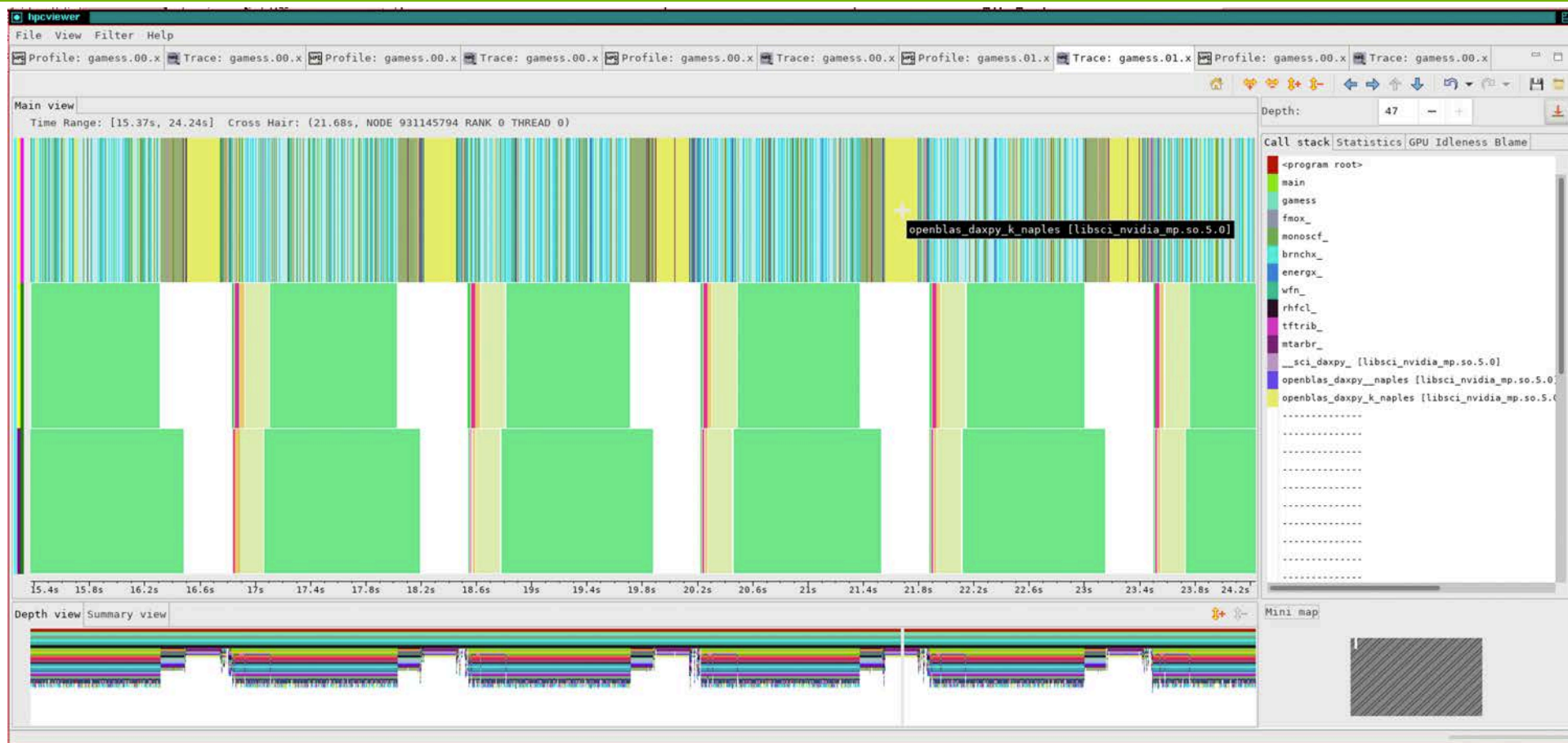


# Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter



GAMESS improved adding Rank 0 Thread 0 to GPU streams

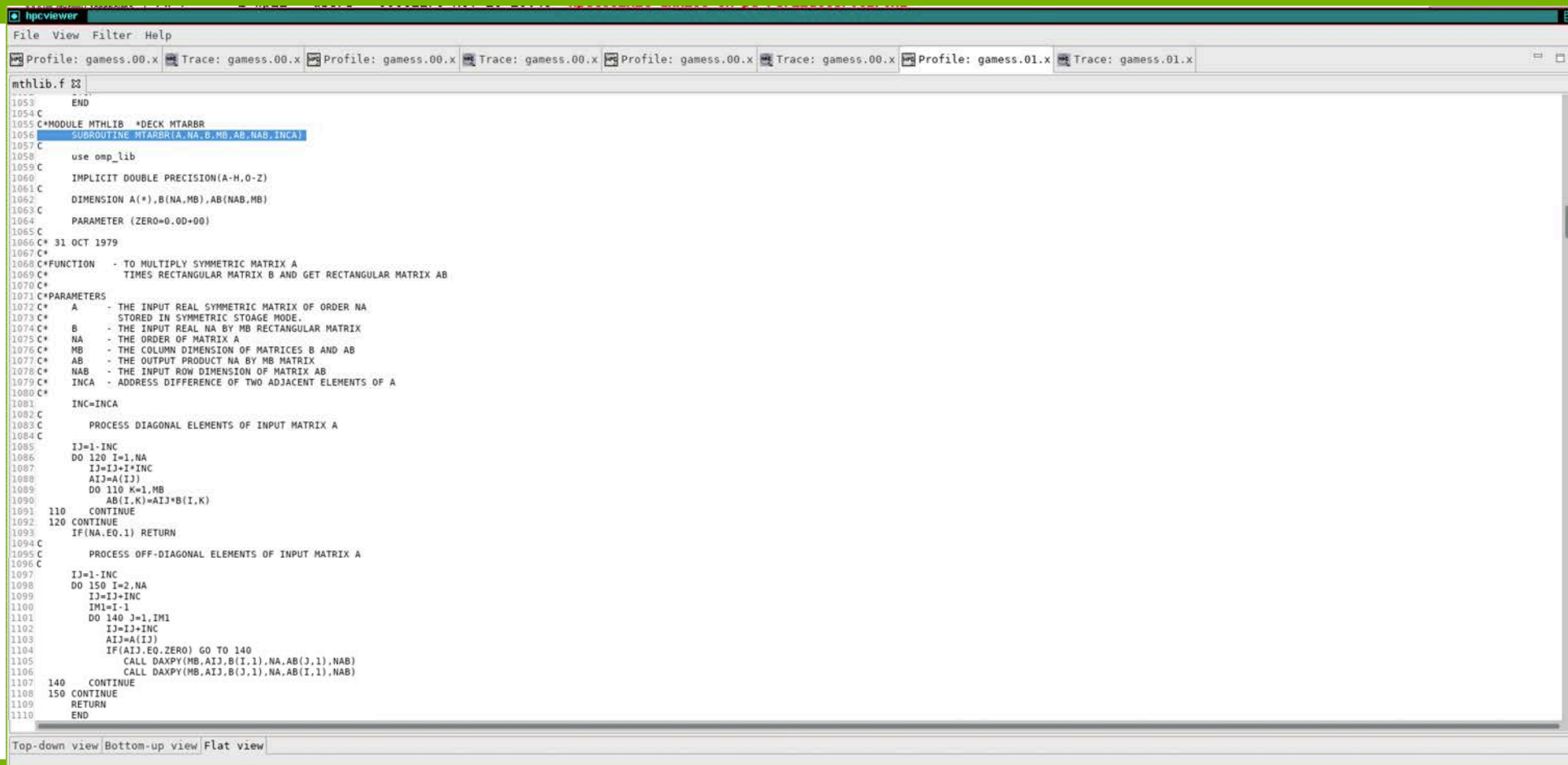
# Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter



1 CPU Stream, 2 GPU Streams: 6 Iterations

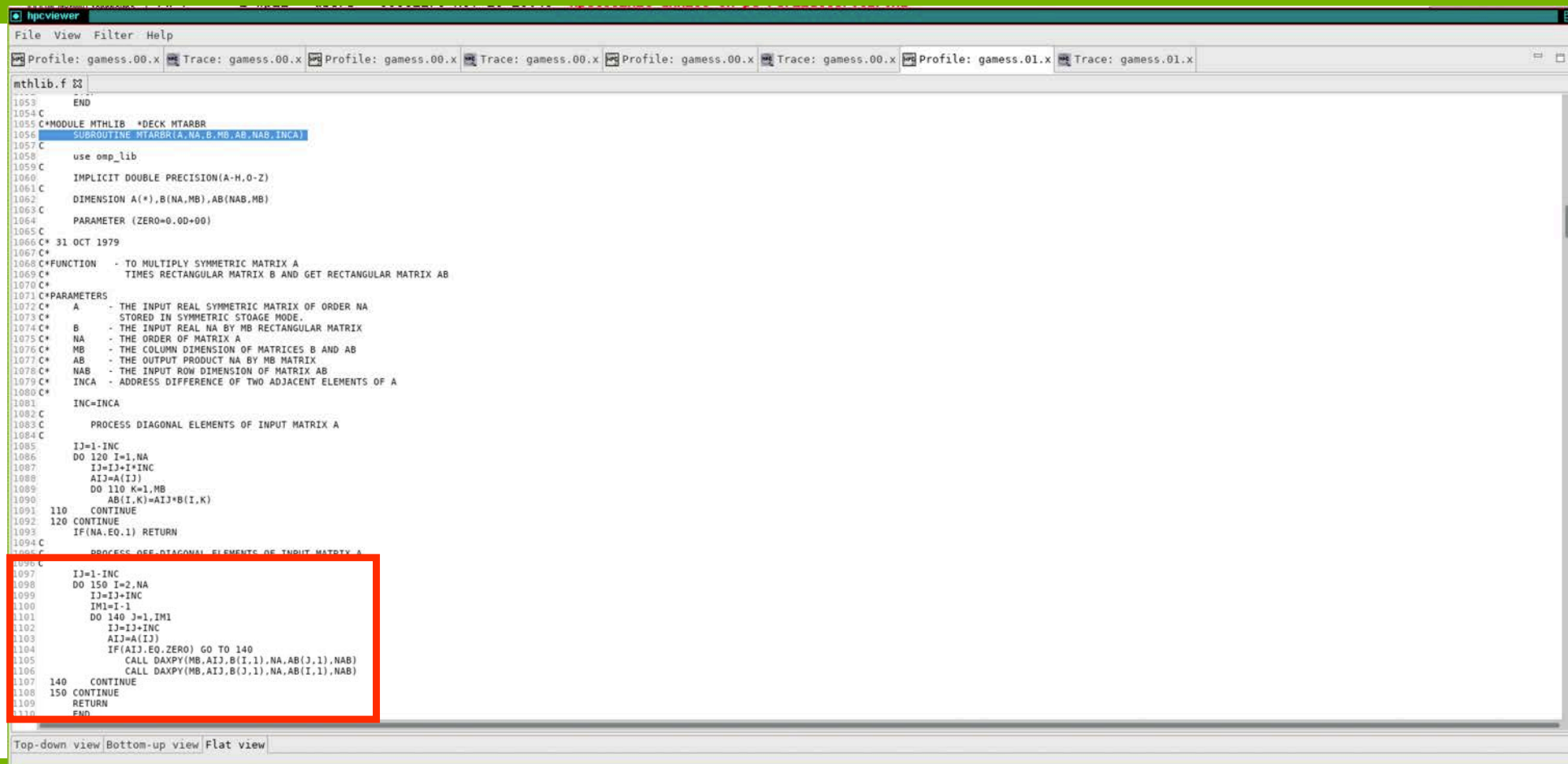


# Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter



```
mthlib.f
END
1053 C
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    ABI(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
```

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



```
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
END
1053 C
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
```

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

```
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
```

# 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, 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 interface. The top pane shows the source code for `CollisionEvent.cc` with line numbers 69 to 85. The bottom pane shows a performance analysis table with columns for 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), and GINS: STL\_IDEP. The table lists various scopes, including `cudaLaunchKernel`, `gpu kernel`, `CycleTrackingKernel`, `CycleTrackingGuts`, `CycleTrackingFunction`, `CollisionEvent`, and `macroscopicCrossSection`. The `macroscopicCrossSection` scope is highlighted in blue.

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 » [1] 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 » [1] CycleTrackingFunction(MonteCarlo*, MC_Particle&, int, P...	8.36e+10 64.4%	4.12e+08 0.3%	7.25e+10 61.1%	3.65e+08 0.3%	5.21e+09 98.9%	1.02e+08 1.9%	9.25e+
» loop at CycleTracking.cc: 118	8.35e+10 64.3%	3.76e+08 0.3%	7.25e+10 61.1%	3.34e+08 0.3%	5.21e+09 98.8%	9.90e+07 1.9%	9.24e+
63 » CollisionEvent(MonteCarlo*, MC_Particle&, unsigned int) [...]	5.20e+10 40.1%	4.99e+09 3.8%	4.44e+10 37.4%	4.02e+09 3.4%	3.85e+09 73.1%	4.89e+08 9.3%	6.37e+
» loop at CollisionEvent.cc: 67	4.09e+10 31.5%	8.15e+08 0.6%	3.42e+10 28.8%	6.54e+08 0.6%	3.54e+09 67.1%	1.27e+08 2.4%	5.67e+
» loop at CollisionEvent.cc: 71	3.85e+10 29.6%	2.70e+09 2.1%	3.22e+10 27.1%	2.06e+09 1.7%	3.27e+09 62.0%	2.28e+08 4.3%	5.33e+
73 » macroscopicCrossSection(MonteCarlo*, int, int, int, 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%			4.63e+
» qs_vector<NuclearDataReaction>::operator[](int)	3.35e+08 0.3%	3.35e+08 0.3%	3.43e+08 0.3%	3.43e+08 0.3%	1.43e+08 0.3%	1.43e+08 0.3%	7.33e+

# Quicksilver: Detailed analysis within a Kernel using PC Sampling

```
Scope
  ▲ 14 » [1] 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)
```



# HPCToolkit Resources

- Documentation
  - User manual for HPCToolkit: <http://hpctoolkit.org/manual/HPCToolkit-users-manual.pdf>
  - Cheat sheet: <https://gitlab.com/hpctoolkit/hpctoolkit/-/wikis/HPCToolkit-cheat-sheet>
  - User manual for hpcviewer: <https://hpctoolkit.gitlab.io/hpcviewer>
  - Tutorial videos
    - <http://hpctoolkit.org/training.html>
    - recorded demo of GPU analysis of Quicksilver: <https://youtu.be/vixa3hGDuGg>
    - recorded tutorial presentation including demo with GPU analysis of GAMESS: <https://vimeo.com/781264043>
- Software
  - Download hpcviewer GUI binaries for your laptop, desktop, cluster, or supercomputer
    - OS: Linux, Windows, MacOS
    - Processors: x86\_64, aarch64, ppc64le
    - <http://hpctoolkit.org/download.html>
  - Install HPCToolkit on your Linux desktop, cluster, or supercomputer using Spack
    - <http://hpctoolkit.org/software-instructions.html>

# Hands-on

# Two Kinds of Hands-on Examples

- Pre-collected databases to explore
  - gain experience using hpctoolkit's hpcviewer graphical user interface to analyze performance data
- Hands-on examples
  - build, run, and view several codes to get the full experience
    - hpcrun: measure an application as it executes
    - hpcstruct: recover program structure information for mapping measurements to source code
    - hpcprof: combine measurements with program structure information
    - hpcviewer: explore profiles and traces

# Performance Databases to Explore

- **On an Aurora login node** For X11 forwarding from the compute node to the outside world

```
% qsub -I -X -l select=2,walltime=1:00:00,place=scatter \  
-l filesystems=flare -A ATPESC2025 -q ATPESC
```

- **On an Aurora compute node**

```
% module load hpctoolkit/modulepath  
% module load hpctoolkit/2025.0.0-rc.1  
% cd /flare/ATPESC2025/EXAMPLES/track6-tools/hpctoolkit/data  
% ls  
  
    arborx    qmcpack  gamess    minitest  pelelmex  quicksilver
```

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



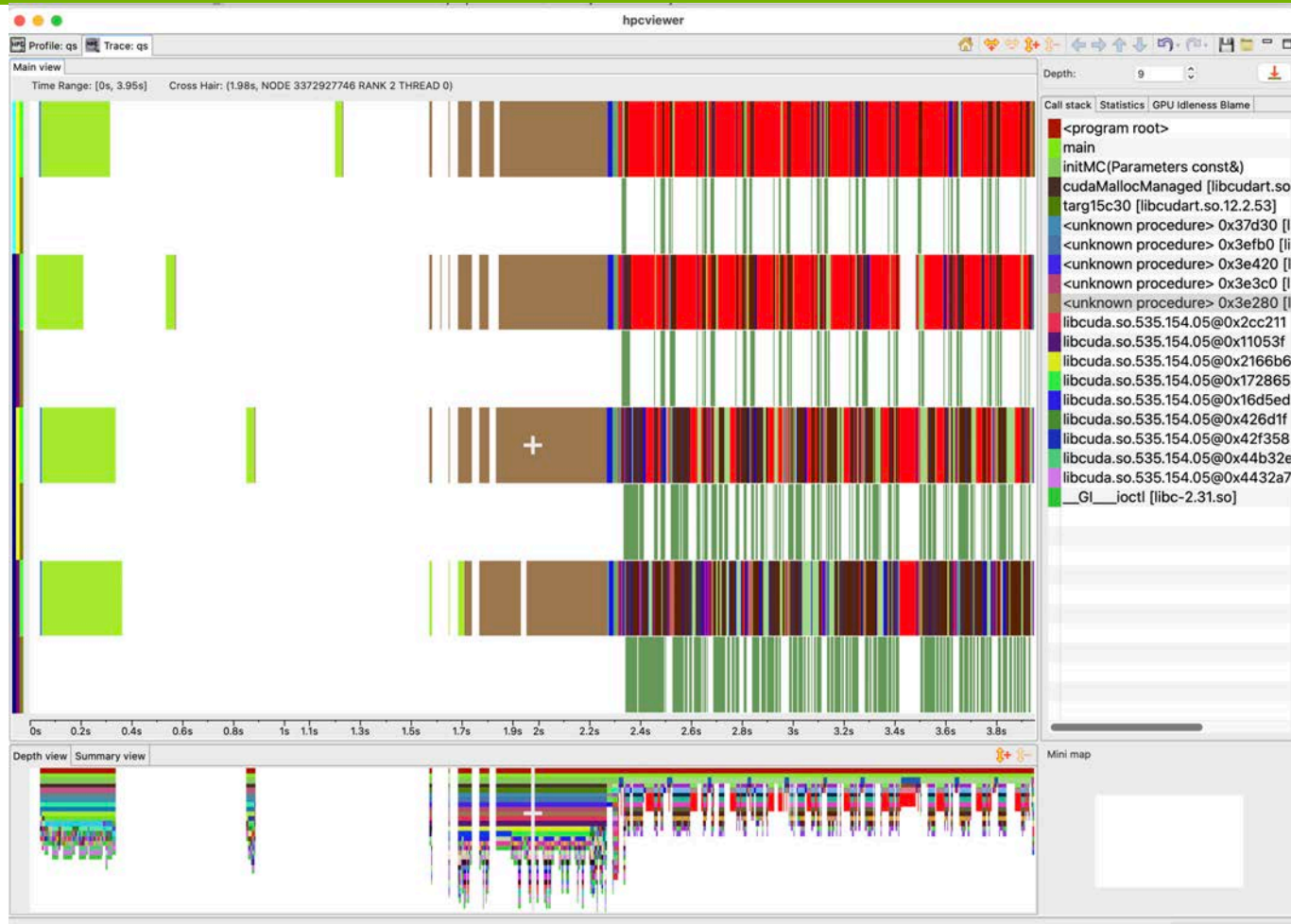
# More about the Available Performance Databases

See </flare/ATPESC2025/EXAMPLES/track6-tools/hpctoolkit/data> for each of the following

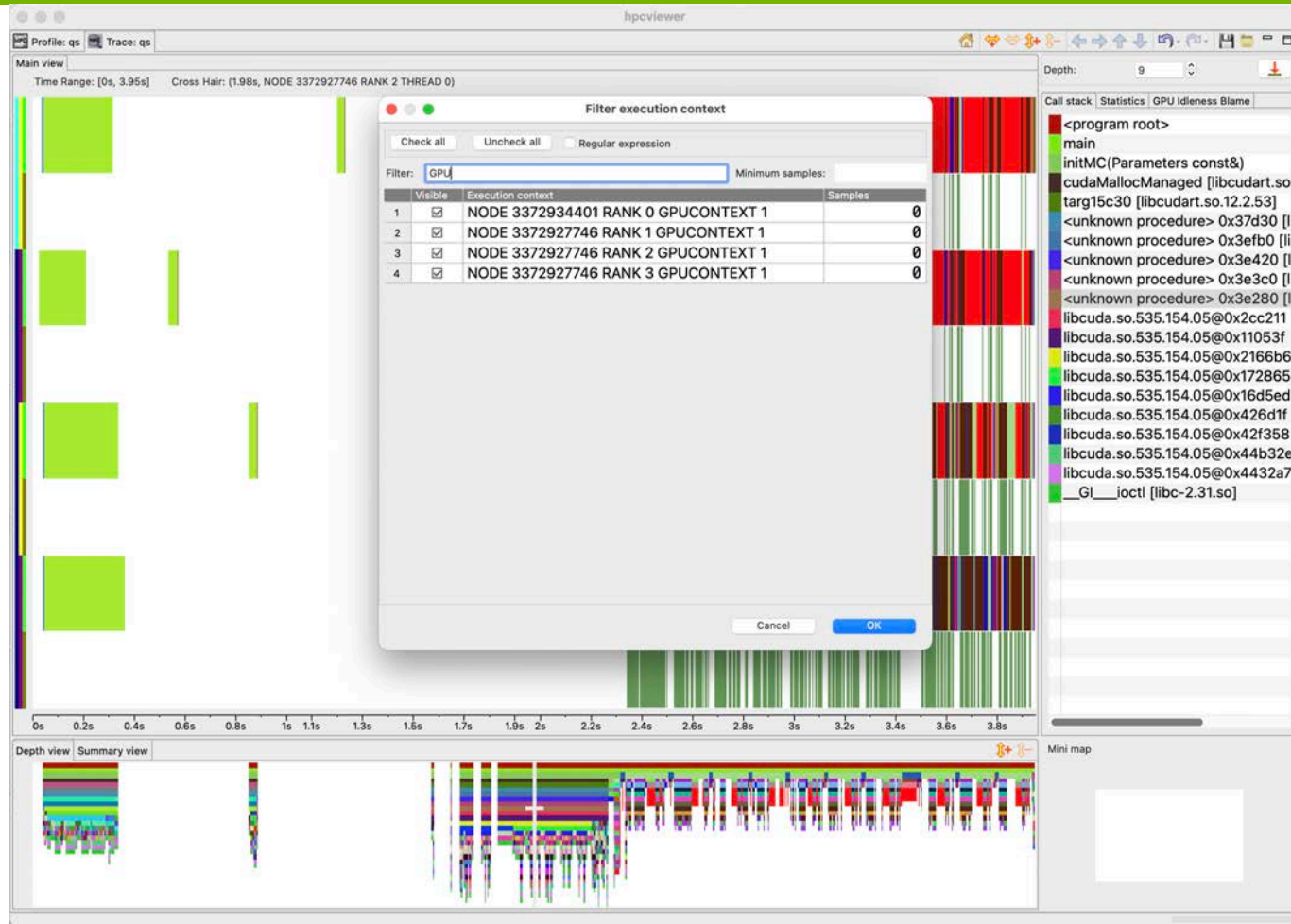
- quicksilver: Monte Carlo particle transport proxy application (**C++ + CUDA**)
  - hpctoolkit-qs-gpu-cuda.d - profile and trace on 4 CPUs + 4 GPUs
  - hpctoolkit-qs-gpu-cuda-pc.d - instruction-level measurements within kernels using PC sampling
  - **EXERCISES**
- pelelmex: Adaptive mesh hydrodynamics simulation code for low Mach number reacting flows (**C++ + AMReX**)
  - pelelmex.db -a large trace with load imbalance from 2025 NERSC hackathon run on 16 CPUs + 16 GPU
- gamess: General Atomic and Molecular Electronic Structure System (**Fortran + OpenMP**)
  - 1.singlegroup-unbalanced/hpctoolkit-gamess-1n-chol-noDS.d
  - 2.singlegroup-balanced/hpctoolkit-gamess-1n-chol-fix\_load\_balance\_noDS.d/
  - 3.multigroup-unbalanced-mtarbr/hpctoolkit-gamess-5n.d/
  - 4.multigroup-balanced/hpctoolkit-gamess-5n-manualbalance.d/
  - 5.multigroup-unbalanced-pc/hpctoolkit-gamess-5n-pc.d/
  - 6.scale/hpctoolkit-gamess-22n-test.d/
- arborx (**C++ + Kokkos**)
- minitest (**SYCL and OpenMP TARGET**)

# Inspecting Pre-collected Quicksilver Data

# Select the Tab “Trace: qs”

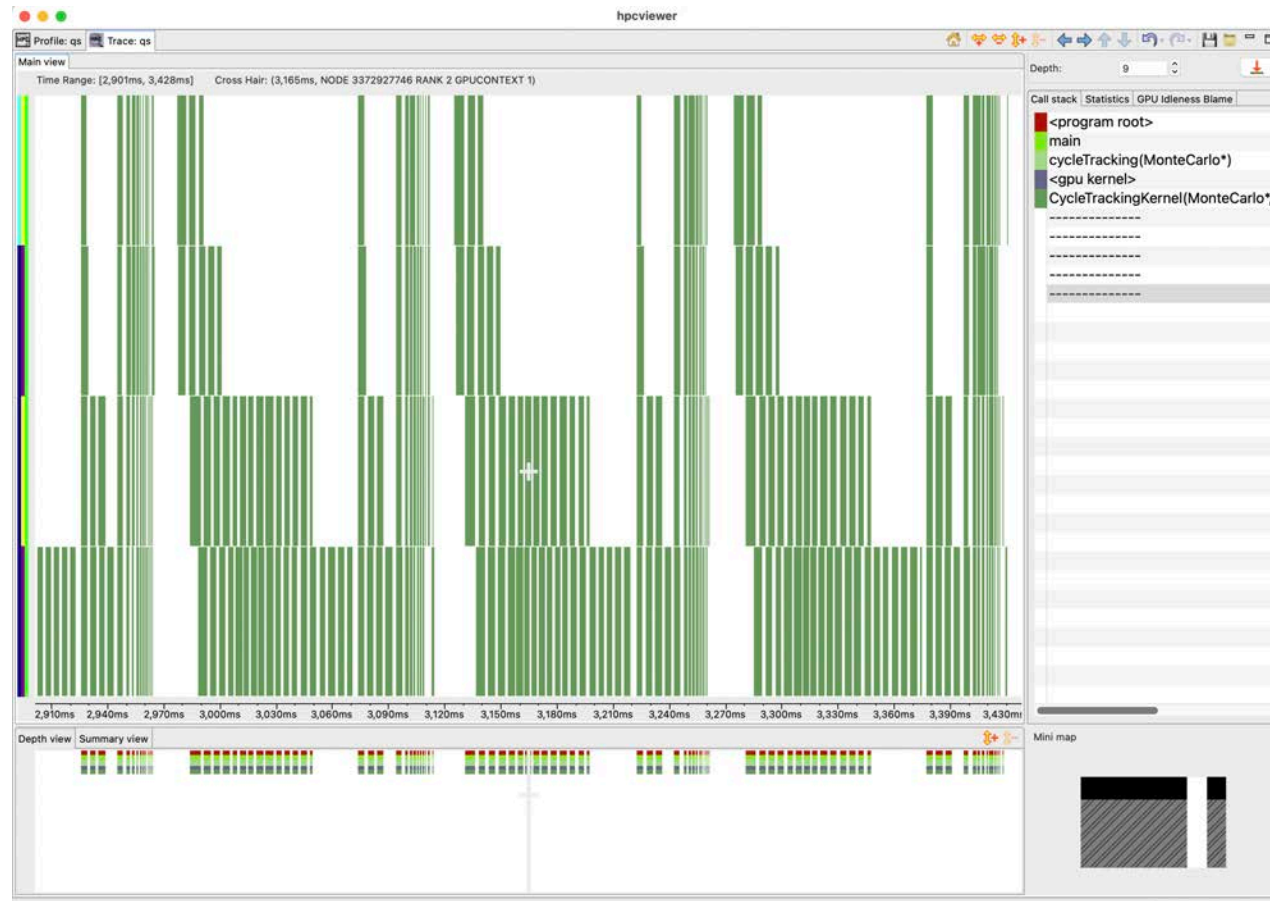


# Use the Filter to “Uncheck all” and Check “GPU” streams






# See Load Imbalance Across the Four GPUs






# Analyzing Quicksilver Traces

## Using a measurement database with profiles and traces

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

# Analyzing Quicksilver Traces

## Using a measurement database with profiles and traces

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

# Analyzing Quicksilver Traces




## Using a measurement database with profiles and traces

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



# Analyzing the Quicksilver Summary Profile

## Using a measurement database with profiles and traces

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

# Analyzing the Quicksilver Summary Profile

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

# The Profile View in the other “PC Sampling” Database

hpcviewer

Profile: qs Trace: qs Profile: qs

CollisionEvent.cc

```

67 for (int isoIndex = 0; isoIndex < numIsos && currentCrossSection >= 0; isoIndex++)
68 {
69     int uniqueNumber = monteCarlo->_materialDatabase->_mat[globalMatIndex]._iso[isoIndex]._gid;
70     int numReacts = monteCarlo->_nuclearData->getNumberReactions(uniqueNumber);
71     for (int reactIndex = 0; reactIndex < numReacts; reactIndex++)
72     {
73         currentCrossSection -= macroscopicCrossSection(monteCarlo, reactIndex, mc_particle.domain, mc_particle.cell,
74             isoIndex, mc_particle.energy_group);
75         if (currentCrossSection < 0)
76         {
77             selectedIso = isoIndex;
78             selectedUniqueNumber = uniqueNumber;
79             selectedReact = reactIndex;
80             break;
81         }
82     }
83 }

```

Top-down view Bottom-up view Flat view

Scope

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

# Analyzing Quicksilver PC Samples

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

Using the default top-down view of the profile

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



Try all the tools: measurement through post-processing

# Hands-on Tutorial Examples on Aurora - 1

## Log onto an Aurora compute node

```
% qsub -I -x -l select=2,walltime=1:00:00,place=scatter \  
    -l filesystems=flare -A ATPESC2025 -q ATPESC
```

## Copy the hands-on materials and set up your environment

```
% cp -r \  
    /flare/ATPESC2025/EXAMPLES/track6-tools/hpctoolkit/HPCTOOLKIT-HANDS-ON \  
    /flare/ATPESC2025/usr/$LOGNAME/HPCTOOLKIT-HANDS-ON  
% cd /flare/ATPESC2025/usr/$LOGNAME/HPCTOOLKIT-HANDS-ON; source setup.sh
```

## Go to an example directory, e.g. MINITEST/minitest.sycl

1. make run # use hpcrun to measure, hpsctruct for binary analysis, hpcprof to integrate
2. make view # use hpcviewer to examine the resulting database

# Hands-on Tutorial Examples on Aurora - 2

## Other examples

qmcpack - a quantum monte-carlo materials simulation code from the exascale computing project. A big, pre-built code that offloads to the GPU using OpenMP

minitest.omp - a simple MPI + OpenMP offloading code

minitest.sycl - a simple MPI + SYCL offloading code

minitest.sycl.gtpin - use Intel's GTPin binary instrumentation tool to collect dynamic instruction counts for GPU instructions and map them back to source code

minitest.sycl.pc - use hardware support to sample GPU instructions during execution. Map samples and stall reasons back to the source code.

# Hands-on Tutorial Example on Polaris: Quicksilver

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

# Viewing Performance Data

## Options

1. Copy a performance database directory to your laptop and open it locally
  - tar up a “database” directory and copy it to your laptop
  - untar on your laptop and open the directory with hpcviewer
2. Access a performance database on a remote system from an instance of hpcviewer running natively on your laptop

Note: accessing performance data a remote system presumes that hpcserver has already been installed on the remote system

— hpcserver has been installed on Aurora



# Downloading, Installing, and Using HPCToolkit's hpcviewer Graphical User Interface on Your Laptop

# hpcviewer Graphical User Interface on Your Laptop

- Prepare to explore performance data on your laptop
- Download and install hpcviewer
  - See <https://hpctoolkit.org/download.html>
    - Select the right one for your laptop: MacOS (Apple Silicon, Intel), Windows, Linux
    - If you don't have Java JDK 17 or 21, you can install one easily from Adoptium (<https://adoptium.net>)
      - it's a one-click install. install Java JDK before running hpcviewer
- User manual for hpcviewer: <https://hpctoolkit.gitlab.io/hpcviewer>

# Viewing Performance Data

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

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

- hpcserver has been installed on Aurora
- you can download and install hpcserver on your local cluster as well (ask in Slack for directions)

# Configuring Hpcviewer Remote Access

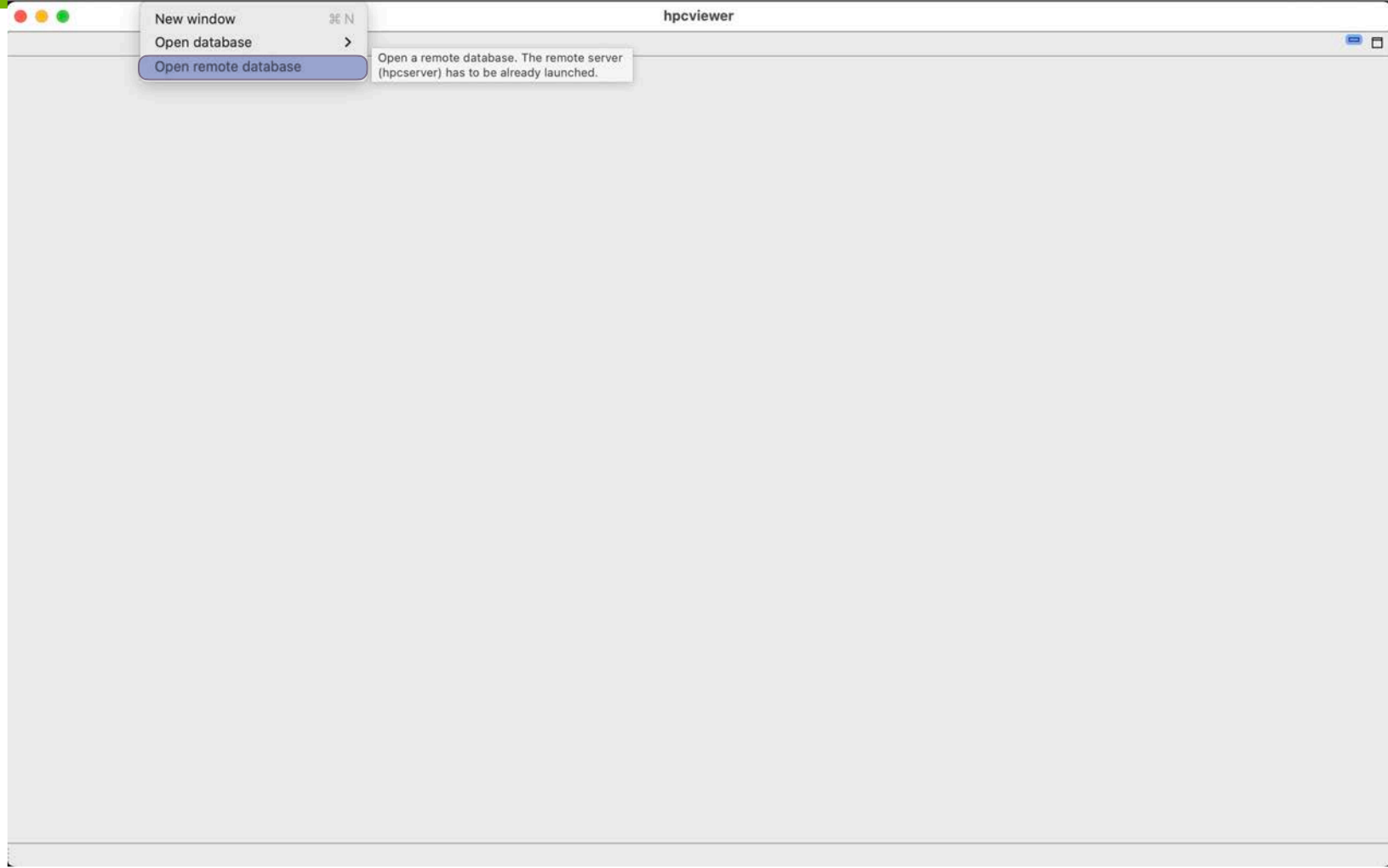
- Run hpcviewer
- From the file menu, select “Open remote database”
- Fill in the hostname/IP address: [aurora.alcf.anl.gov](https://aurora.alcf.anl.gov)
- Fill in your username on Aurora
- Fill in the remote installation directory for hpcviewer’s server:

[/soft/perftools/hpctoolkit/hpcserver](#)

- Select the authentication method: “Use password”
- Click “OK”
- Authenticate using your token as you normally do
- Navigate to a database with the file chooser to

[/flare/ATPESC2025/EXAMPLES/track6-tools/hpctoolkit/data](#)

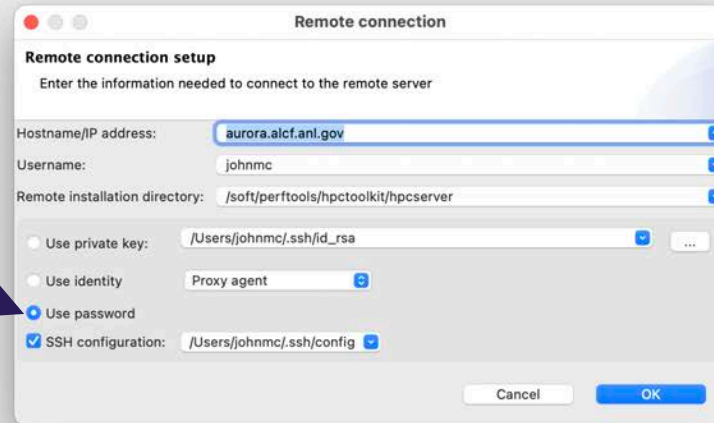
# Opening a Remote Database





# Configuring for remote access to Aurora using hpcserver

Check the option to use a password so you can use MobilePass



The screenshot shows a macOS window titled "hpcviewer" with a "Remote connection" dialog box open. The dialog box has a title bar with standard macOS window controls. Below the title bar, it says "Remote connection setup" and "Enter the information needed to connect to the remote server". The fields are as follows: "Hostname/IP address:" with the value "aurora.alcf.anl.gov"; "Username:" with the value "johnmc"; "Remote installation directory:" with the value "/soft/perftools/hpctoolkit/hpcserver"; "Use private key:" with the value "/Users/johnmc/.ssh/id\_rsa"; "Use identity:" with the value "Proxy agent"; "Use password:" which is selected with a radio button; and "SSH configuration:" with the value "/Users/johnmc/.ssh/config". At the bottom right of the dialog box are "Cancel" and "OK" buttons. A blue arrow points from the text "Check the option to use a password so you can use MobilePass" to the "Use password" radio button.

Remote connection setup

Enter the information needed to connect to the remote server

Hostname/IP address:

Username:

Remote installation directory:

☐ Use private key:

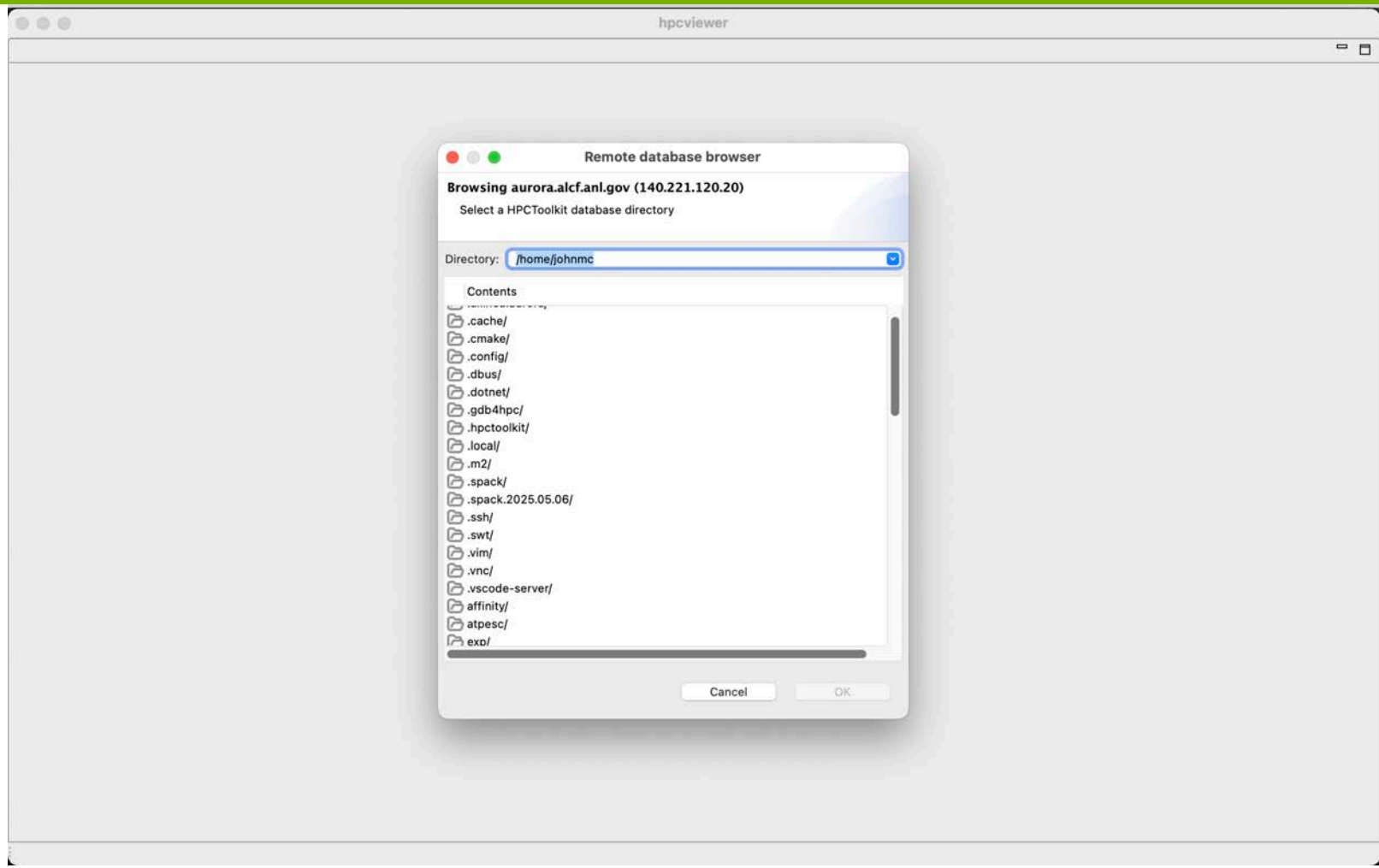
☐ Use identity:

☒ Use password

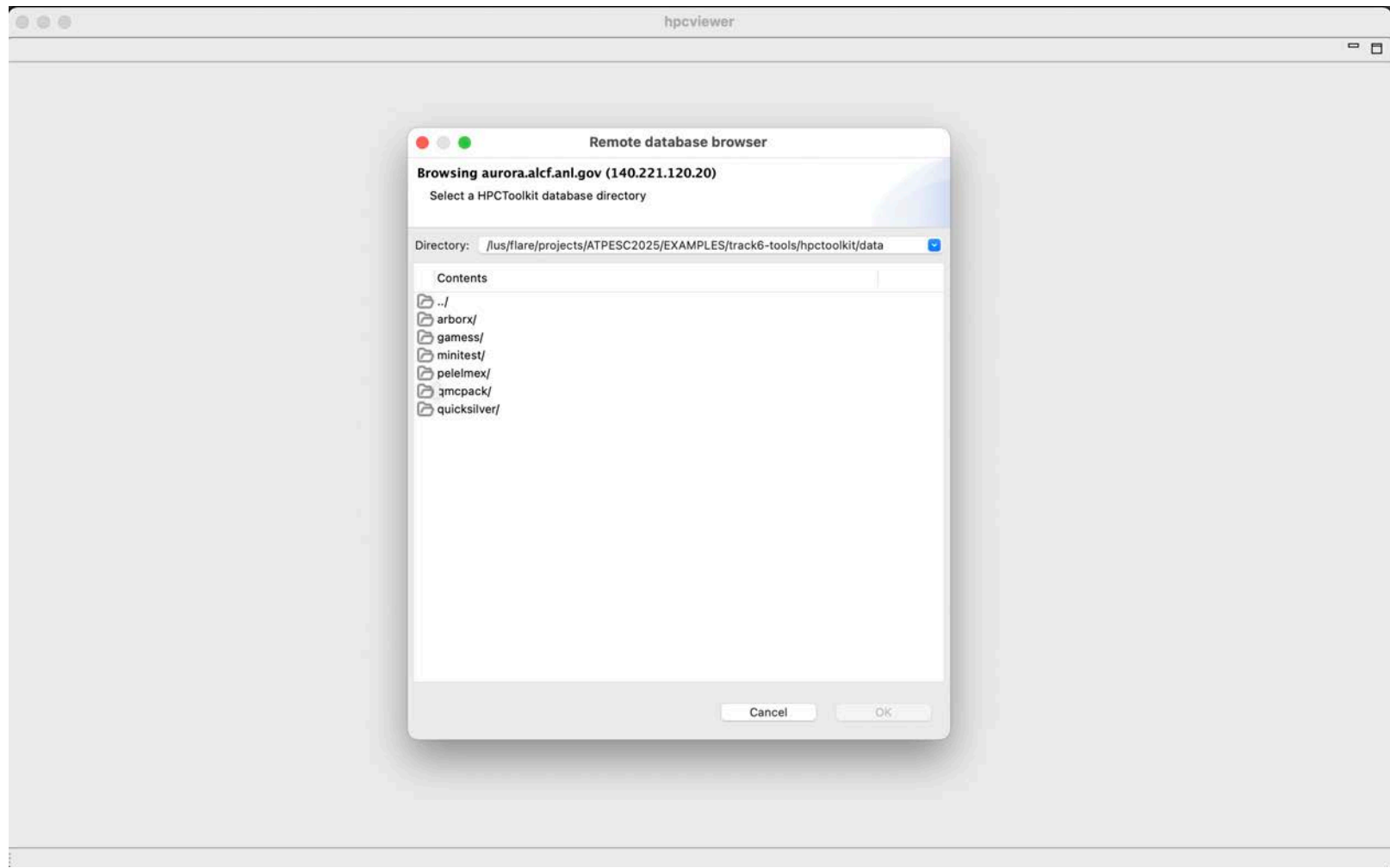
☒ SSH configuration:

Cancel OK

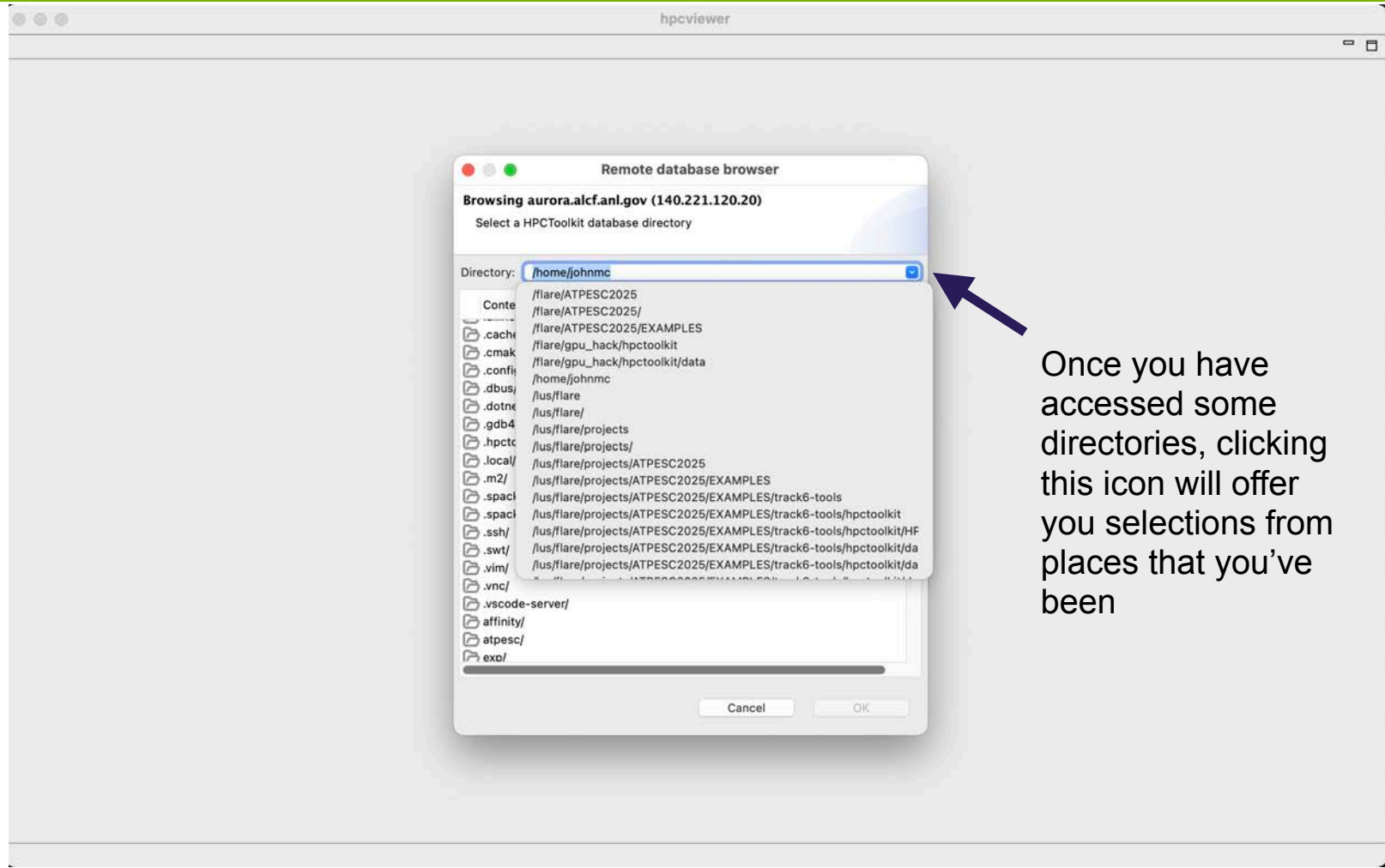
# First View of Aurora: Your Home Directory



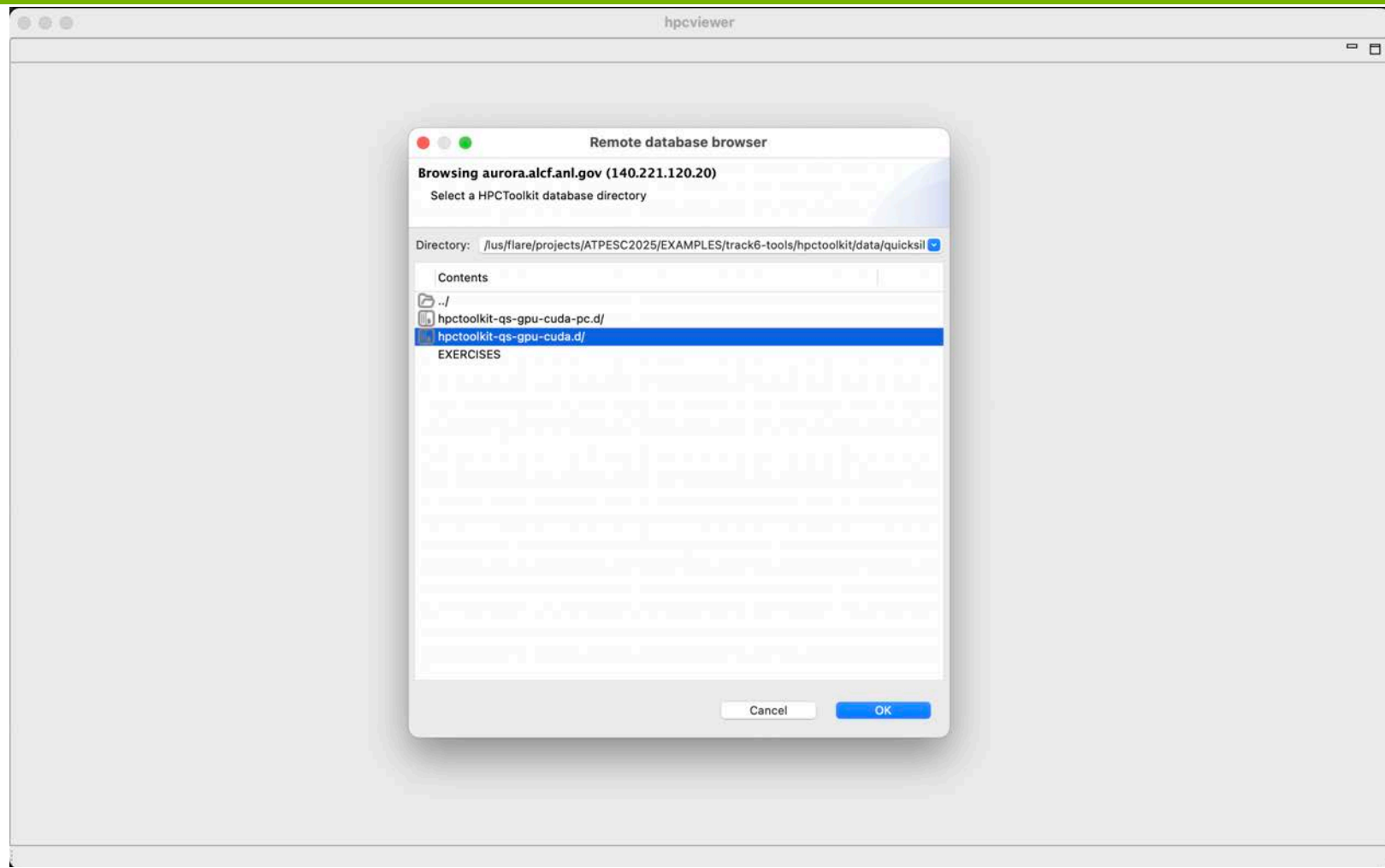
# Navigate to Example Databases



## Navigate to Example Databases

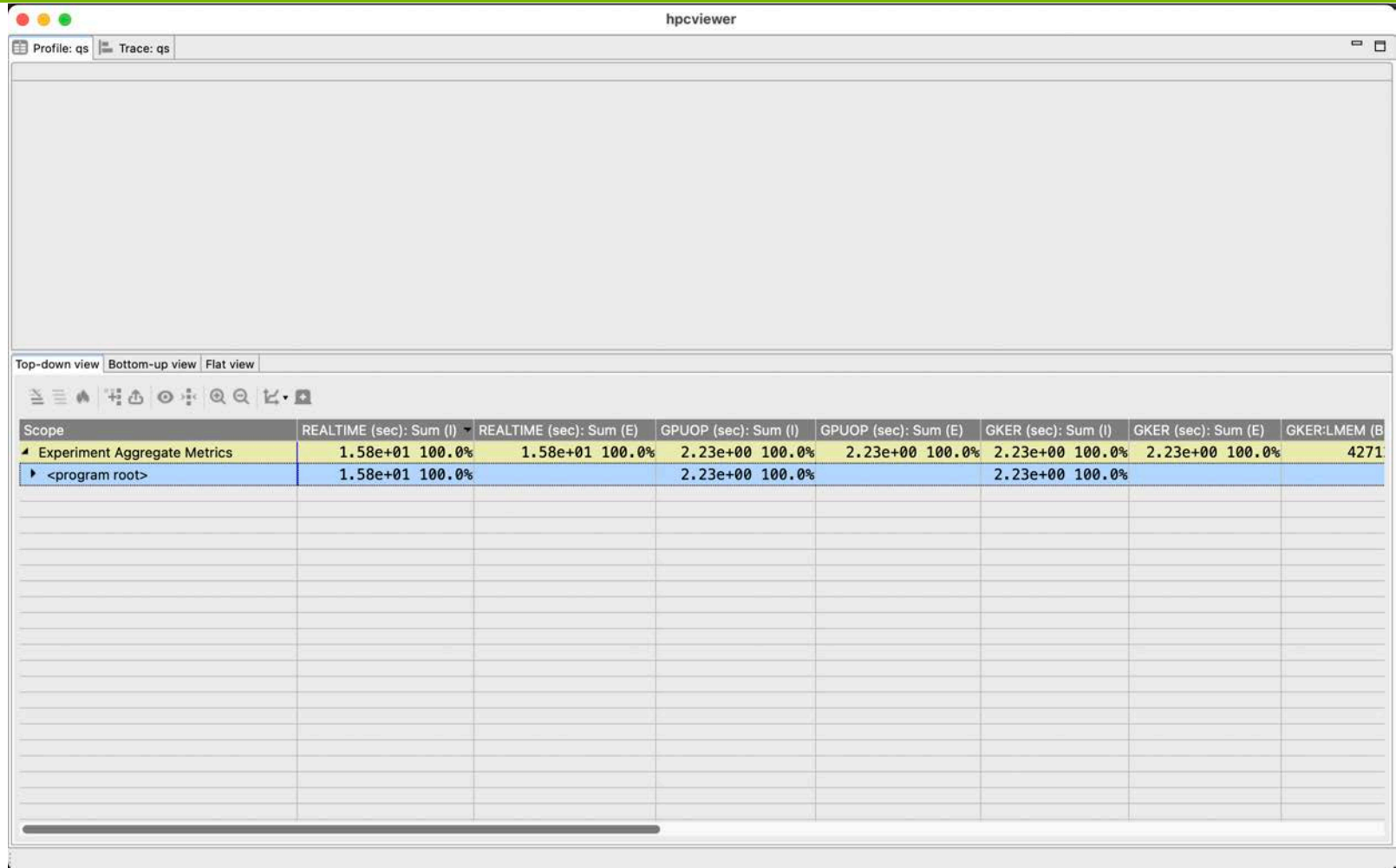


# Select a Quicksilver Database with Traces





## After Selecting hpctoolkit-qs-gpu-cuda.d



# Some hpcviewer Tips

# Information for Using Hpcviewer

- Filtering GPU traces
  - Can use the filter menu to select what execution traces you want to see
    - cpu only, gpu, a mix
    - type a string or a regular expression in the chooser select or unselect the new set
    - only traces that exceed a minimum number of samples
- Filtering GPU calling context tree nodes to hide clutter
  - hide individual CCT nodes: e.g. lines that have no source code mapping library@0x0f450
  - hide subtrees: MPI implementation, implementation of CUDA primitives
- When inspecting GPU activity, be aware that hpcviewer has two modes
  - expose GPU traces or not
    - means: when displaying GPU trace lines, don't just show GPU activity if the time in the middle of a pixel is in a GPU operation. instead, show the first (if any) GPU operation between the time in the middle of the pixel and the middle of the next pixel
    - why? GPU activity is so short, it may be hard to find if we don't "expose" where it is
    - downside: makes the GPU appear more active than it is
      - you can correct hpcviewer's trace-pane statistics by turning off the "Expose GPU traces" mode
  - mode can be selected from <File>:<Preferences>:<Traces>

# 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

# Troubleshooting Tips



# On Linux, hpcviewer crashes at startup!

State saved by different versions of hpcviewer is unfortunately incompatible on Linux. Namely, state saved by hpcviewer/2025.1 and hpcviewer/2025.2 is incompatible.

Typically, removing hpcviewer's saved state, as shown below

```
rm -rf $HOME/.hpctoolkit/hpcviewer
```

will fix the problem and allow you to launch the version of hpcviewer you are trying to use.

Note: when running hpcviewer from Aurora, you must be logged in using “ssh -X” and have a value for the environment variable DISPLAY that indicates a valid X11 display.

# Why can't I see Source Code in hpcviewer?

To relate performance measurements in detail to your application source code, your code must be compiled with a “-g” option in addition to your preferred optimization flags. Otherwise, the compiler doesn't record the information that tools need to map performance to anything finer grain than procedures

- For instance, if you are building with cmake, you will want to build **RelWithDebInfo** rather than **Release** for detailed correlation with source code

# I got the following WARNING from hpcprof

WARNING: Trace for a thread is unexpectedly extremely unordered, falling back to an in-memory sort.

This may indicate an issue during measurement, and WILL significantly increase memory usage!

Affected thread: NODE(BOTH){1930040613, 0} RANK(SINGLE){3} GPUCONTEXT(SINGLE){0}  
GPUSTREAM(SINGLE){0}

It appears that Level Zero time stamps wrap rather than counting monotonically. This can make your traces look surprising where GPU activity is rendered a few minutes behind the rest of application activity.

You can rerun your application and generally the problem will disappear.

(Next slide compares a disordered trace and a correct trace)

# Disordered vs. Correct Trace for qmcpack

Disordered (some or all GPU timestamps shifted left)



Proper (correct) trace alignment between CPU & GPU





## ARGONNE TRAINING PROGRAM ON EXTREME-SCALE COMPUTING

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

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

The U.S. Government retains for itself and others acting on its behalf a nonexclusive, royalty-free license in this video, with the rights to reproduce, to prepare derivative works, and to display publicly.