ARGONNE
ATPESC2023
EXTREME-SCALE COMPUTING

HPCToolkit Performance Tools

Performance analysis of CPU and GPU-accelerated applications at Scale

HPCToolkit at Exascale on Frontier: 8K nodes, 64K MPI ranks + GPU Tiles

John Mellor-Crummey
Professor, Rice University

HPCToolkit Funding Acknowledgments

Government

- Exascale Computing Project 17-SC-20-SC
- Lawrence Livermore National Laboratory Subcontract B658833
- Argonne National Laboratory Subcontract 9F-60073

Corporate

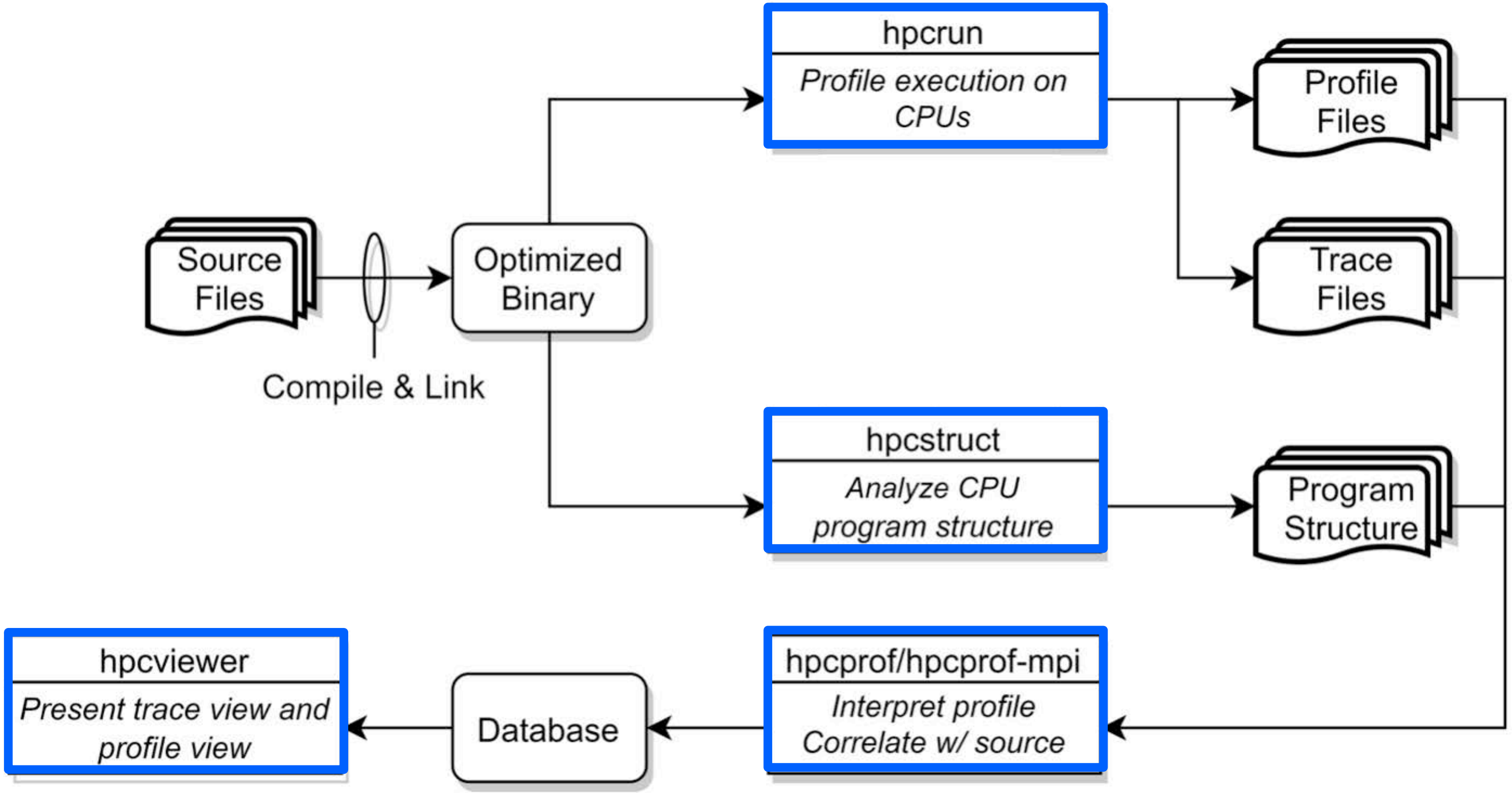
- Advanced Micro Devices
- TotalEnergies EP Research & Technology USA, LLC.

Rice University’s HPCToolkit Performance Tools

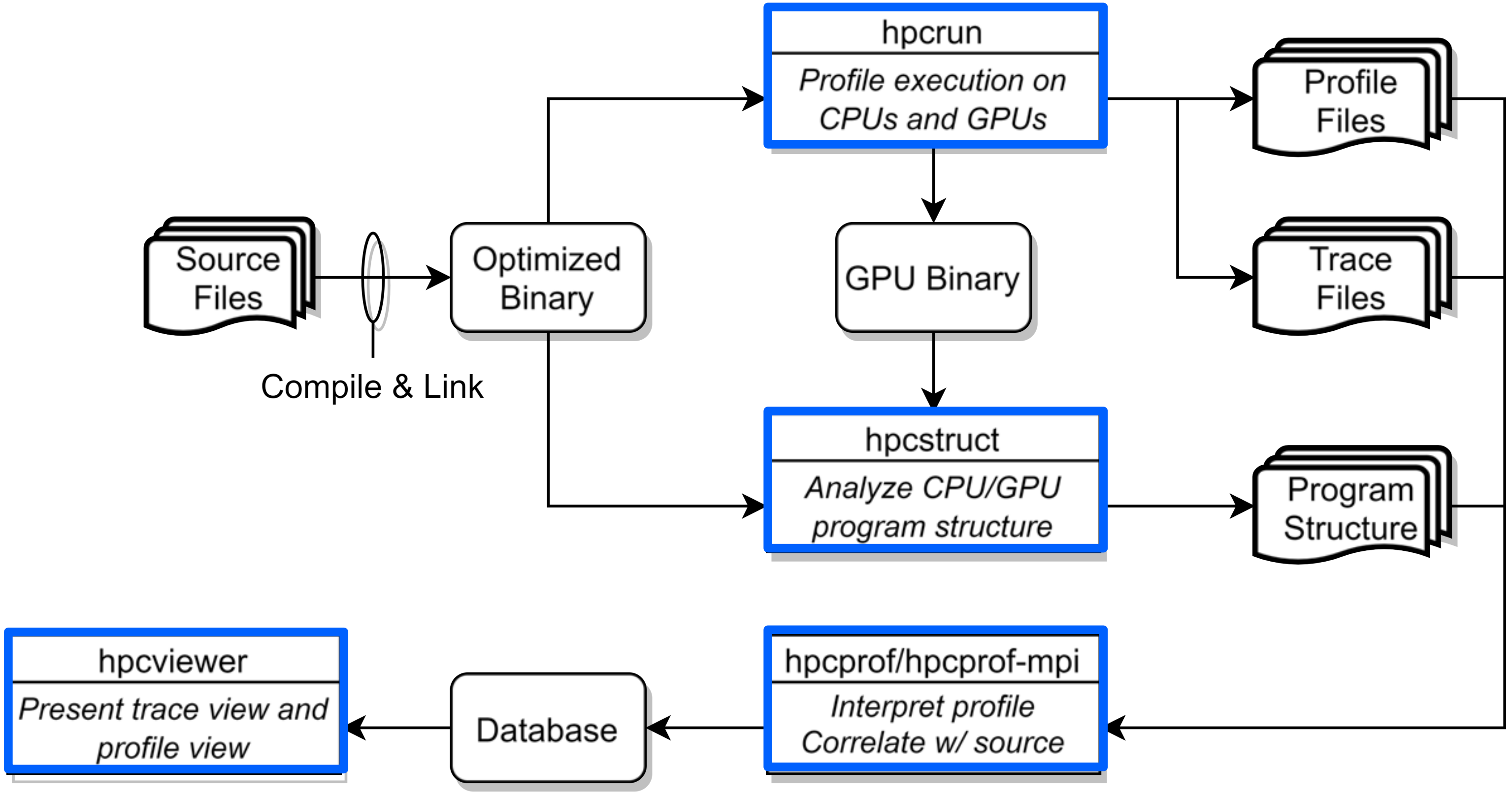
Measure and analyze performance of CPU and GPU-accelerated applications

- 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

HPCToolkit's Workflow for CPU Applications



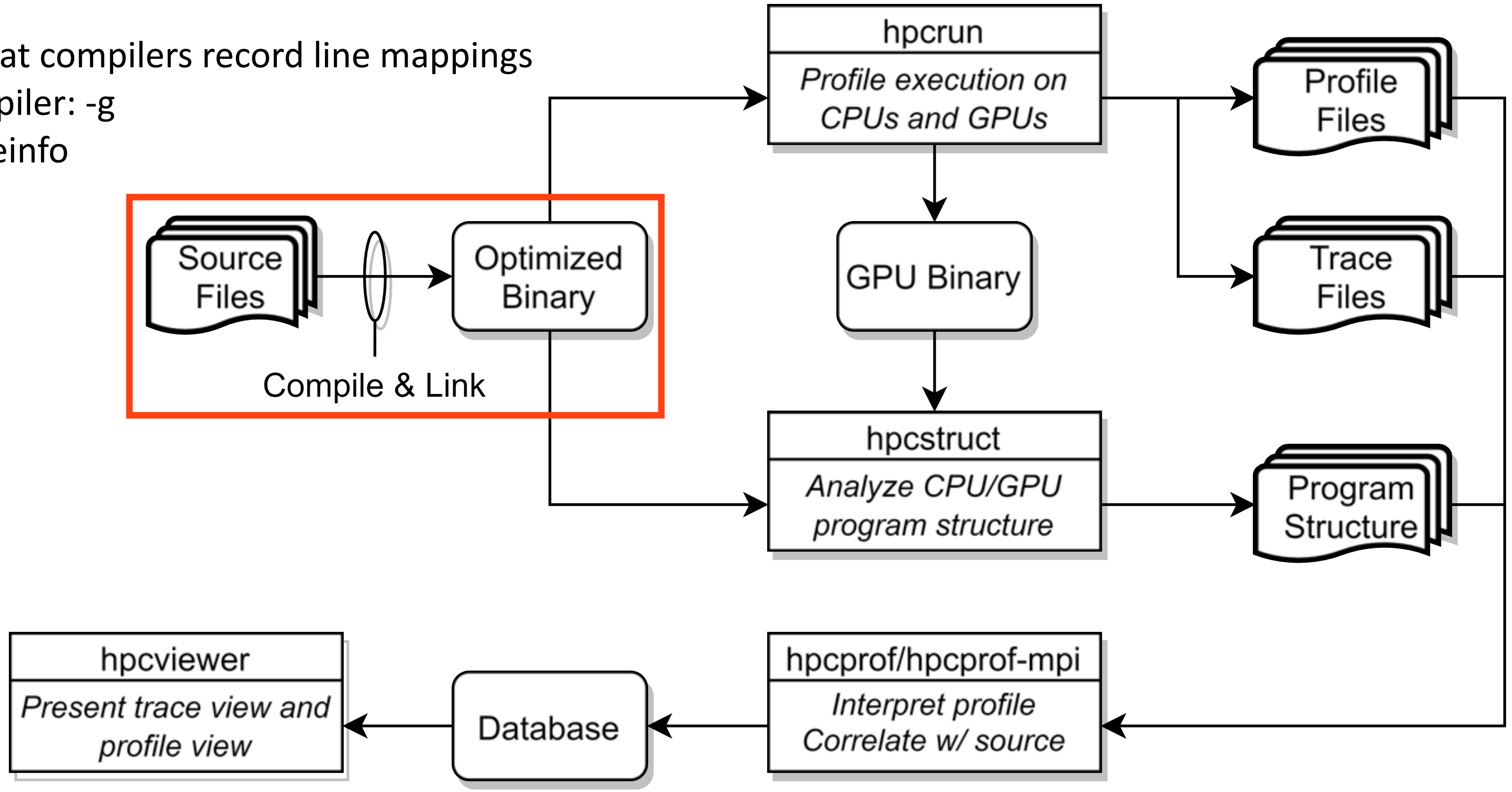
HPCToolkit's Workflow for GPU-accelerated Applications



HPCToolkit's Workflow for GPU-accelerated Applications

Step 1:

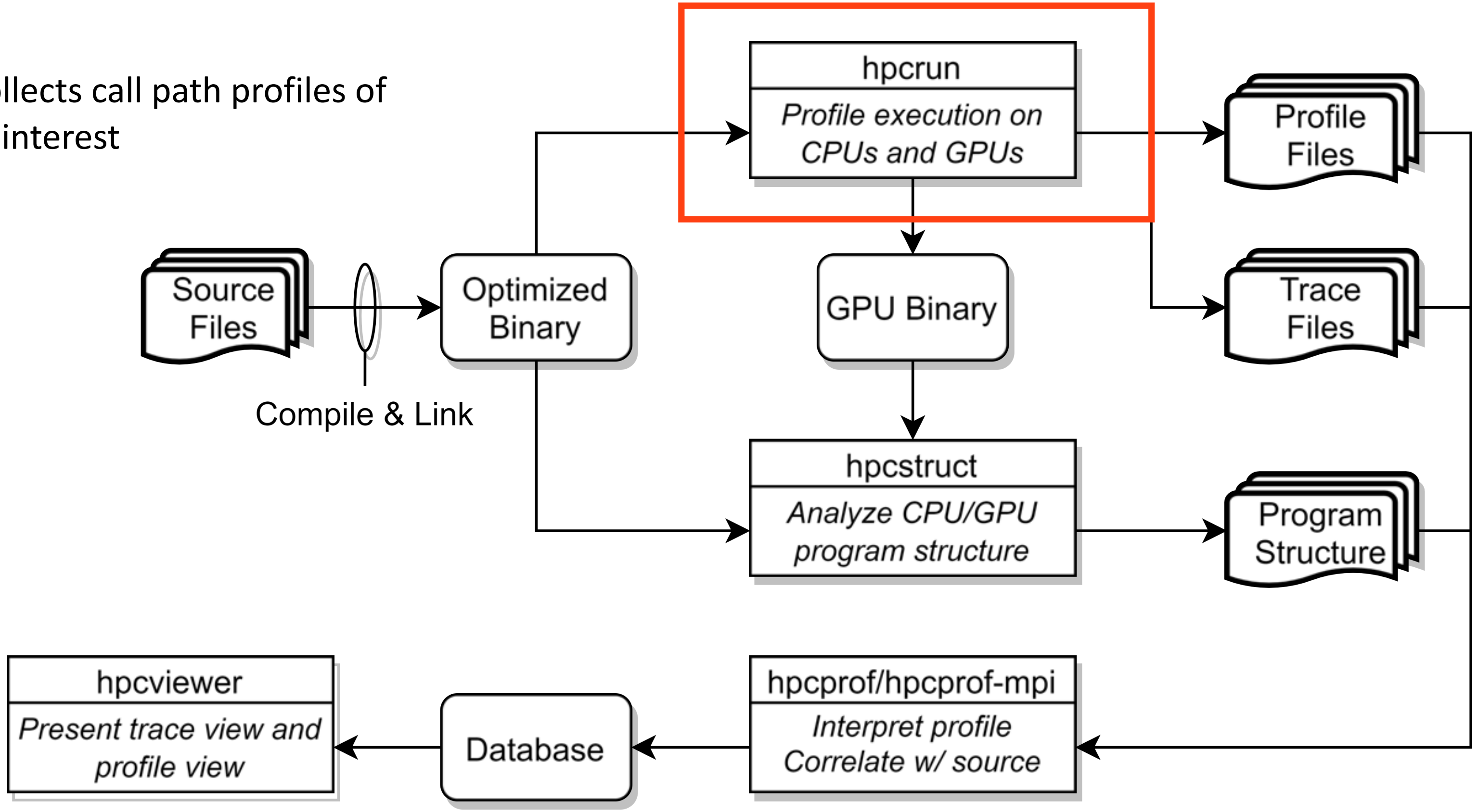
- Ensure that compilers record line mappings
- host compiler: -g
- nvcc: -lineinfo



HPCToolkit's Workflow for GPU-accelerated Applications

Step 2:

- *hpcrun* collects call path profiles of events of interest



Measurement of CPU and GPU-accelerated Applications

CPU

- Sampling on timer interrupts and hardware counter overflows on the CPU

GPU

- Callbacks when GPU operations are launched/completed
- GPU event stream for GPU operations
- PC Samples in GPU kernels (NVIDIA)
- Instruction-level instrumentation (Intel)

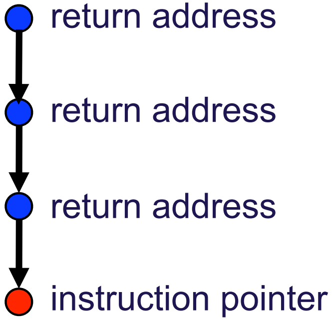
Call Stack Unwinding to Attribute Costs in Context

Unwind when timer or hardware counter overflows

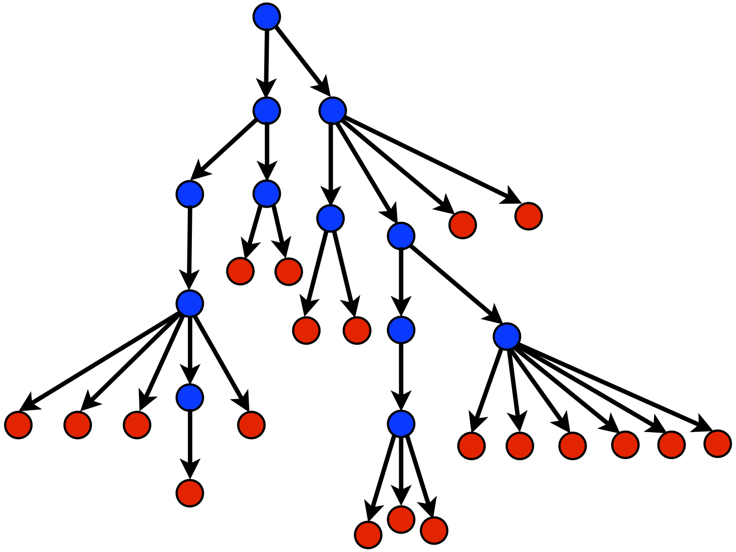
- measurement overhead proportional to sampling frequency rather than call frequency

Unwind to capture context for GPU events such as kernel launches and data copies

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

```
jsrun -n 32 -g 1 -a 1 hpcrun -e gpu=xxx <app>
srun -n 1 -G 1 hpcrun -e gpu=xxx <app>
aprun -n 16 -N 8 -d 8 hpcrun -e gpu=xxx <app>
```

Profiles: aggregated on the fly
- a calling context tree per thread
- a calling context tree per GPU stream
- instruction level measurements

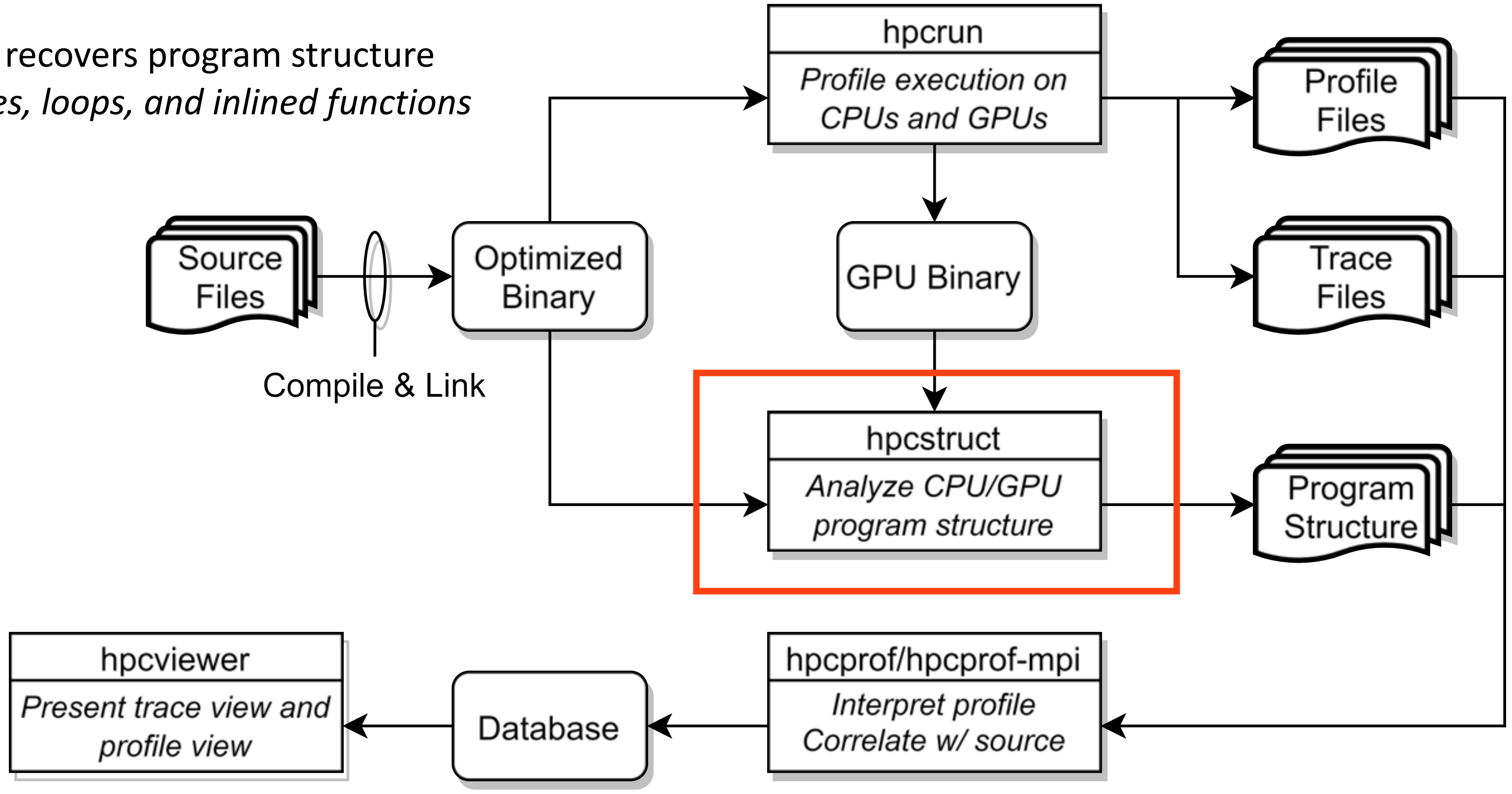
CPU traces
- trace of call stack samples

GPU traces
- trace of call stacks that initiate GPU operations

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

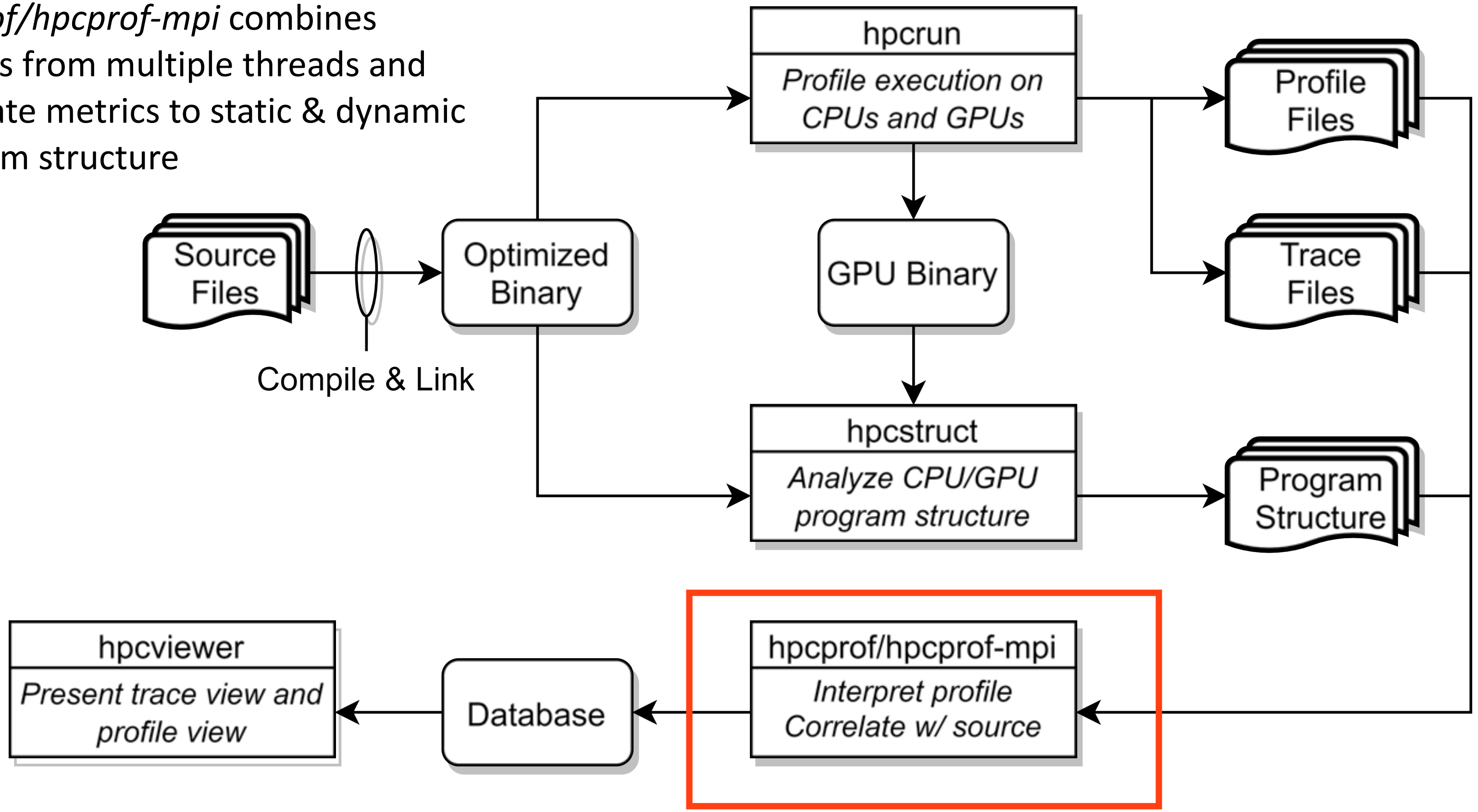
- Recover program structure information
  - Files, functions, inlined templates or functions, loops, source lines
- In parallel, analyze all CPU and GPU binaries that were measured by HPCToolkit
  - default: use size(CPU set)/2 threads
  - analyze large application binaries with 16 threads
  - analyze 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 when NVIDIA PC samples were collected

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 threaded parallelism

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

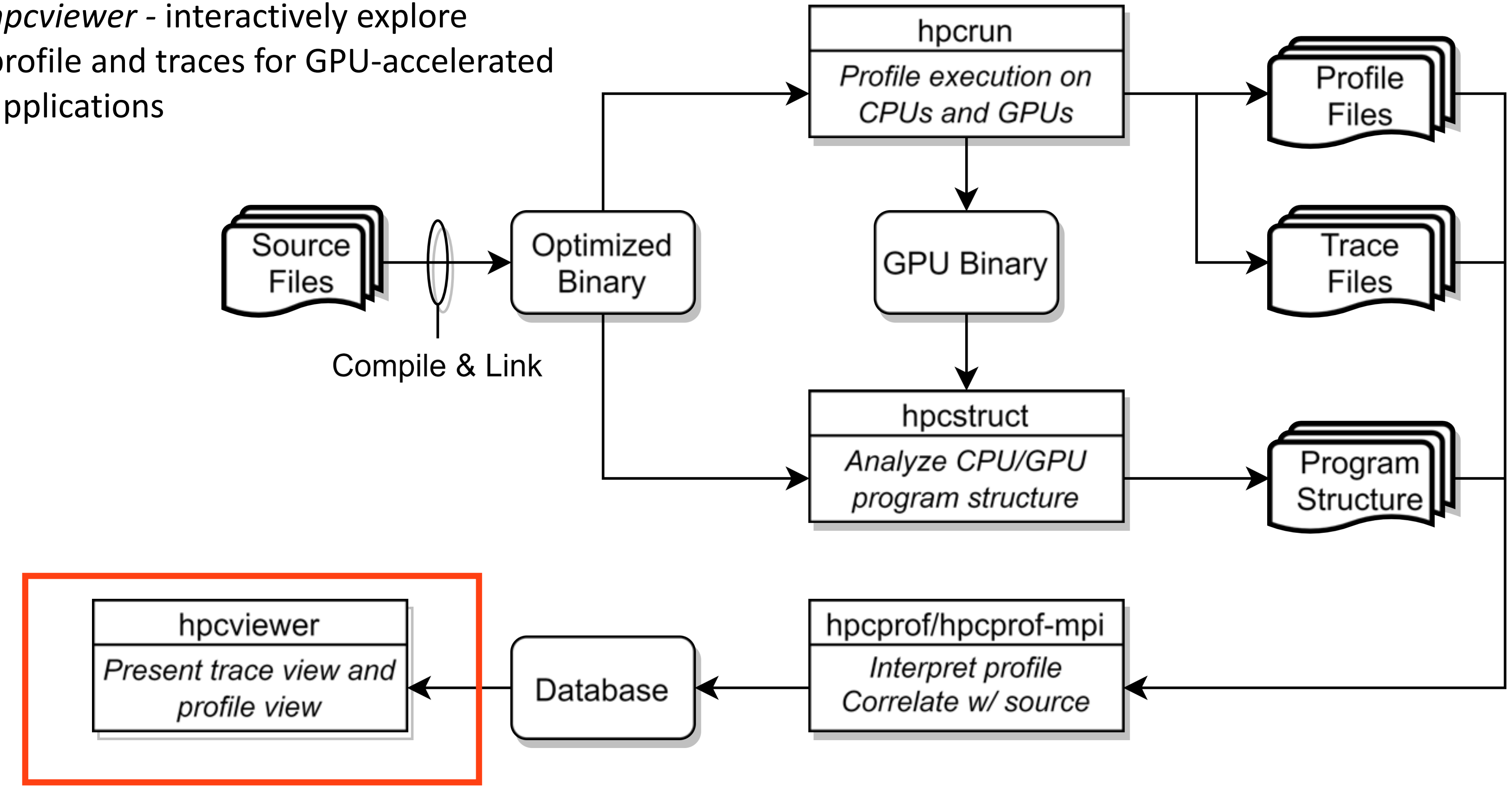
Analyze data from large executions using both distributed-memory and shared-memory parallelism

```
jsrun -n 32 -a 1 hpcprof-mpi <measurement-directory>
srun -n 32 hpcprof-mpi <measurement-directory>
aprun -n 128 -N 8 hpcprof-mpi <measurement-directory>
```

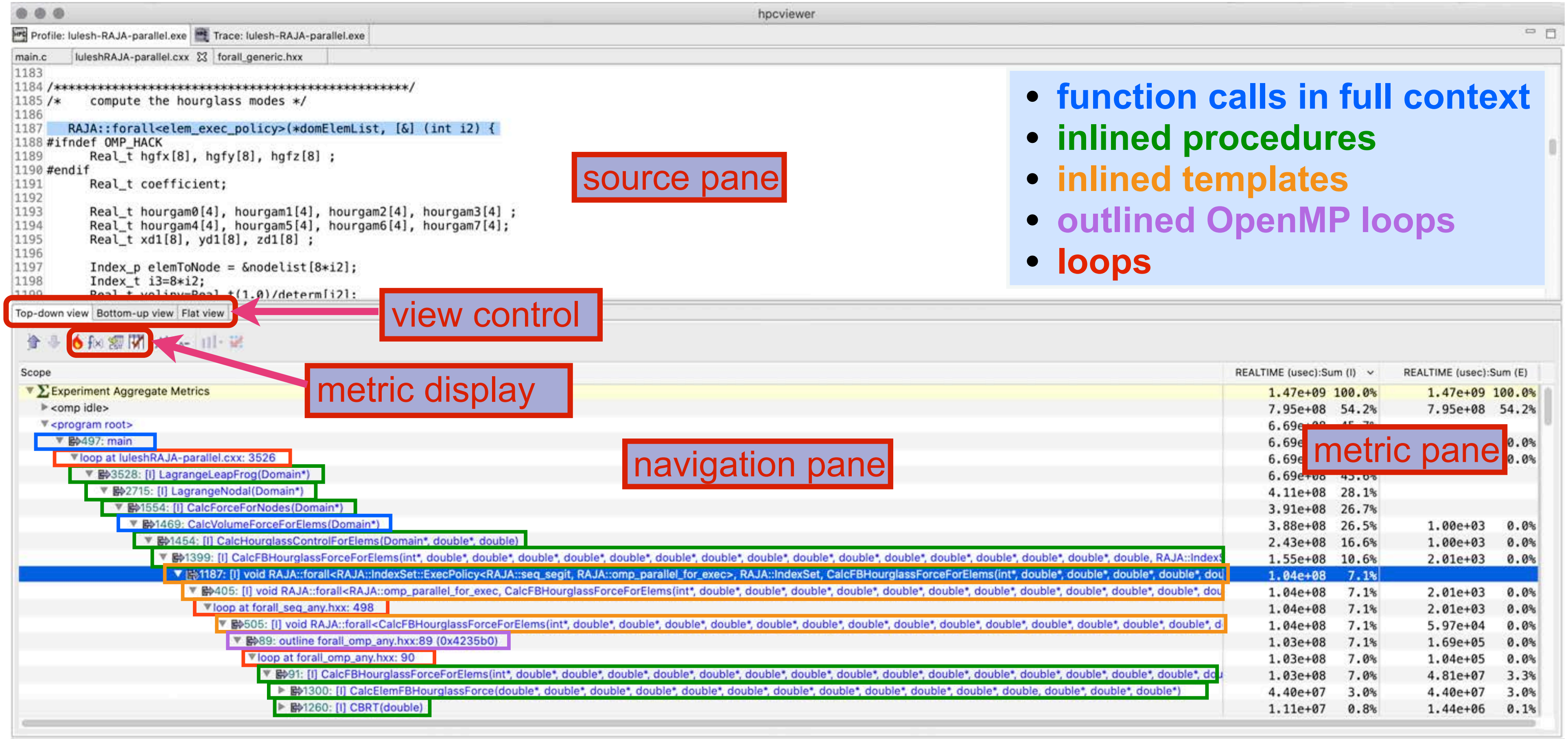
HPCToolkit's Workflow for GPU-accelerated Applications

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

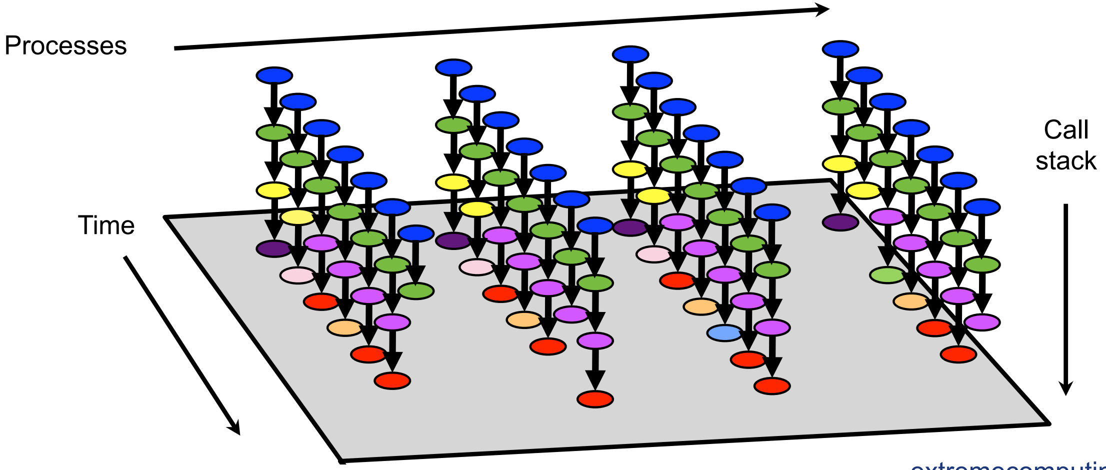
Understanding Temporal Behavior

Profiling compresses out the temporal dimension

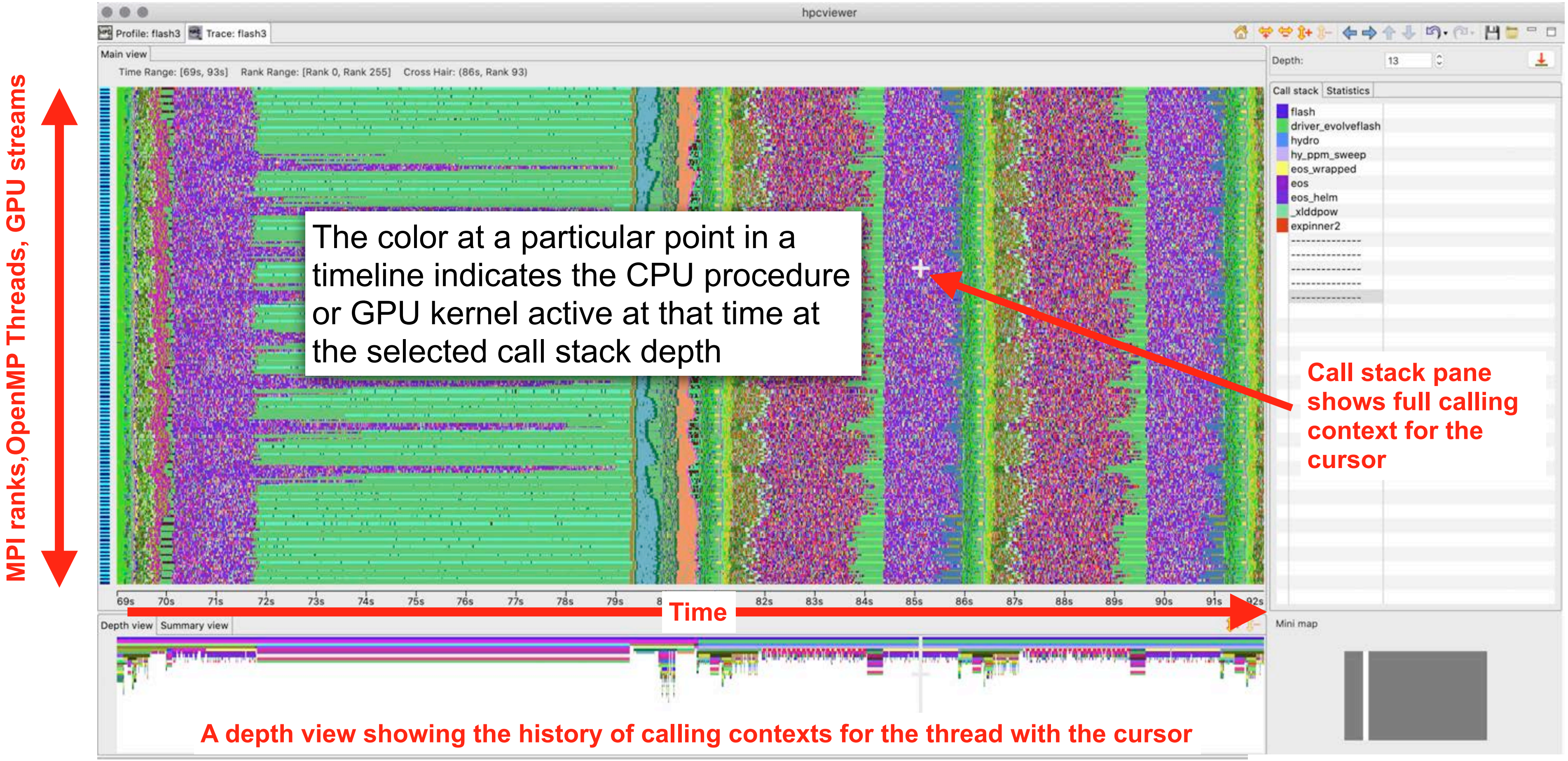
- Temporal patterns, e.g. serial sections and dynamic load imbalance are invisible in profiles

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

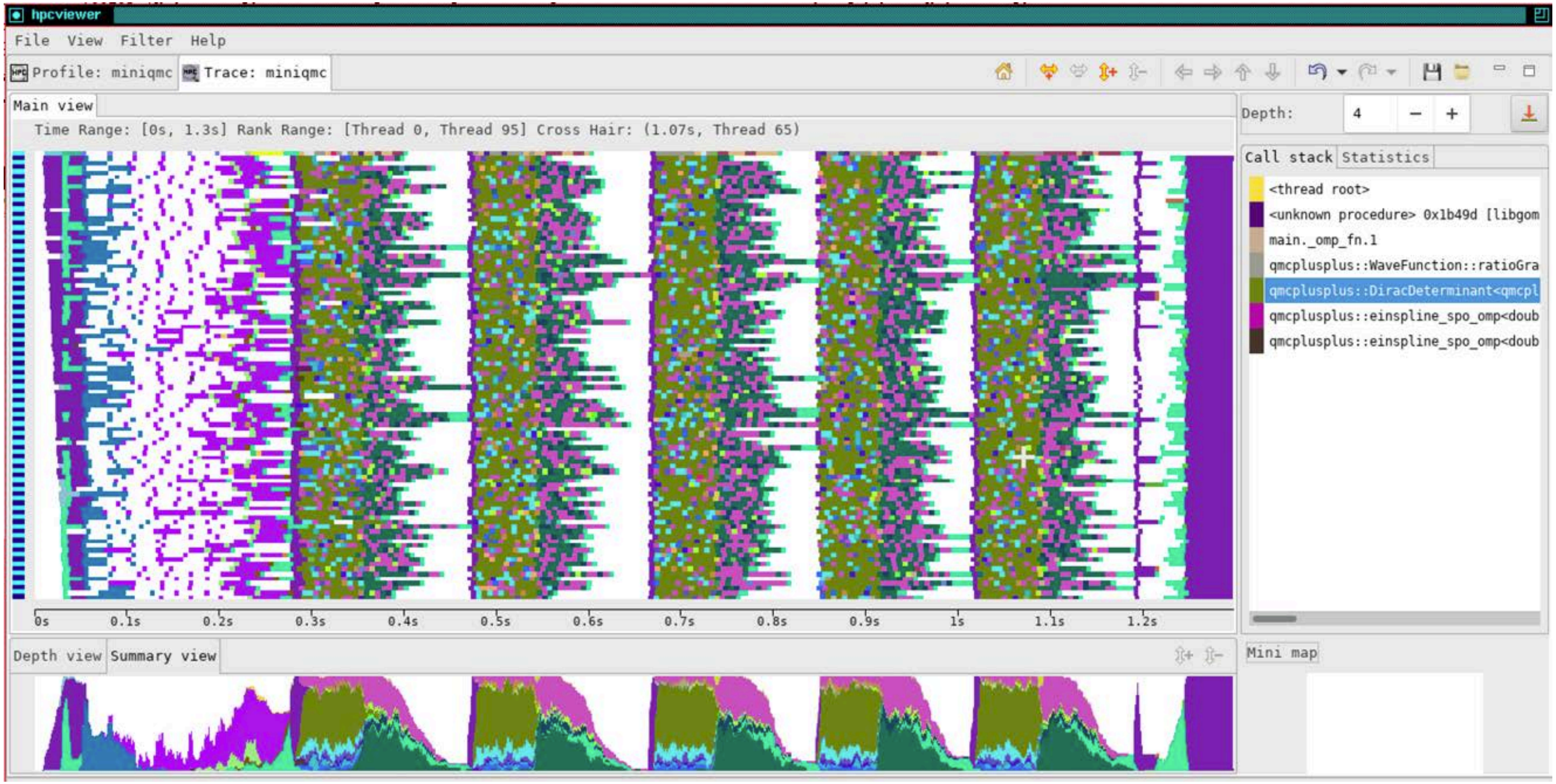


A multi-level call stack based view of execution over time

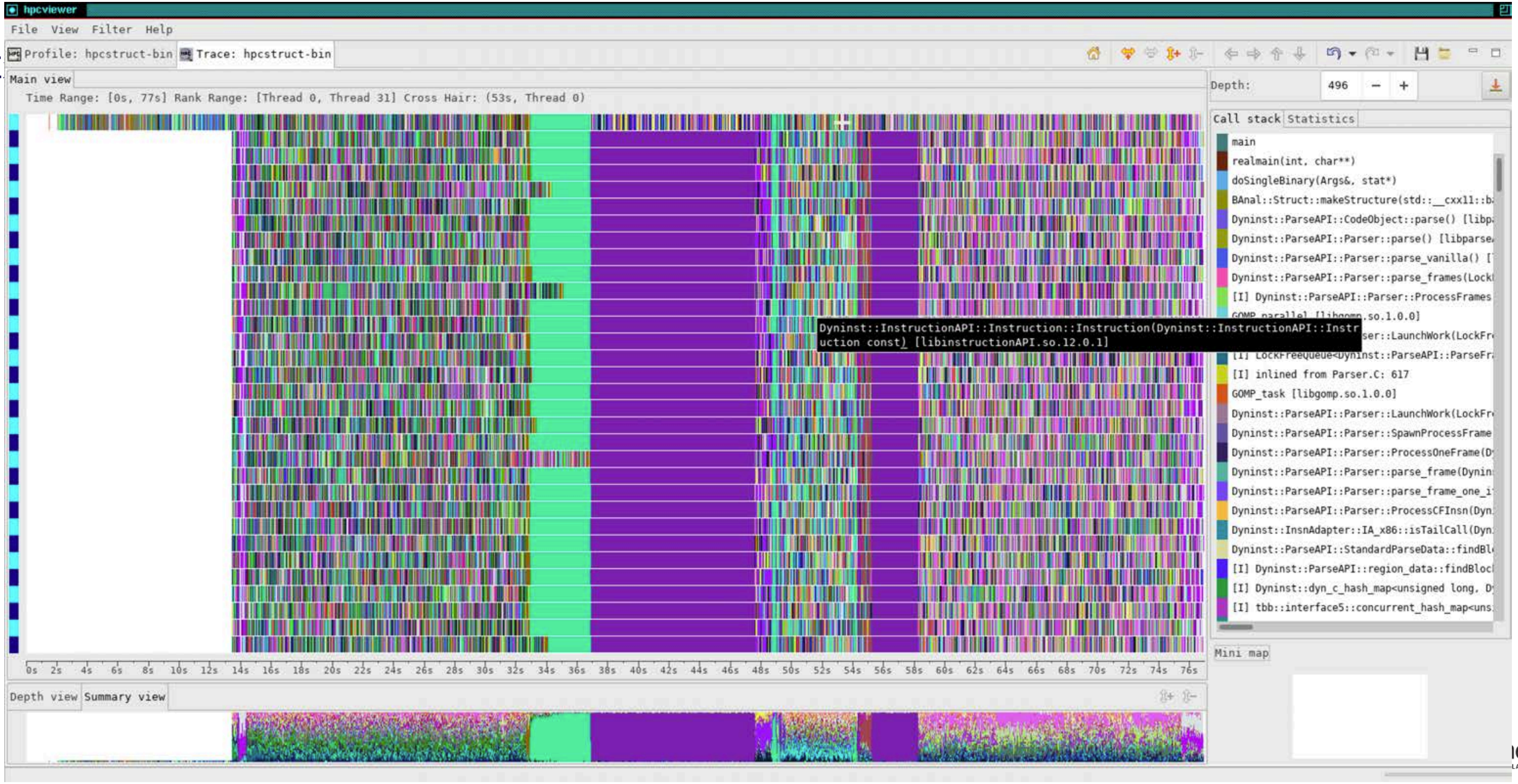
Minimap indicates part of execution trace shown

White Intervals in Traces Indicate Blocking of CPU Threads or GPU Streams

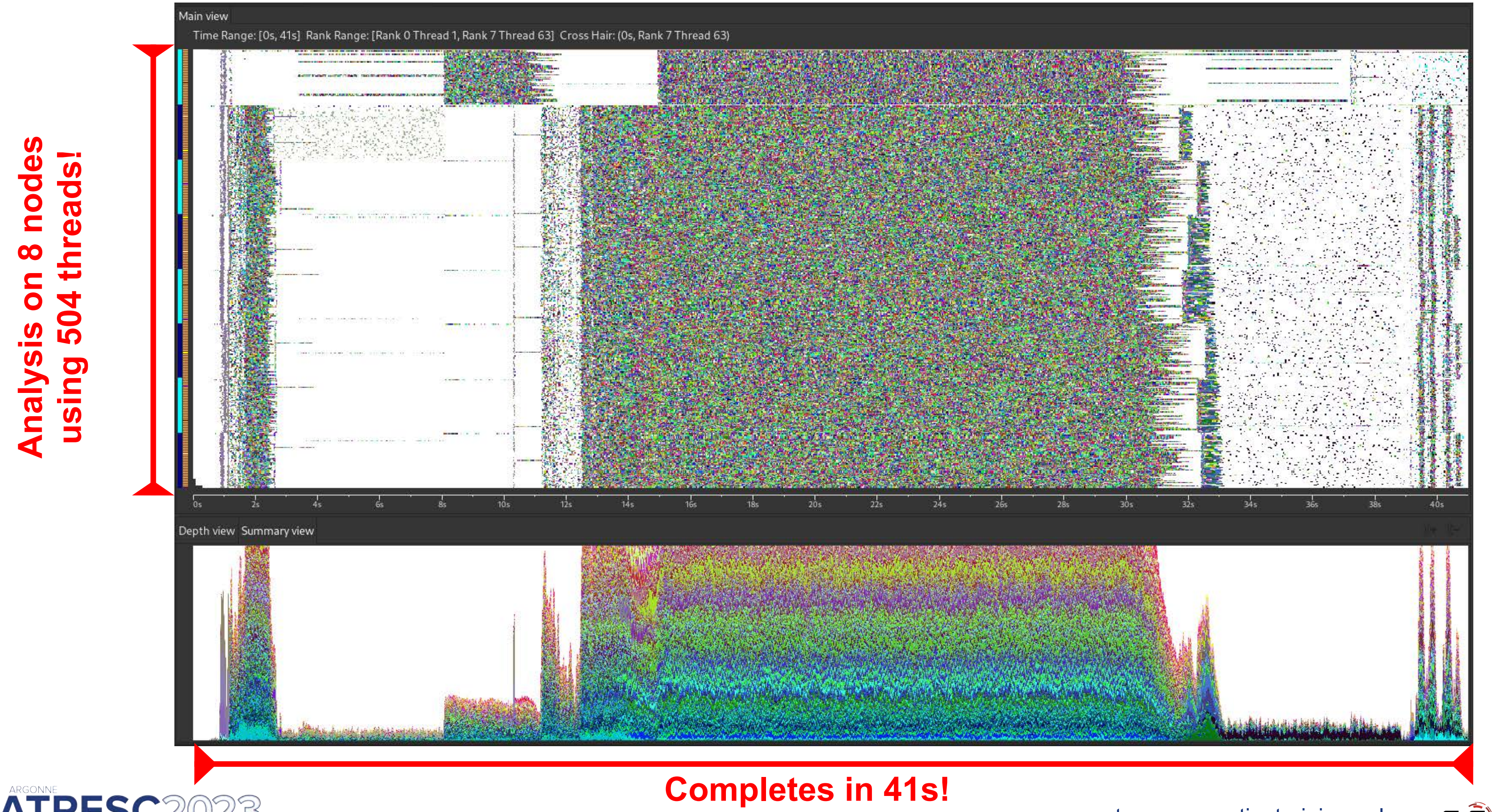
Miniqmc: OpenMP on 32 CPU threads



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



hpcprof-mpi: Analyze Measurements of LAMMPS @ 2K threads + 2K GPUs



Coarse- and Fine-grain Measurement on NVIDIA GPUs: LLNL's Quicksilver

Compute Node

- -2xPower9 + 4xNIVIDIA GPUs

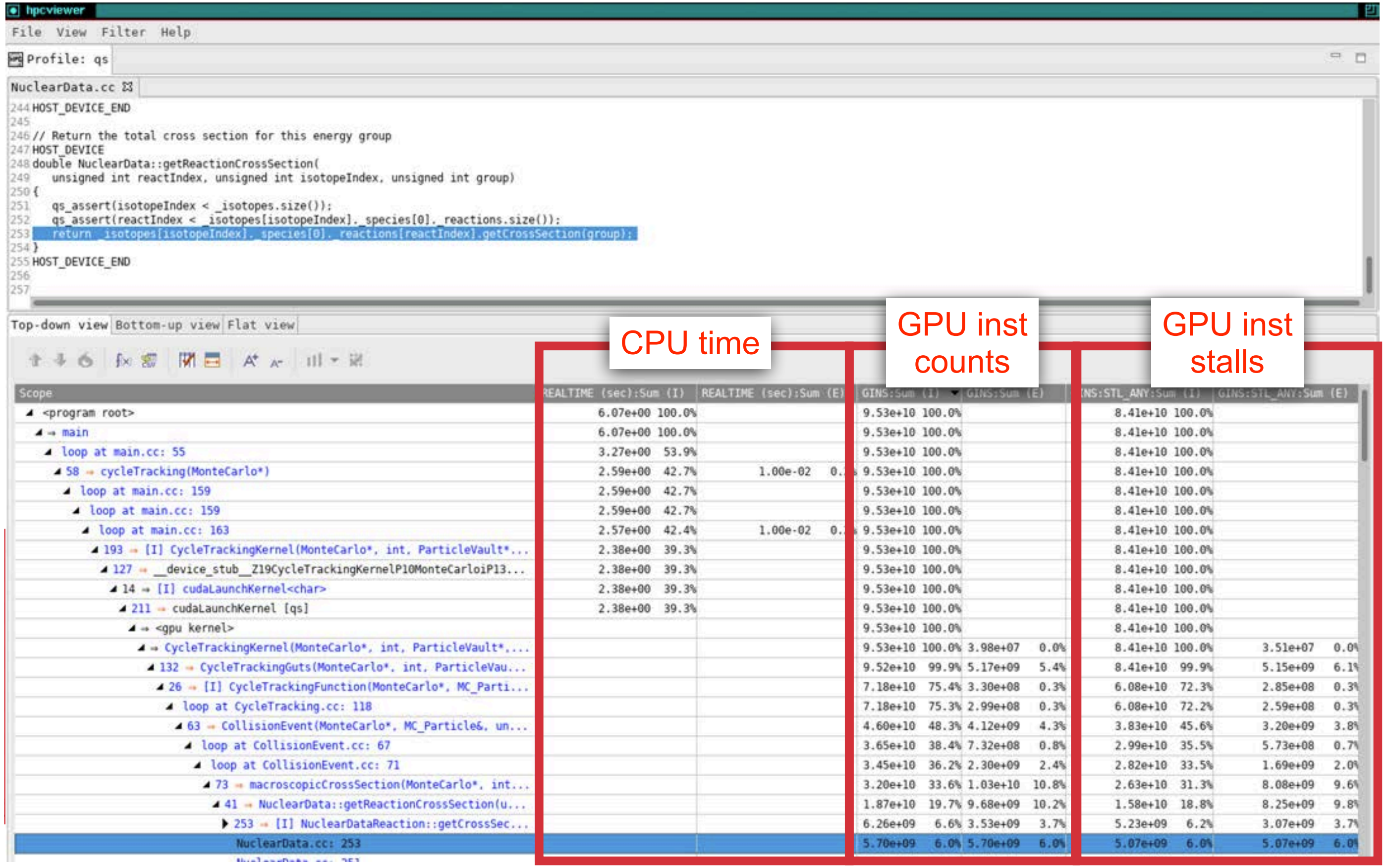
Optimized (-O2) compilation with nvcc

Detailed measurement and attribution using PC sampling

Attribute information to heterogeneous calling context

Key Metrics

- -instructions executed
- -instruction stalls and reasons
- -GPU utilization



K. Zhou, M. W. Krentel, and J. Mellor-Crummey. Tools for top-down performance analysis of GPU-accelerated applications. International Conference on Supercomputing. ACM, NY, NY, USA, June, 2020.

Analysis of PeleC using PC Sampling on an NVIDIA GPU

hpcviewer

Profile: PeleC3d.gnu.TPROF.CUDA.ex

reactor.cpp | Metric properties

```
438   UserData udata = static_cast<ARKODEUserData*>(user_data);
439   udata->dt_save = t;
440
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->rhoe_init_d, udata->rhoesrc_ext_d, udata->rYsrc_d);
452       }
453     });
454 #else
455   for (int icell = 0; icell < udata->ncells_d; icell++) {
456     fKernelSpec(
```

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

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

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%	
2062: amrex::Amr::timeStep(int, double, int, int, double)	1.24e+13 88.6%		1.05e+13 88.7%		5.58e+12 89.3%	
2015: PeleC::advance(double, double, int, int)	1.24e+13 88.5%		1.05e+13 88.6%		5.57e+12 89.2%	
36: PeleC::do_sdc_advance(double, double, int, int)	1.24e+13 88.5%		1.05e+13 88.6%		5.57e+12 89.2%	
loop at Advance.cpp: 302	1.24e+13 88.4%		1.05e+13 88.5%		5.57e+12 89.1%	
308: PeleC::do_sdc_iteration(double, double, int, int, int, int)	1.24e+13 88.4%		1.05e+13 88.5%		5.57e+12 89.1%	
561: PeleC::react_state(double, double, bool, amrex::MultiFab*)	9.61e+12 68.5%		8.29e+12 70.0%		4.17e+12 66.8%	
loop at React.cpp: 109	9.43e+12 67.2%		8.14e+12 68.7%		4.06e+12 65.0%	
210: react(amrex::Box const&, amrex::Array4<double> const&, amrex::Array4<double> cons...	9.39e+12 66.9%		8.10e+12 68.4%		4.03e+12 64.5%	
234: arkEvolve [libsundials_arkode.so.4.7.0]	9.28e+12 66.2%		8.00e+12 67.6%		3.94e+12 63.1%	
erkStep_TakeStep [libsundials_arkode.so.4.7.0]	7.16e+12 51.1%		6.19e+12 52.3%		3.05e+12 48.9%	
cF_RHS(double, _generic_N_Vector*, _generic_N_Vector*, void*)	6.27e+12 44.7%		5.49e+12 46.3%		2.48e+12 39.7%	
443: [I] amrex::launch_global<__nv_dl_wrapper_t<__nv_dl_tag<int (*)(double, _generic_N...	6.27e+12 44.7%		5.49e+12 46.3%		2.48e+12 39.7%	
12: [I] __wrapper__device_stub_launch_global<__nv_dl_wrapper_t<__nv_dl_tag<int (*)(do...	6.27e+12 44.7%		5.49e+12 46.3%		2.48e+12 39.7%	
26: [I] __device_stub__ZN5amrex13launch_globalIZ6cF_RHSdP17_generic_N_VectorS2...	6.27e+12 44.7%		5.49e+12 46.3%		2.48e+12 39.7%	
24: [I] cudaLaunchKernel<char>	6.27e+12 44.7%		5.49e+12 46.3%		2.48e+12 39.7%	
211: cudaLaunchKernel [PeleC3d.gnu.TPROF.CUDA.ex]	6.27e+12 44.7%		5.49e+12 46.3%		2.48e+12 39.7%	
<gpu kernel>	6.27e+12 44.7%		5.49e+12 46.3%		2.48e+12 39.7%	
amrex::launch_global<cF_RHS(double, _generic_N_Vector*, _generic_N_Vector*, v...	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%	
12: [I] cF_RHS(double, _generic_N_Vector*, _generic_N_Vector*, void*)::{lambda...	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: 448	1.11e+12 7.9%	1.11e+12 7.9%	1.11e+12 9.4%	1.11e+12 9.4%	1.10e+12 17.7%	1.10e+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%		

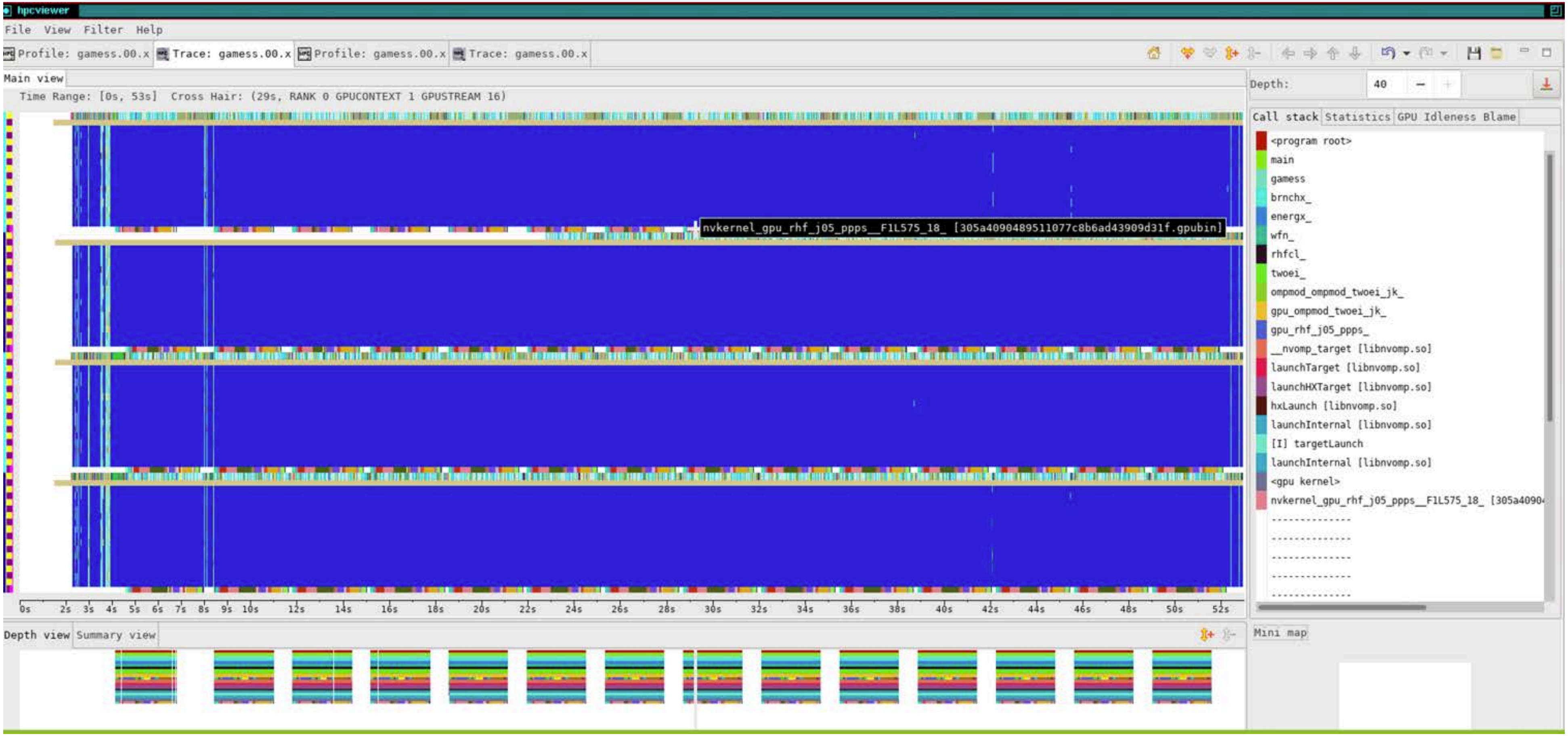
CPU context

GPU context

9.4% GPU stalls outside the loop

mostly memory stalls

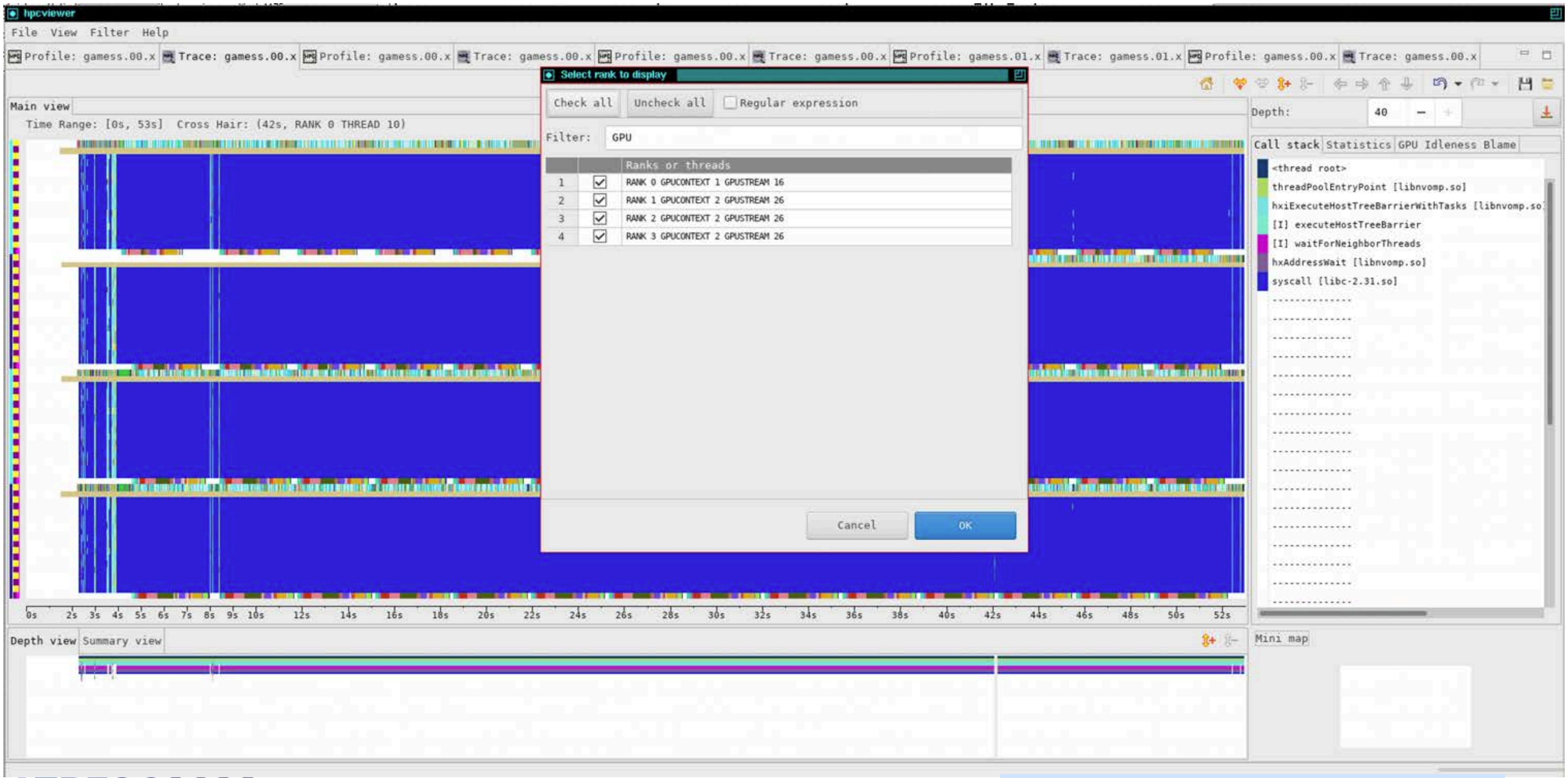
Time-centric Analysis: GAMESS 4 ranks, 4 GPUs on Perlmutter



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

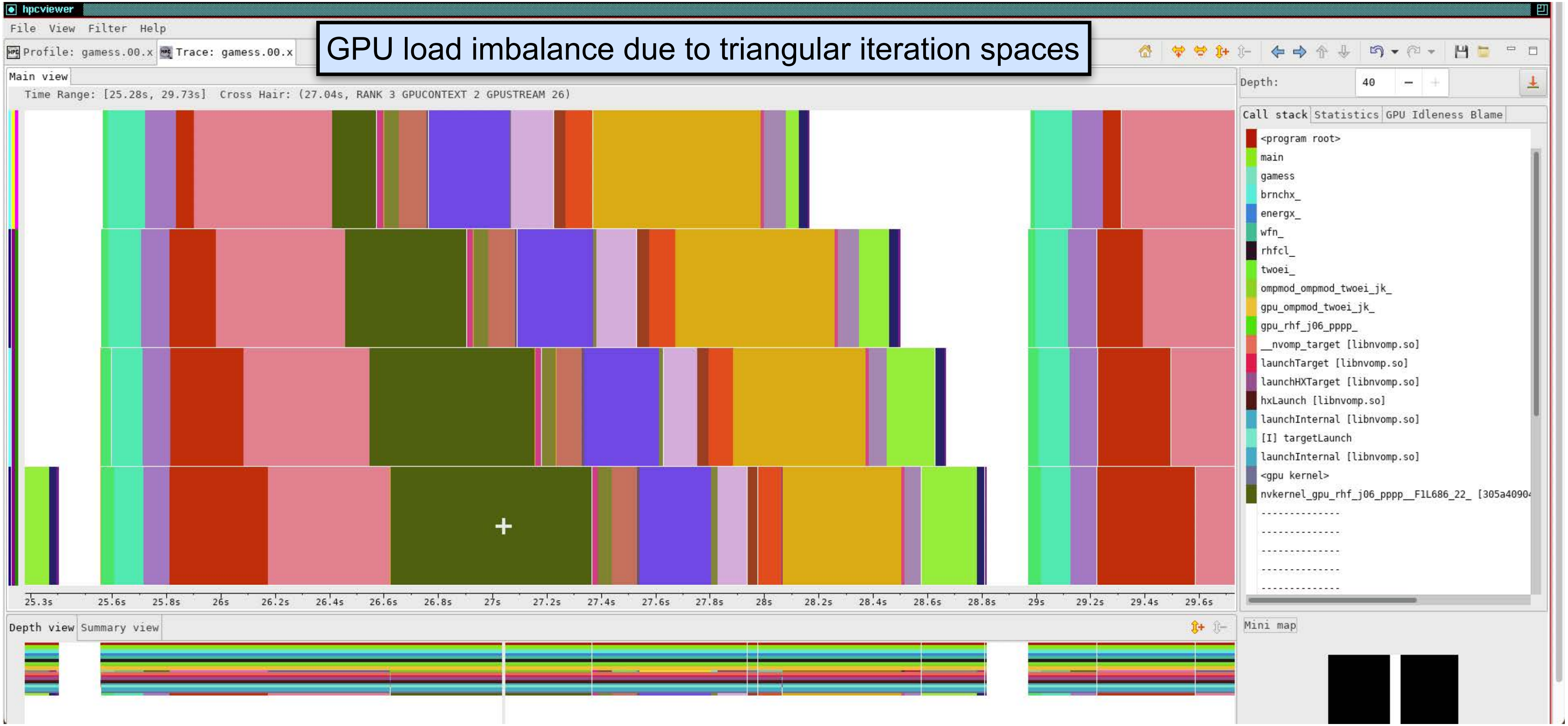
Time-centric Analysis: GAMESS 4 ranks, 4 GPUs on Perlmutter

GAMESS original

All GPU streams, whole execution

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

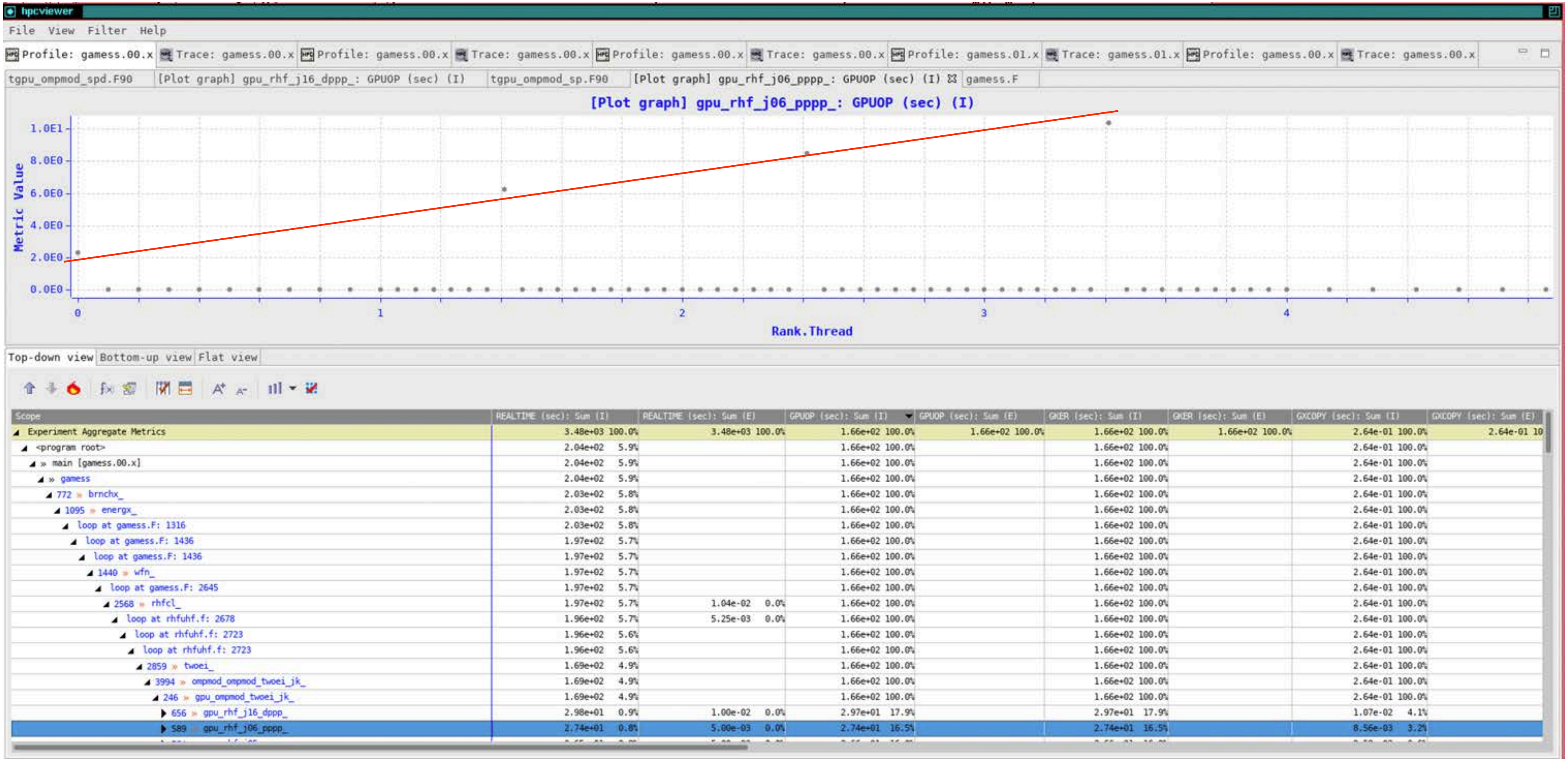
GPU load imbalance due to triangular iteration spaces



GAMESS original

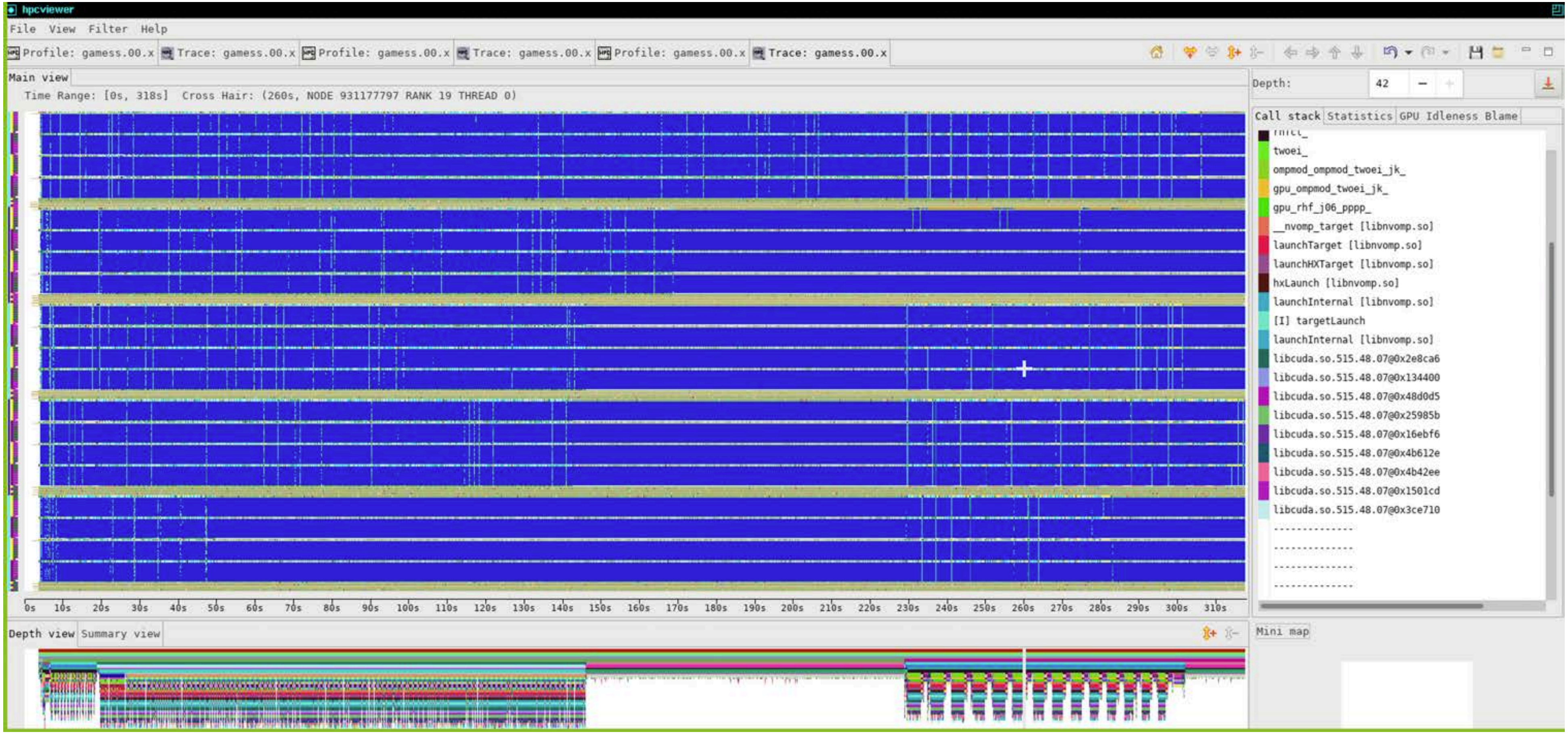
GPU streams: 1 iteration

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



GAMESS original

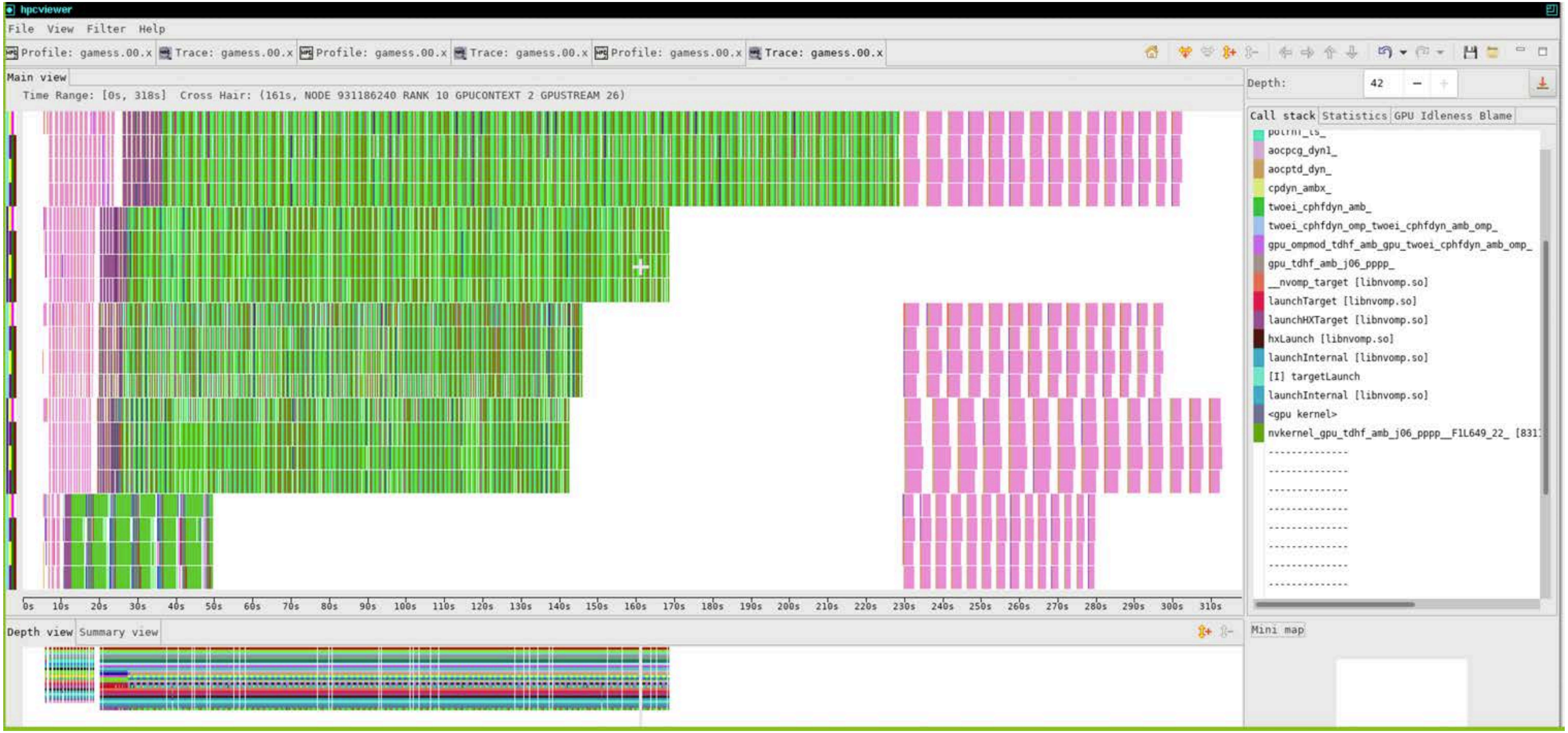
Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter



GAMESS improved

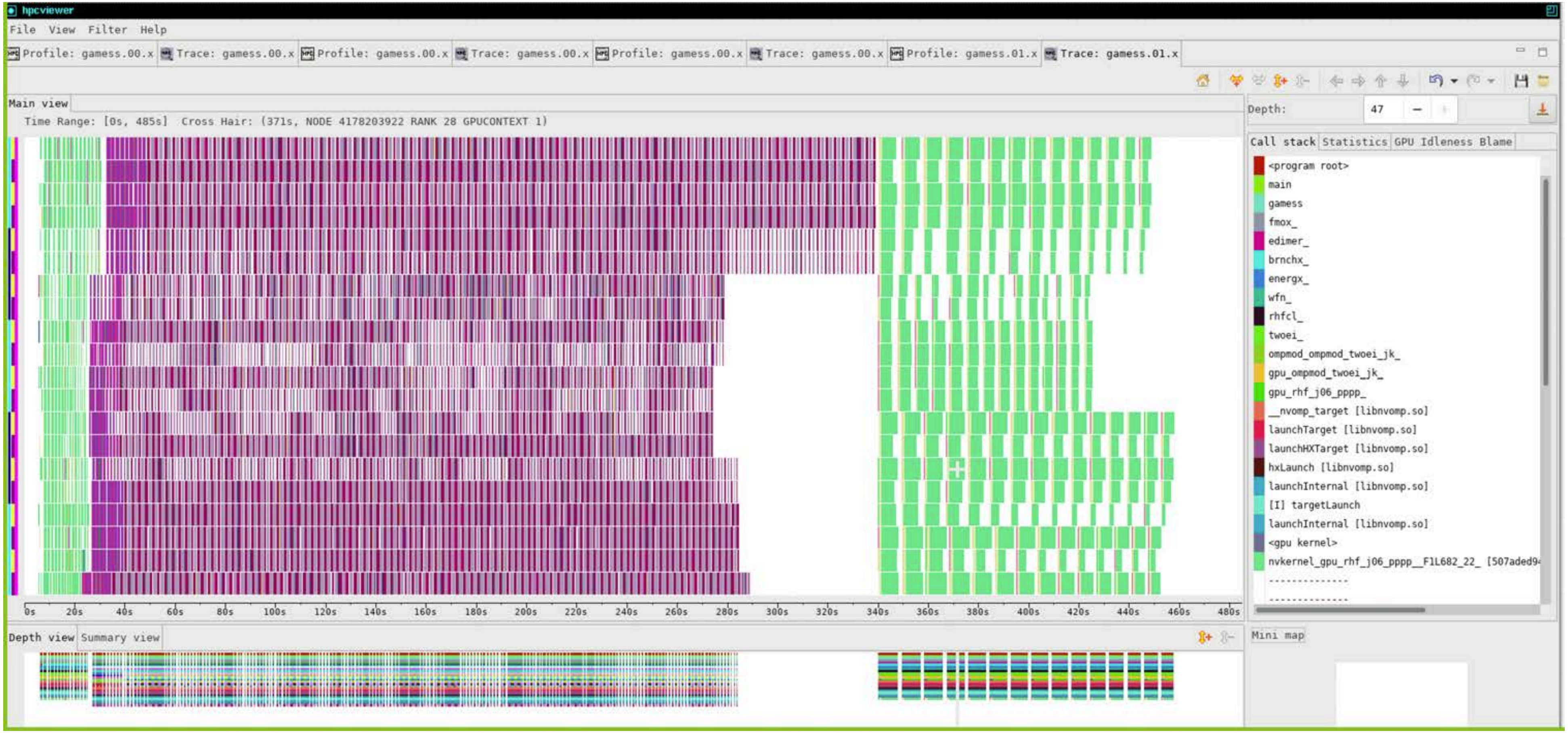
CPU Threads and GPU Streams

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



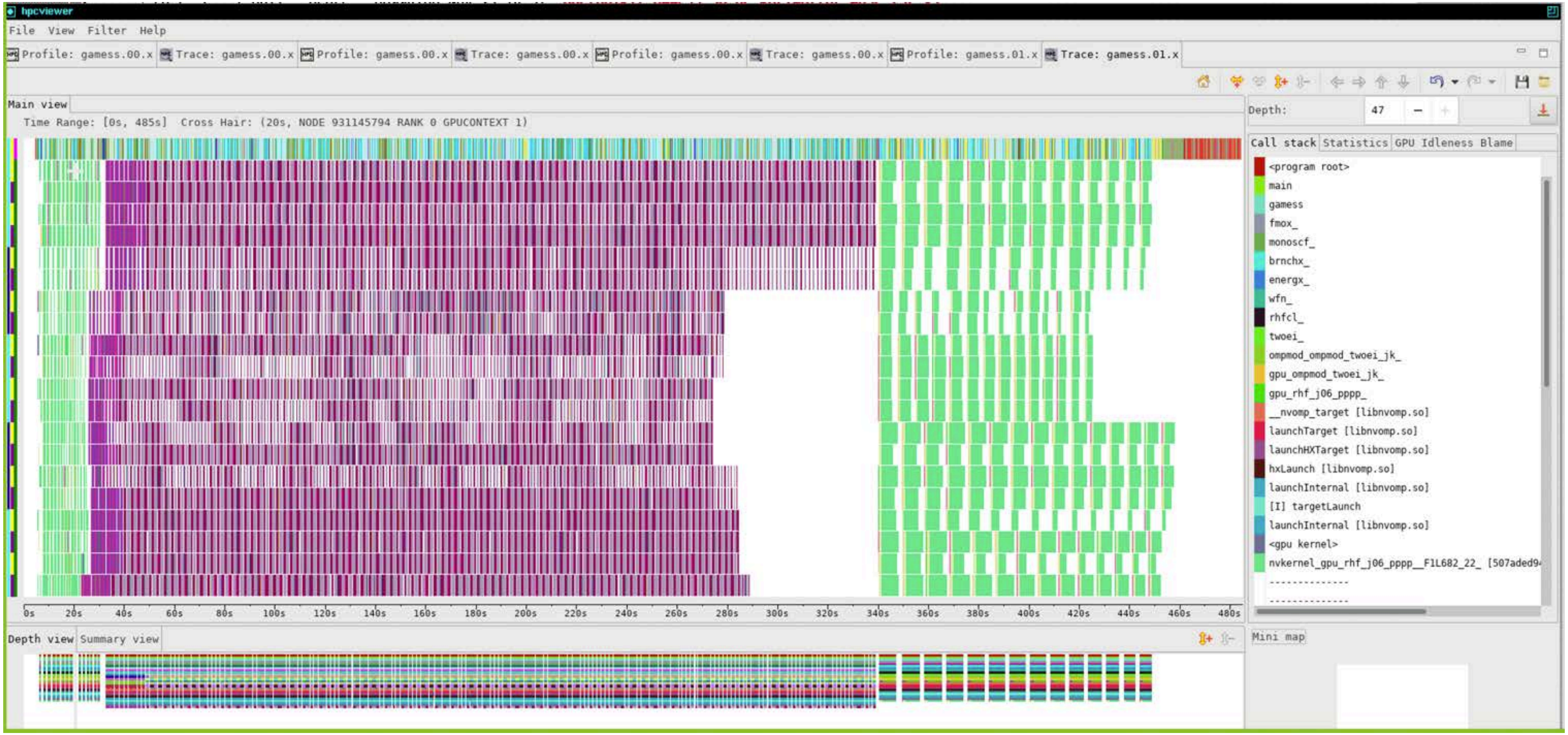
GAMESS improved

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



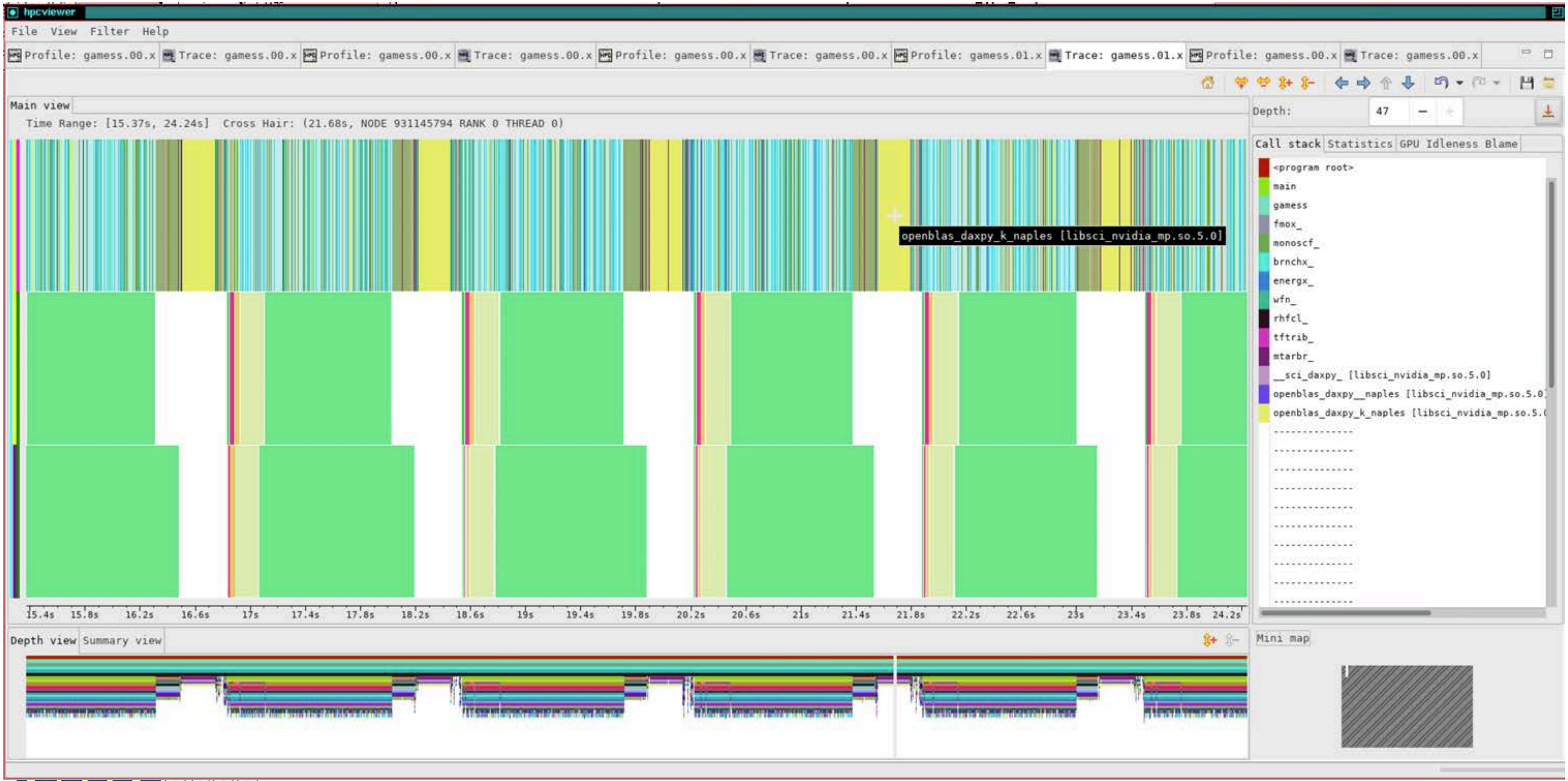
GAMESS improved with better manual distribution of work in input

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

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

```
      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
         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
         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
```

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

mthlib.f

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

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

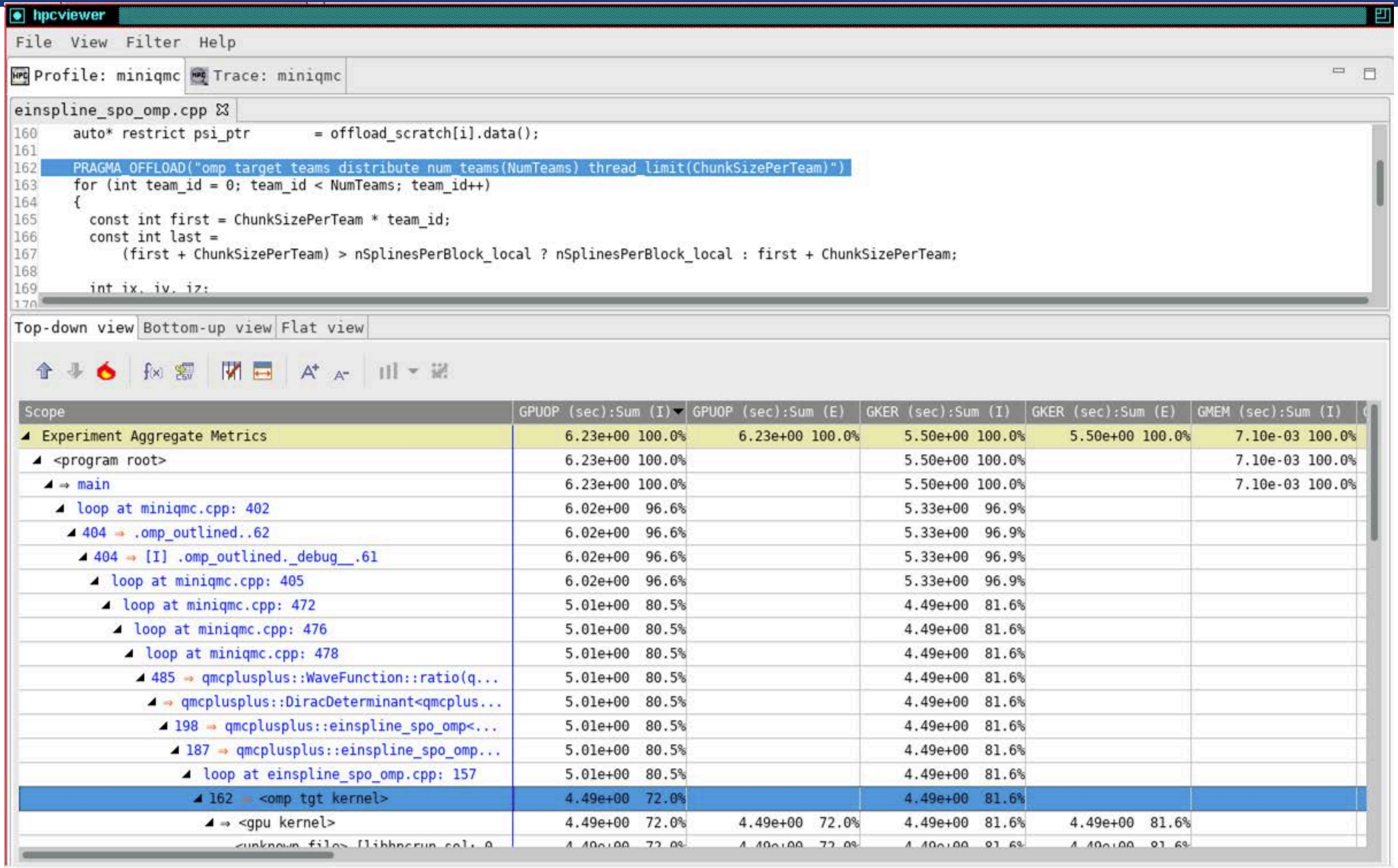
Profile: gamess.00.x | Trace: gamess.00.x | Profile: gamess.00.x | Trace: gamess.00.x | Profile: gamess.00.x | Trace: gamess.00.x | Profile: gamess.01.x | Trace: gamess.01.x

mthlib.f

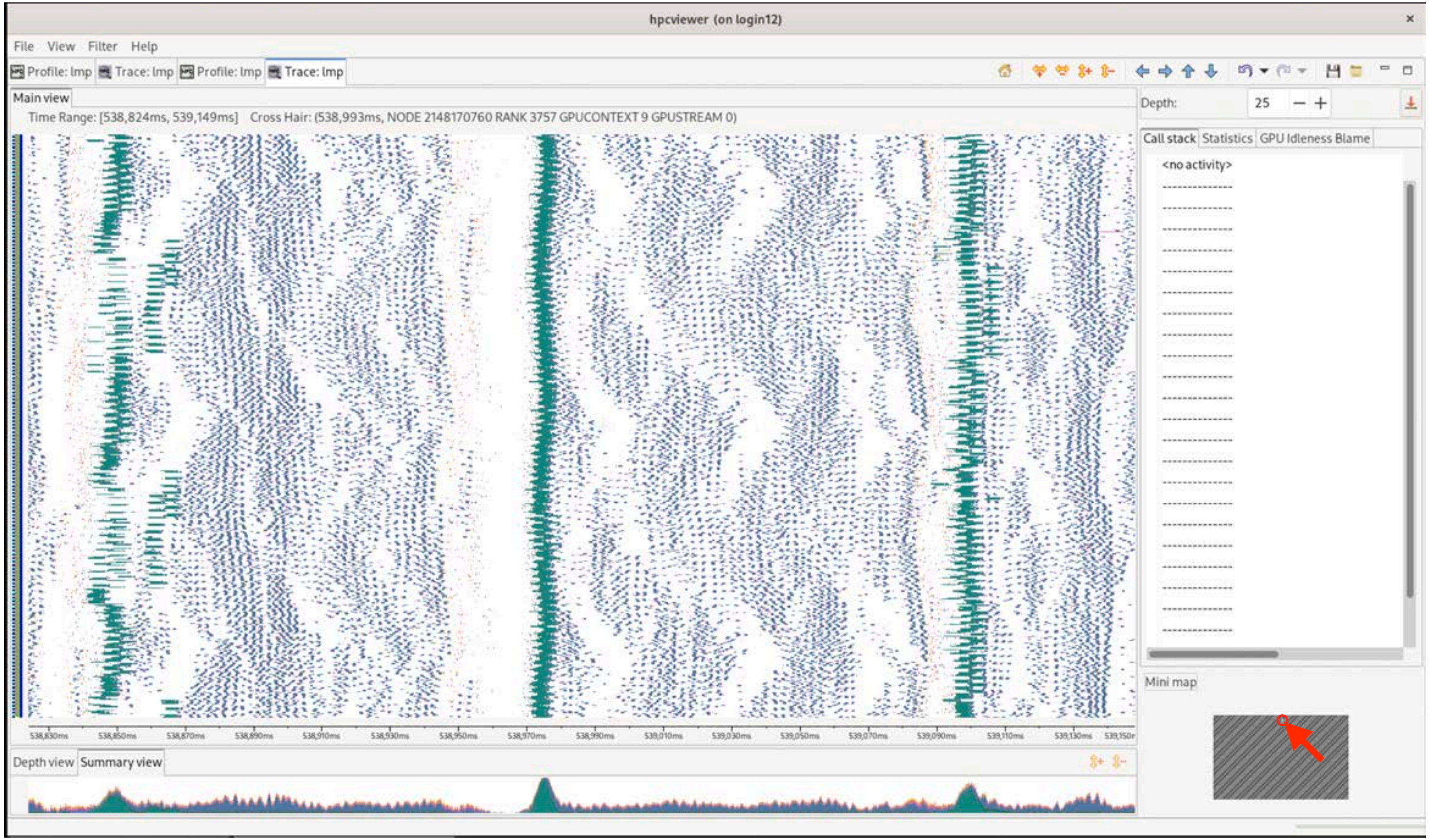
```
C
      IJ=1-INC
      DO 150 I=2,NA
         IJ=IJ+INC
         IM1=I-1
         DO 140 J=1,IM1
            IJ=IJ+INC
            AIJ=A(IJ)
            IF(AIJ.EQ.ZERO) GO TO 140
               CALL DAXPY(MB,AIJ,B(I,1),NA,AB(J,1),NAB)
               CALL DAXPY(MB,AIJ,B(J,1),NA,AB(I,1),NAB)
  140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

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

Measure and Attribute OpenMP Offloading



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



HPCToolkit Status on GPUs

NVIDIA

- heterogeneous profiles
- GPU instruction-level execution and stalls using PC sampling
- traces

AMD

- heterogeneous profiles
- no GPU instruction-level measurements within kernels
- measure OpenMP offloading using OMPT interface
- traces

Intel

- heterogeneous profiles
- GPU instruction-level measurements with instrumentation; heuristic latency attribution to instructions
- measure OpenMP offloading using OMPT interface
- traces

Ongoing Work

- Enhancing measurement to identify root causes of scalability losses
  - Better measurement of delays caused by GPU and communication
- Improving the scalability of hpcprof-mpi
  - Avoid unnecessary serialization of I/O
- Adding a Python-based interface for analysis of performance results
  - Python API supports arbitrary queries and analysis of profiles and traces
  - Automatic analysis to identify notable features in executions
    - e.g. load imbalance, trace line equivalence classes

HPCToolkit Resources

- Documentation
  - User manual
    - http://hpctoolkit.org/manual/HPCToolkit-users-manual.pdf
  - Tutorial videos
    - http://hpctoolkit.org/training.html
- 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

ARGONNE
ATPESC2023
EXTREME-SCALE COMPUTING

HPCToolkit Hands-On Directions

Performance analysis of CPU and GPU-accelerated applications at Scale

John Mellor-Crummey
Professor, Rice University

Sample Performance Databases for You to Analyze on Polaris

- Setup on polaris
  - module use /soft/perftools/hpctoolkit/polaris/modulefiles
  - module load hpctoolkit/default
- Data on theta and polaris: /grand/ATPESC2023/track-6-tools-hpctoolkit/data
  - CPU
    - QMCPACK - quantum Monte Carlo electronic structure calculations (early experiment)
  - GPU-accelerated
    - GAMESS - ab initio quantum chemistry
      - 1.singlegroup-unbalanced
      - 2.singlegroup-balanced
      - 3.multigroup-unbalanced-mtarbr
      - 4.multigroup-balanced
      - 5.multigroup-unbalanced-pc
      - 6.scale
    - PeleC - AMR Solver (AMREx) for compressible reacting flows
    - Pytorch-deepwave - GPU-accelerated reverse-time migration using Pytorch
    - Quicksilver - proxy application for dynamic Monte Carlo transport

GPU: Profiling Quicksilver with HPCToolkit on Polaris or Perlmutter

- git clone https://github.com/hpctoolkit/hpctoolkit-tutorial-examples
- cd hpctoolkit-tutorial-examples/examples/gpu/quicksilver
- polaris:
  - export HPCTOOLKIT_TUTORIAL_PROJECTID=ATPESC2023
  - export HPCTOOLKIT_TUTORIAL_RESERVATION=default
  - source setup-env/polaris.sh
- perlmutter:
  - export HPCTOOLKIT_TUTORIAL_PROJECTID=ntrain5_g
  - export HPCTOOLKIT_TUTORIAL_RESERVATION=default
  - source setup-env/perlmutter.sh
- make build
- make run
- make run-pc
- make view
- make view-pc

CPU: Profiling AMG2013 with HPCToolkit on Theta

- `git clone https://github.com/hpctoolkit/hpctoolkit-tutorial-examples`
- `cd hpctoolkit-tutorial-examples/examples/cpu/mpi+openmp/amg2013`
- `export HPCTOOLKIT_TUTORIAL_PROJECTID=ATPESC2023`
- `export HPCTOOLKIT_TUTORIAL_RESERVATION=debug-cache-quad`
- `source setup-env/theta.sh`
- `make build`
- `make run`
  - # wait for $COBALT_JOBID.done to appear in your directory
- `make analyze`
- Alternatives
  - `make view`
  - `hpcviewer hpctoolkit-amg2013.d`