HPCToolkit: Performance Analysis of GPU-accelerated Kokkos Applications on NVIDIA GPUs

John Mellor-Crummey
Rice University

April 26, 2024

National Nuclear Security Administration

U.S. DEPARTMENT OF ENERGY | Office of Science

STEP SOFTWARE TOOLS ECOSYSTEM PROJECT

RICE

Current Funding for HPCToolkit

- Government
  - Lawrence Livermore National Laboratory Subcontract B658833
  - 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.

A Hands-on Example for the Tutorial: ArborX

A library written in Kokkos that provides performance portable algorithms for geometric search

```
% git clone https://github.com/hpctoolkit/hpctoolkit-tutorial-examples
% cd hpctoolkit-tutorial-examples/examples/gpu/arborx
% source setup/perlmutter.sh
% make all # downloads, builds, measures, and analyzes two executions
% make view
% make view-pc
```

Note: precomputed databases available on Perlmutter at /global/cfs/cdirs/m3977/data/arborx

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
  - ArborX
  - LAMMPS at Exascale
- Coming attractions
- Troubleshooting

Rice University’s HPCToolkit Performance Tools

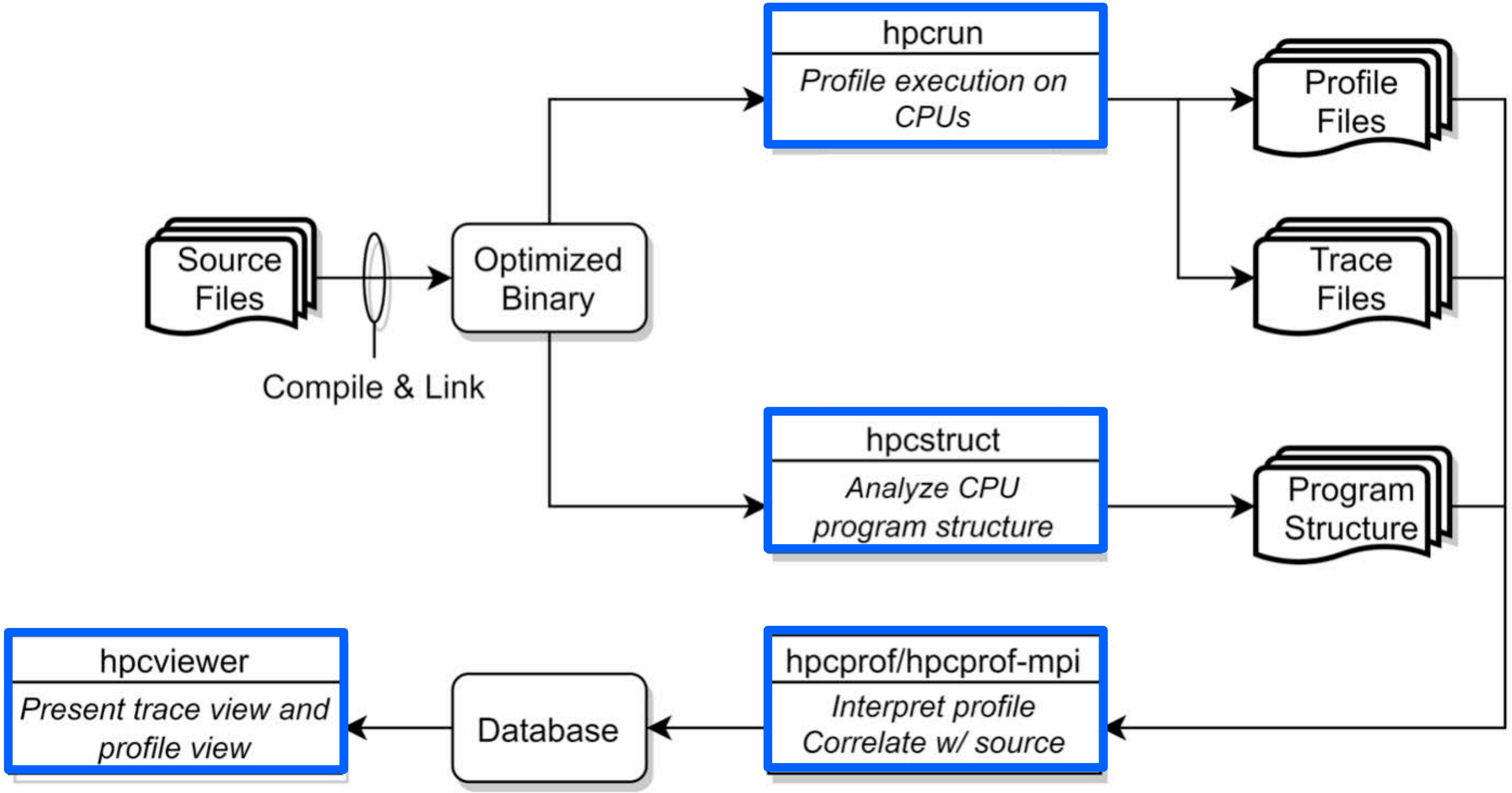
Measure and analyze performance of CPU and GPU-accelerated applications

- 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

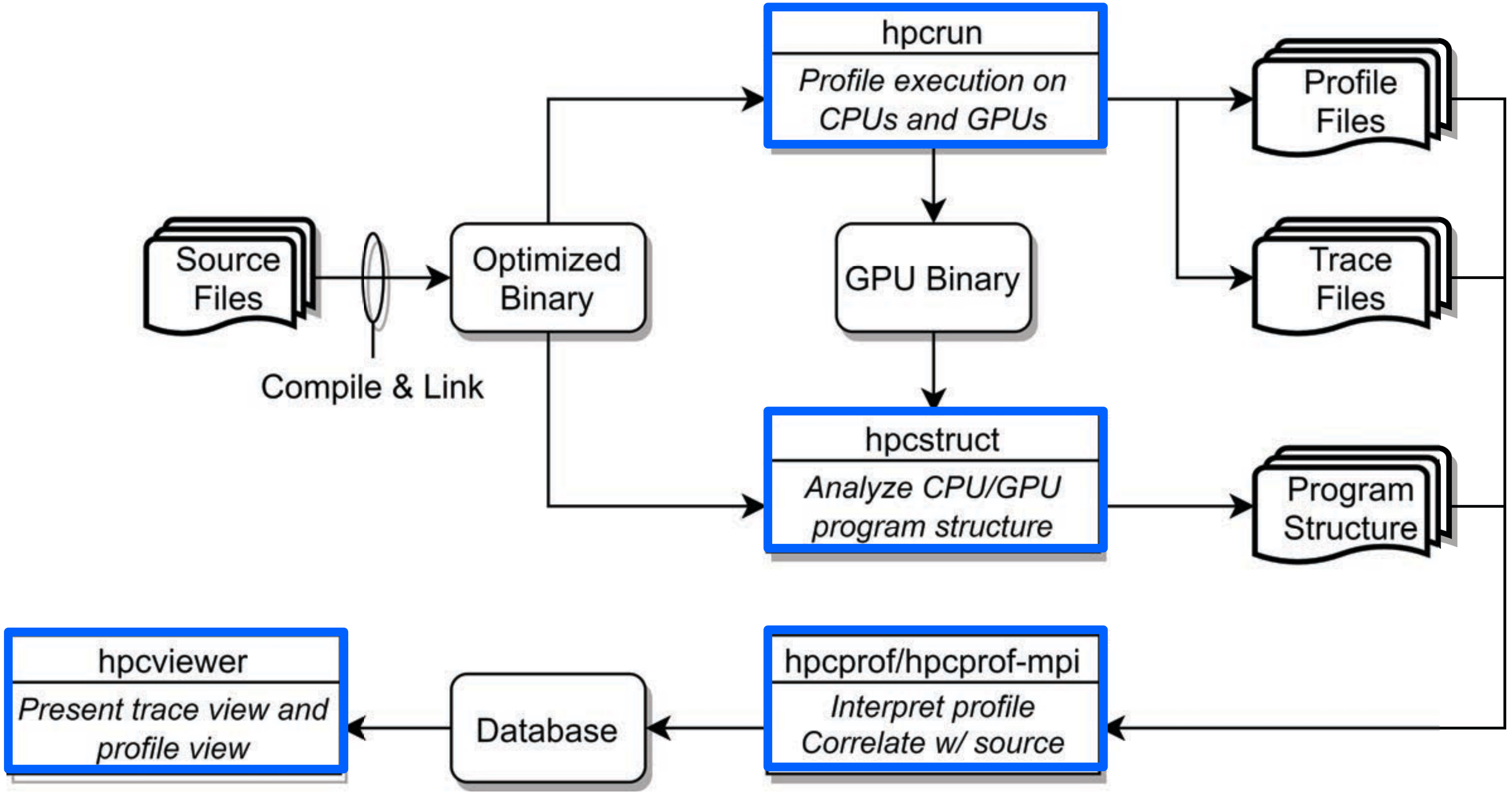
How does HPCToolkit Differ from NVIDIA's Tools?

- 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



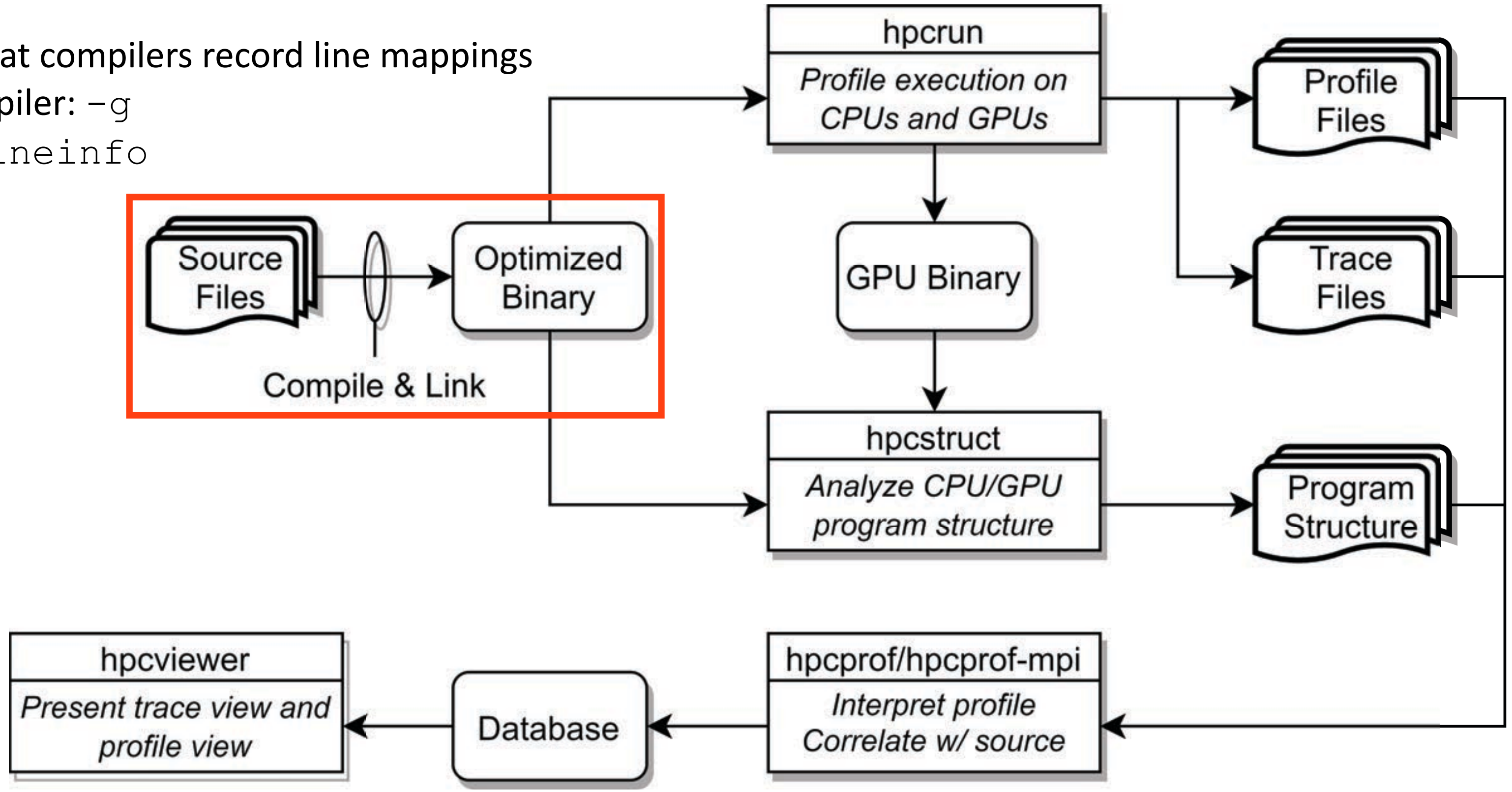
HPCToolkit’s Workflow for GPU-accelerated Applications



HPCToolkit's Workflow for GPU-accelerated Applications

Step 1:

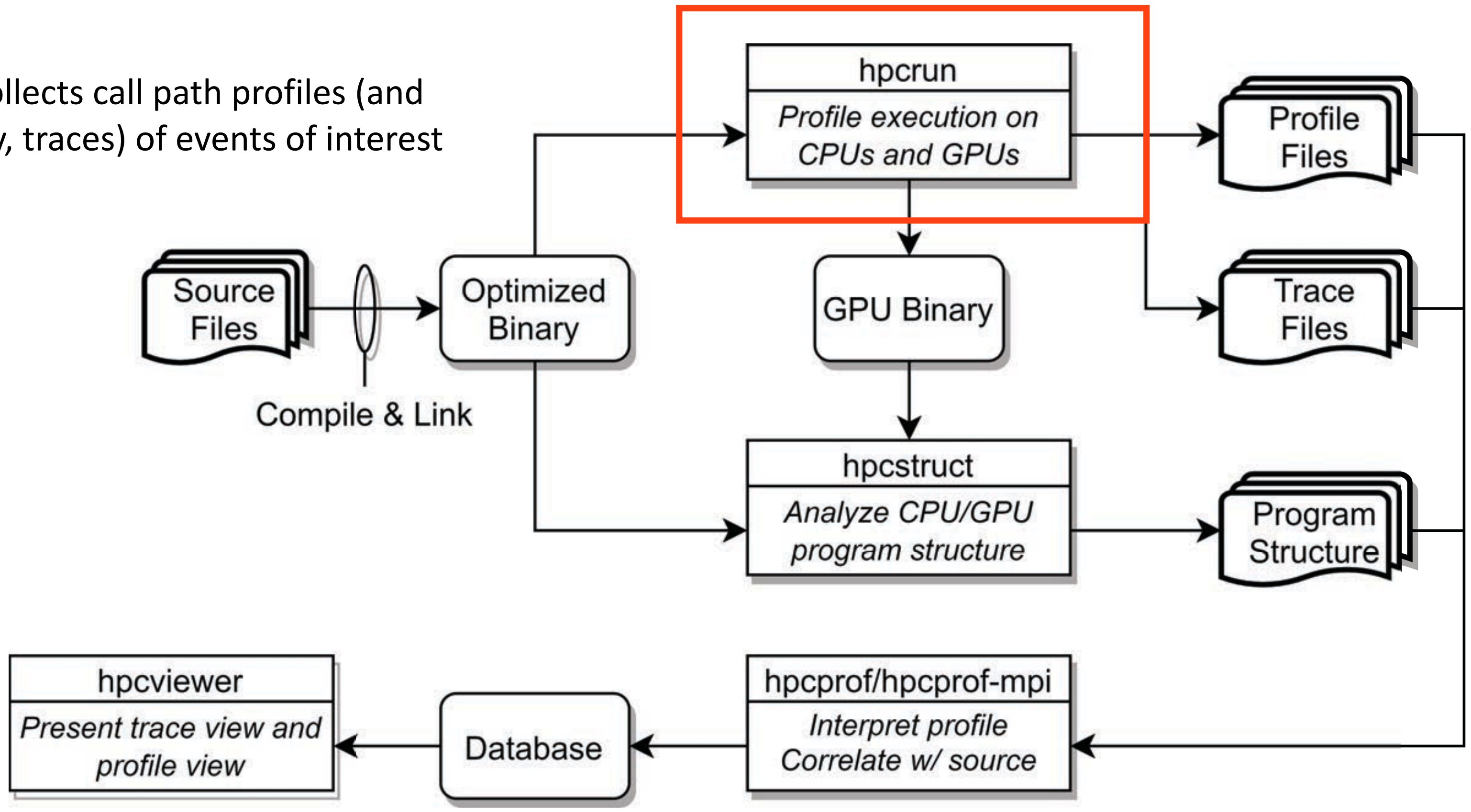
- Ensure that compilers record line mappings
- host compiler: `-g`
- nvcc: `-lineinfo`



HPCToolkit's Workflow for GPU-accelerated Applications

Step 2:

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



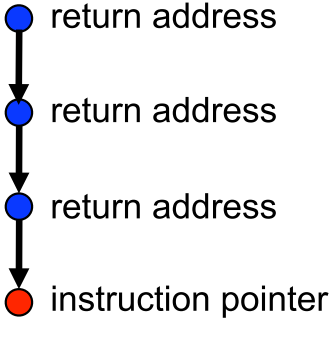
Measurement of CPU and GPU-accelerated Applications

- 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)
- Binary instrumentation of GPU kernels on Intel GPUs for fine-grain measurement

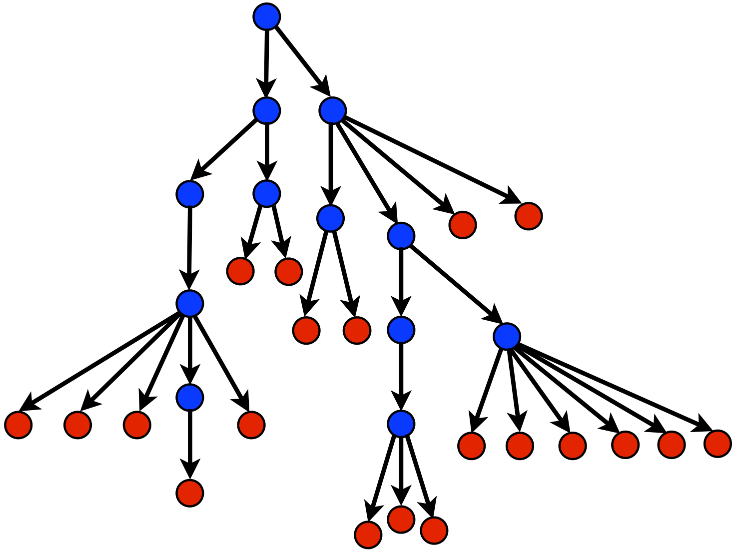
Call Stack Unwinding to Attribute Costs in Context

- 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



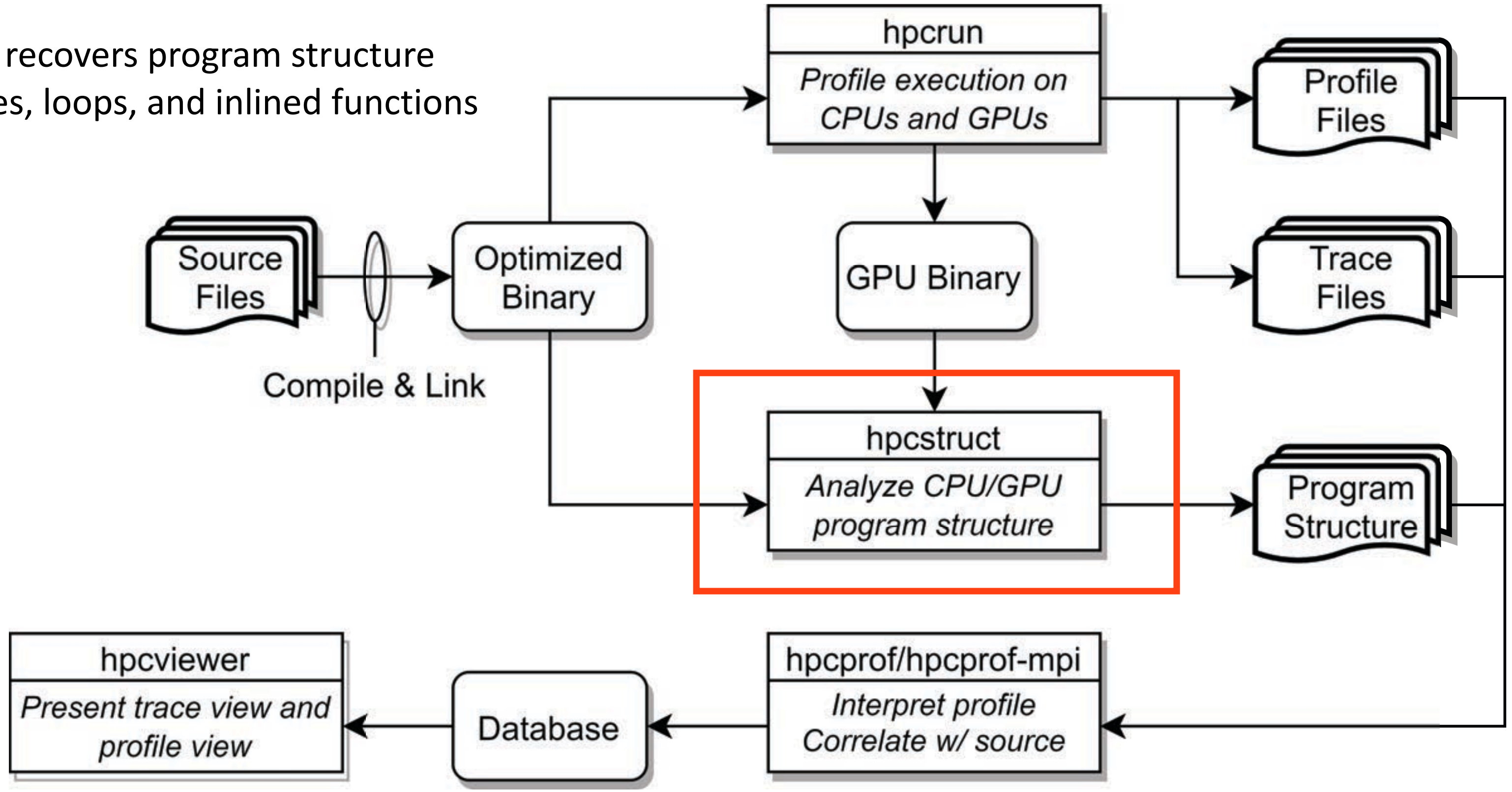
hpcrun: Measure CPU and/or GPU activity

- GPU profiling
  - `hpcrun -e gpu=xxx <app> ….`   *xxx ∈ {nvidia,amd,opencl,level0}*
- GPU instrumentation (Intel GPU only)
  - `hpcrun -e gpu=level0,inst=count,latency <app>`
- 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>`
- Use hpcrun with job launchers
  - `srun -n 1 -G 1 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>
```

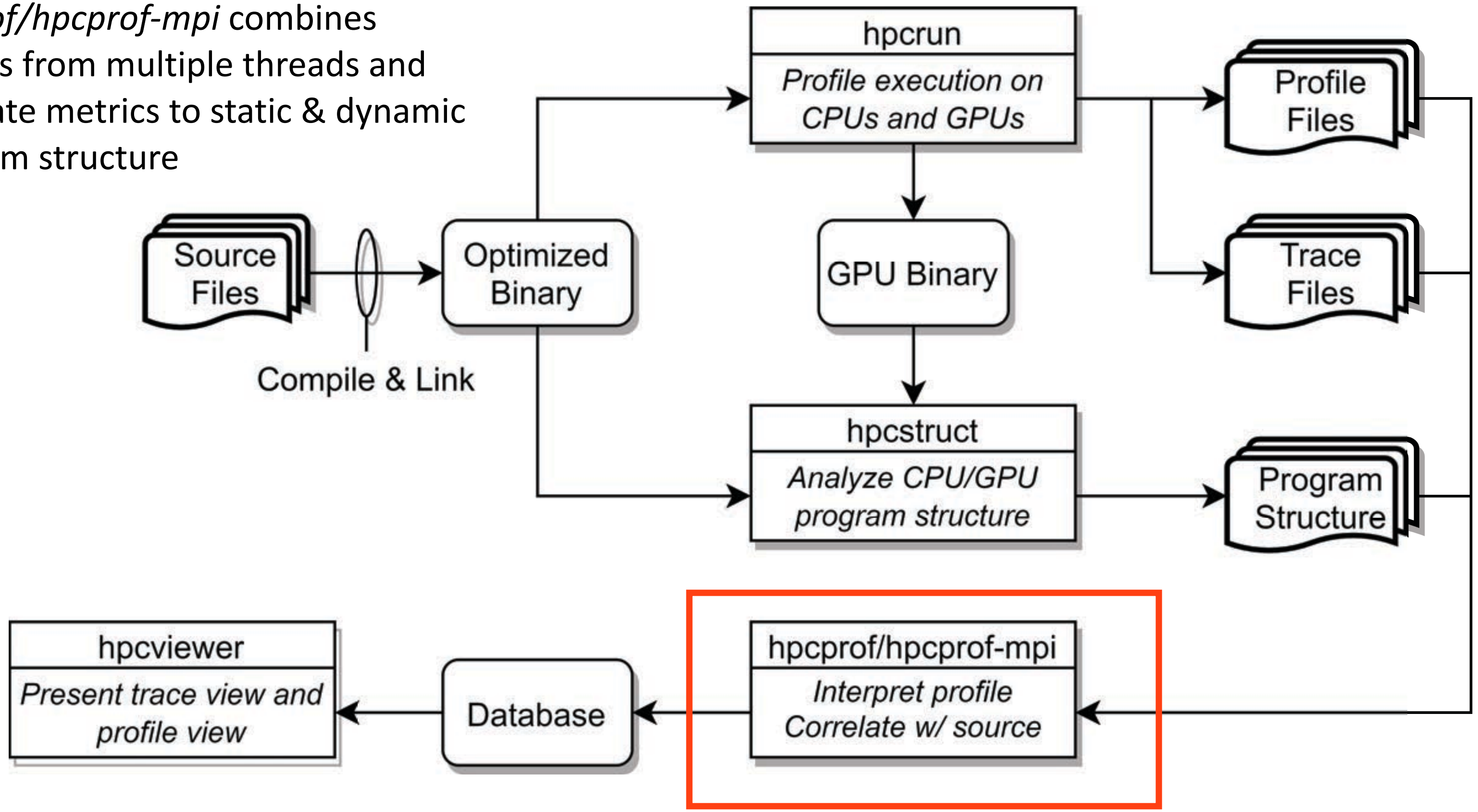
- What it does
  - Recover program structure information
    - Files, functions, inlined templates or functions, loops, source lines
  - In parallel, analyze all CPU and GPU binaries that were measured by HPCToolkit
    - —default: use size(CPU set)/2 threads
    - —analyze large application binaries 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 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

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

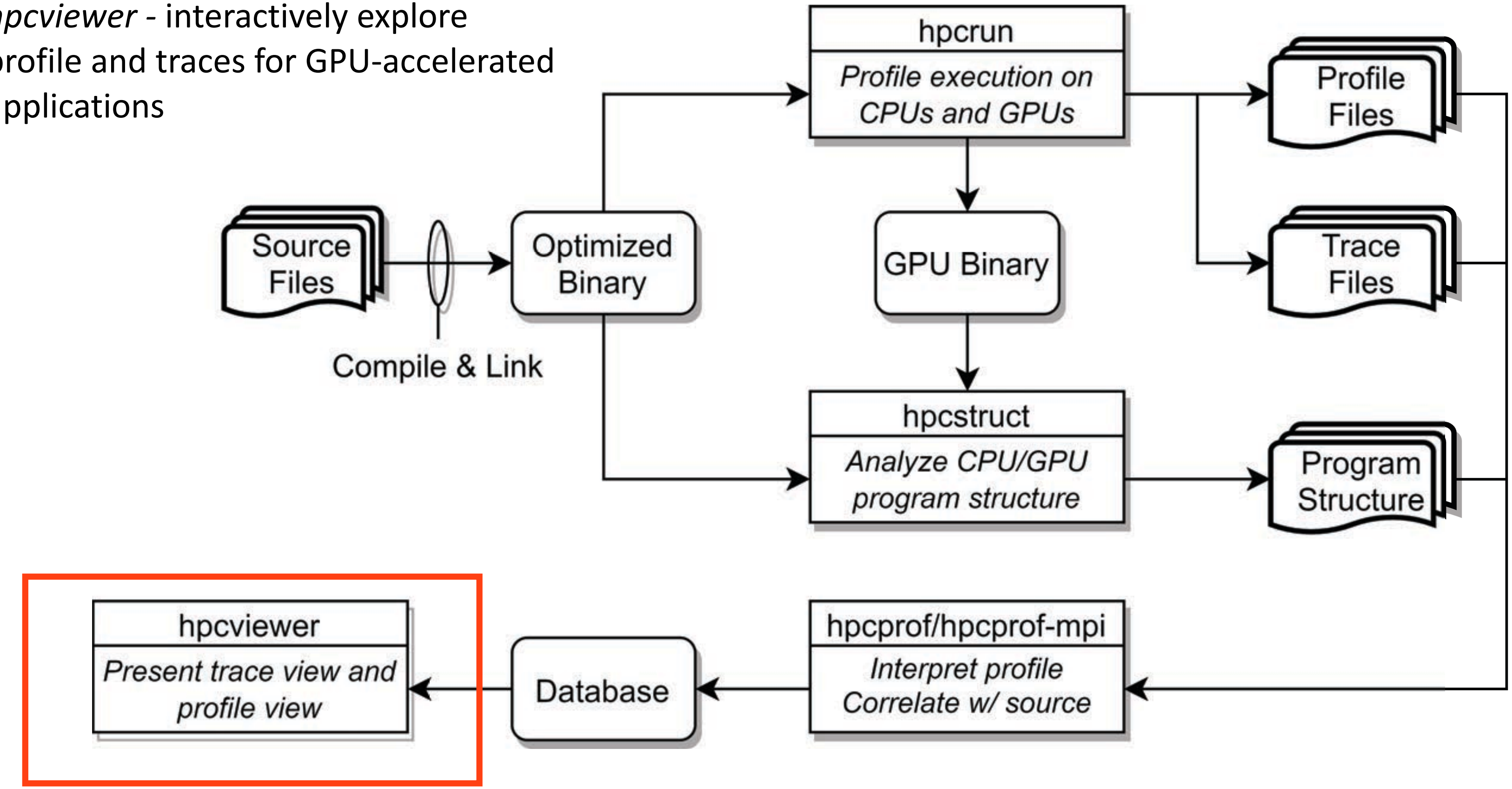
- Analyze data from large executions with distributed-memory parallelism + multithreading

```
srun -N 2 -n 2 -c 126 hpcprof-mpi <measurement-directory>
```

HPCToolkit's Workflow for GPU-accelerated Applications

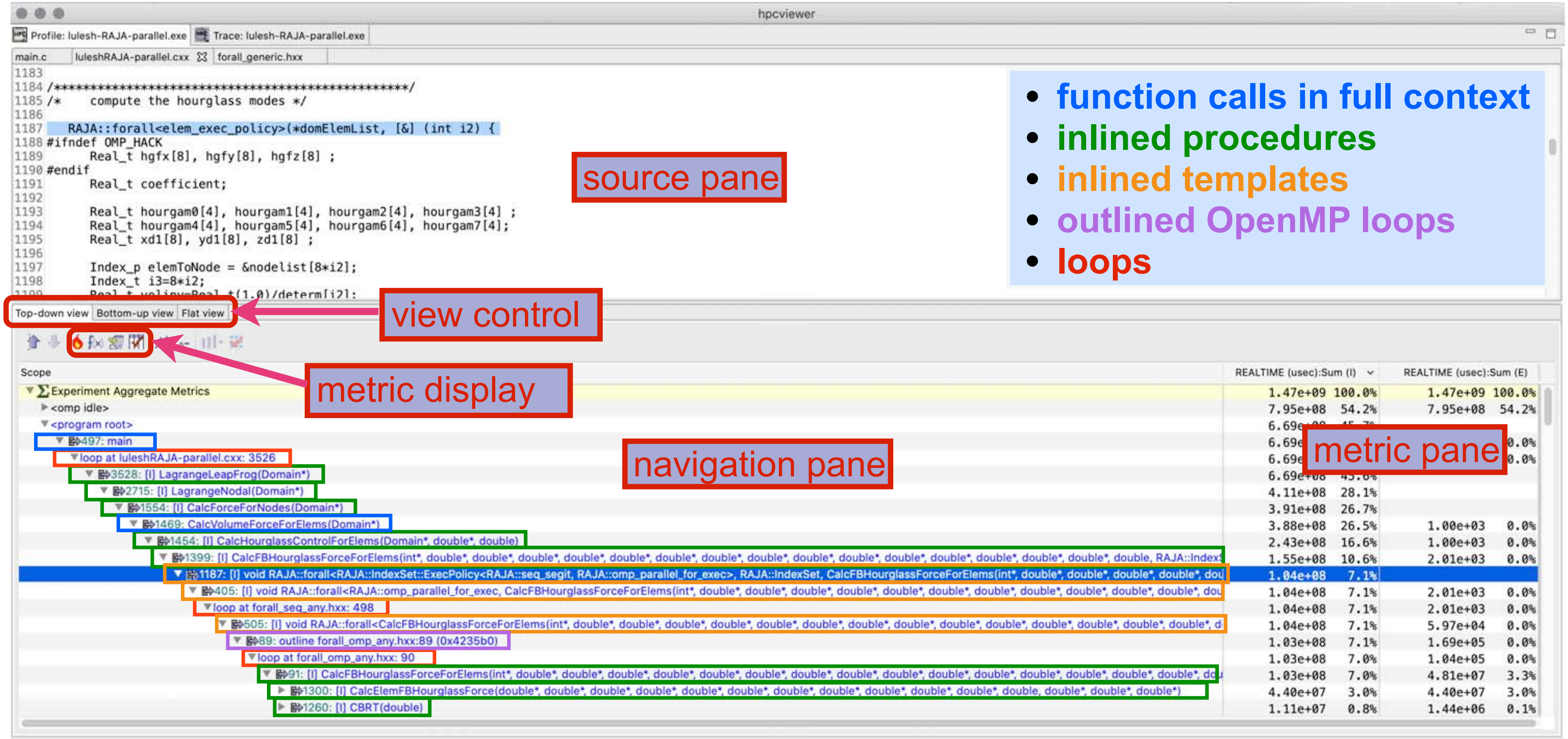
Step 4:

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



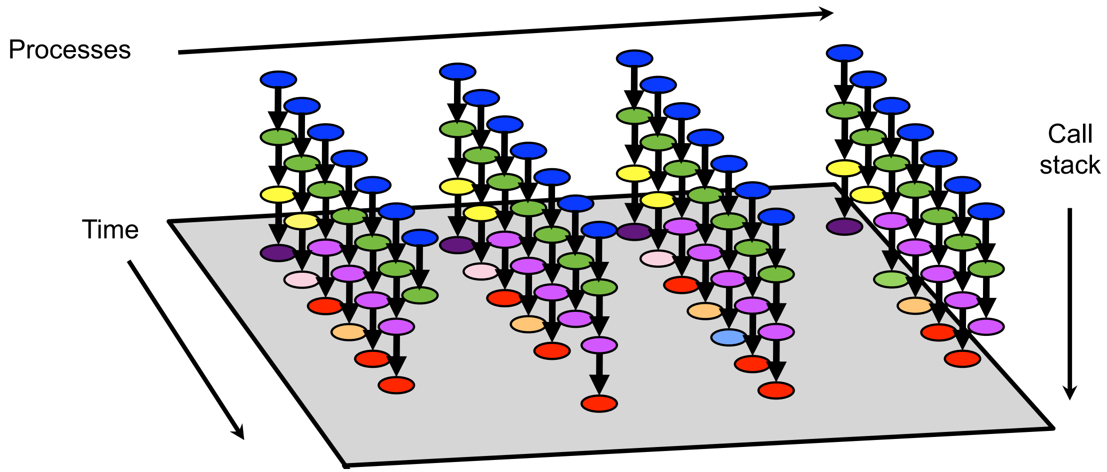
Code-centric Analysis with hpcviewer

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

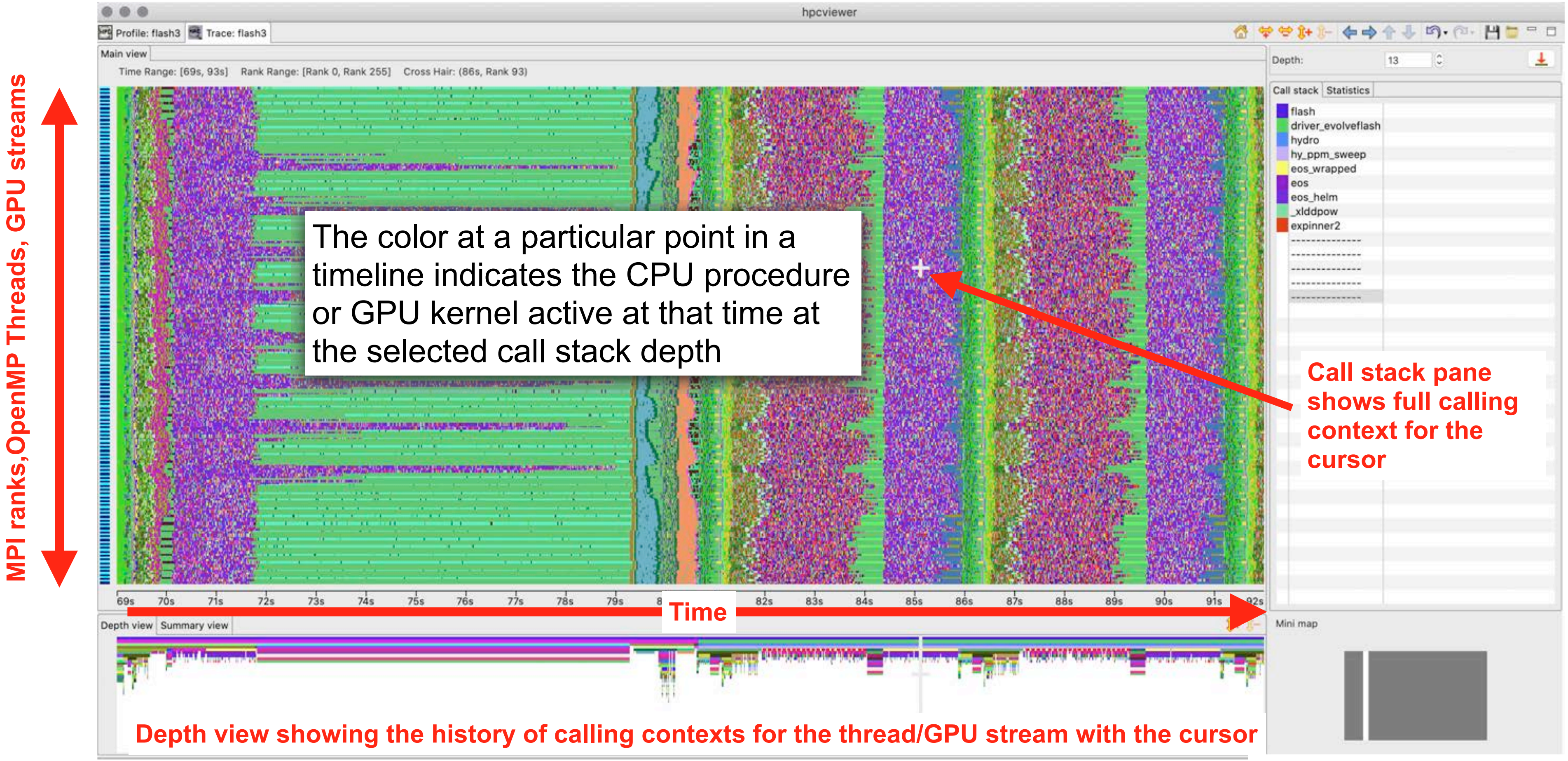


Understanding Temporal Behavior

- Profiling compresses out the temporal dimension
  - Temporal patterns, e.g. serial sections and dynamic load imbalance are invisible in profiles
- What can we do? Trace call path samples
  - N times per second, take a call path sample of each thread
  - Organize the samples for each thread along a time line
  - View how the execution evolves left to right
  - What do we view? assign each procedure a color; view a depth slice of an execution



Time-centric Analysis with hpcviewer



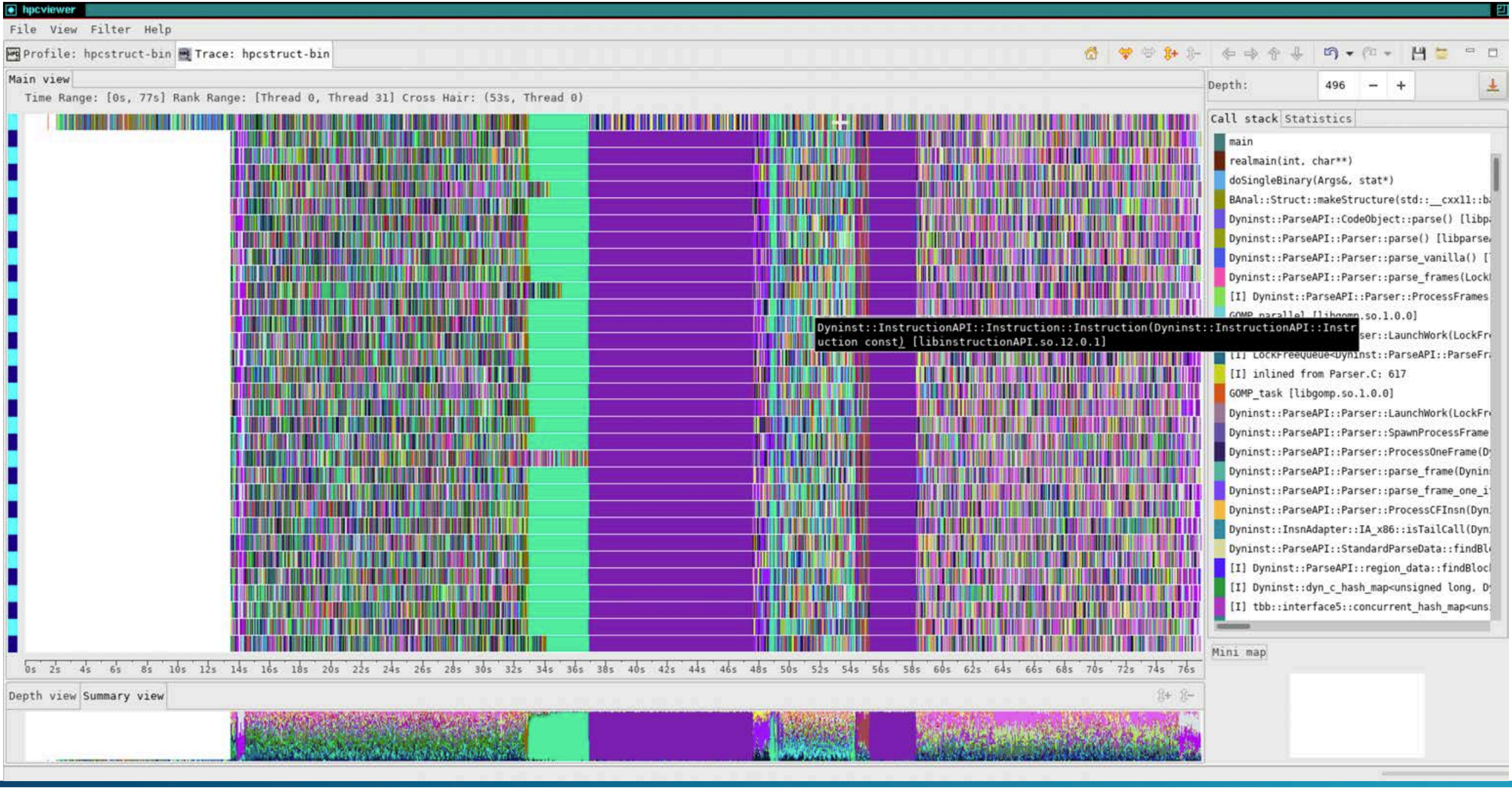
A multi-level call stack based view of execution over time

Minimap indicates part of execution trace shown

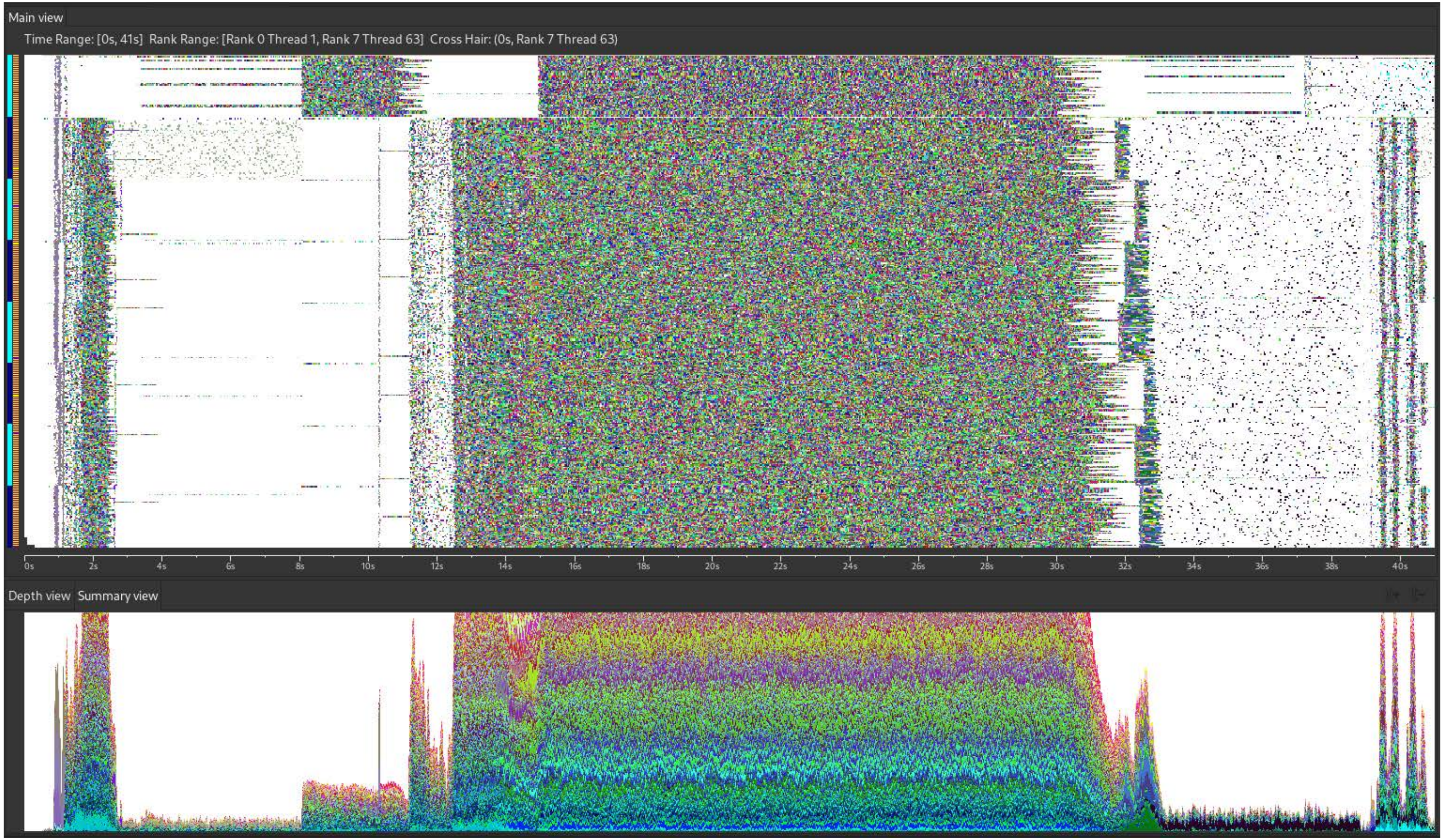
Summary of ECP Developments

- Measurement
  - profile and trace GPU-accelerated applications on AMD, Intel, and NVIDIA GPUs
  - source-level measurement of Python frameworks, e.g. Pytorch
  - record measurement data in sparse formats: benefits GPU-accelerated programs with many metrics
  - implement of OMPT performance tools interface in AMD OpenMP and LLVM
- Binary analysis
  - binary analysis of AMD, Intel, NVIDIA GPU binaries
  - parallel analysis of application binaries to speed recovery of program structure
- Performance analysis and attribution
  - MPI + OpenMP highly parallel analysis of measurement data at exascale
  - sparse representations observed to reduce performance analysis results by > 1000x
  - detailed attribution of PC samples to rich calling contexts within GPU kernels
- Presentation
  - interactive display profiles and terabytes of traces from exascale executions

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



EXASCALE COMPUTING PROJECT

Slide credit: Jonathon Anderson

Case Studies

- GAMESS (OpenMP)
- ArborX (Kokkos)
- LAMMPS (Kokkos) at exascale

Case Study: GAMESS

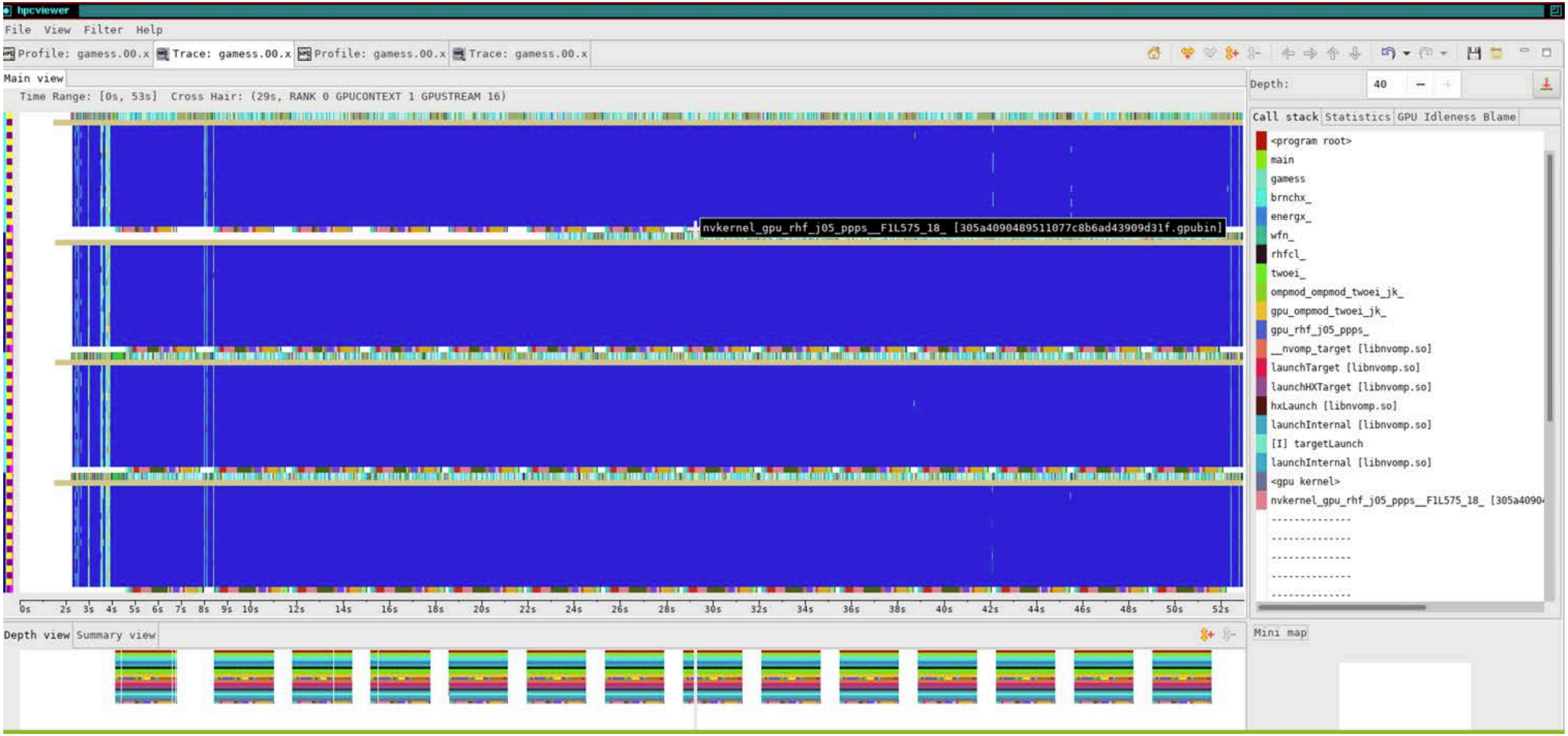
- General Atomic and Molecular Electronic Structure System (GAMESS)
  - general *ab initio* quantum chemistry package
- Calculates the energies, structures, and properties of a wide range of chemical systems

- Experiments
  - GPU-accelerated nodes at a prior Perlmutter hackathon
    - Single node with 4 GPUs
    - Five nodes with 20 GPUs

Perlmutter node at a glance

AMD Milan CPU
4 NVIDIA A100 GPUs
256 GB memory

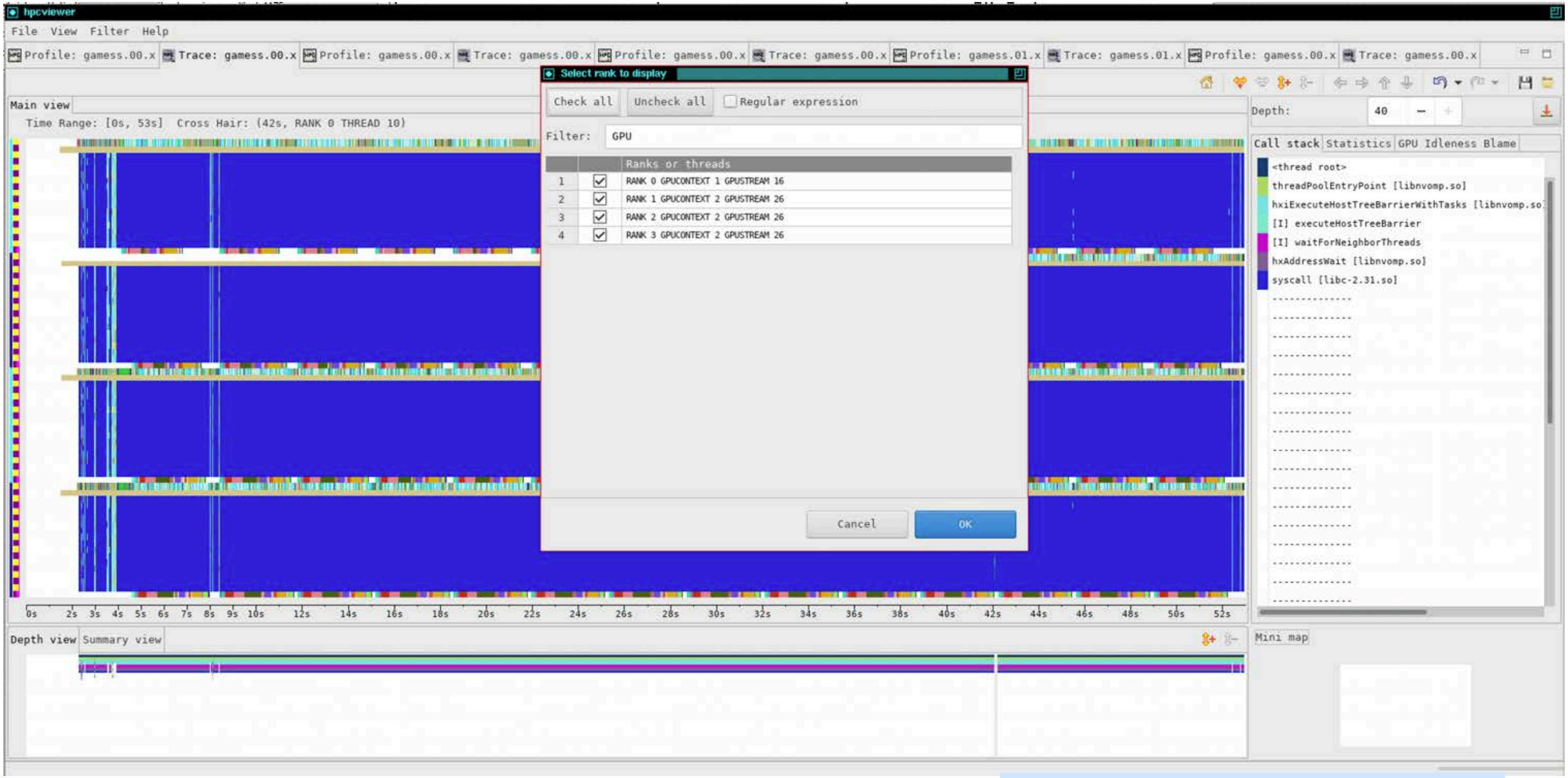
Time-centric Analysis: GAMESS 4 ranks, 4 GPUs on Perlmutter



GAMESS original

All CPU threads and GPU streams

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



GAMESS original

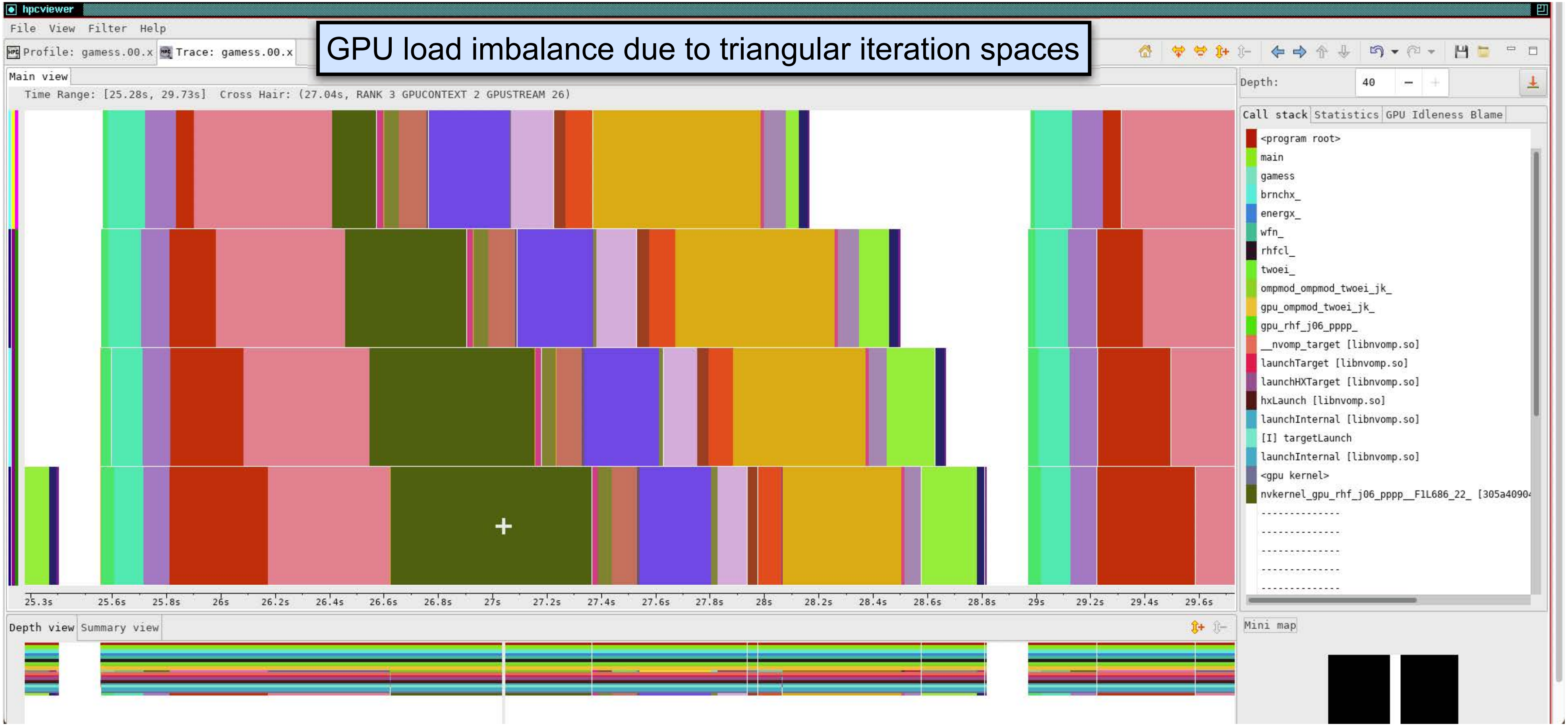
All CPU threads and GPU streams

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

GAMESS original

All GPU streams, whole execution

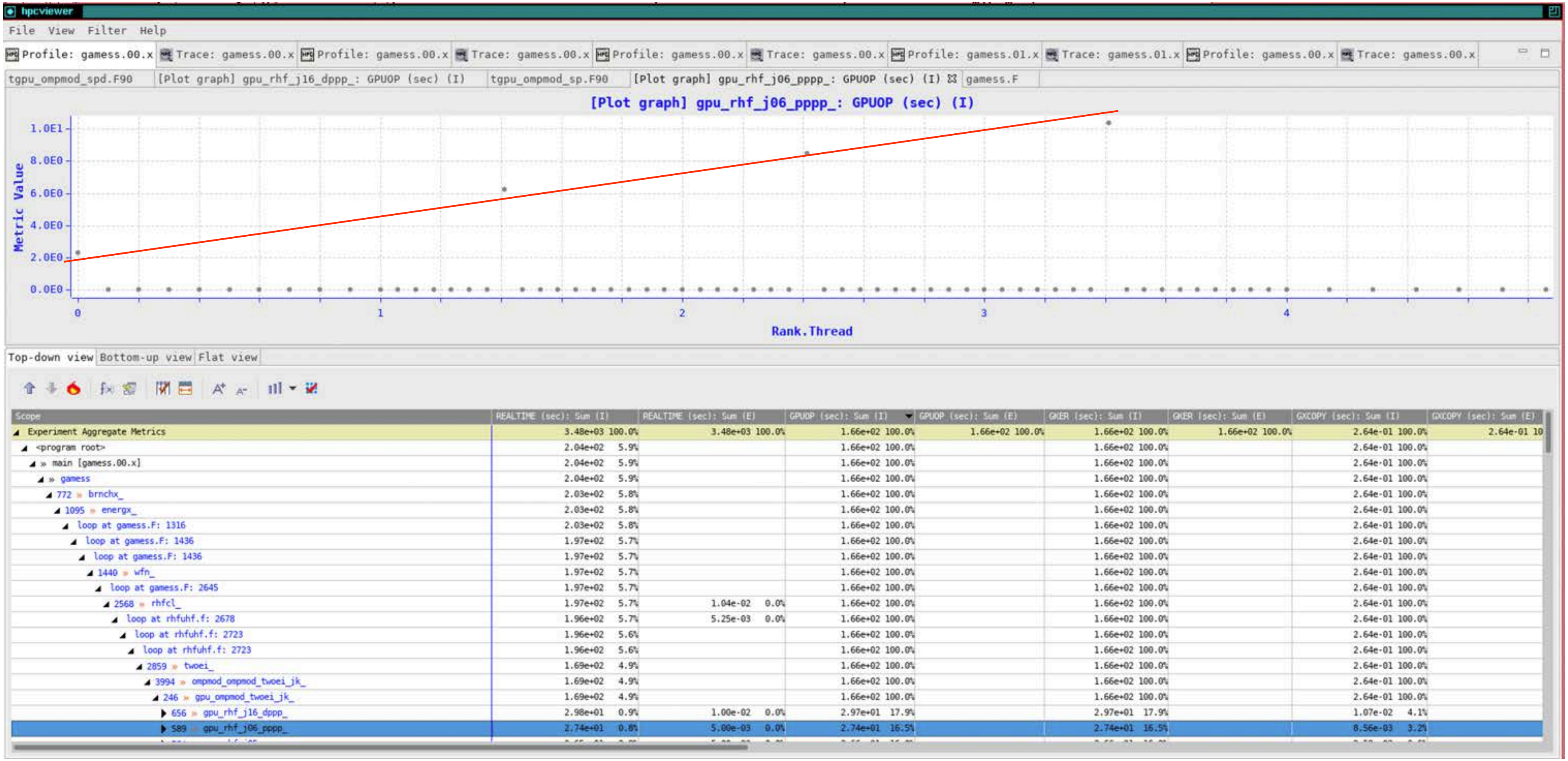
Time-centric Analysis: GAMESS 4 ranks, 4 GPUs on Perlmutter



GAMESS original

GPU streams: 1 iteration

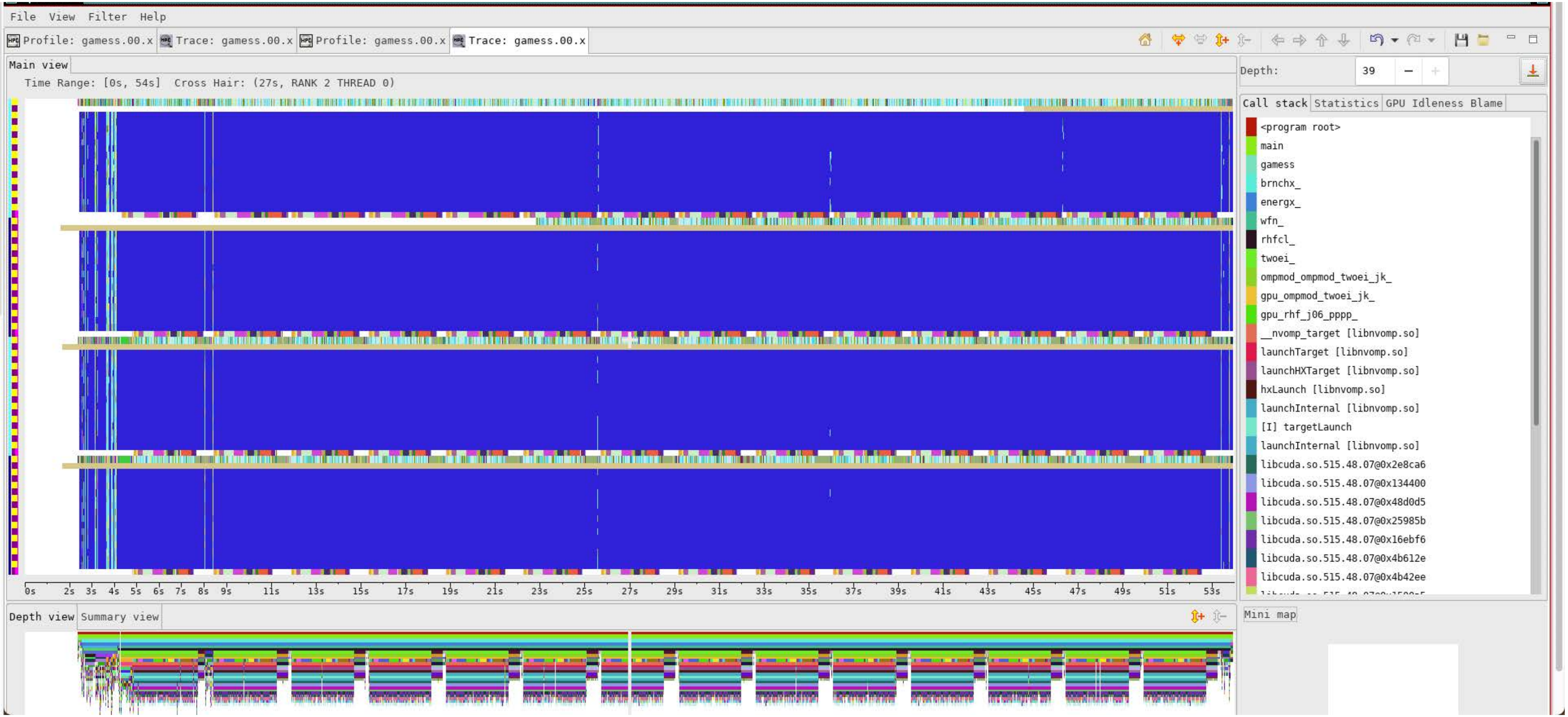
Time-centric Analysis: GAMESS 4 ranks, 4 GPUs on Perlmutter



Scope	REALTIME (sec): Sum (I)	REALTIME (sec): Sum (E)	GPUOP (sec): Sum (I)	GPUOP (sec): Sum (E)	GKER (sec): Sum (I)	GKER (sec): Sum (E)	GXCOPY (sec): Sum (I)	GXCOPY (sec): Sum (E)
Experiment Aggregate Metrics	3.48e+03 100.0%	3.48e+03 100.0%	1.66e+02 100.0%	1.66e+02 100.0%	1.66e+02 100.0%	1.66e+02 100.0%	2.64e-01 100.0%	2.64e-01 10
<program root>	2.04e+02 5.9%		1.66e+02 100.0%		1.66e+02 100.0%		2.64e-01 100.0%	
» main [gamess.00.x]	2.04e+02 5.9%		1.66e+02 100.0%		1.66e+02 100.0%		2.64e-01 100.0%	
» gamess	2.04e+02 5.9%		1.66e+02 100.0%		1.66e+02 100.0%		2.64e-01 100.0%	
772 » brnchx_	2.03e+02 5.8%		1.66e+02 100.0%		1.66e+02 100.0%		2.64e-01 100.0%	
1095 » energx_	2.03e+02 5.8%		1.66e+02 100.0%		1.66e+02 100.0%		2.64e-01 100.0%	
loop at gamess.F: 1316	2.03e+02 5.8%		1.66e+02 100.0%		1.66e+02 100.0%		2.64e-01 100.0%	
loop at gamess.F: 1436	1.97e+02 5.7%		1.66e+02 100.0%		1.66e+02 100.0%		2.64e-01 100.0%	
loop at gamess.F: 1436	1.97e+02 5.7%		1.66e+02 100.0%		1.66e+02 100.0%		2.64e-01 100.0%	
1440 » wfn_	1.97e+02 5.7%		1.66e+02 100.0%		1.66e+02 100.0%		2.64e-01 100.0%	
loop at gamess.F: 2645	1.97e+02 5.7%		1.66e+02 100.0%		1.66e+02 100.0%		2.64e-01 100.0%	
2568 » rhfcl_	1.97e+02 5.7%	1.04e-02 0.0%	1.66e+02 100.0%		1.66e+02 100.0%		2.64e-01 100.0%	
loop at rhfuhf.f: 2678	1.96e+02 5.7%	5.25e-03 0.0%	1.66e+02 100.0%		1.66e+02 100.0%		2.64e-01 100.0%	
loop at rhfuhf.f: 2723	1.96e+02 5.6%		1.66e+02 100.0%		1.66e+02 100.0%		2.64e-01 100.0%	
loop at rhfuhf.f: 2723	1.96e+02 5.6%		1.66e+02 100.0%		1.66e+02 100.0%		2.64e-01 100.0%	
2859 » twoei_	1.69e+02 4.9%		1.66e+02 100.0%		1.66e+02 100.0%		2.64e-01 100.0%	
3994 » ompmod_ompmod_twoei_jk_	1.69e+02 4.9%		1.66e+02 100.0%		1.66e+02 100.0%		2.64e-01 100.0%	
246 » gpu_ompmod_twoei_jk_	1.69e+02 4.9%		1.66e+02 100.0%		1.66e+02 100.0%		2.64e-01 100.0%	
656 » gpu_rhf_j16_dppp_	2.98e+01 0.9%	1.00e-02 0.0%	2.97e+01 17.9%		2.97e+01 17.9%		1.07e-02 4.1%	
589 » gpu_rhf_j06_pppp_	2.74e+01 0.8%	5.00e-03 0.0%	2.74e+01 16.5%		2.74e+01 16.5%		8.56e-03 3.2%	

GAMESS original

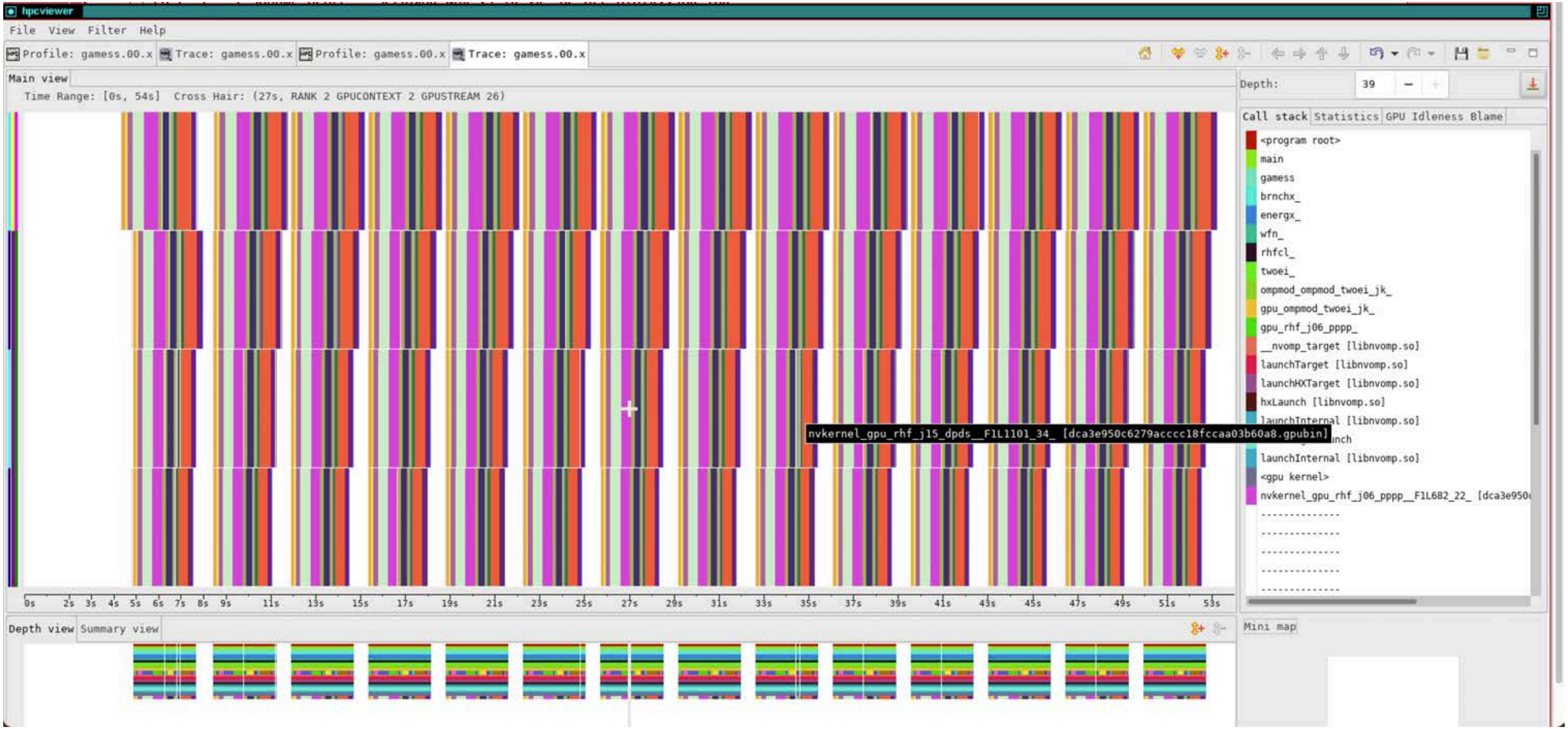
Time-centric Analysis: GAMESS 4 ranks, 4 GPUs on Perlmutter



GAMESS improved

All CPU threads and GPU streams

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



GAMESS improved

All GPU streams, whole execution

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

Improved GPU load balance

GAMESS improved

All GPU streams: 2 iterations

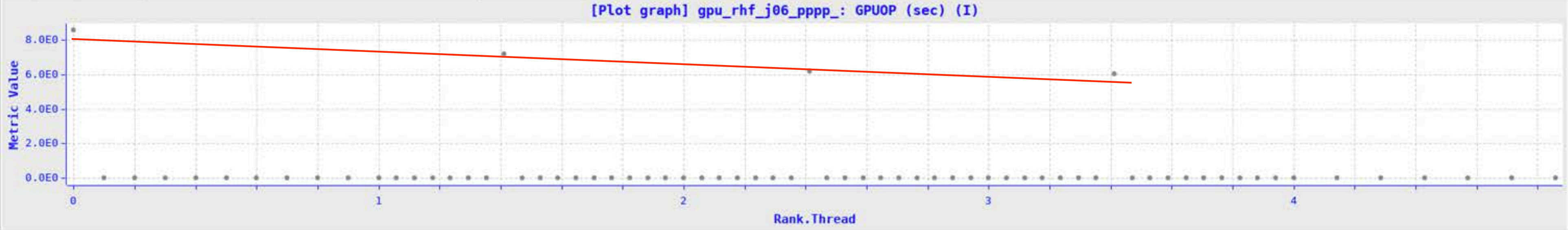
Time-centric Analysis: GAMESS 4 ranks, 4 GPUs on Perlmutter

hpcviewer

File View Filter Help

Profile: gamess.00.x | Trace: gamess.00.x | Profile: gamess.00.x | Trace: gamess.00.x | Profile: gamess.00.x | Trace: gamess.00.x | Profile: gamess.01.x | Trace: gamess.01.x | Profile: gamess.00.x | Trace: gamess.00.x

tgpu_ompmod_sp.F90 | [Plot graph] gpu_rhf_j06_pppp_: GPUOP (sec) (I)

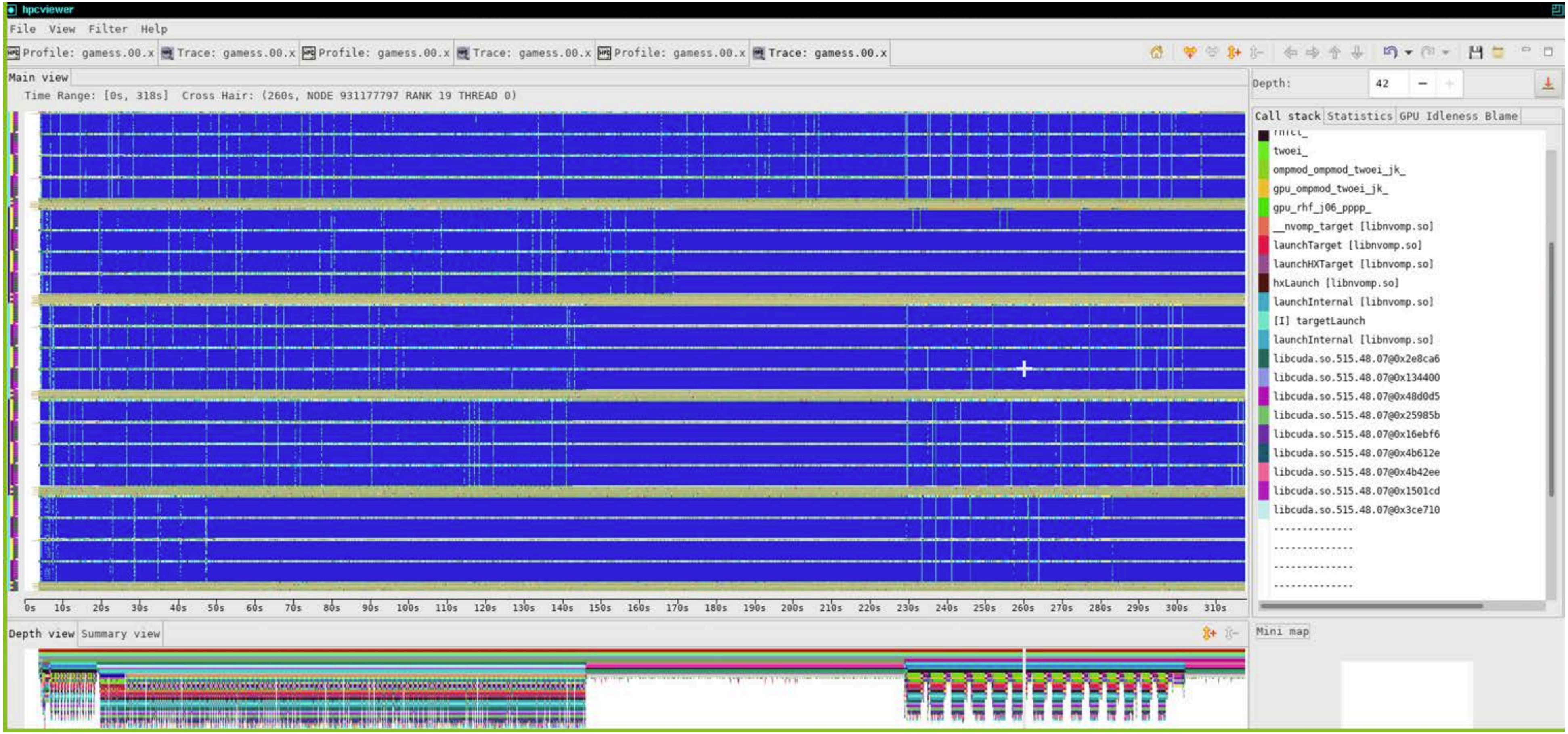


Top-down view | Bottom-up view | Flat view

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

GAMESS improved

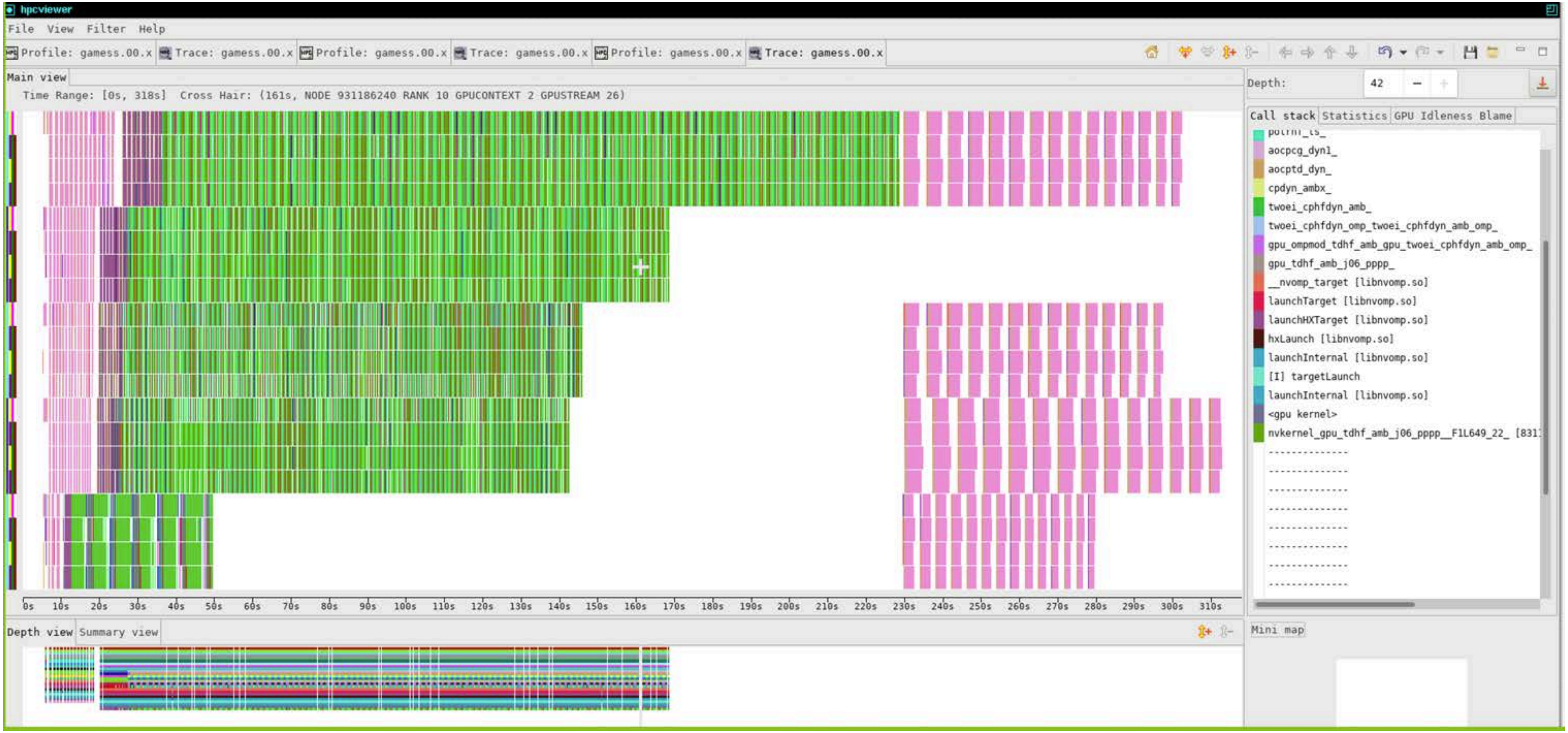
Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter



GAMESS improved

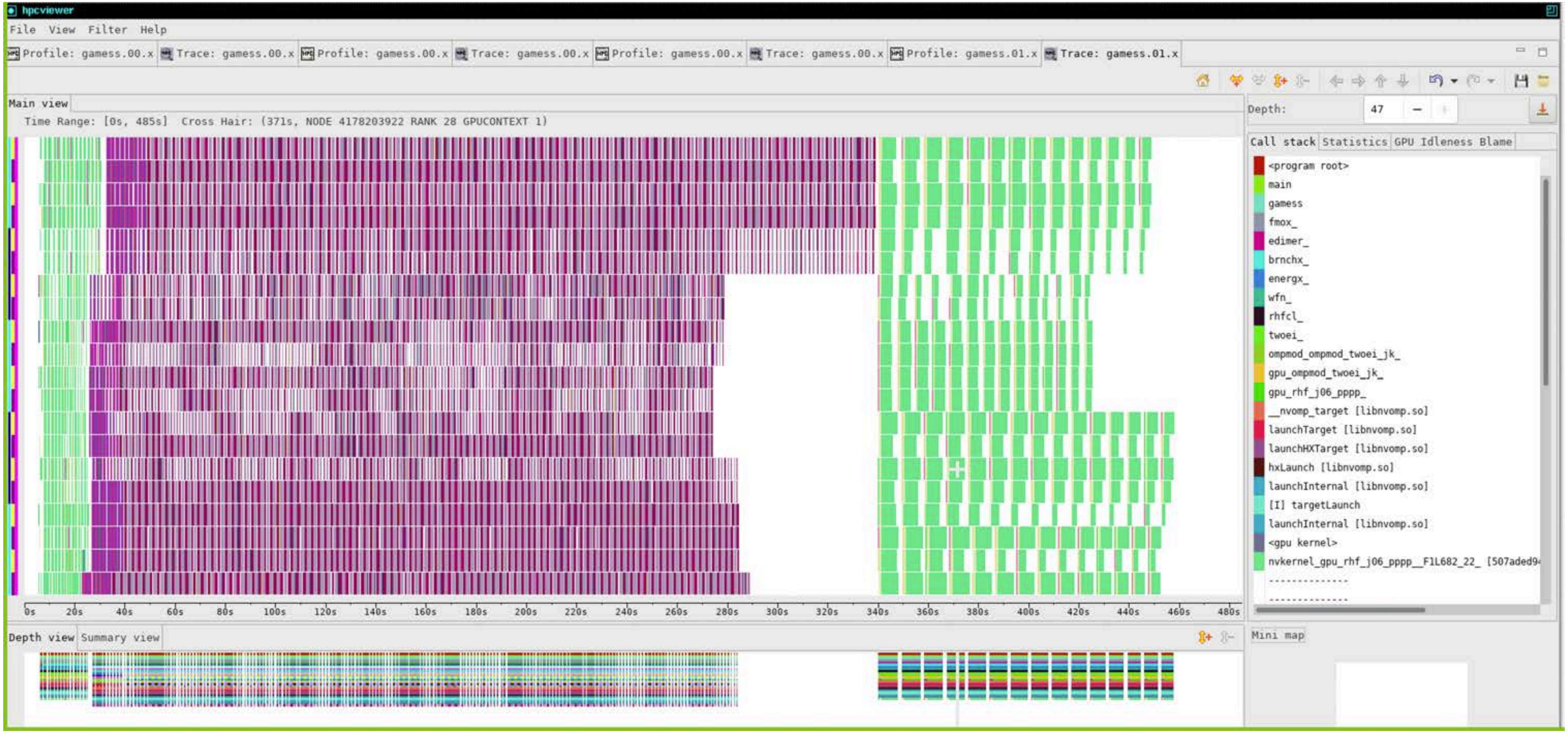
CPU Threads and GPU Streams

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



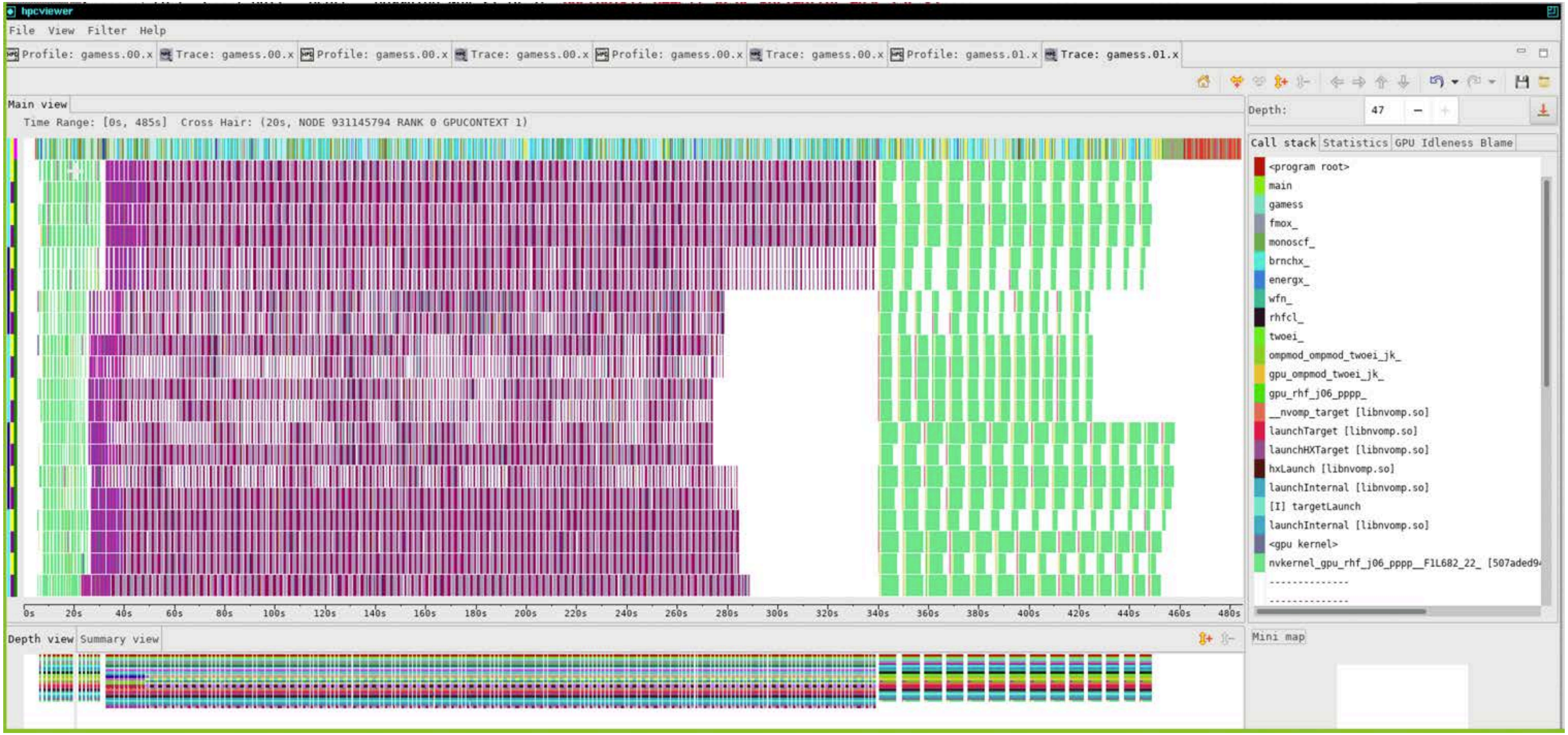
GAMESS improved

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



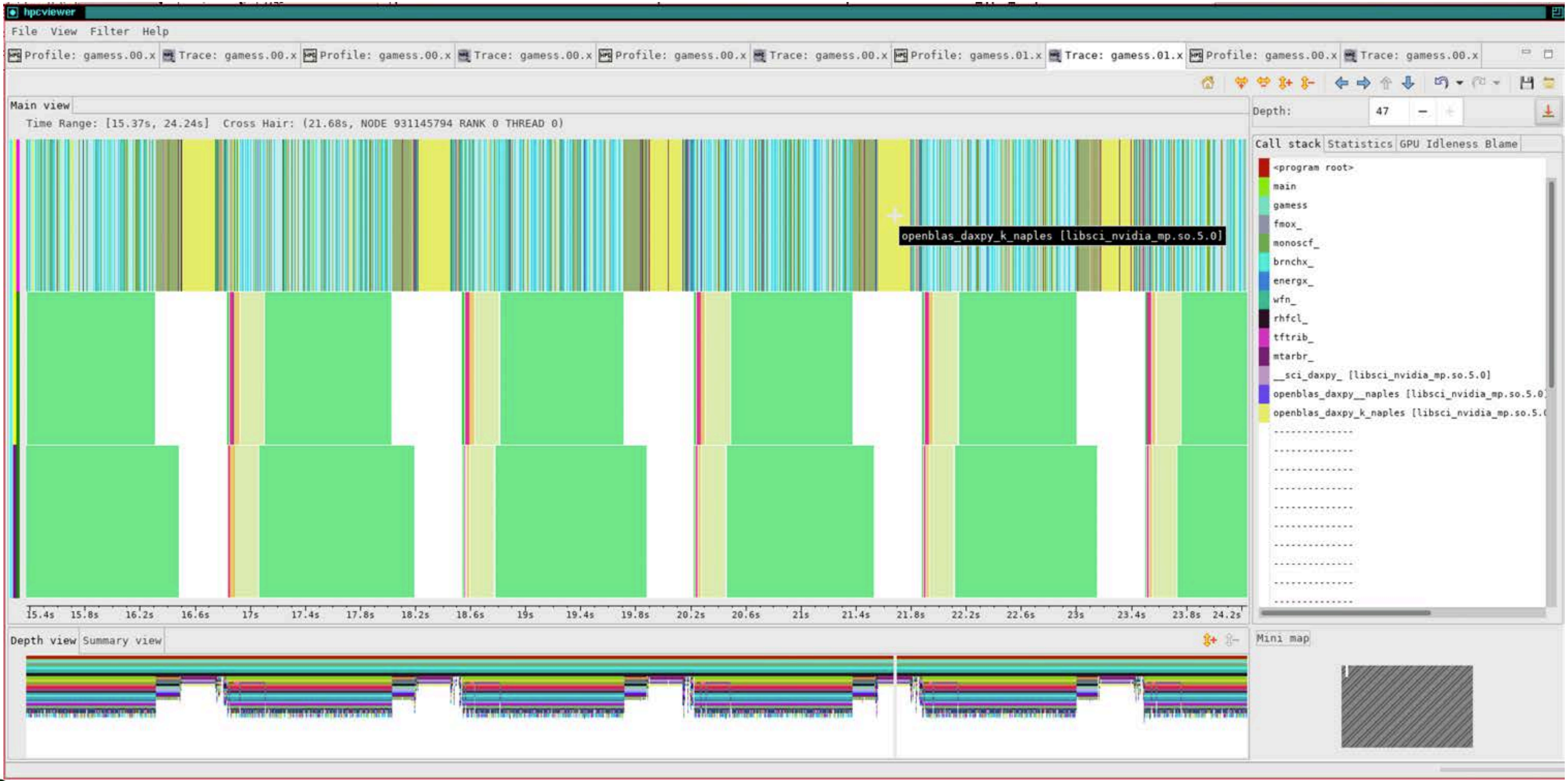
GAMESS improved with better manual distribution of work in input

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



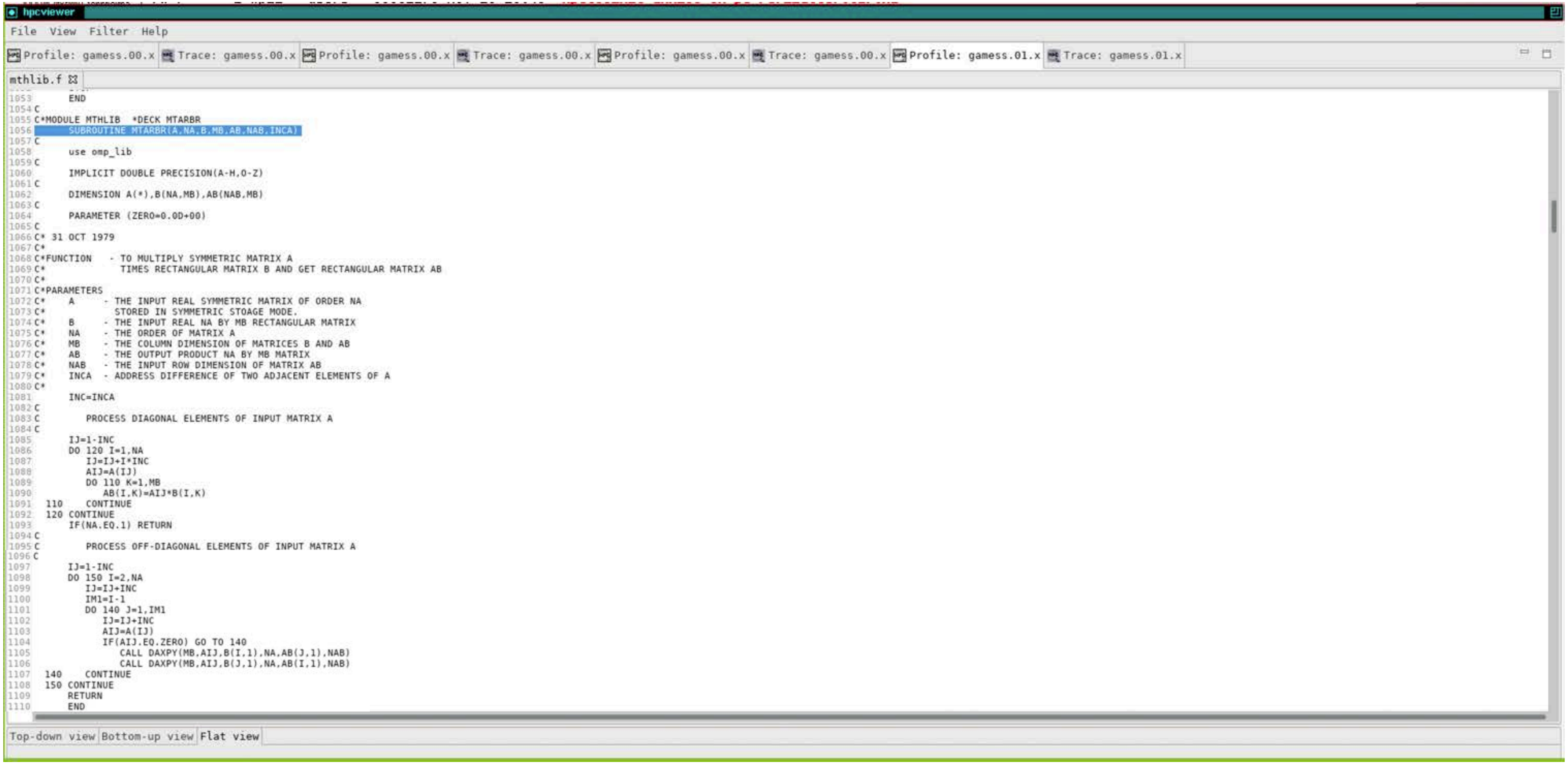
GAMESS improved adding Rank 0 Thread 0 to GPU streams

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



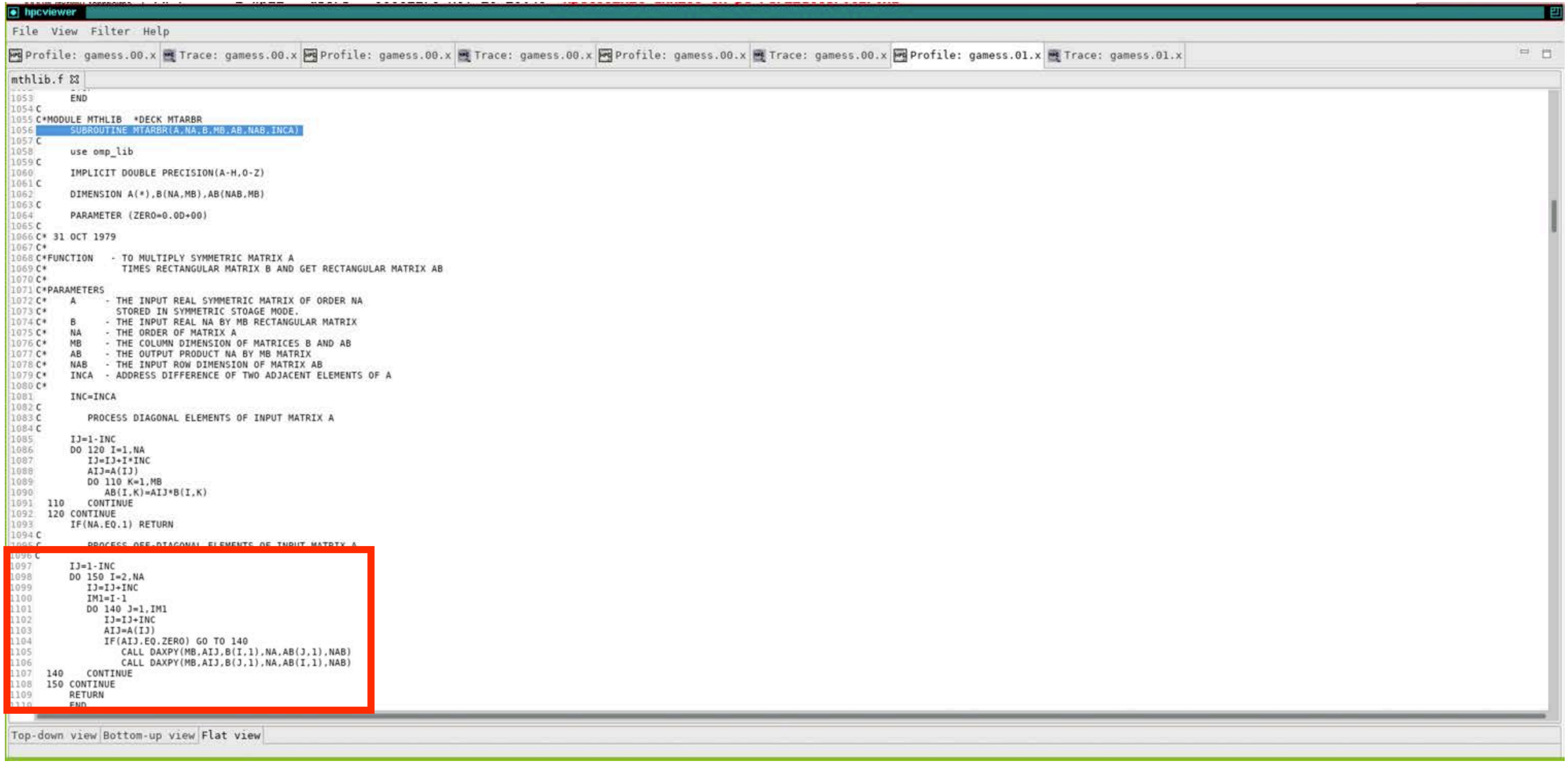
1 CPU Stream, 2 GPU Streams: 6 Iterations

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




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

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



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

hpcviewer

File View Filter Help

Profile: gamess.00.x | Trace: gamess.00.x | Profile: gamess.00.x | Trace: gamess.00.x | Profile: gamess.00.x | Trace: gamess.00.x | Profile: gamess.01.x | Trace: gamess.01.x

mthlib.f

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

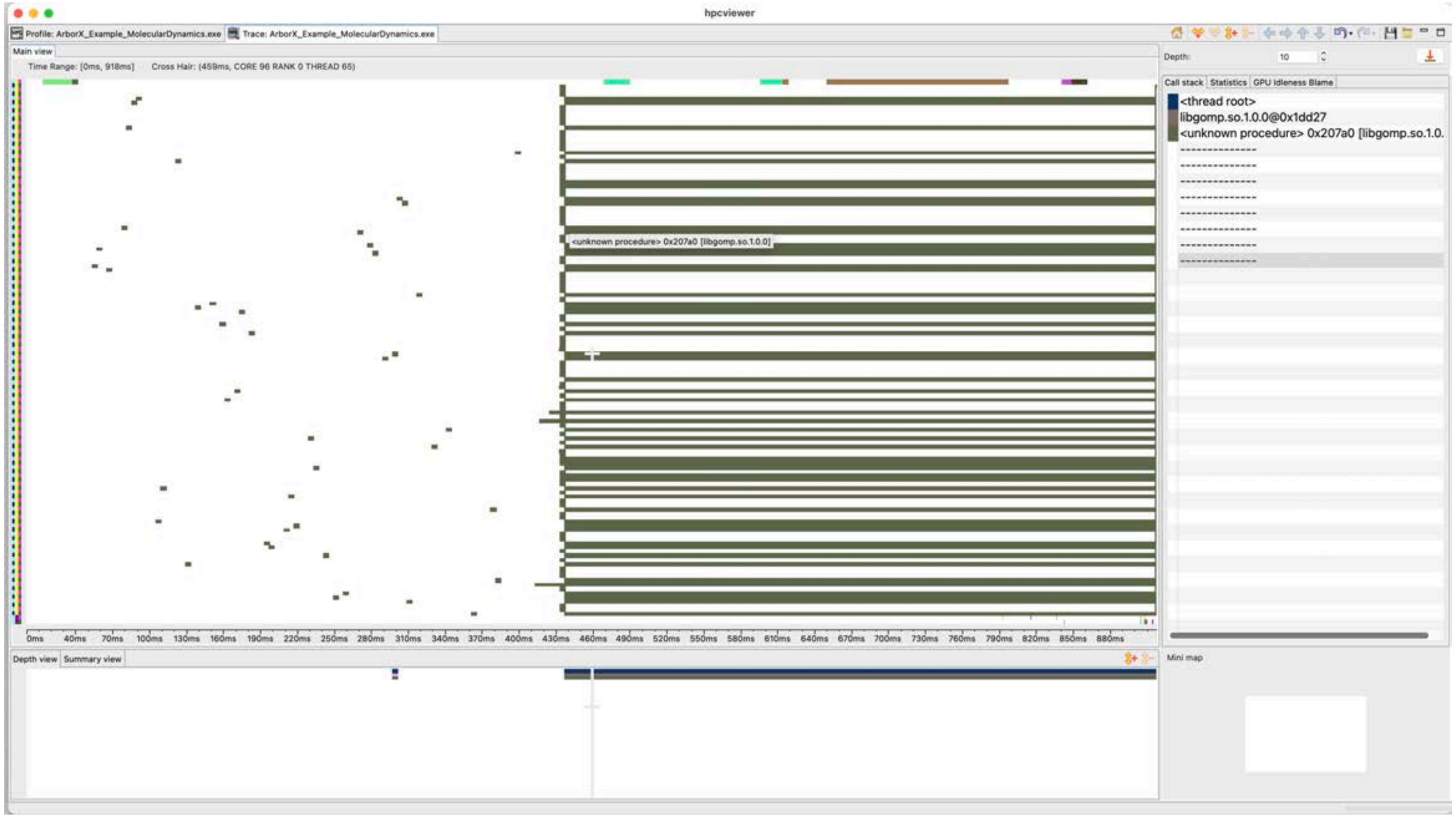
Top-down view | Bottom-up view | Flat view

Case Study: ArborX

- A library written in Kokkos that provides performance portable algorithms for geometric search

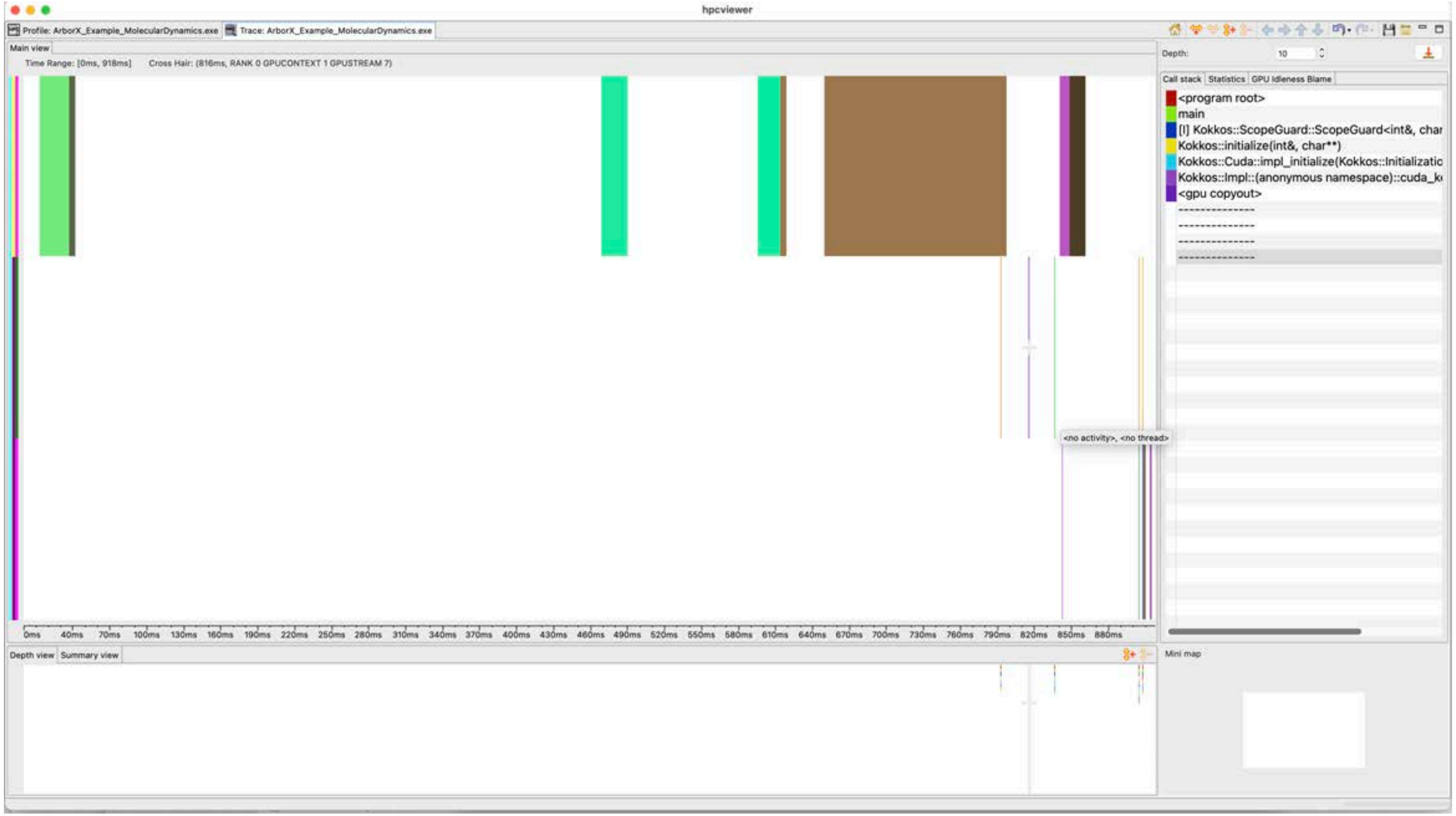
ArborX Trace: Lots of irrelevant CPU Trace Lines for Idle Threads

- Solution: Filter out trace lines with very small numbers of samples

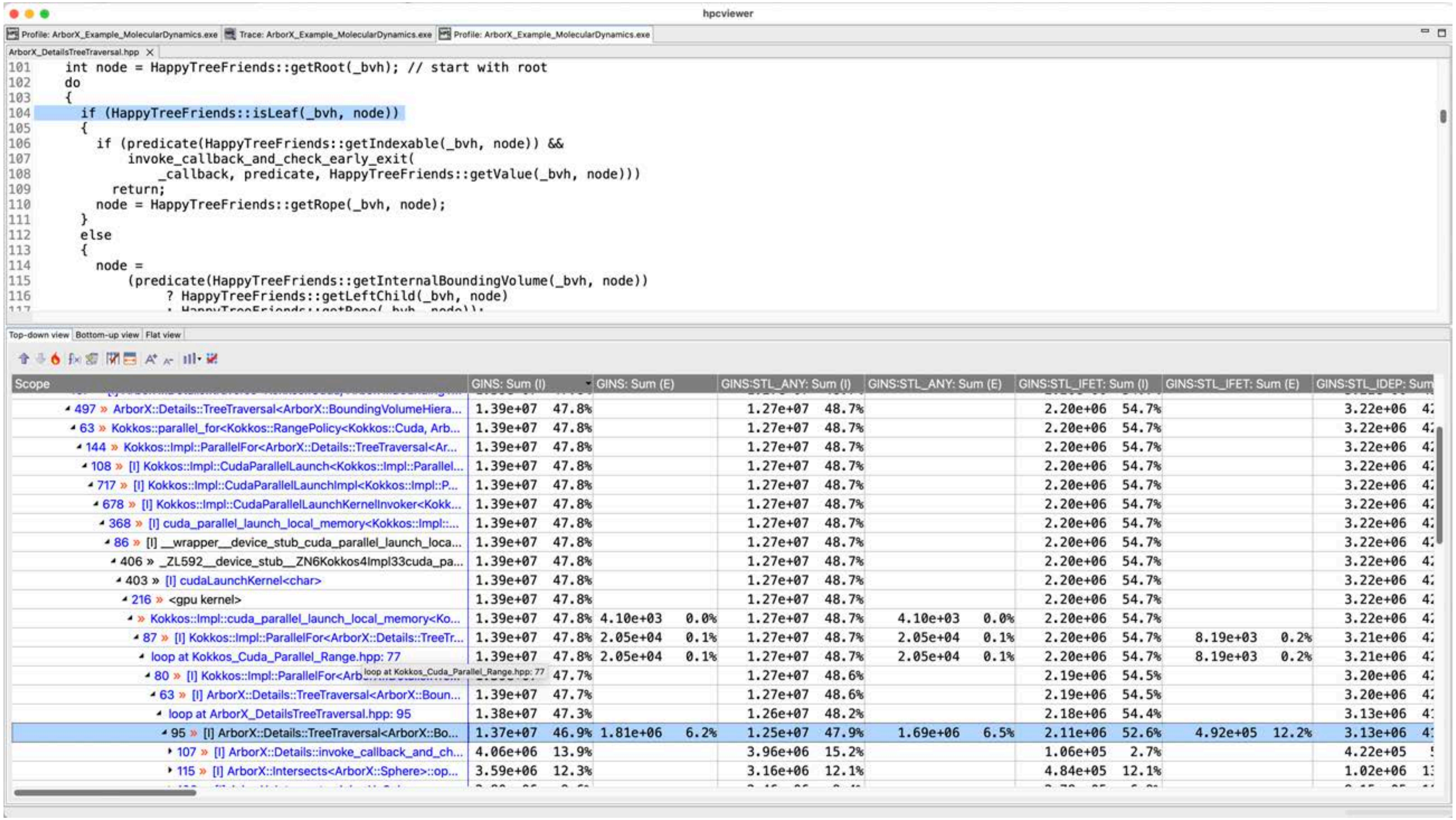


ArborX Trace: Filter to Focus on Relevant CPU and GPU Traces

- Use Filter➞Filter Ranks: select Rank 0 and GPU trace lines



ArborX Trace: PC sampling of ArborX



Key Metrics Available for GPU Kernels

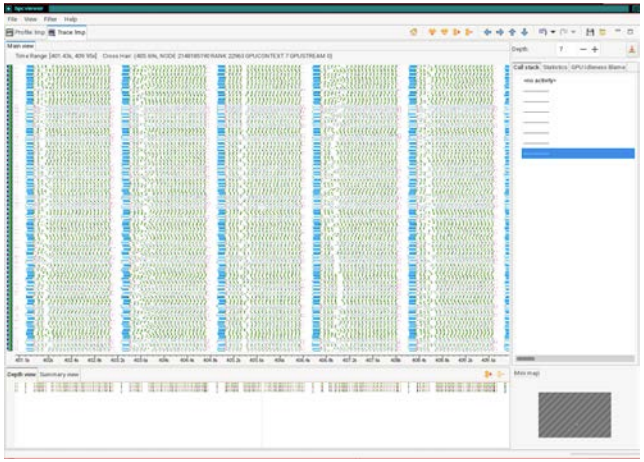
- GPUOP: GPU operation time (kernel launch, copies, etc.)
- GXCOPY:* GPU copies of various kinds
- GKER: GPU kernel time
- GKER:FGP_ACT: fine grain parallelism actual (number of threads used)
- GKER:FGP_MAX: maximum possible fine-grain parallelism (number of threads possible)
- GKER:BLK_THR: threads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

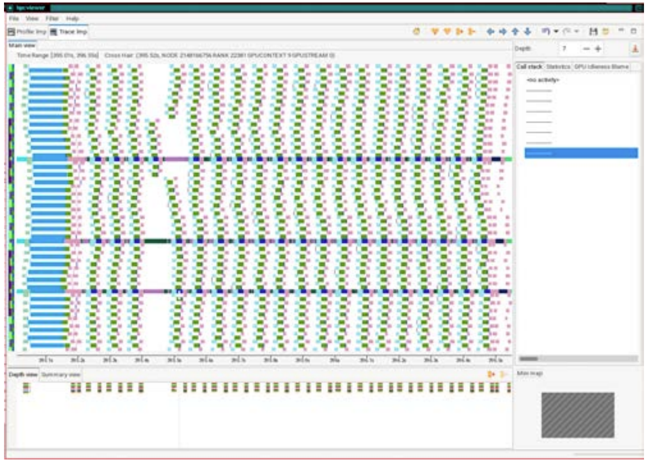
What Metrics Are Available for GPU Kernels with PC Sample

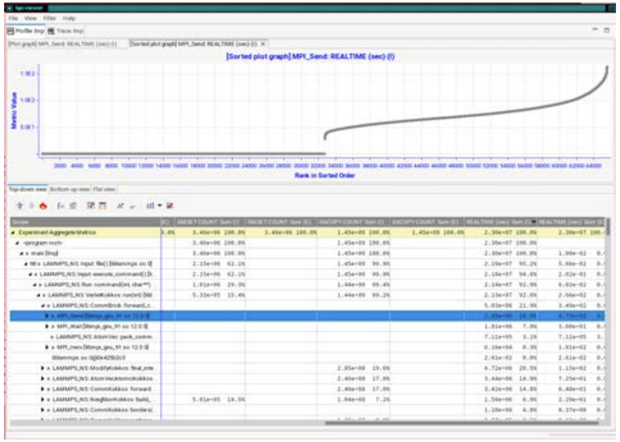
- GINS: GPU instructions
- GINS:STL_ANY: GPU instruction stalls for any reason
- GINS:STL_IFET: GPU instruction stalls for instruction fetch
- GINS:STL_GMEM: GPU instruction stalls for global memory
- GINS:STL_CMEM: GPU instruction stalls for constant memory
- GINS:STL_IDEP: GPU instruction stalls for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

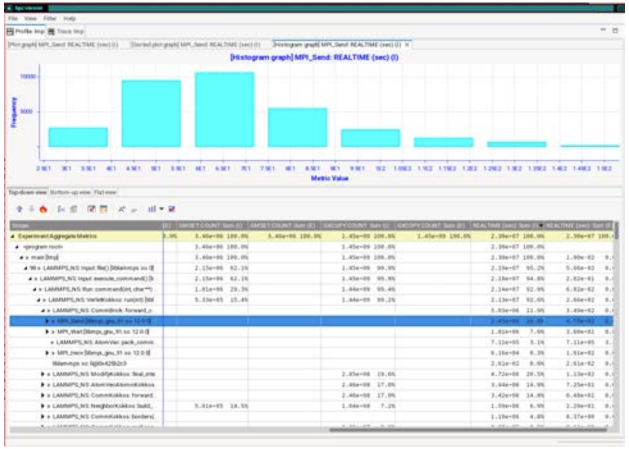
- GSAMP:EXP: expected number of samples
- GSAMP:TOT: total number of samples recorded
- GSAMP:UTIL: GPU utilization = (PC samples expected) / (PC samples total)

LAMMPS on Frontier: Executions with Kernel Duration of Milliseconds

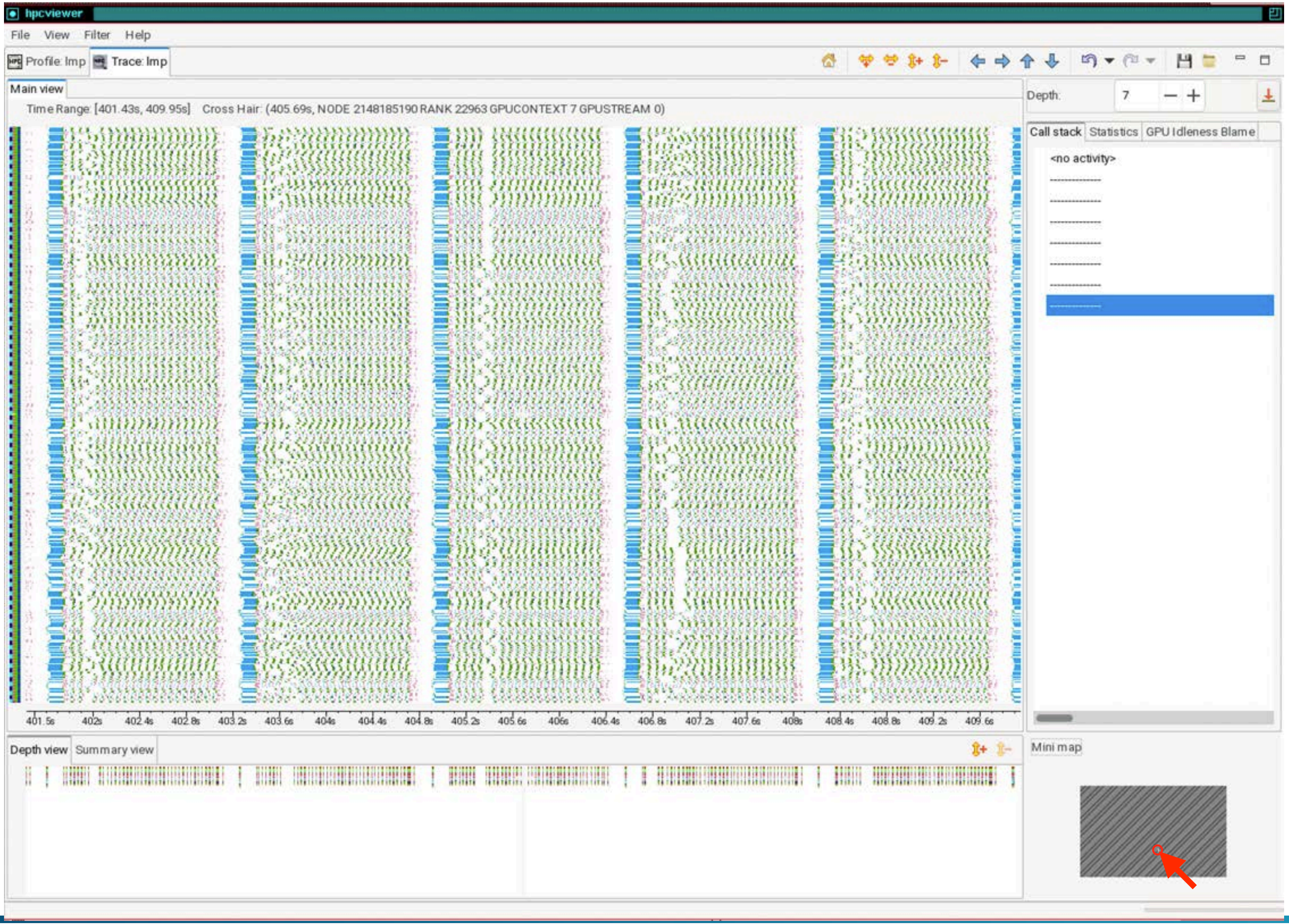




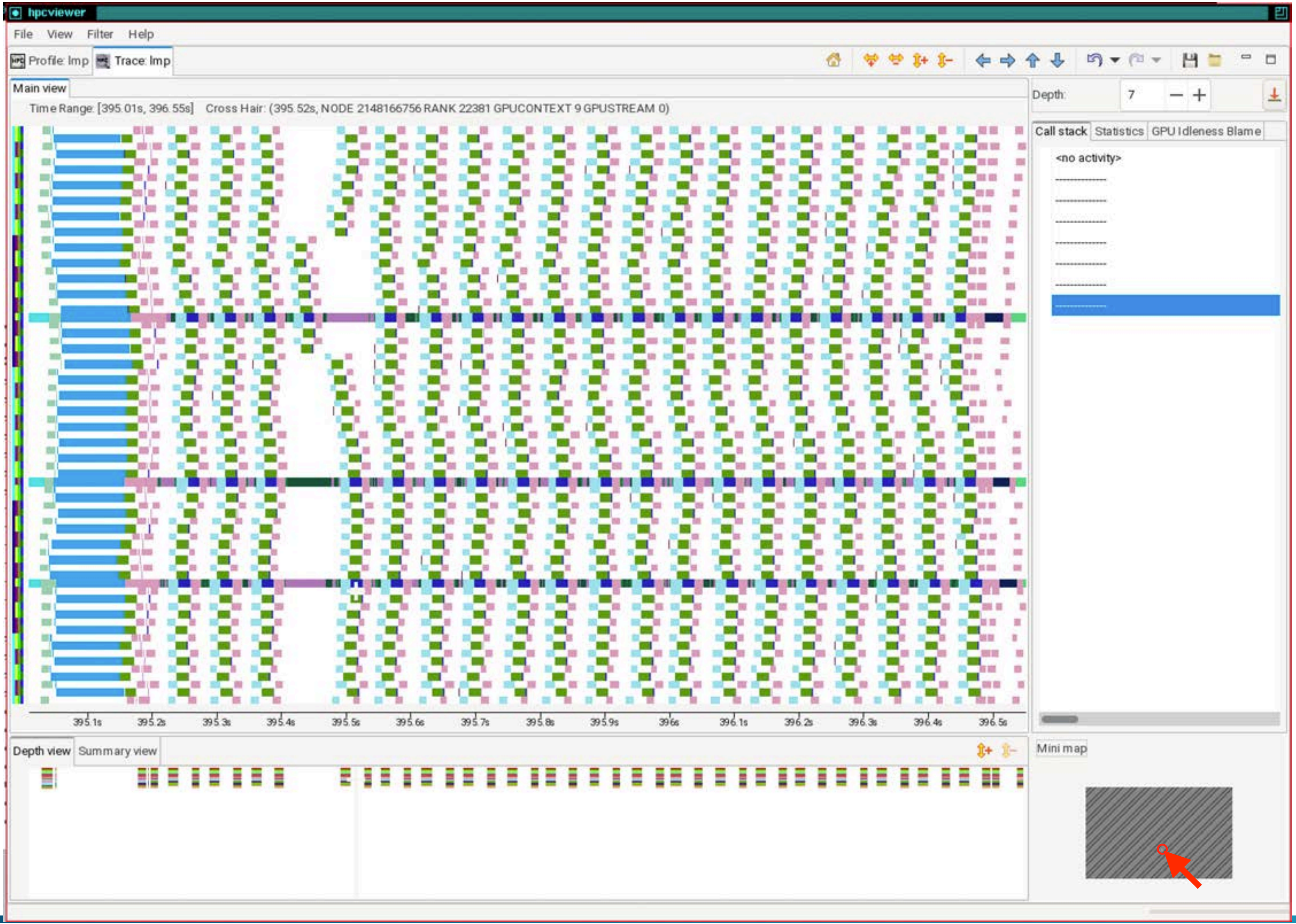




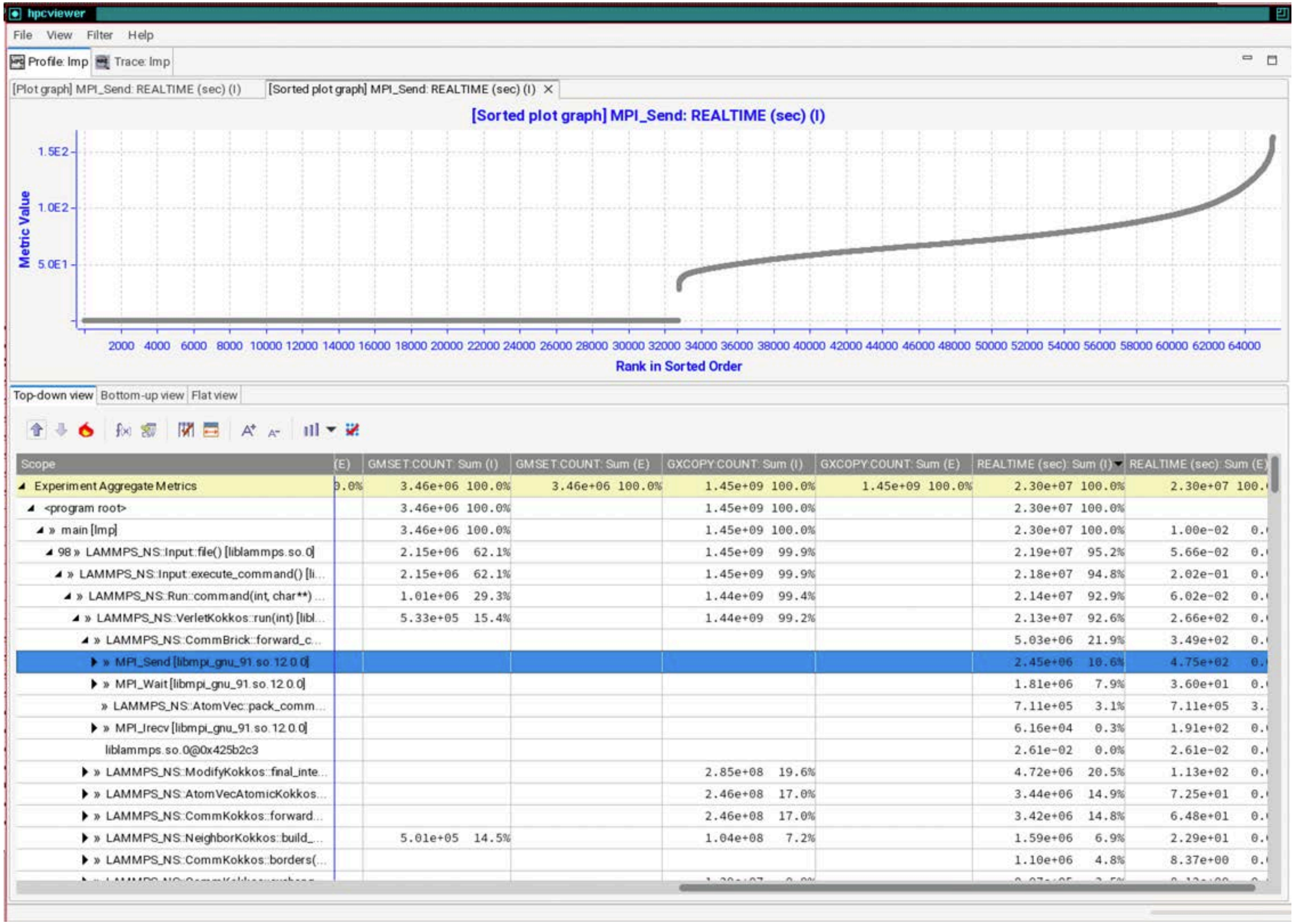
LAMMPS on Frontier: Executions with Kernel Duration of Milliseconds



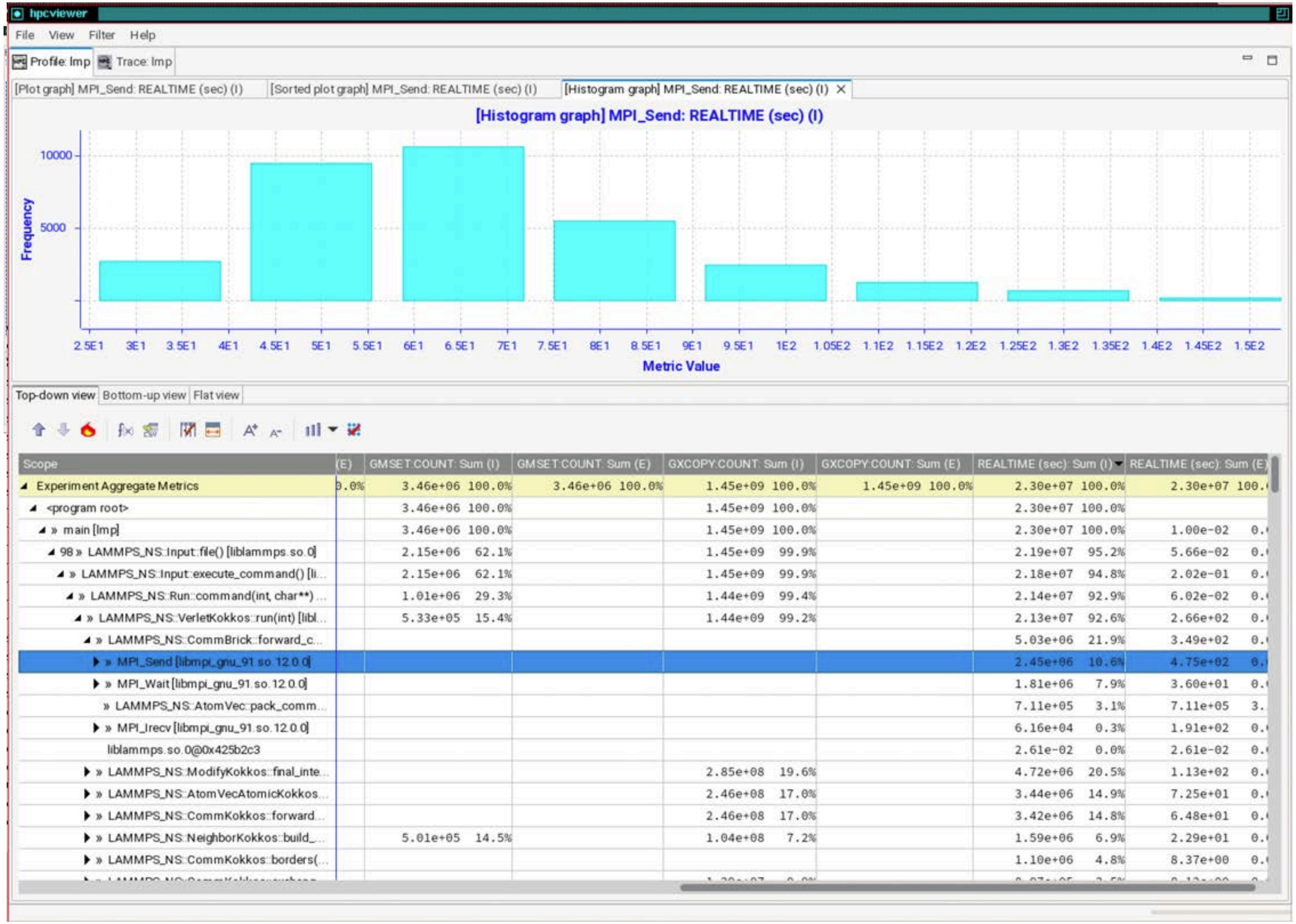
LAMMPS on Frontier: Executions with Kernel Duration of Milliseconds



LAMMPS on Frontier: Executions with Kernel Duration of Milliseconds

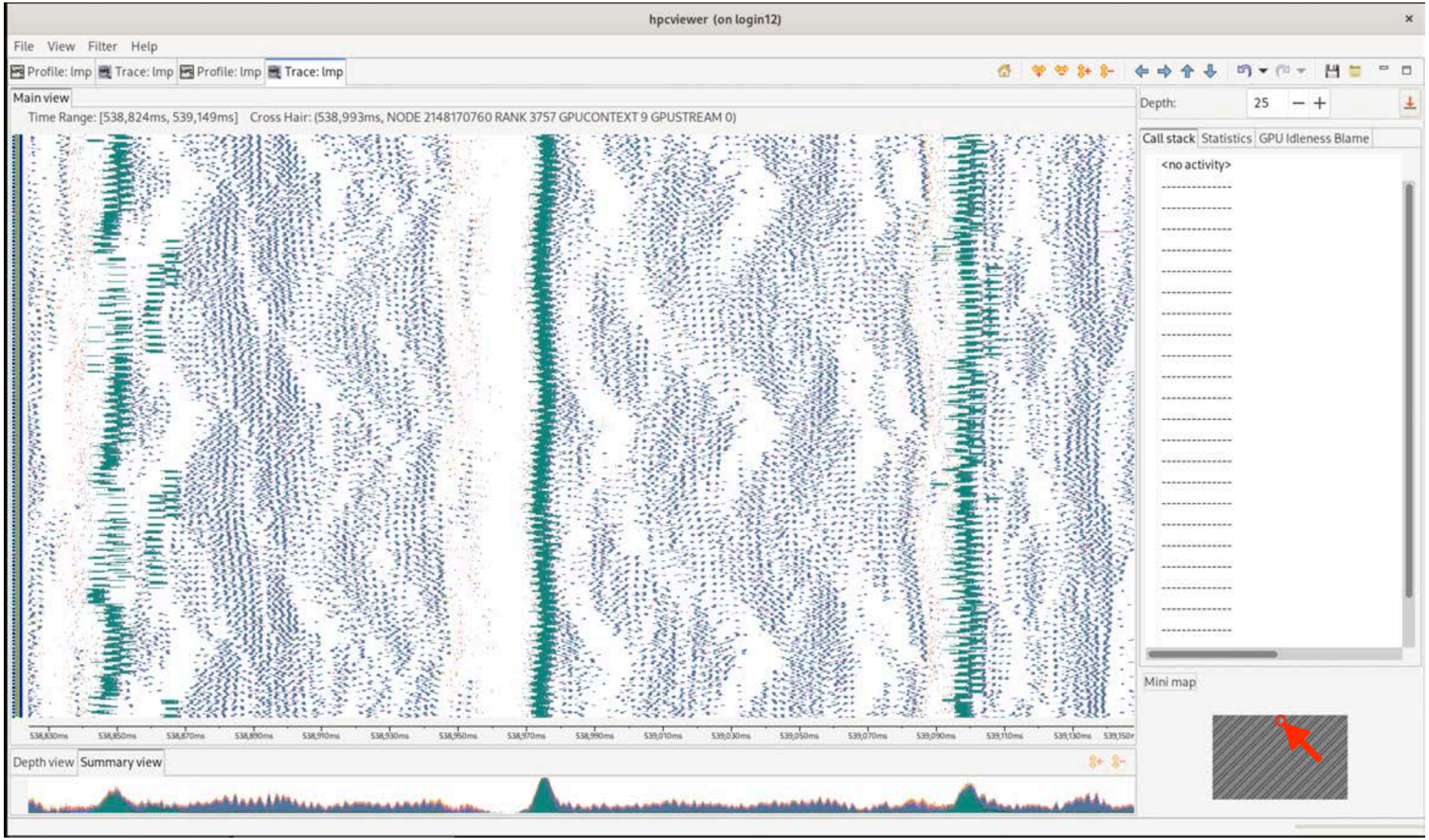


LAMMPS on Frontier: Executions with Kernel Duration of Milliseconds



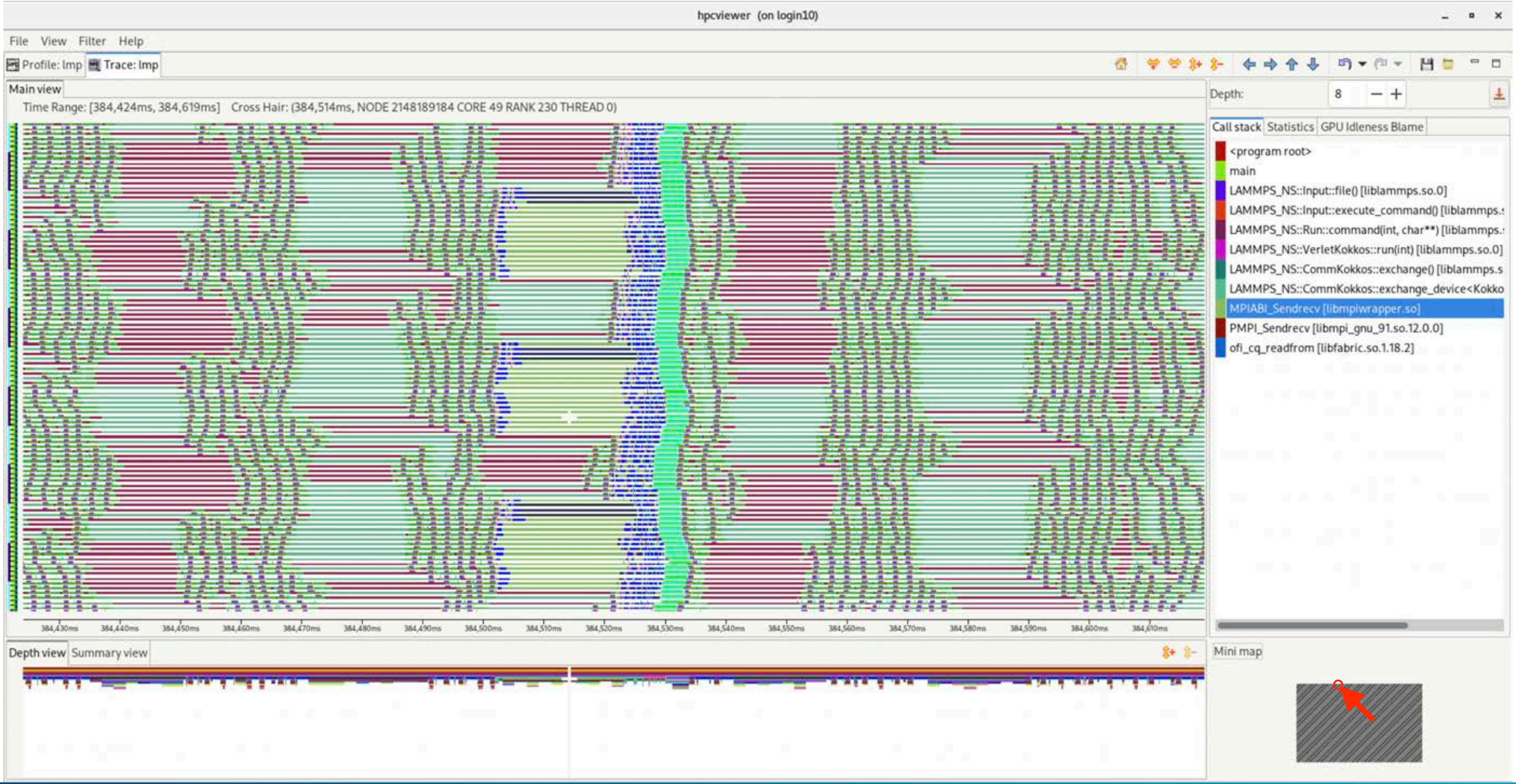
LAMMPS on Frontier: 8K nodes, 64K MPI ranks + 64K GPU tiles

Kernel duration of microseconds



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

Kernel duration of microseconds

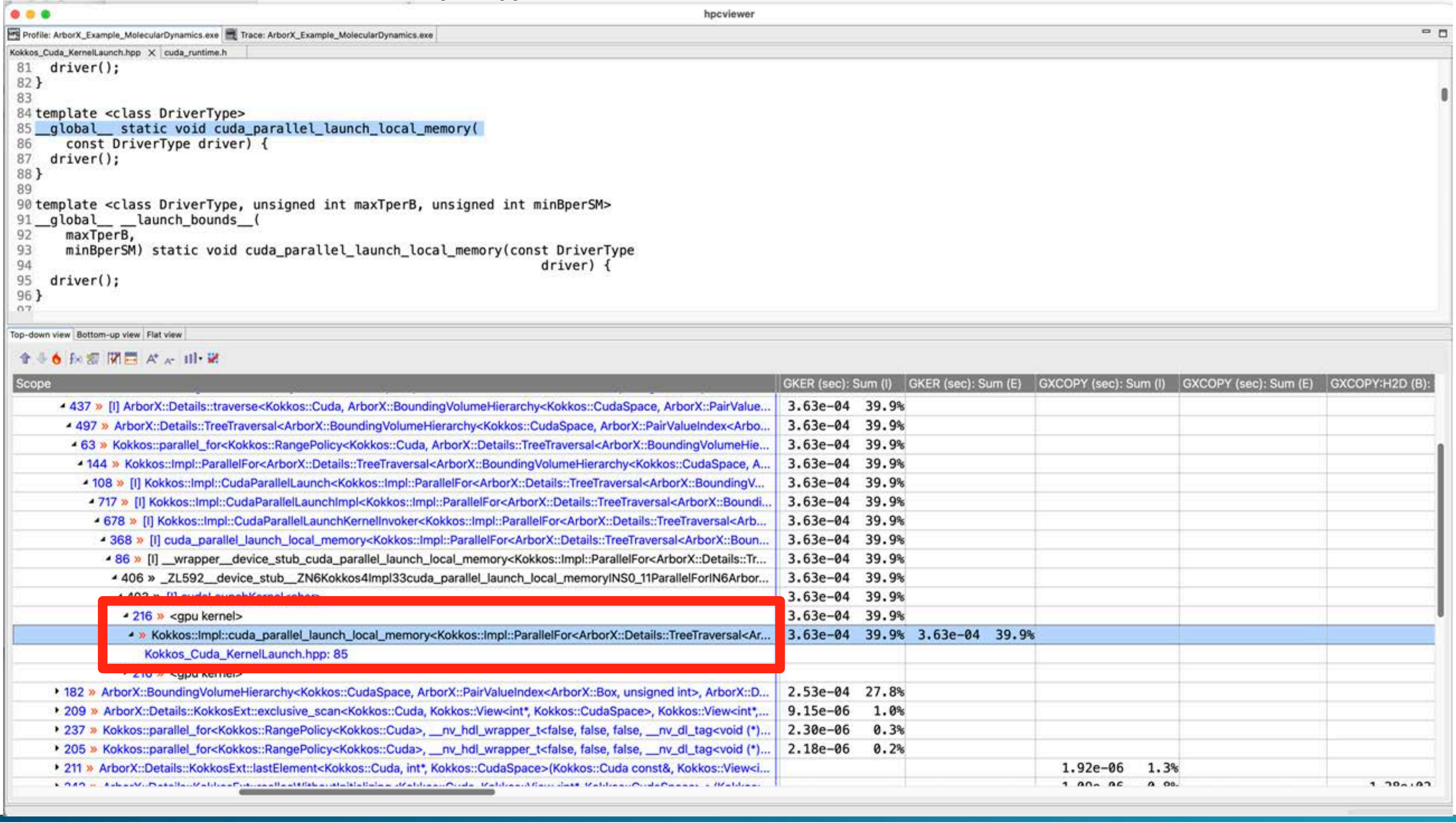


Coming Attractions

- Developing comprehensive support for NVTX/ROCTX/Caliper/Kokkos Labels
- Support for instruction-level measurement and attribution on AMD and Intel GPUs
- New GUI support for analysis of remote data
- Python-based interface for analysis of performance results

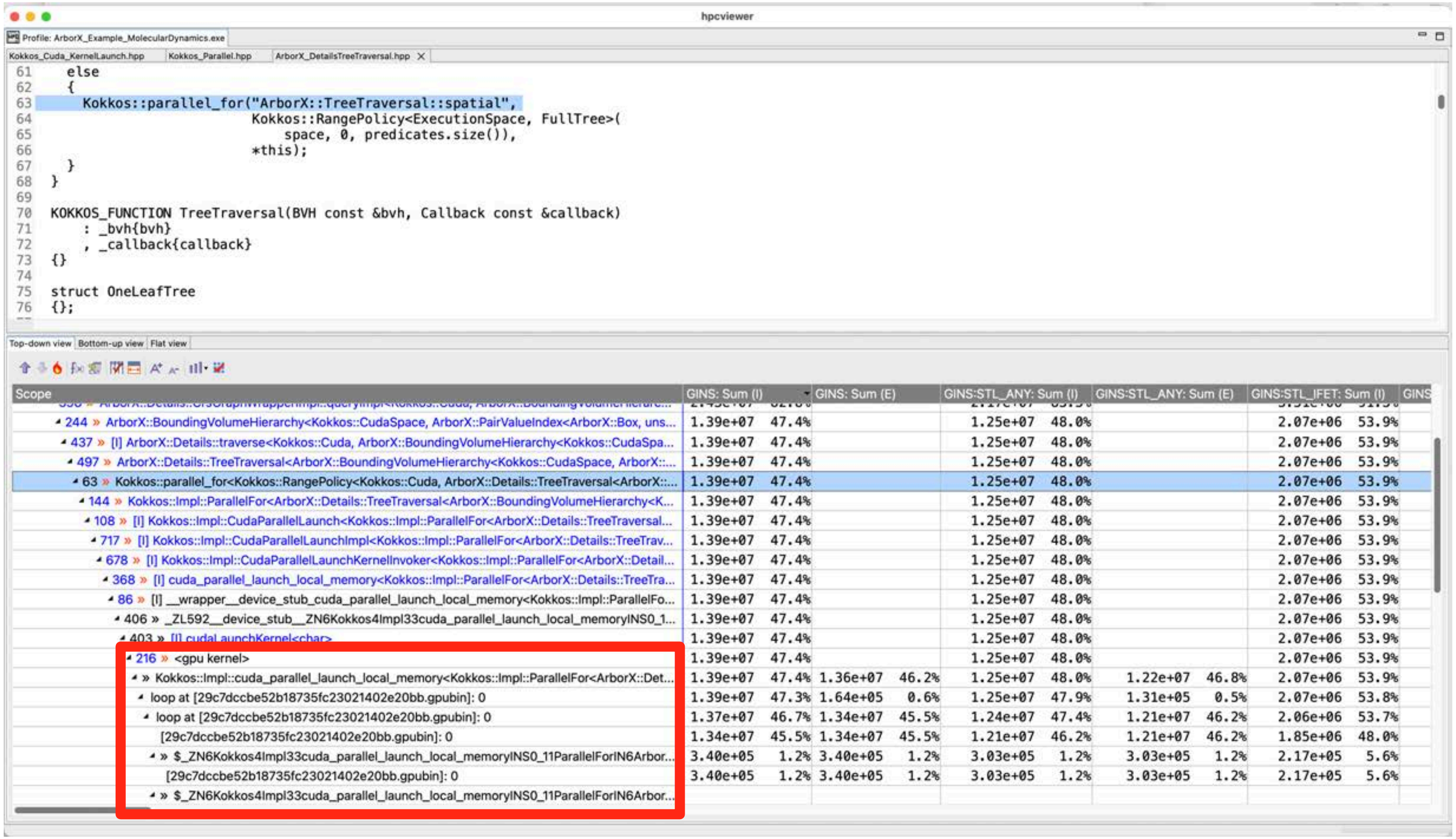
Troubleshooting: Only GPU kernel Name

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



Troubleshooting: No GPU source code lines with PC sampling

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



Troubleshooting: Compiling ArborX with GPU Line Map Info

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

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

HPCToolkit Resources

- Documentation
  - User manual
    - http://hpctoolkit.org/manual/HPCToolkit-users-manual.pdf
  - 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
  - Cheat sheet
    - https://gitlab.com/hpctoolkit/hpctoolkit/-/wikis/home
- 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