HPCToolkit: Performance Tools for GPU-Accelerated Computing

John Mellor-Crummey
Rice University

December 14, 2022

EXASCALE COMPUTING PROJECT

National Nuclear Security Administration

U.S. DEPARTMENT OF ENERGY | Office of Science

RICE

WISCONSIN UNIVERSITY OF WISCONSIN–MADISON

HPCToolkit Funding Acknowledgments

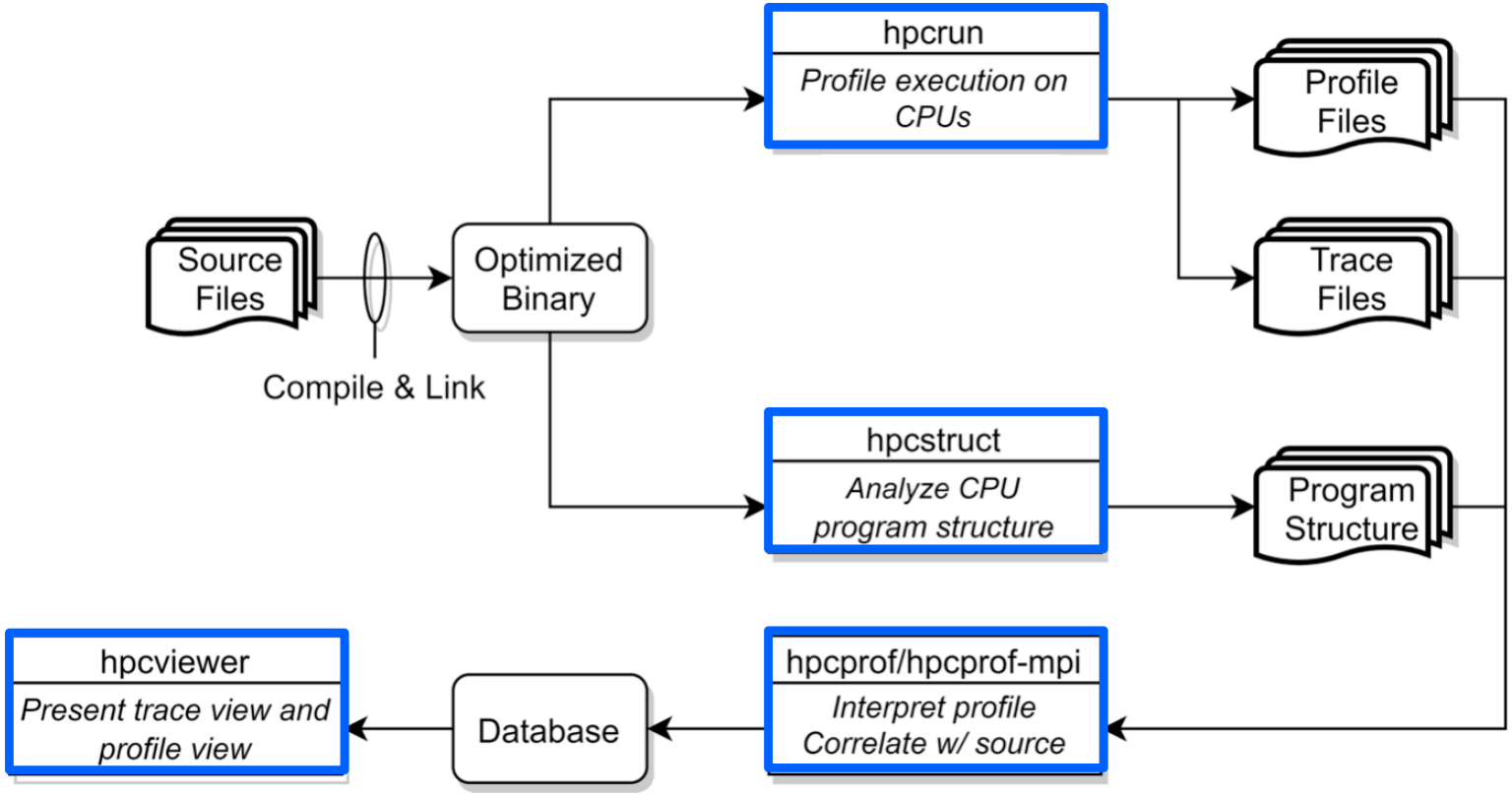
- Government
  - Exascale Computing Project 17-SC-20-SC
  - Lawrence Livermore National Laboratory Subcontract B645220
  - Argonne National Laboratory Subcontract 9F-60073
- Corporate
  - Advanced Micro Devices
  - Intel Corporation
  - TotalEnergies EP Research & Technology USA, LLC.

Rice University's HPCToolkit Performance Tools

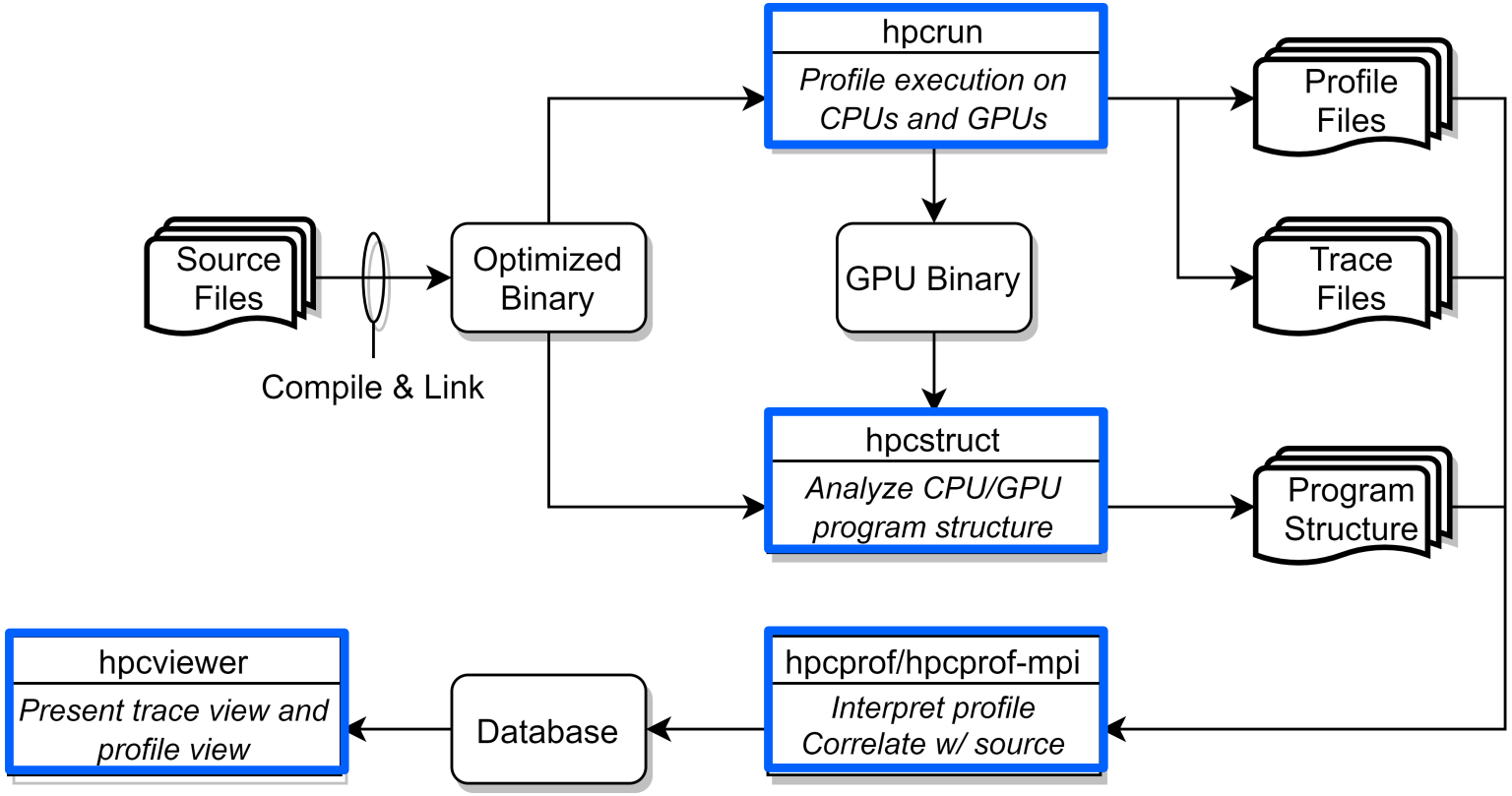
Measure and analyze performance of CPU and GPU-accelerated applications

- 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



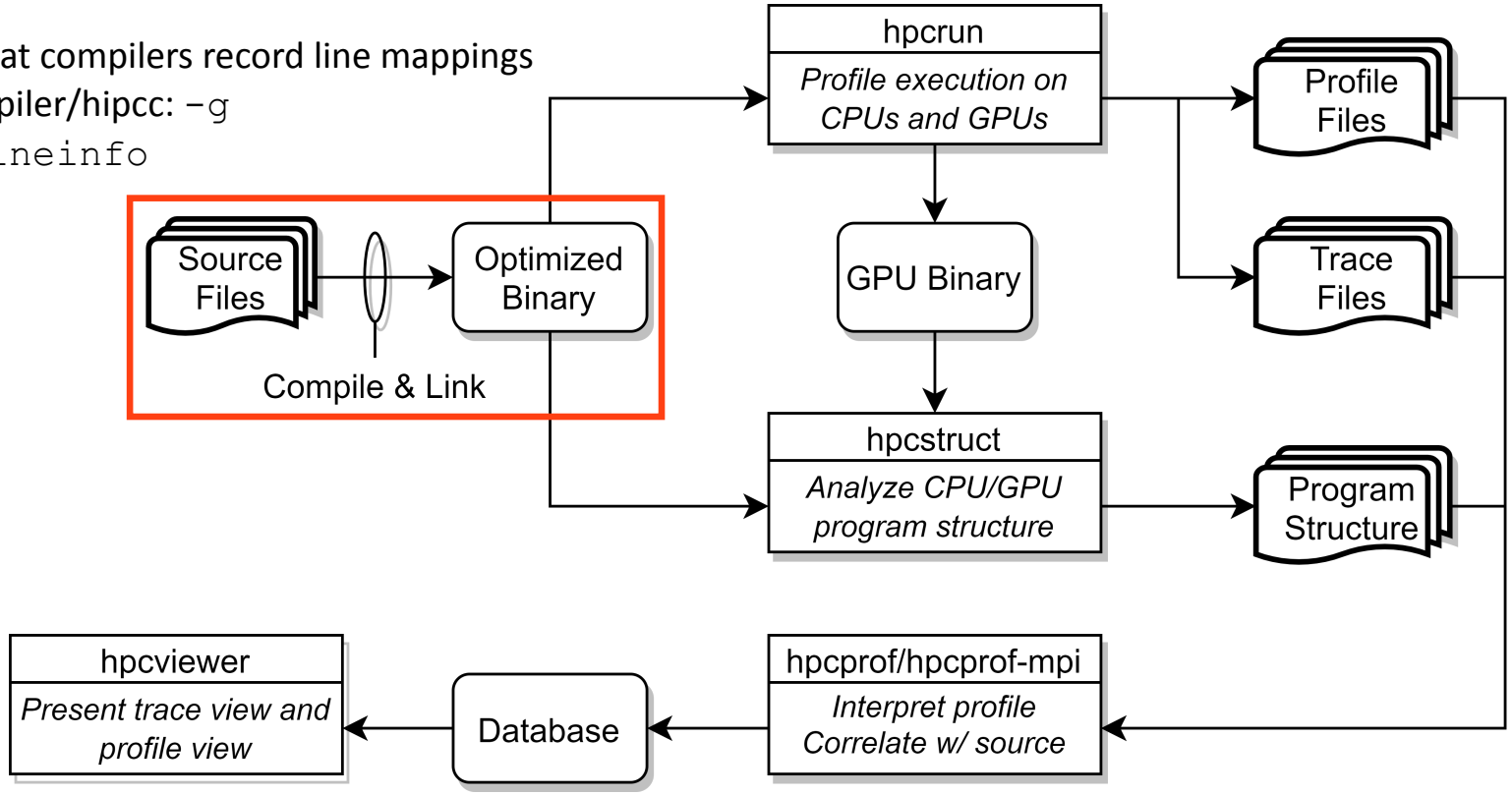
HPCToolkit's Workflow for GPU-accelerated Applications



HPCToolkit's Workflow for GPU-accelerated Applications

Step 1:

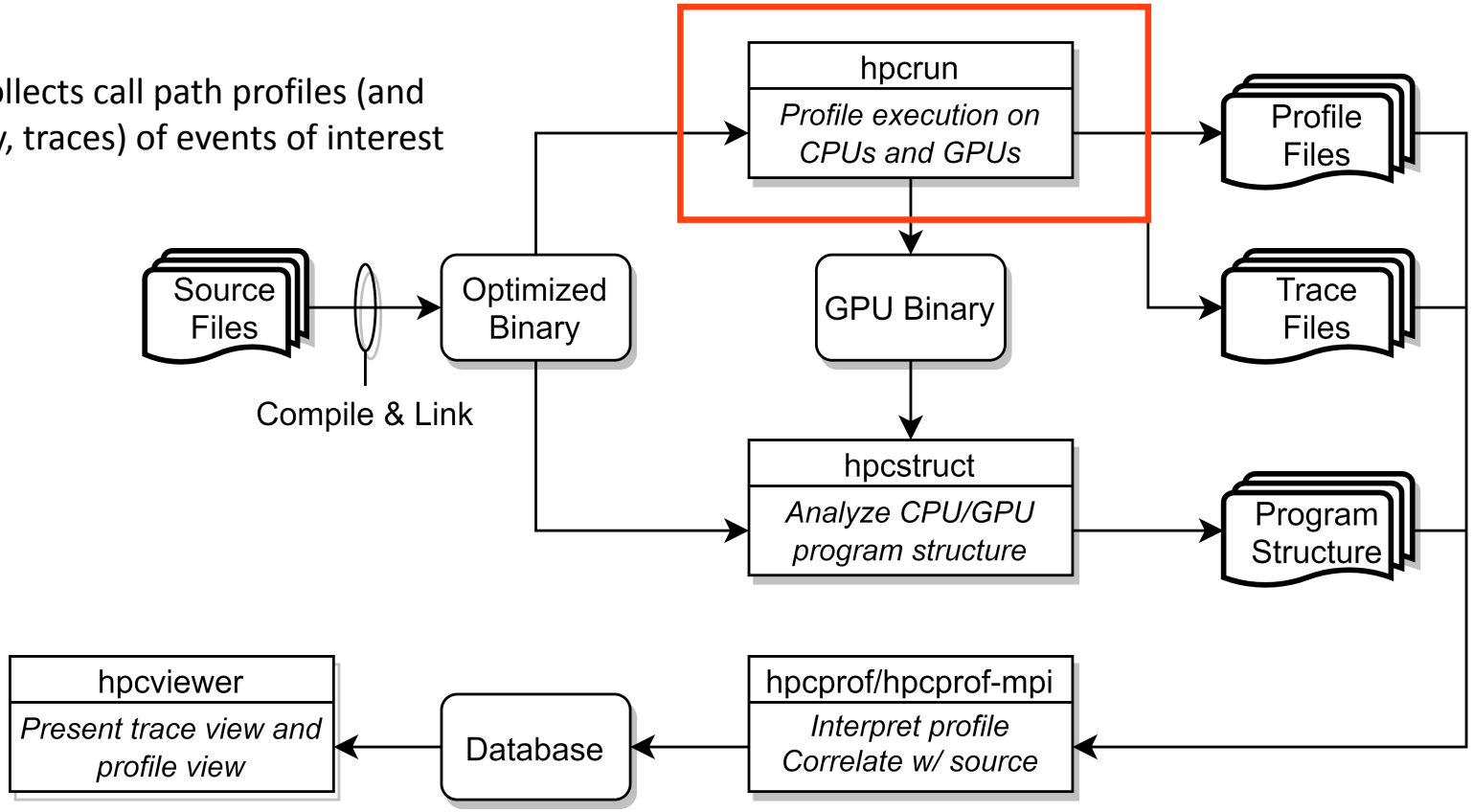
- Ensure that compilers record line mappings
- host compiler/hipcc: `-g`
- nvcc: `-lineinfo`



HPCToolkit's Workflow for GPU-accelerated Applications

Step 2:

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



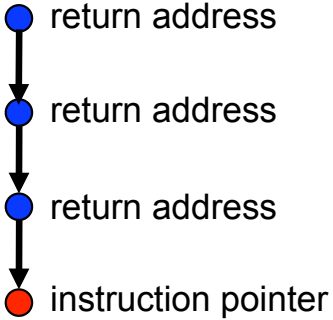
Measurement of CPU and GPU-accelerated Applications

- Sampling using timers and hardware counter overflow on the CPU
- Callbacks when GPU operations are launched and (sometimes) completed
- GPU event stream for GPU operations; PC Samples (NVIDIA)

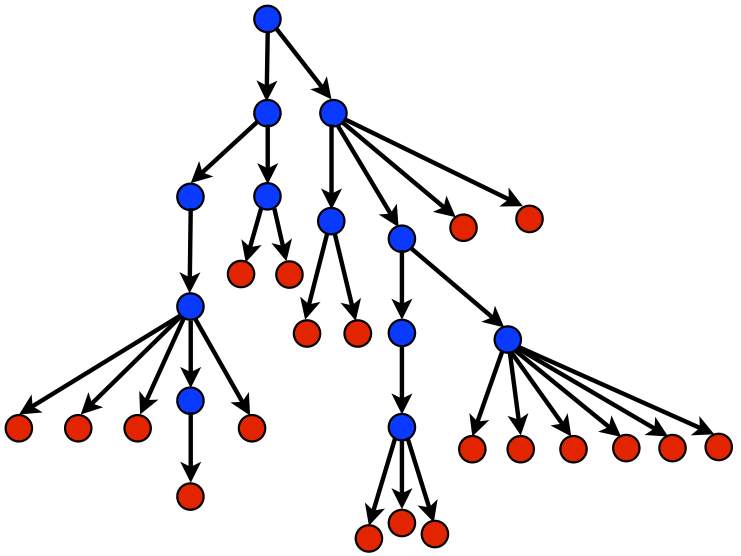
Call Stack Unwinding to Attribute Costs in Context

- 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
  - `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

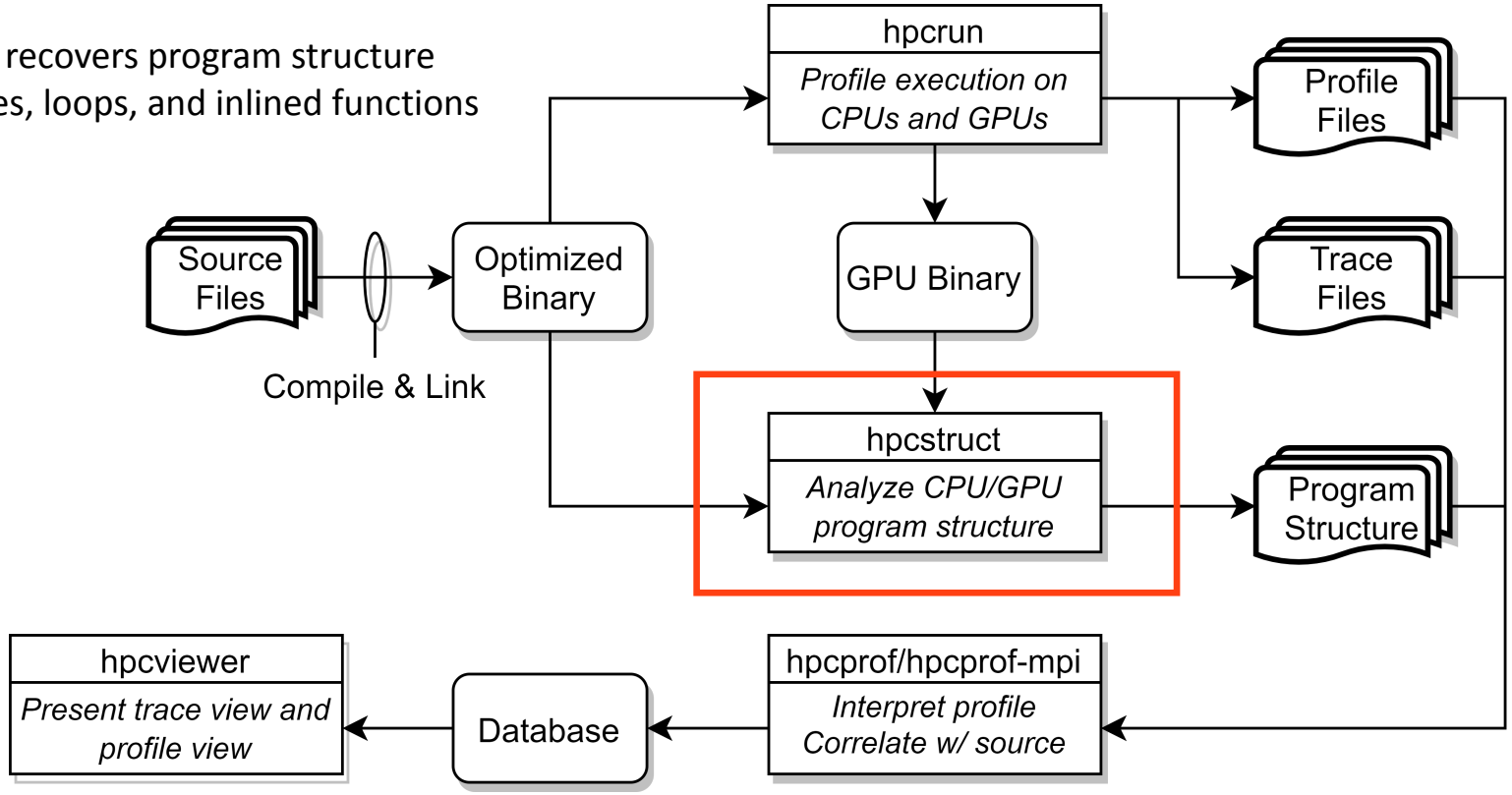
CPU traces
- trace of call stack samples

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>
```

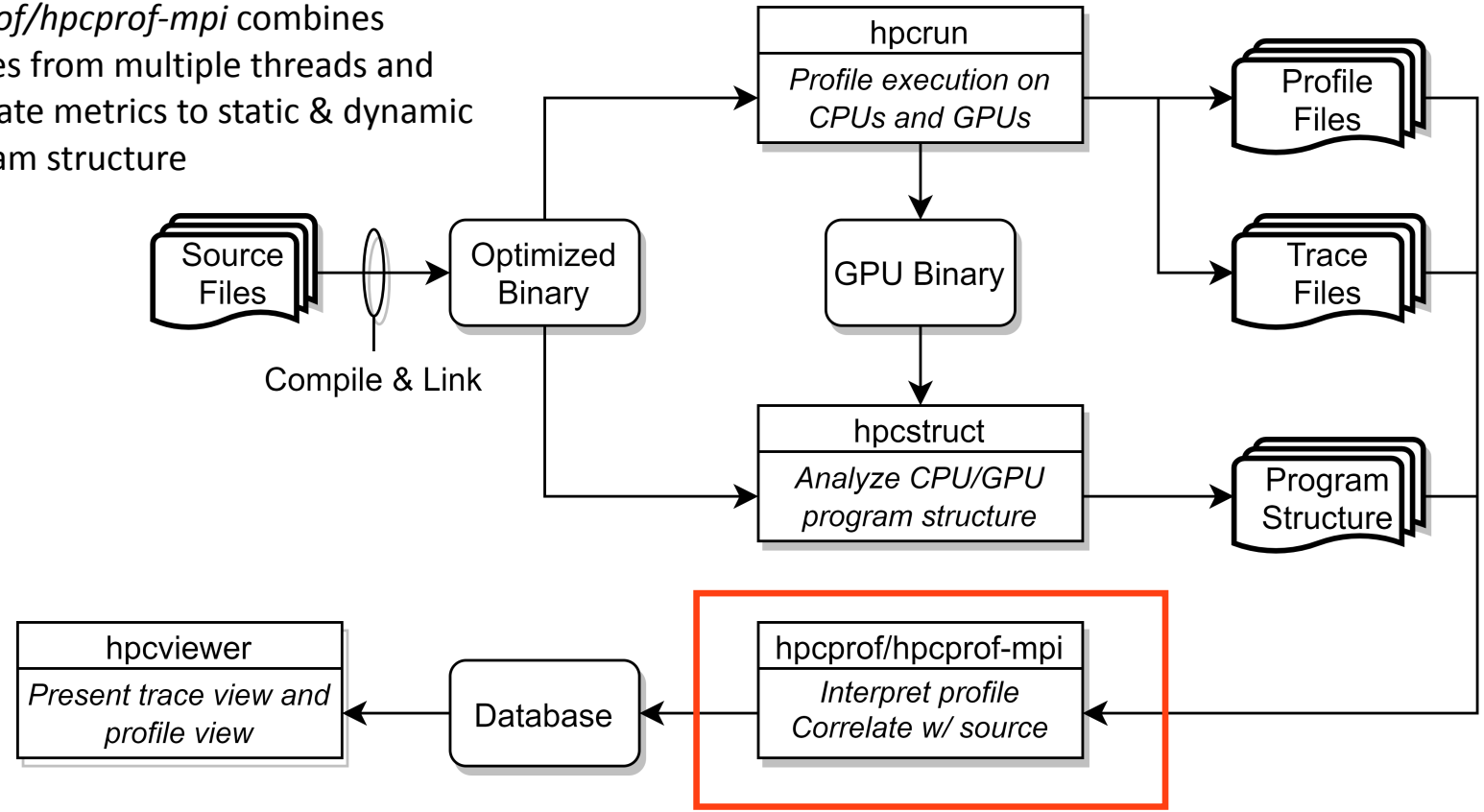
- What it does
  - Recover program structure information
    - Files, functions, inlined templates or functions, loops, source lines
  - In parallel, analyze all CPU and GPU binaries that were measured by HPCToolkit
    - —default: use size(CPU set)/2 threads
    - —analyze large application binaries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

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

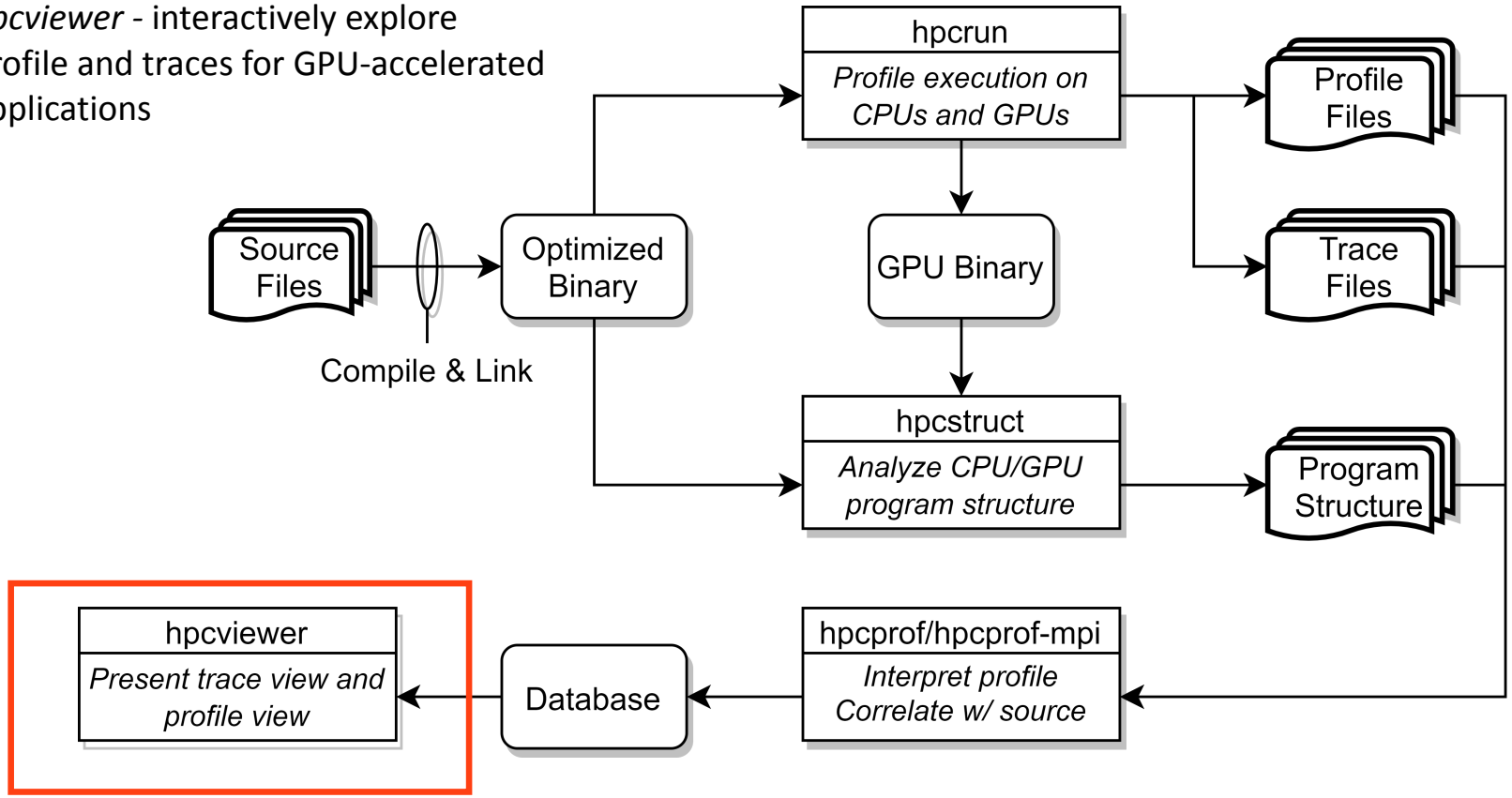
- Analyze data from large executions in parallel

```
jsrun -n 2 -a 1 -c 22 -b packed hpcprof-mpi <measurement-directory>
srun -N 2 -n 2 -c 126 hpcprof-mpi <measurement-directory>
```

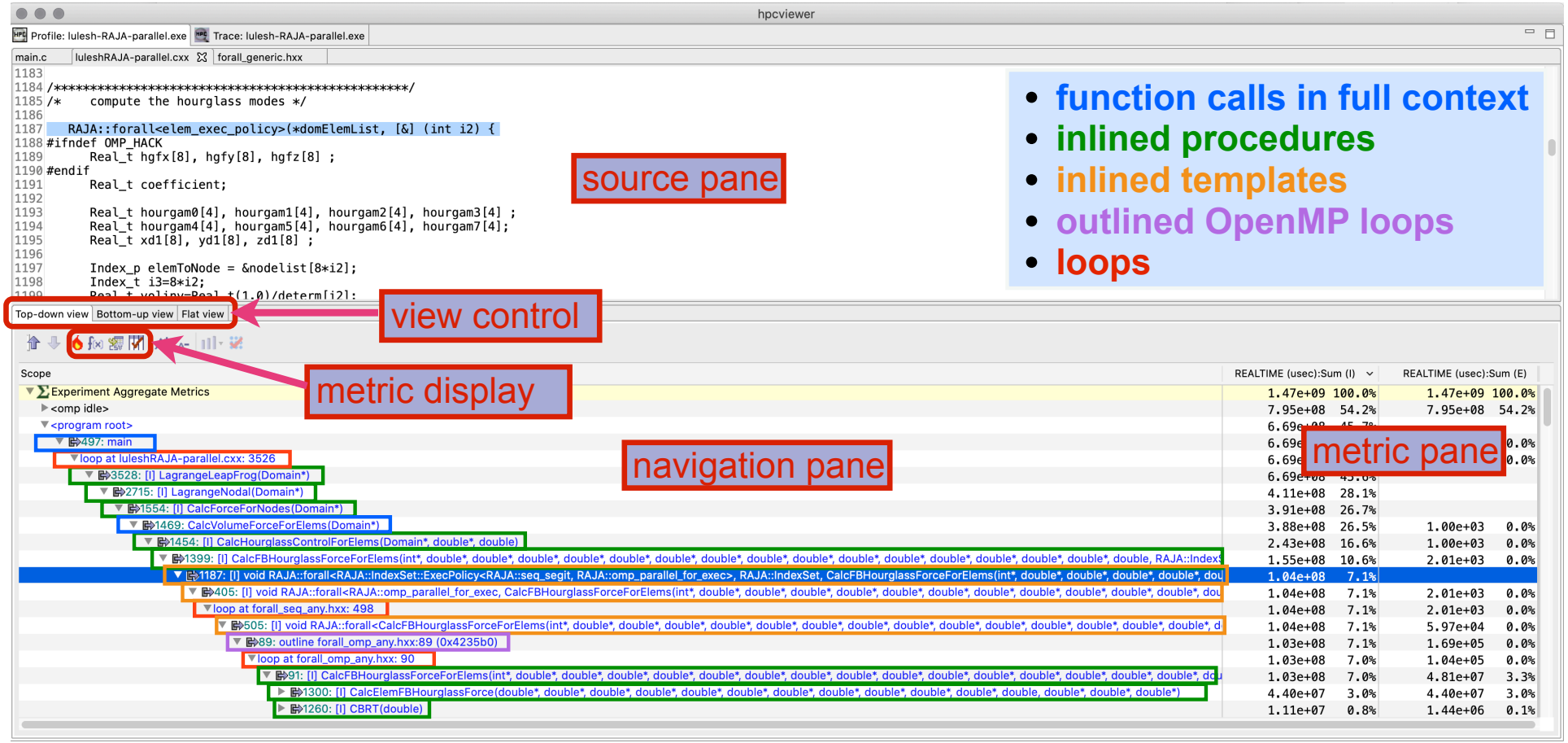
HPCToolkit's Workflow for GPU-accelerated Applications

Step 4:

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

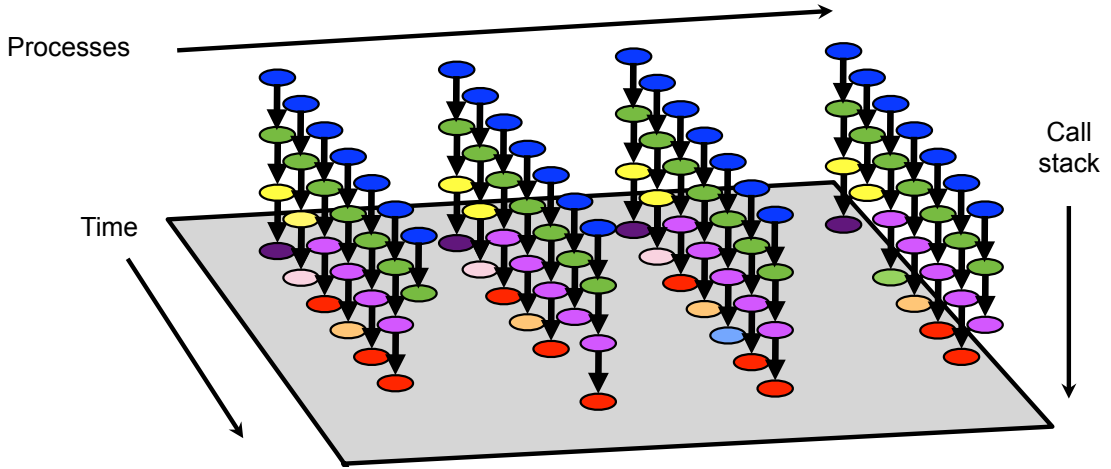


Code-centric Analysis with hpcviewer

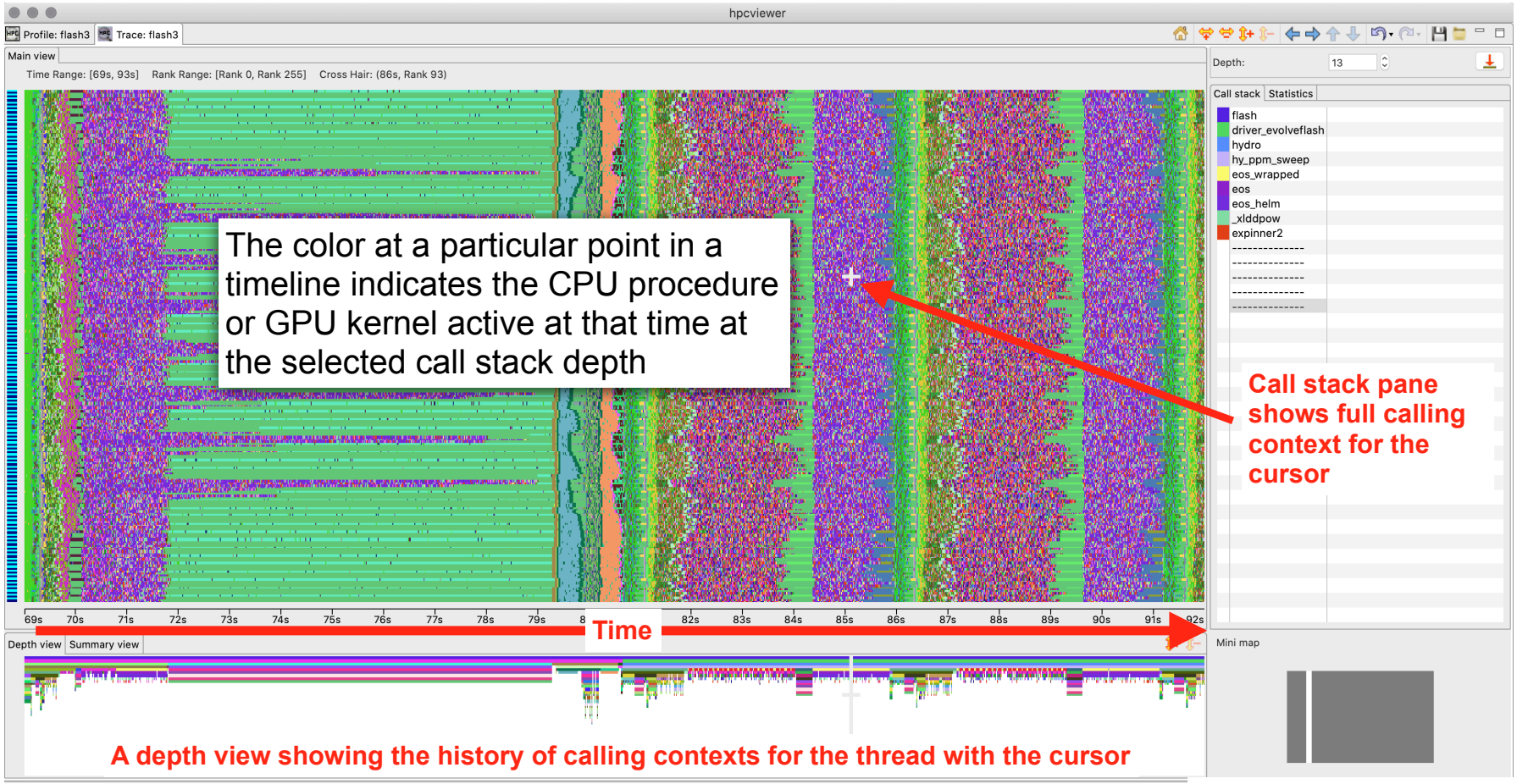


Understanding Temporal Behavior

- 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

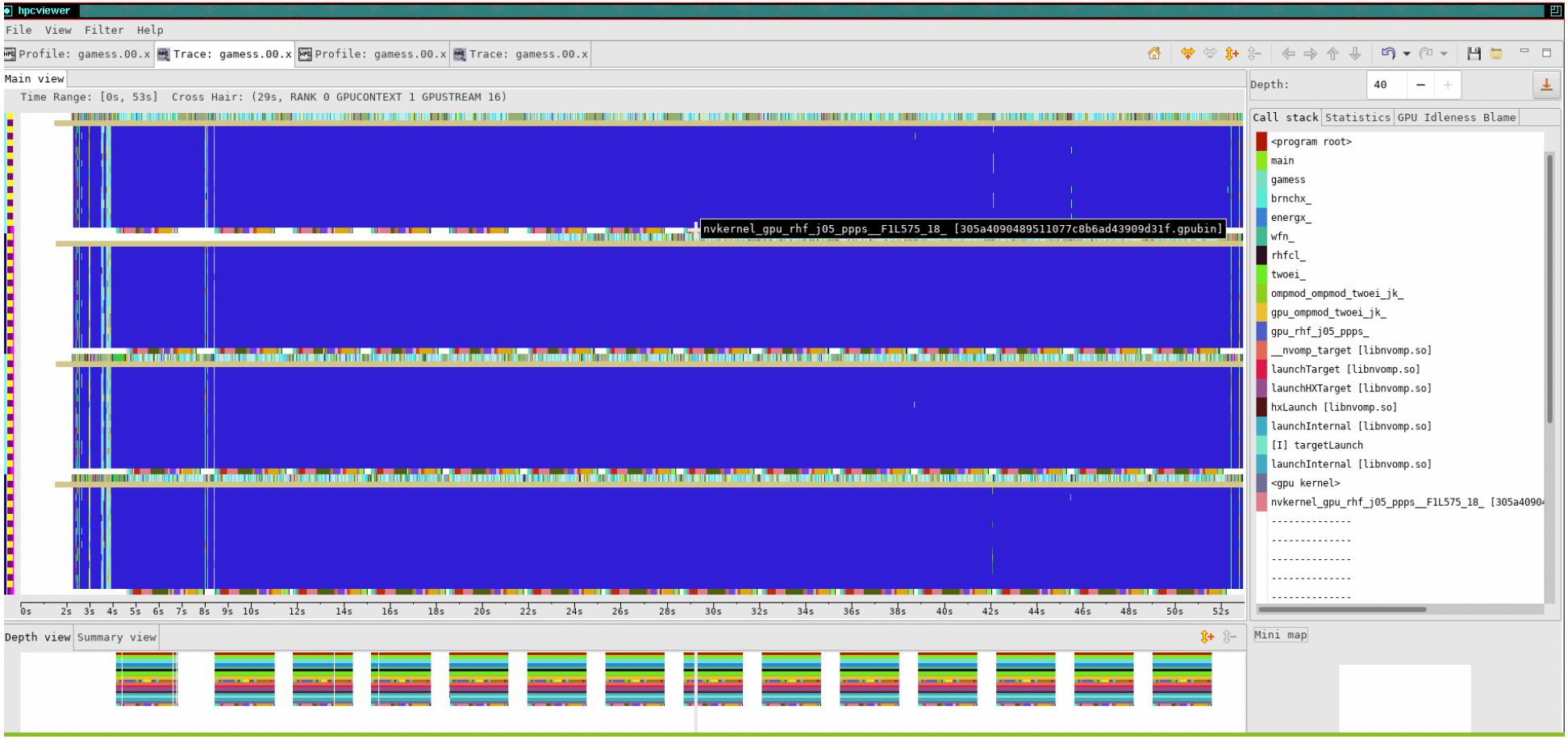
Minimap indicates part of execution trace shown

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

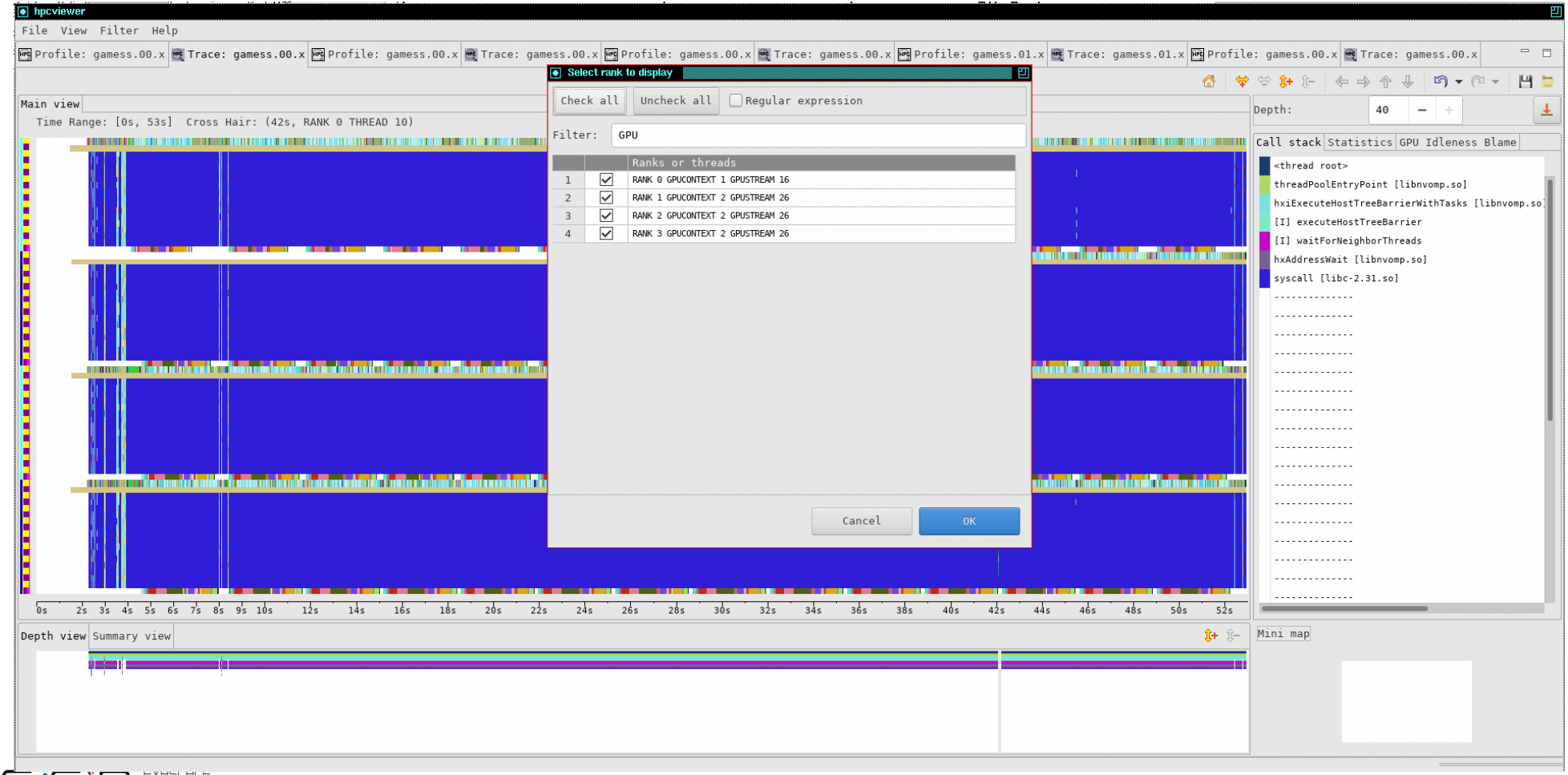
- Experiments
  - GPU-accelerated nodes at a Perlmutter hackathon
    - Single node with 4 GPUs
    - Five nodes with 20 GPUs
  - GPU-accelerated node on Crusher

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



GAMESS original

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

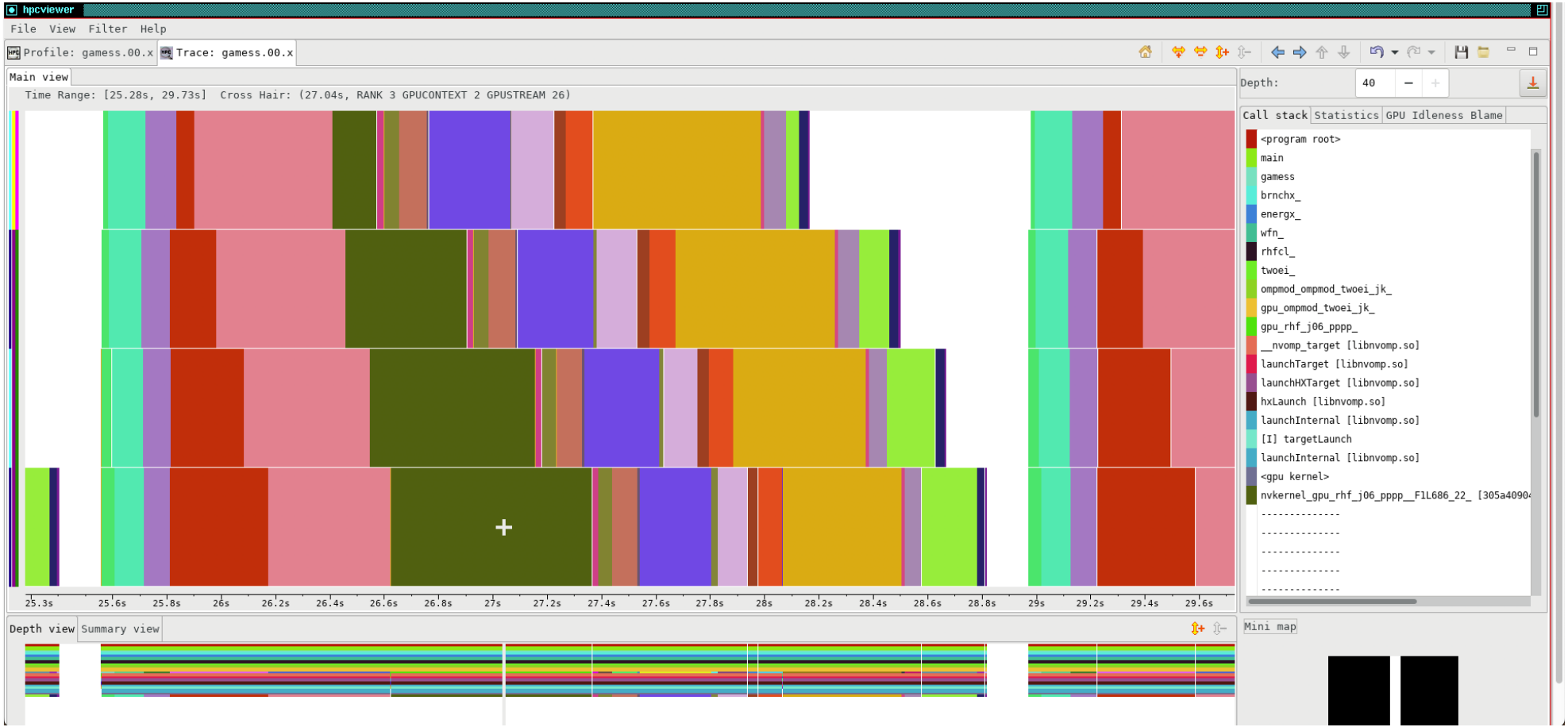


GAMESS original

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

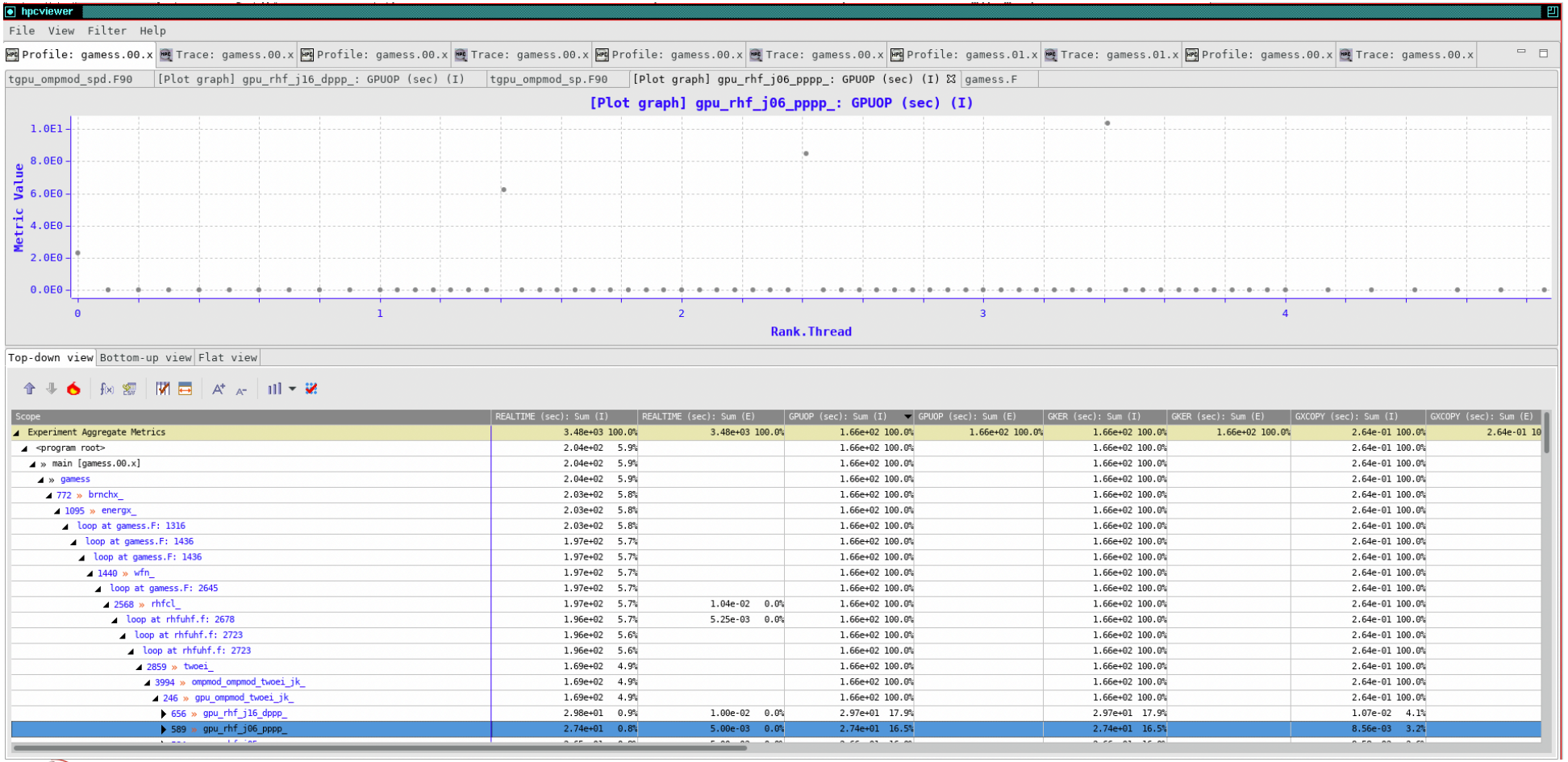
GAMESS original

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



GAMESS original

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

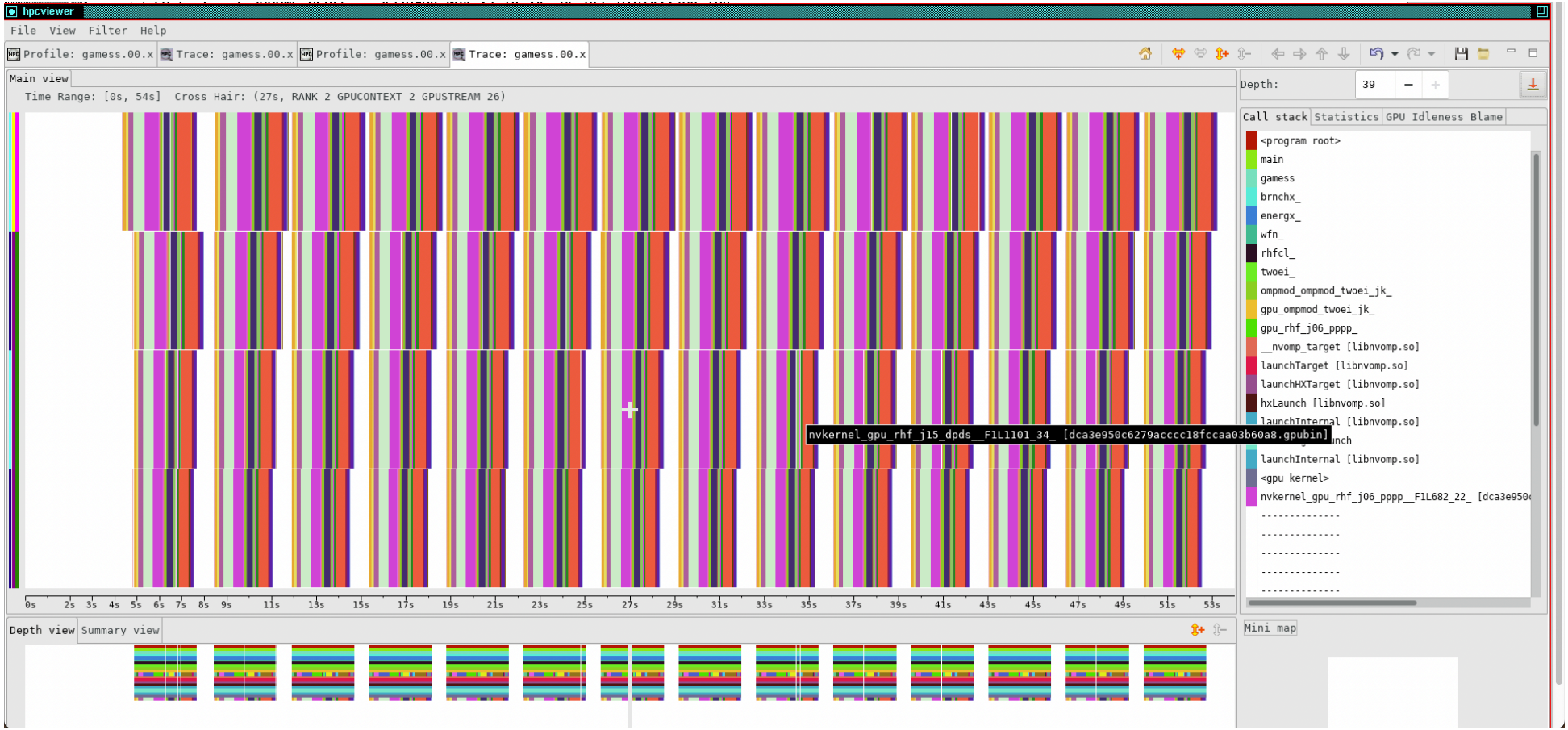


GAMESS original

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

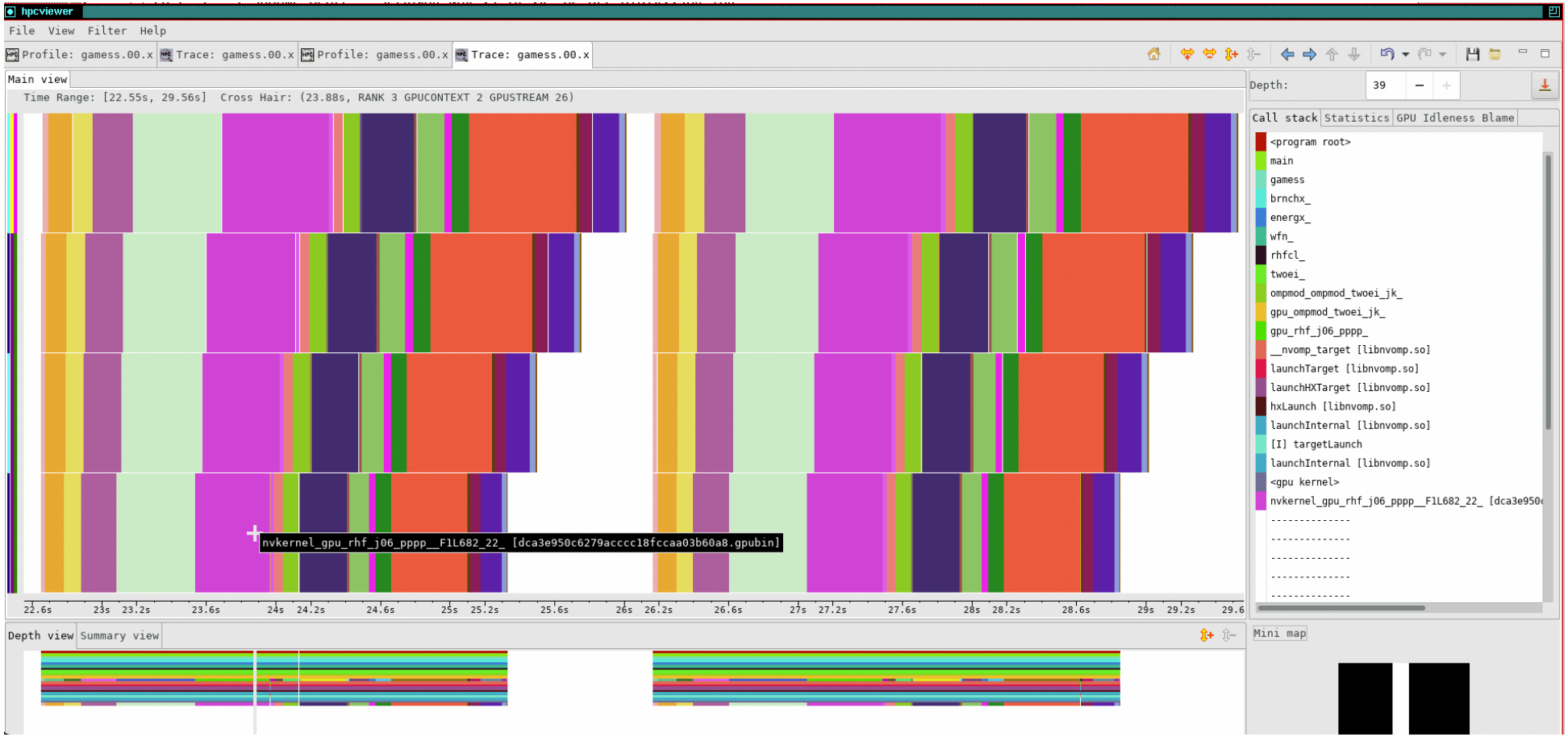
GAMESS improved

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



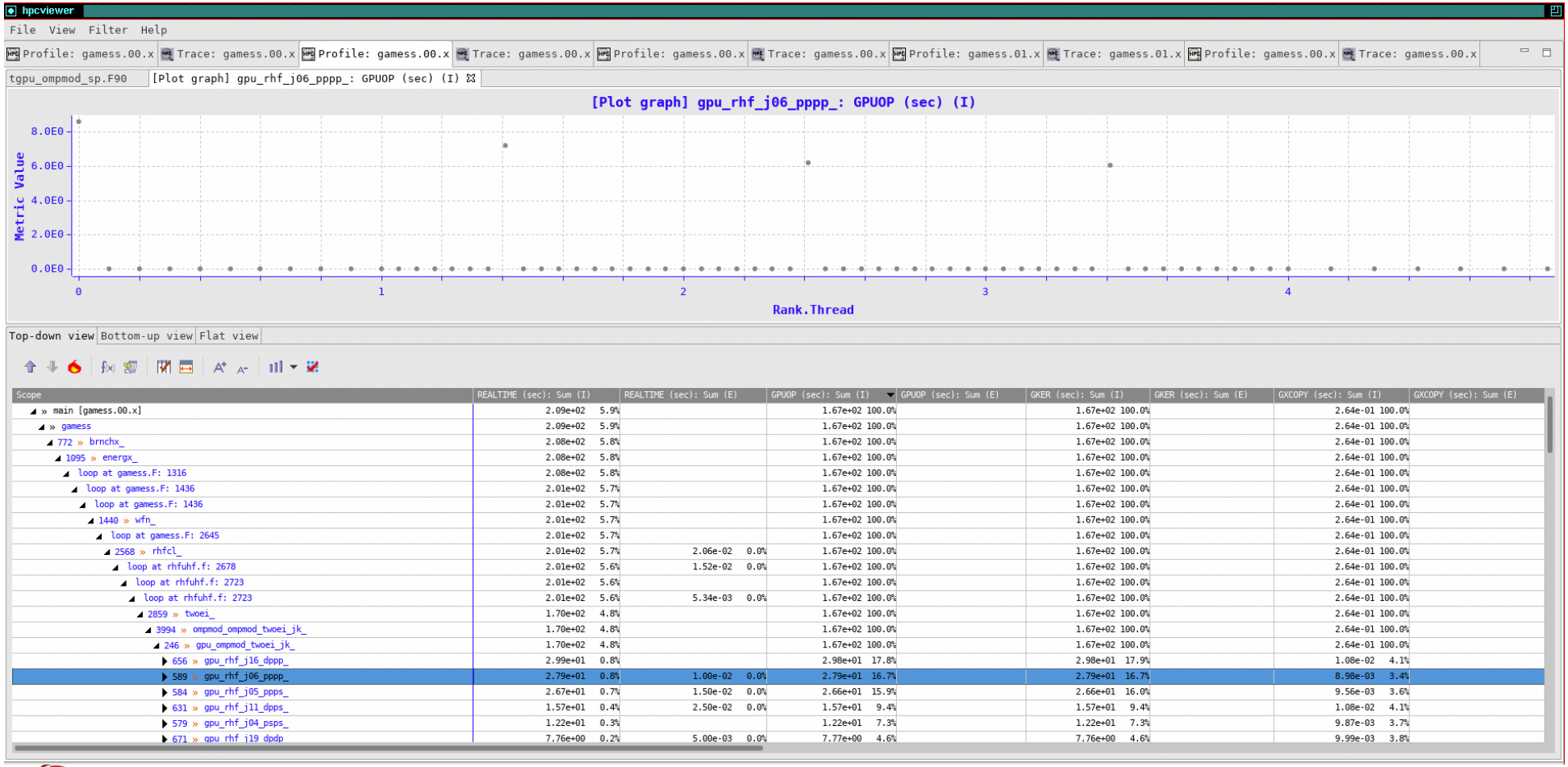
GAMESS improved

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



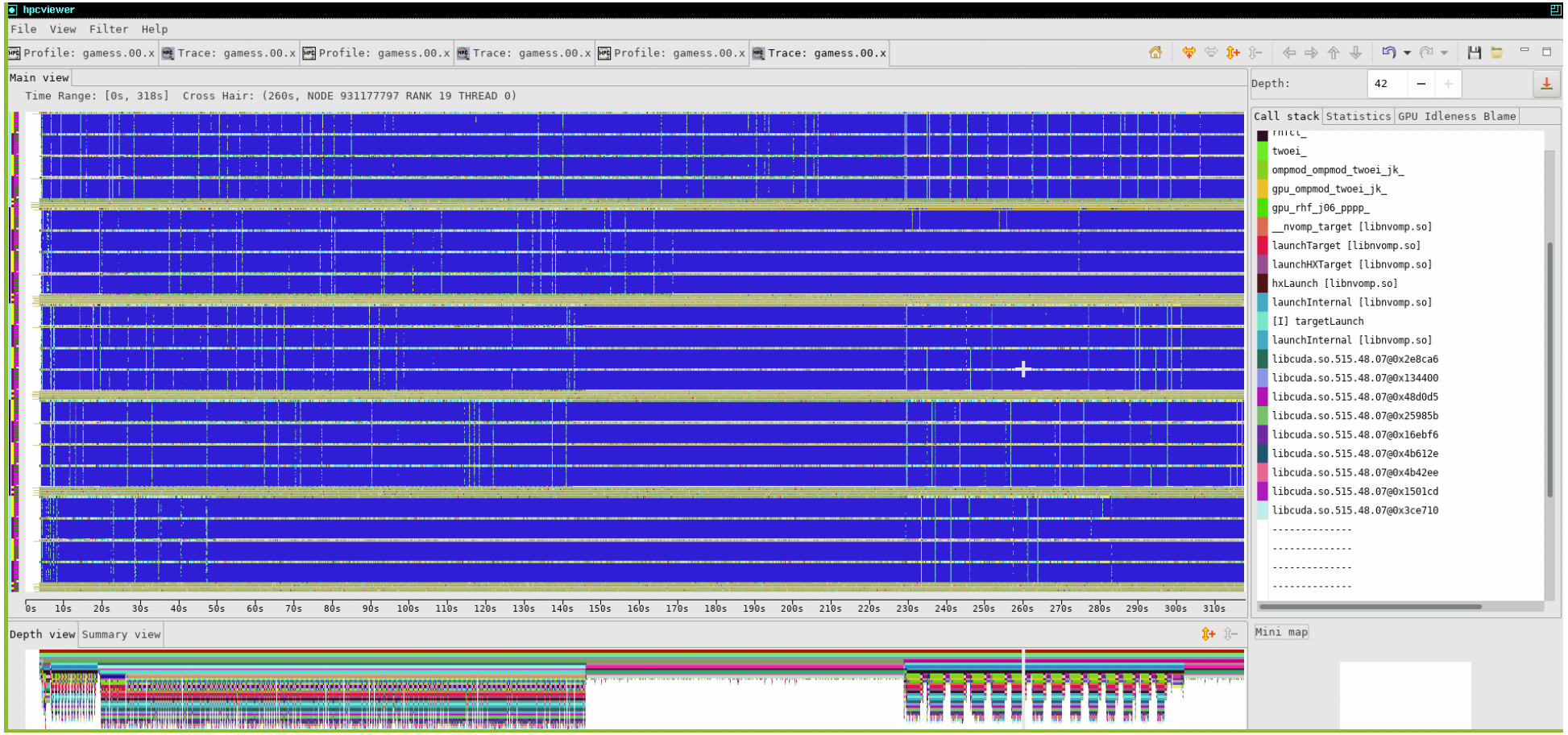
GAMESS improved

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



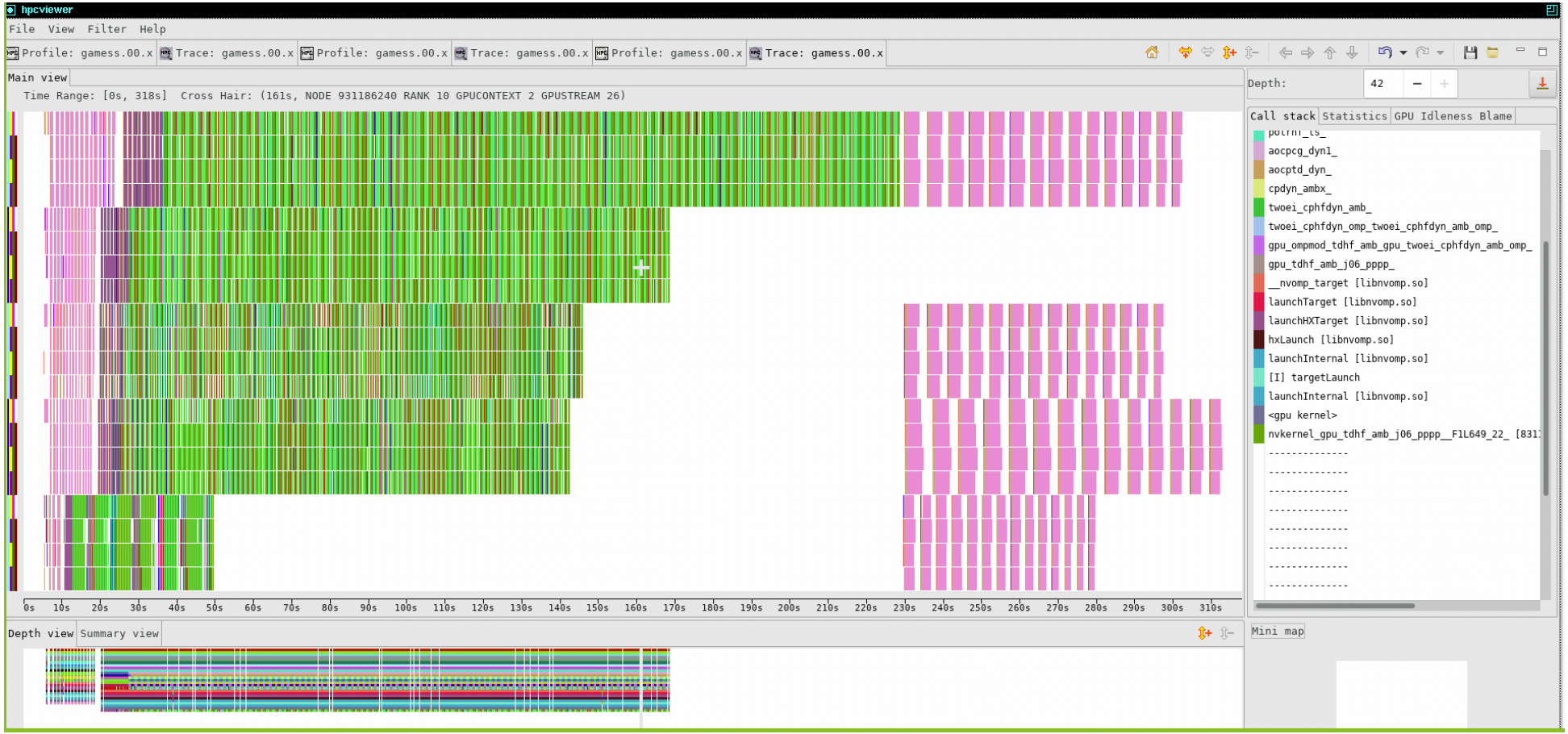
GAMESS improved

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



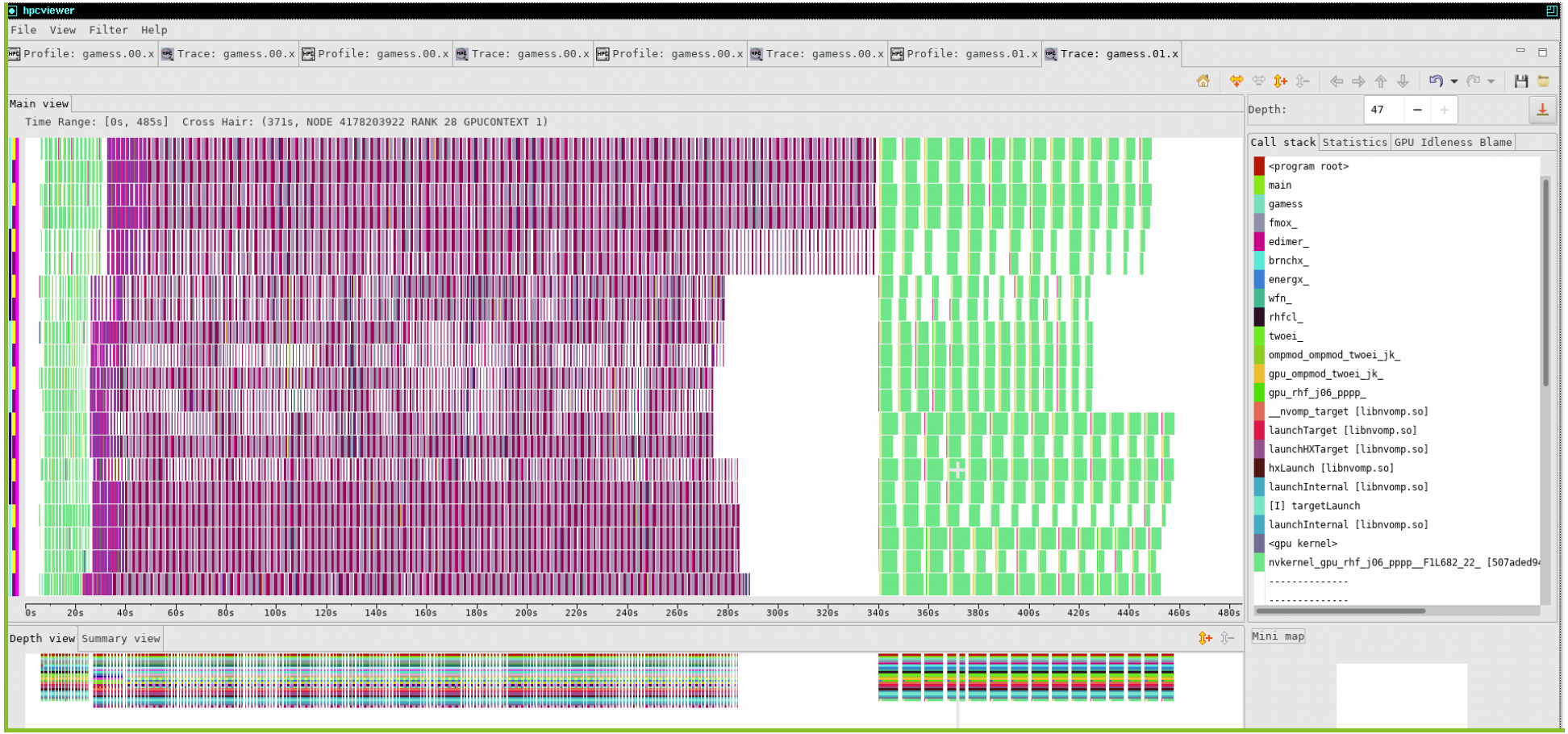
GAMESS improved

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



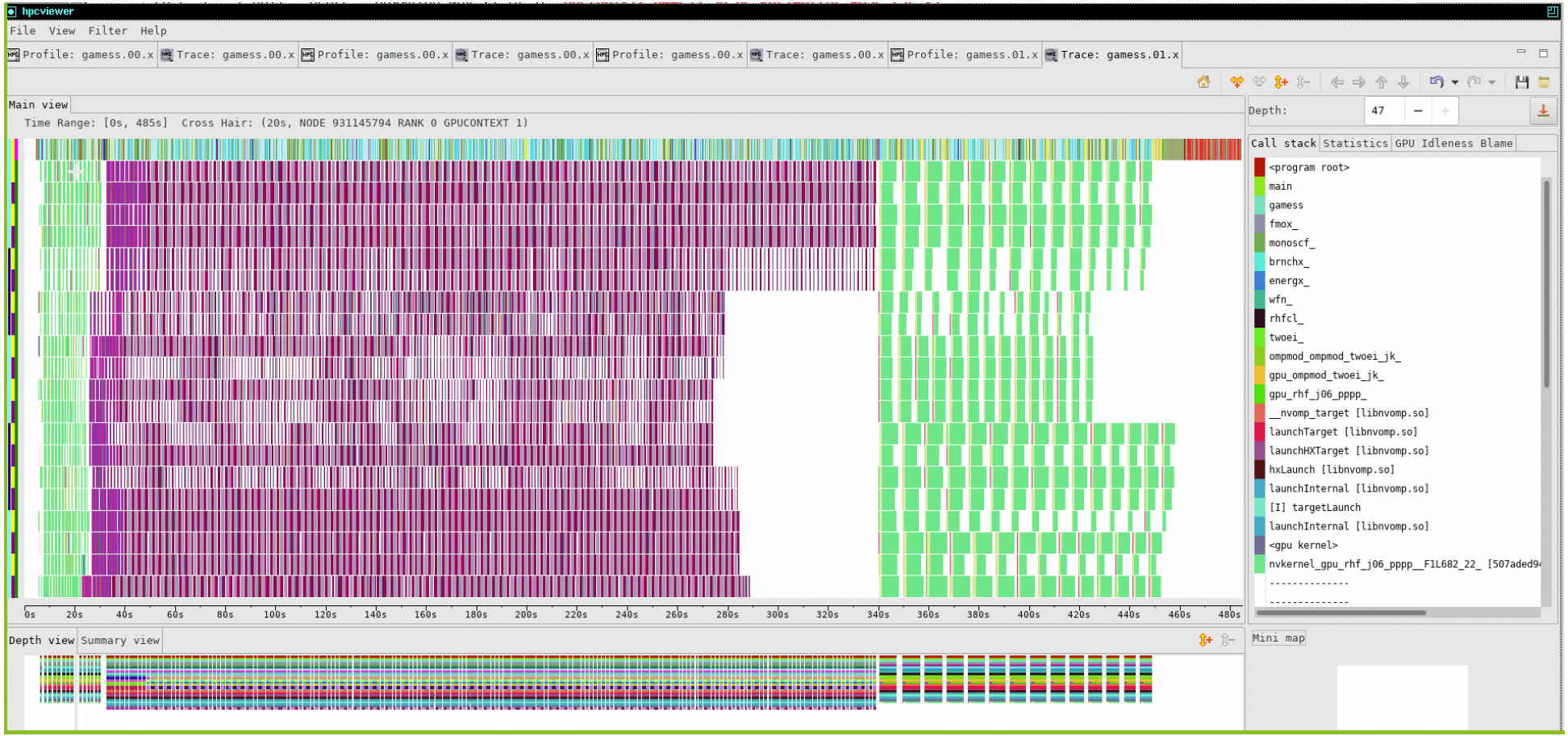
GAMESS improved

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



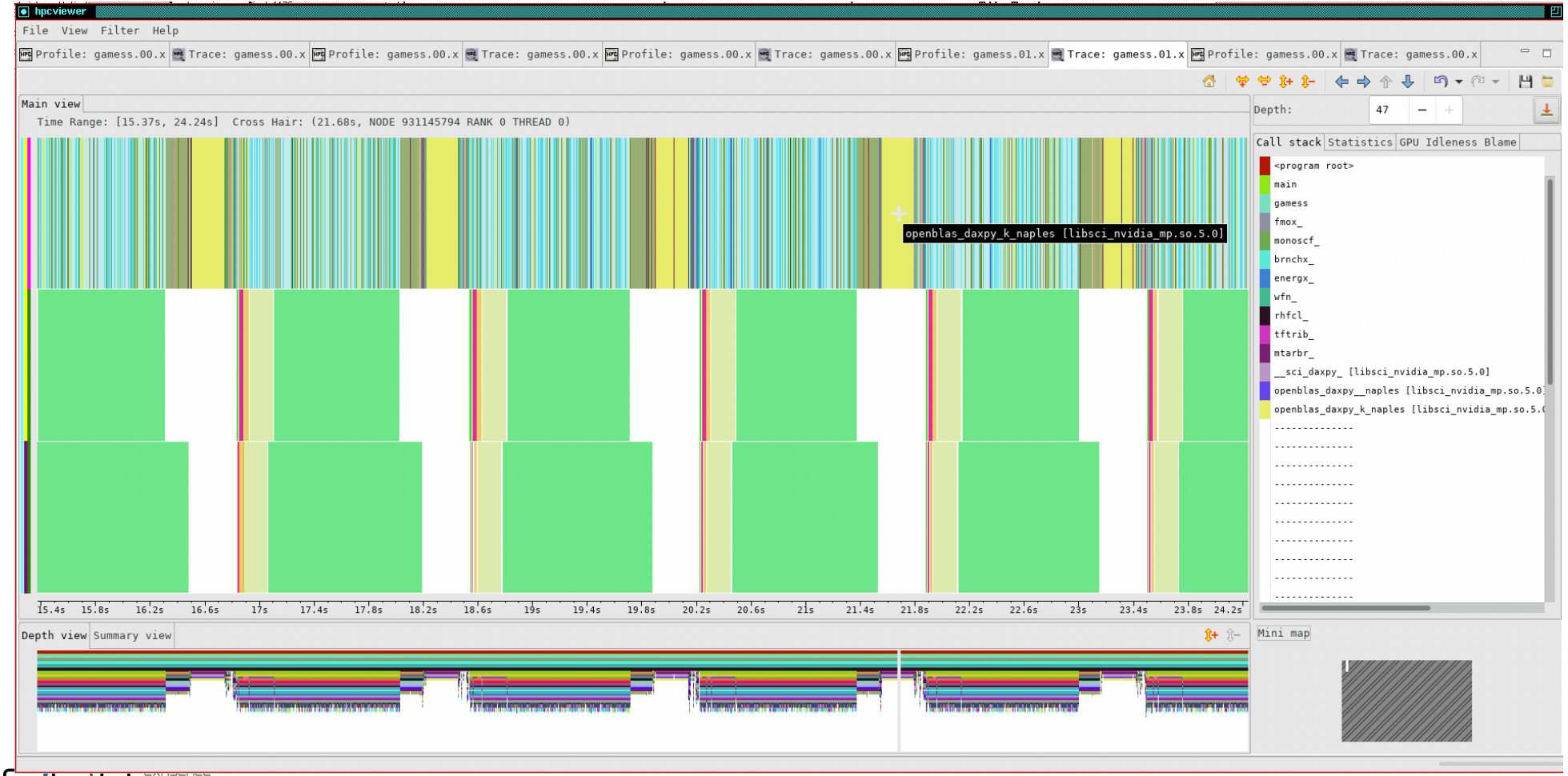
GAMESS improved with better manual distribution of work in input

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

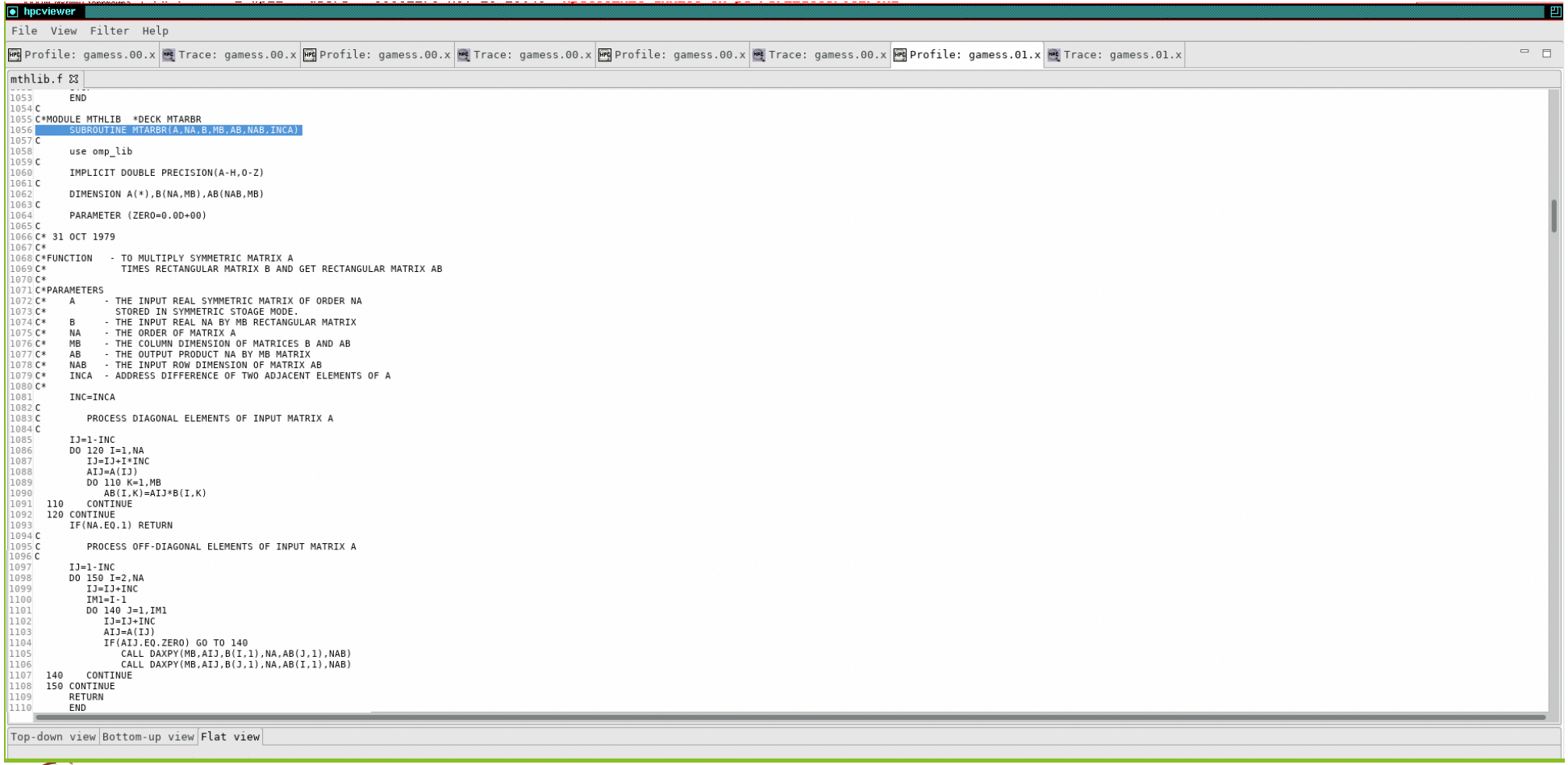


GAMESS improved adding Rank 0 Thread 0 to GPU streams

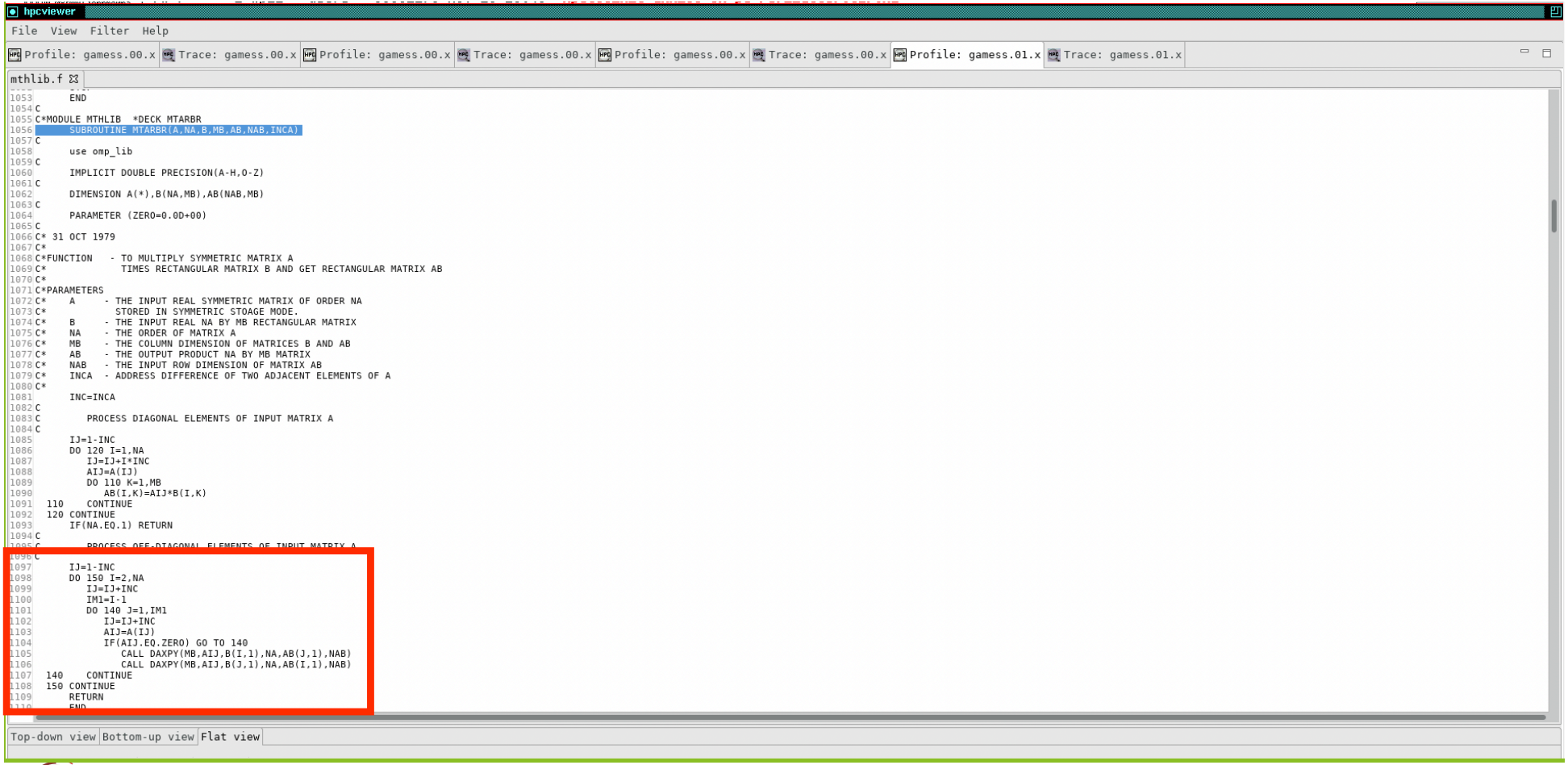
Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter



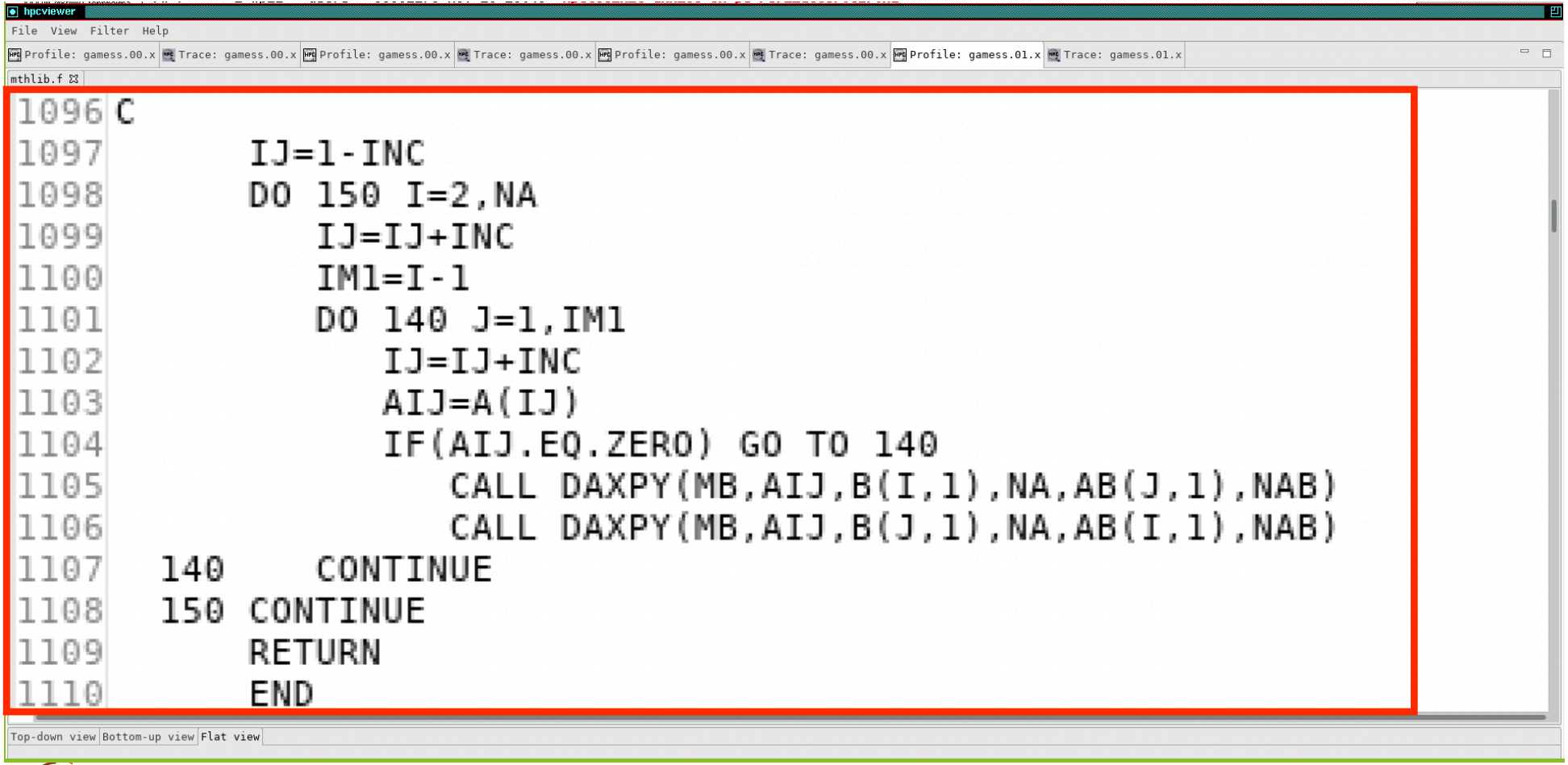
Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter



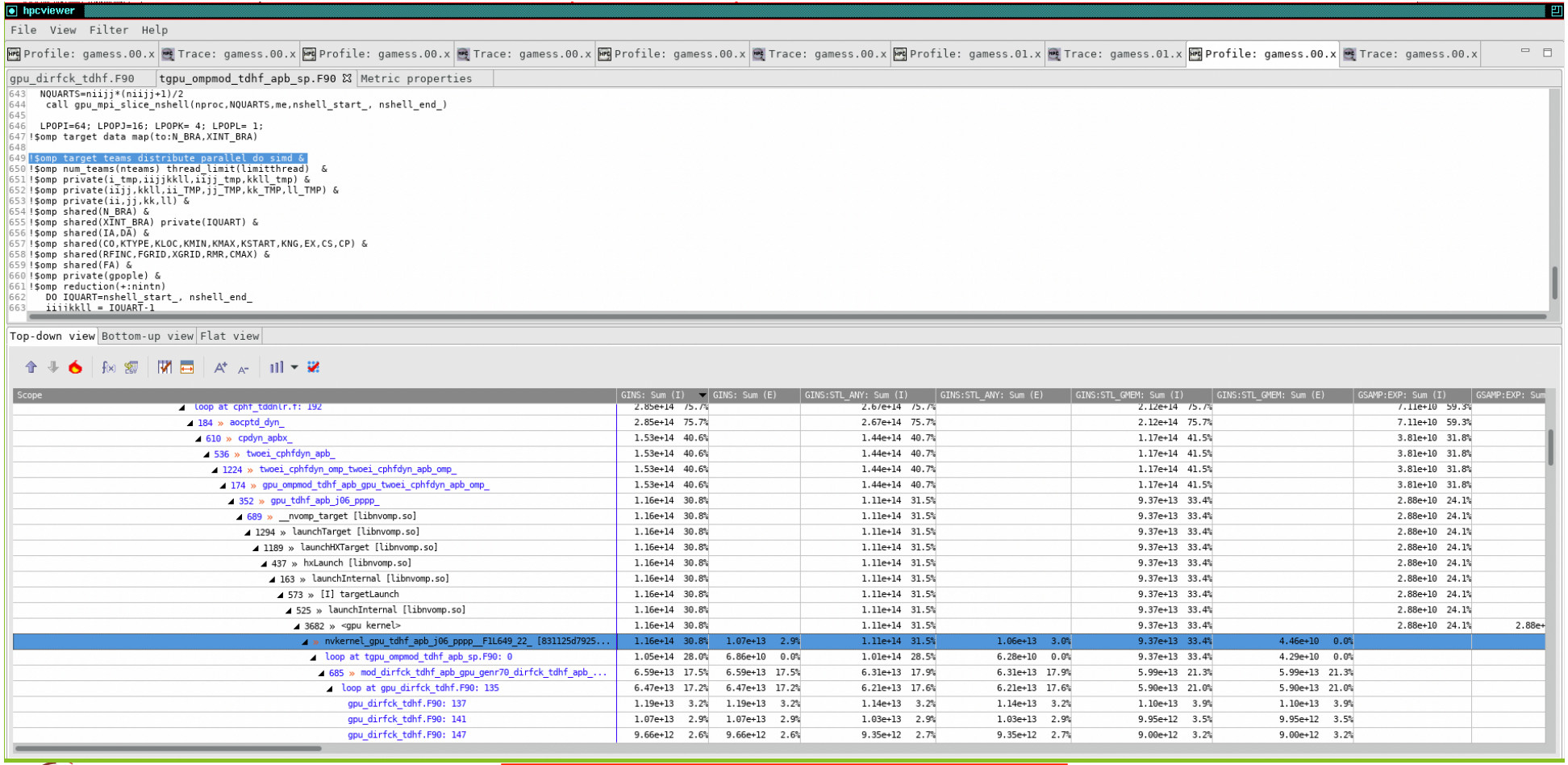
Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter



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

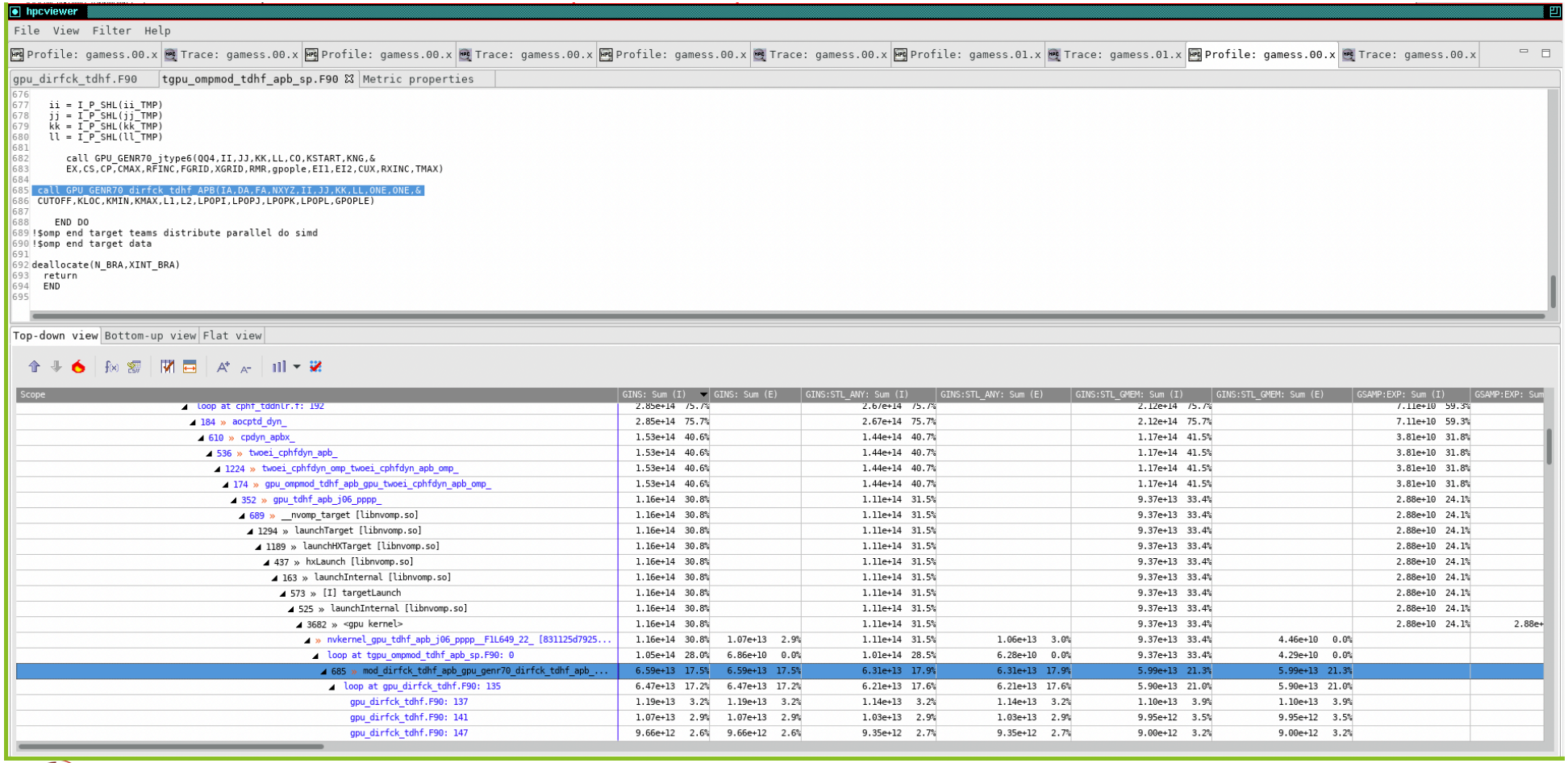


Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter

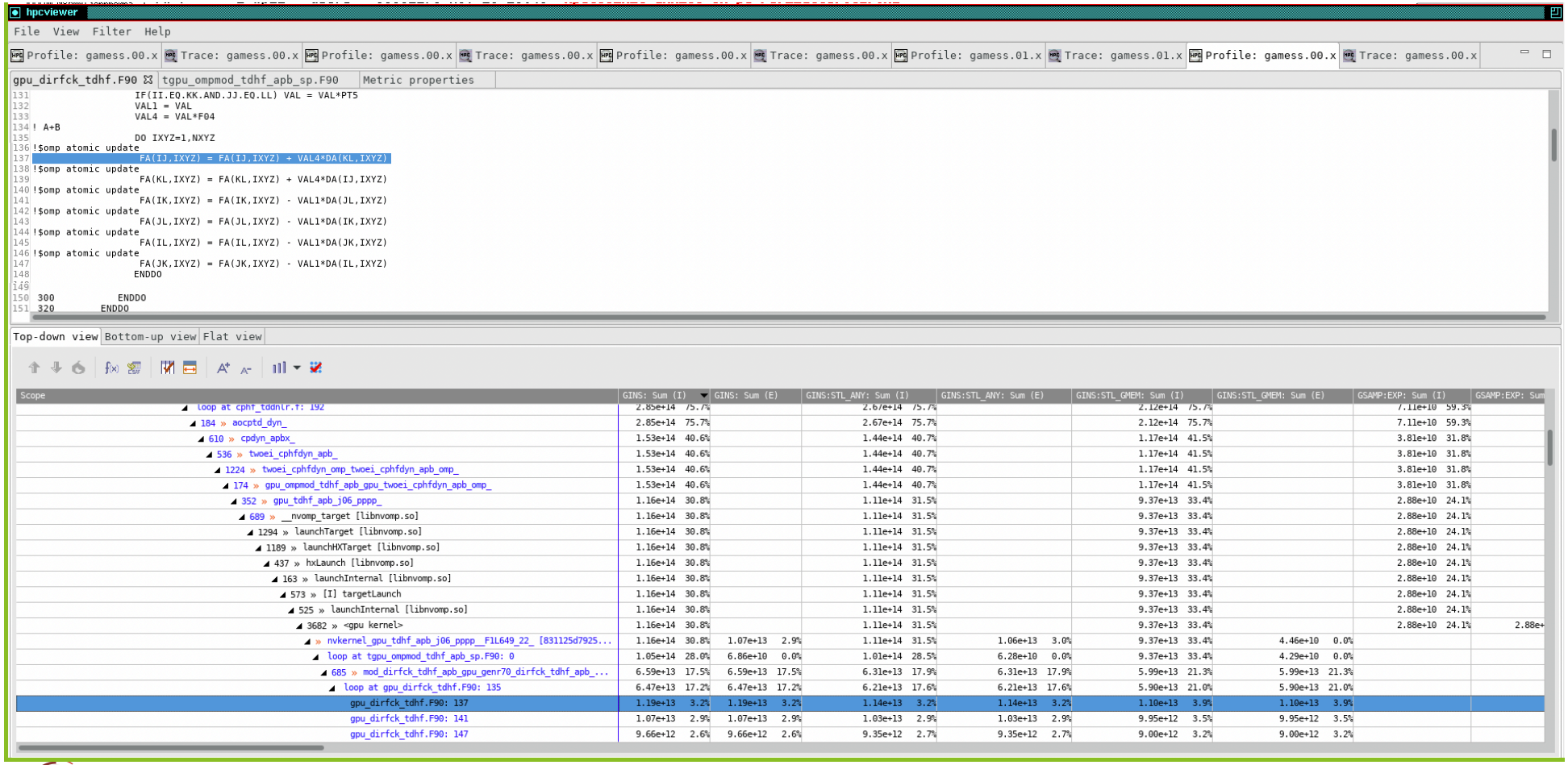


GAMESS improved with PC Sampling

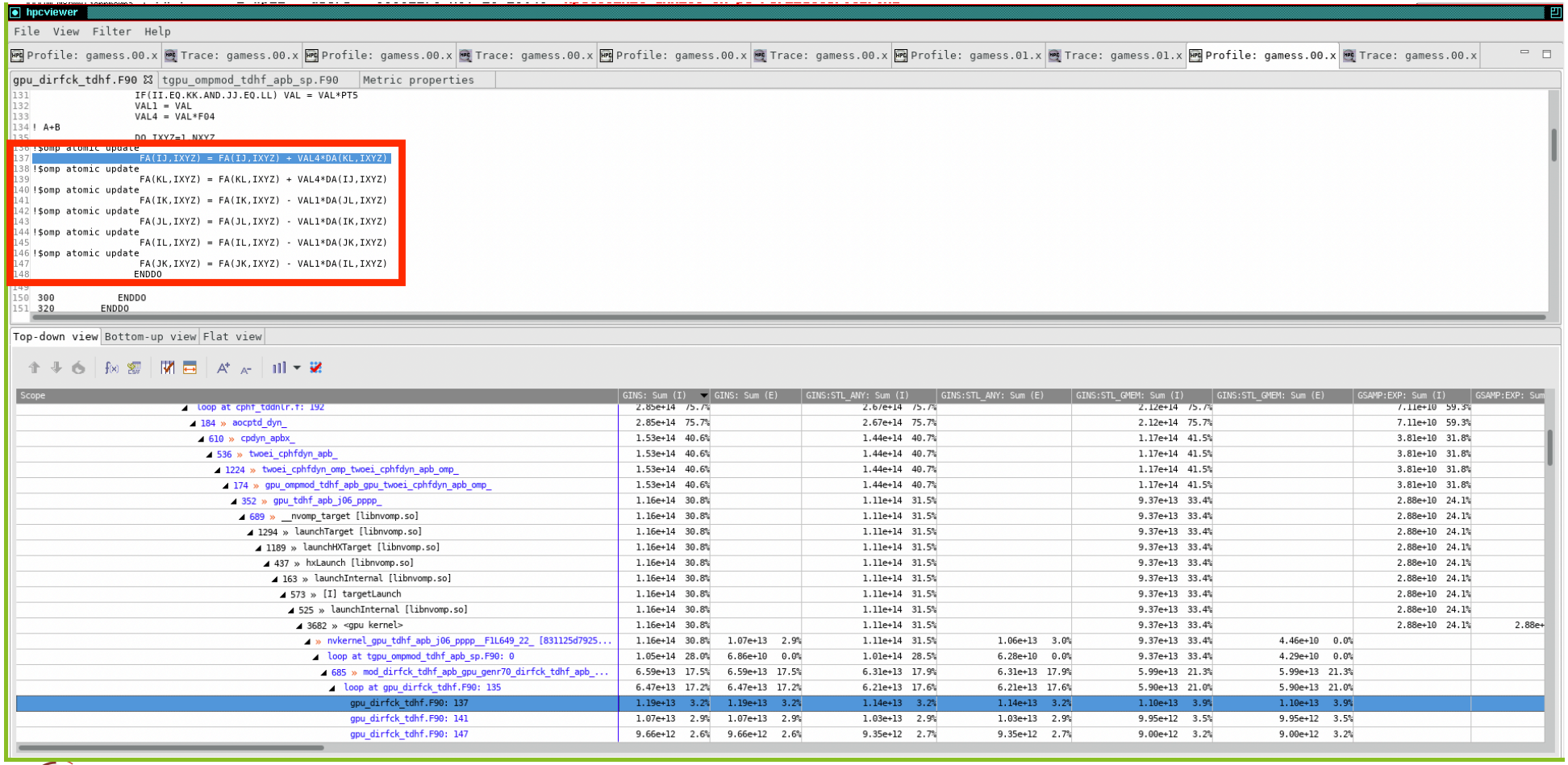
Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter



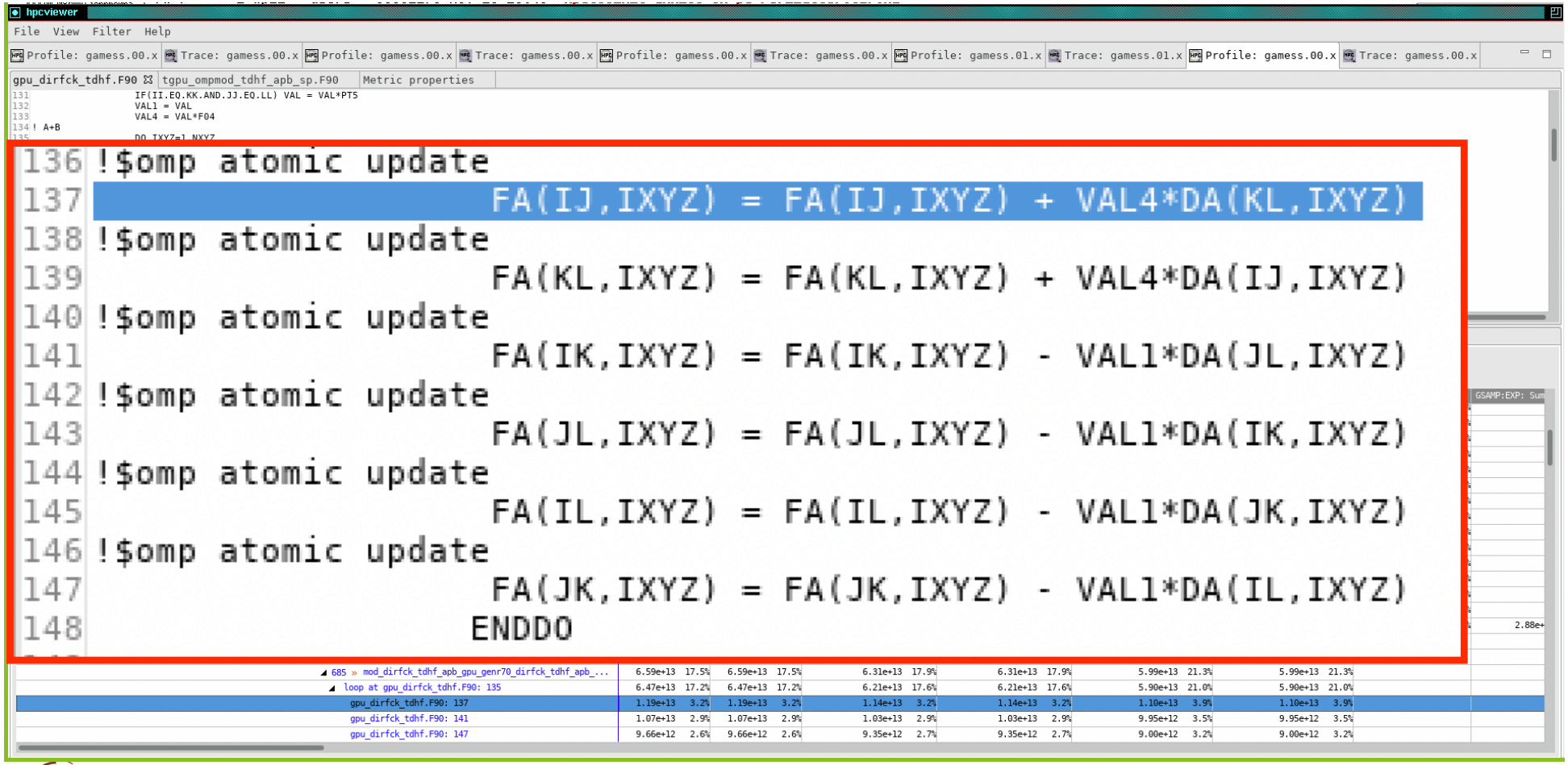
Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter



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

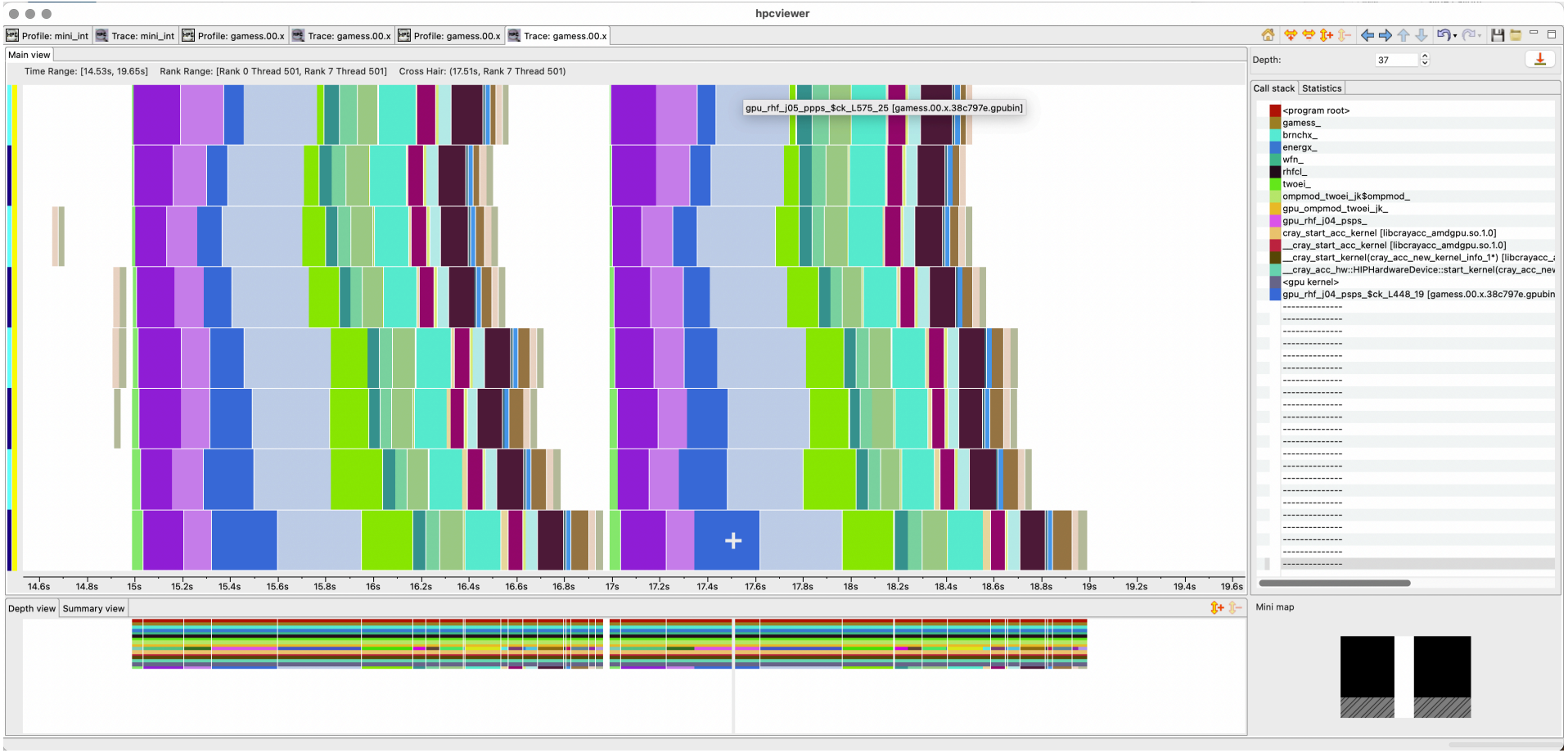


Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter



Time-centric Analysis: GAMESS 16 ranks, 8 GPUs on Crusher

Time-centric Analysis: GAMESS 16 ranks, 8 GPUs on Crusher

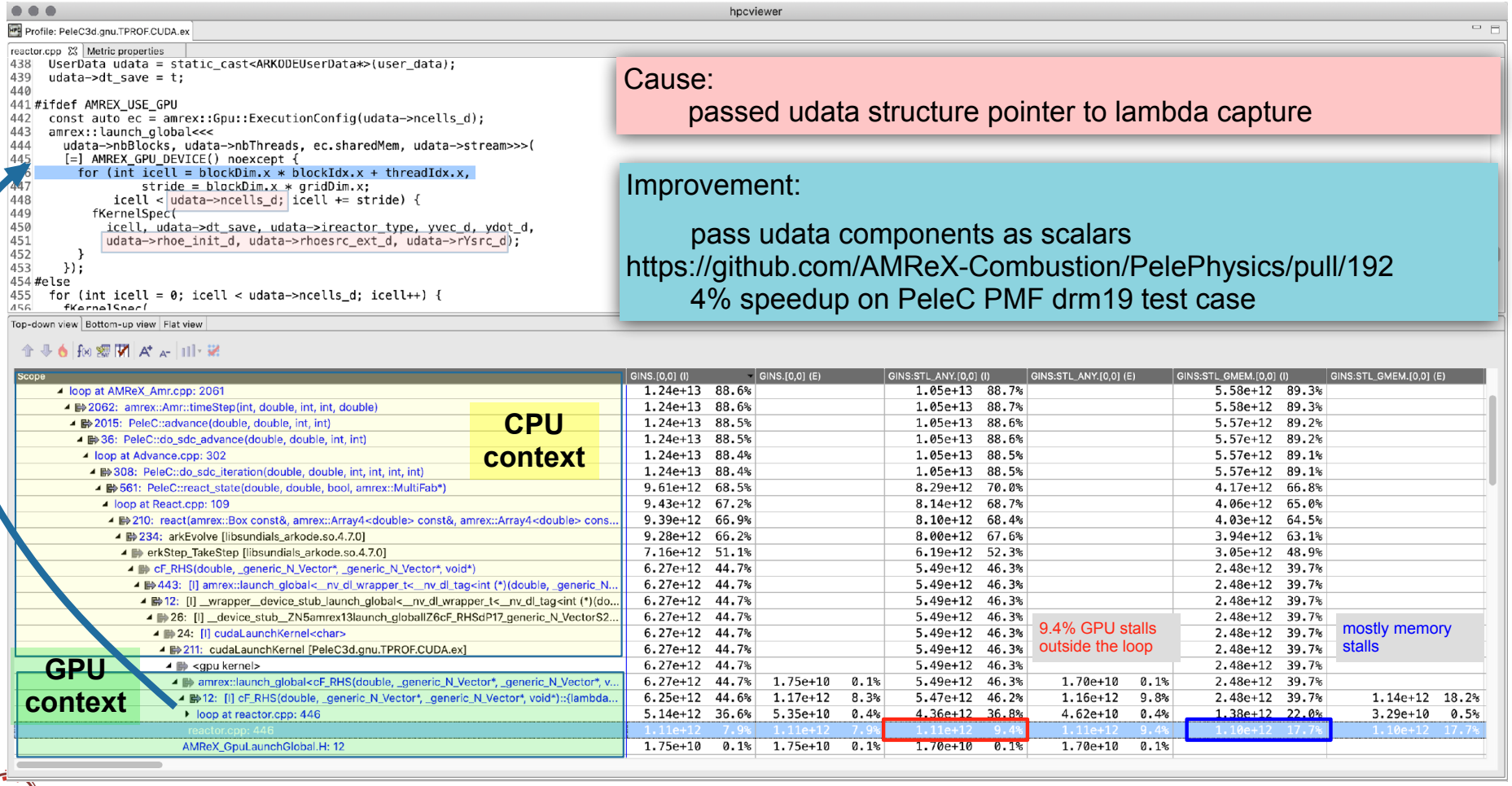


Case Study: PeleC

- An adaptive mesh refinement solver for compressible reacting flows

- Experiment
  - PC Sampling on Summit node with NVIDIA GPU

Analysis of PeleC using PC Sampling on an NVIDIA GPU



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)	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:446	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%		

HPCToolkit Status on GPUs

- NVIDIA
  - heterogeneous profiles, including 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, including GPU instruction-level measurements with kernel instrumentation and heuristic latency attribution to instructions
  - traces

Using HPCToolkit at OLCF

- Summit (NVIDIA GPUs)
  - `module use /gpfs/alpine/csc322/world-shared/modulefiles/ppc64le`
  - `module load hpctoolkit`

- Crusher (AMD GPUs)
  - `module use /gpfs/alpine/csc322/world-shared/modulefiles/x86_64`
  - `module load hpctoolkit`

- Join our ECP Engagement Slack and ask questions
  - https://join.slack.com/t/hpctoolkit-ecp/shared_invite/zt-1lgrzjt93-aEB43o3SVgKFy1yFCjwlyA

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: https://youtu.be/vixa3hGDuGg
  - 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