Analyzing GPU-accelerated Applications Using HPCToolkit

Keren Zhou

Rice University

Outline

- **HPCToolkit GPU Overview**
- Tutorial Examples
  - Laghos
  - Quicksilver
  - PeleC
- Case Studies
  - SuperLU_DIST
  - STRUMPACK
- Summary

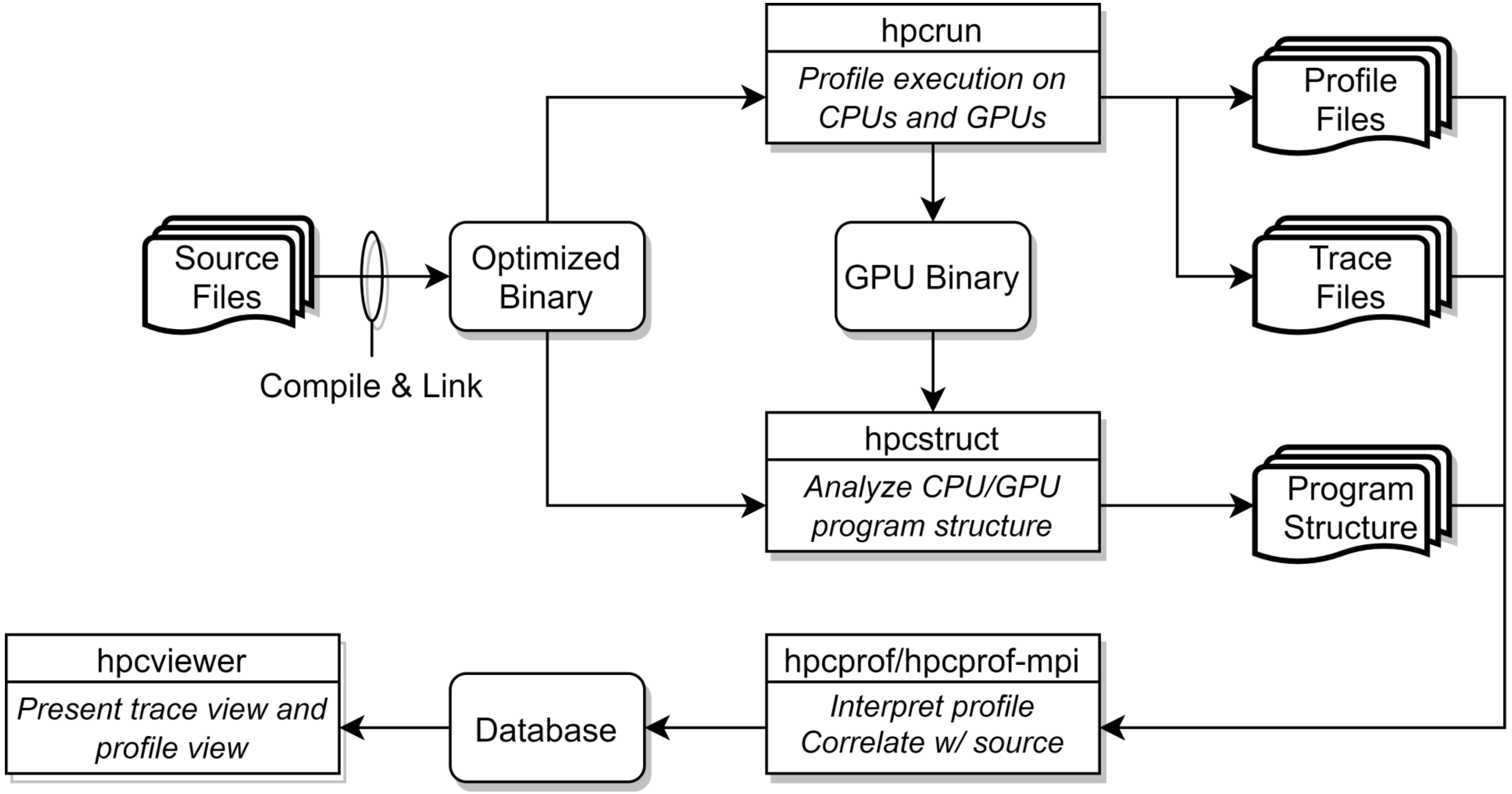
HPCToolkit GPU Highlights

- HPCToolkit support **calling context sensitive** profiling for GPU-accelerated applications
  - CPU calling context
    - Unwind at each GPU API call
  - GPU calling context
    - Reconstruct offline by analyzing GPU functions' call graphs
- Trace view
  - A series of events that happen over time on each process, thread, and GPU stream
- Profile view
  - A correlation of GPU performance metrics with full program calling contexts that span both CPU and GPU

HPCToolkit Packages on Summit and Cori

- Use CUDA < 11.2
  - CUDA 11 is recommended
- Cori
  - module load cgpu
  - module load hpctoolkit/2021.03.01-gpu
- Summit
  - module load hpctoolkit/2021.03.01

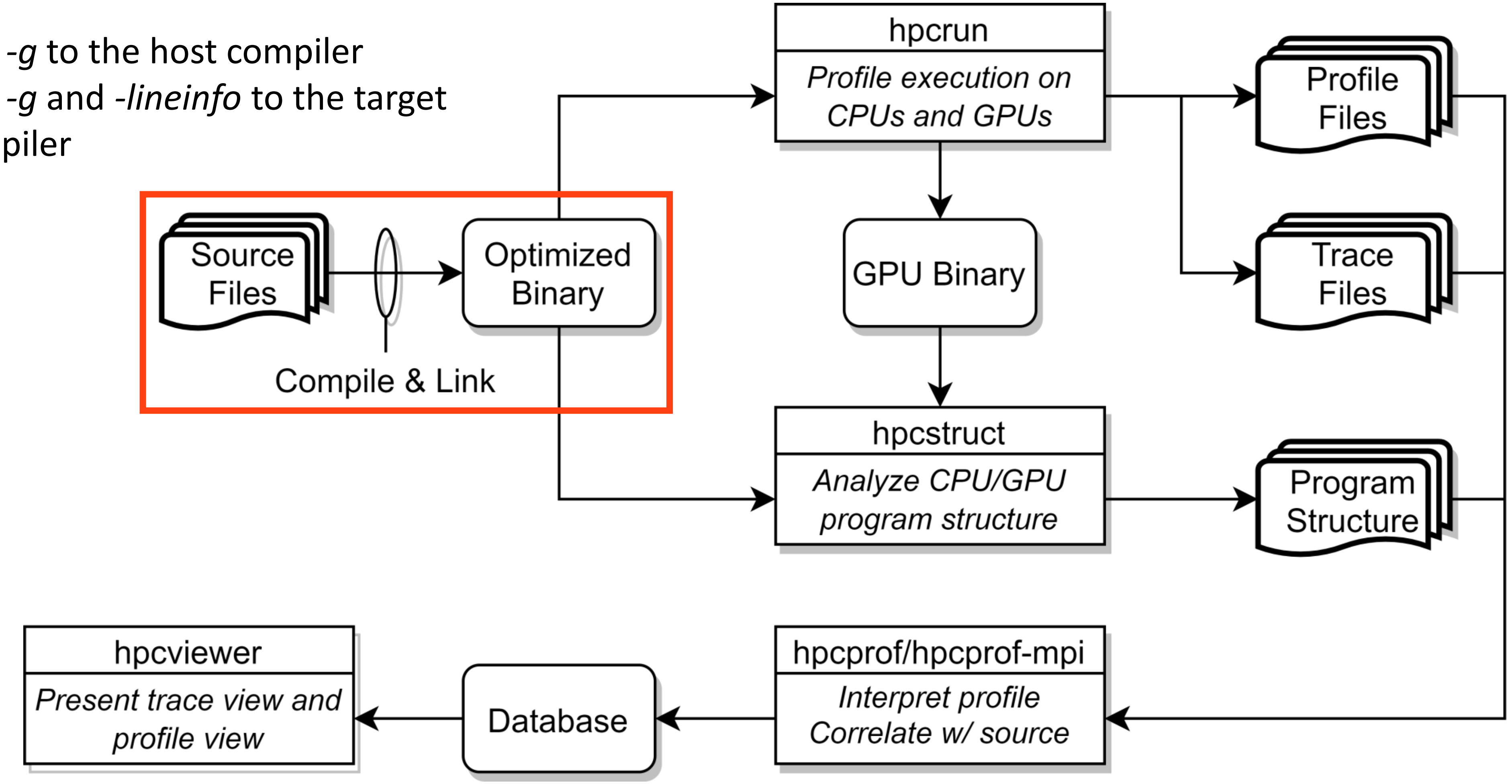
HPCToolkit GPU Workflow



HPCToolkit GPU Workflow

Step 1:

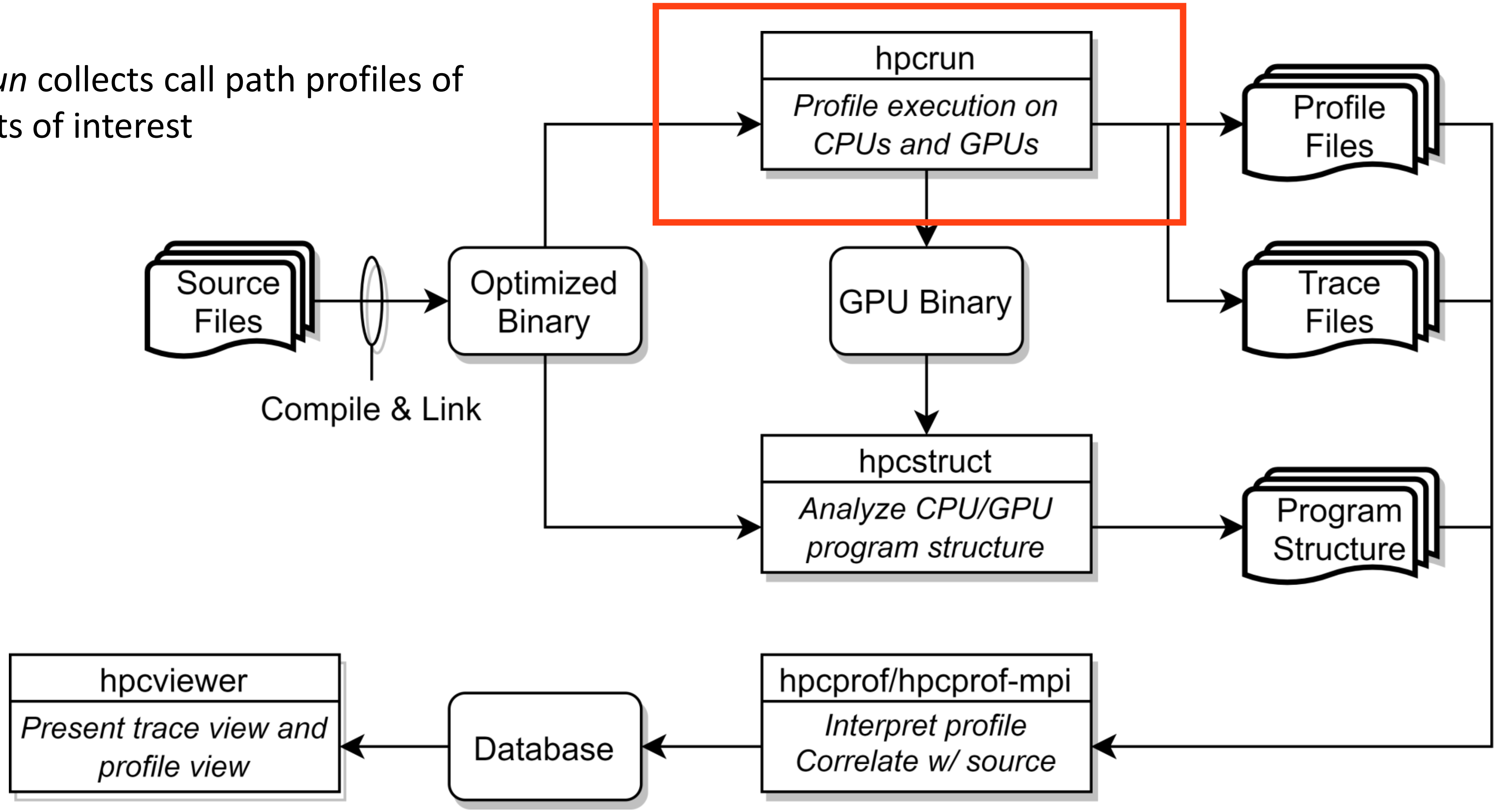
- Add *-g* to the host compiler
- Add *-g* and *-lineinfo* to the target compiler



HPCToolkit GPU Workflow

Step 2:

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



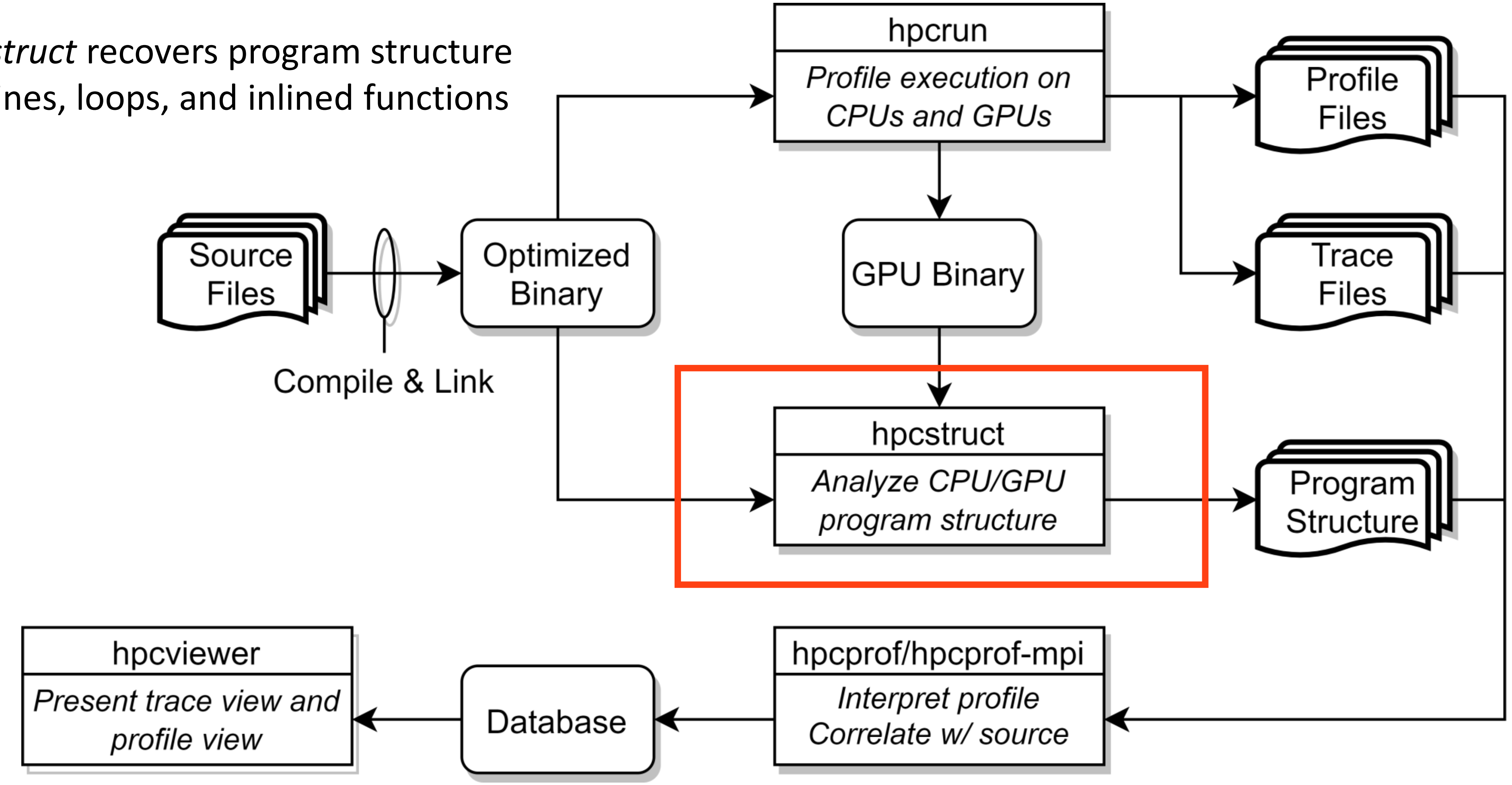
hpcrun

- Measure GPU and CPU execution unobtrusively with *hpcrun*
  - GPU profiling (-e gpu=[nvidia,amd,opencl,level0])
    - hpcrun -e gpu=nvidia <app>
  - GPU tracing (-t)
    - hpcrun -e gpu=nvidia -t <app>
  - GPU PC sampling (NVIDIA GPU only)
    - hpcrun -e gpu=nvidia,pc -t <app>
  - CPU and GPU profiling
    - hpcrun -e REALTIME -e gpu=nvidia -t <app>
  - Use hpcrun with job launchers
    - jsrun -n 1 -g 1 -a 1 hpcrun -e gpu=nvidia <app>
    - srun -n 1 -G 1 hpcrun -e gpu=nvidia <app>
  - Specify output directory
    - hpcrun -o <measurements-dir>
  - List supported events (hundreds of CPU events)
    - hpcrun -L

HPCToolkit GPU Workflow

Step 3:

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



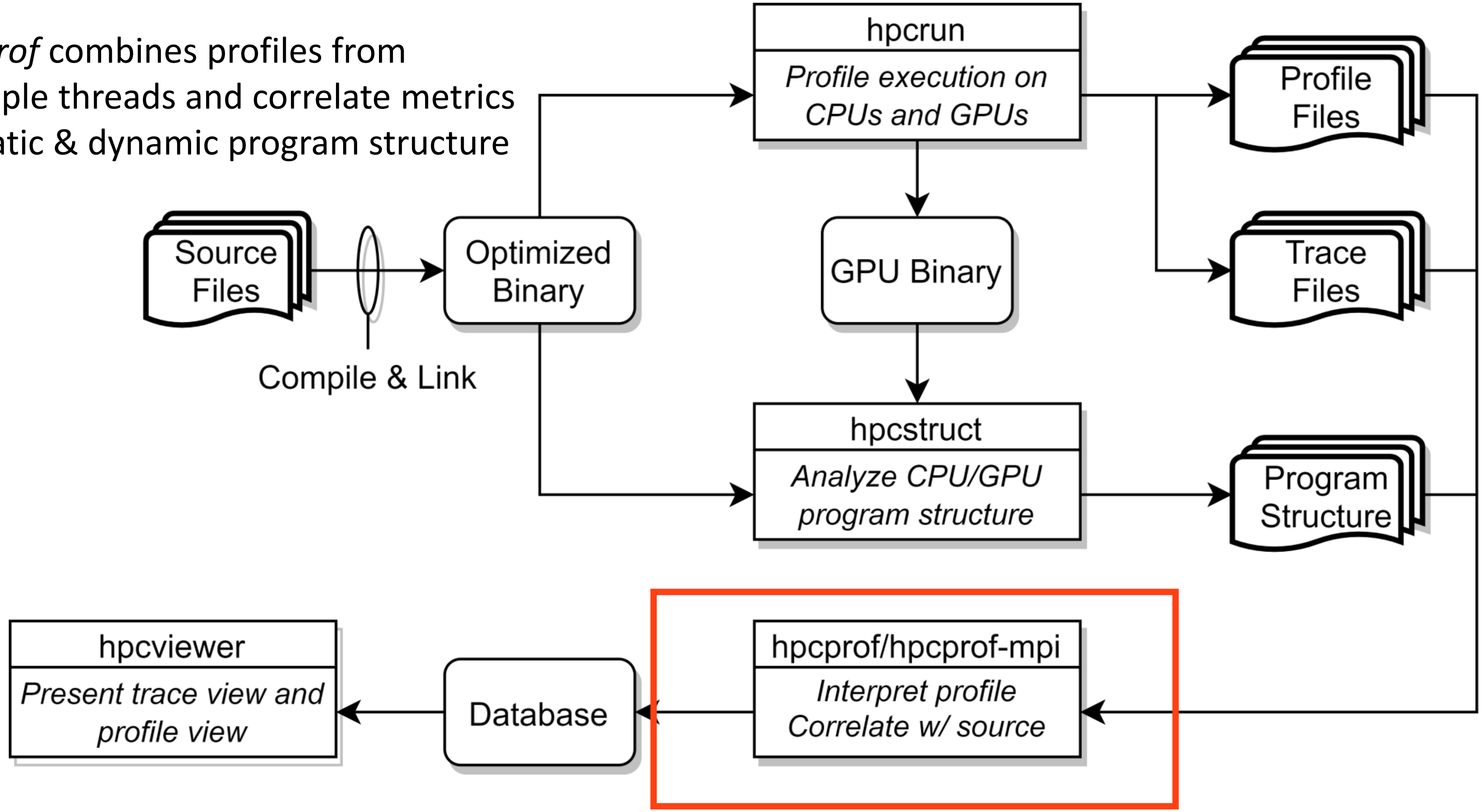
hpcstruct

- Recover program structure with *hpcstruct*
    - Analyze CPU binaries
        - hpcstruct <app>
    - Analyze all GPU binaries in <measurements-dir>
        - hpcstruct <measurements-dir>
        - Parse GPU CFG to recover loop structures and device calling context
            - hpcstruct --gpucfg yes <measurements-dir>
    - Parse binaries in parallel (-j)
        - hpcstruct -j <threads> <binary>, or
        - hpcstruct -j <threads> <measurements-dir>
    - Control parallelism level
        - Adjust the number of threads
        - Adjust the lower bound size to parse GPU binary in parallel
            - hpcstruct --gpu-size <n> -j <threads> <measurements-dir>

HPCToolkit GPU Workflow

Step 4:

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



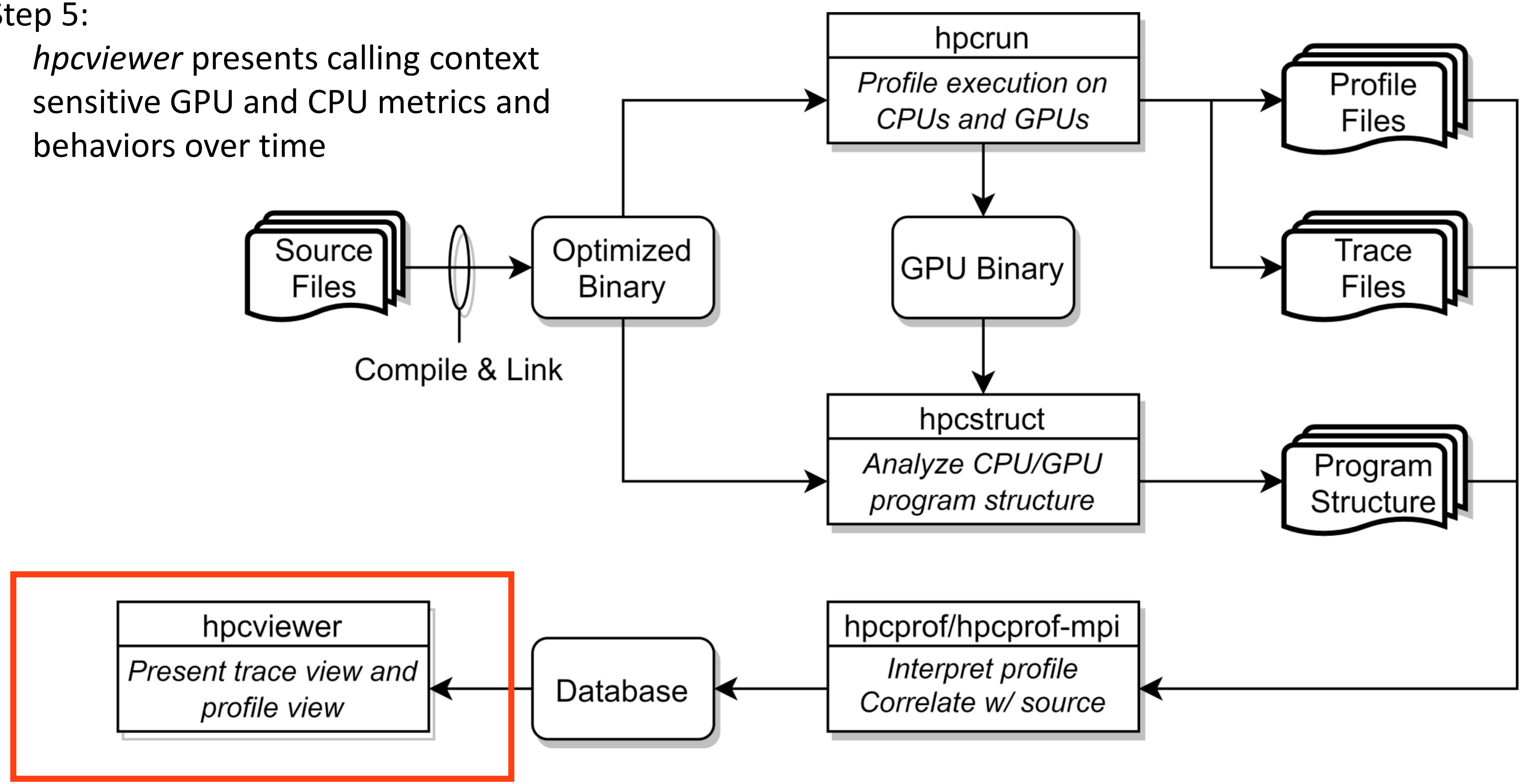
hpcprof/hpcprof-mpi

- Correlate performance data with program structure using hpcprof
  - Use a single process to combine performance data
    - hpcprof -S <app>.hpcstruct <measurements-dir>
  - Specify output directory
    - hpcprof -o <database-dir> -S <app>.hpcstruct <measurements-dir>
  - Use multiple processes to combine performance data
    - jsrun -n <np> hpcprof-mpi -S <app>.hpcstruct <measurements-dir>
    - srun -n <np> hpcprof-mpi -S <app>.hpcstruct <measurements-dir>

HPCToolkit GPU Workflow

Step 5:

- *hpcviewer* presents calling context sensitive GPU and CPU metrics and behaviors over time



Outline

- HPCToolkit GPU Overview
- **Tutorial Examples**
  - Laghos
  - Quicksilver
  - PeleC
- Case Studies
  - SuperLU_DIST
  - STRUMPACK
- Summary

GPU Performance Metrics

- Execution time
  - GPUOPS (sec)
    - The total amount of GPU times spent on kernels and memory operations.
- GPU kernels metrics (GKER)
  - GKER (sec)
  - GKER:BLKS
  - GKER:COUNT
- GPU memory metrics (GXCOPY and GMEM)
  - GXCOPY (sec)
  - GXCOPY:H2D (B)
  - GXCOPY:D2H (B)
- GPU instructions (GINS)
  - GINS
    - Total number of instruction samples
  - GINS:STL_ANY
    - Total number of stalled instruction samples
  - GINS:STL_GMEM
    - Total number of stalled instruction samples (waiting for the results from global memory)

Laghos

- Step-by-step profiling
  - hpcstruct -j <n> for hundreds GPU binaries
  - hpcviewer
    - Bottom-up view
      - Kernel and copy hotspots
    - Top-down view
      - Full context calling
      - Important kernel metrics
- Compare with Nsight Systems
  - HPCToolkit performs profiling and tracing, while Nsight Systems only does tracing
- hpctoolkit-tutorial-examples/examples/gpu/laghos
  - source setup-env/<platform>.sh
  - make build
  - make run-short

Quicksilver

- Step-by-step profiling
  - hpcrun -e gpu=nvidia,pc to collect pc sampling data
  - hpcstruct --gpucfg yes to reconstruct calling context for GPU device functions and loop nests
  - hpcviewer
    - Instruction stalls with their full context calling context
- Compare with Nsight Compute
  - HPCToolkit does not replay GPU kernels
  - HPCToolkit recovers loops and reconstructs approximate calling context trees on GPUs
- hpctoolkit-tutorial-examples/gpu/quicksilver
  - source setup-env/<platform>.sh
  - make build
  - make run-pc

PeleC

- Step-by-step profiling
  - hpcrun -e REALTIME -e gpu=nvidia -t to collect CPU and GPU traces
  - hpcviewer
    - Use filter to hide background CPU threads
    - Zoom in to focus on GPU activities
    - Use procedure-color map to highlight <gpu sync> activities
      - Unnecessary consecutive GPU synchronizations
- hpctoolkit-tutorial-examples/gpu/pelec
  - source setup-env/<platform>.sh
  - make build
  - make run

Outline

- HPCToolkit GPU Overview
- Tutorial Examples
  - Laghos
  - Quicksilver
  - PeleC
- **Case Studies**
  - SuperLU_DIST
  - STRUMPACK
- Summary

SuperLU_DIST

- A GPU-accelerated sparse direct solver
- Test case
  - Pddrive3d
- Environment
  - Summit compute node
  - Single MPI process
  - Single GPU

SuperLU_DIST Observations - 1

- GPU activities are sparse comparing to CPU activities
  - CPU samples are usually taken at a low frequency

SuperLU_DIST Observations - 2

- Expensive CPU computations delay work being offloaded to GPUs
  - Optimizing the CPU code improves this code region by 1.78x.

	REALTIME (sec):Sum (I)		GPUOP (sec):Sum (I)	
529: SchurCompUpdate_GPU	6.95e-01	0.9%	1.54e-01	70.1%
701: cublasDgemm_v2 [libcublas.so.11.1.0.229]	1.00e-01	0.1%	4.80e-02	21.8%
superlu_gpu.cu: 533	6.50e-02	0.1%		
superlu_gpu.cu: 521	6.00e-02	0.1%		
588: cudaMemcpyAsync [libsuperlu_dist.so.7.0.0]	5.50e-02	0.1%	5.02e-03	2.3%
superlu_gpu.cu: 542	5.50e-02	0.1%		
723: Scatter_GPU_kernel(int, int, int, int, int, int, int, int, int, int, ir	5.00e-02	0.1%	5.23e-02	23.7%
563: cudaMemcpyAsync [libsuperlu_dist.so.7.0.0]	4.00e-02	0.1%	4.39e-03	2.0%
561: cudaEventRecord [libsuperlu_dist.so.7.0.0]	4.00e-02	0.1%		
580: cudaMemcpyAsync [libsuperlu_dist.so.7.0.0]	3.50e-02	0.0%	4.27e-03	1.9%
572: cudaMemcpyAsync [libsuperlu_dist.so.7.0.0]	3.50e-02	0.0%	1.58e-02	7.2%
568: cudaMemcpyAsync [libsuperlu_dist.so.7.0.0]	3.00e-02	0.0%	1.56e-02	7.1%
superlu_gpu.cu: 540	3.00e-02	0.0%		
576: cudaMemcpyAsync [libsuperlu_dist.so.7.0.0]	2.00e-02	0.0%	4.41e-03	2.0%

SuperLU_DIST Observations - 2

- Expensive CPU computations delay work being offloaded to GPUs
  - Optimizing the CPU code improves this code region by 1.78x.

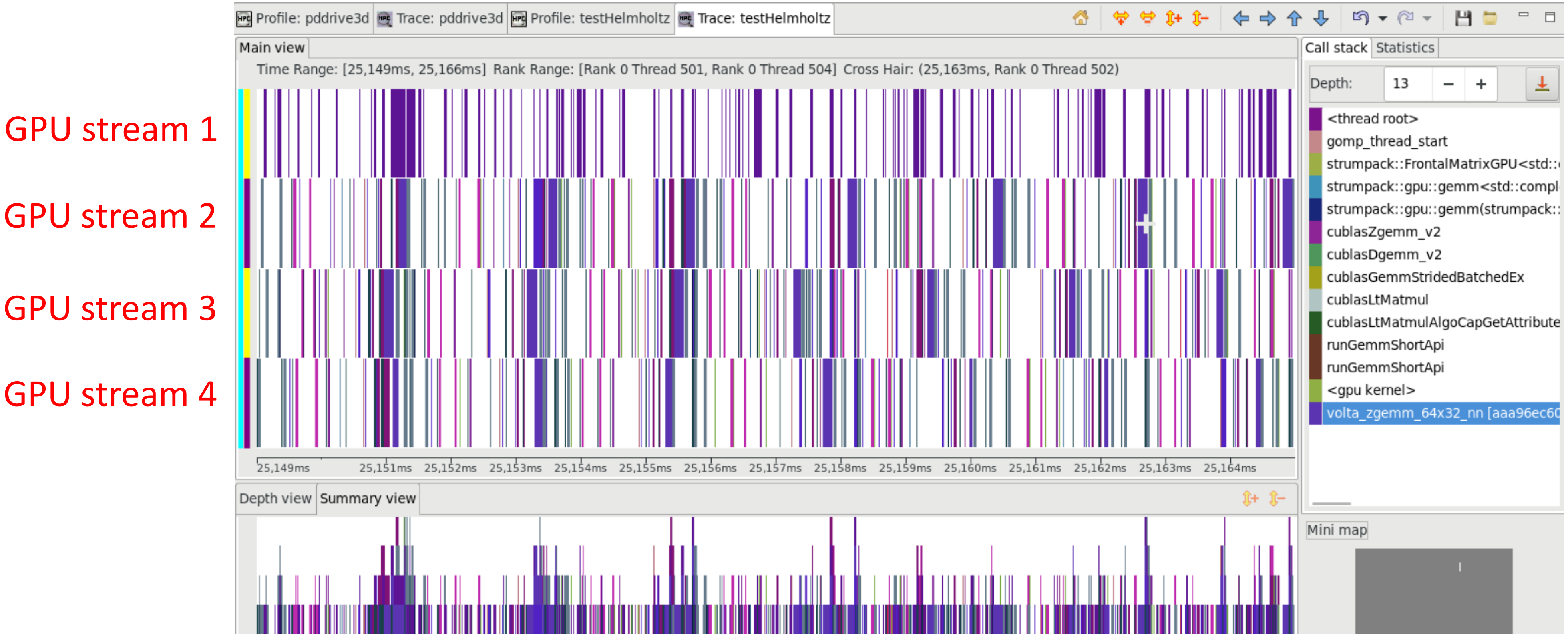
	REALTIME (sec):Sum (I)		GPUOP (sec):Sum (I)	
529: SchurCompUpdate_GPU	6.95e-01	0.9%	1.54e-01	70.1%
701: cublasDgemm_v2 [libcublas.so.11.1.0.229]	1.00e-01	0.1%	4.80e-02	21.8%
superlu_gpu.cu: 533	6.50e-02	0.1%		
superlu_gpu.cu: 521	6.00e-02	0.1%		
588: cudaMemcpyAsync [libsuperlu_dist.so.7.0.0]	5.50e-02	0.1%	5.02e-03	2.3%
superlu_gpu.cu: 542	5.50e-02	0.1%		
723: Scatter_GPU_kernel(int, int, int, int, int, int, int, int, int, int, ir	5.00e-02	0.1%	5.23e-02	23.7%
563: cudaMemcpyAsync [libsuperlu_dist.so.7.0.0]	4.00e-02	0.1%	4.39e-03	2.0%
561: cudaEventRecord [libsuperlu_dist.so.7.0.0]	4.00e-02	0.1%		
580: cudaMemcpyAsync [libsuperlu_dist.so.7.0.0]	3.50e-02	0.0%	4.27e-03	1.9%
572: cudaMemcpyAsync [libsuperlu_dist.so.7.0.0]	3.50e-02	0.0%	1.58e-02	7.2%
568: cudaMemcpyAsync [libsuperlu_dist.so.7.0.0]	3.00e-02	0.0%	1.56e-02	7.1%
superlu_gpu.cu: 540	3.00e-02	0.0%		
576: cudaMemcpyAsync [libsuperlu_dist.so.7.0.0]	2.00e-02	0.0%	4.41e-03	2.0%

STRUMPACK

- Solvers for sparse and dense rank-structured linear systems
- Test case
  - testHelmholtz
- Environment
  - Summit compute node
  - Four MPI processes
  - Four GPUs

STRUMPACK Observations - 1

- cuBLAS kernels are launched to multiple streams to keep GPUs busy



STRUMPACK Observations - 1

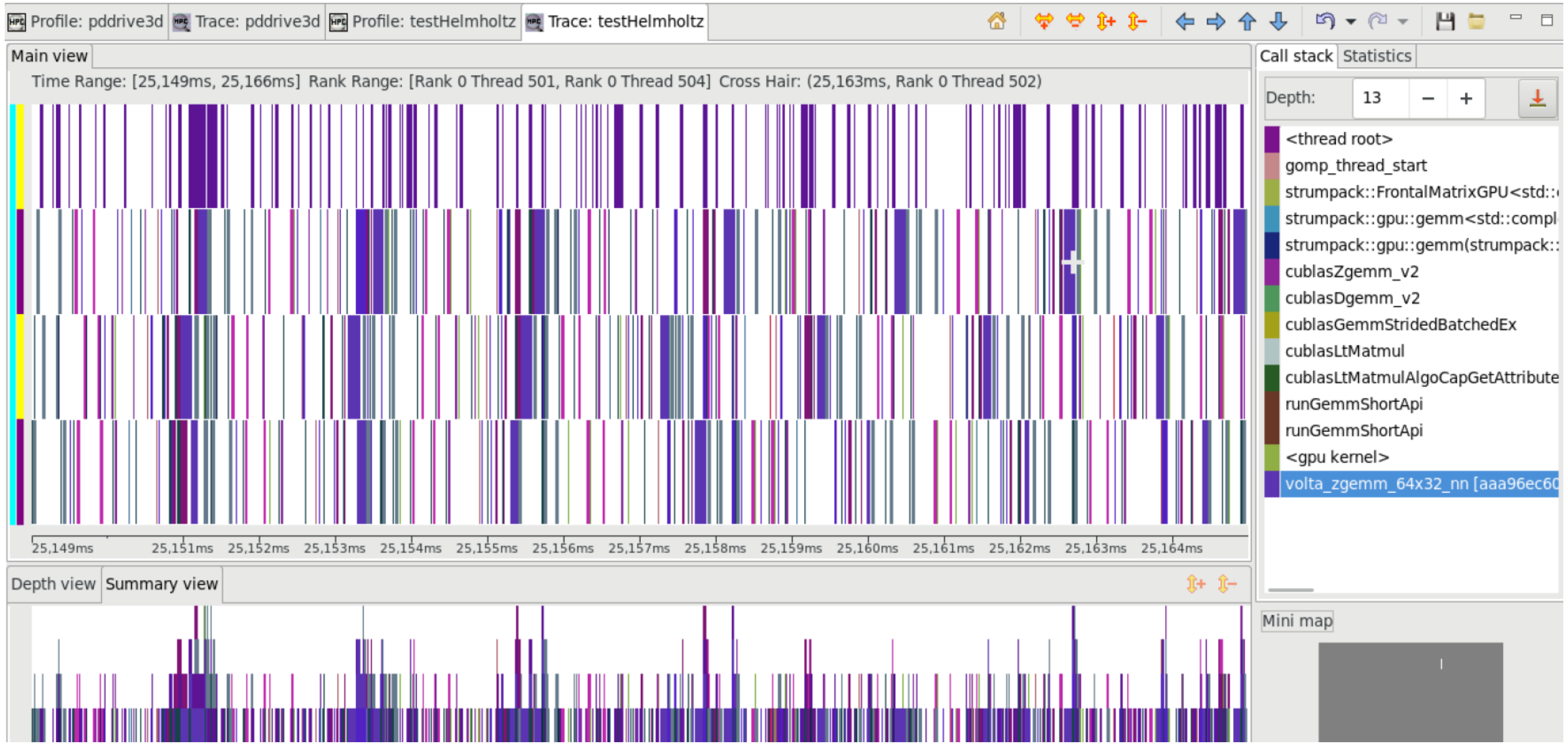
- cuBLAS kernels are launched to multiple streams to keep GPUs busy

GPU stream 1

GPU stream 2

GPU stream 3

GPU stream 4



STRUMPACK Observations - 1

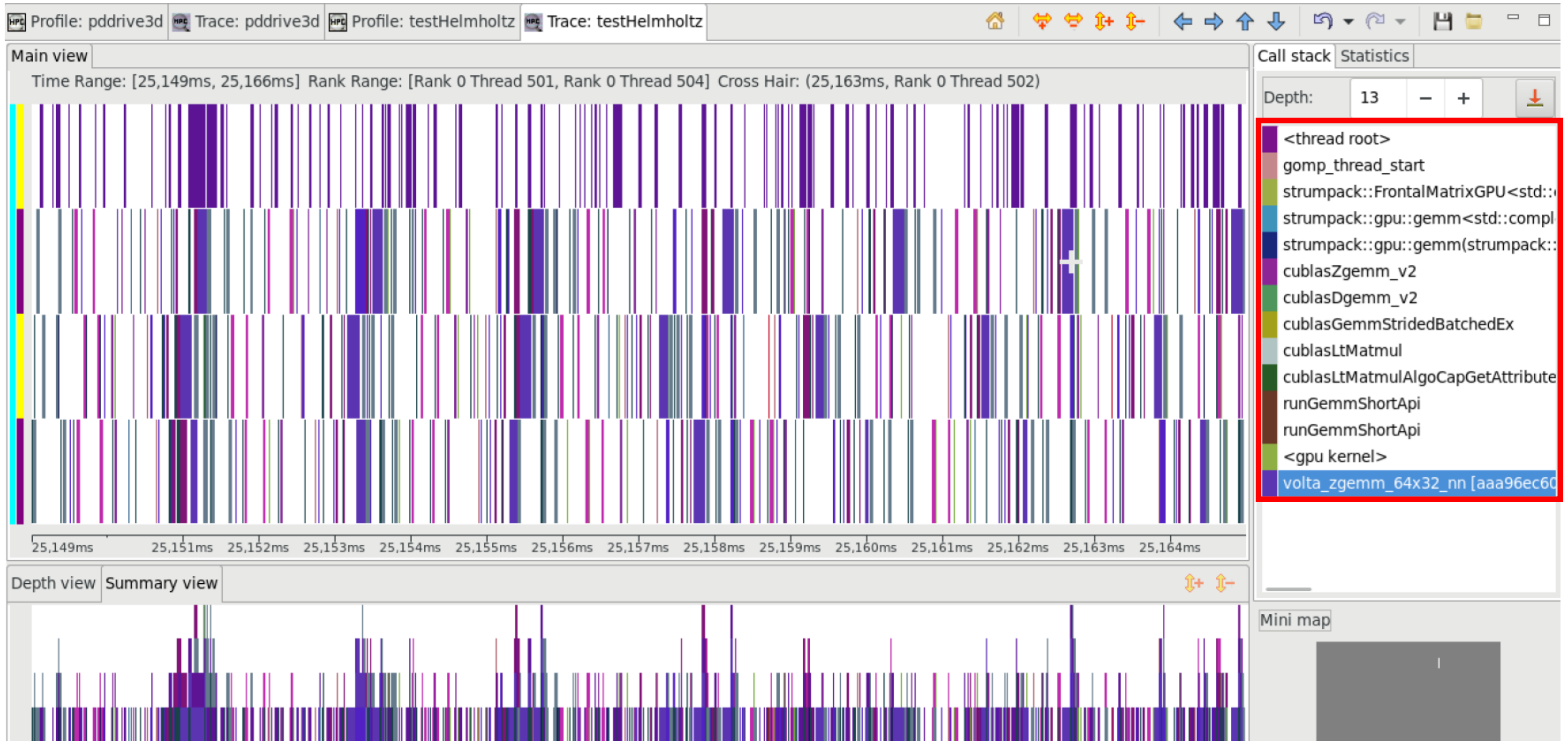
- cuBLAS kernels are launched to multiple streams to keep GPUs busy

GPU stream 1

GPU stream 2

GPU stream 3

GPU stream 4



cuBLAS function call stack

STRUMPACK Observations - 1

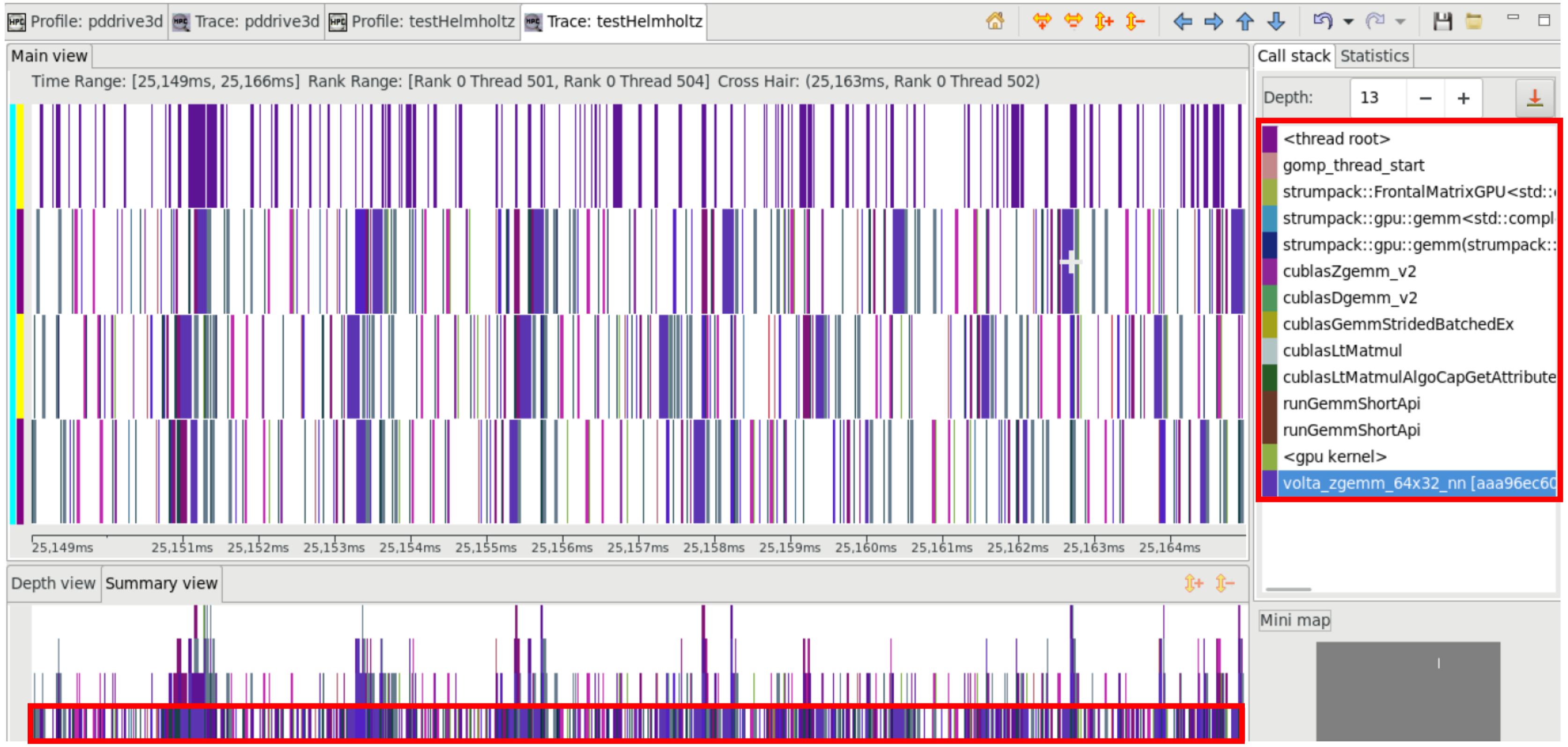
- cuBLAS kernels are launched to multiple streams to keep GPUs busy

GPU stream 1

GPU stream 2

GPU stream 3

GPU stream 4

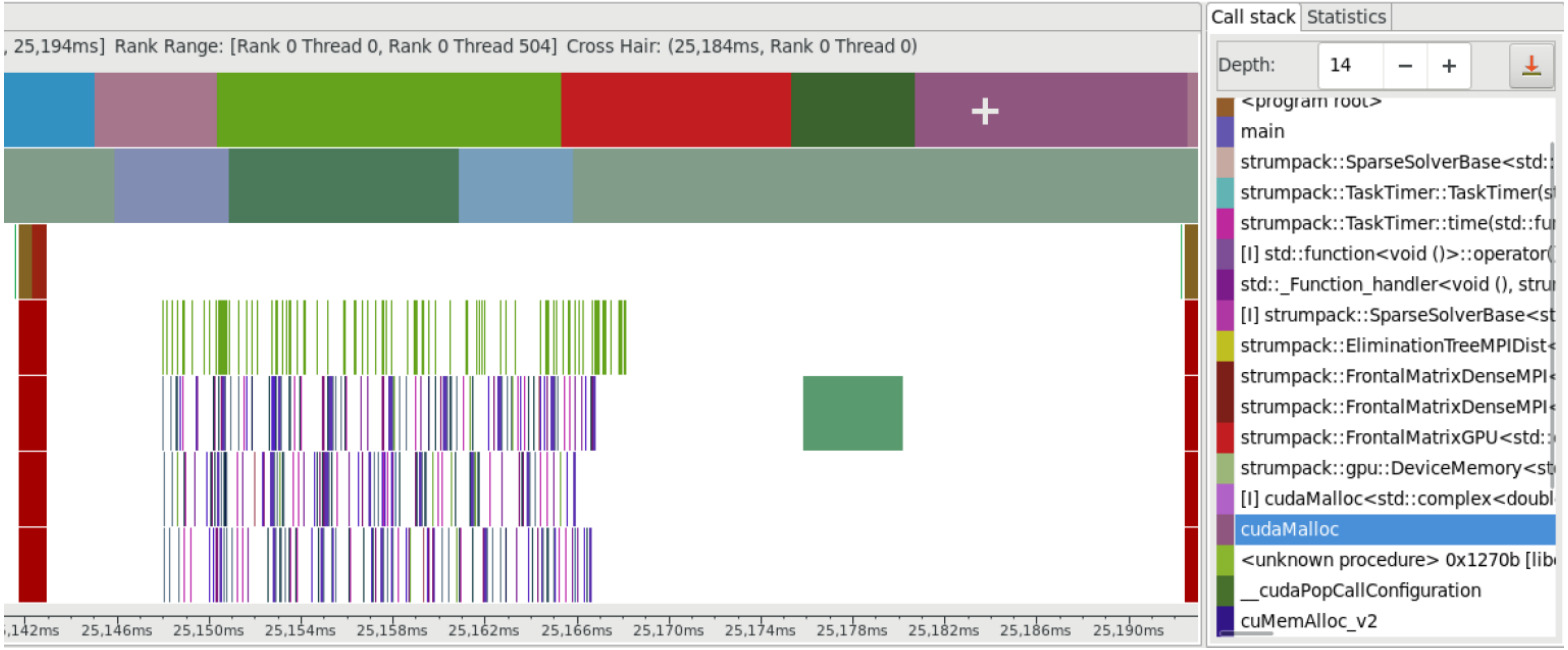


cuBLAS function call stack

Only a small fraction of the space is white

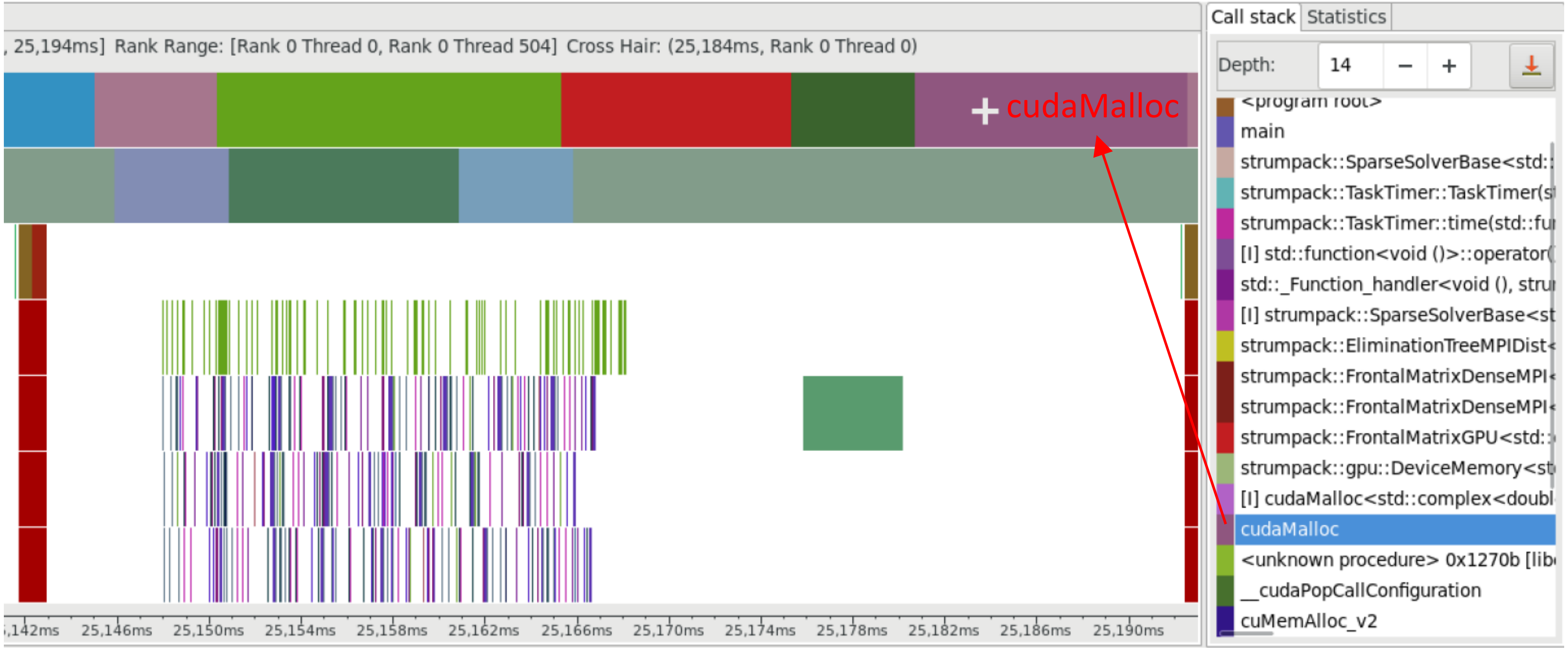
STRUMPACK Observations - 2

- cudaMalloc and cudaFree are the main bottlenecks
  - The STRUMPACK team switched their memory allocation to avoid excessive memory allocations and frees, achieving 1.15x speedup.



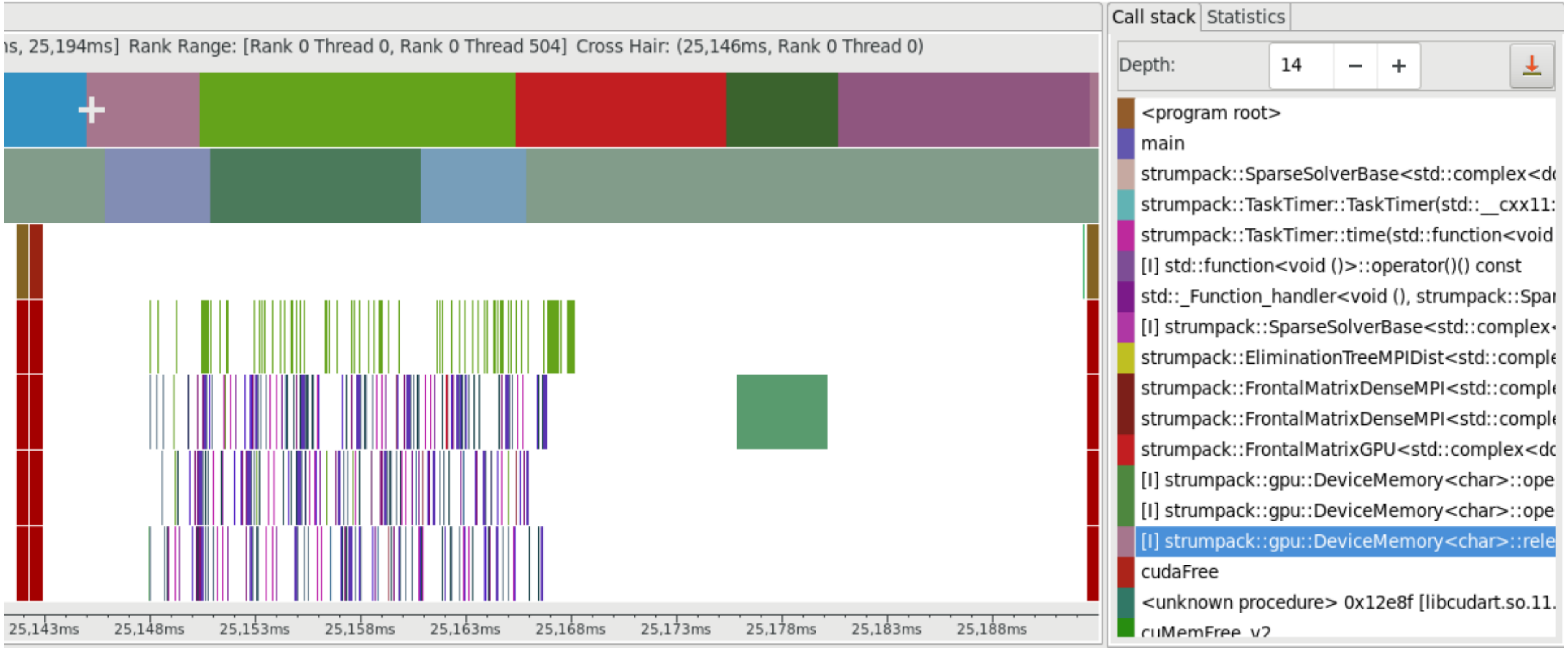
STRUMPACK Observations - 2

- cudaMalloc and cudaFree are the main bottlenecks
  - The STRUMPACK team switched their memory allocation to avoid excessive memory allocations and frees, achieving 1.15x speedup.



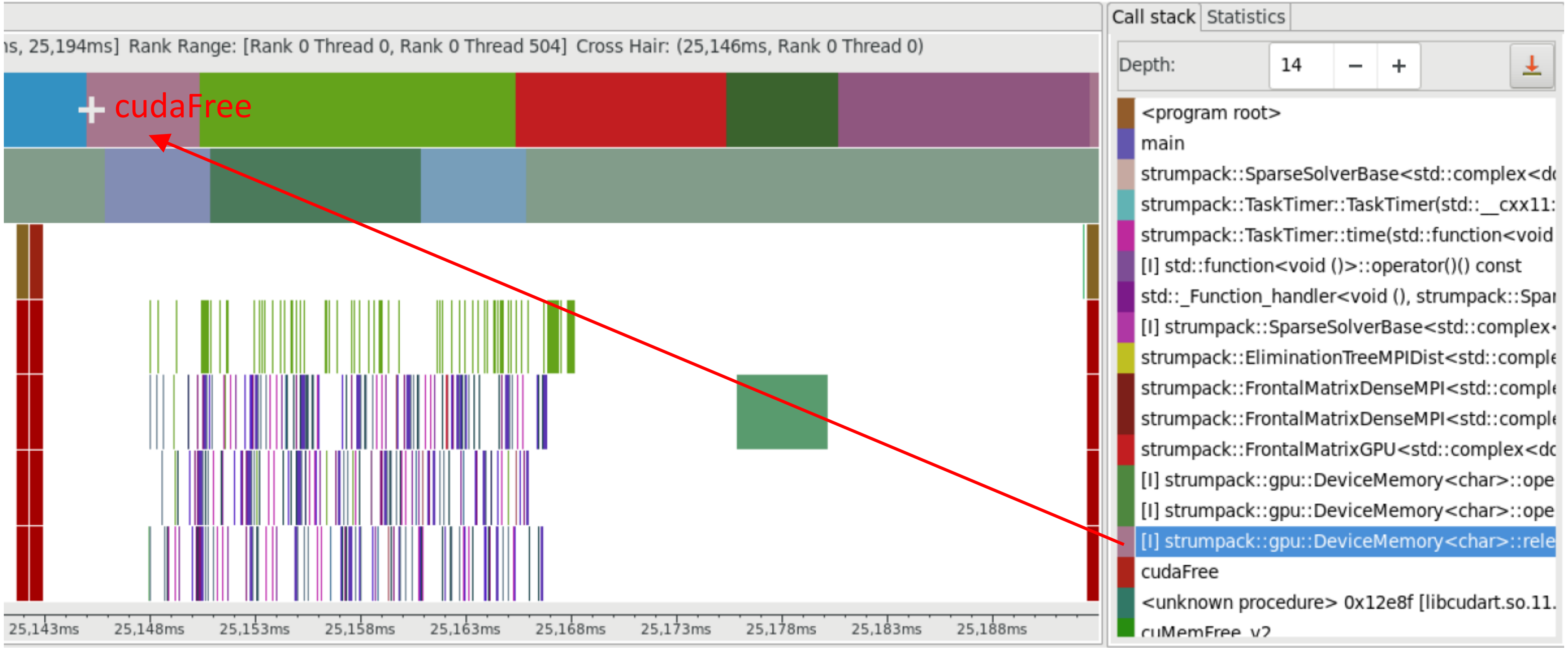
STRUMPACK Observations - 2

- cudaMalloc and cudaFree are the main bottlenecks
  - The STRUMPACK team switched their memory allocation to avoid excessive memory allocations and frees, achieving 1.15x speedup.



STRUMPACK Observations - 2

- cudaMalloc and cudaFree are the main bottlenecks
  - The STRUMPACK team switched their memory allocation to avoid excessive memory allocations and frees, achieving 1.15x speedup.



Outline

- HPCToolkit GPU Overview
- Tutorial Examples
  - Laghos
  - Quicksilver
  - PeleC
- Case Studies
  - SuperLU_DIST
  - STRUMPACK
- **Summary**

Summary

- Rice University's HPCToolkit is a measurement and analysis tool that
  - measures GPU activities and GPU instruction samples and attribute them to their corresponding calling context;
  - provides a trace view of how an execution evolves over time and a profile view that associates metrics with a hierarchy of individual lines, loops, and functions;
  - collects, analyzes, and visualizes profiles within and across nodes
- HPCToolkit's workflow
  - hpcrun
  - hpcstruct
  - hpcprof/hpcprof-mpi
  - hpcviewer

HPCToolkit Caveats - 1

- hpcrun’s measurement time might be dilated if an application has many short-lived kernels due to the cost of call path unwinding and kernel instrumentation (concurrent kernel mode)
  - you need to consider this slowdown when assessing how active the GPU is using profiles or traces
    - this issue affects both HPCToolkit and Nsight Systems
  - HPCToolkit measures GPU kernels with a CUPTI activity that serializes kernels; this will change
- hpcstruct’s control flow analysis for large GPU binaries might take long time due to the overhead by nvdisasm
  - sometimes, nvdisasm can’t analyze GPU binaries, so hpcstruct can’t always recover GPU loops and calling contexts
  - reserve longer time (e.g., two hours) on a compute node if you want CFGs of large GPU binaries
- hpcprof’s approximately attributes costs to GPU calling contexts
- HPCToolkit does not record and present meta data; this will change
  - We don’t show what cores your threads are running on
  - We don’t show how many GPUs are using
  - GPU streams have a thread id starting from 500

HPCToolkit Caveats - 2

- GPU kernel metrics are attributed to
  - kernel itself (useful)
  - the source line for the first machine instruction in the kernel (ignore)
  - erroneously attributed to “aggregate exclusive costs” (ignore)
- Currently, we need to use -t option when collecting PC samples
  - ignore traces collected with PC samples
  - we shouldn’t have to turn on tracing but it is currently needed to compensate for a bug in hpcprof that causes it to omit inclusive metrics without -t
- Currently, PC sampling may significantly slow your execution
  - We have asked NVIDIA to improve CUPTI to lower overhead
  - You might want to collect PC samples for a shorter run
- Currently, metrics are collected in a dense format
  - Not a problem for CPU only profiling with several metrics
  - This leads to a huge space explosion for GPU profiling which might cause you a problem; this is changing
- The installed HPCToolkit version does not have access to GPU hardware counters
  - Needed for roofline analysis; working with the PAPI team to resolve this issue

HPCToolkit Tutorial Example Tips

- Available on Github
  - [HPCToolkit/hpctoolkit-tutorial-examples: CPU and GPU tutorial examples (github.com)]
- Usage
  - Clone the repository and choose an example (e.g., quicksilver)
    - git clone https://github.com/HPCToolkit/hpctoolkit-tutorial-examples.git
    - cd hpctoolkit-tutorial-examples/examples/gpu/quicksilver/
  - Once on the login node
    - export HPCTOOLKIT_TUTORIAL_PROJECTID=<project-id>
    - export HPCTOOLKIT_TUTORIAL_RESERVATION=<reservation-id>
      - SUMMIT: hpctoolkit1 (day1), hpctoolkit2 (day2)
      - Cori-GPU: hpc1_gpu (day1), hpc2_gpu (day2)
      - Cori-CPU: hpc1_knl (day1), hpc2_knl (day2)
  - For each example, on the login node
    - source setup-env/<platform>.sh
    - make build
    - make run-pc
    - hpcviewer *hpctoolkit-quicksilver-gpu-cuda-pc.d*