Performance Analysis of GPU-accelerated OpenMP Applications using HPCToolkit

EXASCALE COMPUTING PROJECT

John Mellor-Crummey
Rice University

August 25, 2023

OpenMP Users Monthly Telecon

National Nuclear Security Administration

U.S. DEPARTMENT OF ENERGY | Office of Science

RICE

WISCONSIN UNIVERSITY OF WISCONSIN–MADISON

Outline

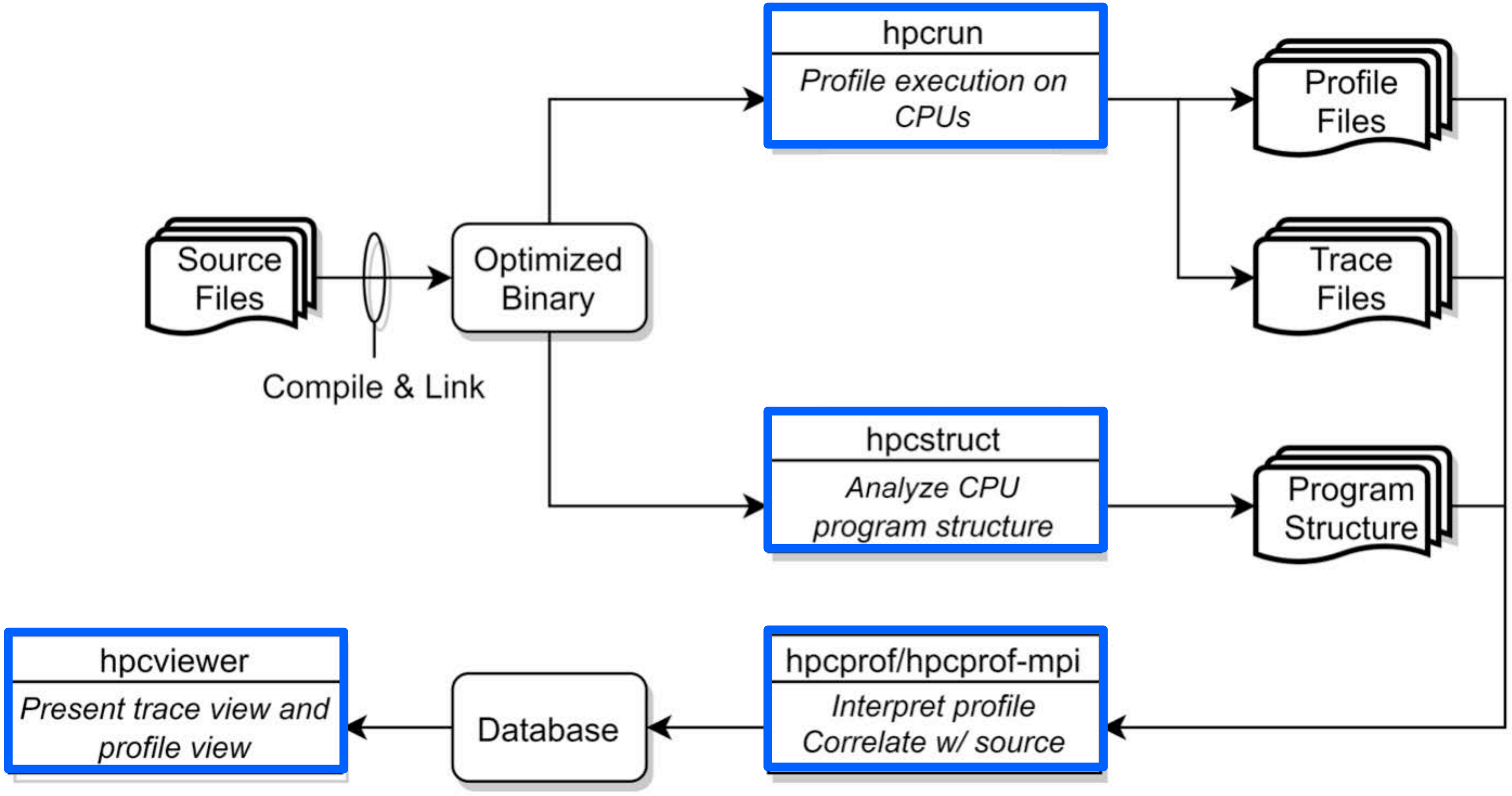
- Introduction to HPCToolkit
  - Overview of HPCToolkit components and their workflow
  - HPCToolkit's graphical user interfaces
- Analyzing the performance of GPU-accelerated codes with HPCToolkit
  - GAMESS
  - GEM
- Status
- Ongoing work

Rice University's HPCToolkit Performance Tools

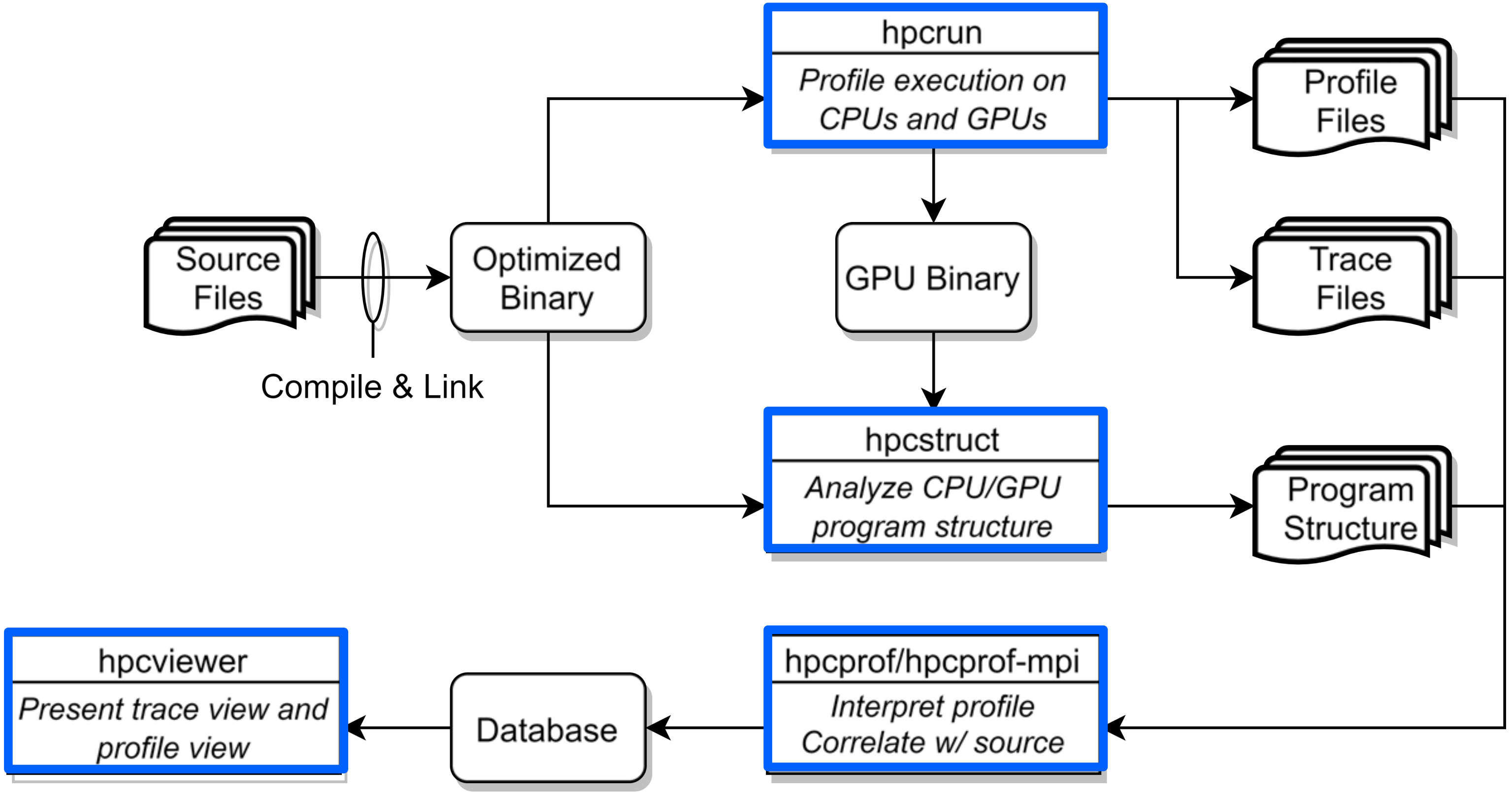
Measure and analyze performance of CPU and GPU-accelerated applications

- 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

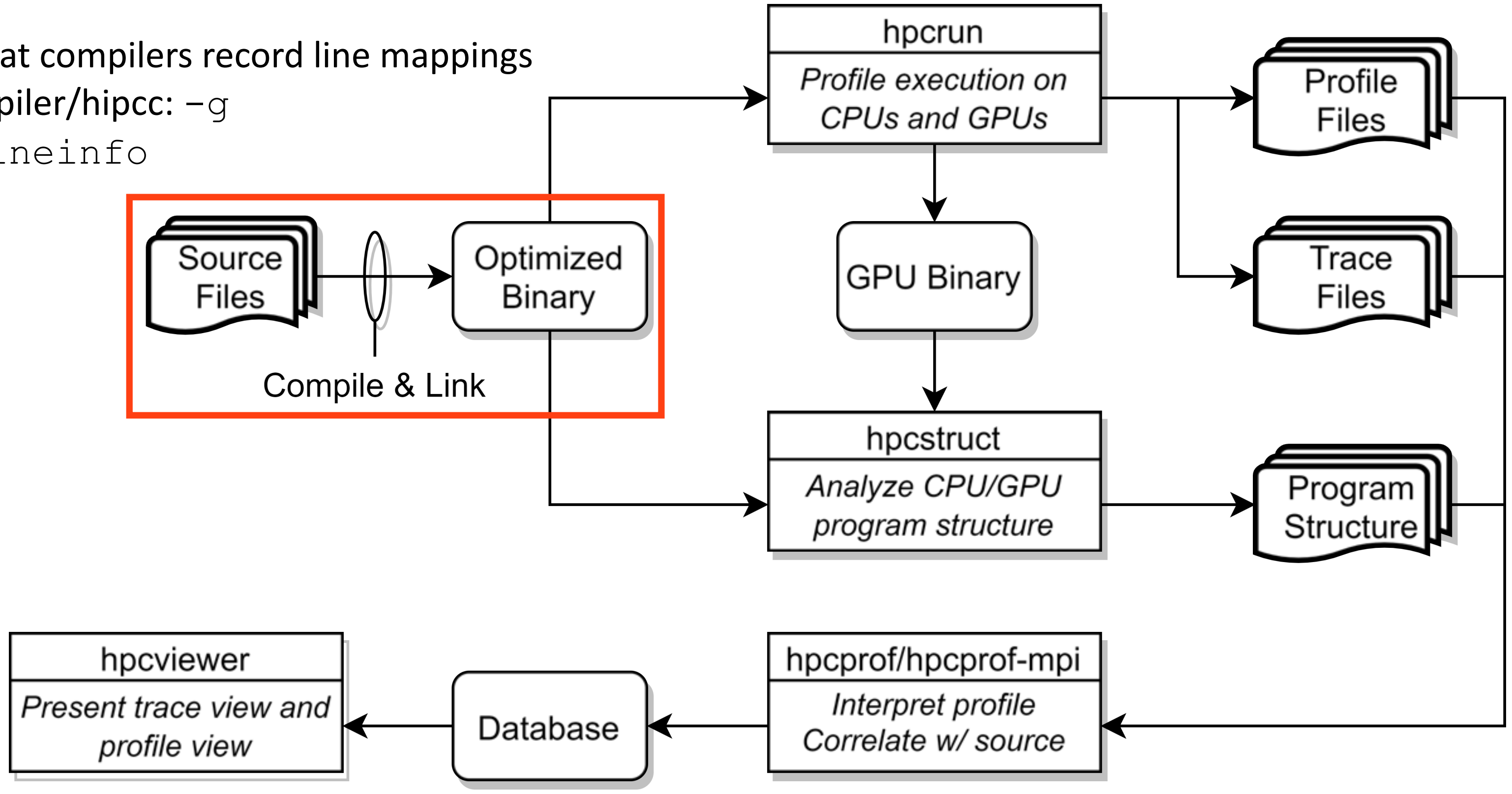


EXASCALE COMPUTING PROJECT

HPCToolkit's Workflow for GPU-accelerated Applications

Step 1:

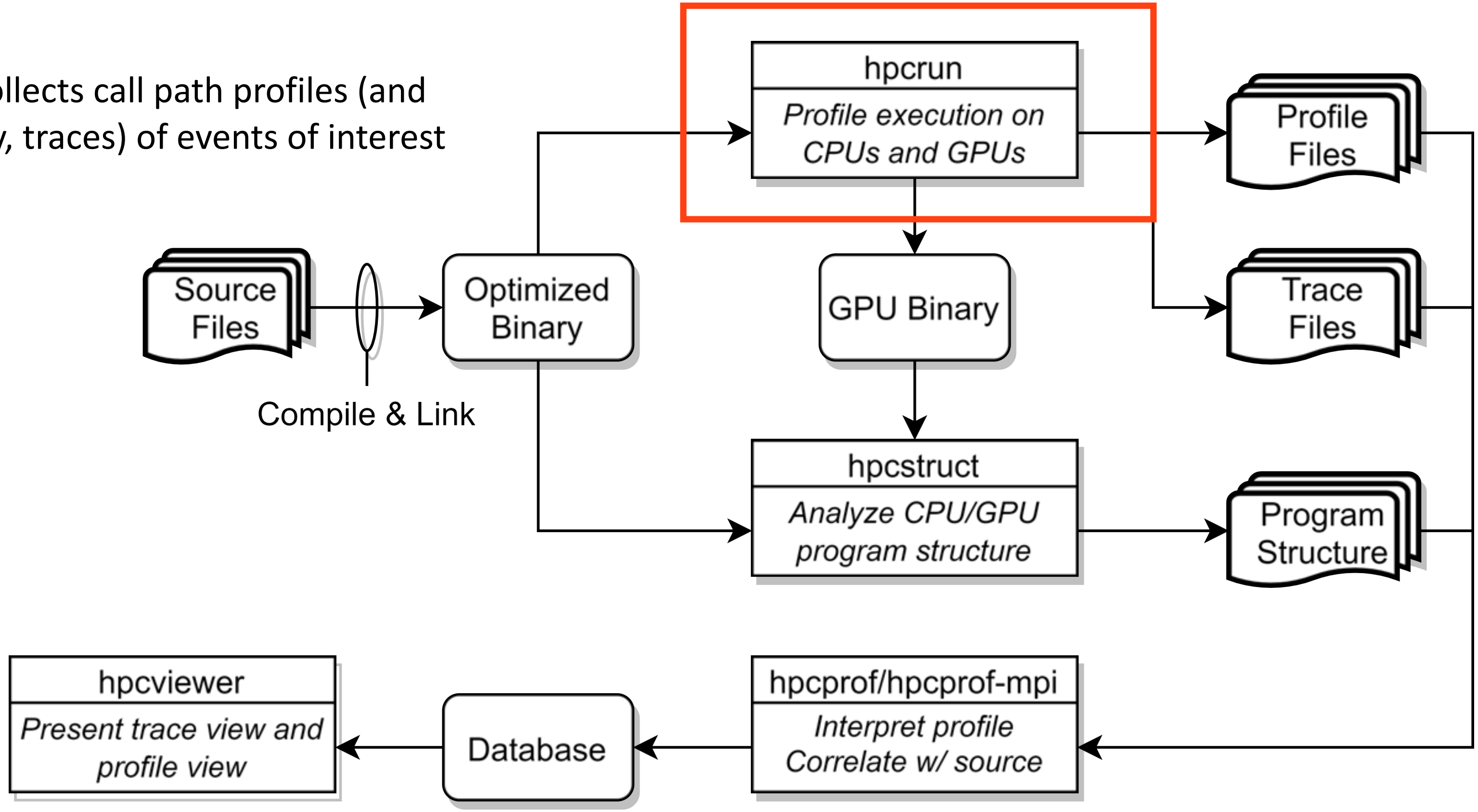
- Ensure that compilers record line mappings
- host compiler/hipcc: `-g`
- nvcc: `-lineinfo`



HPCToolkit's Workflow for GPU-accelerated Applications

Step 2:

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

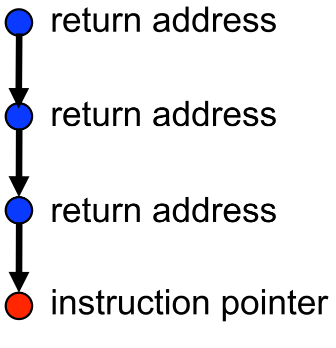


EXASCALE COMPUTING PROJECT

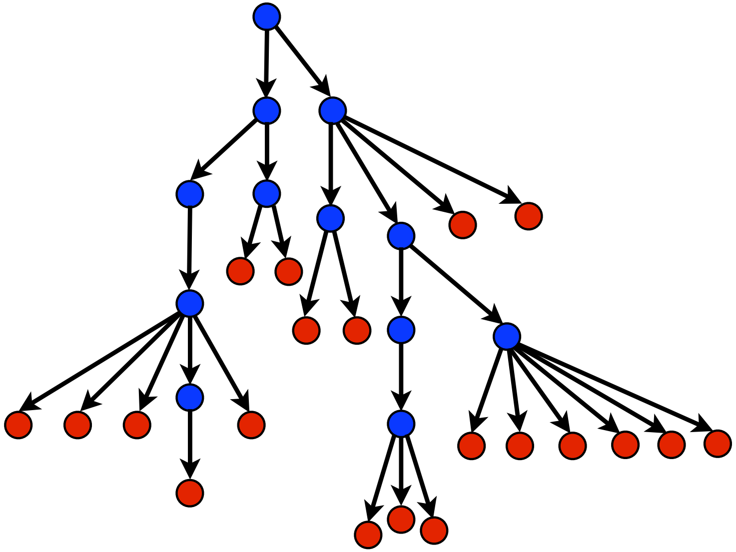
Call Stack Unwinding to Attribute Costs in Context

- 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>`

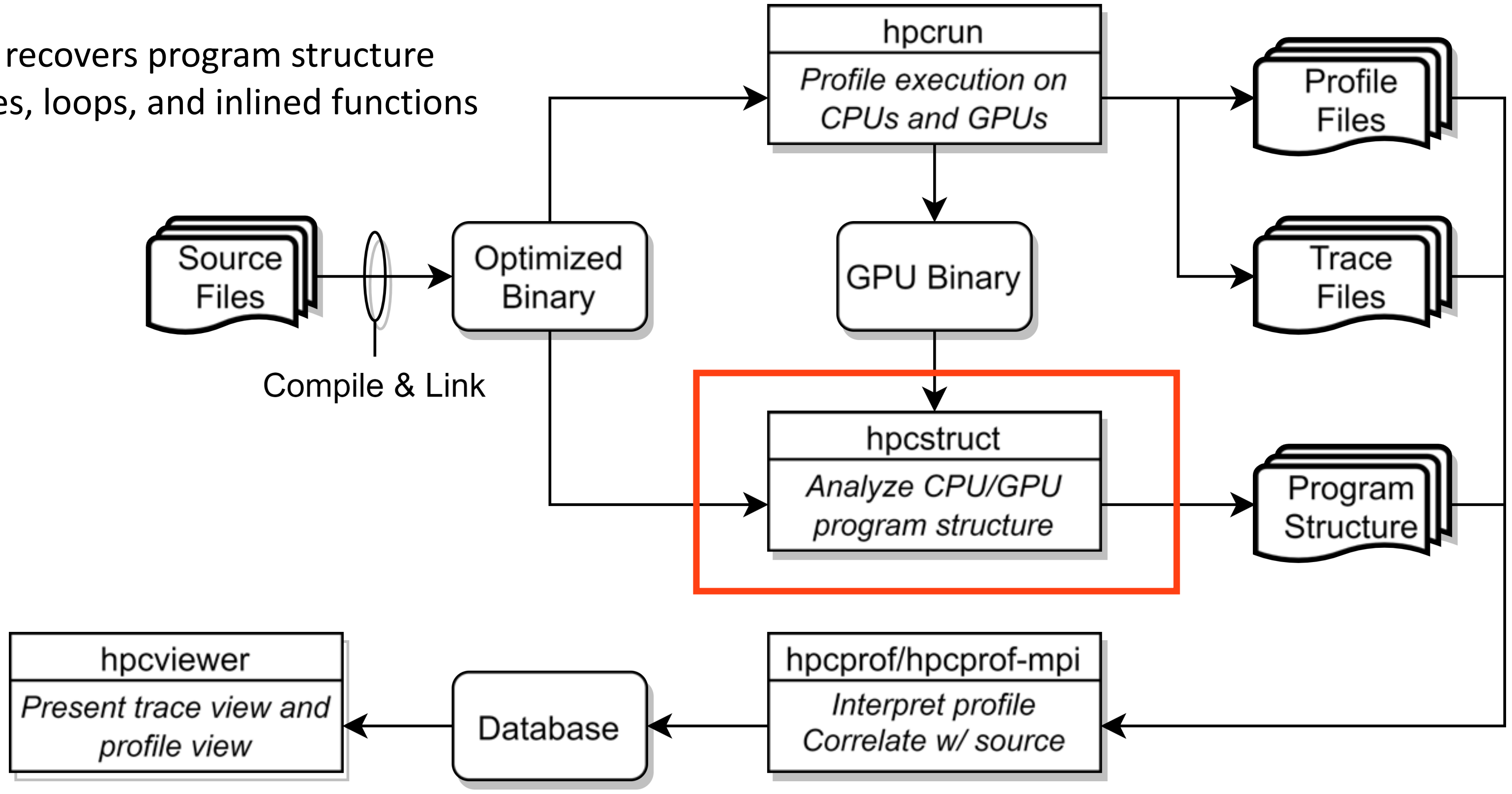
- CPU and GPU Tracing (in addition to profiling)
  - `hpcrun -e CPUTIME -e gpu=xxx -t <app>`

- 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>`

HPCToolkit's Workflow for GPU-accelerated Applications

Step 3:

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



EXASCALE COMPUTING PROJECT

hpcstruct: Analyze CPU and GPU Binaries Using Multiple Threads

- Usage

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

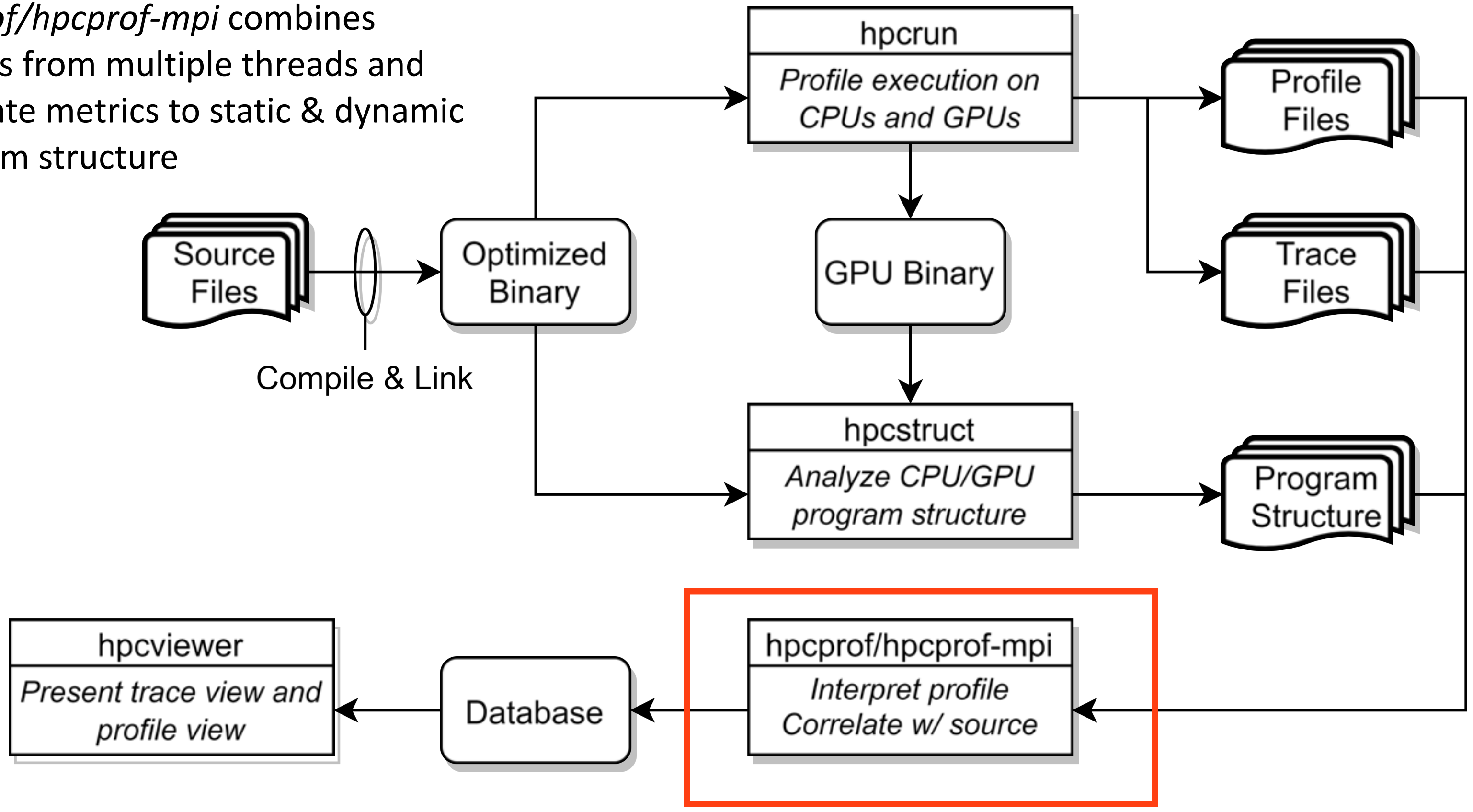
- What it does
  - Recover program structure information
    - Files, functions, inlined templates or functions, loops, source lines
  - In parallel, analyze all CPU and GPU binaries that were measured by HPCToolkit
    - —default: use size(CPU set)/2 threads
    - —analyze large application binaries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

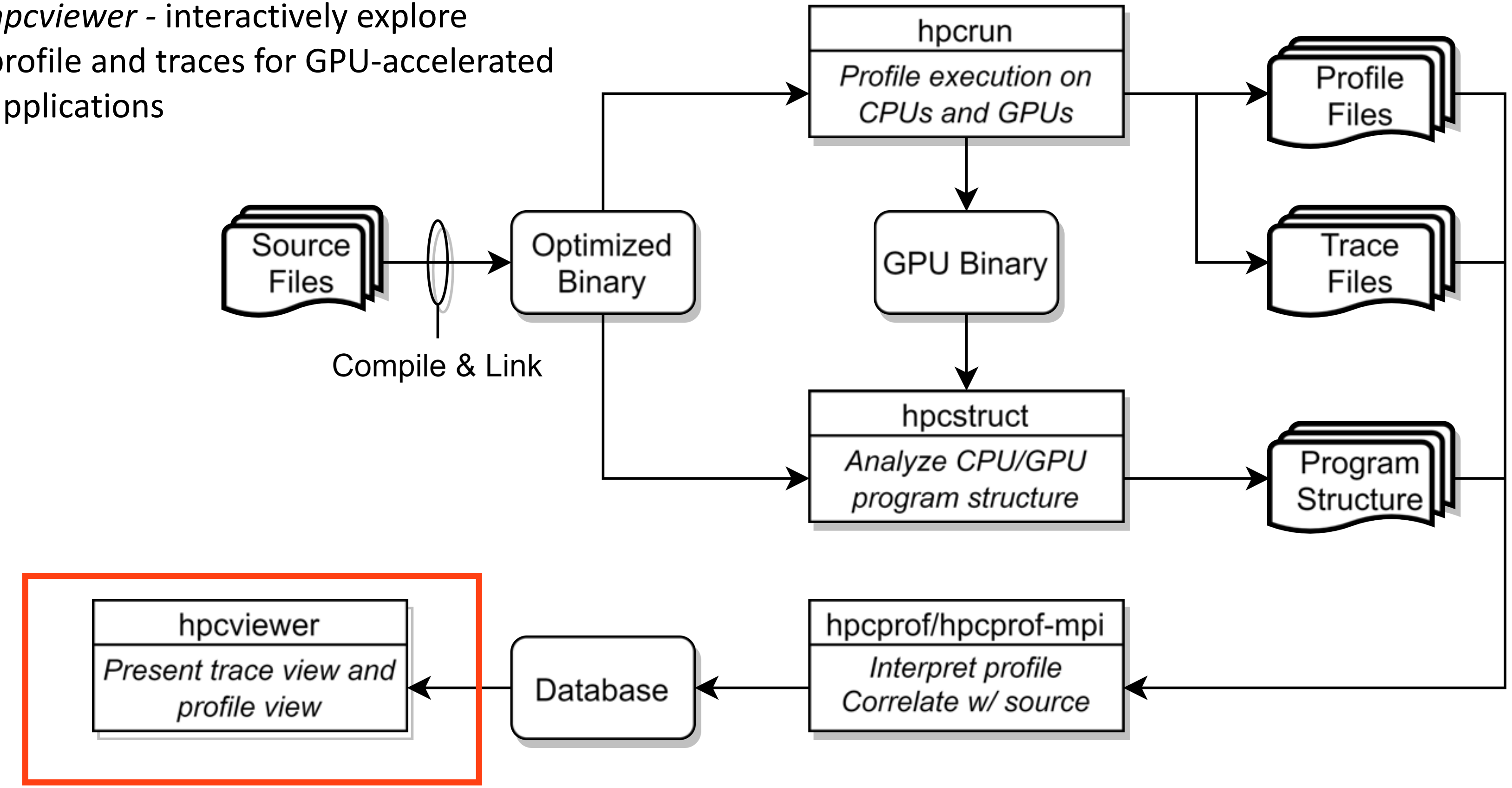
- Analyze data from large executions with distributed-memory parallelism + multithreading

```
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

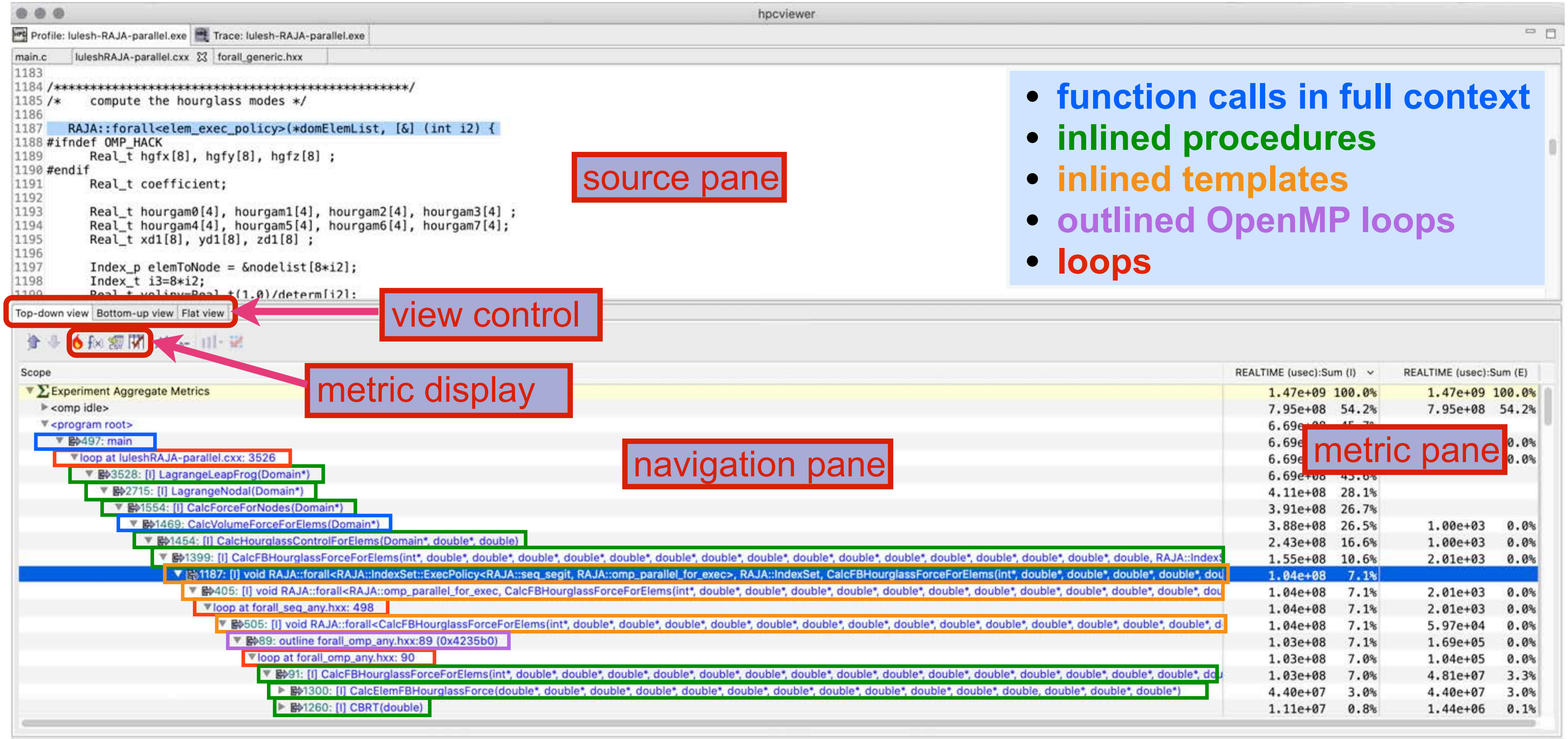
Step 4:

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



EXASCALE COMPUTING PROJECT

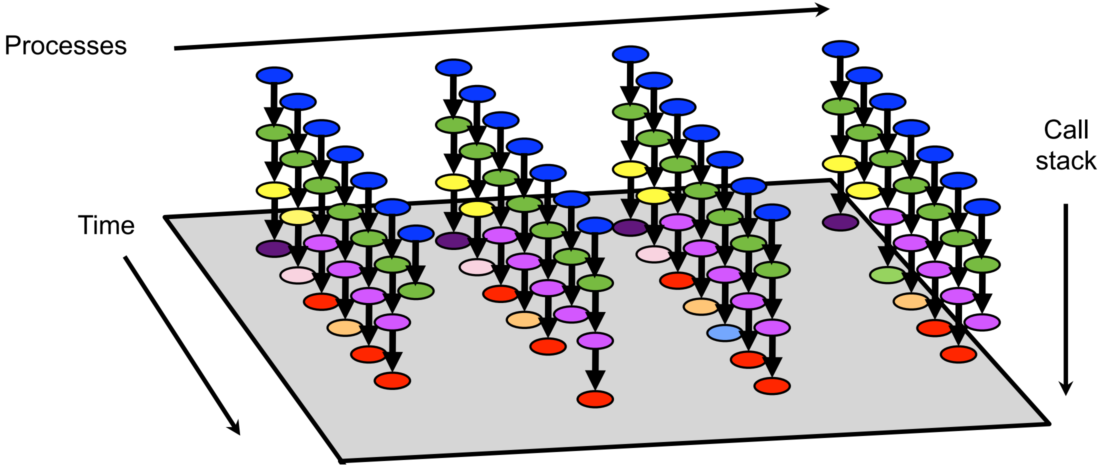
Code-centric Analysis with hpcviewer



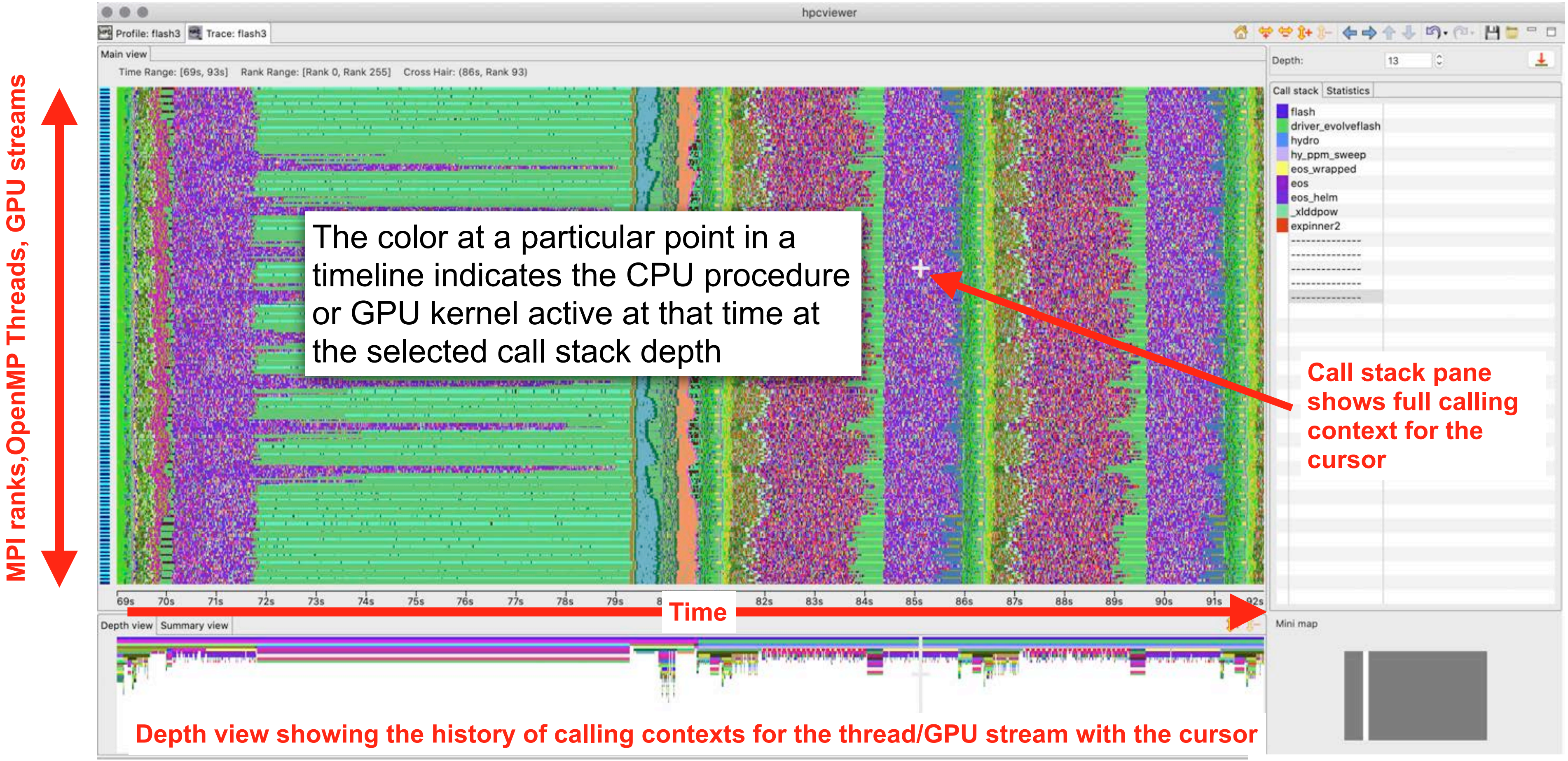
- function calls in full context
- inlined procedures
- inlined templates
- outlined OpenMP loops
- loops

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



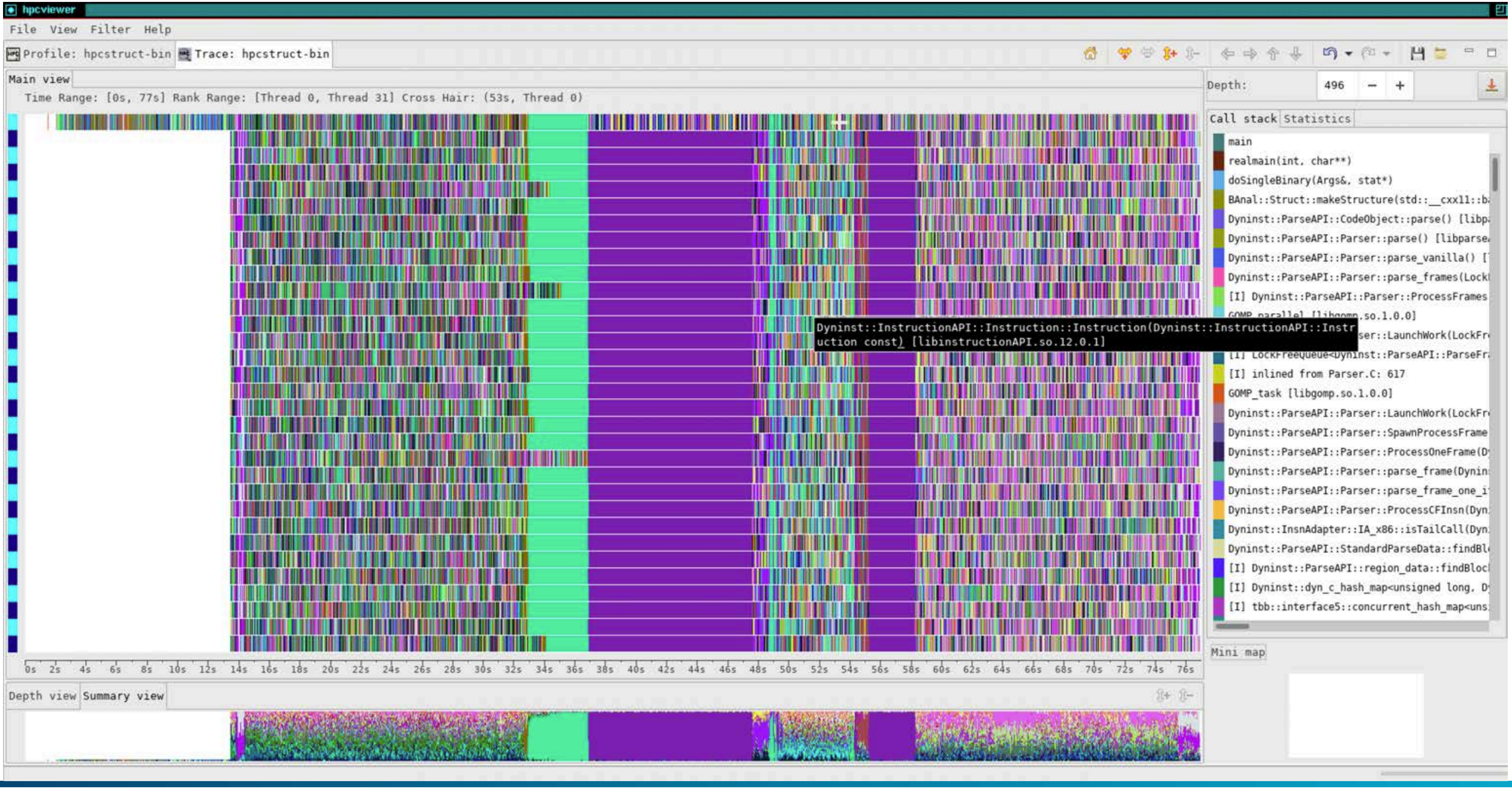
Time-centric Analysis with hpcviewer



Minimap indicates part of execution trace shown

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

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



Case Studies

- GAMESS - an ab initio quantum chemistry package: Fortran + MPI + OpenMP offloading
- GEM - a gyrokinetic turbulence code that simulates both ions and electrons

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

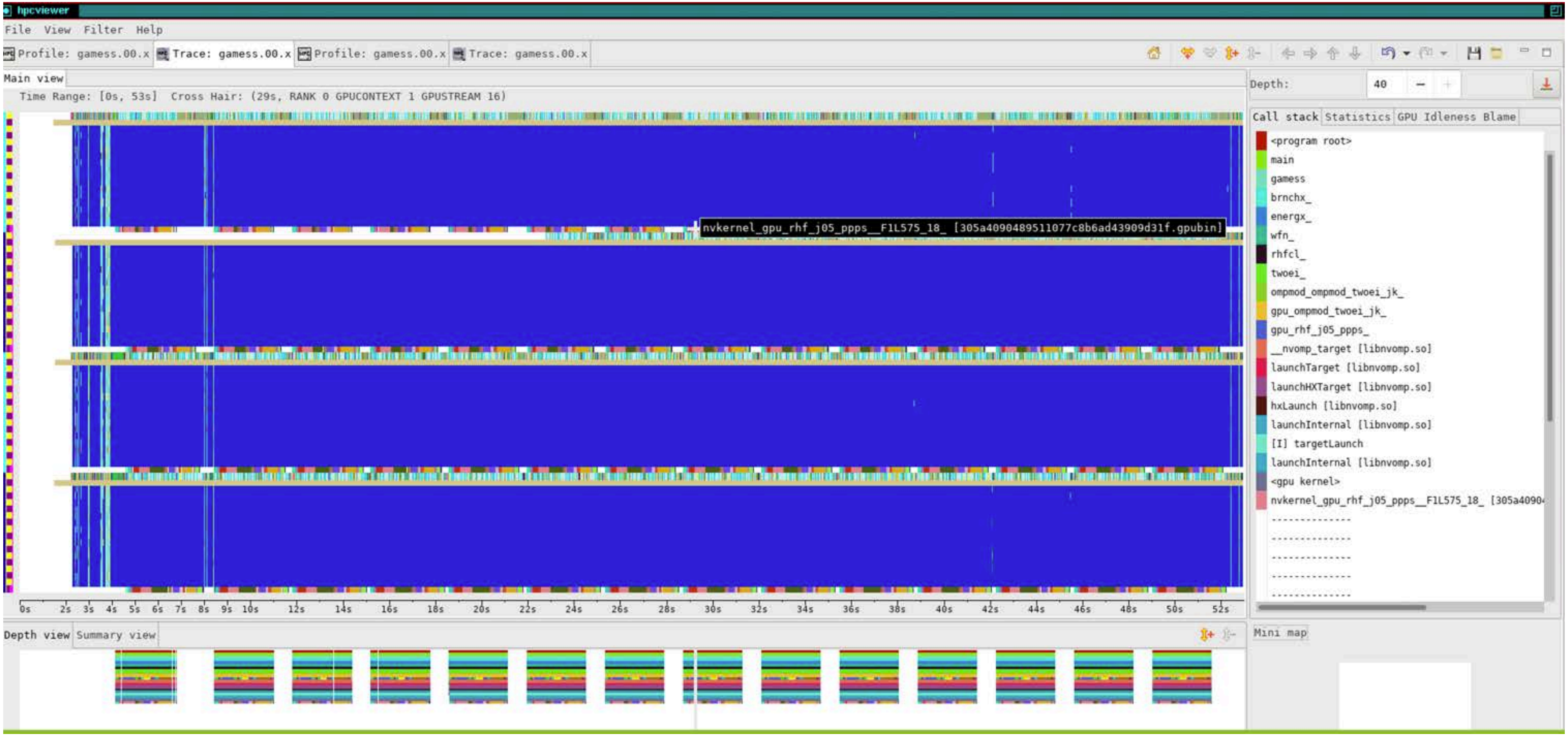
- Experiments
  - GPU-accelerated nodes at a Perlmutter hackathon
    - Single node with 4 GPUs
    - Five nodes with 20 GPUs

Perlmutter node at a glance

AMD Milan CPU
4 NVIDIA A100 GPUs
256 GB memory

EXASCALE COMPUTING PROJECT

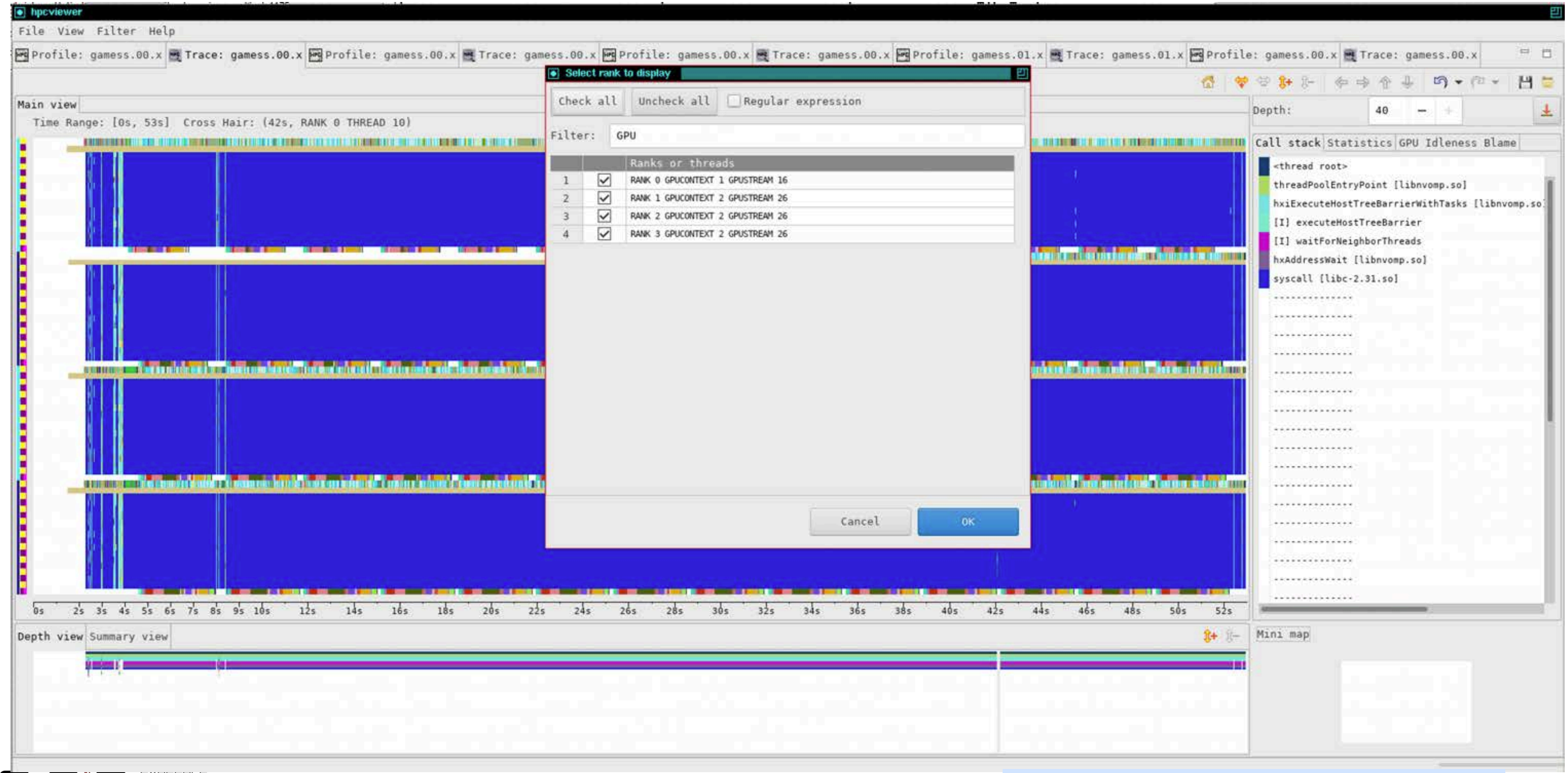
Time-centric Analysis: GAMESS 4 ranks, 4 GPUs on Perlmutter



GAMESS original

All CPU threads and GPU streams

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



GAMESS original

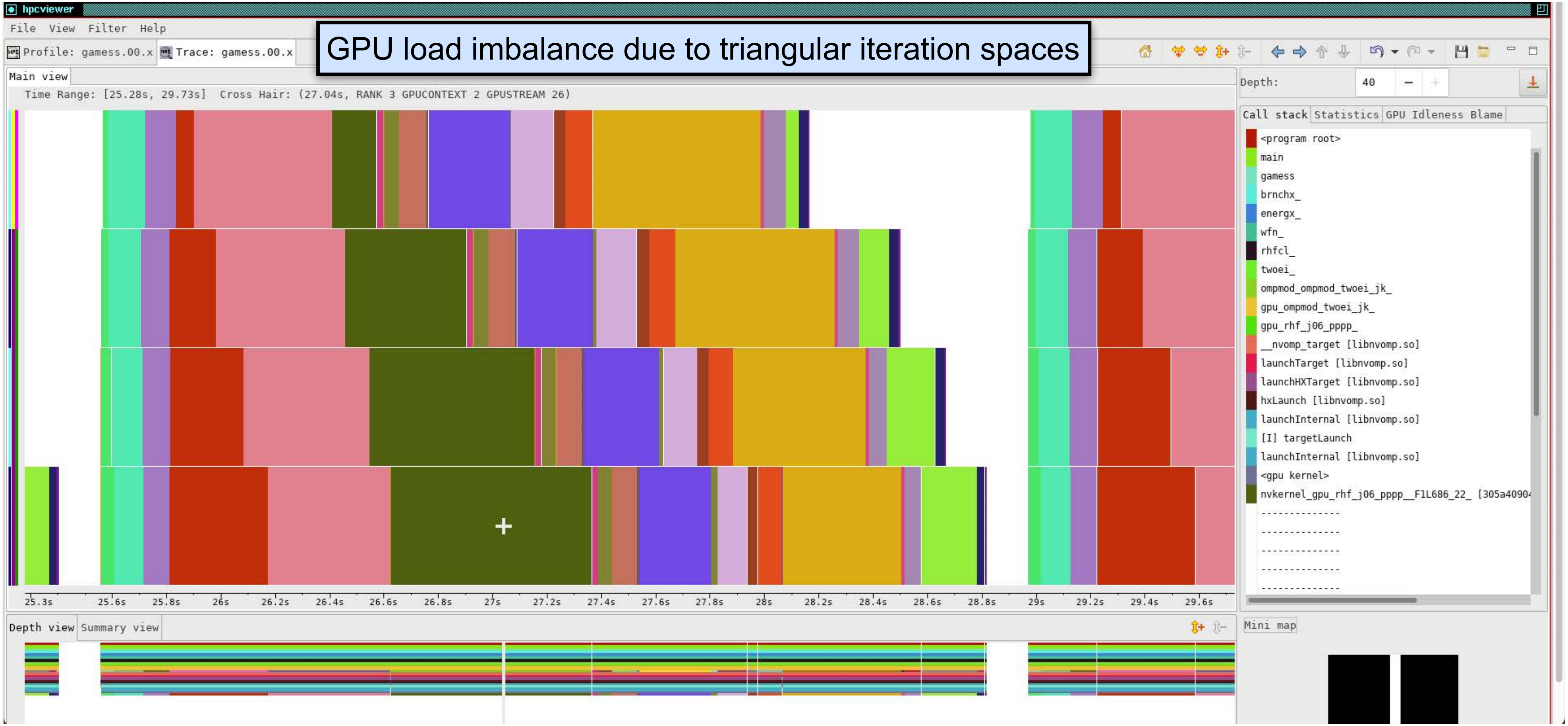
All CPU threads and GPU streams

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

GAMESS original

All GPU streams, whole execution

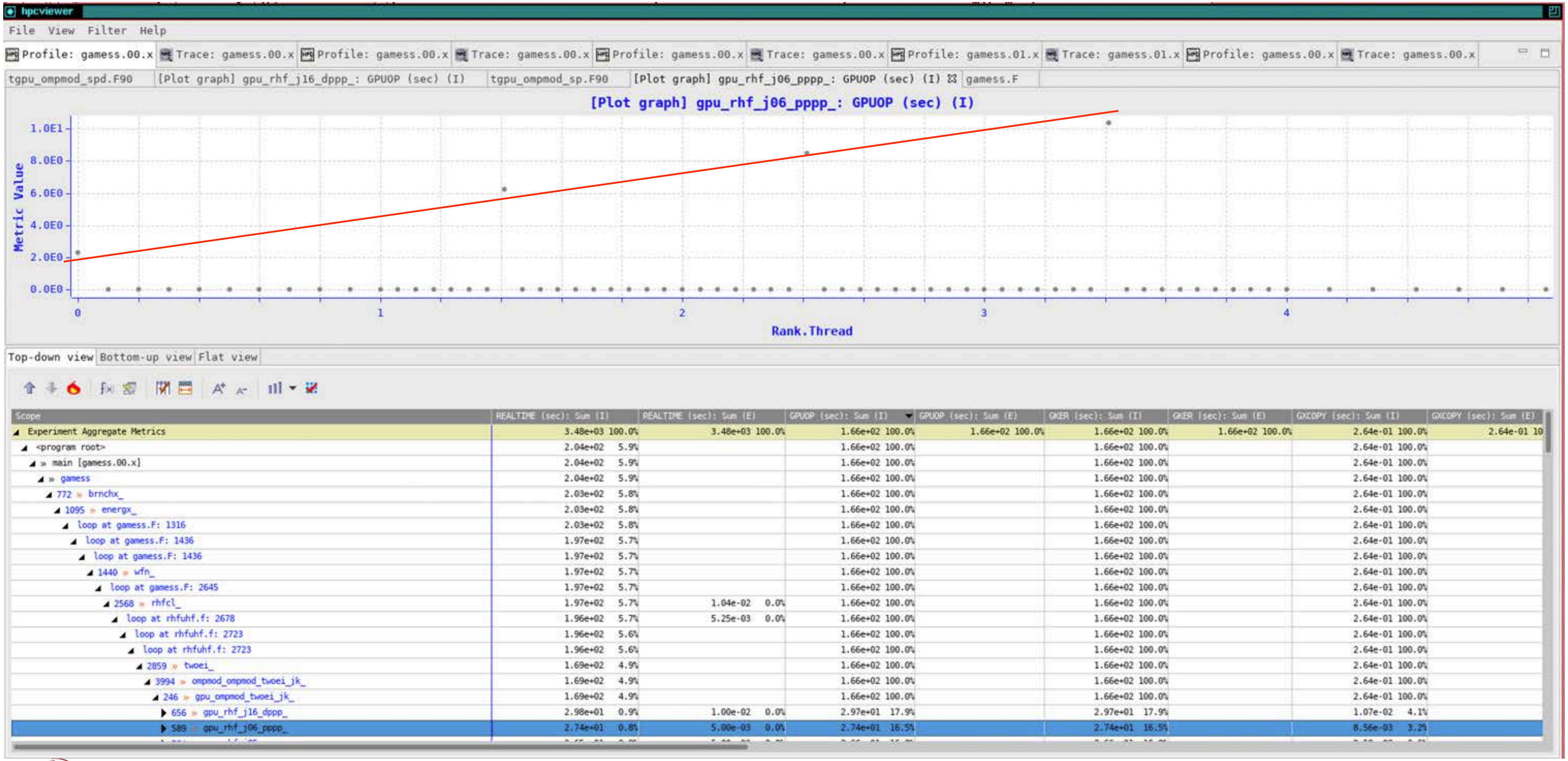
Time-centric Analysis: GAMESS 4 ranks, 4 GPUs on Perlmutter



GAMESS original

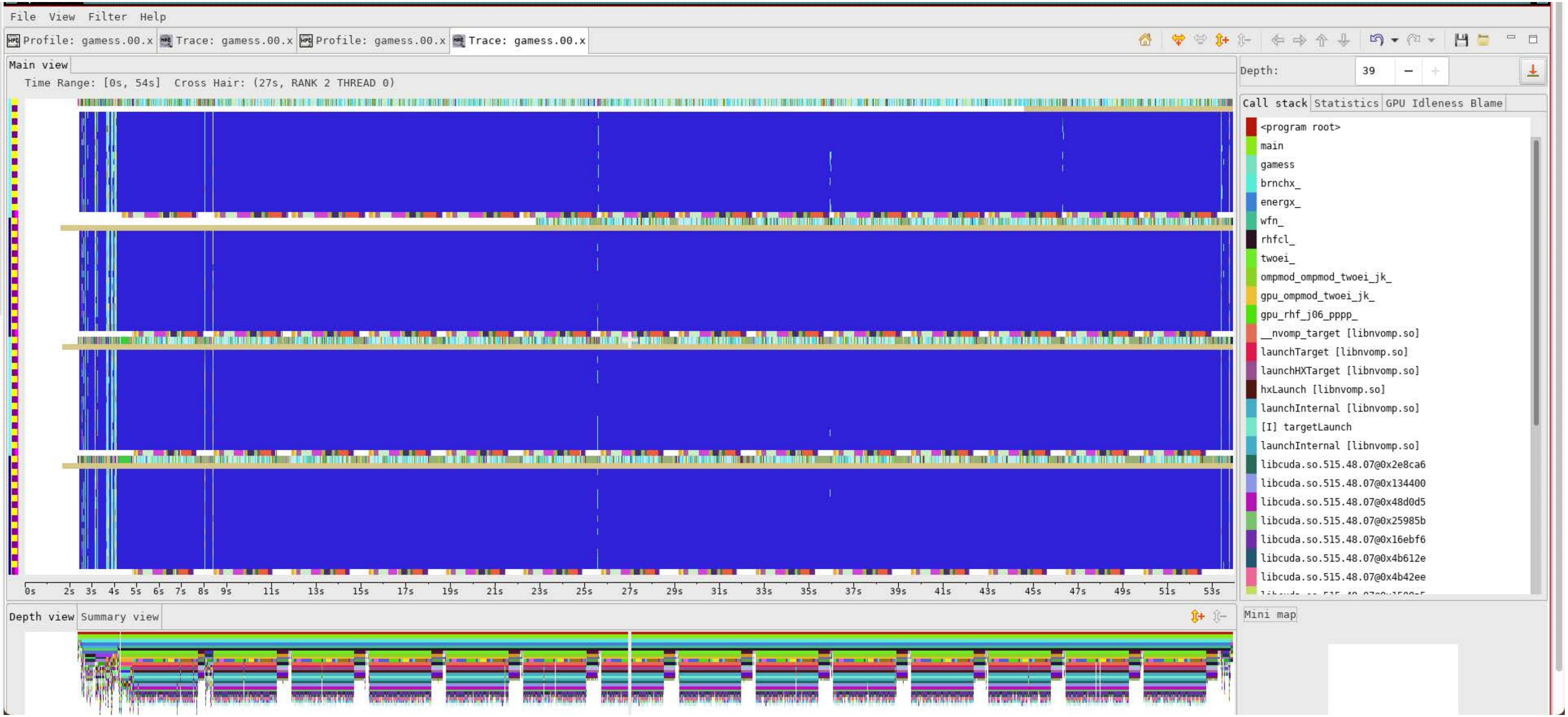
GPU streams: 1 iteration

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



GAMESS original

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

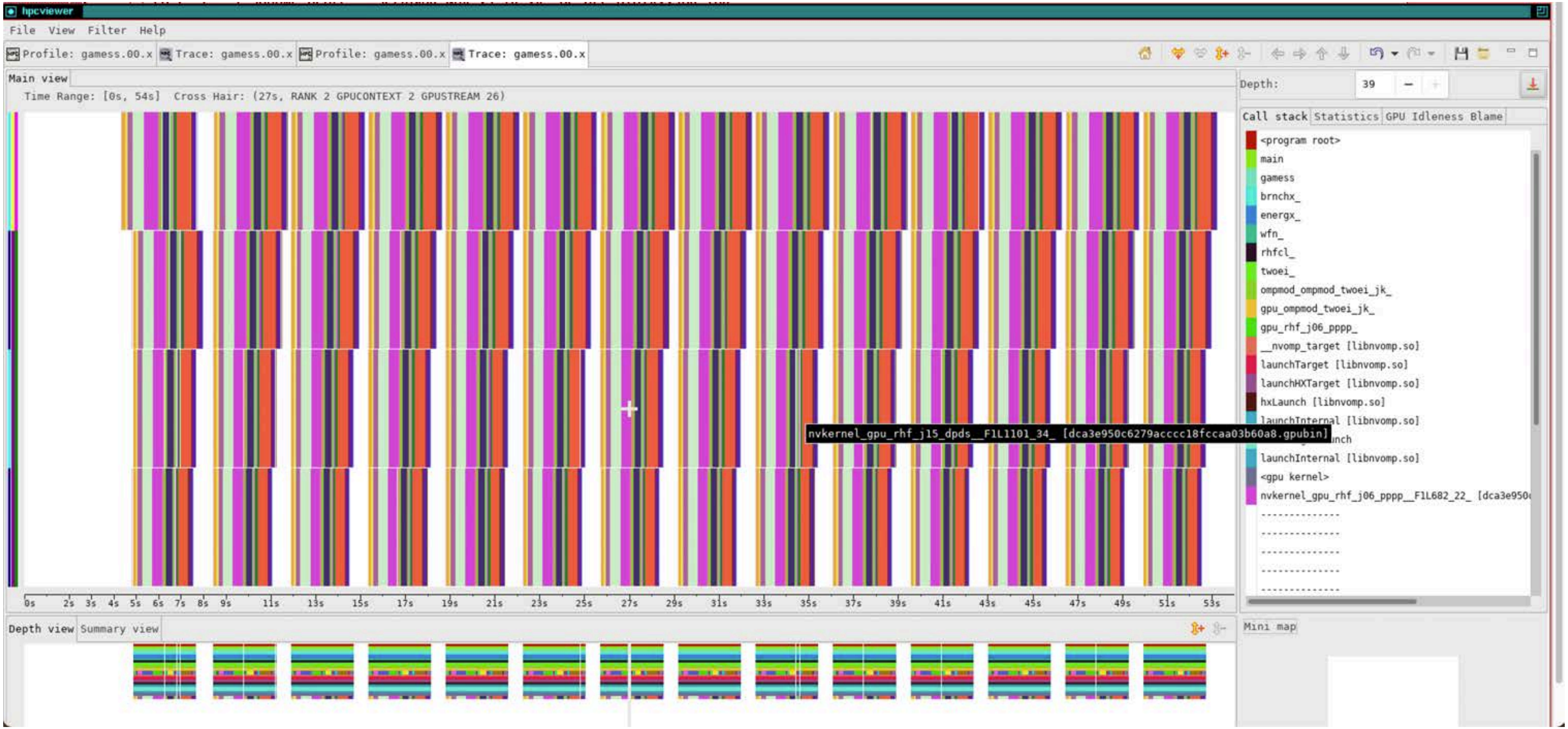


GAMESS improved

All CPU threads and GPU streams

EXASCALE COMPUTING PROJECT

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



GAMESS improved

All GPU streams, whole execution

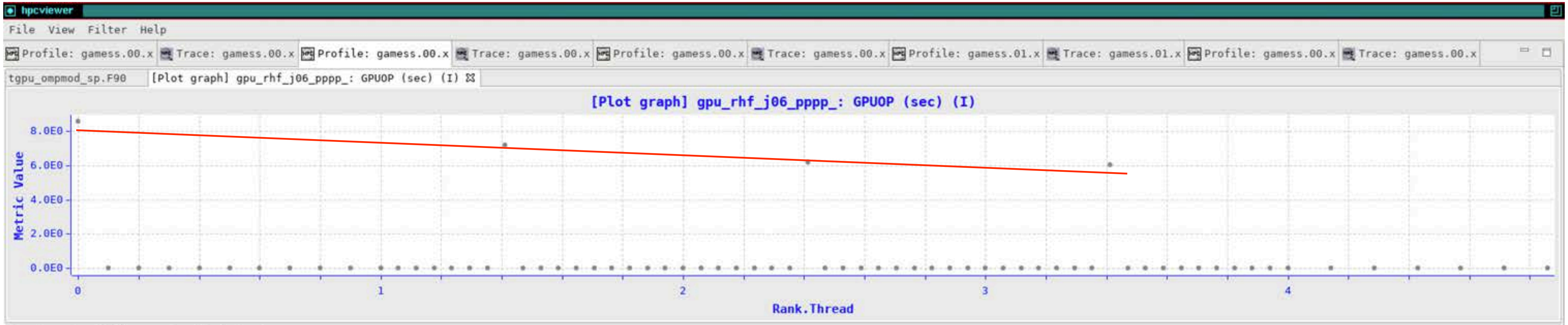
Time-centric Analysis: GAMESS 4 ranks, 4 GPUs on Perlmutter

EXASCALE COMPUTING PROJECT

GAMESS improved

All GPU streams: 2 iterations

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



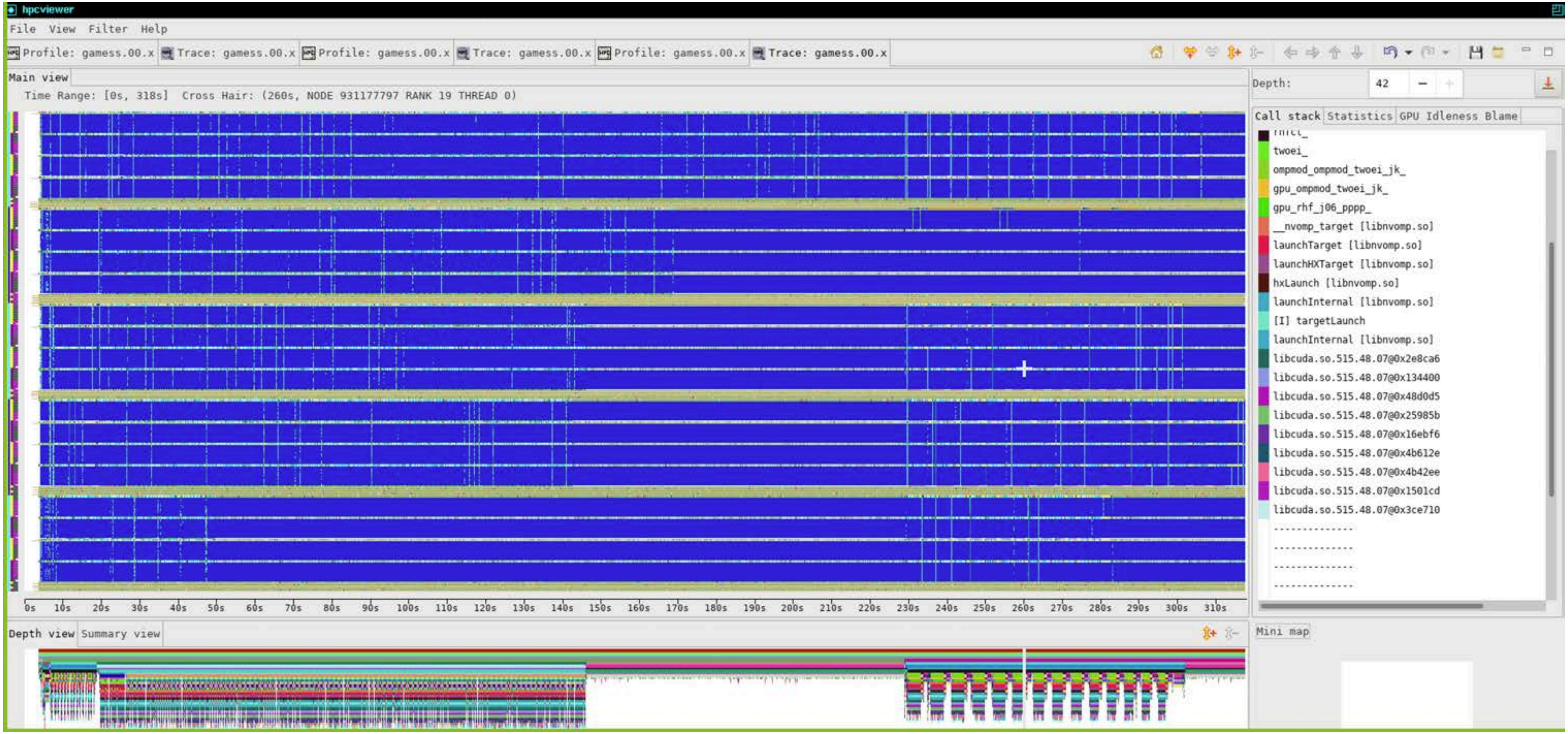
Top-down view | Bottom-up view | Flat view

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%	

EXASCALE COMPUTING PROJECT

GAMESS improved

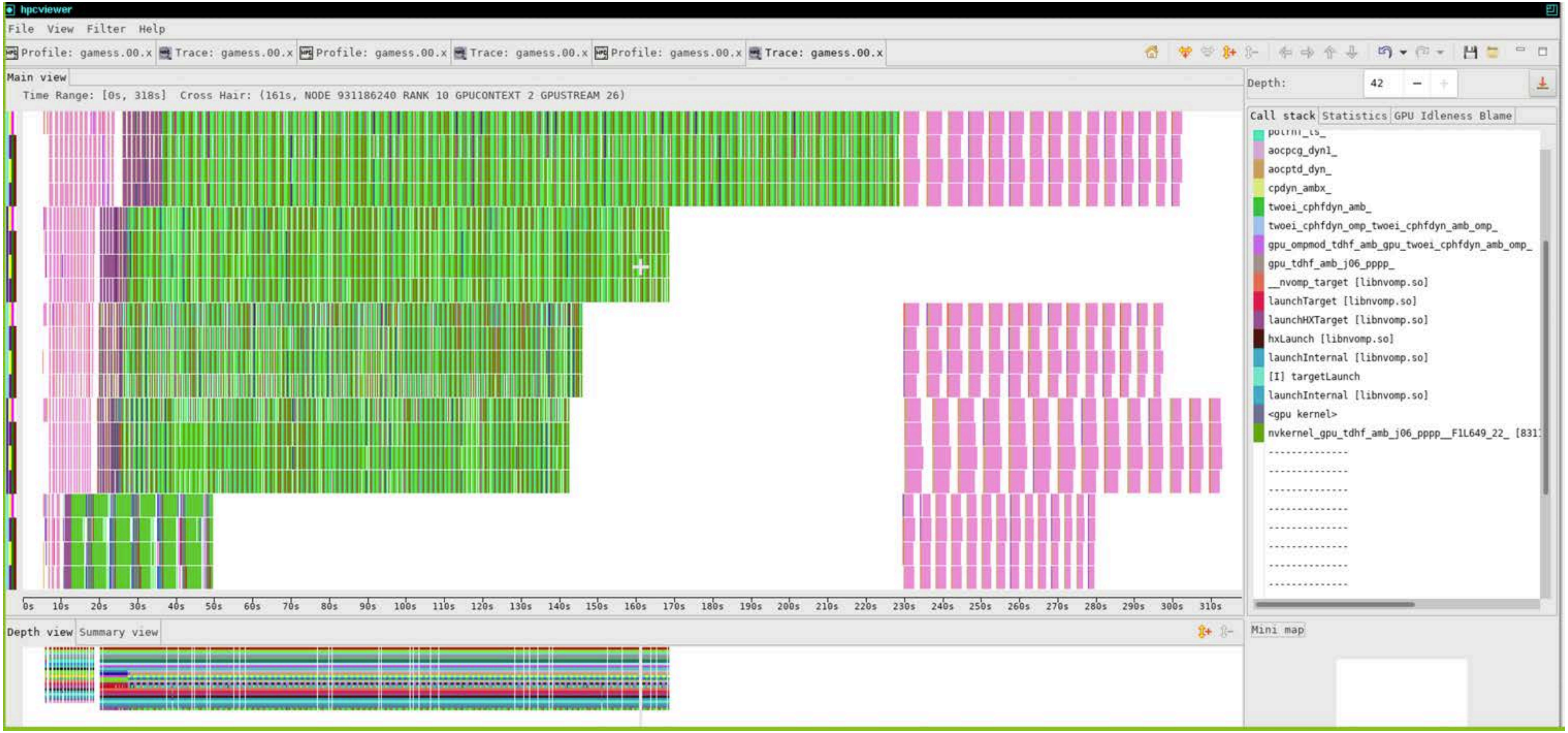
Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter



GAMESS improved

CPU Threads and GPU Streams

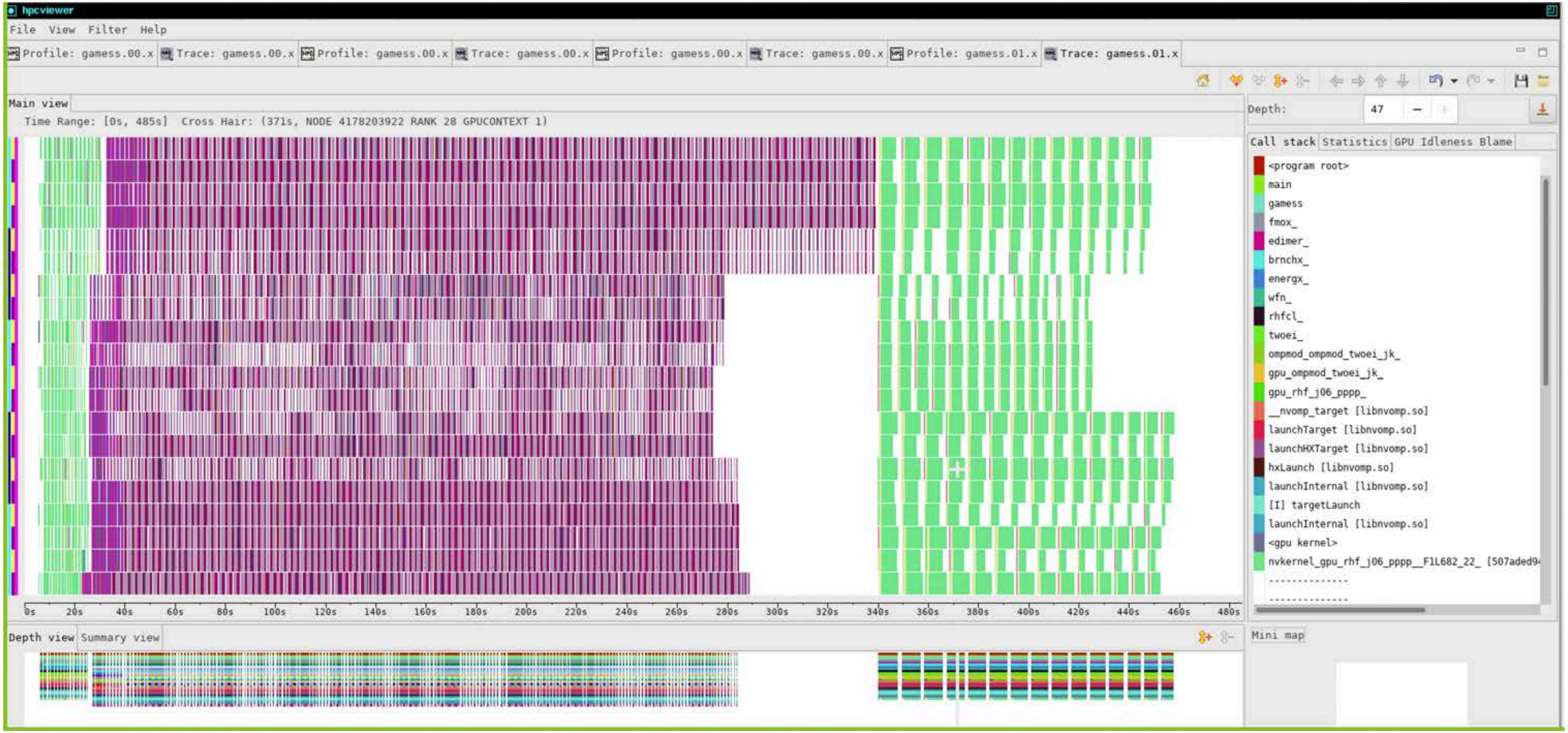
Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter



GAMESS improved

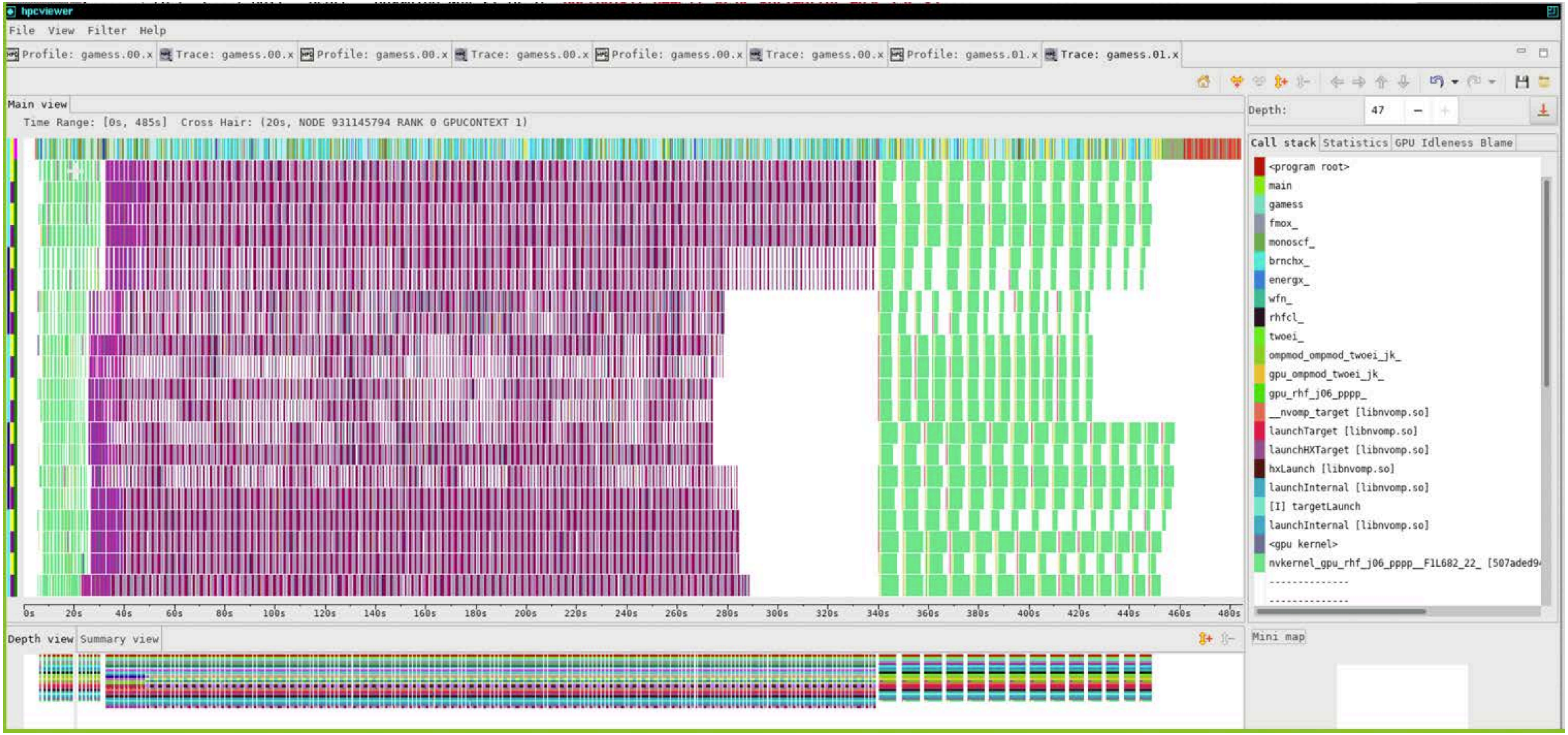
EXASCALE COMPUTING PROJECT

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



GAMESS improved with better manual distribution of work in input

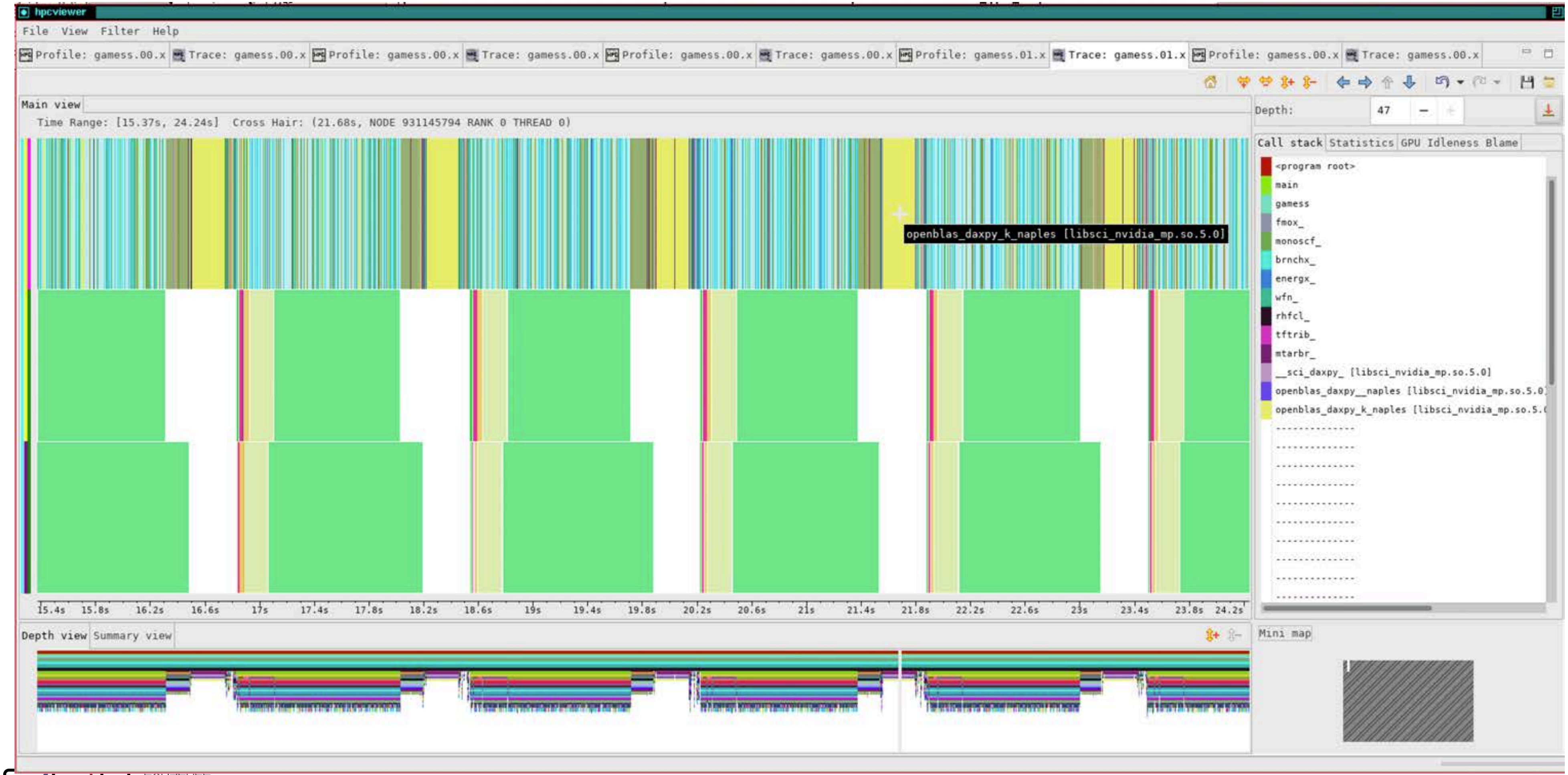
Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter



EXASCALE COMPUTING PROJECT

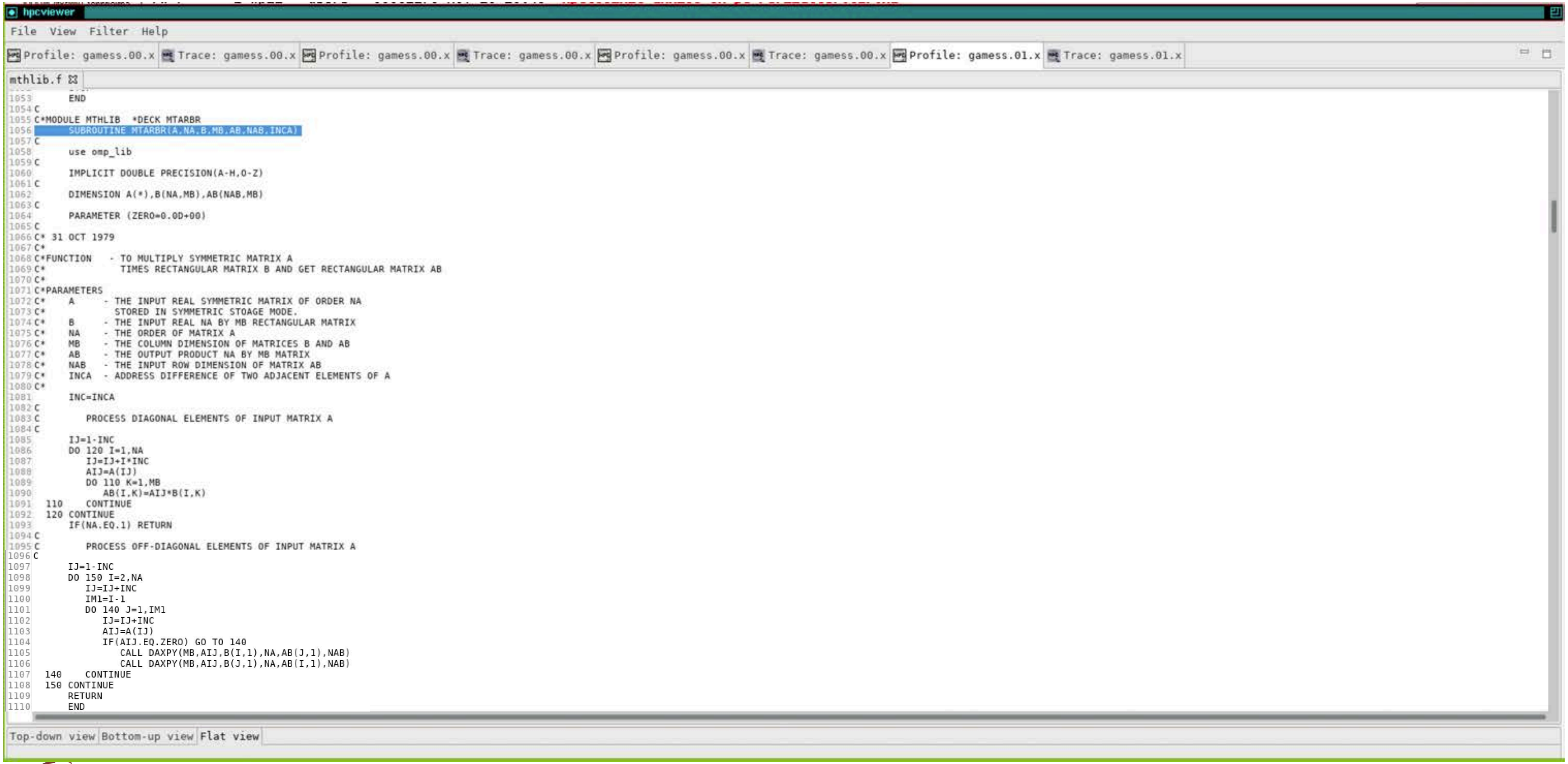
GAMESS improved adding Rank 0 Thread 0 to GPU streams

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



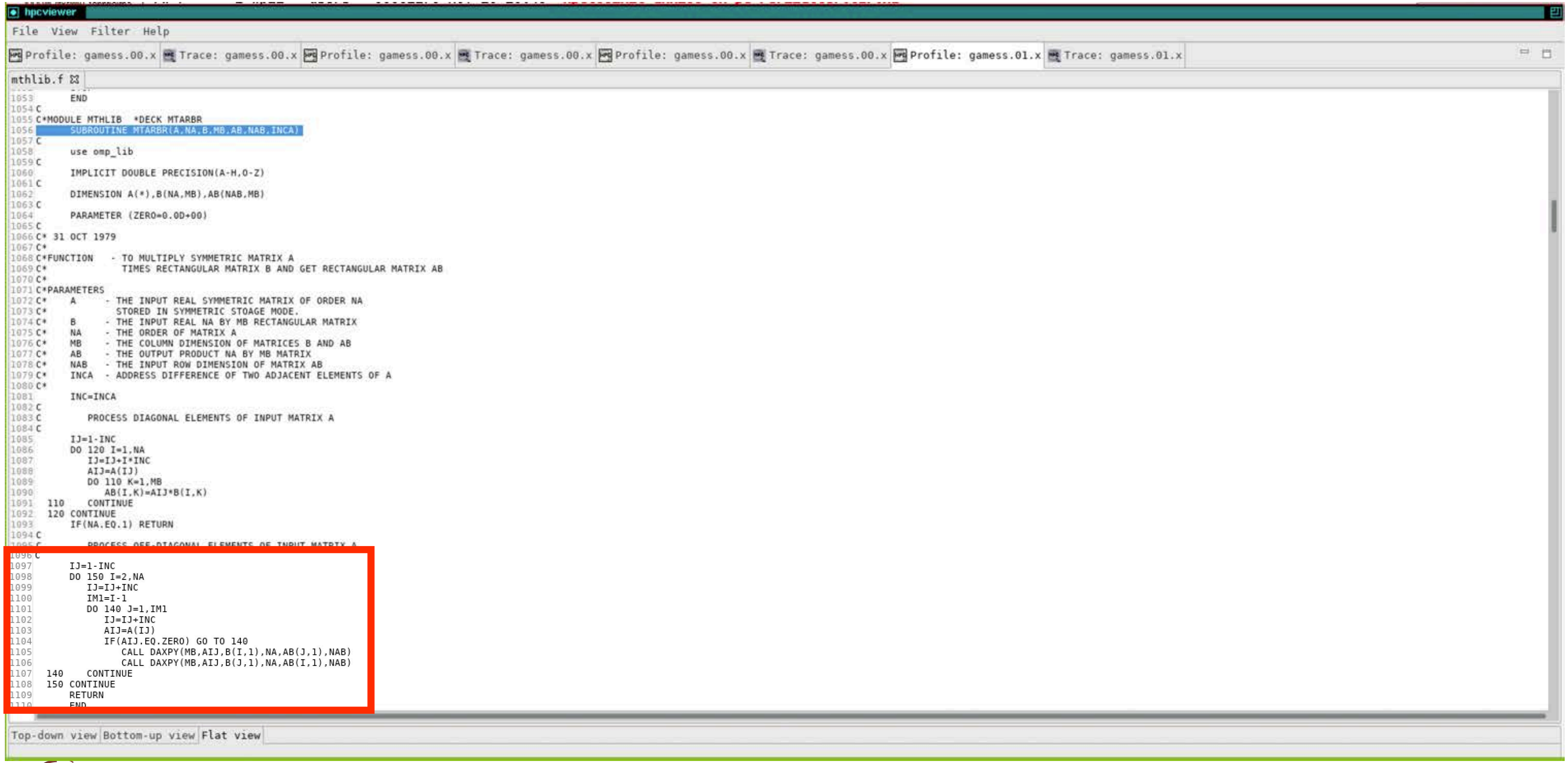
1 CPU Stream, 2 GPU Streams: 6 Iterations

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



EXASCALE COMPUTING PROJECT

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



EXASCALE COMPUTING PROJECT

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

hpcviewer

File View Filter Help

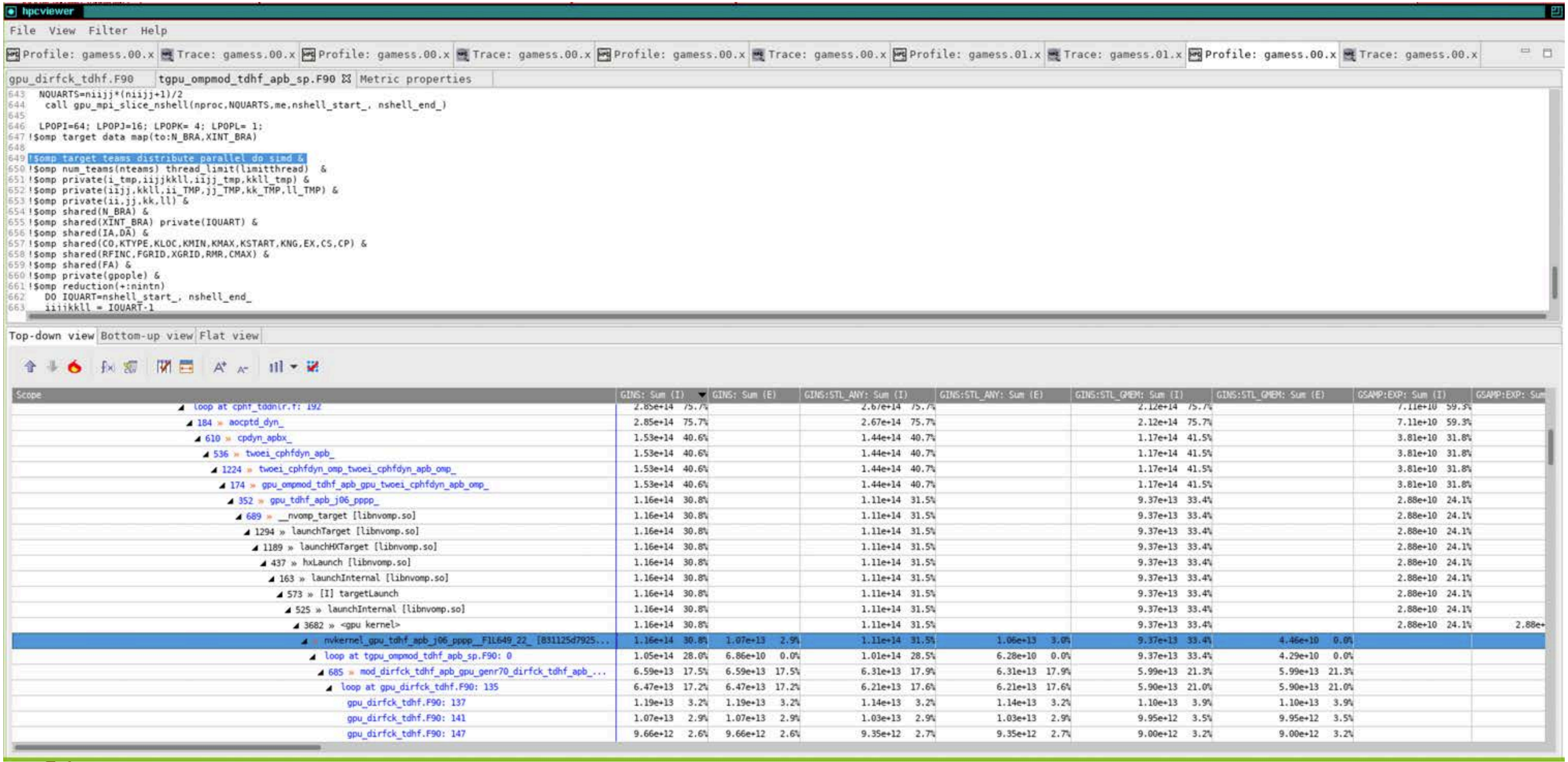
Profile: gamess.00.x | Trace: gamess.00.x | Profile: gamess.00.x | Trace: gamess.00.x | Profile: gamess.00.x | Trace: gamess.00.x | Profile: gamess.01.x | Trace: gamess.01.x

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

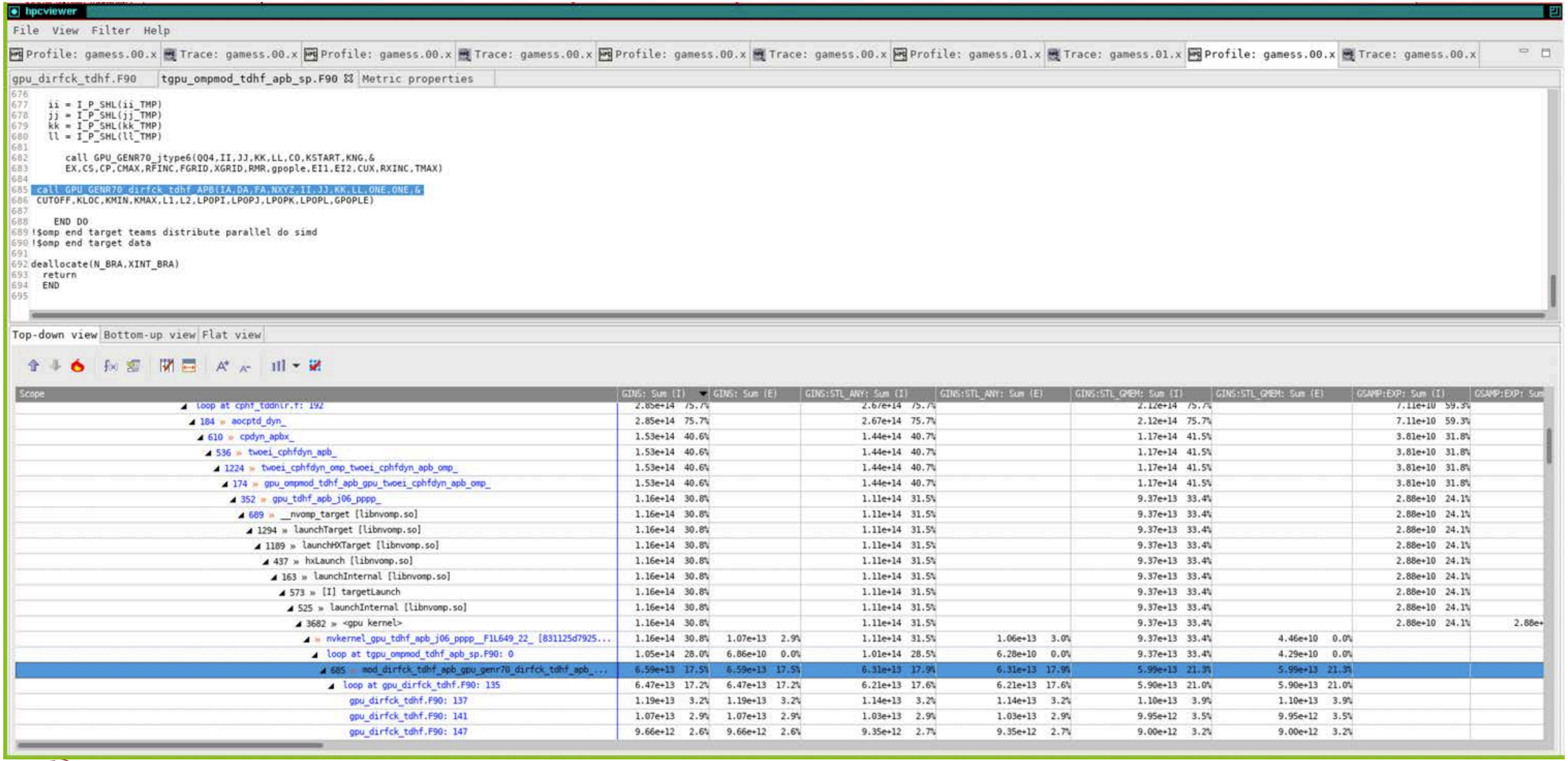
Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter



EXASCALE COMPUTING PROJECT

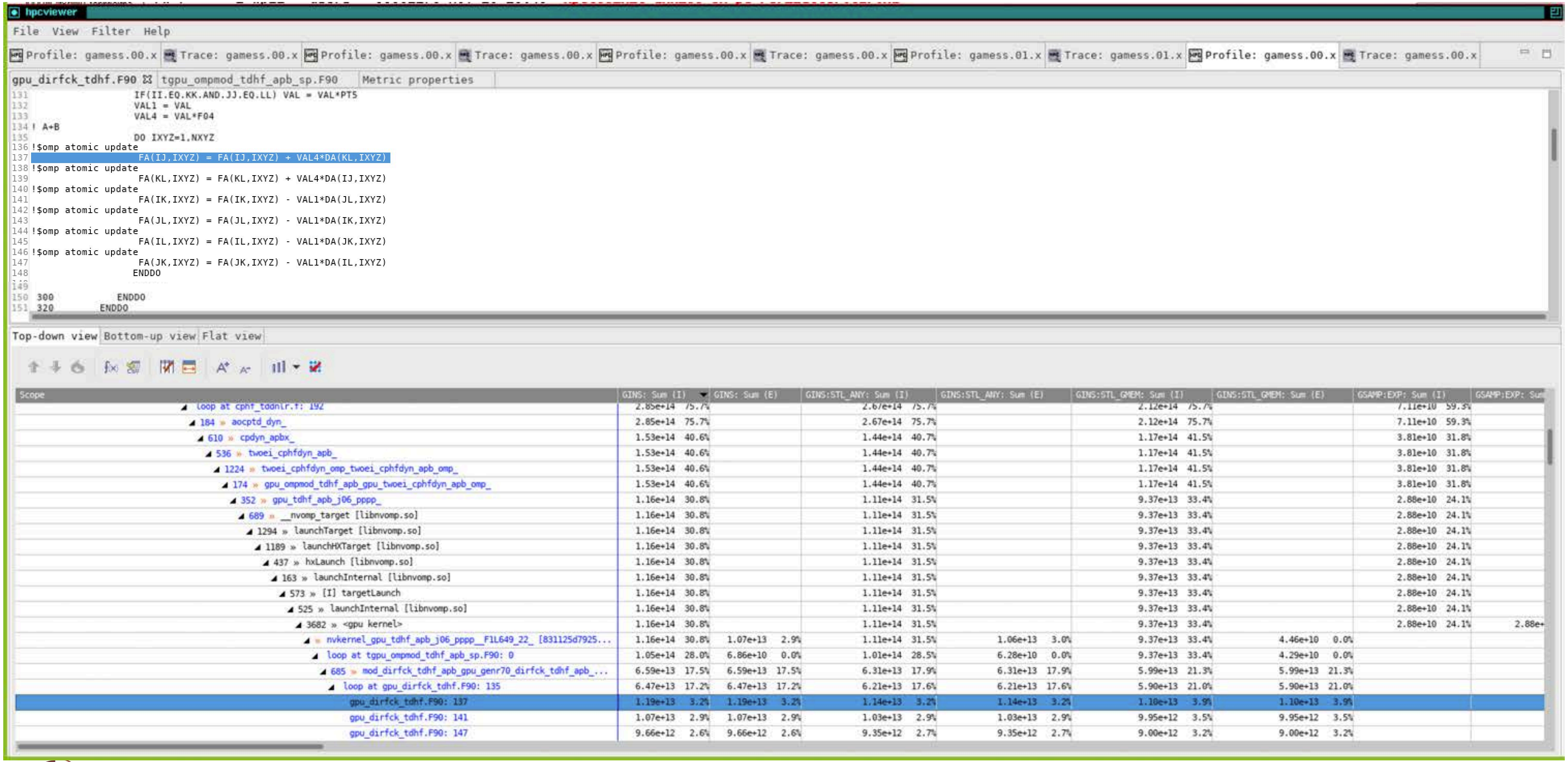
GAMESS improved with PC Sampling

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



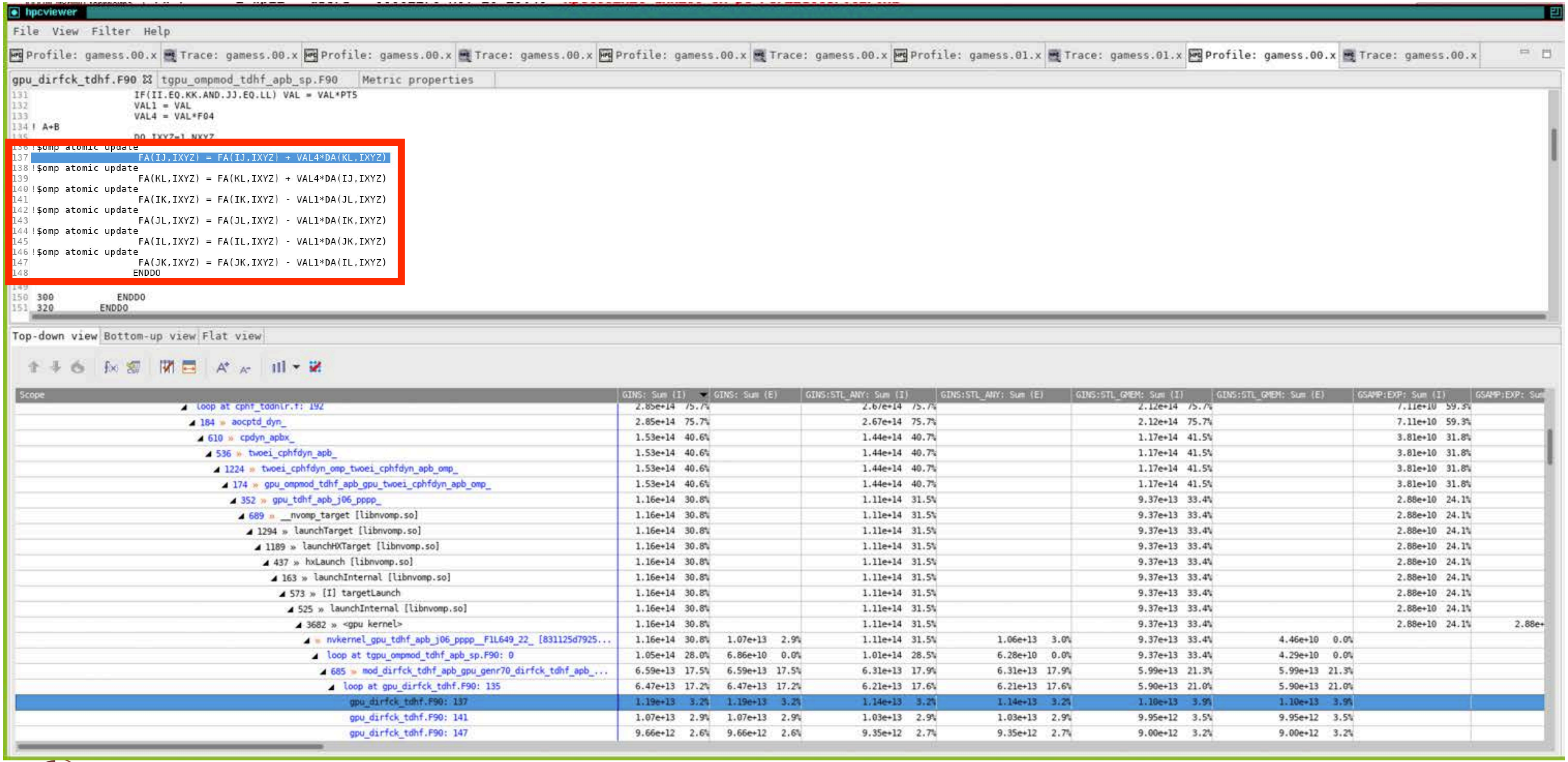
EXASCALE COMPUTING PROJECT

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

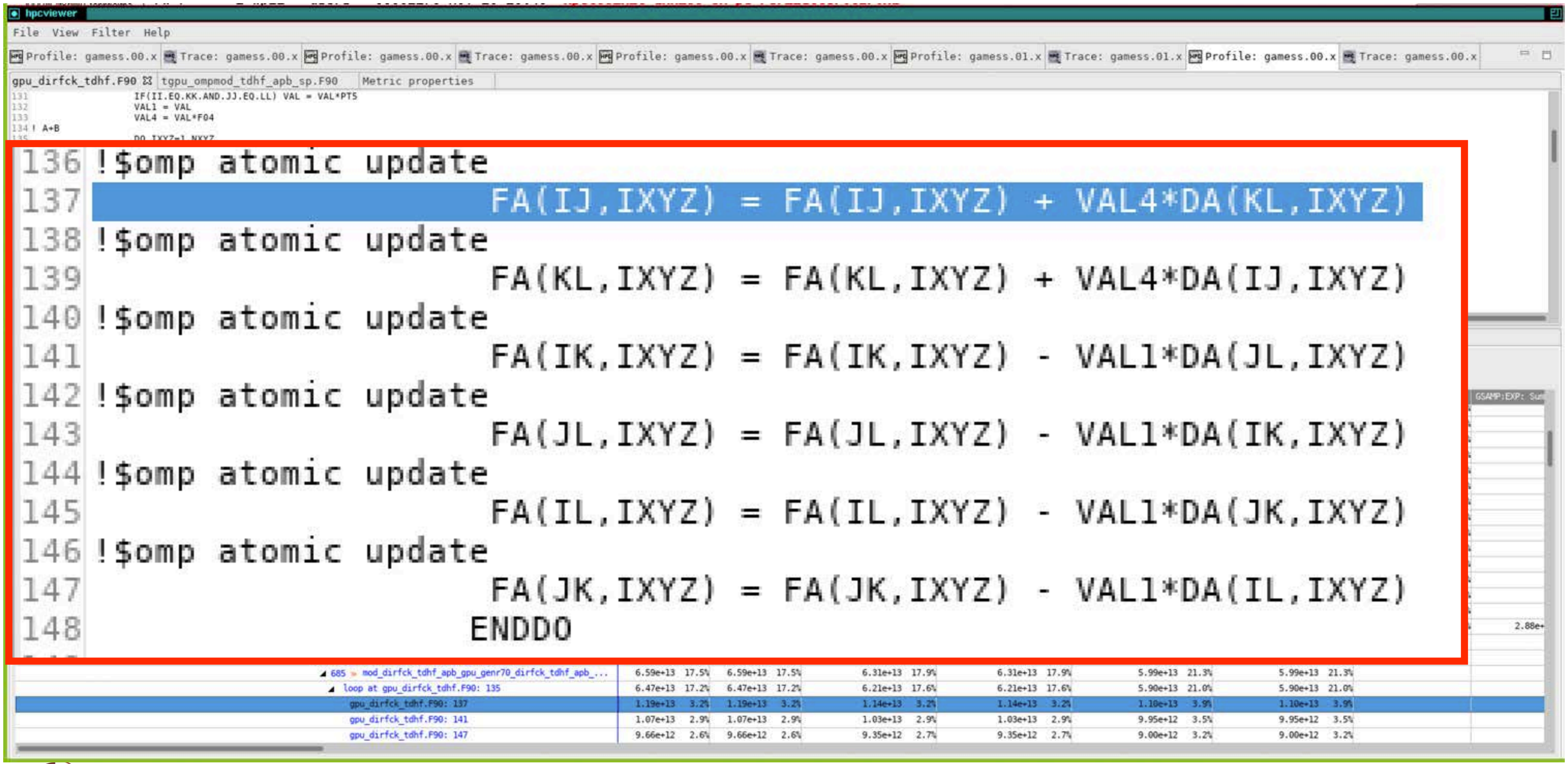


EXASCALE COMPUTING PROJECT

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



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



Case Study: GEM (Gyrokinetic Turbulence Code)

- GEM: a comprehensive electromagnetic delta-f particle-in-cell code that includes the full dynamics of gyrokinetic ions and drift-kinetic electrons
  - Developed by University of Colorado at Boulder, part of ECP WDMApp project
- Code is written in Fortran 90 + MPI + OpenACC, with ongoing porting efforts to OpenMP target offload (https://dl.acm.org/doi/abs/10.1007/978-3-030-97759-7_7)
- Tested platforms: Perlmutter, Crusher, and Frontier using Cray compiler
  - Frontier: 16 nodes, 8 MPI ranks per node, 4 OpenMP threads per rank, 1 GPU per rank, 2 GPU streams per GPU device

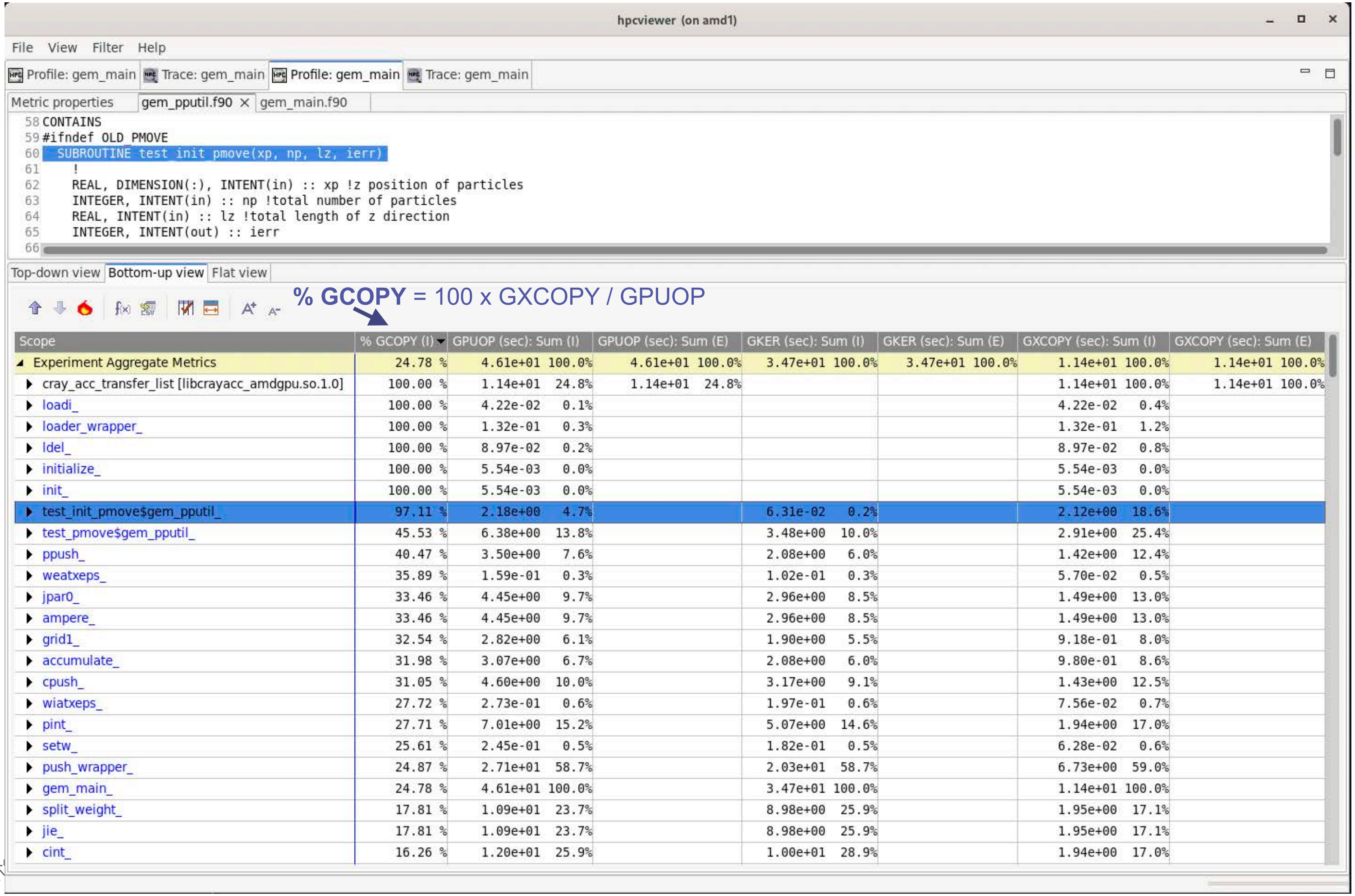
Frontier	Wall-clock Time	Speedup
Without GPU offloading	290.88s	1 (base)
Naive GPU offloading	41.80s	6.96
Optimized GPU offloading	39.52s	7.36

First attempt: not all parallel loops should be offloaded

Rank 0 Thread 0
OpenMP threads
GPU streams

Too much data movement between CPU & GPU

First attempt: not all parallel loops should be offloaded




```
CONTAINS
#ifndef OLD_PMOVE
  SUBROUTINE test_init_pmove(xp, np, lz, ierr)
    !
    REAL, DIMENSION(:), INTENT(in) :: xp !z position of particles
    INTEGER, INTENT(in) :: np !total number of particles
    REAL, INTENT(in) :: lz !total length of z direction
    INTEGER, INTENT(out) :: ierr
```

Scope	% GCOPY (I)	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	24.78 %	4.61e+01 100.0%	4.61e+01 100.0%	3.47e+01 100.0%	3.47e+01 100.0%	1.14e+01 100.0%	1.14e+01 100.0%
cray_acc_transfer_list [libcrayacc_amdgpu.so.1.0]	100.00 %	1.14e+01 24.8%	1.14e+01 24.8%			1.14e+01 100.0%	1.14e+01 100.0%
loadi_	100.00 %	4.22e-02 0.1%				4.22e-02 0.4%	
loader_wrapper_	100.00 %	1.32e-01 0.3%				1.32e-01 1.2%	
ldel_	100.00 %	8.97e-02 0.2%				8.97e-02 0.8%	
initialize_	100.00 %	5.54e-03 0.0%				5.54e-03 0.0%	
init_	100.00 %	5.54e-03 0.0%				5.54e-03 0.0%	
test_init_pmove$gem_pputil_	97.11 %	2.18e+00 4.7%		6.31e-02 0.2%		2.12e+00 18.6%	
test_pmove$gem_pputil_	45.53 %	6.38e+00 13.8%		3.48e+00 10.0%		2.91e+00 25.4%	
ppush_	40.47 %	3.50e+00 7.6%		2.08e+00 6.0%		1.42e+00 12.4%	
weatxeps_	35.89 %	1.59e-01 0.3%		1.02e-01 0.3%		5.70e-02 0.5%	
jpar0_	33.46 %	4.45e+00 9.7%		2.96e+00 8.5%		1.49e+00 13.0%	
ampere_	33.46 %	4.45e+00 9.7%		2.96e+00 8.5%		1.49e+00 13.0%	
grid1_	32.54 %	2.82e+00 6.1%		1.90e+00 5.5%		9.18e-01 8.0%	
accumulate_	31.98 %	3.07e+00 6.7%		2.08e+00 6.0%		9.80e-01 8.6%	
cpush_	31.05 %	4.60e+00 10.0%		3.17e+00 9.1%		1.43e+00 12.5%	
wiatxeps_	27.72 %	2.73e-01 0.6%		1.97e-01 0.6%		7.56e-02 0.7%	
pint_	27.71 %	7.01e+00 15.2%		5.07e+00 14.6%		1.94e+00 17.0%	
setw_	25.61 %	2.45e-01 0.5%		1.82e-01 0.5%		6.28e-02 0.6%	
push_wrapper_	24.87 %	2.71e+01 58.7%		2.03e+01 58.7%		6.73e+00 59.0%	
gem_main_	24.78 %	4.61e+01 100.0%		3.47e+01 100.0%		1.14e+01 100.0%	
split_weight_	17.81 %	1.09e+01 23.7%		8.98e+00 25.9%		1.95e+00 17.1%	
jie_	17.81 %	1.09e+01 23.7%		8.98e+00 25.9%		1.95e+00 17.1%	
cint_	16.26 %	1.20e+01 25.9%		1.00e+01 28.9%		1.94e+00 17.0%	

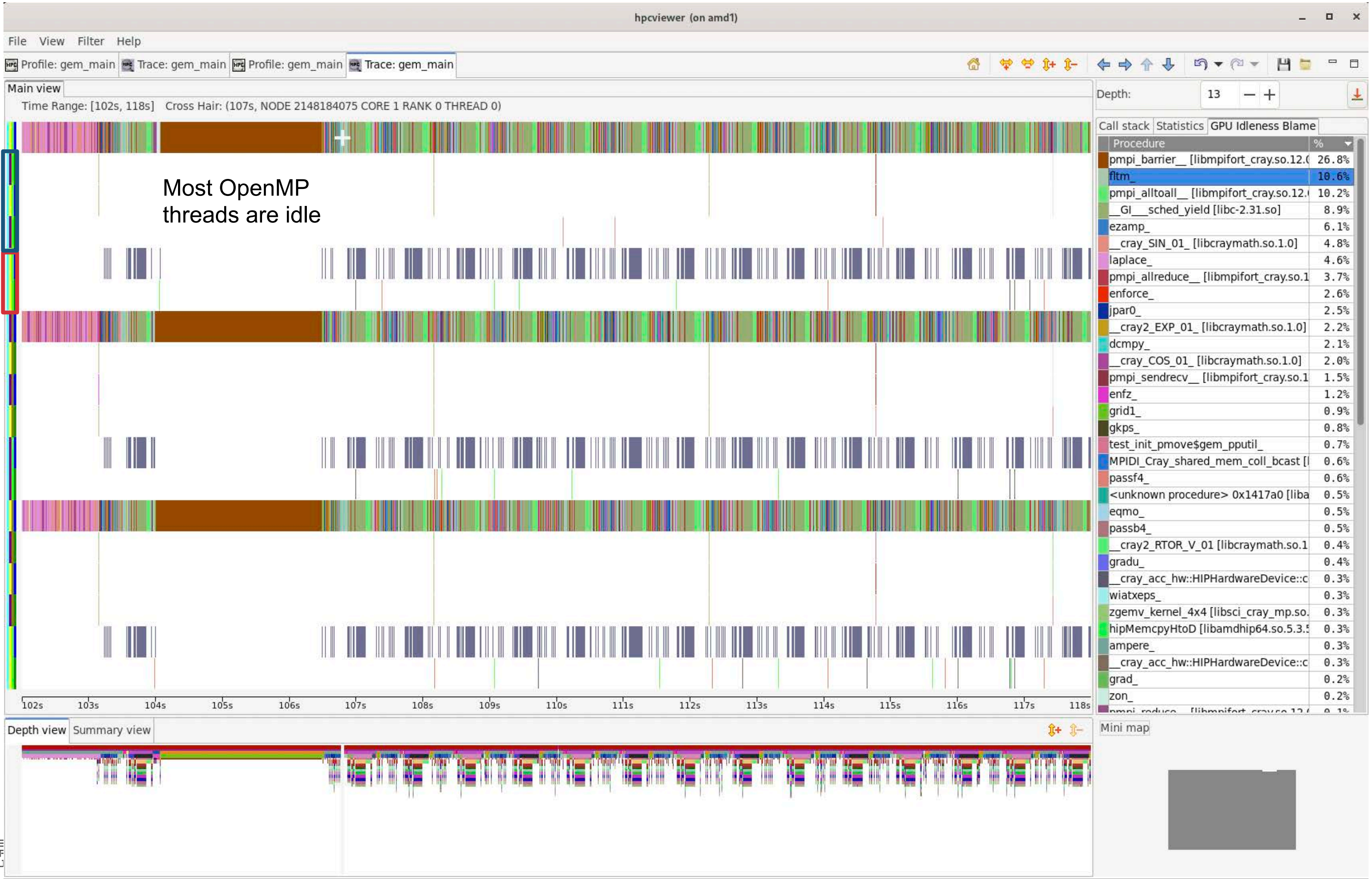
Procedures `test_init_pmove` and `test_pmove` have high data movement compared to GPU computation

Use CPU threads to reduce GPU idleness

Rank 0 Thread 0

OpenMP threads

GPU streams



10.6% of GPU idle occurs when the main CPU thread executes `fltm_` procedure.

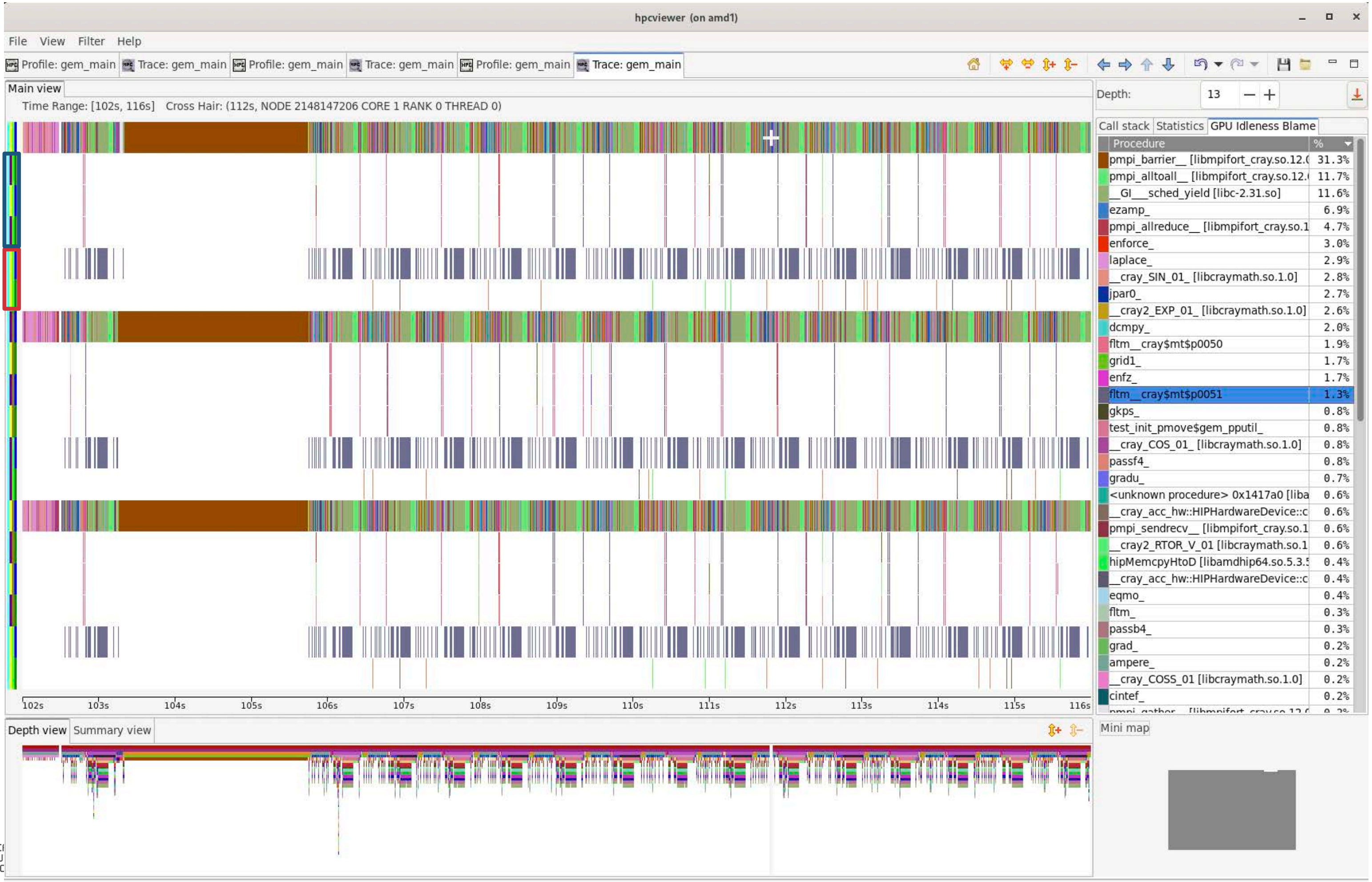
Parallelizing this procedure should reduce GPU idleness.

Final step: parallelizing `fltm_` procedure to reduce GPU idleness

Rank 0 Thread 0

OpenMP threads

GPU streams



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
  - hardware counters to measure kernels
  - 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
  - identify measurement of delays caused by GPU and communication
- Developing comprehensive support for NVTX/ROCTX/Caliper/Kokkos Labels
- Support for instruction-level measurement and attribution on AMD and Intel GPUs
- Improving the scalability of hpcprof-mpi
  - avoid unnecessary serialization of I/O
- Developing new GUI support for analysis of remote data
- Adding a Python-based interface for analysis of performance results
  - developing a Python API to support arbitrary queries and analysis of profiles and traces
  - developing a tool that presents high-level performance reports
  - exploring automated analysis to identify notable features in executions
    - e.g. load imbalance, trace line equivalence classes