Using HPCToolkit to Measure and Analyze the Performance of GPU-Accelerated Applications

ECP Project WBS 2.3.2.08

EXASCALE COMPUTING PROJECT

John Mellor-Crummey and Keren Zhou
Rice University

ECP Annual Meeting
February 5, 2020

Download GPU application examples to run and measure:
`git clone https://github.com/HPCToolkit/hpctoolkit-tutorial-examples`

National Nuclear Security Administration
U.S. DEPARTMENT OF ENERGY | Office of Science
RICE
WISCONSIN UNIVERSITY OF WISCONSIN–MADISON

Acknowledgments

- **Current funding**
  - DOE Exascale Computing Project (Subcontract 4000151982)
  - NSF Software Infrastructure for Sustained Innovation (Collaborative Agreement 1450273)
  - DOE Labs: ANL (Subcontract 9F-60073), Tri-labs (LLNL Subcontract B633244)
  - Industry: AMD
- **Team**
  - Lead Institution: Rice University
    - PI: Prof. John Mellor-Crummey
    - Research staff: Laksono Adhianto, Mark Krentel, Xiaozhu Meng, Scott Warren
    - Contractor: Marty Itzkowitz
    - Students: Keren Zhou, Jonathon Anderson, Vladimir Indjic
    - Summer interns: Tijana Jovanovic, Aleksa Simovic
  - Subcontractor: University of Wisconsin – Madison
    - Lead: Prof. Barton Miller

Performance Analysis Challenges for GPU-accelerated Supercomputers

- **Myriad performance concerns**
  - Computation performance
    - Principal concern: keep GPUs busy and computing productively
      - need extreme-scale data parallelism!
  - Data movement costs within and between memory spaces
  - Internode communication
  - I/O
- **Many ways to hurt performance**
  - insufficient parallelism, load imbalance, serialization, replicated work, parallel overheads …
- **Hardware and execution model complexity**
  - Multiple compute engines with vastly different characteristics, capabilities, and concerns
  - Multiple memory spaces with different performance characteristics
    - CPU and GPU have different complex memory hierarchies
  - Asynchronous execution

Measurement Challenges for GPU-accelerated Supercomputers

- **Extreme-scale parallelism**
  - Serialization within tools will disrupt parallel performance
- **Multiple measurement modalities and interfaces**
  - Sampling on the CPU
  - Callbacks when GPU operations are launched
  - GPU event stream
- **Frequent GPU kernel launches require a low-overhead measurement substrate**
- **Importance of third-party measurement interfaces**
  - Tools can only measure what GPU hardware can monitor
    - support for fine-grain measurement will be essential to diagnose GPU inefficiencies
  - Linux perf_events for kernel measurement
  - GPU monitoring libraries from vendors

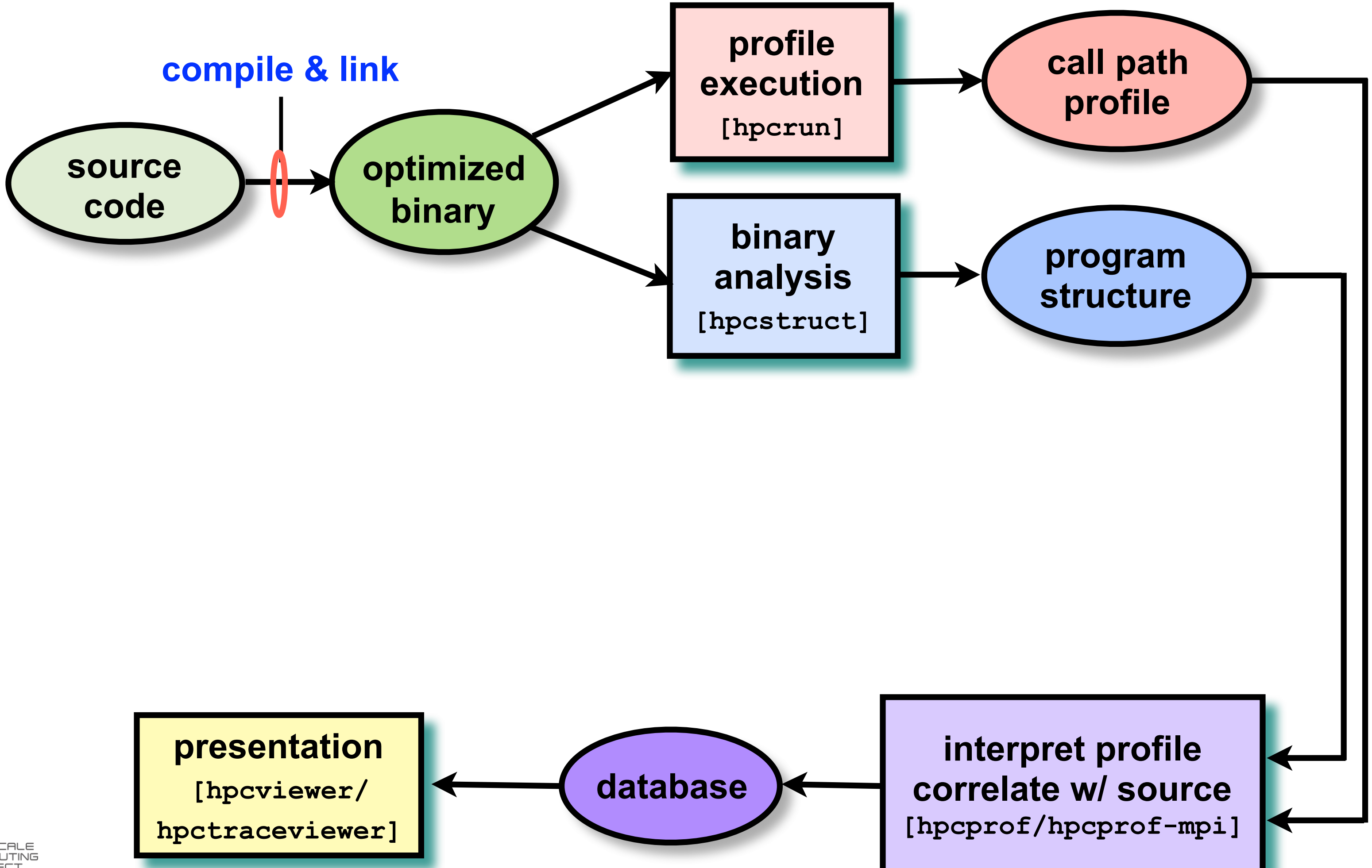
Outline

- **Performance measurement and analysis challenges for GPU-accelerated supercomputers**
- **Introduction to HPCToolkit performance tools**
  - Overview of HPCToolkit components and their workflow
  - HPCToolkit's graphical user interfaces and using them effectively
- **Analyzing the performance of GPU-accelerated supercomputers with HPCToolkit**
  - Overview of HPCToolkit's GPU performance measurement capabilities
  - Collecting measurements
  - Analysis and attribution
  - Exploring measurements and analysis results
- **Experiences with analysis and tuning of GPU-accelerated codes**
  - Computation, memory hierarchy, and data movement issues
- **Obtaining HPCToolkit**

Rice University’s HPCToolkit Performance Tools

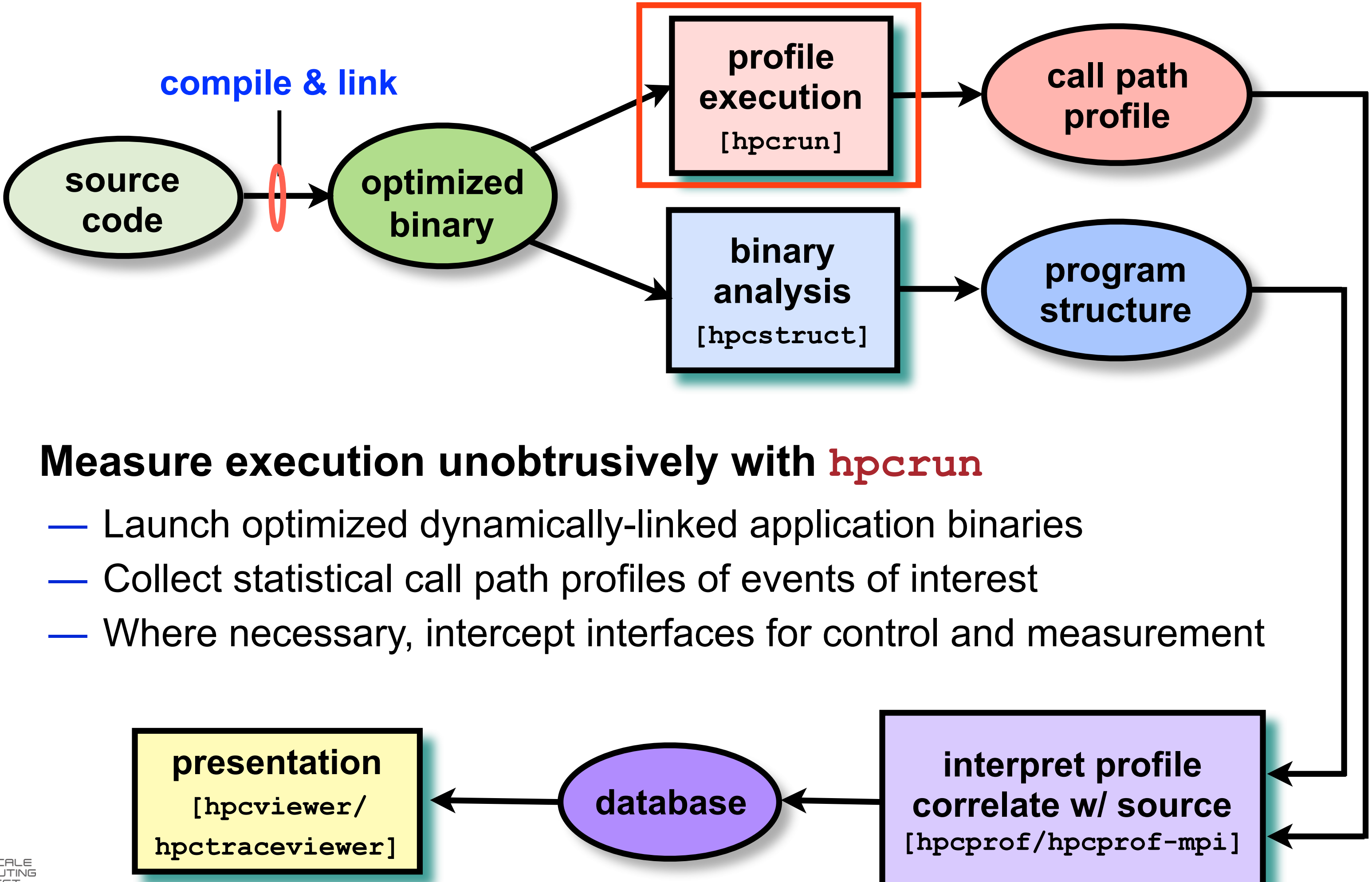
- **Employs binary-level measurement and analysis**
  - Observes executions of fully optimized, dynamically-linked applications
  - Supports multi-lingual codes with external binary-only libraries
- **Collects sampling-based measurements of CPU**
  - Controllable overhead
  - Minimize systematic error and avoid blind spots
  - Enable data collection for large-scale parallelism
- **GPU performance using measurement APIs provided by vendors**
  - Callbacks to monitor launch of GPU operations
  - Activity API to monitor and present information about asynchronous operations on GPU devices
  - PC sampling for fine-grain measurement
- **Associates metrics with both static and dynamic context**
  - Loop nests, procedures, inlined code, calling context on both CPU and GPU
- **Enables one to specify and compute derived CPU and GPU performance metrics of your choosing**
  - Diagnosis often requires more than one species of metric
- **Supports top-down performance analysis**
  - Identify costs of interest and drill down to causes: up and down call chains, over time

HPCToolkit Workflow



EXASCALE COMPUTING PROJECT

HPCToolkit Workflow



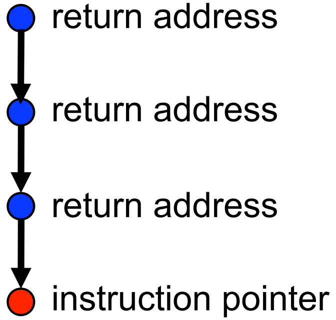
Measure execution unobtrusively with `hpcrun`

- Launch optimized dynamically-linked application binaries
- Collect statistical call path profiles of events of interest
- Where necessary, intercept interfaces for control and measurement

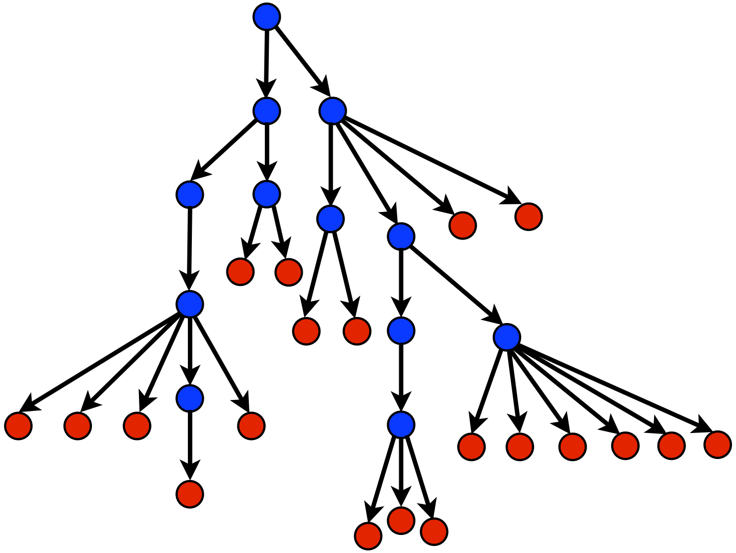
Call Path Profiling

- **Measure and attribute costs in context**
  - Sample timer or hardware counter overflows
  - Gather CPU calling context using stack unwinding

Call path sample

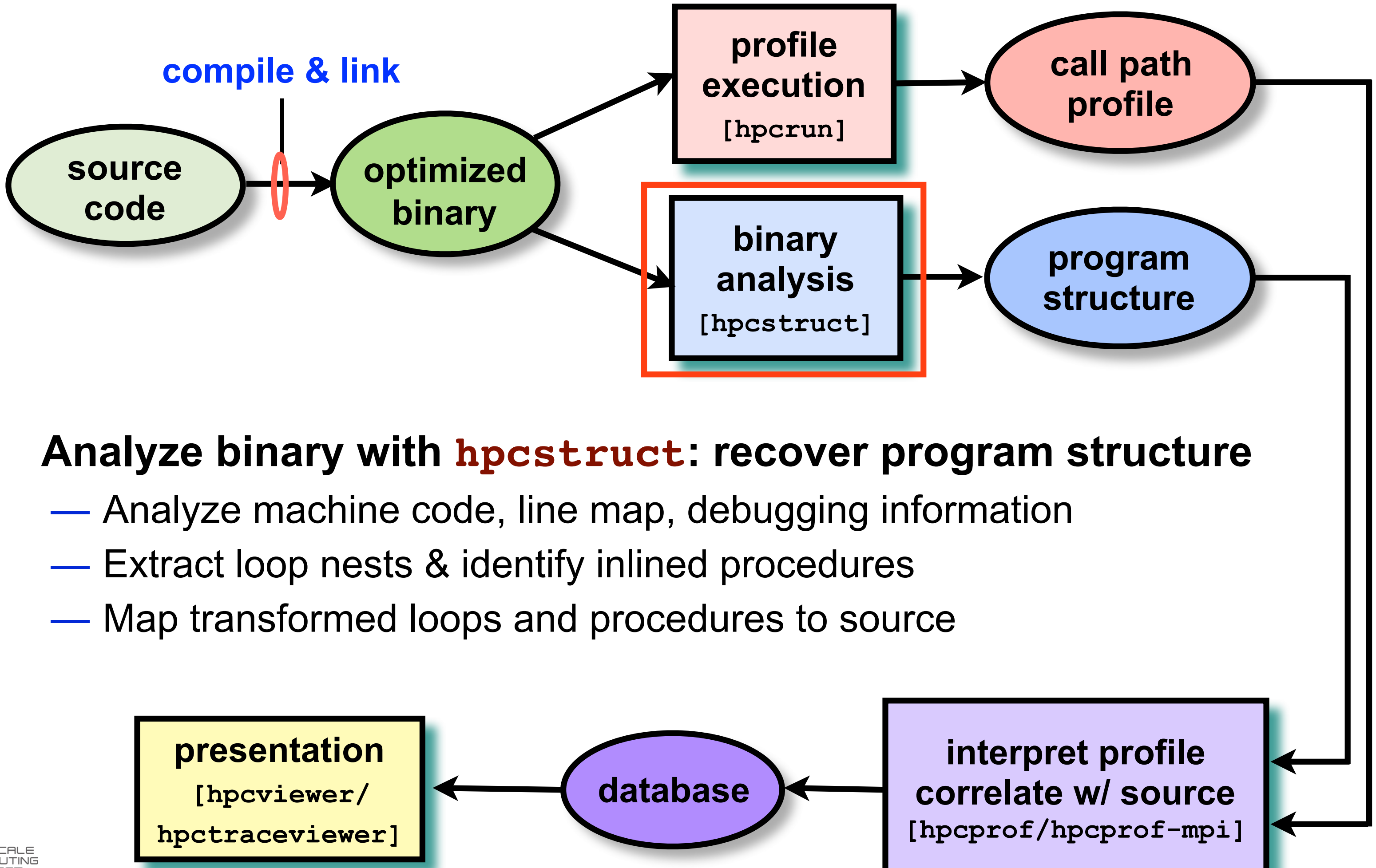


Calling context tree



Overhead proportional to sampling frequency, not call frequency

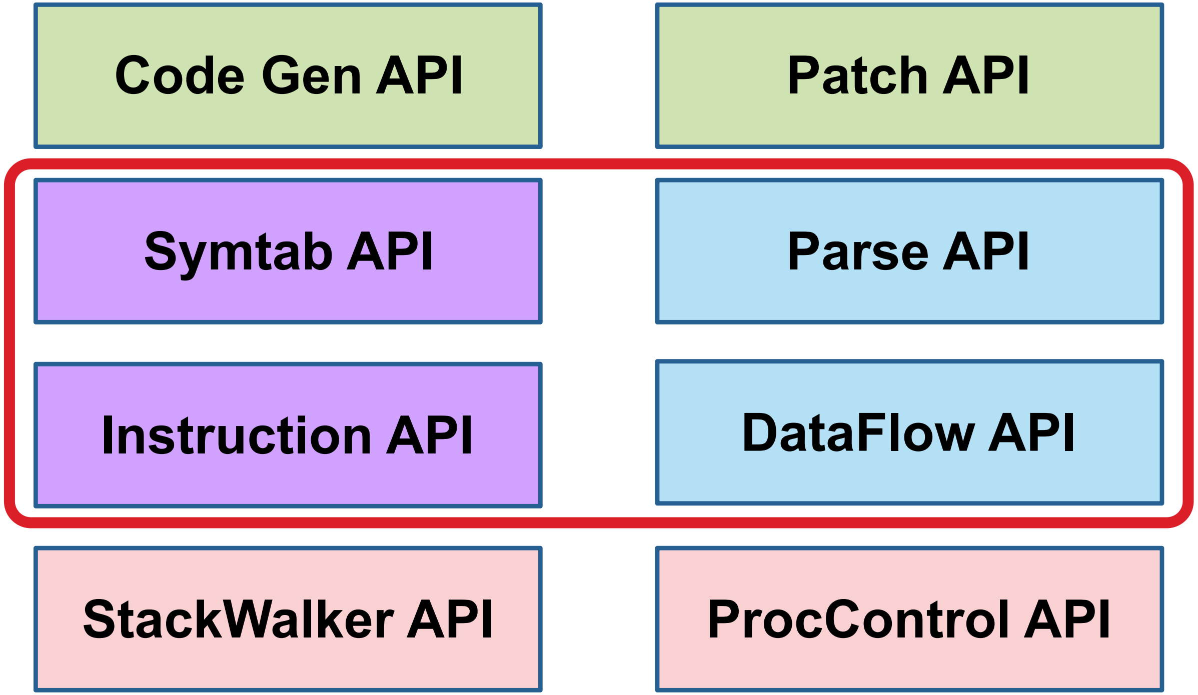
HPCToolkit Workflow



Analyze binary with `hpcstruct`: recover program structure

- Analyze machine code, line map, debugging information
- Extract loop nests & identify inlined procedures
- Map transformed loops and procedures to source

Dyninst: A Toolkit for Binary Analysis and Instrumentation



Architectures

X86_64

Power/BE

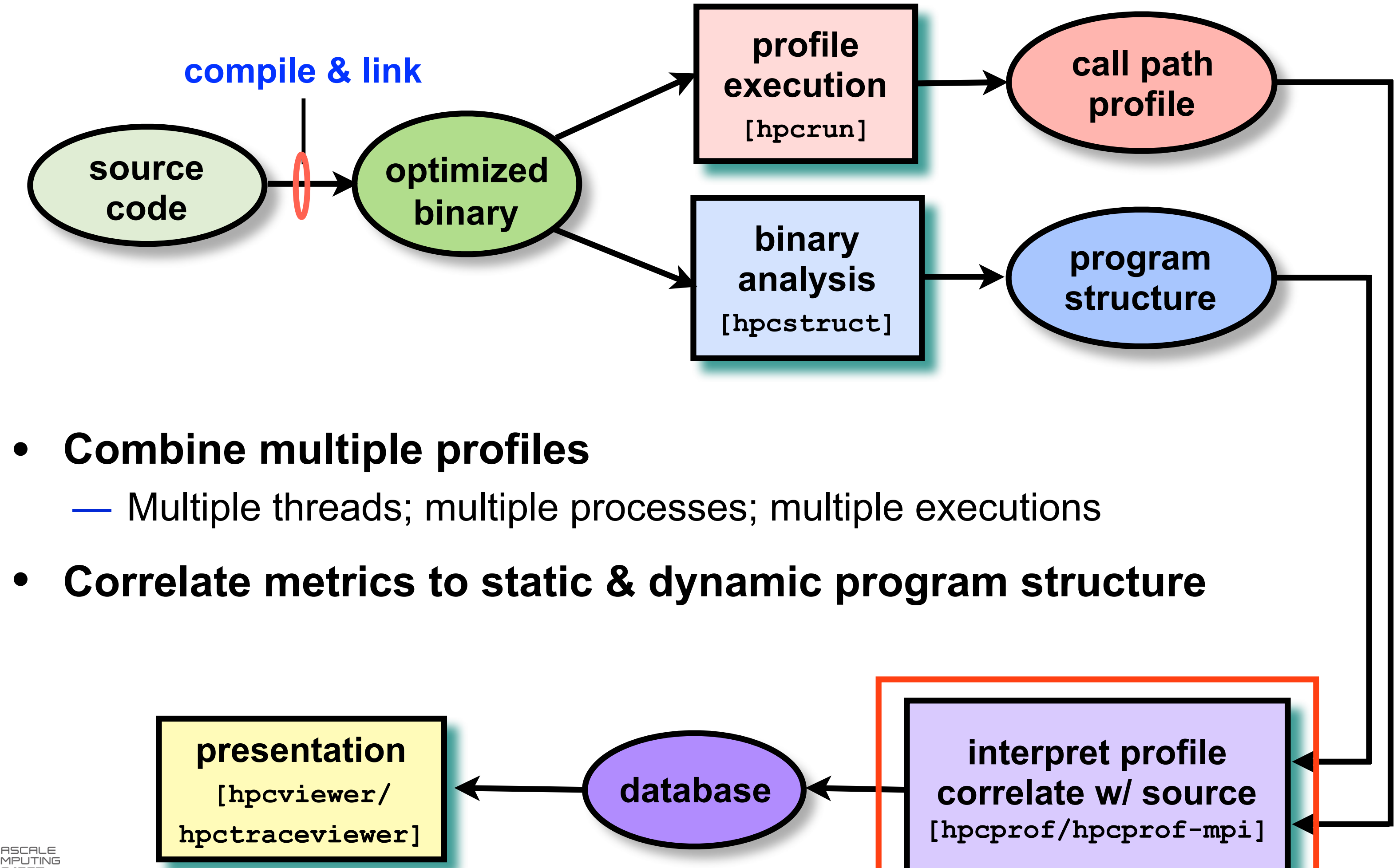
Power/LE

ARM

CUDA

Lead Institution: University of Wisconsin – Madison

HPCToolkit Workflow


compile & link

source code

optimized binary

profile execution [hpcrun]

call path profile

binary analysis [hpcstruct]

program structure

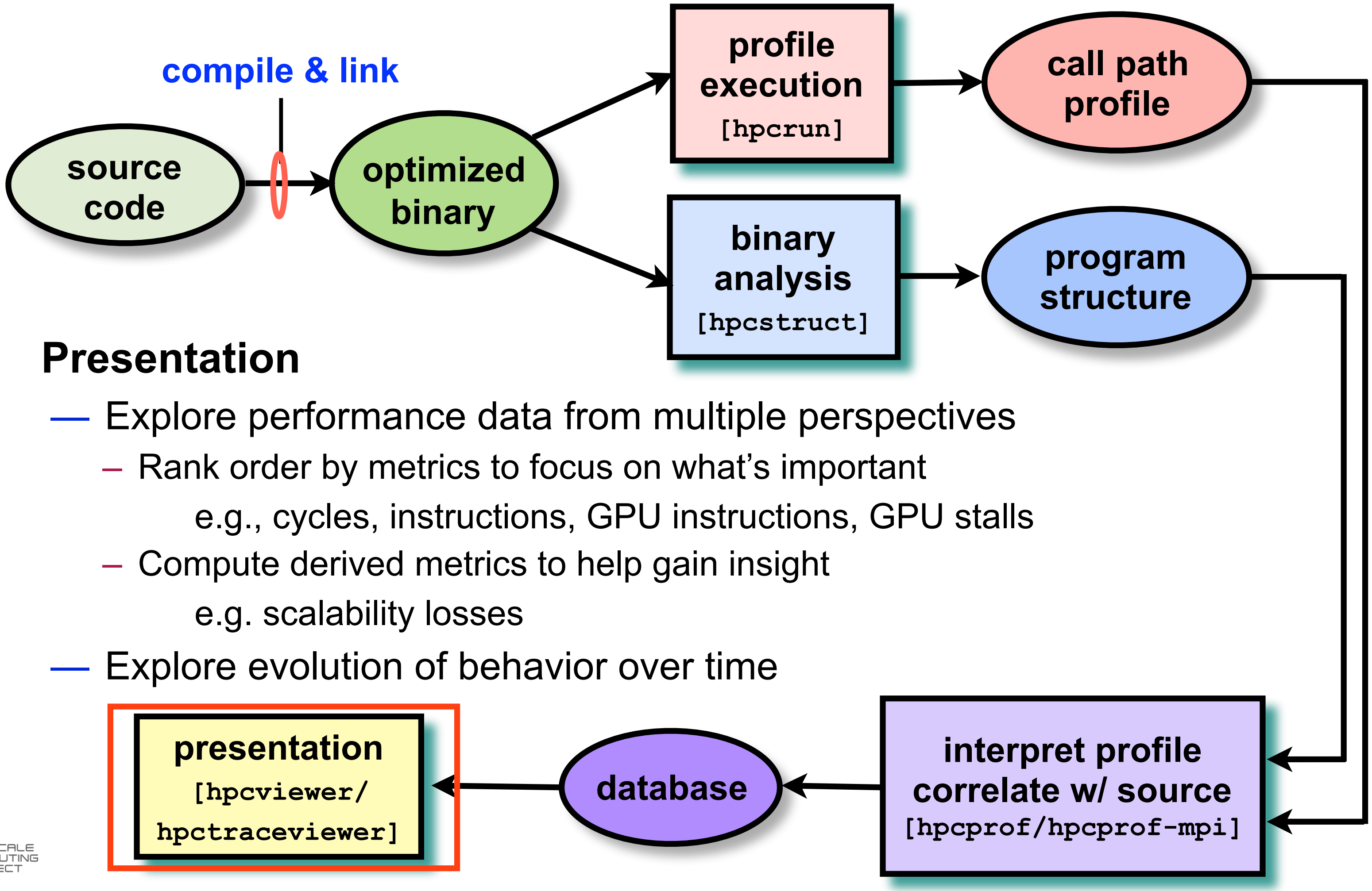
interpret profile correlate w/ source [hpcprof/hpcprof-mpi]

database

presentation [hpcviewer/ hpctraceviewer]

- **Combine multiple profiles**
  - Multiple threads; multiple processes; multiple executions
- **Correlate metrics to static & dynamic program structure**

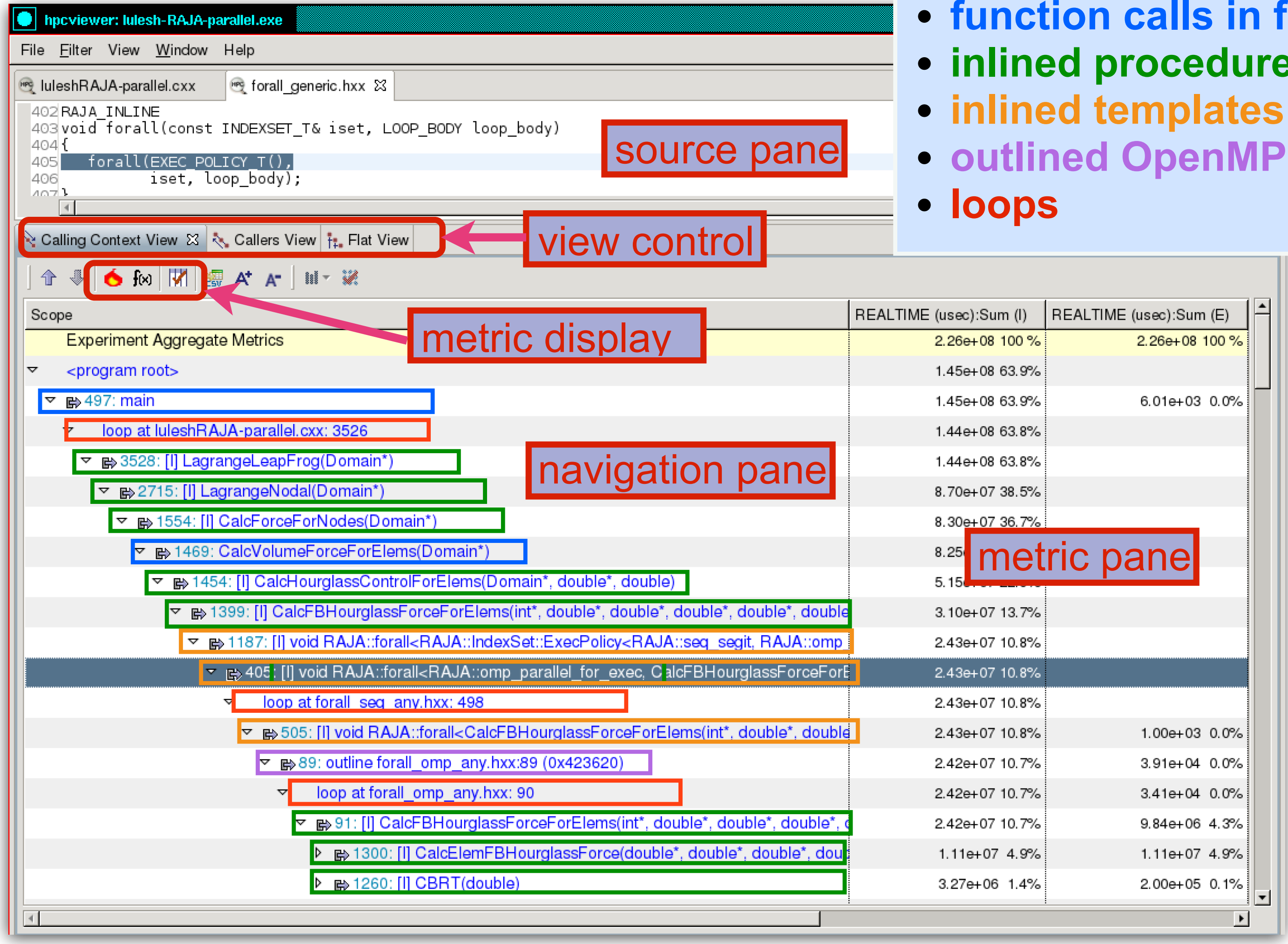
HPCToolkit Workflow



Presentation

- Explore performance data from multiple perspectives
  - Rank order by metrics to focus on what's important
    e.g., cycles, instructions, GPU instructions, GPU stalls
  - Compute derived metrics to help gain insight
    e.g. scalability losses
- Explore evolution of behavior over time

Code-centric Analysis with hpcviewer



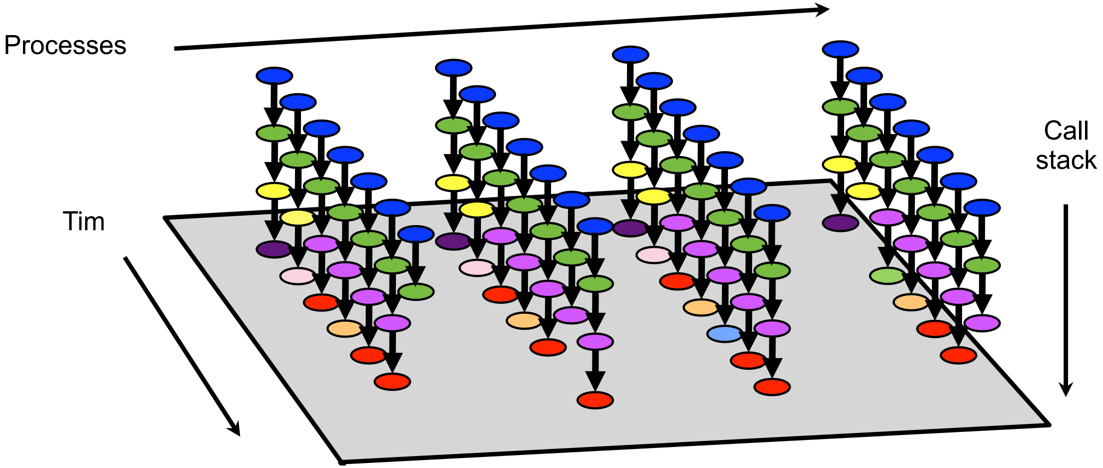
```
RAJA_INLINE
void forall(const INDEXSET_T& iset, LOOP_BODY loop_body)
{
   forall(EXEC_POLICY_T(),
          iset, loop_body);
}
```

```
Scope                                                                     REALTIME (usec):Sum (I)   REALTIME (usec):Sum (E)
Experiment Aggregate Metrics                                                    2.26e+08 100 %            2.26e+08 100 %
<program root>                                                                  1.45e+08 63.9%
  497: main                                                                     1.45e+08 63.9%            6.01e+03  0.0%
    loop at luleshRAJA-parallel.cxx: 3526                                       1.44e+08 63.8%
      3528: [I] LagrangeLeapFrog(Domain*)                                       1.44e+08 63.8%
        2715: [I] LagrangeNodal(Domain*)                                        8.70e+07 38.5%
          1554: [I] CalcForceForNodes(Domain*)                                  8.30e+07 36.7%
            1469: CalcVolumeForceForElems(Domain*)                              8.25
              1454: [I] CalcHourglassControlForElems(Domain*, double*, double)  5.15
                1399: [I] CalcFBHourglassForceForElems(int*, double*, double*, double*, double*, double  3.10e+07 13.7%
                  1187: [I] void RAJA::forall<RAJA::IndexSet::ExecPolicy<RAJA::seq_segit, RAJA::omp_  2.43e+07 10.8%
                    405: [I] void RAJA::forall<RAJA::omp_parallel_for_exec, CalcFBHourglassForceForE  2.43e+07 10.8%
                      loop at forall_seq_any.hxx: 498                           2.43e+07 10.8%
                        505: [I] void RAJA::forall<CalcFBHourglassForceForElems(int*, double*, double  2.43e+07 10.8%   1.00e+03  0.0%
                          89: outline forall_omp_any.hxx:89 (0x423620)          2.42e+07 10.7%            3.91e+04  0.0%
                            loop at forall_omp_any.hxx: 90                      2.42e+07 10.7%            3.41e+04  0.0%
                              91: [I] CalcFBHourglassForceForElems(int*, double*, double*, double*, d  2.42e+07 10.7%   9.84e+06  4.3%
                                1300: [I] CalcElemFBHourglassForce(double*, double*, double*, dou  1.11e+07  4.9%   1.11e+07  4.9%
                                1260: [I] CBRT(double)                          3.27e+06  1.4%            2.00e+05  0.1%
```

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

Understanding Temporal Behavior

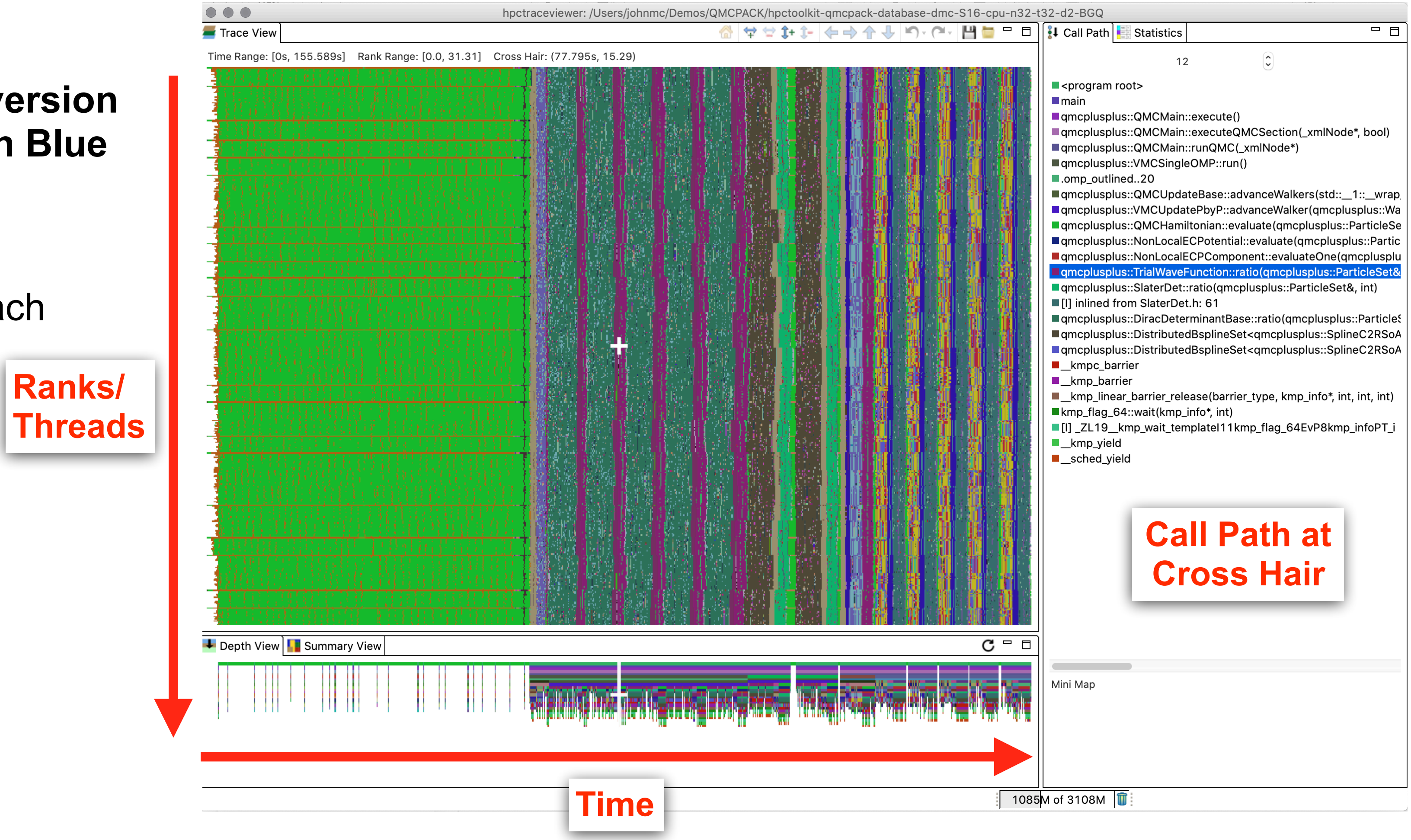
- **Profiling compresses out the temporal dimension**
  - Temporal patterns, e.g. serial sections and dynamic load imbalance are invisible in profiles
- **What can we do? Trace call path samples**
  - N times per second, take a call path sample of each thread
  - Organize the samples for each thread 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 hpctraceviewer

Experimental version of QMCPack on Blue Gene Q

- 32 ranks
- 32 threads each



Time Range: [0s, 155.589s]   Rank Range: [0.0, 31.31]   Cross Hair: (77.795s, 15.29)

12

```
<program root>
main
qmcplusplus::QMCMain::execute()
qmcplusplus::QMCMain::executeQMCSection(_xmlNode*, bool)
qmcplusplus::QMCMain::runQMC(_xmlNode*)
qmcplusplus::VMCSingleOMP::run()
.omp_outlined..20
qmcplusplus::QMCUpdateBase::advanceWalkers(std::__1::__wrap
qmcplusplus::VMCUpdatePbyP::advanceWalker(qmcplusplus::Wa
qmcplusplus::QMCHamiltonian::evaluate(qmcplusplus::ParticleSe
qmcplusplus::NonLocalECPotential::evaluate(qmcplusplus::Partic
qmcplusplus::NonLocalECPComponent::evaluateOne(qmcplusplu
qmcplusplus::TrialWaveFunction::ratio(qmcplusplus::ParticleSet&
qmcplusplus::SlaterDet::ratio(qmcplusplus::ParticleSet&, int)
[I] inlined from SlaterDet.h: 61
qmcplusplus::DiracDeterminantBase::ratio(qmcplusplus::Particle
qmcplusplus::DistributedBsplineSet<qmcplusplus::SplineC2RSoA
qmcplusplus::DistributedBsplineSet<qmcplusplus::SplineC2RSoA
__kmpc_barrier
__kmp_barrier
__kmp_linear_barrier_release(barrier_type, kmp_info*, int, int, int)
kmp_flag_64::wait(kmp_info*, int)
[I] _ZL19__kmp_wait_templateI11kmp_flag_64EvP8kmp_infoPT_i
__kmp_yield
__sched_yield
```

Ranks/ Threads

Time

Call Path at Cross Hair

Outline

- **Performance measurement and analysis challenges for GPU-accelerated supercomputers**
- **Introduction to HPCToolkit performance tools**
  - Overview of HPCToolkit components and their workflow
  - HPCToolkit's graphical user interfaces and using them effectively
- **Analyzing the performance of GPU-accelerated supercomputers with HPCToolkit**
  - Overview of HPCToolkit's GPU performance measurement capabilities
  - Collecting measurements
  - Analysis and attribution
  - Exploring measurements and analysis results
- **Experiences with analysis and tuning of GPU-accelerated codes**
  - Computation, memory hierarchy, and data movement issues
- **Obtaining HPCToolkit**

hpctraceviewer Panes and their Purposes

- **Trace View pane**
  - Displays a sequence of samples for each trace line rendered
  - Title bar shows time interval rendered, rank interval rendered, cross hair location
- **Call Path pane**
  - Show the call path of the selected thread at the cross hair
- **Depth View pane**
  - Show the call stack over time for the thread marked by the cross hair
  - Unusual changes or clustering of deep call stacks can indicate behaviors of potential interest
- **Summary View pane**
  - At each point in time, a histogram of colors above in a vertical column of the Trace View

Rendering Traces with hpctraceviewer

- **hpctraceviewer renders traces by sampling the [rank x time] rectangle in the viewport**
  - Don’t try to summarize activity in a time interval represented by a pixel
  - Just pick the last activity before the sample point in time
- **Cost of rendering a large execution is [H x T lg N] for traces of length N**
  - The number of trace lines that can be rendered is limited by the number of vertical pixels H
  - Binary search along rendered trace lines to extract values for pixels
- **It can be used to analyze large data: thousands of ranks and threads**
  - Data is kept on disk, memory mapped, and read only as needed

Understanding How hpctraceviewer Paints Traces

- **CPU trace lines**
  - Given: (procedure f, t) (procedure g, t’) (procedure h, t’’)
    - Default painting algorithm
      - paint color “f” in [t,t’); paint color “g” in [t’, t’’)
    - Midpoint painting algorithm
      - paint color “f” in [t, (t+t’)/2); paint color “g” in [(t+t’)/2, (t’+t’’)/2)
- **GPU trace lines**
  - Given GPU operations “f” in interval [t, t’) and and “g” in interval [t’’, t’’’)
    - paint color “f” in [t, t’); paint color white in [t’, t’’); paint color “g” in [t’’, t’’’)

Analysis Strategies with Time-centric hpctraceviewer

- **Use top-down analysis to understand the broad characteristics of the parallel execution**
- **Click on a point of interest in the Trace View to see the call path there**
- **Zoom in on individual phases of the execution or more generally subsets of [rank, time]**
  - The mini-map tracks what subset of the execution you are viewing
- **Home, undo, redo buttons allow you to move back and forth in a sequence of zooms**
- **Drill down the call path to see what is going on at the call path leaves**
  - Hold your mouse over the call path depth selector. a tool tip will tell you the maximum depth
  - Type the maximum call stack depth number into the depth selector
- **Use the summary view to see a histogram about what fraction of threads or ranks is doing at each time**
- **The summary view can facilitate analysis of how behavior changes over time**
- **The statistics view can show you the fraction of [rank x time] spent in each procedure at the selected depth level**

Understanding the Navigation Pane in Code-centric hpcviewer

- **<program root>: the top of the call chain for the executable**
- **<thread root>: the top of the call chain for any pthreads**
- **<partial call paths>**
  - The presence of partial call paths indicates that hpcrun was unable to fully unwind the call stack
  - Even if a large fraction of call paths are “partial” unwinds, bottom-up and flat views can be very informative
- **Sometimes functions appear in the navigation pane and appear to be a root**
  - This means that hpcrun believed that the unwind was complete and successful
  - Ideally, this would have been placed under <partial call paths>

Understanding the Navigation Pane in Code-centric hpcviewer

- **Treat inlined functions as if regular functions**
- **Calling an inlined function**

380 [I] boost::unique_lock<Dyninst::dyn_mutex>::unique_lock(Dyninst::dyn_mutex&)

[I] is a tag used to indicate that the called function is inlined

callsite is a hyperlink to the file and source line where the inlined function is called

callee is a hyperlink to the definition of the inlined function

- **If no source file is available, the caller line number and the callee will be in black**

Analysis Strategies with Code-centric hpcviewer

- **Use top-down analysis to understand the broad characteristics of the execution**
  - Are there specific unique subtrees in the computation that use or waste a lot of resources?
  - Select a costly node and drill down the “hottest path” rooted there with the flame button
  - One can select a node other than the root and use the flame button to look in its subtree
  - Hold your mouse over a long name in the navigation pane to see the full name in a tool tip
- **Use bottom-up analysis to identify costly procedures and their callers**
  - Pick a metric of interest, e.g. cycles
  - Sort by cycles in descending order
  - Pick the top routine and use the flame button to look up the call stack to its callers
  - Repeat for a few routines of particular interest, e.g. network wait, lock wait, memory alloc, …
- **Use the flat view to explore the full costs associated with code at various granularities**
  - Sort by a cost of interest; use the flame button to explore an interesting load module
  - Use the “flatten” button to melt away load modules, files, and functions to identify the most costly loop

Outline

- **Performance measurement and analysis challenges for GPU-accelerated supercomputers**
- **Introduction to HPCToolkit performance tools**
  - Overview of HPCToolkit components and their workflow
  - HPCToolkit's graphical user interfaces and using them effectively
- **Analyzing the performance of GPU-accelerated supercomputers with HPCToolkit**
  - Overview of HPCToolkit's GPU performance measurement capabilities
  - Collecting measurements
  - Analysis and attribution
  - Exploring measurements and analysis results
- **Experiences with analysis and tuning of GPU-accelerated codes**
  - Computation, memory hierarchy, and data movement issues
- **Obtaining HPCToolkit**

Measurement and Analysis of GPU-accelerated Computations

- **What HPCToolkit GUIs present for GPU-accelerated applications**
  - Profile views displaying call paths that integrate CPU and GPU call paths
  - Trace views that attribute CPU threads and GPU streams to full heterogeneous call paths
- **What HPCToolkit collects**
  - Heterogeneous call path profiles and call path traces
- **How HPCToolkit collects information**
  - CPU
    - Sampling-based measurement of application thread activity in user space and in the kernel
    - Measurement of blocking time using Linux perf_events context switch notifications
  - GPU
    - Coarse-grain measurement of GPU operations (memory copies, kernel launches, …)
    - Fine-grain measurement of GPU kernels using PC Sampling (NVIDIA only)

GPU Monitoring Capabilities of HPCToolkit

Measurement Capability	NVIDIA	AMD
kernel launches, explicit memory copies, synchronization	**callbacks + activity API**	**callbacks + Activity API**
instruction-level measurement and analysis	**PC sampling, analysis of GPU binaries**	**no**
kernel characteristics	**Activity API**	**(available statically)**

Intel oneAPI Level 0 specification released in December (not widely known)
https://spec.oneapi.com/versions/latest/oneL0/index.html

Preparing a GPU-accelerated Program for HPCToolkit

- **HPCToolkit doesn't need any modifications to your Makefiles**
  - it can measure fully-optimized code without special preparation
- **To get the most from your measurement and analysis**
  - Compile your program with line numbers
    - CPU (all compilers)
      - add "-g" to your compiler optimization flags
    - NVIDIA GPUs
      - compiling with nvcc
        - add "-lineinfo" to your optimization flags for GPU line numbers
        - adding -G provides full information about inlining and GPU code structure but disables optimization
      - compiling with xlc
        - line information is unavailable for optimized code
    - AMD GPUs, no special preparation needed
      - current AMD GPUs and ROCM software stack lack capabilities for fine-grain measurement and attribution
    - Intel GPUs
      - HPCToolkit is currently oblivious to their presence

Using HPCToolkit to Measure an Execution

- **Sequential program**
  - `hpcrun [measurement options] program [program args]`
- **Parallel program**
  - `mpirun -n <nodes> [mpi options] hpcrun [measurement options] \`
    `program [program args]`
  - Similar launches with job managers
    - LSF: jsrun
    - SLURM: srun

CPU Time-based Sample Sources - Linux thread-centric timers

- **CPUTIME (DEFAULT if no sample source is specified)**
  - CPU time used by the thread in microseconds
  - Does not include time blocked in the kernel
    - **disadvantage: completely overlooks time a thread is blocked**
    - **advantage: a blocked thread is never unblocked by sampling**
- **REALTIME**
  - Real time used by the thread in microseconds
  - Includes time blocked in the kernel
    - **advantage: shows where a thread spends its time, even when blocked**
    - **disadvantages**
      - **activates a blocked thread to take a sample**
      - **a blocked thread appears active even when blocked**

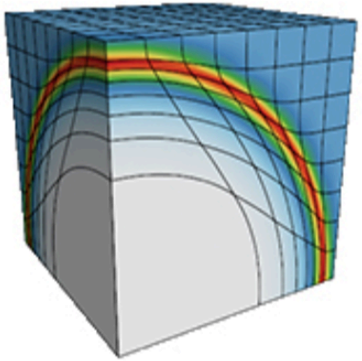
Note: Only use one Linux timer to measure an execution

CPU Sample Sources - Linux perf_event monitoring subsystem

- **Kernel subsystem for performance monitoring**
- **Access and manipulate**
  - Hardware counters: cycles, instructions, …
  - Software counters: context switches, page faults, …
- **Available in Linux kernels 2.6.31+**
- **Characteristics**
  - Monitors activity in user space and in the kernel
    - Can see costs in GPU drivers

Case Study: Measurement and Analysis of GPU-accelerated Laghos

Laghos (LAGrangian High-Order Solver) is a LLNL ASC co-design mini-app that was developed as part of the CEED software suite, a collection of software benchmarks, miniapps, libraries and APIs for efficient exascale discretization based on high-order finite element and spectral element methods.



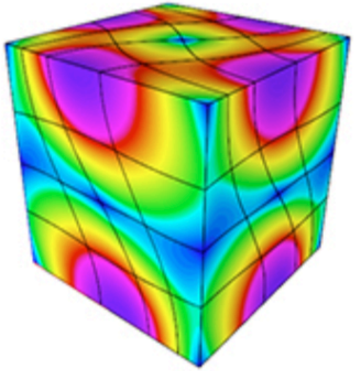


Figure credit: https://computing.llnl.gov/projects/co-design/laghos

Applying the GPU Operation Measurement Workflow to Laghos

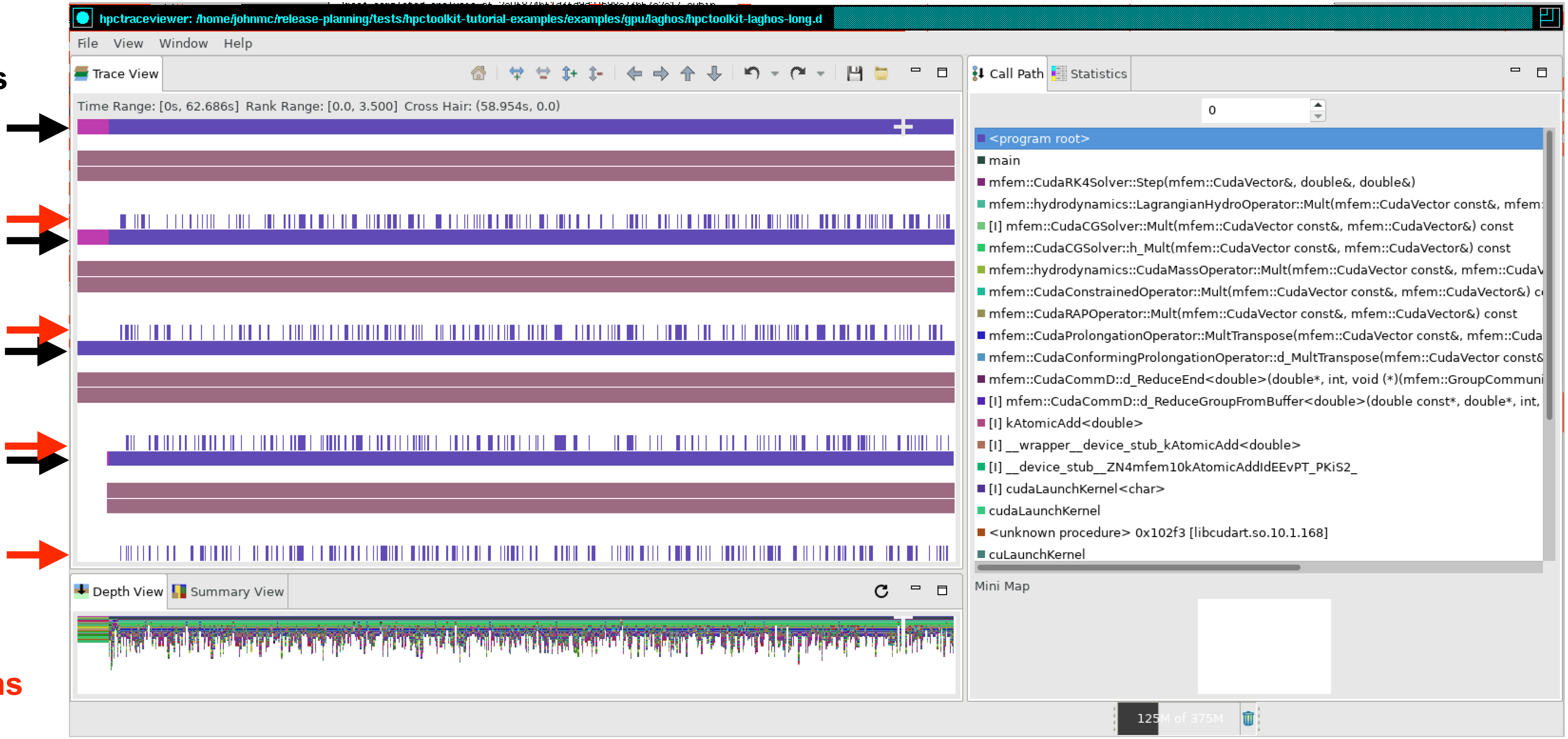
```
# measure an execution of laghos
time mpirun -np 4 hpcrun -o $OUT -e cycles -e gpu=nvidia -t \
   ${LAGHOS_DIR}/laghos -p 0 -m ${LAGHOS_DIR}/../data/square01_quad.mesh \
   -rs 3 -tf 0.75 -pa

# compute program structure information for the laghos binary
hpcstruct -j 16 laghos

# compute program structure information for the laghos cubins
hpcstruct -j 16 $OUT

# combine the measurements with the program structure information
mpirun -n 4  hpcprof-mpi -S laghos.hpcstruct $OUT
```

Computing Program Structure Information for NVIDIA cubins

- **When a GPU-accelerated application runs, HPCToolkit collects unique GPU binaries**
  - Currently, NVIDIA does not provide an API that provides a URI for cubins it launches
  - CUPTI presents cubins to tools as an interval in the heap (starting address, length)
  - HPCToolkit computes an MD5 hash for each cubin and saves one copy
    - stores save cubins in hpcrun's measurement directory: <measurement directory>/cubins
- **Analyze the cubins collected during an execution**
  - `hpcstruct -j 16 <measurement directory>`
    - lightweight analysis based only on cubin symbols and line map
  - `hpcstruct -j 16 —gpucfg yes <measurement directory>`
    - heavyweight analysis based only on cubin symbols, line map, control flow graph
      - uses nvdisasm to compute control flow graph
    - fine-grain analysis only needed to interpret PC sampling experiments
  - `hpcstruct` analyzes cubins in parallel using thread count specified with -j

Initial hpctraceviewer view of Laghos (long) Execution

MPI Ranks



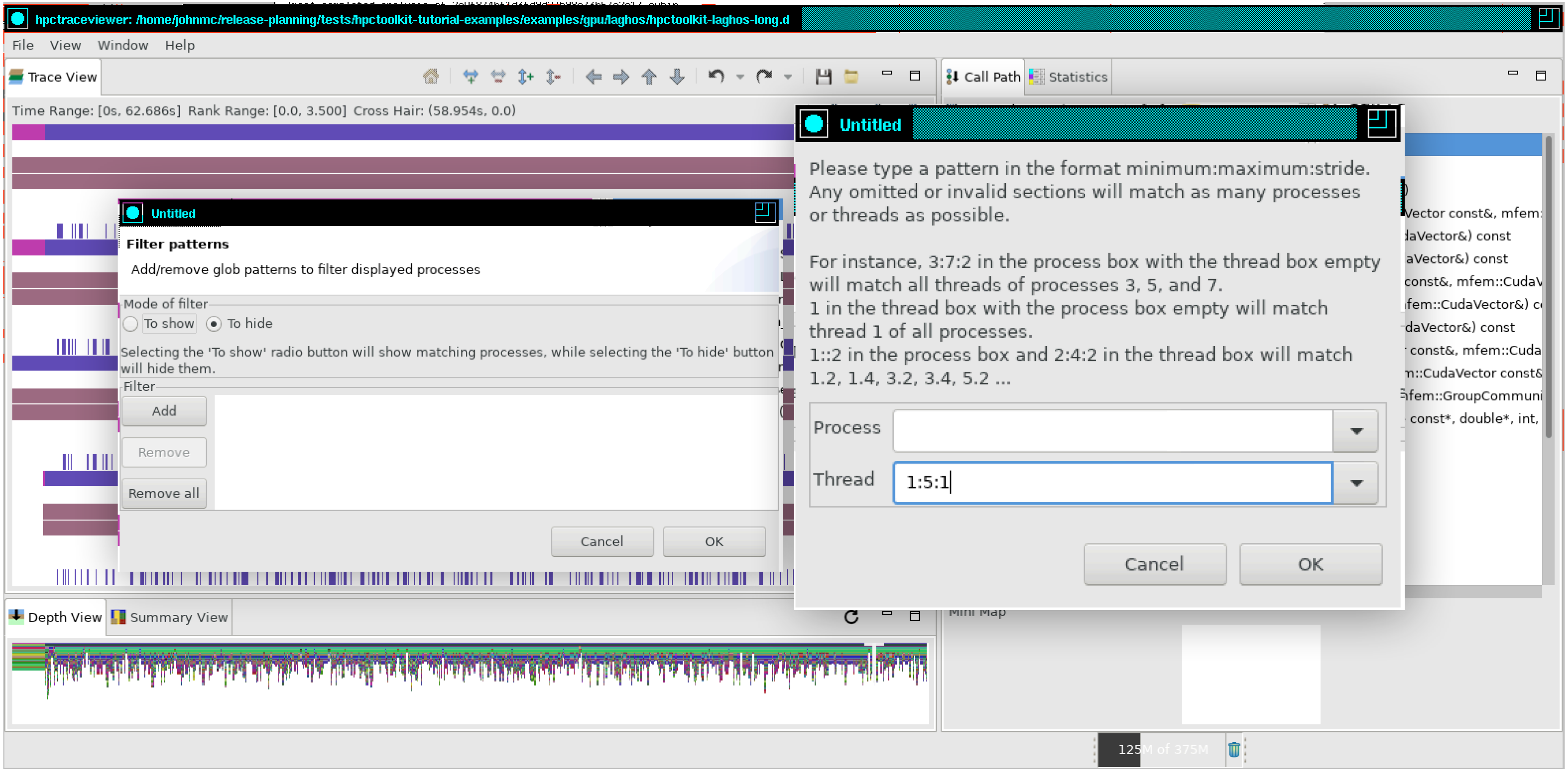
Time Range: [0s, 62.686s] Rank Range: [0.0, 3.500] Cross Hair: (58.954s, 0.0)

0

- <program root>
- main
- mfem::CudaRK4Solver::Step(mfem::CudaVector&, double&, double&)
- mfem::hydrodynamics::LagrangianHydroOperator::Mult(mfem::CudaVector const&, mfem:
- [I] mfem::CudaCGSolver::Mult(mfem::CudaVector const&, mfem::CudaVector&) const
- mfem::CudaCGSolver::h_Mult(mfem::CudaVector const&, mfem::CudaVector&) const
- mfem::hydrodynamics::CudaMassOperator::Mult(mfem::CudaVector const&, mfem::CudaV
- mfem::CudaConstrainedOperator::Mult(mfem::CudaVector const&, mfem::CudaVector&) c
- mfem::CudaRAPOperator::Mult(mfem::CudaVector const&, mfem::CudaVector&) const
- mfem::CudaProlongationOperator::MultTranspose(mfem::CudaVector const&, mfem::Cuda
- mfem::CudaConformingProlongationOperator::d_MultTranspose(mfem::CudaVector const&
- mfem::CudaCommD::d_ReduceEnd<double>(double*, int, void (*)(mfem::GroupCommuni
- [I] mfem::CudaCommD::d_ReduceGroupFromBuffer<double>(double const*, double*, int,
- [I] kAtomicAdd<double>
- [I] __wrapper__device_stub_kAtomicAdd<double>
- [I] __device_stub__ZN4mfem10kAtomicAddIdEEvPT_PKiS2_
- [I] cudaLaunchKernel<char>
- cudaLaunchKernel
- <unknown procedure> 0x102f3 [libcudart.so.10.1.168]
- cuLaunchKernel

GPU Streams

Hiding the Empty MPI Helper Threads



hpctraceviewer: /home/johnmc/release-planning/tests/hpctoolkit-tutorial-examples/examples/gpu/laghos/hpctoolkit-laghos-long.d

Time Range: [0s, 62.686s] Rank Range: [0.0, 3.500] Cross Hair: (58.954s, 0.0)

Untitled

Filter patterns

Add/remove glob patterns to filter displayed processes

Mode of filter

○ To show ● To hide

Selecting the 'To show' radio button will show matching processes, while selecting the 'To hide' button will hide them.

Filter

Add | Remove | Remove all

Cancel | OK

Untitled

Please type a pattern in the format minimum:maximum:stride. Any omitted or invalid sections will match as many processes or threads as possible.

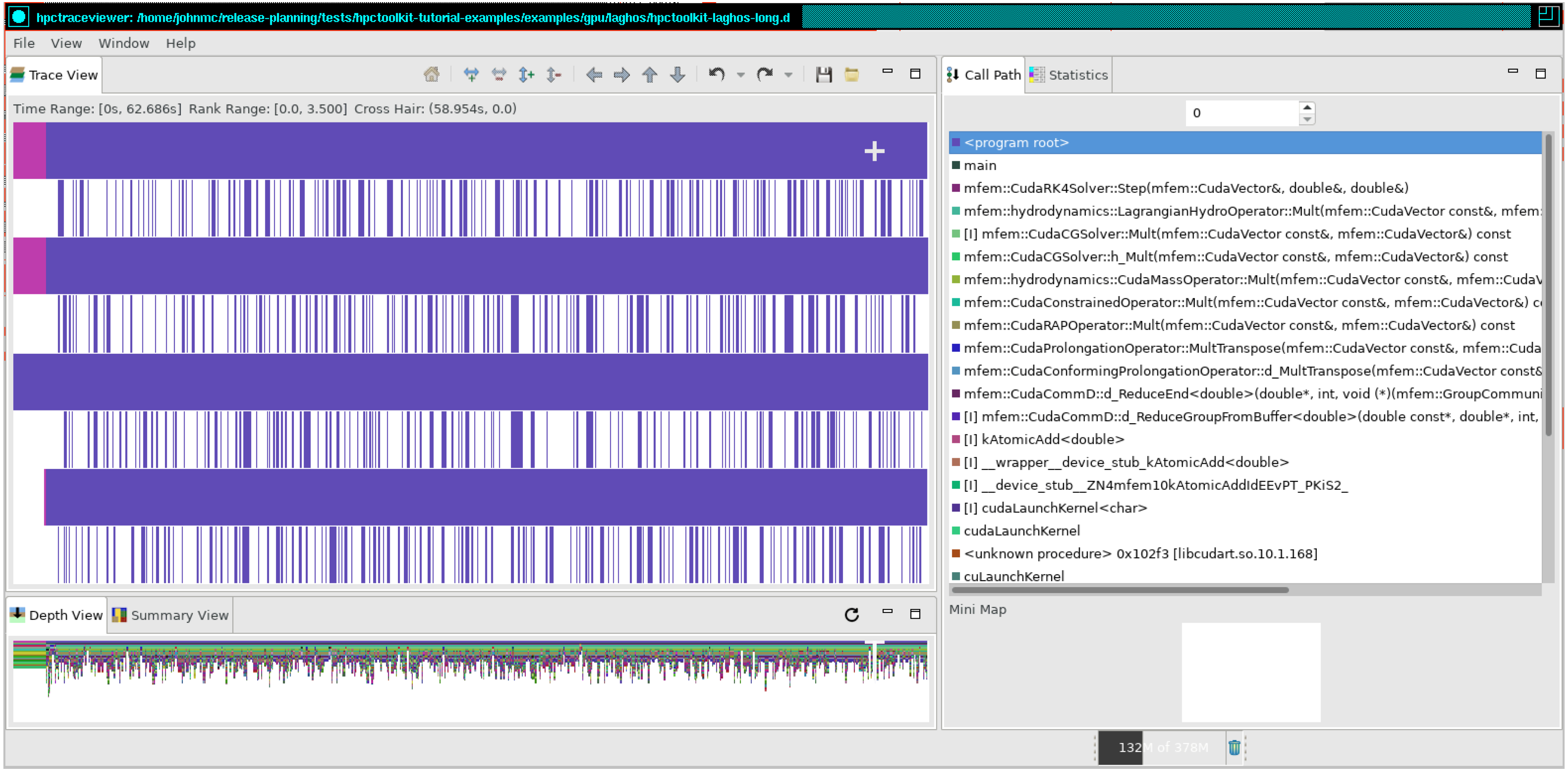
For instance, 3:7:2 in the process box with the thread box empty will match all threads of processes 3, 5, and 7.
1 in the thread box with the process box empty will match thread 1 of all processes.
1::2 in the process box and 2:4:2 in the thread box will match 1.2, 1.4, 3.2, 3.4, 5.2 ...

Process:

Thread: 1:5:1

Cancel | OK

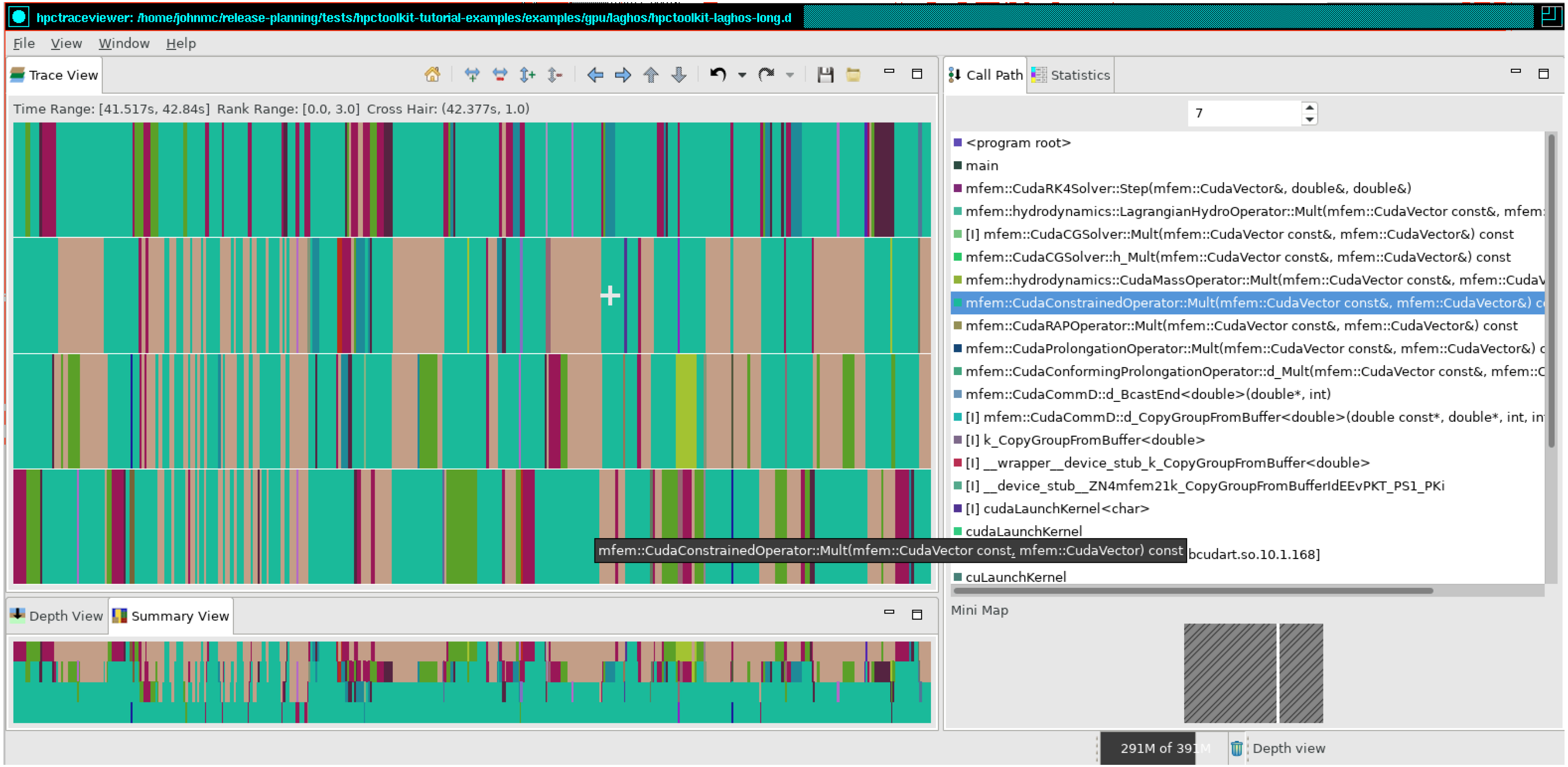
After Hiding the Empty MPI Helper Threads



Time Range: [0s, 62.686s] Rank Range: [0.0, 3.500] Cross Hair: (58.954s, 0.0)

```
0
<program root>
main
mfem::CudaRK4Solver::Step(mfem::CudaVector&, double&, double&)
mfem::hydrodynamics::LagrangianHydroOperator::Mult(mfem::CudaVector const&, mfem:
[I] mfem::CudaCGSolver::Mult(mfem::CudaVector const&, mfem::CudaVector&) const
mfem::CudaCGSolver::h_Mult(mfem::CudaVector const&, mfem::CudaVector&) const
mfem::hydrodynamics::CudaMassOperator::Mult(mfem::CudaVector const&, mfem::CudaV
mfem::CudaConstrainedOperator::Mult(mfem::CudaVector const&, mfem::CudaVector&) c
mfem::CudaRAPOperator::Mult(mfem::CudaVector const&, mfem::CudaVector&) const
mfem::CudaProlongationOperator::MultTranspose(mfem::CudaVector const&, mfem::Cuda
mfem::CudaConformingProlongationOperator::d_MultTranspose(mfem::CudaVector const&
mfem::CudaCommD::d_ReduceEnd<double>(double*, int, void (*)(mfem::GroupCommuni
[I] mfem::CudaCommD::d_ReduceGroupFromBuffer<double>(double const*, double*, int,
[I] kAtomicAdd<double>
[I] __wrapper__device_stub_kAtomicAdd<double>
[I] __device_stub__ZN4mfem10kAtomicAddIdEEvPT_PKiS2_
[I] cudaLaunchKernel<char>
cudaLaunchKernel
<unknown procedure> 0x102f3 [libcudart.so.10.1.168]
cuLaunchKernel
```

A Detail of Only the MPI Threads



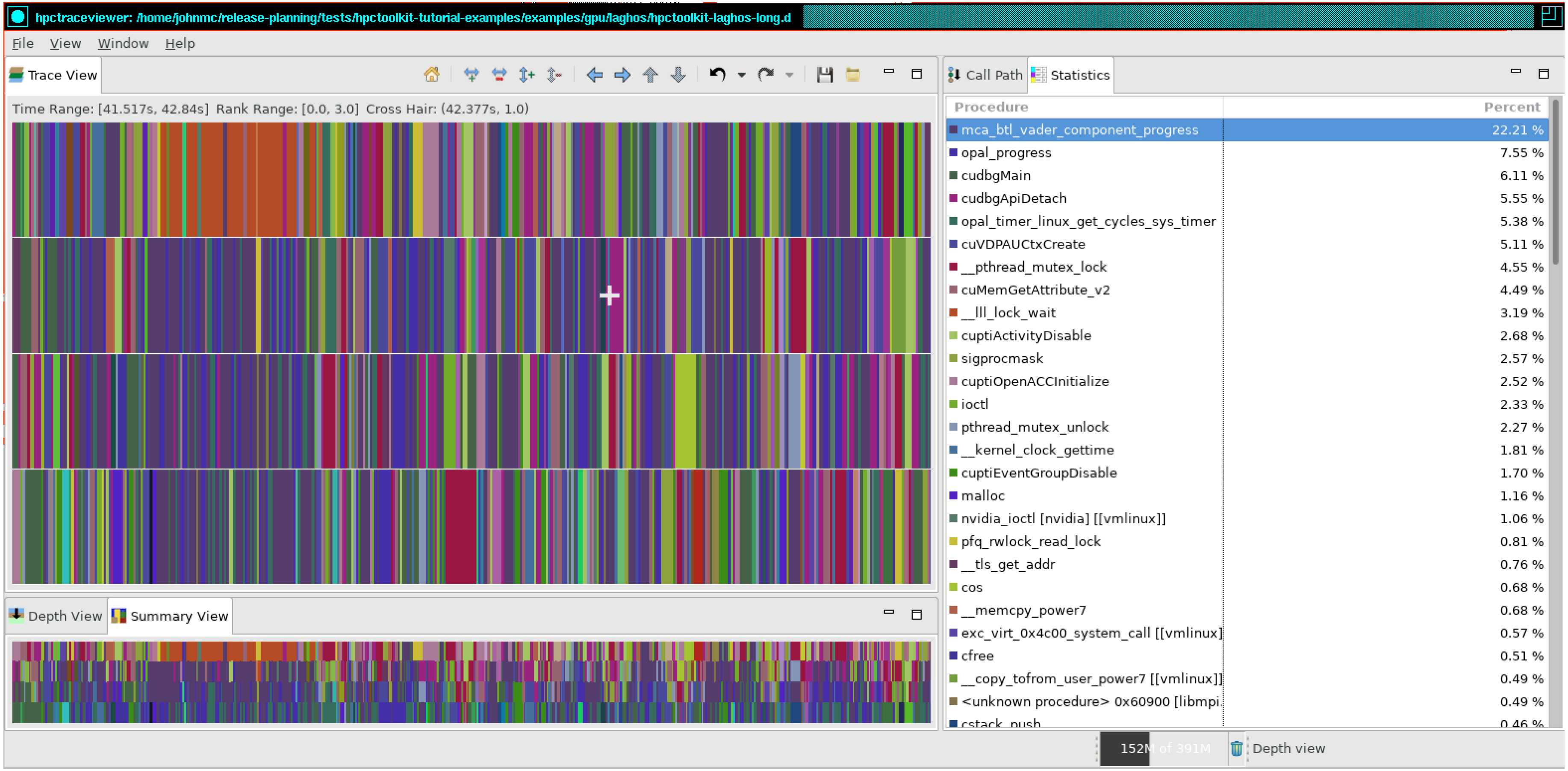
Time Range: [41.517s, 42.84s] Rank Range: [0.0, 3.0] Cross Hair: (42.377s, 1.0)

7

- <program root>
- main
- mfem::CudaRK4Solver::Step(mfem::CudaVector&, double&, double&)
- mfem::hydrodynamics::LagrangianHydroOperator::Mult(mfem::CudaVector const&, mfem:
- [I] mfem::CudaCGSolver::Mult(mfem::CudaVector const&, mfem::CudaVector&) const
- mfem::CudaCGSolver::h_Mult(mfem::CudaVector const&, mfem::CudaVector&) const
- mfem::hydrodynamics::CudaMassOperator::Mult(mfem::CudaVector const&, mfem::CudaV
- mfem::CudaConstrainedOperator::Mult(mfem::CudaVector const&, mfem::CudaVector&) c
- mfem::CudaRAPOperator::Mult(mfem::CudaVector const&, mfem::CudaVector&) const
- mfem::CudaProlongationOperator::Mult(mfem::CudaVector const&, mfem::CudaVector&) c
- mfem::CudaConformingProlongationOperator::d_Mult(mfem::CudaVector const&, mfem::C
- mfem::CudaCommD::d_BcastEnd<double>(double*, int)
- [I] mfem::CudaCommD::d_CopyGroupFromBuffer<double>(double const*, double*, int, in
- [I] k_CopyGroupFromBuffer<double>
- [I] __wrapper__device_stub_k_CopyGroupFromBuffer<double>
- [I] __device_stub__ZN4mfem21k_CopyGroupFromBufferIdEEvPKT_PS1_PKi
- [I] cudaLaunchKernel<char>
- cudaLaunchKernel
- bcudart.so.10.1.168]
- cuLaunchKernel

mfem::CudaConstrainedOperator::Mult(mfem::CudaVector const, mfem::CudaVector) const

Only the MPI Threads - Analysis using the Statistics Panel



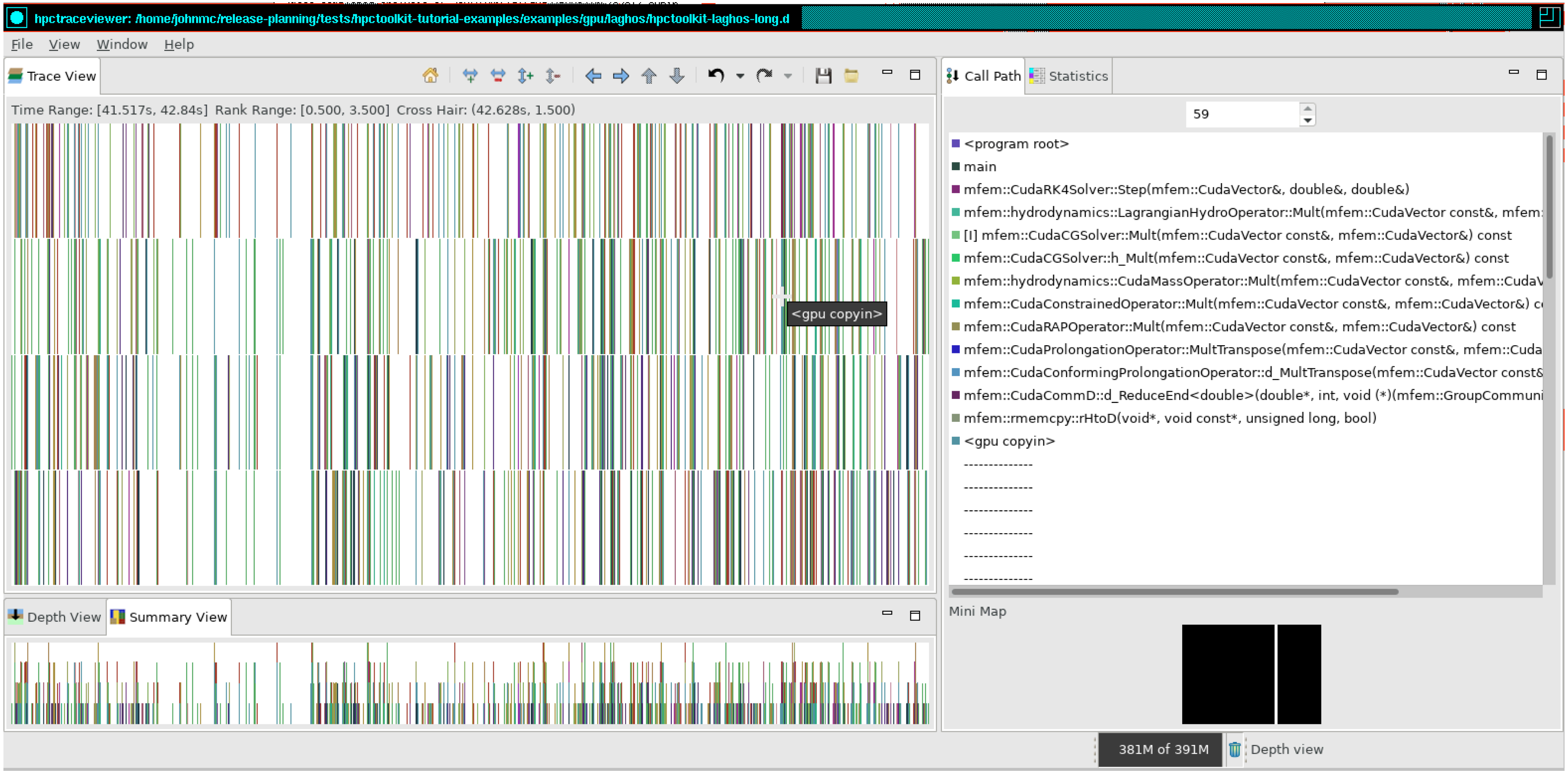
hpctraceviewer: /home/johnmc/release-planning/tests/hpctoolkit-tutorial-examples/examples/gpu/laghos/hpctoolkit-laghos-long.d

Time Range: [41.517s, 42.84s] Rank Range: [0.0, 3.0] Cross Hair: (42.377s, 1.0)

Procedure	Percent
mca_btl_vader_component_progress	22.21 %
opal_progress	7.55 %
cudbgMain	6.11 %
cudbgApiDetach	5.55 %
opal_timer_linux_get_cycles_sys_timer	5.38 %
cuVDPAUCtxCreate	5.11 %
__pthread_mutex_lock	4.55 %
cuMemGetAttribute_v2	4.49 %
__lll_lock_wait	3.19 %
cuptiActivityDisable	2.68 %
sigprocmask	2.57 %
cuptiOpenACCInitialize	2.52 %
ioctl	2.33 %
pthread_mutex_unlock	2.27 %
__kernel_clock_gettime	1.81 %
cuptiEventGroupDisable	1.70 %
malloc	1.16 %
nvidia_ioctl [nvidia] [[vmlinux]]	1.06 %
pfq_rwlock_read_lock	0.81 %
__tls_get_addr	0.76 %
cos	0.68 %
__memcpy_power7	0.68 %
exc_virt_0x4c00_system_call [[vmlinux]]	0.57 %
cfree	0.51 %
__copy_tofrom_user_power7 [[vmlinux]]	0.49 %
<unknown procedure> 0x60900 [libmpi.	0.49 %
cstack_push	0.46 %

152M of 391M

Only the GPU Threads - Inspecting the Callpath for a Kernel



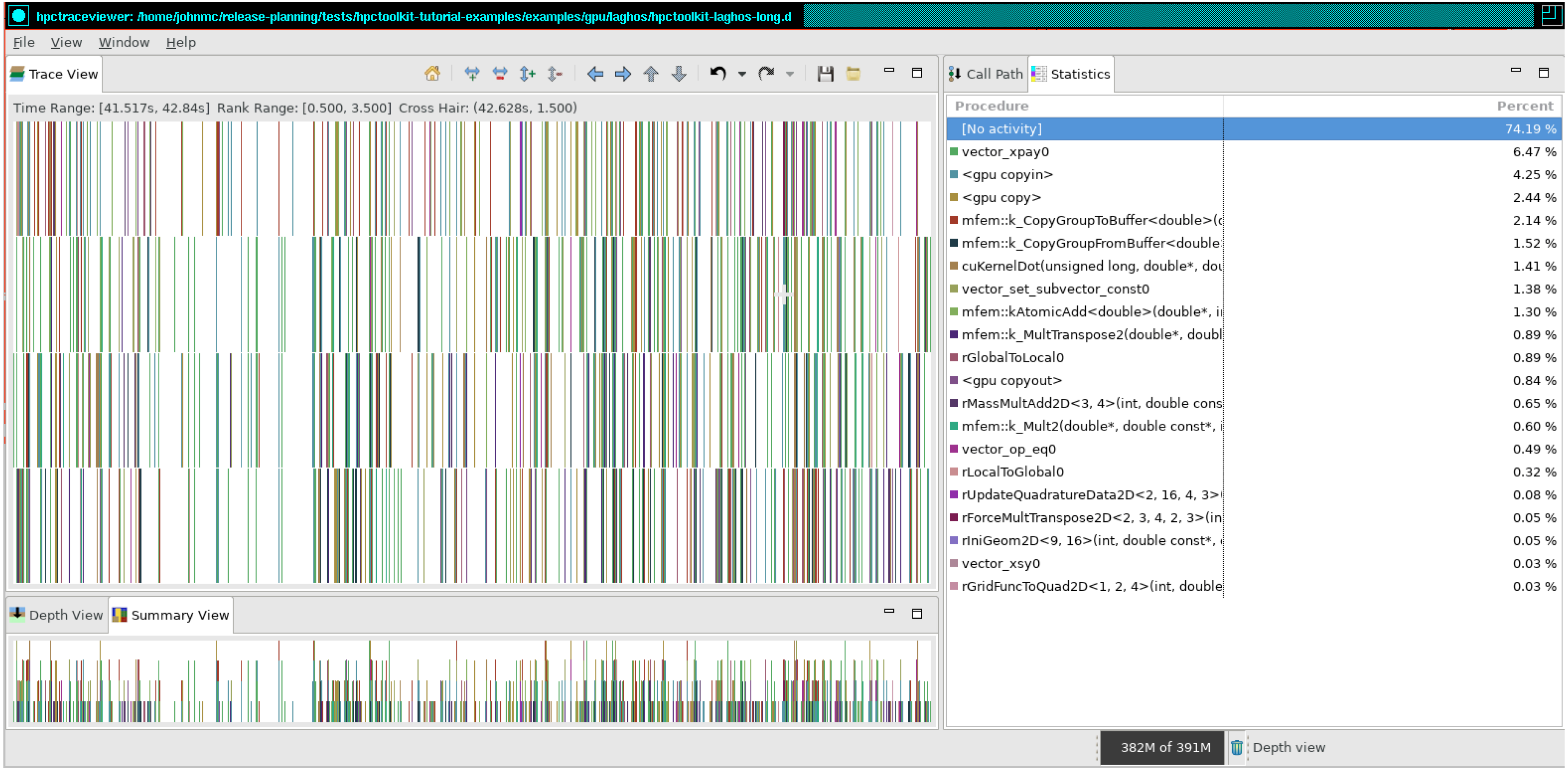
hpctraceviewer: /home/johnmc/release-planning/tests/hpctoolkit-tutorial-examples/examples/gpu/laghos/hpctoolkit-laghos-long.d

Time Range: [41.517s, 42.84s] Rank Range: [0.500, 3.500] Cross Hair: (42.628s, 1.500)

<gpu copyin>

■ <program root>
■ main
■ mfem::CudaRK4Solver::Step(mfem::CudaVector&, double&, double&)
■ mfem::hydrodynamics::LagrangianHydroOperator::Mult(mfem::CudaVector const&, mfem:
■ [I] mfem::CudaCGSolver::Mult(mfem::CudaVector const&, mfem::CudaVector&) const
■ mfem::CudaCGSolver::h_Mult(mfem::CudaVector const&, mfem::CudaVector&) const
■ mfem::hydrodynamics::CudaMassOperator::Mult(mfem::CudaVector const&, mfem::CudaV
■ mfem::CudaConstrainedOperator::Mult(mfem::CudaVector const&, mfem::CudaVector&) c
■ mfem::CudaRAPOperator::Mult(mfem::CudaVector const&, mfem::CudaVector&) const
■ mfem::CudaProlongationOperator::MultTranspose(mfem::CudaVector const&, mfem::Cuda
■ mfem::CudaConformingProlongationOperator::d_MultTranspose(mfem::CudaVector const&
■ mfem::CudaCommD::d_ReduceEnd<double>(double*, int, void (*)(mfem::GroupCommuni
■ mfem::rmemcpy::rHtoD(void*, void const*, unsigned long, bool)
■ <gpu copyin>

Only the GPU Threads - Analysis Using the Statistics Panel



hpctraceviewer: /home/johnmc/release-planning/tests/hpctoolkit-tutorial-examples/examples/gpu/laghos/hpctoolkit-laghos-long.d

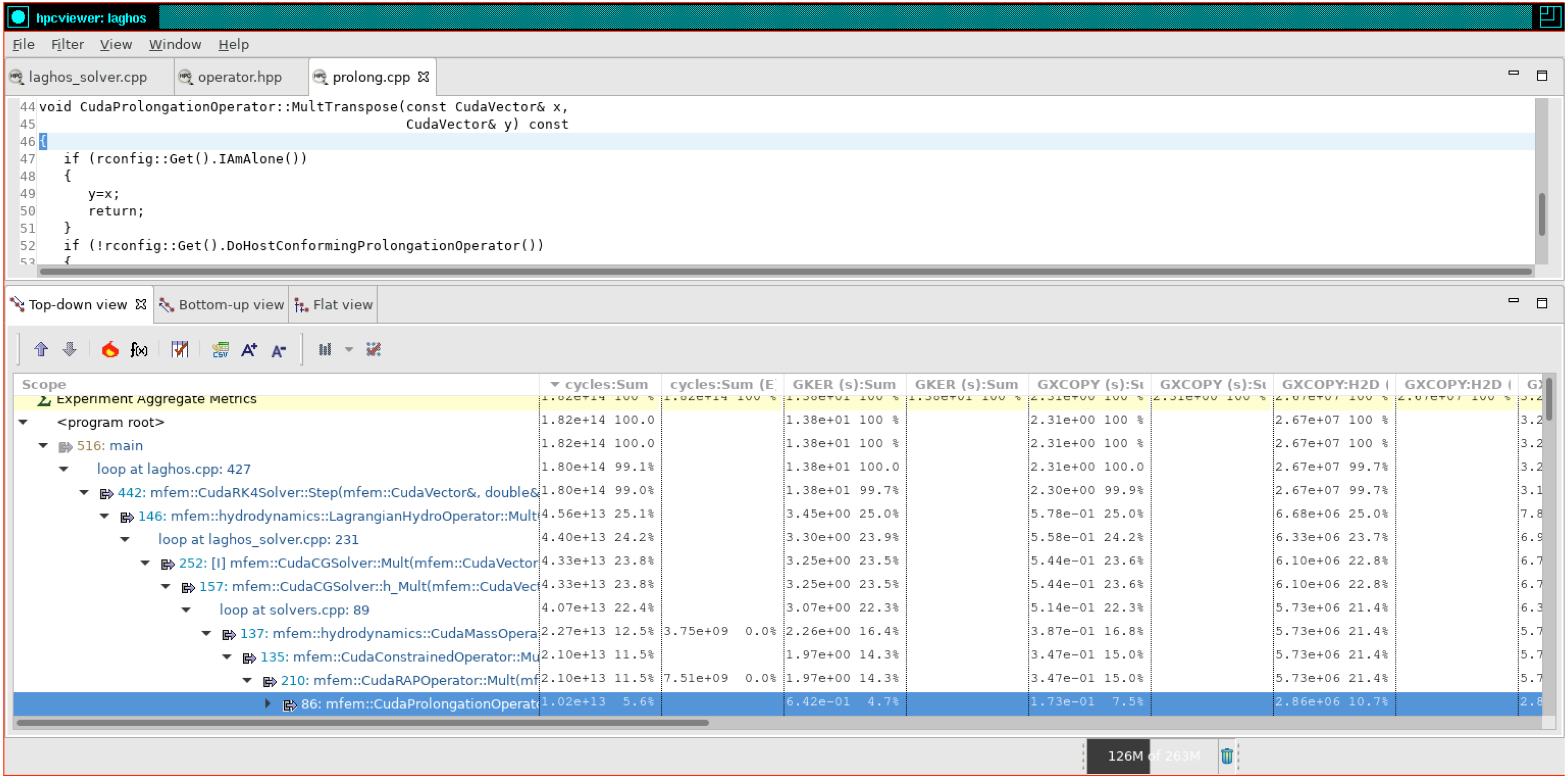
Time Range: [41.517s, 42.84s] Rank Range: [0.500, 3.500] Cross Hair: (42.628s, 1.500)

Procedure	Percent
[No activity]	74.19 %
vector_xpay0	6.47 %
<gpu copyin>	4.25 %
<gpu copy>	2.44 %
mfem::k_CopyGroupToBuffer<double>(d	2.14 %
mfem::k_CopyGroupFromBuffer<double	1.52 %
cuKernelDot(unsigned long, double*, dou	1.41 %
vector_set_subvector_const0	1.38 %
mfem::kAtomicAdd<double>(double*, i	1.30 %
mfem::k_MultTranspose2(double*, doubl	0.89 %
rGlobalToLocal0	0.89 %
<gpu copyout>	0.84 %
rMassMultAdd2D<3, 4>(int, double cons	0.65 %
mfem::k_Mult2(double*, double const*, i	0.60 %
vector_op_eq0	0.49 %
rLocalToGlobal0	0.32 %
rUpdateQuadratureData2D<2, 16, 4, 3>(	0.08 %
rForceMultTranspose2D<2, 3, 4, 2, 3>(in	0.05 %
rIniGeom2D<9, 16>(int, double const*, d	0.05 %
vector_xsy0	0.03 %
rGridFuncToQuad2D<1, 2, 4>(int, double	0.03 %

Some Cautions When Analyzing GPU Traces

- **There are overheads introduced by NVIDIA’s monitoring API that we can’t avoid**
- **When analyzing traces from your program and compare GPU activity to [no activity]**
  - Time your program without any tools
  - Time your program when tracing with HPCToolkit or nvprof
  - Re-weight [no activity] by the ratio of unmonitored time to monitored time
- **While this is a concern for traces, this should be less a concern for profiles**
  - On the CPU, HPCToolkit compensates for monitoring overhead in profiles by not measuring it

Using hpcviewer to See the Source-centric View

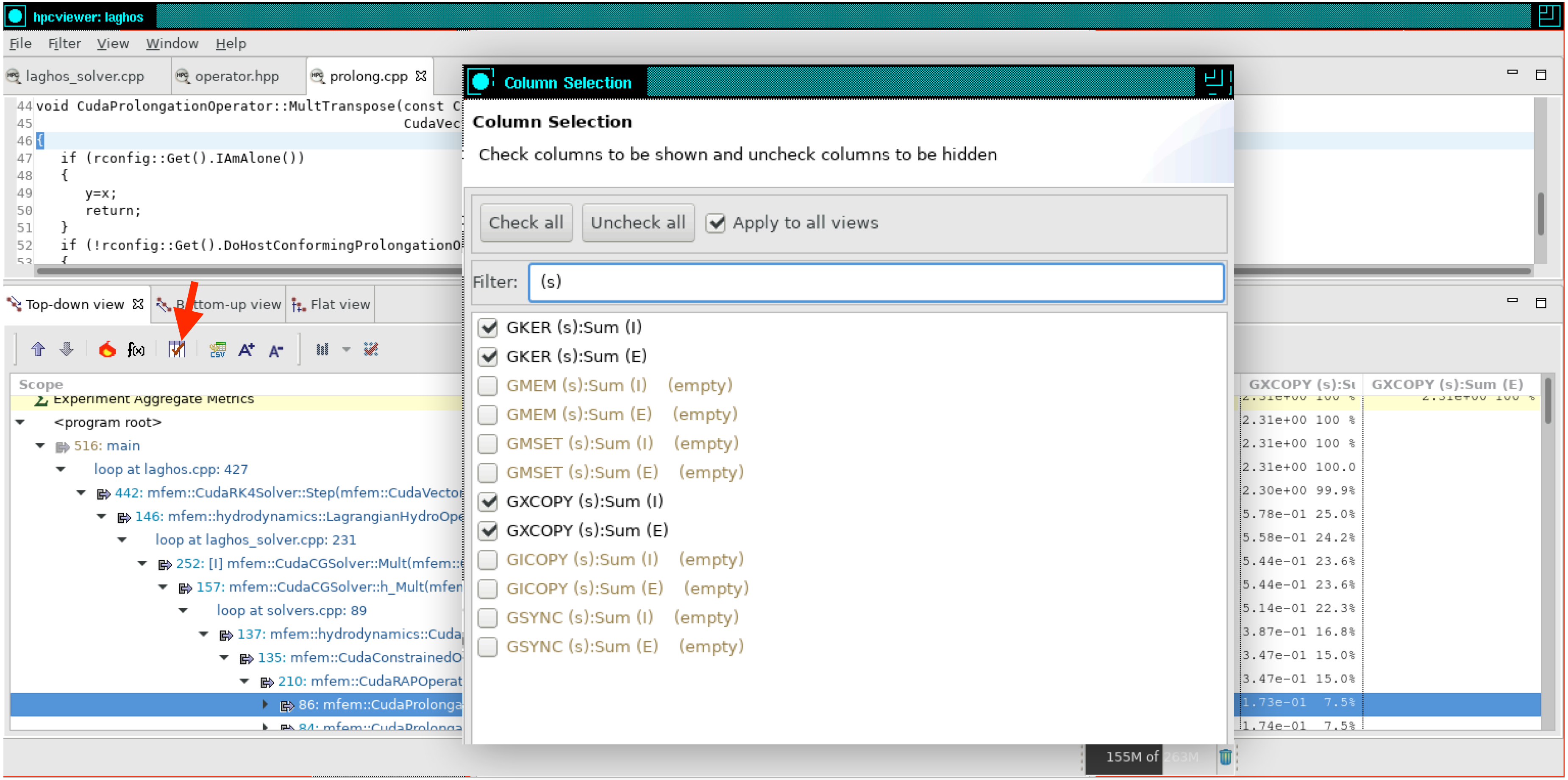


```
void CudaProlongationOperator::MultTranspose(const CudaVector& x,
                                             CudaVector& y) const
{
   if (rconfig::Get().IAmAlone())
   {
      y=x;
      return;
   }
   if (!rconfig::Get().DoHostConformingProlongationOperator())
   {
```

Scope	cycles:Sum	cycles:Sum (E)	GKER (s):Sum	GKER (s):Sum	GXCOPY (s):Su	GXCOPY (s):Su	GXCOPY:H2D (	GXCOPY:H2D (	GX
Experiment Aggregate Metrics	1.82e+14 100 %	1.82e+14 100 %	1.38e+01 100 %	1.38e+01 100 %	2.31e+00 100 %	2.31e+00 100 %	2.67e+07 100 %	2.67e+07 100 %	3.2
<program root>	1.82e+14 100.0		1.38e+01 100 %		2.31e+00 100 %		2.67e+07 100 %		3.2
516: main	1.82e+14 100.0		1.38e+01 100 %		2.31e+00 100 %		2.67e+07 100 %		3.2
loop at laghos.cpp: 427	1.80e+14 99.1%		1.38e+01 100.0		2.31e+00 100.0		2.67e+07 99.7%		3.2
442: mfem::CudaRK4Solver::Step(mfem::CudaVector&, double&	1.80e+14 99.0%		1.38e+01 99.7%		2.30e+00 99.9%		2.67e+07 99.7%		3.1
146: mfem::hydrodynamics::LagrangianHydroOperator::Mult	4.56e+13 25.1%		3.45e+00 25.0%		5.78e-01 25.0%		6.68e+06 25.0%		7.8
loop at laghos_solver.cpp: 231	4.40e+13 24.2%		3.30e+00 23.9%		5.58e-01 24.2%		6.33e+06 23.7%		6.9
252: [I] mfem::CudaCGSolver::Mult(mfem::CudaVector	4.33e+13 23.8%		3.25e+00 23.5%		5.44e-01 23.6%		6.10e+06 22.8%		6.7
157: mfem::CudaCGSolver::h_Mult(mfem::CudaVect	4.33e+13 23.8%		3.25e+00 23.5%		5.44e-01 23.6%		6.10e+06 22.8%		6.7
loop at solvers.cpp: 89	4.07e+13 22.4%		3.07e+00 22.3%		5.14e-01 22.3%		5.73e+06 21.4%		6.3
137: mfem::hydrodynamics::CudaMassOpera	2.27e+13 12.5%	3.75e+09 0.0%	2.26e+00 16.4%		3.87e-01 16.8%		5.73e+06 21.4%		5.7
135: mfem::CudaConstrainedOperator::Mu	2.10e+13 11.5%		1.97e+00 14.3%		3.47e-01 15.0%		5.73e+06 21.4%		5.7
210: mfem::CudaRAPOperator::Mult(mf	2.10e+13 11.5%	7.51e+09 0.0%	1.97e+00 14.3%		3.47e-01 15.0%		5.73e+06 21.4%		5.7
86: mfem::CudaProlongationOperat	1.02e+13 5.6%		6.42e-01 4.7%		1.73e-01 7.5%		2.86e+06 10.7%		2.8

EXASCALE COMPUTING PROJECT

Selecting Metrics to Display Using the Column Selector



```
void CudaProlongationOperator::MultTranspose(const C
                                      CudaVec
{
   if (rconfig::Get().IAmAlone())
   {
      y=x;
      return;
   }
   if (!rconfig::Get().DoHostConformingProlongationO
   {
```

Column Selection

Check columns to be shown and uncheck columns to be hidden

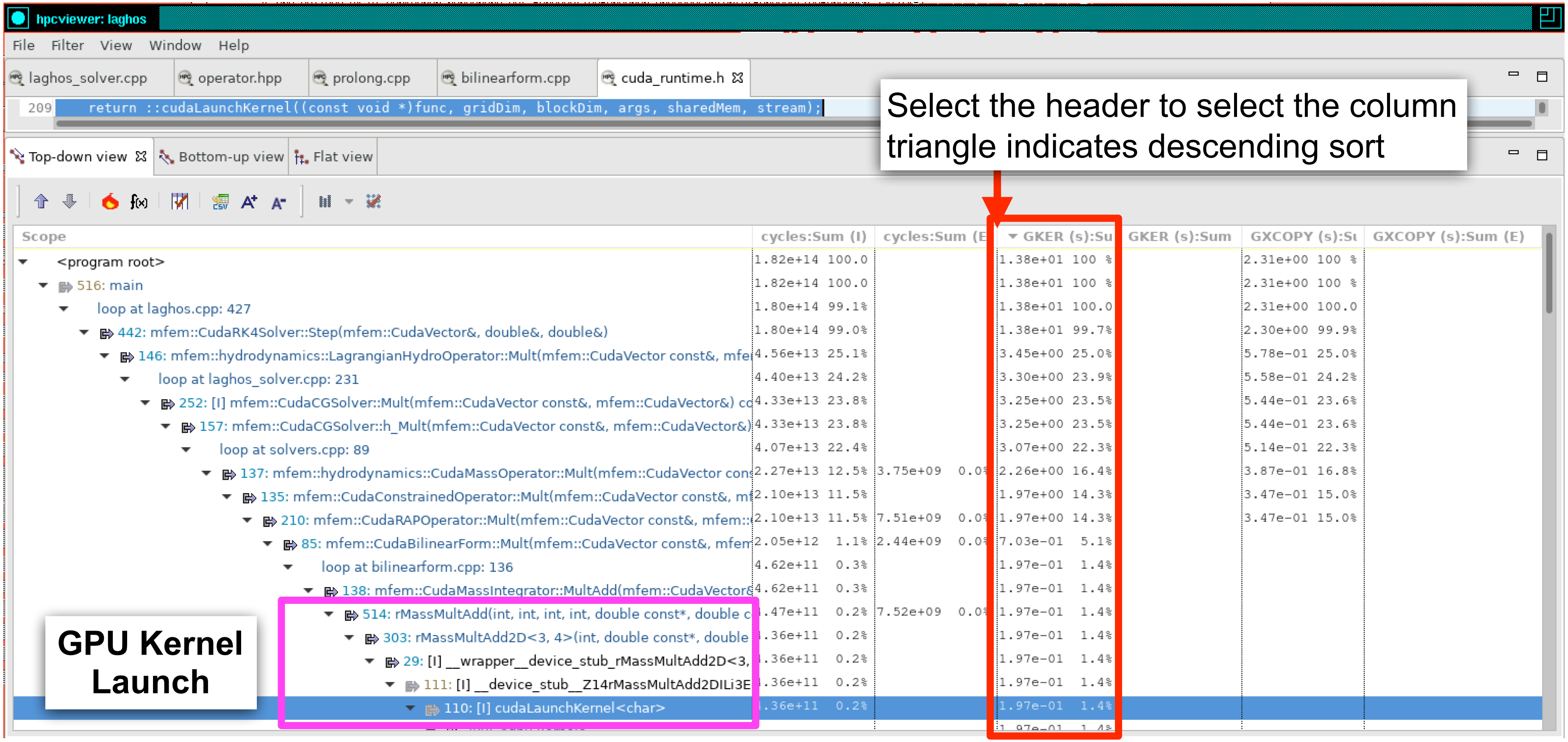
Check all | Uncheck all | ☑ Apply to all views

Filter: (s)

- ☑ GKER (s):Sum (I)
- ☑ GKER (s):Sum (E)
- ☐ GMEM (s):Sum (I) (empty)
- ☐ GMEM (s):Sum (E) (empty)
- ☐ GMSET (s):Sum (I) (empty)
- ☐ GMSET (s):Sum (E) (empty)
- ☑ GXCOPY (s):Sum (I)
- ☑ GXCOPY (s):Sum (E)
- ☐ GICOPY (s):Sum (I) (empty)
- ☐ GICOPY (s):Sum (E) (empty)
- ☐ GSYNC (s):Sum (I) (empty)
- ☐ GSYNC (s):Sum (E) (empty)

Scope	GXCOPY (s):Su	GXCOPY (s):Sum (E)
Experiment Aggregate Metrics	2.31e+00 100 %	2.31e+00 100 %
<program root>	2.31e+00 100 %	
516: main	2.31e+00 100 %	
loop at laghos.cpp: 427	2.31e+00 100.0	
442: mfem::CudaRK4Solver::Step(mfem::CudaVector	2.30e+00 99.9%	
146: mfem::hydrodynamics::LagrangianHydroOpe	5.78e-01 25.0%	
loop at laghos_solver.cpp: 231	5.58e-01 24.2%	
252: [I] mfem::CudaCGSolver::Mult(mfem::(	5.44e-01 23.6%	
157: mfem::CudaCGSolver::h_Mult(mfen	5.44e-01 23.6%	
loop at solvers.cpp: 89	5.14e-01 22.3%	
137: mfem::hydrodynamics::Cuda	3.87e-01 16.8%	
135: mfem::CudaConstrainedO	3.47e-01 15.0%	
210: mfem::CudaRAPOperat	3.47e-01 15.0%	
86: mfem::CudaProlonga	1.73e-01 7.5%	
84: mfem::CudaProlonga	1.74e-01 7.5%	

Using GPU Kernel Time to Guide Top-down Exploration



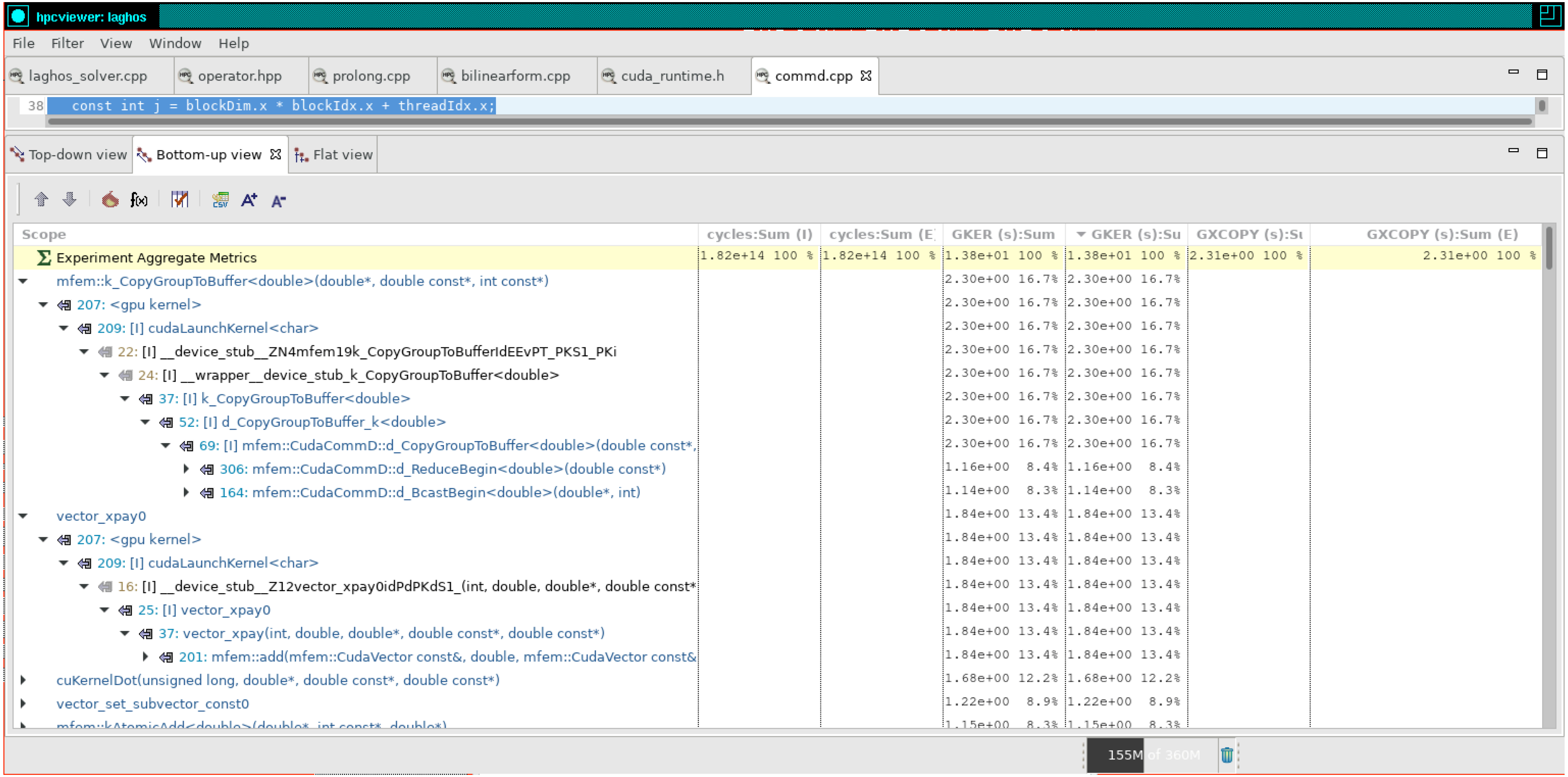
```
209    return ::cudaLaunchKernel((const void *)func, gridDim, blockDim, args, sharedMem, stream);
```

Scope	cycles:Sum (I)	cycles:Sum (E	▼ GKER (s):Su	GKER (s):Sum	GXCOPY (s):Su	GXCOPY (s):Sum (E)
<program root>	1.82e+14 100.0		1.38e+01 100 %		2.31e+00 100 %	
516: main	1.82e+14 100.0		1.38e+01 100 %		2.31e+00 100 %	
loop at laghos.cpp: 427	1.80e+14 99.1%		1.38e+01 100.0		2.31e+00 100.0	
442: mfem::CudaRK4Solver::Step(mfem::CudaVector&, double&, double&)	1.80e+14 99.0%		1.38e+01 99.7%		2.30e+00 99.9%	
146: mfem::hydrodynamics::LagrangianHydroOperator::Mult(mfem::CudaVector const&, mfe	4.56e+13 25.1%		3.45e+00 25.0%		5.78e-01 25.0%	
loop at laghos_solver.cpp: 231	4.40e+13 24.2%		3.30e+00 23.9%		5.58e-01 24.2%	
252: [I] mfem::CudaCGSolver::Mult(mfem::CudaVector const&, mfem::CudaVector&) cc	4.33e+13 23.8%		3.25e+00 23.5%		5.44e-01 23.6%	
157: mfem::CudaCGSolver::h_Mult(mfem::CudaVector const&, mfem::CudaVector&)	4.33e+13 23.8%		3.25e+00 23.5%		5.44e-01 23.6%	
loop at solvers.cpp: 89	4.07e+13 22.4%		3.07e+00 22.3%		5.14e-01 22.3%	
137: mfem::hydrodynamics::CudaMassOperator::Mult(mfem::CudaVector con	2.27e+13 12.5%	3.75e+09 0.0	2.26e+00 16.4%		3.87e-01 16.8%	
135: mfem::CudaConstrainedOperator::Mult(mfem::CudaVector const&, mf	2.10e+13 11.5%		1.97e+00 14.3%		3.47e-01 15.0%	
210: mfem::CudaRAPOperator::Mult(mfem::CudaVector const&, mfem::	2.10e+13 11.5%	7.51e+09 0.0	1.97e+00 14.3%		3.47e-01 15.0%	
85: mfem::CudaBilinearForm::Mult(mfem::CudaVector const&, mfem	2.05e+12 1.1%	2.44e+09 0.0	7.03e-01 5.1%			
loop at bilinearform.cpp: 136	4.62e+11 0.3%		1.97e-01 1.4%			
138: mfem::CudaMassIntegrator::MultAdd(mfem::CudaVector&	4.62e+11 0.3%		1.97e-01 1.4%			
514: rMassMultAdd(int, int, int, int, double const*, double c	.47e+11 0.2%	7.52e+09 0.0	1.97e-01 1.4%			
303: rMassMultAdd2D<3, 4>(int, double const*, double	.36e+11 0.2%		1.97e-01 1.4%			
29: [I] __wrapper__device_stub_rMassMultAdd2D<3,	.36e+11 0.2%		1.97e-01 1.4%			
111: [I] __device_stub__Z14rMassMultAdd2DILi3E	.36e+11 0.2%		1.97e-01 1.4%			
110: [I] cudaLaunchKernel<char>	.36e+11 0.2%		1.97e-01 1.4%			
			1.97e-01 1.4%			

Select the header to select the column
triangle indicates descending sort

GPU Kernel Launch

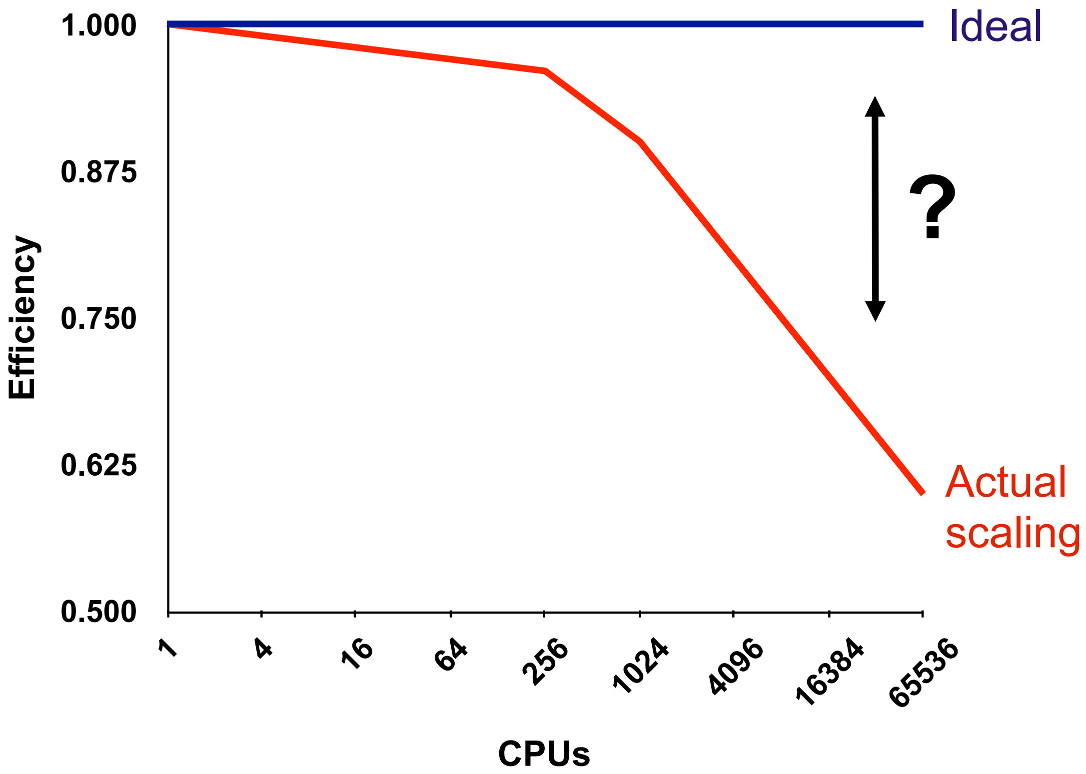
Using GPU Kernel Time to Guide Bottom-up Exploration



```
38   const int j = blockDim.x * blockIdx.x + threadIdx.x;
```

Scope	cycles:Sum (I)	cycles:Sum (E)	GKER (s):Sum	GKER (s):Su	GXCOPY (s):Su	GXCOPY (s):Sum (E)
Σ Experiment Aggregate Metrics	1.82e+14 100 %	1.82e+14 100 %	1.38e+01 100 %	1.38e+01 100 %	2.31e+00 100 %	2.31e+00 100 %
mfem::k_CopyGroupToBuffer<double>(double*, double const*, int const*)			2.30e+00 16.7%	2.30e+00 16.7%		
207: <gpu kernel>			2.30e+00 16.7%	2.30e+00 16.7%		
209: [I] cudaLaunchKernel<char>			2.30e+00 16.7%	2.30e+00 16.7%		
22: [I] __device_stub__ZN4mfem19k_CopyGroupToBufferIdEEvPT_PKS1_PKi			2.30e+00 16.7%	2.30e+00 16.7%		
24: [I] __wrapper__device_stub_k_CopyGroupToBuffer<double>			2.30e+00 16.7%	2.30e+00 16.7%		
37: [I] k_CopyGroupToBuffer<double>			2.30e+00 16.7%	2.30e+00 16.7%		
52: [I] d_CopyGroupToBuffer_k<double>			2.30e+00 16.7%	2.30e+00 16.7%		
69: [I] mfem::CudaCommD::d_CopyGroupToBuffer<double>(double const*,			2.30e+00 16.7%	2.30e+00 16.7%		
306: mfem::CudaCommD::d_ReduceBegin<double>(double const*)			1.16e+00 8.4%	1.16e+00 8.4%		
164: mfem::CudaCommD::d_BcastBegin<double>(double*, int)			1.14e+00 8.3%	1.14e+00 8.3%		
vector_xpay0			1.84e+00 13.4%	1.84e+00 13.4%		
207: <gpu kernel>			1.84e+00 13.4%	1.84e+00 13.4%		
209: [I] cudaLaunchKernel<char>			1.84e+00 13.4%	1.84e+00 13.4%		
16: [I] __device_stub__Z12vector_xpay0idPdPKdS1_(int, double, double*, double const*			1.84e+00 13.4%	1.84e+00 13.4%		
25: [I] vector_xpay0			1.84e+00 13.4%	1.84e+00 13.4%		
37: vector_xpay(int, double, double*, double const*, double const*)			1.84e+00 13.4%	1.84e+00 13.4%		
201: mfem::add(mfem::CudaVector const&, double, mfem::CudaVector const&			1.84e+00 13.4%	1.84e+00 13.4%		
cuKernelDot(unsigned long, double*, double const*, double const*)			1.68e+00 12.2%	1.68e+00 12.2%		
vector_set_subvector_const0			1.22e+00 8.9%	1.22e+00 8.9%		
mfem::kAtomicAdd<double>(double*, int const*, double*)			1.15e+00 8.3%	1.15e+00 8.3%		

The Problem of Scaling



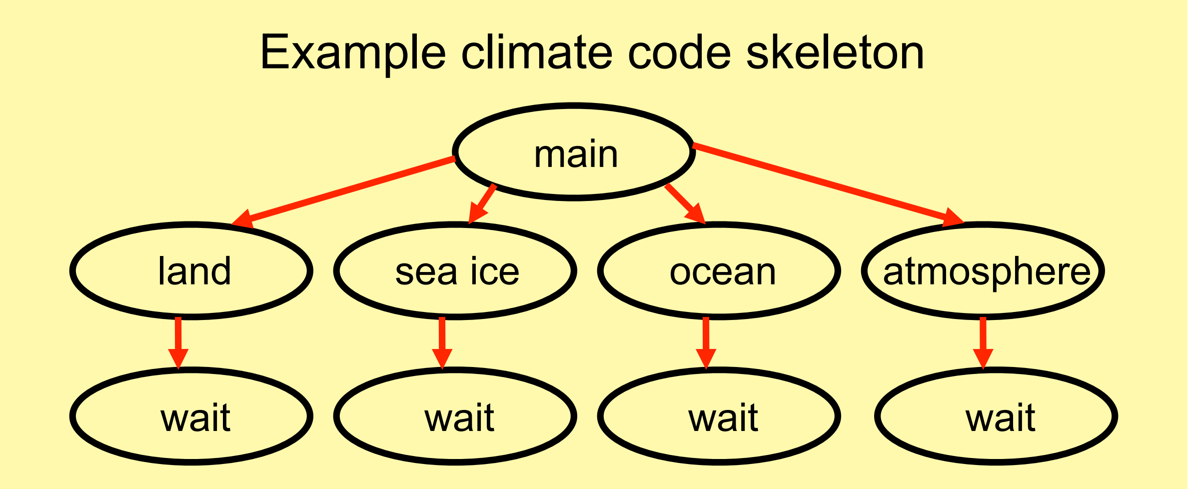
Note: higher is better

Wanted: Scalability Analysis

- Isolate scalability bottlenecks
- Guide user to problems
- Quantify the magnitude of each problem

Challenges for Pinpointing Scalability Bottlenecks

- **Parallel applications**
  - — **modern software uses layers of libraries**
  - — **performance is often context dependent**



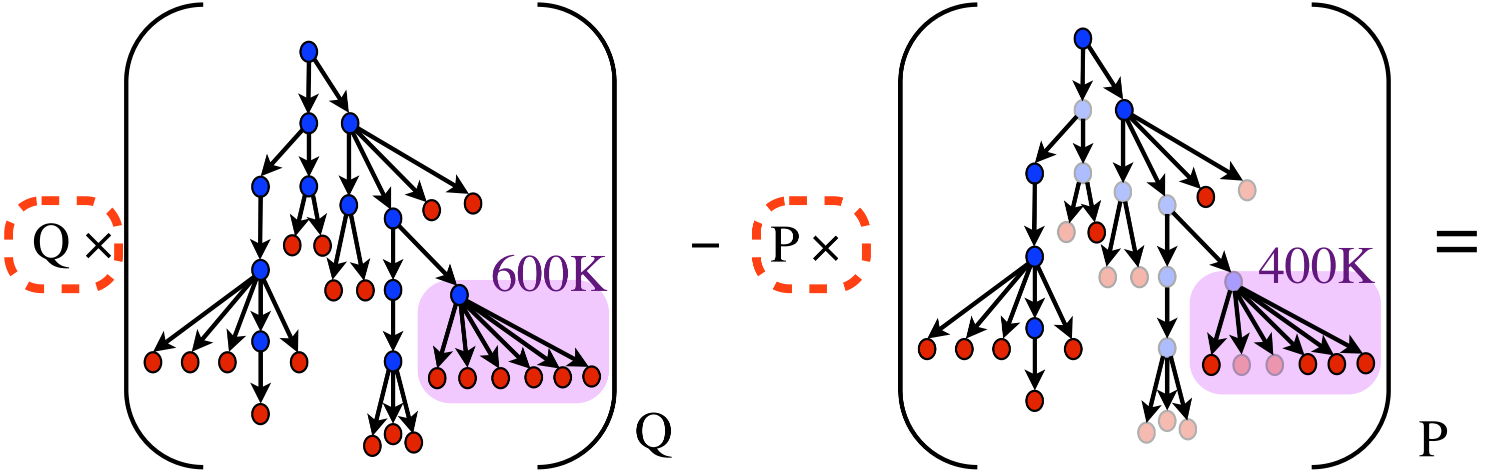
- **Monitoring**
  - — **bottleneck nature: computation, data movement, synchronization?**
  - — **2 pragmatic constraints**
    - – **acceptable data volume**
    - – **low perturbation for use in production runs**

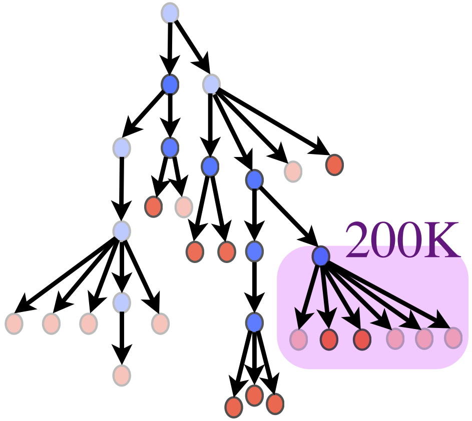
Performance Analysis with Expectations

- You have performance expectations for your parallel code
  - strong scaling: linear speedup
  - weak scaling: constant execution time

- Put your expectations to work
  - measure performance under different conditions
    - e.g. different levels of parallelism and/or different problem size
  - express your expectations as an equation
  - compute the deviation from expectations for each calling context
    - for both inclusive and exclusive costs
  - correlate the metrics with the source code
  - explore the annotated call tree interactively

Pinpointing and Quantifying Scalability Bottlenecks

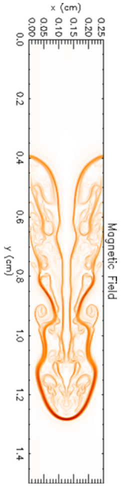




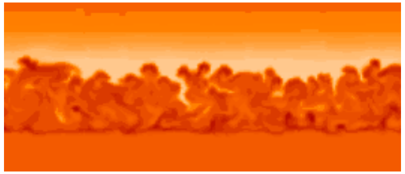
coefficients for analysis
of strong scaling

Scalability Analysis Demo: FLASH3

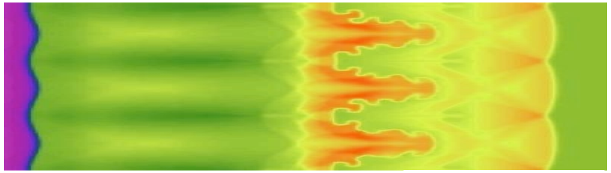
Code: University of Chicago FLASH3
Simulation: white dwarf detonation
Platform: Blue Gene/P
Experiment: 8192 vs. 256 cores
Scaling type: weak



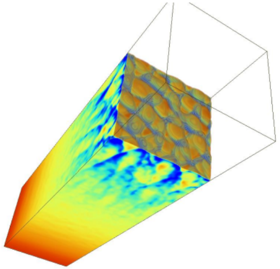
Magnetic Rayleigh-Taylor



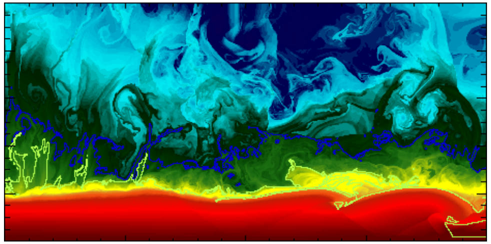
Nova outbursts on white dwarfs



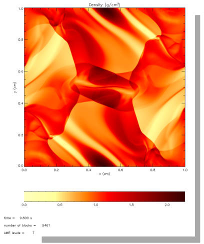
Laser-driven shock instabilities



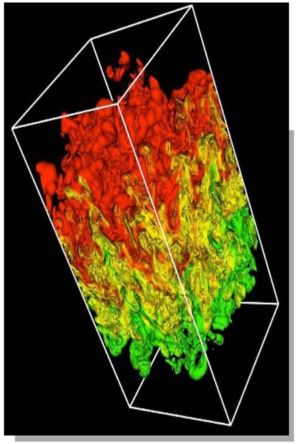
Cellular detonation



Helium burning on neutron stars



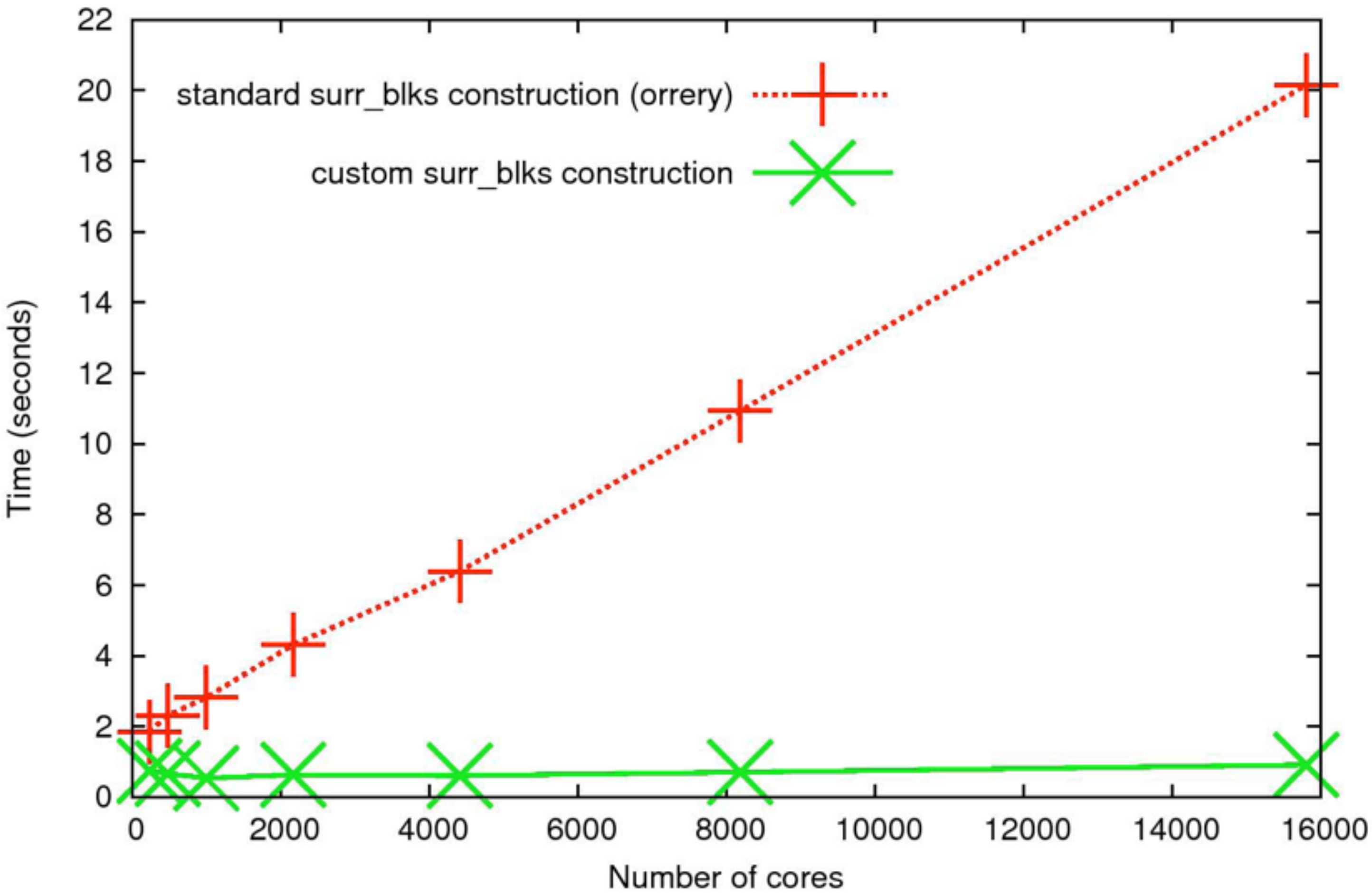
Orzag/Tang MHD vortex



Rayleigh-Taylor instability

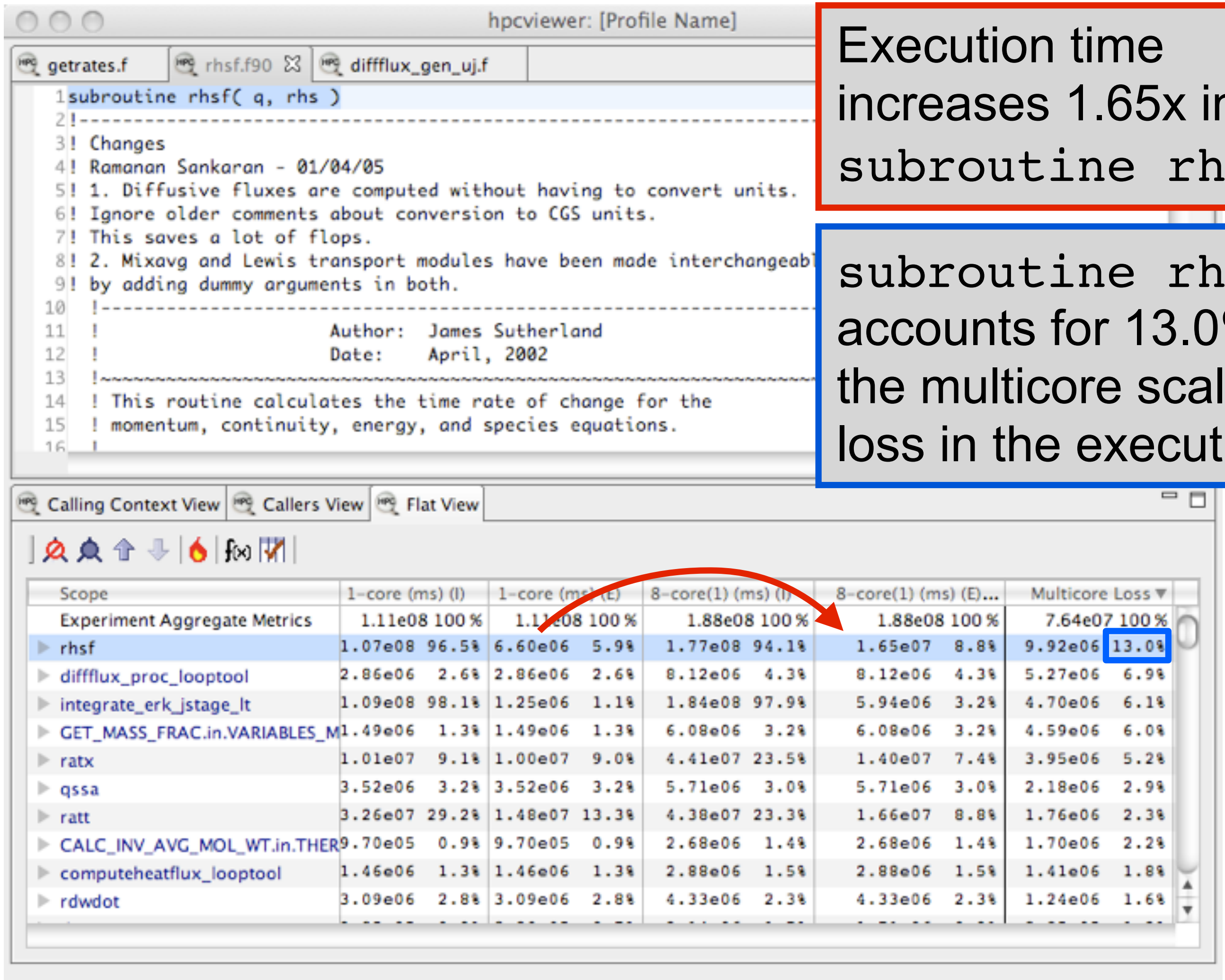
Figures courtesy of FLASH Team, University of Chicago

Improved Flash Scaling of AMR Setup



Graph courtesy of Anshu Dubey, U Chicago

S3D: Multicore Losses at the Procedure Level



```
subroutine rhsf( q, rhs )
!-----------------------------------------------------------------------
! Changes
! Ramanan Sankaran - 01/04/05
! 1. Diffusive fluxes are computed without having to convert units.
! Ignore older comments about conversion to CGS units.
! This saves a lot of flops.
! 2. Mixavg and Lewis transport modules have been made interchangeabl
! by adding dummy arguments in both.
  !---------------------------------------------------------------------
  !                     Author:  James Sutherland
  !                     Date:    April, 2002
  !~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  ! This routine calculates the time rate of change for the
  ! momentum, continuity, energy, and species equations.
  !
```

Scope	1-core (ms) (I)	1-core (ms) (E)	8-core(1) (ms) (I)	8-core(1) (ms) (E)...	Multicore Loss ▼
Experiment Aggregate Metrics	1.11e08 100 %	1.11e08 100 %	1.88e08 100 %	1.88e08 100 %	7.64e07 100 %
rhsf	1.07e08 96.5%	6.60e06 5.9%	1.77e08 94.1%	1.65e07 8.8%	9.92e06 13.0%
diffflux_proc_looptool	2.86e06 2.6%	2.86e06 2.6%	8.12e06 4.3%	8.12e06 4.3%	5.27e06 6.9%
integrate_erk_jstage_lt	1.09e08 98.1%	1.25e06 1.1%	1.84e08 97.9%	5.94e06 3.2%	4.70e06 6.1%
GET_MASS_FRAC.in.VARIABLES_M	1.49e06 1.3%	1.49e06 1.3%	6.08e06 3.2%	6.08e06 3.2%	4.59e06 6.0%
ratx	1.01e07 9.1%	1.00e07 9.0%	4.41e07 23.5%	1.40e07 7.4%	3.95e06 5.2%
qssa	3.52e06 3.2%	3.52e06 3.2%	5.71e06 3.0%	5.71e06 3.0%	2.18e06 2.9%
ratt	3.26e07 29.2%	1.48e07 13.3%	4.38e07 23.3%	1.66e07 8.8%	1.76e06 2.3%
CALC_INV_AVG_MOL_WT.in.THER	9.70e05 0.9%	9.70e05 0.9%	2.68e06 1.4%	2.68e06 1.4%	1.70e06 2.2%
computeheatflux_looptool	1.46e06 1.3%	1.46e06 1.3%	2.88e06 1.5%	2.88e06 1.5%	1.41e06 1.8%
rdwdot	3.09e06 2.8%	3.09e06 2.8%	4.33e06 2.3%	4.33e06 2.3%	1.24e06 1.6%

Execution time increases 1.65x in `subroutine rhsf`

`subroutine rhsf` accounts for 13.0% of the multicore scaling loss in the execution

Applying the GPU PC Sampling Measurement Workflow to Laghos

```
# measure an execution of laghos using pc sampling
time mpirun -np 4 hpcrun -o $OUT -e cycles -e gpu=nvidia,pc -t \
   ${LAGHOS_DIR}/laghos -p 0 -m ${LAGHOS_DIR}/../data/square01_quad.mesh \
   -rs 1 -tf 0.05 -pa

# compute program structure information for the laghos binary
hpcstruct -j 16 laghos

# compute program structure information for the laghos cubins with CFG
hpcstruct —-gpucfg yes —j 16 $OUT

# combine the measurements with the program structure information
mpirun -n 4  hpcprof-mpi -S laghos.hpcstruct $OUT
```

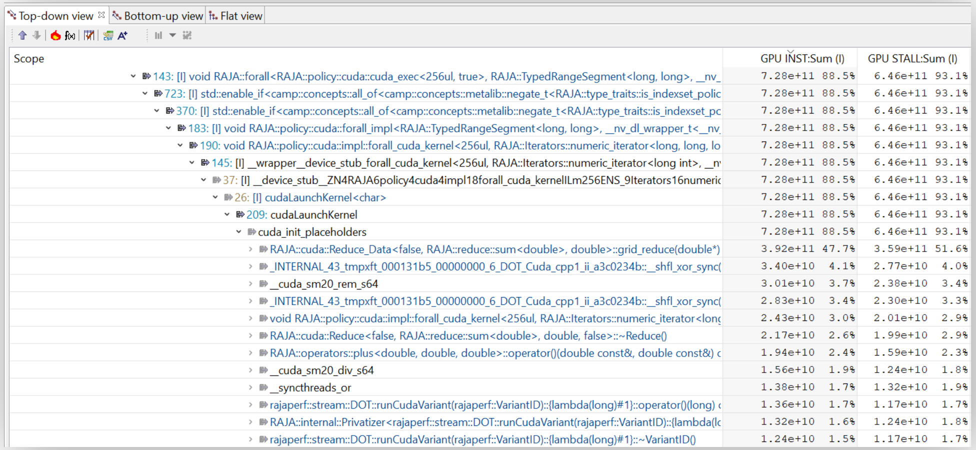
HPCToolkit’s GPU Instruction Sampling Metrics (NVIDIA Only)

Metric	Definition
GINST:STL_ANY	GPU instruction stalls: any (sum of all **STALL** metrics other than **NONE**)
GINST:STL_NONE	GPU instruction stalls: no stall
GINST:STL_IFET	GPU instruction stalls: await availability of next instruction (fetch or branch delay)
GINST:STL_IDEP	GPU instruction stalls: await satisfaction of instruction input dependence
GINST:STL_GMEM	GPU instruction stalls: await completion of global memory access
GINST:STL_TMEM	GPU instruction stalls: texture memory request queue full
GINST:STL_SYNC	GPU instruction stalls: await completion of thread or memory synchronization
GINST:STL_CMEM	GPU instruction stalls: await completion of constant or immediate memory access
GINST:STL_PIPE	GPU instruction stalls: await completion of required compute resources
GINST:STL_MTHR	GPU instruction stalls: global memory request queue full
GINST:STL_NSEL	GPU instruction stalls: not selected for issue but ready
GINST:STL_OTHR	GPU instruction stalls: other
GINST:STL_SLP	GPU instruction stalls: sleep

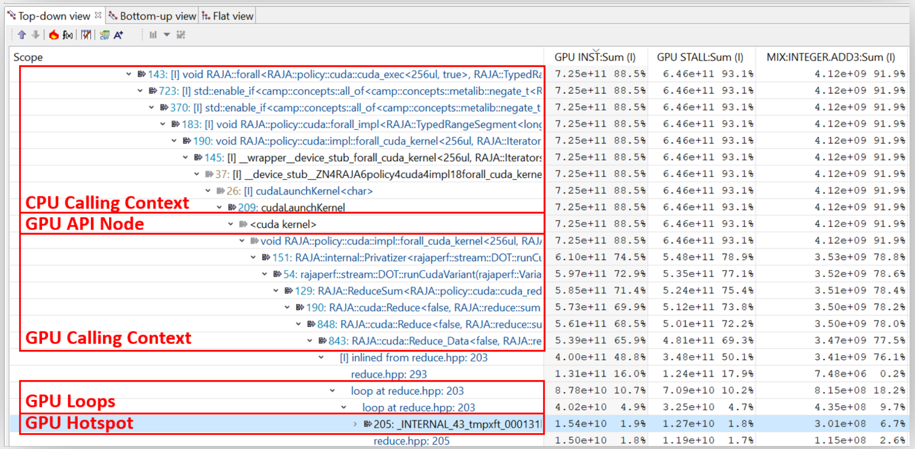
Approximation of GPU Calling Contexts to Understand Performance

- GPU code from C++ template-based programming models is complex
- NVIDIA GPUs collect flat PC samples
- Flat profiles for instantiations of complex C++ templates are inscrutable

- HPCToolkit reconstructs approximate GPU calling contexts
  - Reconstruct call graph from machine code
  - Infer calls at call sites
    - PC samples of call instructions indicate calls
      - Use call counts to apportion costs to call sites
    - PC samples in a routine

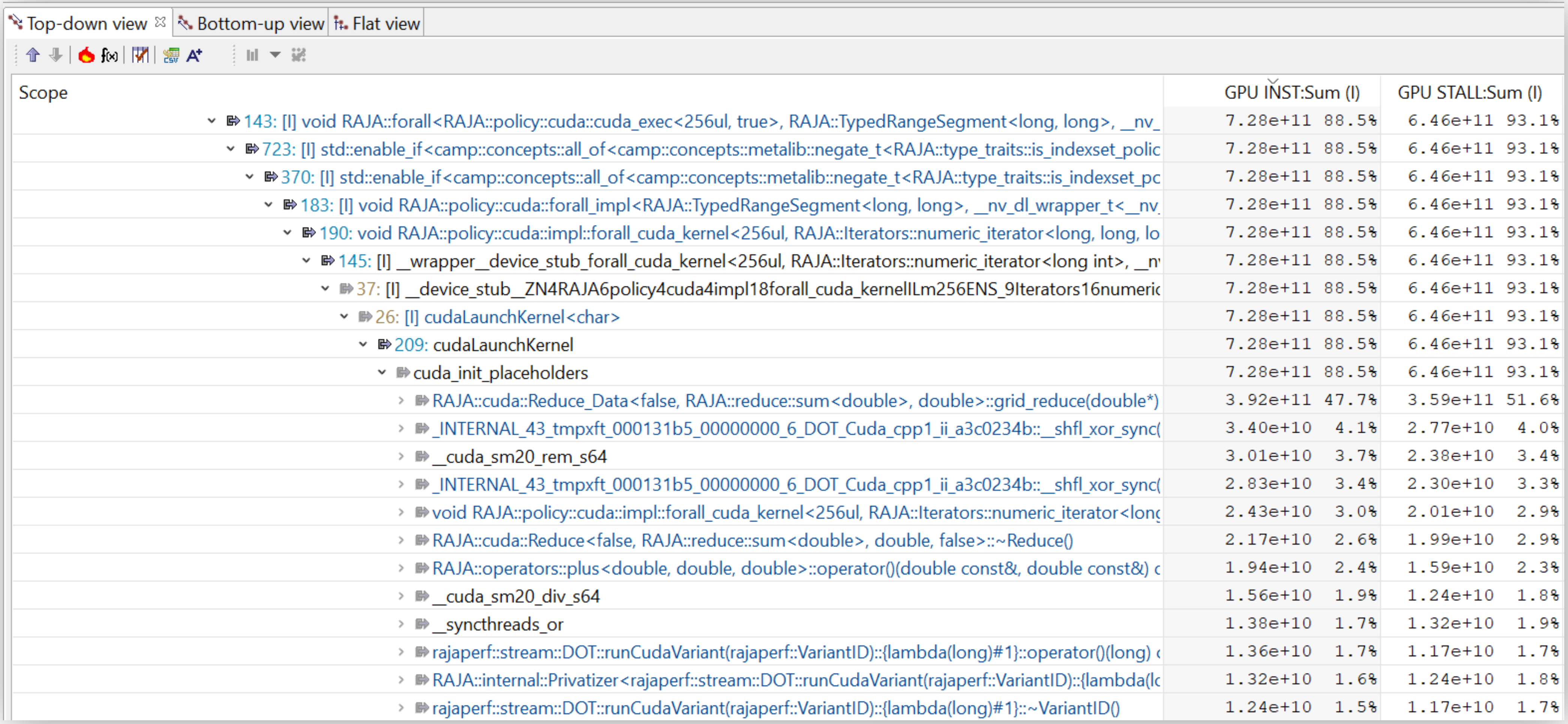


Scope	GPU INST:Sum (I)		GPU STALL:Sum (I)	
143: [I] void RAJA::forall<RAJA::policy::cuda::cuda_exec<256ul, true>, RAJA::TypedRangeSegment<long, long>, __nv_	7.28e+11	88.5%	6.46e+11	93.1%
723: [I] std::enable_if<camp::concepts::all_of<camp::concepts::metalib::negate_t<RAJA::type_traits::is_indexset_polic	7.28e+11	88.5%	6.46e+11	93.1%
370: [I] std::enable_if<camp::concepts::all_of<camp::concepts::metalib::negate_t<RAJA::type_traits::is_indexset_pc	7.28e+11	88.5%	6.46e+11	93.1%
183: [I] void RAJA::policy::cuda::forall_impl<RAJA::TypedRangeSegment<long, long>, __nv_dl_wrapper_t<__nv	7.28e+11	88.5%	6.46e+11	93.1%
190: void RAJA::policy::cuda::impl::forall_cuda_kernel<256ul, RAJA::Iterators::numeric_iterator<long, long, lo	7.28e+11	88.5%	6.46e+11	93.1%
145: [I] __wrapper__device_stub_forall_cuda_kernel<256ul, RAJA::Iterators::numeric_iterator<long int>, __n	7.28e+11	88.5%	6.46e+11	93.1%
37: [I] __device_stub__ZN4RAJA6policy4cuda4impl18forall_cuda_kernelILm256ENS_9Iterators16numeric	7.28e+11	88.5%	6.46e+11	93.1%
26: [I] cudaLaunchKernel<char>	7.28e+11	88.5%	6.46e+11	93.1%
209: cudaLaunchKernel	7.28e+11	88.5%	6.46e+11	93.1%
cuda_init_placeholders	7.28e+11	88.5%	6.46e+11	93.1%
RAJA::cuda::Reduce_Data<false, RAJA::reduce::sum<double>, double>::grid_reduce(double*)	3.92e+11	47.7%	3.59e+11	51.6%
_INTERNAL_43_tmpxft_000131b5_00000000_6_DOT_Cuda_cpp1_ii_a3c0234b::__shfl_xor_sync(	3.40e+10	4.1%	2.77e+10	4.0%
__cuda_sm20_rem_s64	3.01e+10	3.7%	2.38e+10	3.4%
_INTERNAL_43_tmpxft_000131b5_00000000_6_DOT_Cuda_cpp1_ii_a3c0234b::__shfl_xor_sync(	2.83e+10	3.4%	2.30e+10	3.3%
void RAJA::policy::cuda::impl::forall_cuda_kernel<256ul, RAJA::Iterators::numeric_iterator<lonc	2.43e+10	3.0%	2.01e+10	2.9%
RAJA::cuda::Reduce<false, RAJA::reduce::sum<double>, double, false>::~Reduce()	2.17e+10	2.6%	1.99e+10	2.9%
RAJA::operators::plus<double, double, double>::operator()(double const&, double const&) c	1.94e+10	2.4%	1.59e+10	2.3%
__cuda_sm20_div_s64	1.56e+10	1.9%	1.24e+10	1.8%
__syncthreads_or	1.38e+10	1.7%	1.32e+10	1.9%
rajaperf::stream::DOT::runCudaVariant(rajaperf::VariantID)::{lambda(long)#1}::operator()(long) c	1.36e+10	1.7%	1.17e+10	1.7%
RAJA::internal::Privatizer<rajaperf::stream::DOT::runCudaVariant(rajaperf::VariantID)::{lambda(lc	1.32e+10	1.6%	1.24e+10	1.8%
rajaperf::stream::DOT::runCudaVariant(rajaperf::VariantID)::{lambda(long)#1}::~VariantID()	1.24e+10	1.5%	1.17e+10	1.7%



Scope	GPU INST:Sum (I)		GPU STALL:Sum (I)		MIX:INTEGER.ADD3:Sum (I)	
143: [I] void RAJA::forall<RAJA::policy::cuda::cuda_exec<256ul, true>, RAJA::TypedRa	7.25e+11	88.5%	6.46e+11	93.1%	4.12e+09	91.9%
723: [I] std::enable_if<camp::concepts::all_of<camp::concepts::metalib::negate_t<R	7.25e+11	88.5%	6.46e+11	93.1%	4.12e+09	91.9%
370: [I] std::enable_if<camp::concepts::all_of<camp::concepts::metalib::negate_t	7.25e+11	88.5%	6.46e+11	93.1%	4.12e+09	91.9%
183: [I] void RAJA::policy::cuda::forall_impl<RAJA::TypedRangeSegment<lonc	7.25e+11	88.5%	6.46e+11	93.1%	4.12e+09	91.9%
190: void RAJA::policy::cuda::impl::forall_cuda_kernel<256ul, RAJA::Iterator	7.25e+11	88.5%	6.46e+11	93.1%	4.12e+09	91.9%
145: [I] __wrapper__device_stub_forall_cuda_kernel<256ul, RAJA::Iterators	7.25e+11	88.5%	6.46e+11	93.1%	4.12e+09	91.9%
37: [I] __device_stub__ZN4RAJA6policy4cuda4impl18forall_cuda_kerne	7.25e+11	88.5%	6.46e+11	93.1%	4.12e+09	91.9%
26: [I] cudaLaunchKernel<char>	7.25e+11	88.5%	6.46e+11	93.1%	4.12e+09	91.9%
209: cudaLaunchKernel	7.25e+11	88.5%	6.46e+11	93.1%	4.12e+09	91.9%
<cuda kernel>	7.25e+11	88.5%	6.46e+11	93.1%	4.12e+09	91.9%
void RAJA::policy::cuda::impl::forall_cuda_kernel<256ul, RAJ/	7.25e+11	88.5%	6.46e+11	93.1%	4.12e+09	91.9%
151: RAJA::internal::Privatizer<rajaperf::stream::DOT::runCu	6.10e+11	74.5%	5.48e+11	78.9%	3.53e+09	78.8%
54: rajaperf::stream::DOT::runCudaVariant(rajaperf::Varia	5.97e+11	72.9%	5.35e+11	77.1%	3.52e+09	78.6%
129: RAJA::ReduceSum<RAJA::policy::cuda::cuda_red	5.85e+11	71.4%	5.24e+11	75.4%	3.51e+09	78.4%
190: RAJA::cuda::Reduce<false, RAJA::reduce::sum	5.73e+11	69.9%	5.12e+11	73.8%	3.50e+09	78.2%
848: RAJA::cuda::Reduce<false, RAJA::reduce::su	5.61e+11	68.5%	5.01e+11	72.2%	3.50e+09	78.0%
843: RAJA::cuda::Reduce_Data<false, RAJA::re	5.39e+11	65.9%	4.81e+11	69.3%	3.47e+09	77.5%
[I] inlined from reduce.hpp: 203	4.00e+11	48.8%	3.48e+11	50.1%	3.41e+09	76.1%
reduce.hpp: 293	1.31e+11	16.0%	1.24e+11	17.9%	7.48e+06	0.2%
loop at reduce.hpp: 203	8.78e+10	10.7%	7.09e+10	10.2%	8.15e+08	18.2%
loop at reduce.hpp: 203	4.02e+10	4.9%	3.25e+10	4.7%	4.35e+08	9.7%
205: _INTERNAL_43_tmpxft_000131	1.54e+10	1.9%	1.27e+10	1.8%	3.01e+08	6.7%
reduce.hpp: 205	1.50e+10	1.8%	1.19e+10	1.7%	1.15e+08	2.6%

Approximation of GPU Calling Contexts to Understand Performance

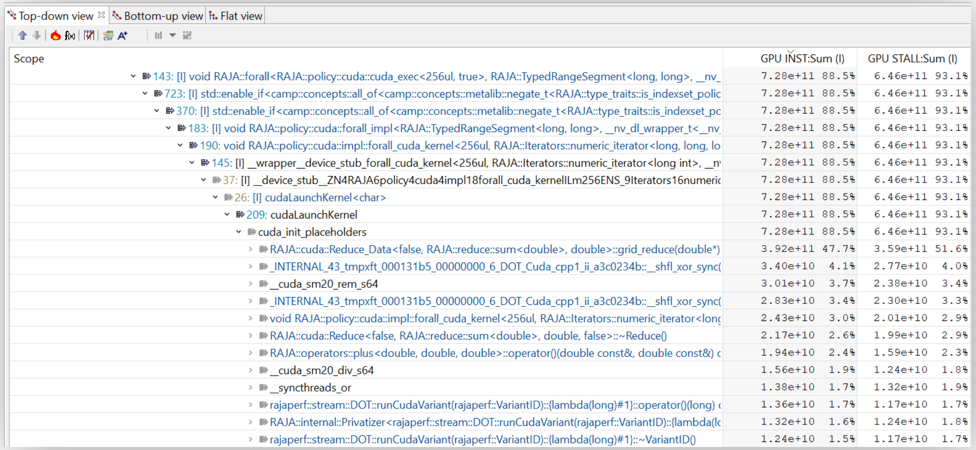


Scope	GPU INST:Sum (I)		GPU STALL:Sum (I)	
143: [I] void RAJA::forall<RAJA::policy::cuda::cuda_exec<256ul, true>, RAJA::TypedRangeSegment<long, long>, __nv_	7.28e+11	88.5%	6.46e+11	93.1%
723: [I] std::enable_if<camp::concepts::all_of<camp::concepts::metalib::negate_t<RAJA::type_traits::is_indexset_polic	7.28e+11	88.5%	6.46e+11	93.1%
370: [I] std::enable_if<camp::concepts::all_of<camp::concepts::metalib::negate_t<RAJA::type_traits::is_indexset_pc	7.28e+11	88.5%	6.46e+11	93.1%
183: [I] void RAJA::policy::cuda::forall_impl<RAJA::TypedRangeSegment<long, long>, __nv_dl_wrapper_t<__nv_	7.28e+11	88.5%	6.46e+11	93.1%
190: void RAJA::policy::cuda::impl::forall_cuda_kernel<256ul, RAJA::Iterators::numeric_iterator<long, long, lo	7.28e+11	88.5%	6.46e+11	93.1%
145: [I] __wrapper__device_stub_forall_cuda_kernel<256ul, RAJA::Iterators::numeric_iterator<long int>, __n	7.28e+11	88.5%	6.46e+11	93.1%
37: [I] __device_stub__ZN4RAJA6policy4cuda4impl18forall_cuda_kernelILm256ENS_9Iterators16numeric	7.28e+11	88.5%	6.46e+11	93.1%
26: [I] cudaLaunchKernel<char>	7.28e+11	88.5%	6.46e+11	93.1%
209: cudaLaunchKernel	7.28e+11	88.5%	6.46e+11	93.1%
cuda_init_placeholders	7.28e+11	88.5%	6.46e+11	93.1%
RAJA::cuda::Reduce_Data<false, RAJA::reduce::sum<double>, double>::grid_reduce(double*)	3.92e+11	47.7%	3.59e+11	51.6%
_INTERNAL_43_tmpxft_000131b5_00000000_6_DOT_Cuda_cpp1_ii_a3c0234b::__shfl_xor_sync(	3.40e+10	4.1%	2.77e+10	4.0%
__cuda_sm20_rem_s64	3.01e+10	3.7%	2.38e+10	3.4%
_INTERNAL_43_tmpxft_000131b5_00000000_6_DOT_Cuda_cpp1_ii_a3c0234b::__shfl_xor_sync(	2.83e+10	3.4%	2.30e+10	3.3%
void RAJA::policy::cuda::impl::forall_cuda_kernel<256ul, RAJA::Iterators::numeric_iterator<lonc	2.43e+10	3.0%	2.01e+10	2.9%
RAJA::cuda::Reduce<false, RAJA::reduce::sum<double>, double, false>::~Reduce()	2.17e+10	2.6%	1.99e+10	2.9%
RAJA::operators::plus<double, double, double>::operator()(double const&, double const&) c	1.94e+10	2.4%	1.59e+10	2.3%
__cuda_sm20_div_s64	1.56e+10	1.9%	1.24e+10	1.8%
__syncthreads_or	1.38e+10	1.7%	1.32e+10	1.9%
rajaperf::stream::DOT::runCudaVariant(rajaperf::VariantID)::{lambda(long)#1}::operator()(long) c	1.36e+10	1.7%	1.17e+10	1.7%
RAJA::internal::Privatizer<rajaperf::stream::DOT::runCudaVariant(rajaperf::VariantID)::{lambda(lc	1.32e+10	1.6%	1.24e+10	1.8%
rajaperf::stream::DOT::runCudaVariant(rajaperf::VariantID)::{lambda(long)#1}::~VariantID()	1.24e+10	1.5%	1.17e+10	1.7%

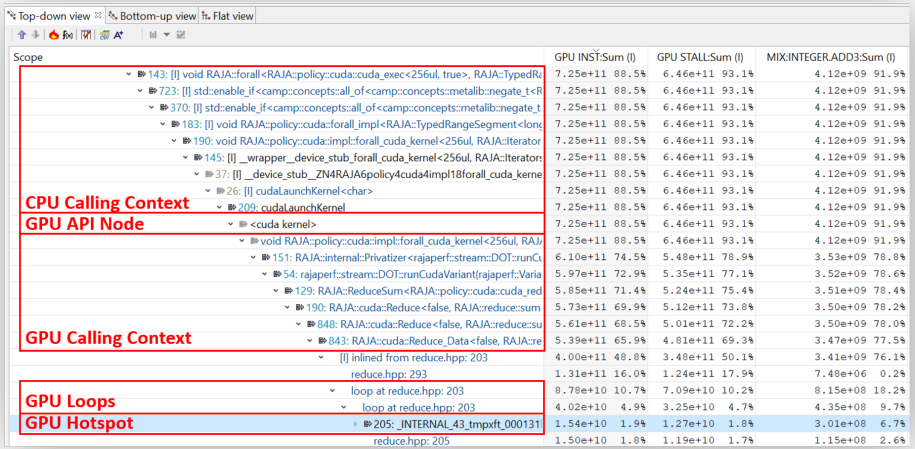
Approximation of GPU Calling Contexts to Understand Performance

- GPU code from C++ template-based programming models is complex
- NVIDIA GPUs collect flat PC samples
- Flat profiles for instantiations of complex C++ templates are inscrutable

- HPCToolkit reconstructs approximate GPU calling contexts
  - Reconstruct call graph from machine code
  - Infer calls at call sites
    - PC samples of call instructions indicate calls
      - Use call counts to apportion costs to call sites
    - PC samples in a routine

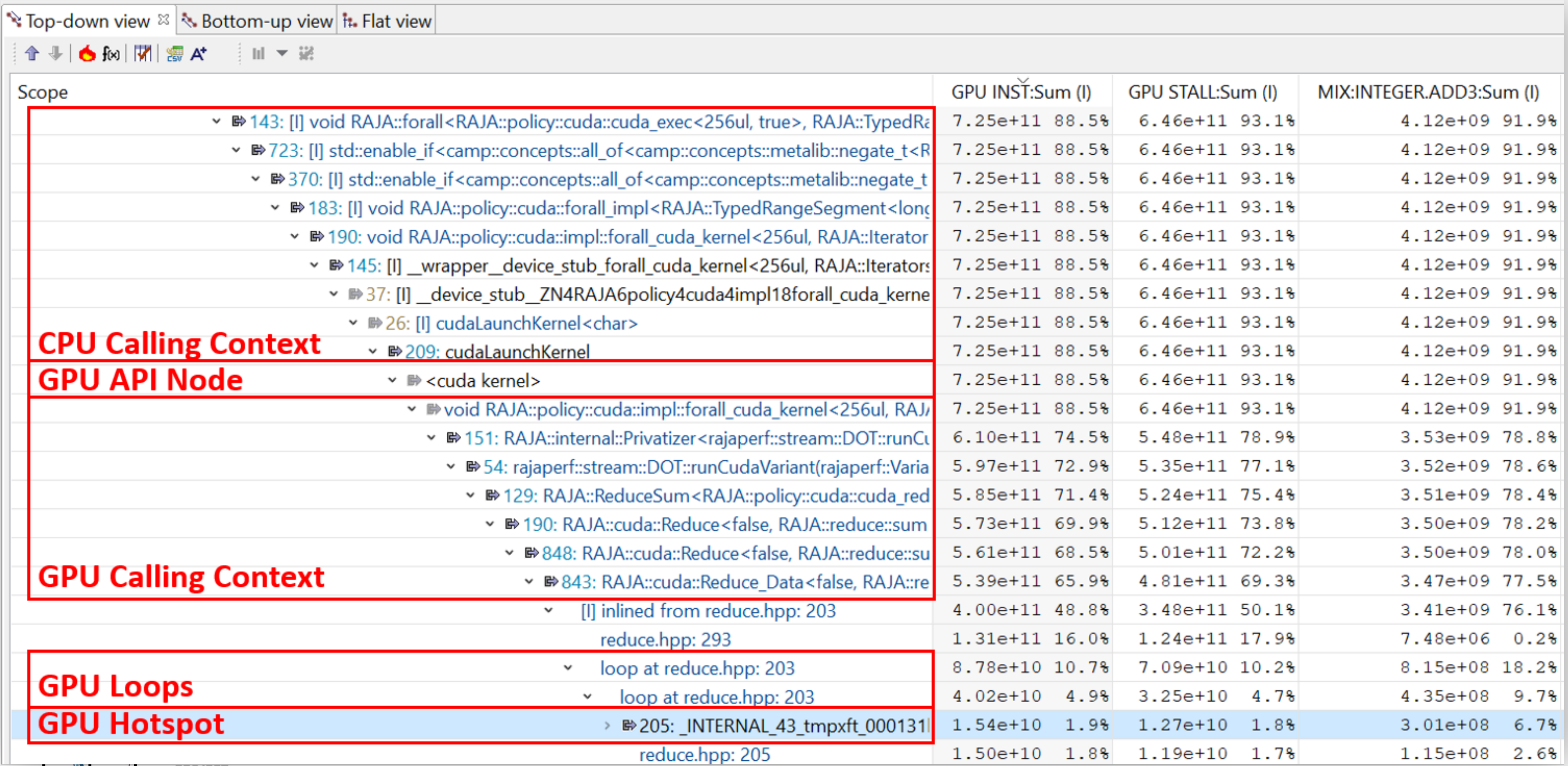


Scope	GPU INST:Sum (I)	GPU STALL:Sum (I)
143: [I] void RAJA::forall<RAJA::policy::cuda::cuda_exec<256ul, true>, RAJA::TypedRangeSegment<long, long>, __nv_	7.28e+11 88.5%	6.46e+11 93.1%
723: [I] std::enable_if<camp::concepts::all_of<camp::concepts::metalib::negate_t<RAJA::type_traits::is_indexset_polic	7.28e+11 88.5%	6.46e+11 93.1%
370: [I] std::enable_if<camp::concepts::all_of<camp::concepts::metalib::negate_t<RAJA::type_traits::is_indexset_pc	7.28e+11 88.5%	6.46e+11 93.1%
183: [I] void RAJA::policy::cuda::forall_impl<RAJA::TypedRangeSegment<long, long>, __nv_dl_wrapper_t<__nv_	7.28e+11 88.5%	6.46e+11 93.1%
190: void RAJA::policy::cuda::impl::forall_cuda_kernel<256ul, RAJA::Iterators::numeric_iterator<long, long, lo	7.28e+11 88.5%	6.46e+11 93.1%
145: [I] __wrapper__device_stub_forall_cuda_kernel<256ul, RAJA::Iterators::numeric_iterator<long int>, __n	7.28e+11 88.5%	6.46e+11 93.1%
37: [I] __device_stub__ZN4RAJA6policy4cuda4impl18forall_cuda_kernelILm256ENS_9Iterators16numeric	7.28e+11 88.5%	6.46e+11 93.1%
26: [I] cudaLaunchKernel<char>	7.28e+11 88.5%	6.46e+11 93.1%
209: cudaLaunchKernel	7.28e+11 88.5%	6.46e+11 93.1%
cuda_init_placeholders	7.28e+11 88.5%	6.46e+11 93.1%
RAJA::cuda::Reduce_Data<false, RAJA::reduce::sum<double>, double>::grid_reduce(double*)	3.92e+11 47.7%	3.59e+11 51.6%
_INTERNAL_43_tmpxft_000131b5_00000000_6_DOT_Cuda_cpp1_ii_a3c0234b::__shfl_xor_sync(	3.40e+10 4.1%	2.77e+10 4.0%
__cuda_sm20_rem_s64	3.01e+10 3.7%	2.38e+10 3.4%
_INTERNAL_43_tmpxft_000131b5_00000000_6_DOT_Cuda_cpp1_ii_a3c0234b::__shfl_xor_sync(	2.83e+10 3.4%	2.30e+10 3.3%
void RAJA::policy::cuda::impl::forall_cuda_kernel<256ul, RAJA::Iterators::numeric_iterator<lonc	2.43e+10 3.0%	2.01e+10 2.9%
RAJA::cuda::Reduce<false, RAJA::reduce::sum<double>, double, false>::~Reduce()	2.17e+10 2.6%	1.99e+10 2.9%
RAJA::operators::plus<double, double, double>::operator()(double const&, double const&) c	1.94e+10 2.4%	1.59e+10 2.3%
__cuda_sm20_div_s64	1.56e+10 1.9%	1.24e+10 1.8%
__syncthreads_or	1.38e+10 1.7%	1.32e+10 1.9%
rajaperf::stream::DOT::runCudaVariant(rajaperf::VariantID)::{lambda(long)#1}::operator()(long) c	1.36e+10 1.7%	1.17e+10 1.7%
RAJA::internal::Privatizer<rajaperf::stream::DOT::runCudaVariant(rajaperf::VariantID)::{lambda(lc	1.32e+10 1.6%	1.24e+10 1.8%
rajaperf::stream::DOT::runCudaVariant(rajaperf::VariantID)::{lambda(long)#1}::~VariantID()	1.24e+10 1.5%	1.17e+10 1.7%



Scope	GPU INST:Sum (I)	GPU STALL:Sum (I)	MIX:INTEGER.ADD3:Sum (I)
143: [I] void RAJA::forall<RAJA::policy::cuda::cuda_exec<256ul, true>, RAJA::TypedRa	7.25e+11 88.5%	6.46e+11 93.1%	4.12e+09 91.9%
723: [I] std::enable_if<camp::concepts::all_of<camp::concepts::metalib::negate_t<R	7.25e+11 88.5%	6.46e+11 93.1%	4.12e+09 91.9%
370: [I] std::enable_if<camp::concepts::all_of<camp::concepts::metalib::negate_t	7.25e+11 88.5%	6.46e+11 93.1%	4.12e+09 91.9%
183: [I] void RAJA::policy::cuda::forall_impl<RAJA::TypedRangeSegment<lonc	7.25e+11 88.5%	6.46e+11 93.1%	4.12e+09 91.9%
190: void RAJA::policy::cuda::impl::forall_cuda_kernel<256ul, RAJA::Iterator	7.25e+11 88.5%	6.46e+11 93.1%	4.12e+09 91.9%
145: [I] __wrapper__device_stub_forall_cuda_kernel<256ul, RAJA::Iterators	7.25e+11 88.5%	6.46e+11 93.1%	4.12e+09 91.9%
37: [I] __device_stub__ZN4RAJA6policy4cuda4impl18forall_cuda_kerne	7.25e+11 88.5%	6.46e+11 93.1%	4.12e+09 91.9%
26: [I] cudaLaunchKernel<char>	7.25e+11 88.5%	6.46e+11 93.1%	4.12e+09 91.9%
209: cudaLaunchKernel	7.25e+11 88.5%	6.46e+11 93.1%	4.12e+09 91.9%
<cuda kernel>	7.25e+11 88.5%	6.46e+11 93.1%	4.12e+09 91.9%
void RAJA::policy::cuda::impl::forall_cuda_kernel<256ul, RAJ/	7.25e+11 88.5%	6.46e+11 93.1%	4.12e+09 91.9%
151: RAJA::internal::Privatizer<rajaperf::stream::DOT::runCu	6.10e+11 74.5%	5.48e+11 78.9%	3.53e+09 78.8%
54: rajaperf::stream::DOT::runCudaVariant(rajaperf::Varia	5.97e+11 72.9%	5.35e+11 77.1%	3.52e+09 78.6%
129: RAJA::ReduceSum<RAJA::policy::cuda::cuda_red	5.85e+11 71.4%	5.24e+11 75.4%	3.51e+09 78.4%
190: RAJA::cuda::Reduce<false, RAJA::reduce::sum	5.73e+11 69.9%	5.12e+11 73.8%	3.50e+09 78.2%
848: RAJA::cuda::Reduce<false, RAJA::reduce::su	5.61e+11 68.5%	5.01e+11 72.2%	3.50e+09 78.0%
843: RAJA::cuda::Reduce_Data<false, RAJA::re	5.39e+11 65.9%	4.81e+11 69.3%	3.47e+09 77.5%
[I] inlined from reduce.hpp: 203	4.00e+11 48.8%	3.48e+11 50.1%	3.41e+09 76.1%
reduce.hpp: 293	1.31e+11 16.0%	1.24e+11 17.9%	7.48e+06 0.2%
loop at reduce.hpp: 203	8.78e+10 10.7%	7.09e+10 10.2%	8.15e+08 18.2%
loop at reduce.hpp: 203	4.02e+10 4.9%	3.25e+10 4.7%	4.35e+08 9.7%
205: _INTERNAL_43_tmpxft_000131	1.54e+10 1.9%	1.27e+10 1.8%	3.01e+08 6.7%
reduce.hpp: 205	1.50e+10 1.8%	1.19e+10 1.7%	1.15e+08 2.6%

Approximation of GPU Calling Contexts to Understand Performance



Scope	GPU INST:Sum (I)	GPU STALL:Sum (I)	MIX:INTEGER.ADD3:Sum (I)
143: [I] void RAJA::forall<RAJA::policy::cuda::cuda_exec<256ul, true>, RAJA::TypedRa	7.25e+11 88.5%	6.46e+11 93.1%	4.12e+09 91.9%
723: [I] std::enable_if<camp::concepts::all_of<camp::concepts::metalib::negate_t<R	7.25e+11 88.5%	6.46e+11 93.1%	4.12e+09 91.9%
370: [I] std::enable_if<camp::concepts::all_of<camp::concepts::metalib::negate_t	7.25e+11 88.5%	6.46e+11 93.1%	4.12e+09 91.9%
183: [I] void RAJA::policy::cuda::forall_impl<RAJA::TypedRangeSegment<long	7.25e+11 88.5%	6.46e+11 93.1%	4.12e+09 91.9%
190: void RAJA::policy::cuda::impl::forall_cuda_kernel<256ul, RAJA::Iterator	7.25e+11 88.5%	6.46e+11 93.1%	4.12e+09 91.9%
145: [I] __wrapper__device_stub_forall_cuda_kernel<256ul, RAJA::Iterators	7.25e+11 88.5%	6.46e+11 93.1%	4.12e+09 91.9%
37: [I] __device_stub__ZN4RAJA6policy4cuda4impl18forall_cuda_kerne	7.25e+11 88.5%	6.46e+11 93.1%	4.12e+09 91.9%
26: [I] cudaLaunchKernel<char>	7.25e+11 88.5%	6.46e+11 93.1%	4.12e+09 91.9%
209: cudaLaunchKernel	7.25e+11 88.5%	6.46e+11 93.1%	4.12e+09 91.9%
<cuda kernel>	7.25e+11 88.5%	6.46e+11 93.1%	4.12e+09 91.9%
void RAJA::policy::cuda::impl::forall_cuda_kernel<256ul, RAJ	7.25e+11 88.5%	6.46e+11 93.1%	4.12e+09 91.9%
151: RAJA::internal::Privatizer<rajaperf::stream::DOT::runCu	6.10e+11 74.5%	5.48e+11 78.9%	3.53e+09 78.8%
54: rajaperf::stream::DOT::runCudaVariant(rajaperf::Varia	5.97e+11 72.9%	5.35e+11 77.1%	3.52e+09 78.6%
129: RAJA::ReduceSum<RAJA::policy::cuda::cuda_red	5.85e+11 71.4%	5.24e+11 75.4%	3.51e+09 78.4%
190: RAJA::cuda::Reduce<false, RAJA::reduce::sum	5.73e+11 69.9%	5.12e+11 73.8%	3.50e+09 78.2%
848: RAJA::cuda::Reduce<false, RAJA::reduce::su	5.61e+11 68.5%	5.01e+11 72.2%	3.50e+09 78.0%
843: RAJA::cuda::Reduce_Data<false, RAJA::re	5.39e+11 65.9%	4.81e+11 69.3%	3.47e+09 77.5%
[I] inlined from reduce.hpp: 203	4.00e+11 48.8%	3.48e+11 50.1%	3.41e+09 76.1%
reduce.hpp: 293	1.31e+11 16.0%	1.24e+11 17.9%	7.48e+06 0.2%
loop at reduce.hpp: 203	8.78e+10 10.7%	7.09e+10 10.2%	8.15e+08 18.2%
loop at reduce.hpp: 203	4.02e+10 4.9%	3.25e+10 4.7%	4.35e+08 9.7%
205: _INTERNAL_43_tmpxft_000131	1.54e+10 1.9%	1.27e+10 1.8%	3.01e+08 6.7%
reduce.hpp: 205	1.50e+10 1.8%	1.19e+10 1.7%	1.15e+08 2.6%

CPU Calling Context

GPU API Node

GPU Calling Context

GPU Loops

GPU Hotspot

Accuracy of GPU Calling Context Recovery: Case Studies

- **Compute approximate call counts as the basis for partitioning the cost of function invocations across call sites**
  - Use call samples at call sites, data flow analysis to propagate call approximation upward
    - if samples were collected in some function f, if no calls to f were sampled, equally attribute f to each of its call sites
  - How accurate is our approximation?
- **Evaluation methodology**
  - Use NVIDIA's nvbit to
    - instrument call and return for GPU functions
    - instrument basic blocks to collect block histogram

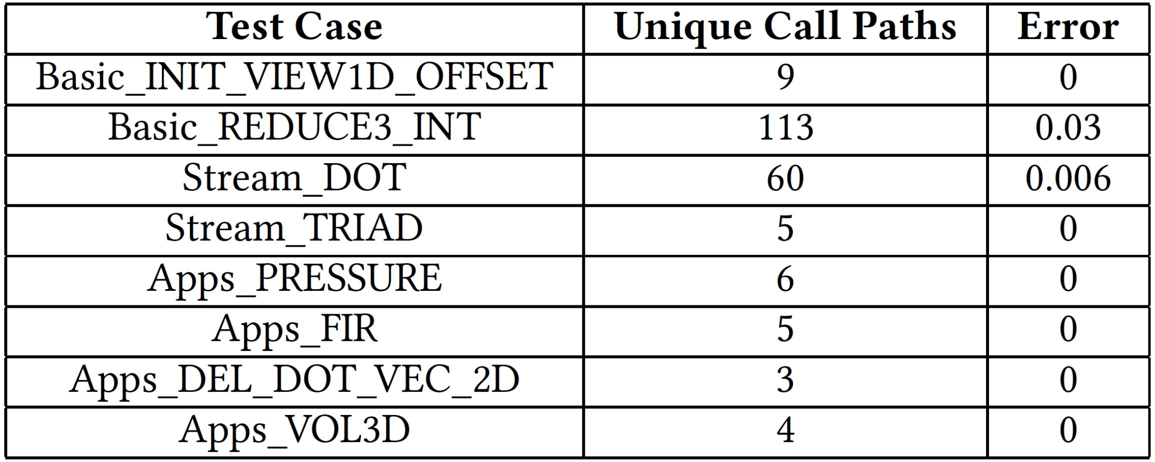
Accuracy of GPU Calling Context Recovery: Case Studies

- **Error partitioning a function’s cost among call sites**

$$Error = \sqrt{\sum_{i=0}^{n-1} \frac{\left(\sqrt{\sum_{j=0}^{i_c-1} \frac{(f_N(i,j)-f_H(i,j))^2}{i_c}}\right)^2}{n}}$$

geometric mean across GPU functions of (root mean square error of call attribution across all of a function’s call sites comparing our approximation vs. attribution using exact nvbit measurements)

- **Experimental study**



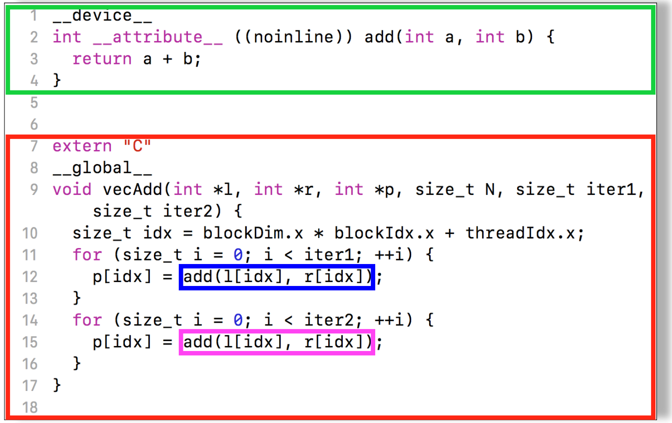
Test Case	Unique Call Paths	Error
Basic_INIT_VIEW1D_OFFSET	9	0
Basic_REDUCE3_INT	113	0.03
Stream_DOT	60	0.006
Stream_TRIAD	5	0
Apps_PRESSURE	6	0
Apps_FIR	5	0
Apps_DEL_DOT_VEC_2D	3	0
Apps_VOL3D	4	0

Costs of GPU Functions Distributed Among Their Call Sites

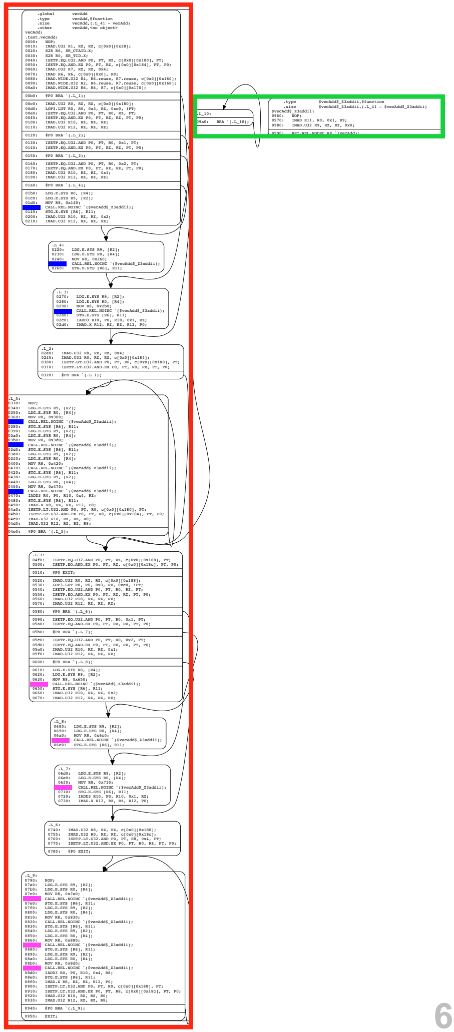
- **Use call site frequency approximation**
- **Use Gprof assumption: all calls to a function incur exactly the same cost**
  - known to not be true in all cases, but a useful assumption nevertheless

GPU call site attribution example

- **Case study: call function GPU “vectorAdd”***
  - iter1 = N
  - iter2 = 2N

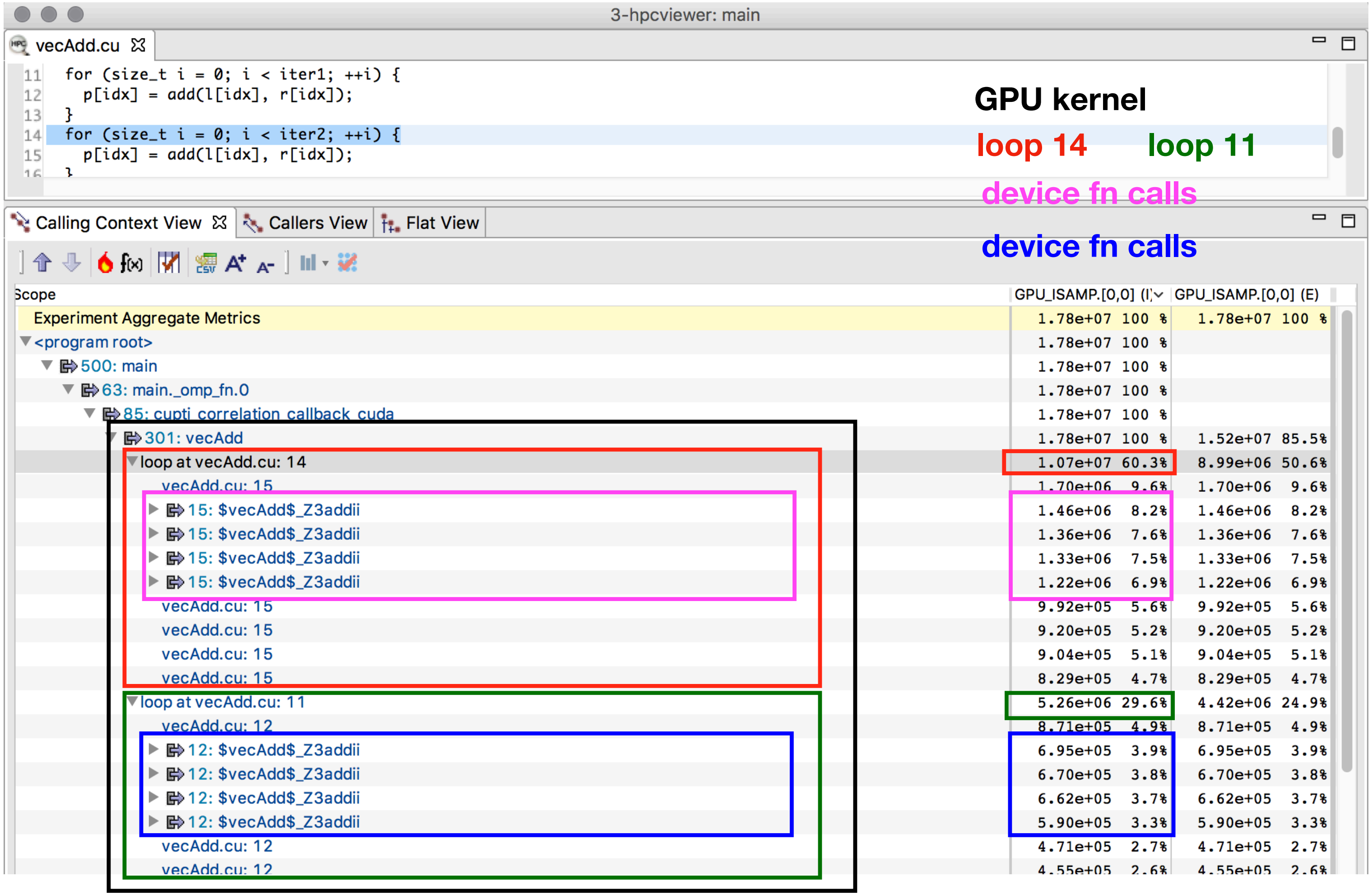


```
__device__
int __attribute__ ((noinline)) add(int a, int b) {
  return a + b;
}


extern "C"
__global__
void vecAdd(int *l, int *r, int *p, size_t N, size_t iter1,
    size_t iter2) {
  size_t idx = blockDim.x * blockIdx.x + threadIdx.x;
  for (size_t i = 0; i < iter1; ++i) {
    p[idx] = add(l[idx], r[idx]);
  }
  for (size_t i = 0; i < iter2; ++i) {
    p[idx] = add(l[idx], r[idx]);
  }
}
```



Note: the computation by the function is synthetic and is not a vector addition. The name came from code that was hacked to do perform an unrelated computation.

Profiling Result for GPU-accelerated Example



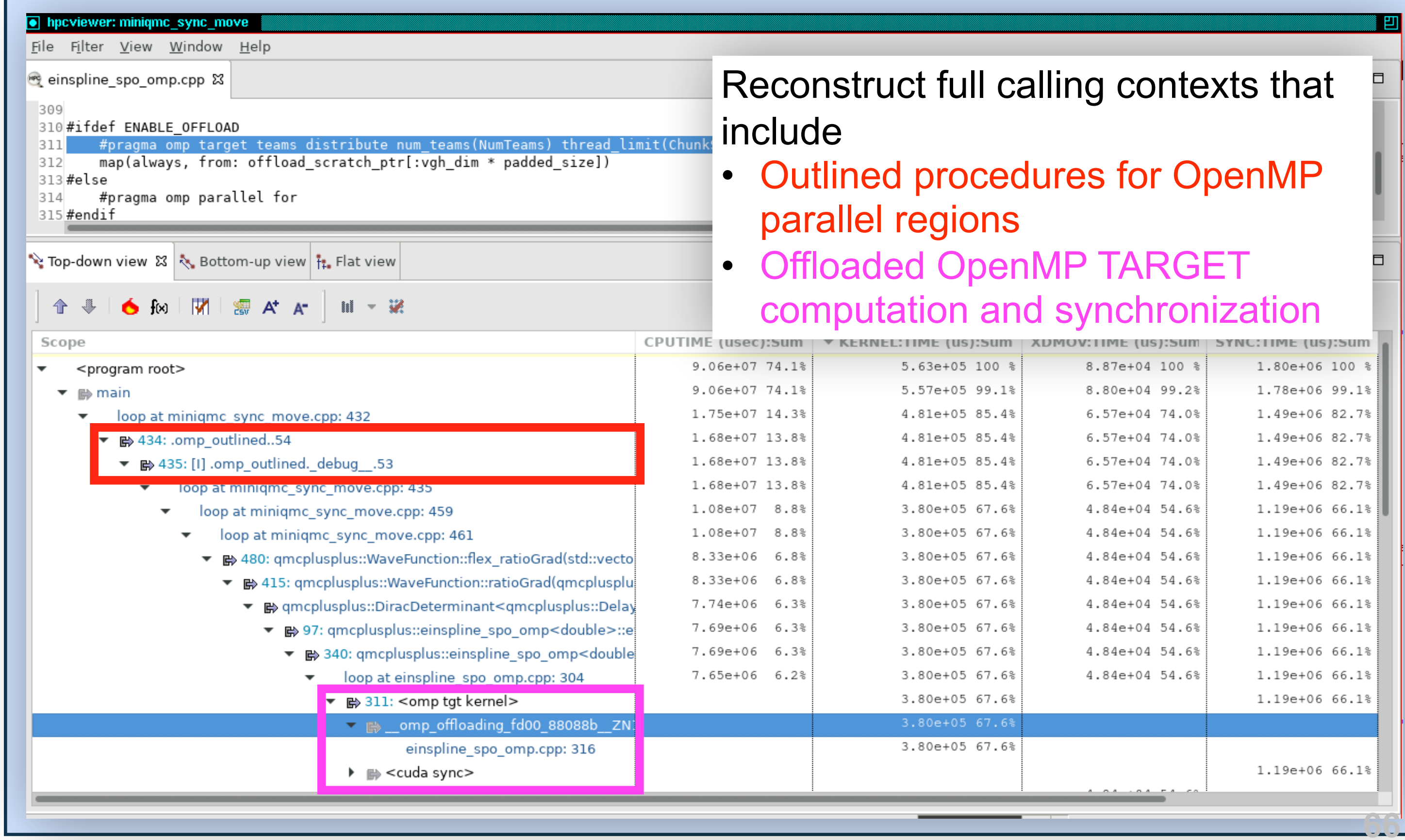
```
for (size_t i = 0; i < iter1; ++i) {
  p[idx] = add(l[idx], r[idx]);
}
for (size_t i = 0; i < iter2; ++i) {
  p[idx] = add(l[idx], r[idx]);
}
```

Scope	GPU_ISAMP.[0,0] (I	GPU_ISAMP.[0,0] (E)
Experiment Aggregate Metrics	1.78e+07 100 %	1.78e+07 100 %
<program root>	1.78e+07 100 %	
500: main	1.78e+07 100 %	
63: main._omp_fn.0	1.78e+07 100 %	
85: cupti_correlation_callback_cuda	1.78e+07 100 %	
301: vecAdd	1.78e+07 100 %	1.52e+07 85.5%
loop at vecAdd.cu: 14	1.07e+07 60.3%	8.99e+06 50.6%
vecAdd.cu: 15	1.70e+06 9.6%	1.70e+06 9.6%
15: $vecAdd$_Z3addii	1.46e+06 8.2%	1.46e+06 8.2%
15: $vecAdd$_Z3addii	1.36e+06 7.6%	1.36e+06 7.6%
15: $vecAdd$_Z3addii	1.33e+06 7.5%	1.33e+06 7.5%
15: $vecAdd$_Z3addii	1.22e+06 6.9%	1.22e+06 6.9%
vecAdd.cu: 15	9.92e+05 5.6%	9.92e+05 5.6%
vecAdd.cu: 15	9.20e+05 5.2%	9.20e+05 5.2%
vecAdd.cu: 15	9.04e+05 5.1%	9.04e+05 5.1%
vecAdd.cu: 15	8.29e+05 4.7%	8.29e+05 4.7%
loop at vecAdd.cu: 11	5.26e+06 29.6%	4.42e+06 24.9%
vecAdd.cu: 12	8.71e+05 4.9%	8.71e+05 4.9%
12: $vecAdd$_Z3addii	6.95e+05 3.9%	6.95e+05 3.9%
12: $vecAdd$_Z3addii	6.70e+05 3.8%	6.70e+05 3.8%
12: $vecAdd$_Z3addii	6.62e+05 3.7%	6.62e+05 3.7%
12: $vecAdd$_Z3addii	5.90e+05 3.3%	5.90e+05 3.3%
vecAdd.cu: 12	4.71e+05 2.7%	4.71e+05 2.7%
vecAdd.cu: 12	4.55e+05 2.6%	4.55e+05 2.6%

Support for OpenMP TARGET

- HPCToolkit implementation of OMPT OpenMP API
    - host monitoring
        - leverages callbacks for regions, threads, tasks
        - employs OMPT API for call stack introspection
    - GPU monitoring
        - leverages callbacks for device initialization, kernel launch, data operations
    - reconstruction of user-level calling contexts
- Leverages implementation of OMPT in LLVM OpenMP and libomptarget

ECP QMCPACK Project: miniqmc using OpenMP TARGET
(Power9 + NVIDIA V100)



```
309
310 #ifdef ENABLE_OFFLOAD
311     #pragma omp target teams distribute num_teams(NumTeams) thread_limit(Chunk
312     map(always, from: offload_scratch_ptr[:vgh_dim * padded_size])
313 #else
314     #pragma omp parallel for
315 #endif
```

Scope	CPUTIME (usec):Sum	KERNEL:TIME (us):Sum	XDMOV:TIME (us):Sum	SYNC:TIME (us):Sum
<program root>	9.06e+07 74.1%	5.63e+05 100 %	8.87e+04 100 %	1.80e+06 100 %
main	9.06e+07 74.1%	5.57e+05 99.1%	8.80e+04 99.2%	1.78e+06 99.1%
loop at miniqmc_sync_move.cpp: 432	1.75e+07 14.3%	4.81e+05 85.4%	6.57e+04 74.0%	1.49e+06 82.7%
434: .omp_outlined..54	1.68e+07 13.8%	4.81e+05 85.4%	6.57e+04 74.0%	1.49e+06 82.7%
435: [I] .omp_outlined._debug__.53	1.68e+07 13.8%	4.81e+05 85.4%	6.57e+04 74.0%	1.49e+06 82.7%
loop at miniqmc_sync_move.cpp: 435	1.68e+07 13.8%	4.81e+05 85.4%	6.57e+04 74.0%	1.49e+06 82.7%
loop at miniqmc_sync_move.cpp: 459	1.08e+07 8.8%	3.80e+05 67.6%	4.84e+04 54.6%	1.19e+06 66.1%
loop at miniqmc_sync_move.cpp: 461	1.08e+07 8.8%	3.80e+05 67.6%	4.84e+04 54.6%	1.19e+06 66.1%
480: qmcplusplus::WaveFunction::flex_ratioGrad(std::vecto	8.33e+06 6.8%	3.80e+05 67.6%	4.84e+04 54.6%	1.19e+06 66.1%
415: qmcplusplus::WaveFunction::ratioGrad(qmcplusplu	8.33e+06 6.8%	3.80e+05 67.6%	4.84e+04 54.6%	1.19e+06 66.1%
qmcplusplus::DiracDeterminant<qmcplusplus::Delay	7.74e+06 6.3%	3.80e+05 67.6%	4.84e+04 54.6%	1.19e+06 66.1%
97: qmcplusplus::einspline_spo_omp<double>::e	7.69e+06 6.3%	3.80e+05 67.6%	4.84e+04 54.6%	1.19e+06 66.1%
340: qmcplusplus::einspline_spo_omp<double	7.69e+06 6.3%	3.80e+05 67.6%	4.84e+04 54.6%	1.19e+06 66.1%
loop at einspline_spo_omp.cpp: 304	7.65e+06 6.2%	3.80e+05 67.6%	4.84e+04 54.6%	1.19e+06 66.1%
311: <omp tgt kernel>		3.80e+05 67.6%		1.19e+06 66.1%
__omp_offloading_fd00_88088b__ZN		3.80e+05 67.6%		
einspline_spo_omp.cpp: 316		3.80e+05 67.6%		
<cuda sync>				1.19e+06 66.1%

Reconstruct full calling contexts that include

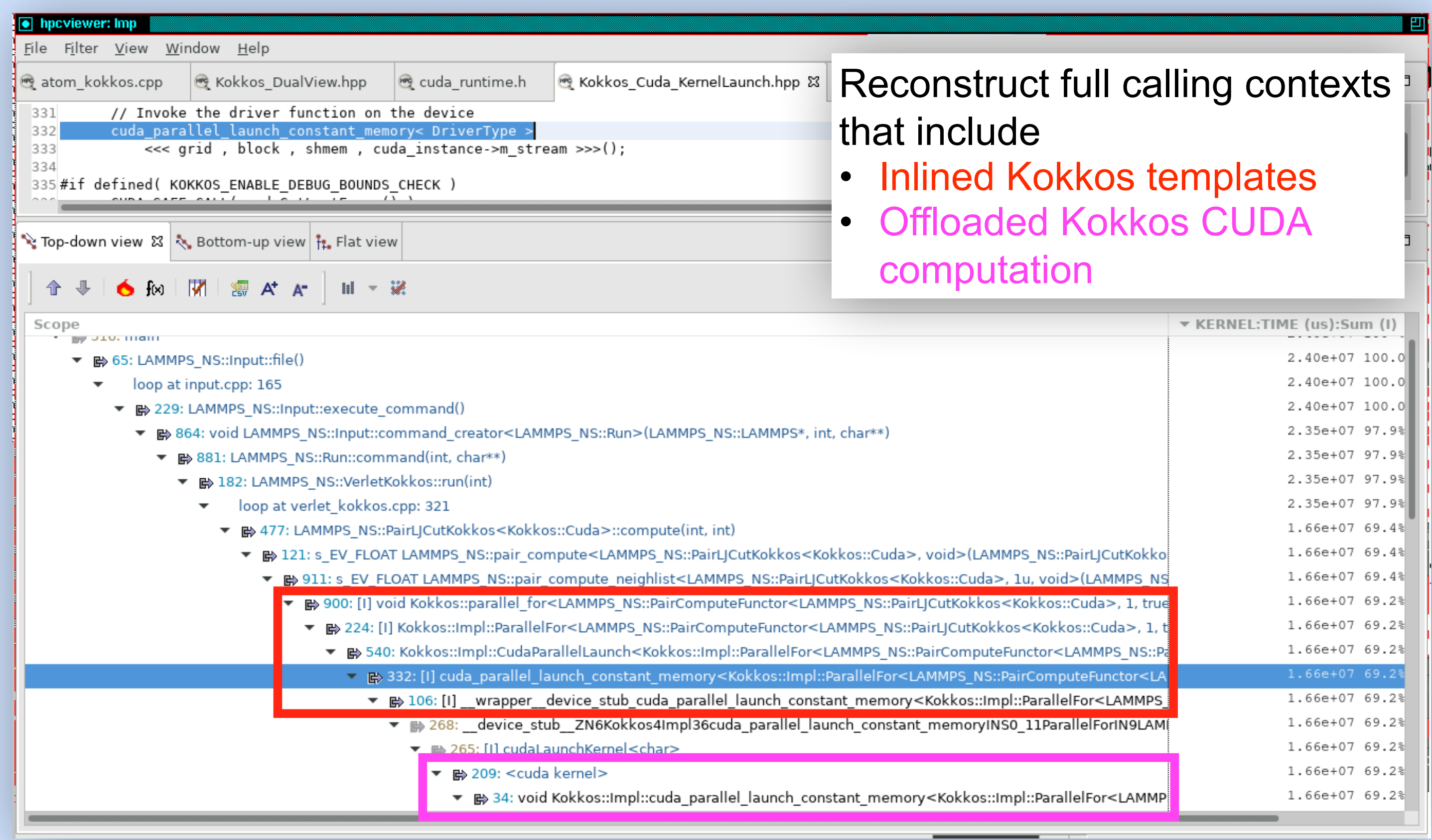
- Outlined procedures for OpenMP parallel regions
- Offloaded OpenMP TARGET computation and synchronization

EXASCALE COMPUTING PROJECT

Support for RAJA and and Kokkos C++ Template-based Models

- RAJA and Kokkos provide portability layers atop C++ template-based programming abstractions
- HPCToolkit employs binary analysis to recover information about procedures, inlined functions and templates, and loops
  - Enables both developers and users to understand complex template instantiation present with these models

ECP EXAALT Project: lammps using Kokkos over CUDA
(Power9 + NVIDIA V100)



```
    // Invoke the driver function on the device
    cuda_parallel_launch_constant_memory< DriverType >
        <<< grid , block , shmem , cuda_instance->m_stream >>>();

#if defined( KOKKOS_ENABLE_DEBUG_BOUNDS_CHECK )
```

```
65: LAMMPS_NS::Input::file()                                                                                           2.40e+07 100.0
  loop at input.cpp: 165                                                                                               2.40e+07 100.0
    229: LAMMPS_NS::Input::execute_command()                                                                           2.40e+07 100.0
      864: void LAMMPS_NS::Input::command_creator<LAMMPS_NS::Run>(LAMMPS_NS::LAMMPS*, int, char**)                     2.35e+07 97.9%
        881: LAMMPS_NS::Run::command(int, char**)                                                                      2.35e+07 97.9%
          182: LAMMPS_NS::VerletKokkos::run(int)                                                                       2.35e+07 97.9%
            loop at verlet_kokkos.cpp: 321                                                                             2.35e+07 97.9%
              477: LAMMPS_NS::PairLJCutKokkos<Kokkos::Cuda>::compute(int, int)                                         1.66e+07 69.4%
                121: s_EV_FLOAT LAMMPS_NS::pair_compute<LAMMPS_NS::PairLJCutKokkos<Kokkos::Cuda>, void>(LAMMPS_NS::PairLJCutKokko  1.66e+07 69.4%
                  911: s_EV_FLOAT LAMMPS_NS::pair_compute_neighlist<LAMMPS_NS::PairLJCutKokkos<Kokkos::Cuda>, 1u, void>(LAMMPS_NS  1.66e+07 69.4%
                    900: [I] void Kokkos::parallel_for<LAMMPS_NS::PairComputeFunctor<LAMMPS_NS::PairLJCutKokkos<Kokkos::Cuda>, 1, true  1.66e+07 69.2%
                      224: [I] Kokkos::Impl::ParallelFor<LAMMPS_NS::PairComputeFunctor<LAMMPS_NS::PairLJCutKokkos<Kokkos::Cuda>, 1, t  1.66e+07 69.2%
                        540: Kokkos::Impl::CudaParallelLaunch<Kokkos::Impl::ParallelFor<LAMMPS_NS::PairComputeFunctor<LAMMPS_NS::Pa  1.66e+07 69.2%
                          332: [I] cuda_parallel_launch_constant_memory<Kokkos::Impl::ParallelFor<LAMMPS_NS::PairComputeFunctor<LA  1.66e+07 69.2%
                            106: [I] __wrapper__device_stub_cuda_parallel_launch_constant_memory<Kokkos::Impl::ParallelFor<LAMMPS_  1.66e+07 69.2%
                              268: __device_stub__ZN6Kokkos4Impl36cuda_parallel_launch_constant_memoryINS0_11ParallelForIN9LAM  1.66e+07 69.2%
                                265: [I] cudaLaunchKernel<char>                                                        1.66e+07 69.2%
                                  209: <cuda kernel>                                                                   1.66e+07 69.2%
                                    34: void Kokkos::Impl::cuda_parallel_launch_constant_memory<Kokkos::Impl::ParallelFor<LAMMP  1.66e+07 69.2%
```

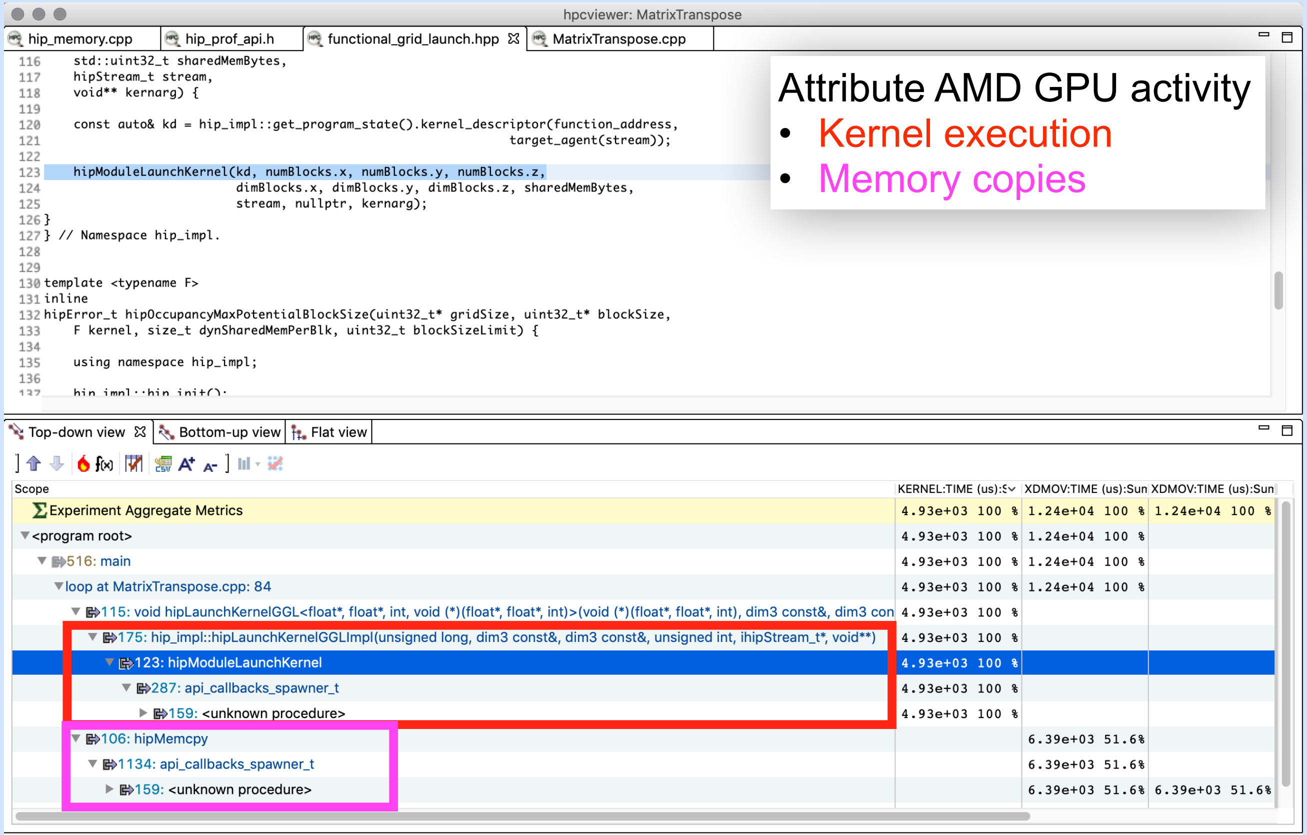
Reconstruct full calling contexts that include

- Inlined Kokkos templates
- Offloaded Kokkos CUDA computation

Prototype Integration with AMD’s Roctracer GPU Monitoring Framework

- Use AMD Roctracer activity API to trace GPU activity
  - kernel launches
  - explicit memory copies
- Current prototype supports AMD’s HIP programming model

AMD MatrixTranspose Testcase for Roctracer
(AMD Ryzen + AMD 580 GPU)



```
    std::uint32_t sharedMemBytes,
    hipStream_t stream,
    void** kernarg) {

    const auto& kd = hip_impl::get_program_state().kernel_descriptor(function_address,
                                                                      target_agent(stream));

    hipModuleLaunchKernel(kd, numBlocks.x, numBlocks.y, numBlocks.z,
                          dimBlocks.x, dimBlocks.y, dimBlocks.z, sharedMemBytes,
                          stream, nullptr, kernarg);
}
} // Namespace hip_impl.


template <typename F>
inline
hipError_t hipOccupancyMaxPotentialBlockSize(uint32_t* gridSize, uint32_t* blockSize,
    F kernel, size_t dynSharedMemPerBlk, uint32_t blockSizeLimit) {

    using namespace hip_impl;

    hip_impl::hip_init();
```

Scope	KERNEL:TIME (us):S	XDMOV:TIME (us):Sun	XDMOV:TIME (us):Sun
Σ Experiment Aggregate Metrics	4.93e+03 100 %	1.24e+04 100 %	1.24e+04 100 %
▼<program root>	4.93e+03 100 %	1.24e+04 100 %	
▼516: main	4.93e+03 100 %	1.24e+04 100 %	
▼loop at MatrixTranspose.cpp: 84	4.93e+03 100 %	1.24e+04 100 %	
▼115: void hipLaunchKernelGGL<float*, float*, int, void (*)(float*, float*, int)>(void (*)(float*, float*, int), dim3 const&, dim3 con	4.93e+03 100 %		
▼175: hip_impl::hipLaunchKernelGGLImpl(unsigned long, dim3 const&, dim3 const&, unsigned int, ihipStream_t*, void**)	4.93e+03 100 %		
▼123: hipModuleLaunchKernel	4.93e+03 100 %		
▼287: api_callbacks_spawner_t	4.93e+03 100 %		
▶159: <unknown procedure>	4.93e+03 100 %		
▼106: hipMemcpy		6.39e+03 51.6%	
▼1134: api_callbacks_spawner_t		6.39e+03 51.6%	
▶159: <unknown procedure>		6.39e+03 51.6%	6.39e+03 51.6%

Attribute AMD GPU activity
- Kernel execution
- Memory copies

EXASCALE COMPUTING PROJECT

HPCToolkit Challenges and Limitations

- **Fine-grain measurement and attribution of GPU performance**
  - PC sampling overhead on NIVIDIA GPUs is currently very high: a function of NVIDIA's CUPTI implementation
  - No available hardware support for fine-grain measurement on Intel and AMD GPUs
- **GPU tracing in HPCToolkit**
  - Creates one tool thread per GPU stream when tracing
  - OK for a small number of streams but many streams can be problematic
- **Cost of call path sampling**
  - Call path unwinding of GPU kernel invocations is costly (~2x execution dilation for Laghos)
  - Best solution is to avoid some of it, e.g. sample GPU kernel invocations
- **Currently, hpcprof and hpcprof-mpi compute dense vectors of metrics**
  - Designed for few CPU metrics, not O(100) GPU metrics: space and time problem for analysis

Outline

- **Performance measurement and analysis challenges for GPU-accelerated supercomputers**
- **Introduction to HPCToolkit performance tools**
  - Overview of HPCToolkit components and their workflow
  - HPCToolkit's graphical user interfaces and using them effectively
- **Analyzing the performance of GPU-accelerated supercomputers with HPCToolkit**
  - Overview of HPCToolkit's GPU performance measurement capabilities
  - Collecting measurements
  - Analysis and attribution
  - Exploring measurements and analysis results
- **Experiences with analysis and tuning of GPU-accelerated codes**
  - Computation, memory hierarchy, and data movement issues
- **Obtaining HPCToolkit**

Analysis and Optimization Case Studies

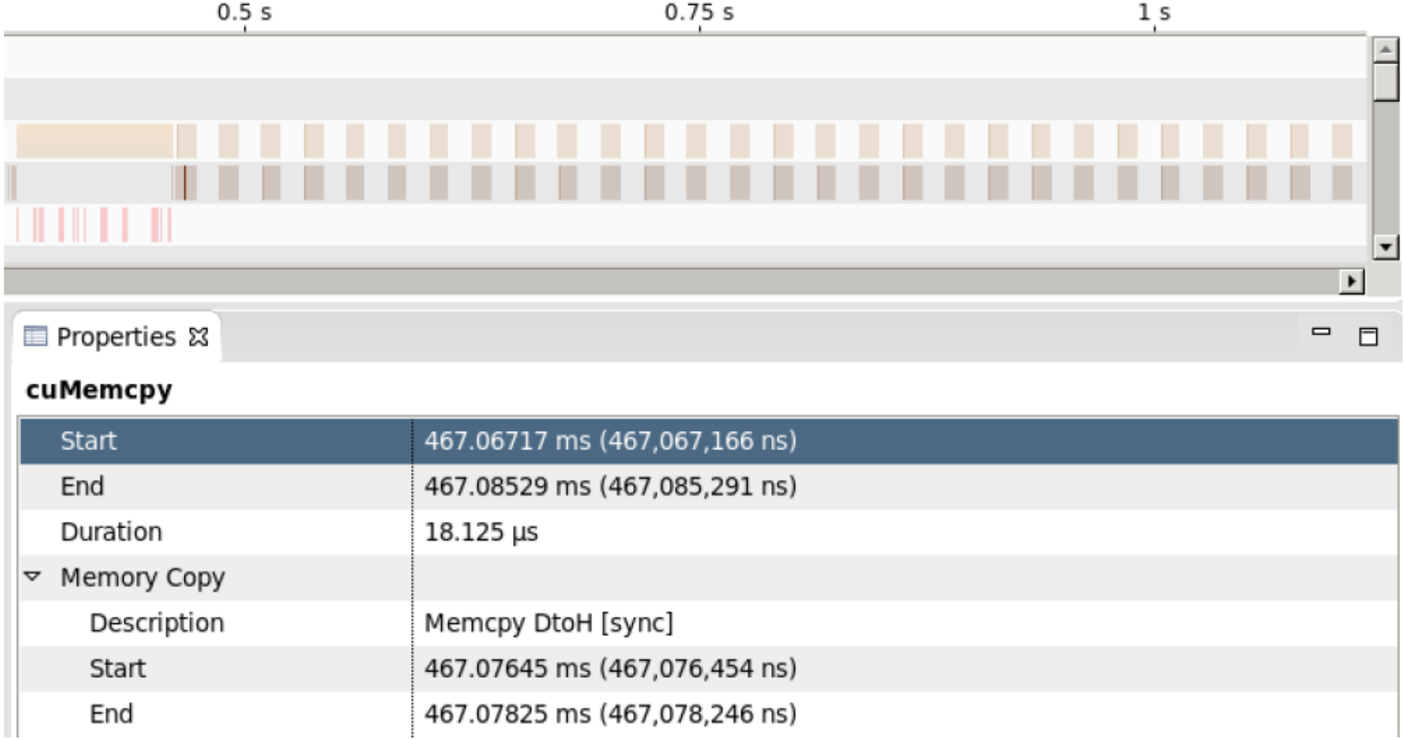
- **Environments**
  - Summit
    - cuda/10.1.168
    - gcc/6.4.0
  - Local
    - cuda/10.1.168
    - gcc/7.3.0

Case 1: Locating expensive GPU APIs with profile view

- **Laghos**
  - 1 MPI process
  - 1 GPU stream per process

nvprof: missing CPU calling context

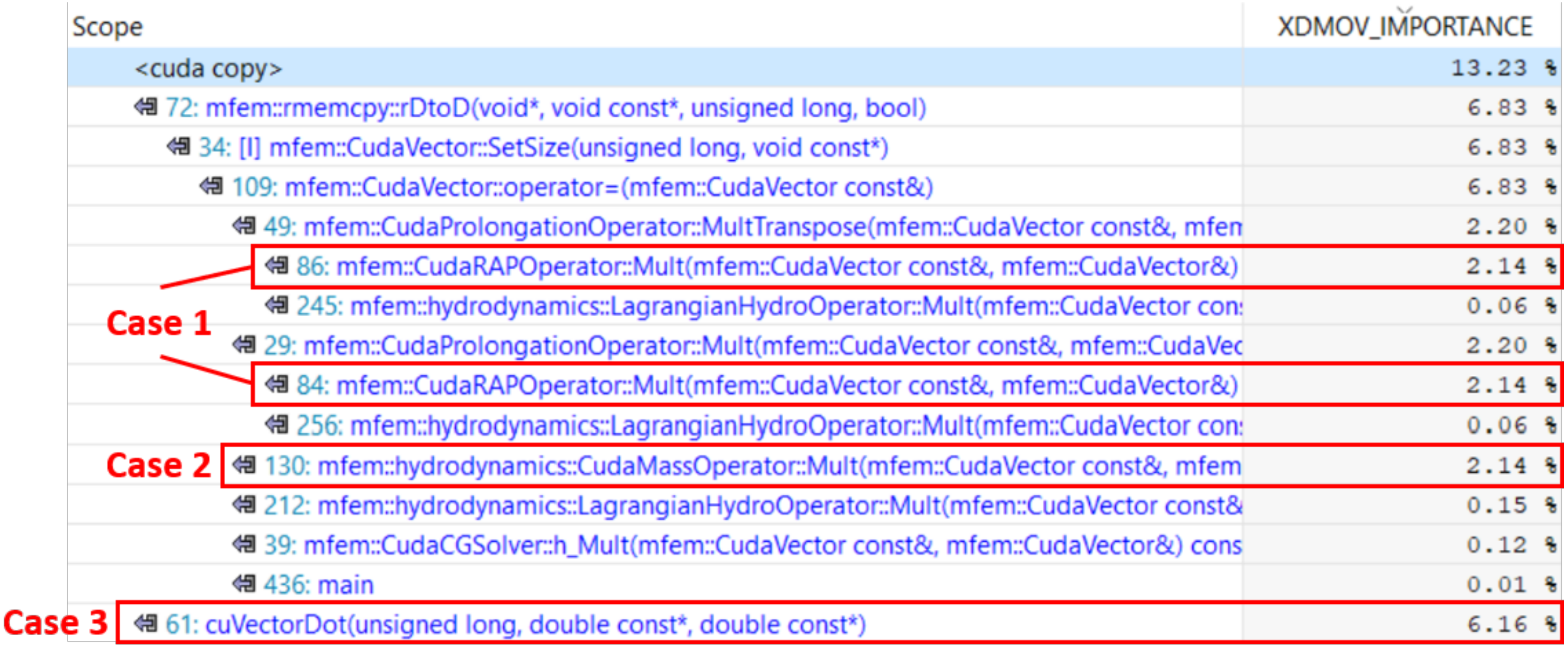
- **Goal: Associate every GPU API with its CPU calling context**



cuMemcpy

Start	467.06717 ms (467,067,166 ns)
End	467.08529 ms (467,085,291 ns)
Duration	18.125 µs
▽ Memory Copy	
Description	Memcpy DtoH [sync]
Start	467.07645 ms (467,076,454 ns)
End	467.07825 ms (467,078,246 ns)

Context-aware optimizations



Scope	XDMOV_IMPORTANCE
<cuda copy>	13.23 %
72: mfem::rmemcpy::rDtoD(void*, void const*, unsigned long, bool)	6.83 %
34: [I] mfem::CudaVector::SetSize(unsigned long, void const*)	6.83 %
109: mfem::CudaVector::operator=(mfem::CudaVector const&)	6.83 %
49: mfem::CudaProlongationOperator::MultTranspose(mfem::CudaVector const&, mfen	2.20 %
86: mfem::CudaRAPOperator::Mult(mfem::CudaVector const&, mfem::CudaVector&)	2.14 %
245: mfem::hydrodynamics::LagrangianHydroOperator::Mult(mfem::CudaVector con:	0.06 %
29: mfem::CudaProlongationOperator::Mult(mfem::CudaVector const&, mfem::CudaVec	2.20 %
84: mfem::CudaRAPOperator::Mult(mfem::CudaVector const&, mfem::CudaVector&)	2.14 %
256: mfem::hydrodynamics::LagrangianHydroOperator::Mult(mfem::CudaVector con:	0.06 %
130: mfem::hydrodynamics::CudaMassOperator::Mult(mfem::CudaVector const&, mfem	2.14 %
212: mfem::hydrodynamics::LagrangianHydroOperator::Mult(mfem::CudaVector const&	0.15 %
39: mfem::CudaCGSolver::h_Mult(mfem::CudaVector const&, mfem::CudaVector&) cons	0.12 %
436: main	0.01 %
61: cuVectorDot(unsigned long, double const*, double const*)	6.16 %

Performance insight: Pin host memory page

- **A small amount of memory is transferred from device to host each time, repeated 197000 times**

Scope	▼ GXCOPY (s):Sum (I)	GXCOPY:COUNT:Sum (I)	GXCOPY:D2H (B):Sum (I)
▼ 61: cuVectorDot(unsigned long, double const*, double const*)	3.67e-01 46.3%	1.97e+05 37.9%	7.81e+06 20.4%

- **Avoid the cost of the transfer between pageable and pinned host arrays by directly allocating our host arrays in pinned memory**
  - Use pinned memory when data movement frequency is high but size is small

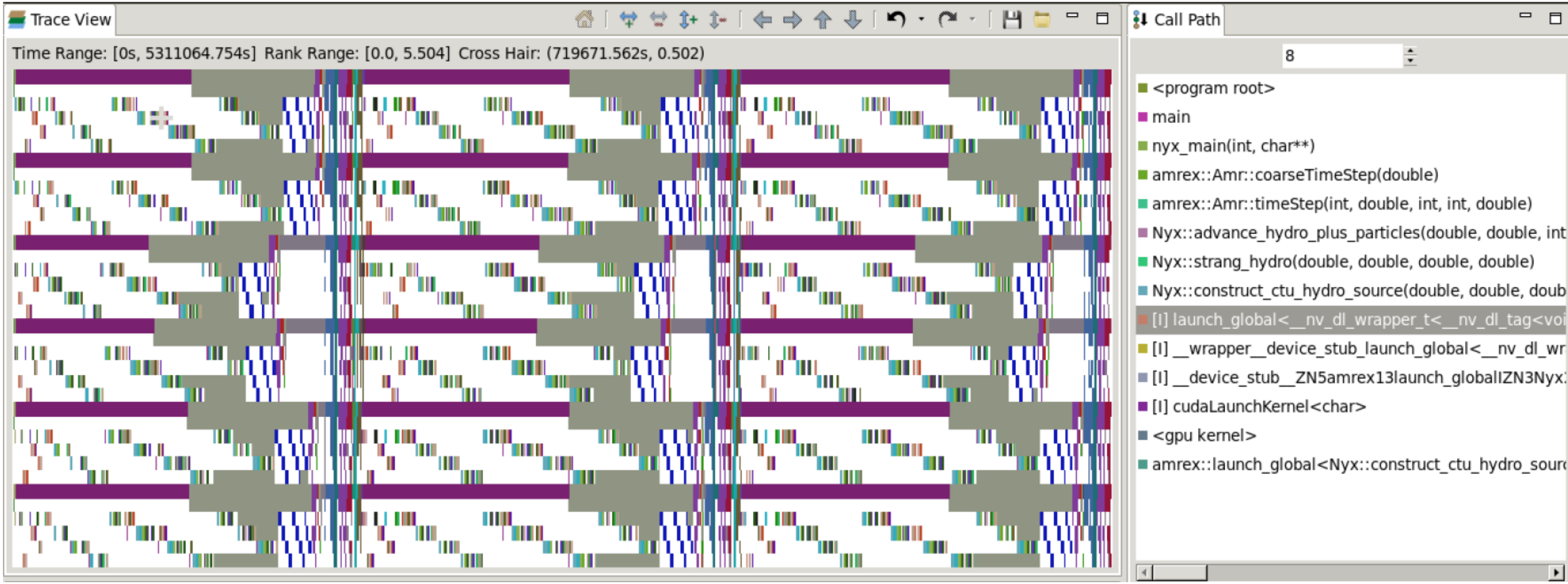
Case 2: Trace Applications at Large-scale

- **Nyx**
  - 6 MPI processes
  - 16 GPU stream per process
- **DCA++**
  - 60 MPI processes
  - 128 GPU stream per process

nvprof: Non-scalable Tracing of DCA++

- **nvprof**
  - With CPU profiling enabled, hangs on Summit
  - Without CPU profiling
    - Collects 1.1 GB data
- **Hpctoolkit**
  - CPU+GPU hybrid profiling with full calling context
    - Collects 0.13 GB data
    - Data can be further reduced by sampling GPU events

Nyx trace view

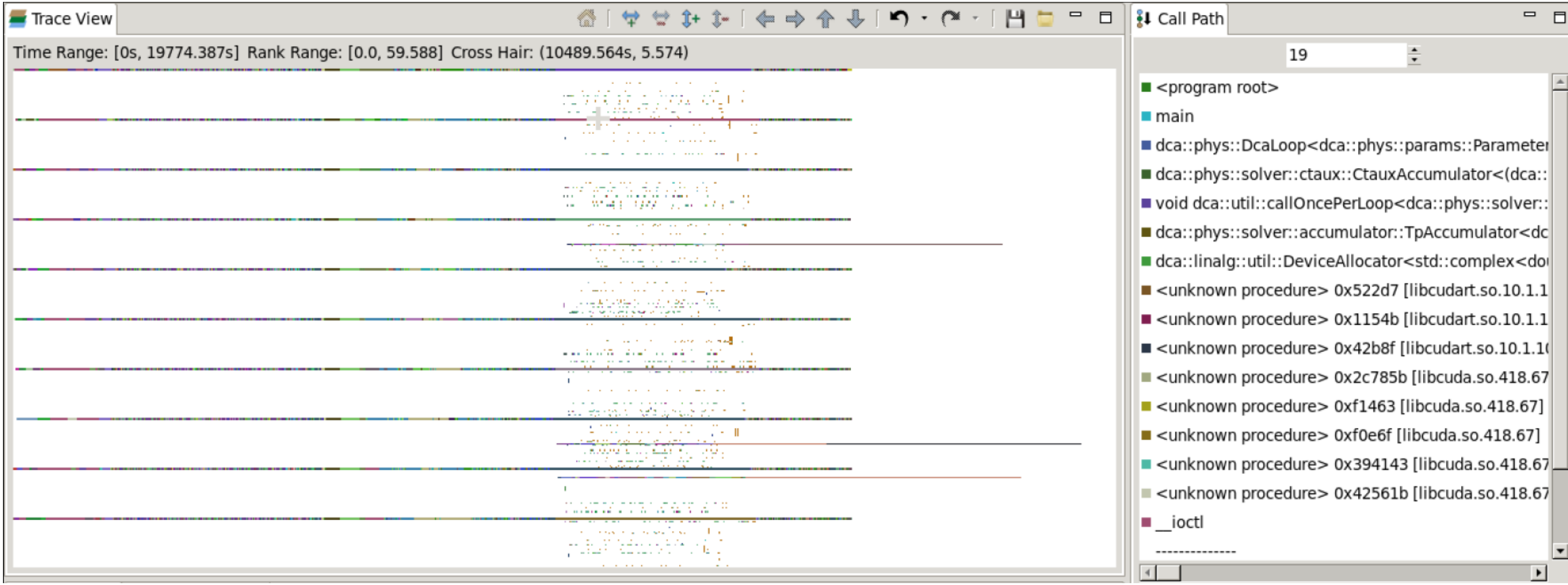


Time Range: [0s, 5311064.754s] Rank Range: [0.0, 5.504] Cross Hair: (719671.562s, 0.502)

8

- <program root>
- main
- nyx_main(int, char**)
- amrex::Amr::coarseTimeStep(double)
- amrex::Amr::timeStep(int, double, int, int, double)
- Nyx::advance_hydro_plus_particles(double, double, int
- Nyx::strang_hydro(double, double, double, double)
- Nyx::construct_ctu_hydro_source(double, double, doub
- [I] launch_global<__nv_dl_wrapper_t<__nv_dl_tag<voi
- [I] __wrapper__device_stub_launch_global<__nv_dl_wr
- [I] __device_stub__ZN5amrex13launch_globalIZN3Nyx
- [I] cudaLaunchKernel<char>
- <gpu kernel>
- amrex::launch_global<Nyx::construct_ctu_hydro_sour

DCA++ trace view



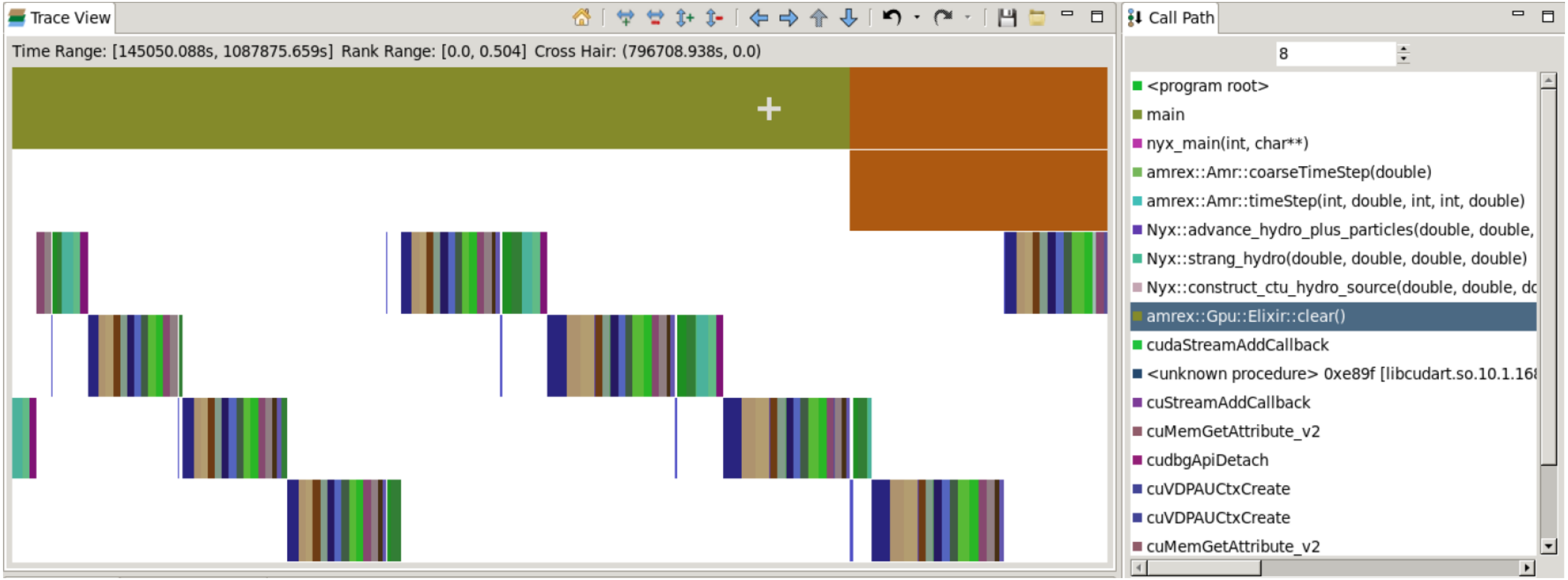
Time Range: [0s, 19774.387s] Rank Range: [0.0, 59.588] Cross Hair: (10489.564s, 5.574)

- ■ <program root>
- ■ main
- ■ dca::phys::DcaLoop<dca::phys::params::Paramete
- ■ dca::phys::solver::ctaux::CtauxAccumulator<(dca::
- ■ void dca::util::callOncePerLoop<dca::phys::solver::
- ■ dca::phys::solver::accumulator::TpAccumulator<dc
- ■ dca::linalg::util::DeviceAllocator<std::complex<do
- ■ <unknown procedure> 0x522d7 [libcudart.so.10.1.1
- ■ <unknown procedure> 0x1154b [libcudart.so.10.1.1
- ■ <unknown procedure> 0x42b8f [libcudart.so.10.1.1
- ■ <unknown procedure> 0x2c785b [libcuda.so.418.67
- ■ <unknown procedure> 0xf1463 [libcuda.so.418.67]
- ■ <unknown procedure> 0xf0e6f [libcuda.so.418.67]
- ■ <unknown procedure> 0x394143 [libcuda.so.418.67
- ■ <unknown procedure> 0x42561b [libcuda.so.418.67
- ■ __ioctl

EXASCALE COMPUTING PROJECT

Nyx insufficient GPU stream parallelism

- **On GPU, streams are not working concurrently**



Time Range: [145050.088s, 1087875.659s] Rank Range: [0.0, 0.504] Cross Hair: (796708.938s, 0.0)

- <program root>
- main
- nyx_main(int, char**)
- amrex::Amr::coarseTimeStep(double)
- amrex::Amr::timeStep(int, double, int, int, double)
- Nyx::advance_hydro_plus_particles(double, double,
- Nyx::strang_hydro(double, double, double, double)
- Nyx::construct_ctu_hydro_source(double, double, dc
- amrex::Gpu::Elixir::clear()
- cudaStreamAddCallback
- <unknown procedure> 0xe89f [libcudart.so.10.1.16
- cuStreamAddCallback
- cuMemGetAttribute_v2
- cudbgApiDetach
- cuVDPAUCtxCreate
- cuVDPAUCtxCreate
- cuMemGetAttribute_v2

Nyx cudaCallBack issue

- **On CPU, amrex::Gpu::Exlixir::clear() invokes stream callbacks**

```
void
Elixir::clear () noexcept
{
#ifdef AMREX_USE_GPU
    if (Gpu::inLaunchRegion())
    {
        if (m_p != nullptr) {
            void** p = static_cast<void**>(std::malloc(2*sizeof(void*)));
            p[0] = m_p;
            p[1] = (void*)m_arena;
            AMREX_HIP_OR_CUDA(
                AMREX_HIP_SAFE_CALL ( hipStreamAddCallback(Gpu::gpuStream(),
                                                           amrex_elixir_delete, p, 0));,
                AMREX_CUDA_SAFE_CALL(cudaStreamAddCallback(Gpu::gpuStream(),
                                                           amrex_elixir_delete, p, 0)););
            Gpu::callbackAdded();
        }
    }
    else
#endif
```

Nyx performance insight

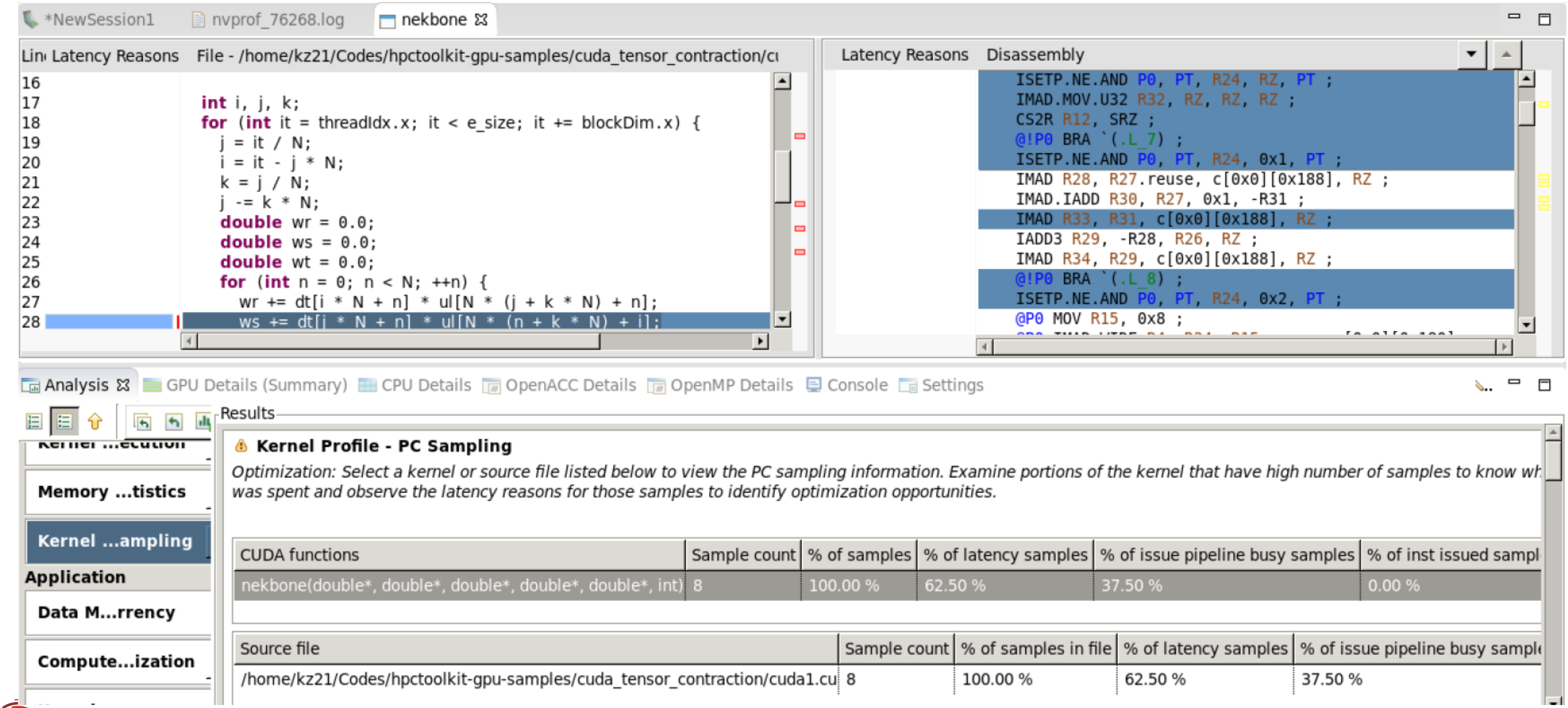
- **A bug present in the current version of CUDA (10.1). If a callBack is called in a place where multiple streams are used, the device kernels artificially synchronize and have no overlap.**
- **Fixed in CUDA-10.2?**
- **Workaround**
  - The Elixir object holds a copy of the data pointer to prevent it from being destroyed before the related device kernels are completed
  - Allocate new objects outside the compute loop and delete them after the completion of the work

Case 3: Fine-grained GPU Kernel Tuning

- **Nekbone: A lightweight subset of Nek5000 that mimics the essential computational complexity of Nek5000**

nvprof: Limited source level performance metrics

- **No loop structure, No GPU calling context, No instruction mix**



```
    int i, j, k;
    for (int it = threadIdx.x; it < e_size; it += blockDim.x) {
      j = it / N;
      i = it - j * N;
      k = j / N;
      j -= k * N;
      double wr = 0.0;
      double ws = 0.0;
      double wt = 0.0;
      for (int n = 0; n < N; ++n) {
        wr += dt[i * N + n] * ul[N * (j + k * N) + n];
        ws += dt[i * N + n] * ul[N * (n + k * N) + i];
```

```
ISETP.NE.AND P0, PT, R24, RZ, PT ;
IMAD.MOV.U32 R32, RZ, RZ, RZ ;
CS2R R12, SRZ ;
@!P0 BRA `(.L_7) ;
ISETP.NE.AND P0, PT, R24, 0x1, PT ;
IMAD R28, R27.reuse, c[0x0][0x188], RZ ;
IMAD.IADD R30, R27, 0x1, -R31 ;
IMAD R33, R31, c[0x0][0x188], RZ ;
IADD3 R29, -R28, R26, RZ ;
IMAD R34, R29, c[0x0][0x188], RZ ;
@!P0 BRA `(.L_8) ;
ISETP.NE.AND P0, PT, R24, 0x2, PT ;
@P0 MOV R15, 0x8 ;
```

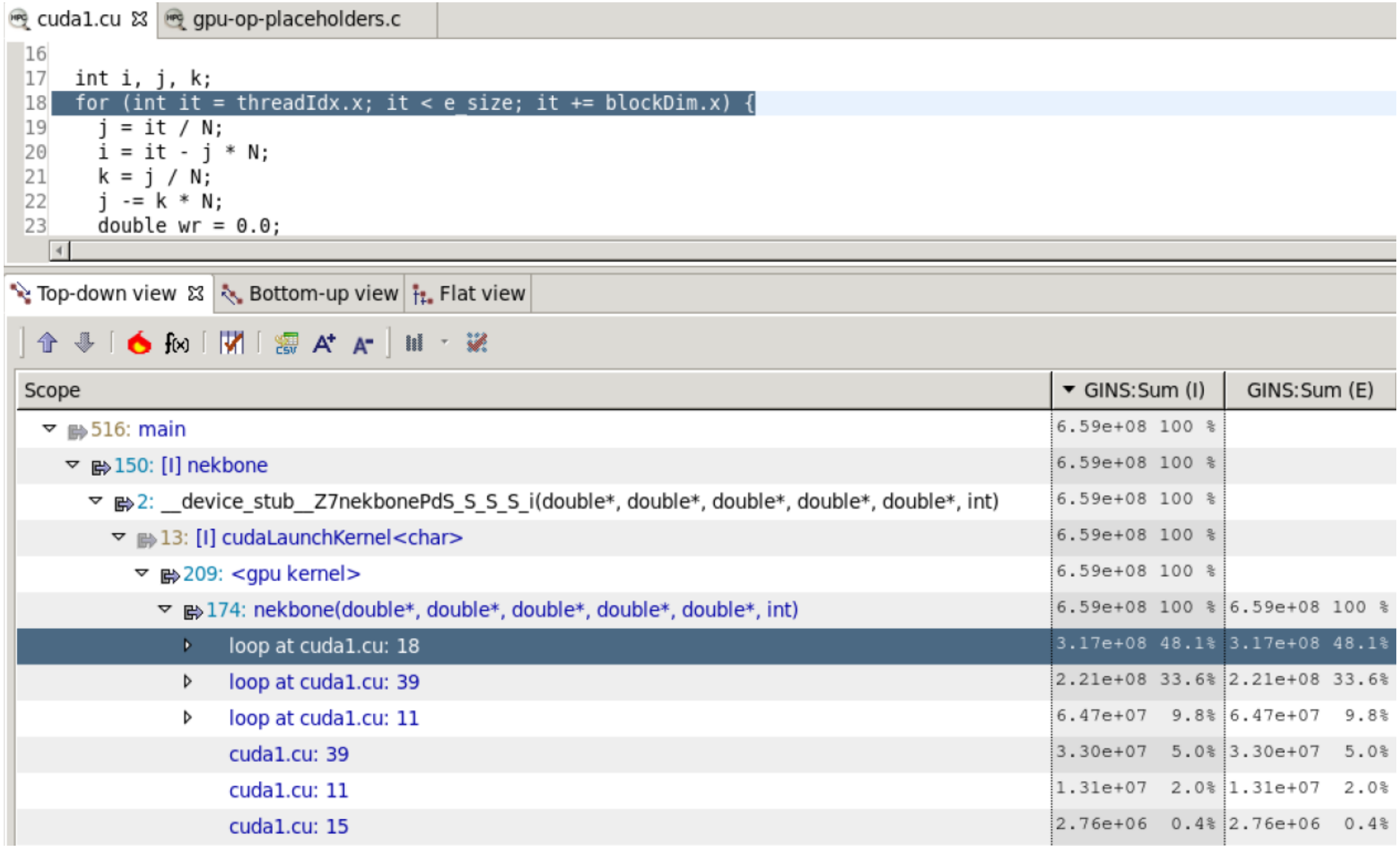
Kernel Profile - PC Sampling

Optimization: Select a kernel or source file listed below to view the PC sampling information. Examine portions of the kernel that have high number of samples to know wh was spent and observe the latency reasons for those samples to identify optimization opportunities.

CUDA functions	Sample count	% of samples	% of latency samples	% of issue pipeline busy samples	% of inst issued sampl
nekbone(double*, double*, double*, double*, double*, int)	8	100.00 %	62.50 %	37.50 %	0.00 %

Source file	Sample count	% of samples in file	% of latency samples	% of issue pipeline busy sample
/home/kz21/Codes/hpctoolkit-gpu-samples/cuda_tensor_contraction/cuda1.cu	8	100.00 %	62.50 %	37.50 %

Nekbone Profile View

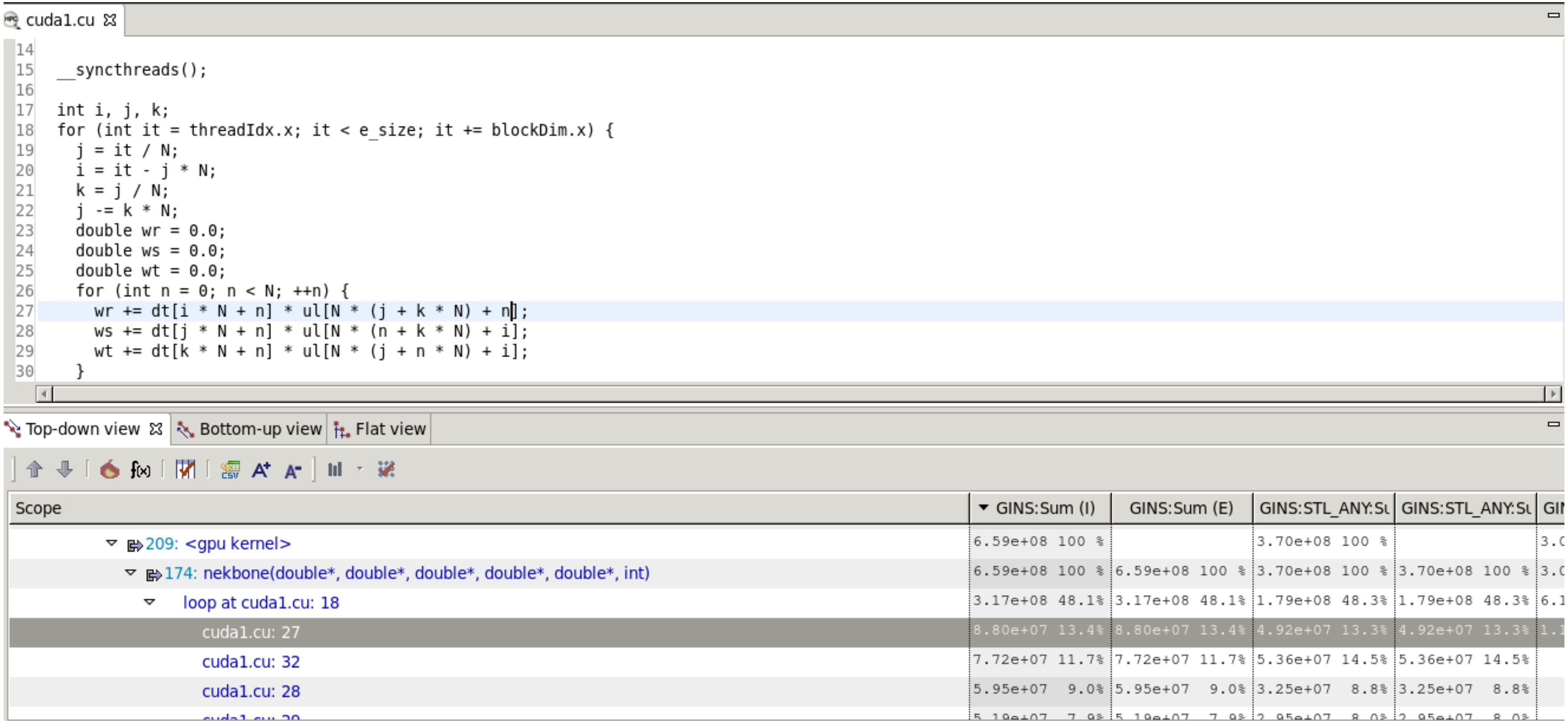


```
16
17   int i, j, k;
18   for (int it = threadIdx.x; it < e_size; it += blockDim.x) {
19     j = it / N;
20     i = it - j * N;
21     k = j / N;
22     j -= k * N;
23     double wr = 0.0;
```

Scope	GINS:Sum (I)	GINS:Sum (E)
516: main	6.59e+08 100 %	
150: [I] nekbone	6.59e+08 100 %	
2: __device_stub__Z7nekbonePdS_S_S_S_i(double*, double*, double*, double*, double*, int)	6.59e+08 100 %	
13: [I] cudaLaunchKernel<char>	6.59e+08 100 %	
209: <gpu kernel>	6.59e+08 100 %	
174: nekbone(double*, double*, double*, double*, double*, int)	6.59e+08 100 %	6.59e+08 100 %
loop at cuda1.cu: 18	3.17e+08 48.1%	3.17e+08 48.1%
loop at cuda1.cu: 39	2.21e+08 33.6%	2.21e+08 33.6%
loop at cuda1.cu: 11	6.47e+07 9.8%	6.47e+07 9.8%
cuda1.cu: 39	3.30e+07 5.0%	3.30e+07 5.0%
cuda1.cu: 11	1.31e+07 2.0%	1.31e+07 2.0%
cuda1.cu: 15	2.76e+06 0.4%	2.76e+06 0.4%

Performance insight 1: Execution dependency

- **The hotspot statement is waiting for *j* and *k***

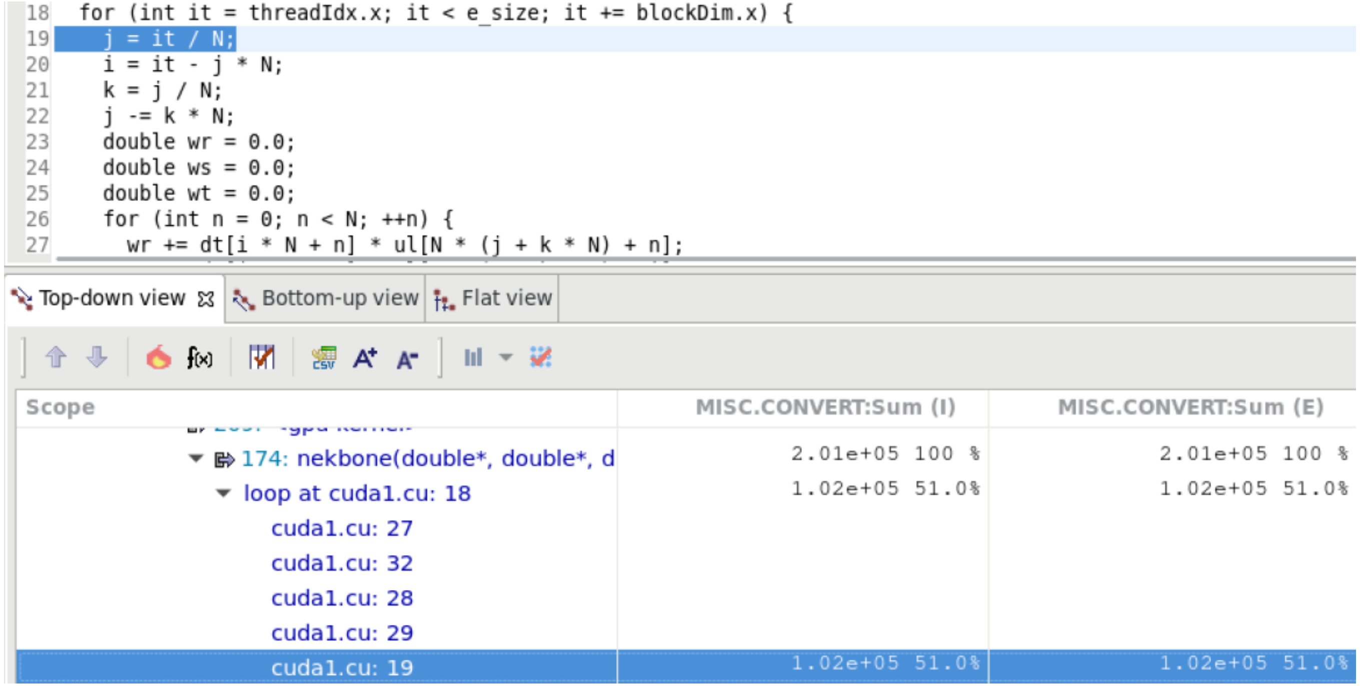


```
14
15   __syncthreads();
16
17   int i, j, k;
18   for (int it = threadIdx.x; it < e_size; it += blockDim.x) {
19     j = it / N;
20     i = it - j * N;
21     k = j / N;
22     j -= k * N;
23     double wr = 0.0;
24     double ws = 0.0;
25     double wt = 0.0;
26     for (int n = 0; n < N; ++n) {
27       wr += dt[i * N + n] * ul[N * (j + k * N) + n];
28       ws += dt[j * N + n] * ul[N * (n + k * N) + i];
29       wt += dt[k * N + n] * ul[N * (j + n * N) + i];
30     }
```

Scope	GINS:Sum (I)	GINS:Sum (E)	GINS:STL_ANY:Su	GINS:STL_ANY:Su	GI
209: <gpu kernel>	6.59e+08 100 %		3.70e+08 100 %		3.0
174: nekbone(double*, double*, double*, double*, double*, int)	6.59e+08 100 %	6.59e+08 100 %	3.70e+08 100 %	3.70e+08 100 %	3.0
loop at cuda1.cu: 18	3.17e+08 48.1%	3.17e+08 48.1%	1.79e+08 48.3%	1.79e+08 48.3%	6.1
cuda1.cu: 27	8.80e+07 13.4%	8.80e+07 13.4%	4.92e+07 13.3%	4.92e+07 13.3%	1.1
cuda1.cu: 32	7.72e+07 11.7%	7.72e+07 11.7%	5.36e+07 14.5%	5.36e+07 14.5%	
cuda1.cu: 28	5.95e+07 9.0%	5.95e+07 9.0%	3.25e+07 8.8%	3.25e+07 8.8%	
cuda1.cu: 29	5.19e+07 7.9%	5.19e+07 7.9%	2.95e+07 8.0%	2.95e+07 8.0%	

Strength reduction

- **MISC.CONVERT: I2F, F2I, MUFU instructions**
  - NVIDIA GPUs convert integer to float for division
  - High latency and low throughput instruction
- **Replace *j = it / N* by *j = it x (1/N)* and precompute *1/N***



```
18  for (int it = threadIdx.x; it < e_size; it += blockDim.x) {
19    j = it / N;
20    i = it - j * N;
21    k = j / N;
22    j -= k * N;
23    double wr = 0.0;
24    double ws = 0.0;
25    double wt = 0.0;
26    for (int n = 0; n < N; ++n) {
27      wr += dt[i * N + n] * ul[N * (j + k * N) + n];
```

Scope	MISC.CONVERT:Sum (I)	MISC.CONVERT:Sum (E)
174: nekbone(double*, double*, d	2.01e+05 100 %	2.01e+05 100 %
loop at cuda1.cu: 18	1.02e+05 51.0%	1.02e+05 51.0%
cuda1.cu: 27		
cuda1.cu: 32		
cuda1.cu: 28		
cuda1.cu: 29		
cuda1.cu: 19	1.02e+05 51.0%	1.02e+05 51.0%

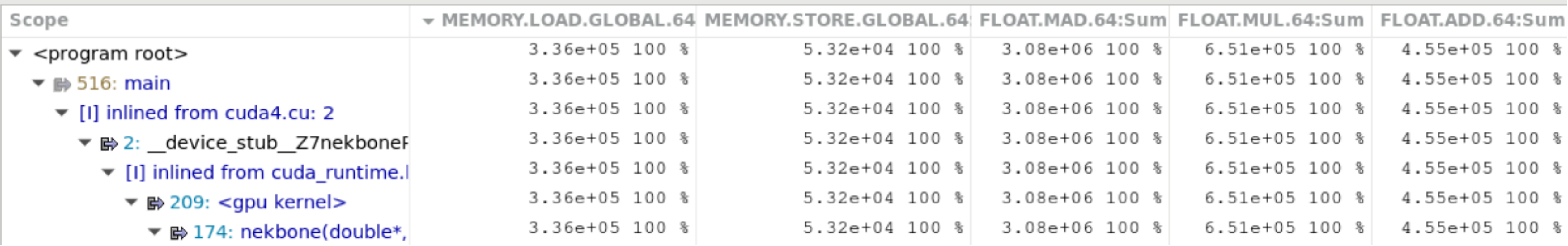
Coming Attraction: Instruction-level Analysis

Separate GPU instructions into classes

- **Memory operations**
  - instruction (load, store)
  - size
  - memory kind (global memory, texture memory, constant memory)
- **Floating point**
  - instruction (add, mul, mad)
  - size
  - compute unit (tensor unit, floating point unit)
- **Integer operations**
- **Control operations**
  - branches, calls

Performance insight 2: Instruction Throughput

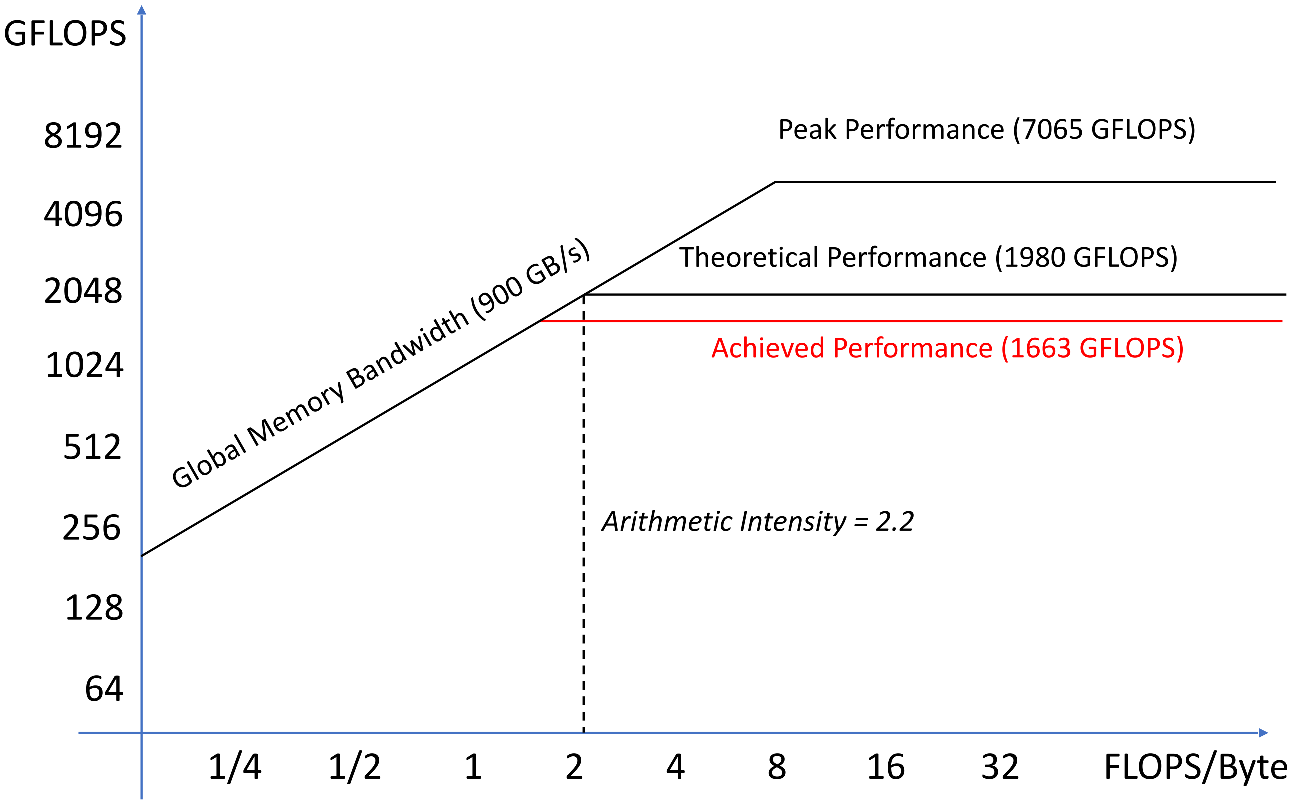
- **Estimate instruction throughput based on pc samples**
- $THROUGHPUT = \frac{INS}{TIME}$
- $GFLOPS = THROUGHPUT_{DP}$
- $Arithmetic\ Intensity = \frac{THROUGHPUT_{GMEM}}{THROUGHPUT_{DP}}$



Scope	▼ MEMORY.LOAD.GLOBAL.64	MEMORY.STORE.GLOBAL.64	FLOAT.MAD.64:Sum	FLOAT.MUL.64:Sum	FLOAT.ADD.64:Sum
▼ <program root>	3.36e+05 100 %	5.32e+04 100 %	3.08e+06 100 %	6.51e+05 100 %	4.55e+05 100 %
▼ 516: main	3.36e+05 100 %	5.32e+04 100 %	3.08e+06 100 %	6.51e+05 100 %	4.55e+05 100 %
▼ [I] inlined from cuda4.cu: 2	3.36e+05 100 %	5.32e+04 100 %	3.08e+06 100 %	6.51e+05 100 %	4.55e+05 100 %
▼ 2: __device_stub__Z7nekboneP	3.36e+05 100 %	5.32e+04 100 %	3.08e+06 100 %	6.51e+05 100 %	4.55e+05 100 %
▼ [I] inlined from cuda_runtime.	3.36e+05 100 %	5.32e+04 100 %	3.08e+06 100 %	6.51e+05 100 %	4.55e+05 100 %
▼ 209: <gpu kernel>	3.36e+05 100 %	5.32e+04 100 %	3.08e+06 100 %	6.51e+05 100 %	4.55e+05 100 %
▼ 174: nekbone(double*,	3.36e+05 100 %	5.32e+04 100 %	3.08e+06 100 %	6.51e+05 100 %	4.55e+05 100 %

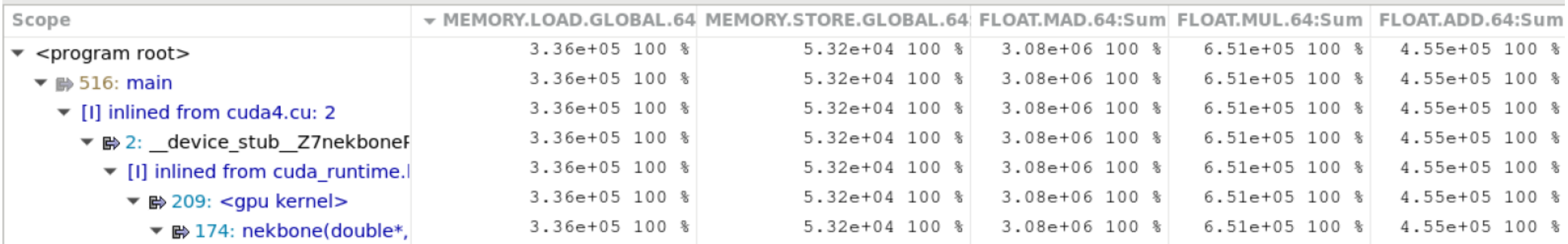
Roofline analysis

- **83.9% of peak performance**



Performance insight 3: unfused DMUL and DADD

- **DMUL:** 6.51×10^5
- **DADD:** 4.55×10^5
- **If all paired DMUL and DADD instructions are fused to MAD instructions**
  - $$\frac{\left(4.55 \times 10^5 + 3.08 \times 10^6\right)}{3.08 \times 10^6} = 14.7\%$$
  - 1663 GFLOPS × 114.7% = 1908 GFLOPS (99% of peak)



Scope	▾ MEMORY.LOAD.GLOBAL.64	MEMORY.STORE.GLOBAL.64:	FLOAT.MAD.64:Sum	FLOAT.MUL.64:Sum	FLOAT.ADD.64:Sum
▾ <program root>	3.36e+05 100 %	5.32e+04 100 %	3.08e+06 100 %	6.51e+05 100 %	4.55e+05 100 %
▾ 516: main	3.36e+05 100 %	5.32e+04 100 %	3.08e+06 100 %	6.51e+05 100 %	4.55e+05 100 %
▾ [I] inlined from cuda4.cu: 2	3.36e+05 100 %	5.32e+04 100 %	3.08e+06 100 %	6.51e+05 100 %	4.55e+05 100 %
▾ 2: __device_stub__Z7nekboneF	3.36e+05 100 %	5.32e+04 100 %	3.08e+06 100 %	6.51e+05 100 %	4.55e+05 100 %
▾ [I] inlined from cuda_runtime.I	3.36e+05 100 %	5.32e+04 100 %	3.08e+06 100 %	6.51e+05 100 %	4.55e+05 100 %
▾ 209: <gpu kernel>	3.36e+05 100 %	5.32e+04 100 %	3.08e+06 100 %	6.51e+05 100 %	4.55e+05 100 %
▾ 174: nekbone(double*,	3.36e+05 100 %	5.32e+04 100 %	3.08e+06 100 %	6.51e+05 100 %	4.55e+05 100 %

Case Study Acknowledgements

- **ORNL**
  - Ronnie Chatterjee
- **IBM**
  - Eric Liu
- **NERSC**
  - Christopher Daley
  - Jean Sexton
  - Kevin Gott

Outline

- **Performance measurement and analysis challenges for GPU-accelerated supercomputers**
- **Introduction to HPCToolkit performance tools**
  - Overview of HPCToolkit components and their workflow
  - HPCToolkit's graphical user interfaces and using them effectively
- **Analyzing the performance of GPU-accelerated supercomputers with HPCToolkit**
  - Overview of HPCToolkit's GPU performance measurement capabilities
  - Collecting measurements
  - Analysis and attribution
  - Exploring measurements and analysis results
- **Experiences with analysis and tuning of GPU-accelerated codes**
  - Computation, memory hierarchy, and data movement issues
- **Obtaining HPCToolkit**

Installing HPCToolkit for Analysis of GPU-accelerated Codes

- **Full instructions: http://hpctoolkit.org/software-instructions.html**
- **The short form**
  - Clone spack
    - command: `git clone https://github.com/spack/spack`
  - Configure a packages.yaml file
    - specify your platform's installation of CUDA or ROCM
    - specify your platform's installation of MPI
    - use an appropriate GCC compiler
      - ensure that a GCC version >= 5 is on your path. typically, we use GCC 7.3
      - `spack compiler find`
  - Install software for your platform using spack
    - NVIDIA GPUs: `spack install hpctoolkit@master +cuda +mpi`
    - AMD GPUs: `spack install hpctoolkit@master +rocm +mpi`