A Brief Introduction to HPCToolkit

Keren Zhou
Department of Computer Science
Rice University

http://hpctoolkit.org

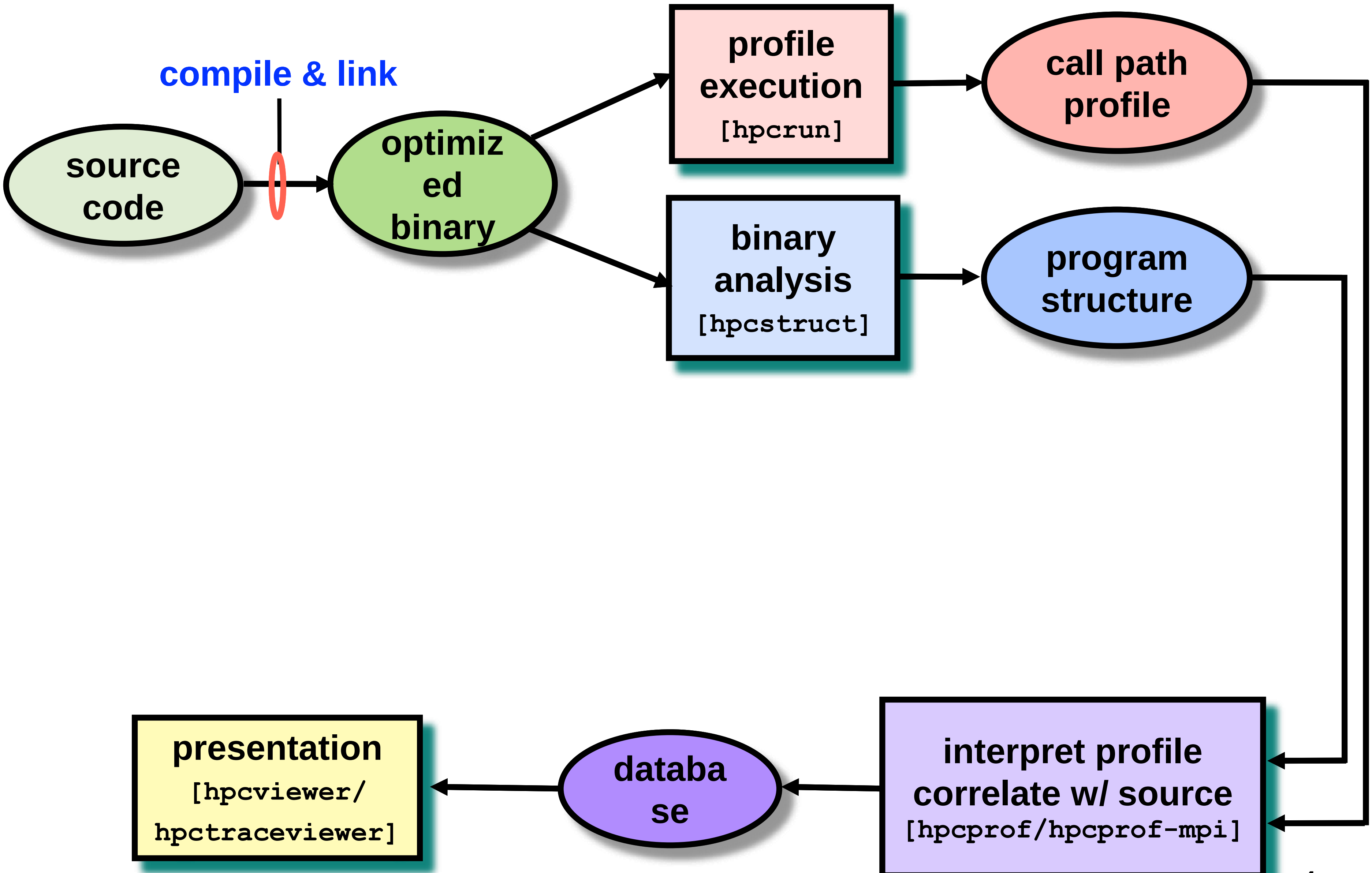
Outline

- **Overview of Rice's HPCToolkit**
- **OpenMP issues**
- **Using HPCToolkit's GUIs to analyze program performance**
- **Other capabilities**

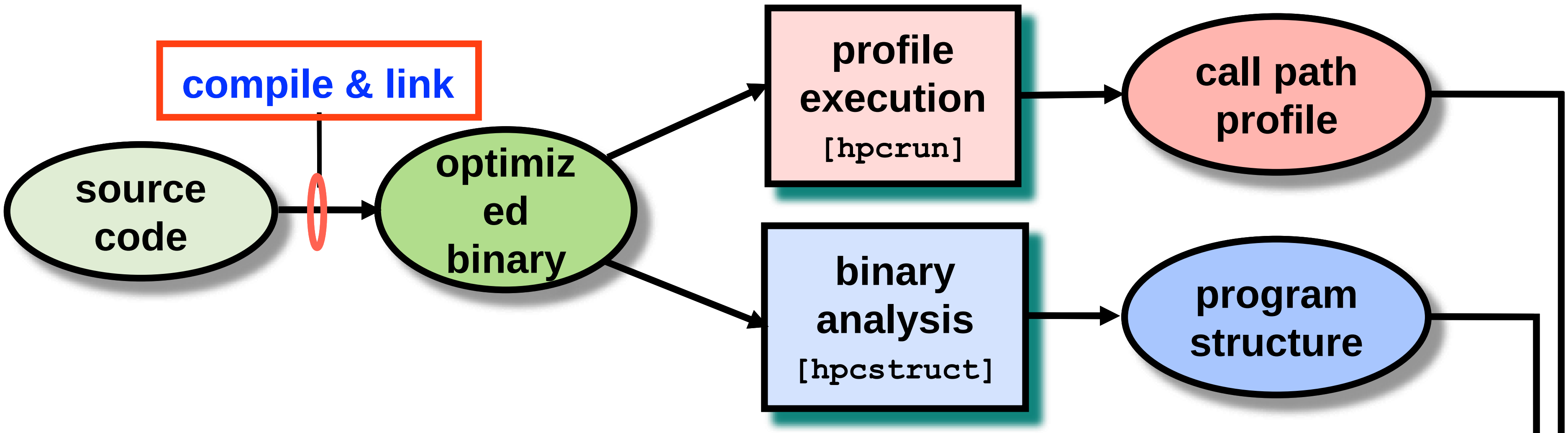
Rice University's HPCToolkit

- Employs binary-level measurement and analysis
  - observe fully optimized, dynamically linked executions
  - support multi-lingual codes with external binary-only libraries
- Uses sampling-based measurement (avoid instrumentation)
  - controllable overhead
  - minimize systematic error and avoid blind spots
  - enable data collection for large-scale parallelism
- Collects and correlates multiple derived performance metrics
  - diagnosis often requires more than one species of metric
- Associates metrics with both static and dynamic context
  - loop nests, procedures, inlined code, calling context
- Supports top-down performance analysis
  - identify costs of interest and drill down to causes
    - up and down call chains
    - over time

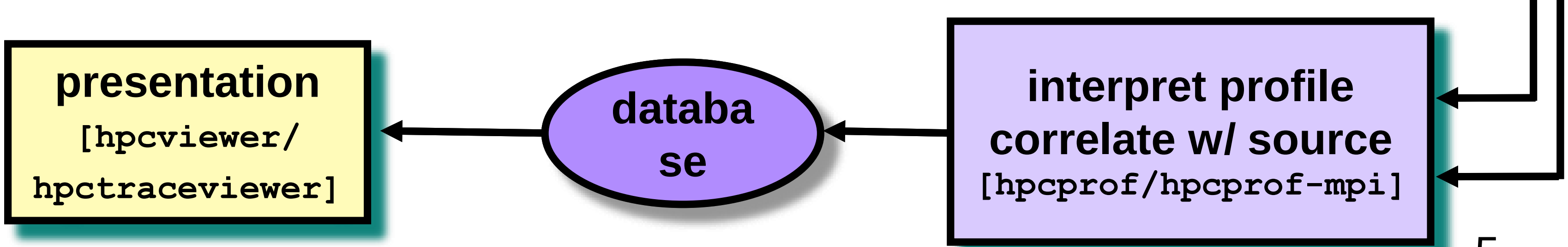
HPCToolkit Workflow



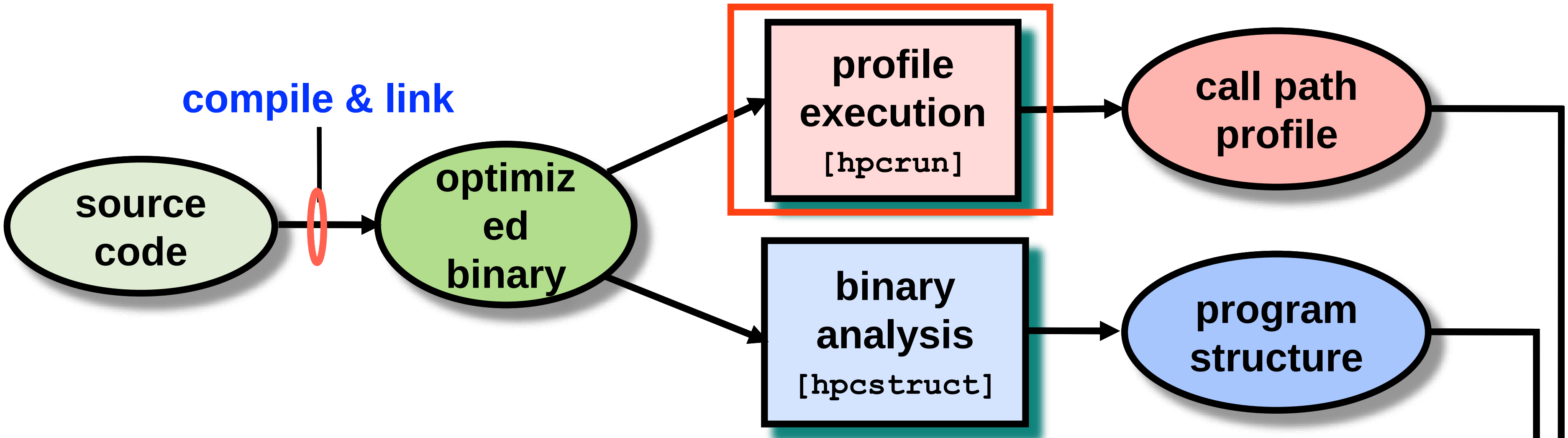
HPCToolkit Workflow



- For dynamically-linked executables, e.g., Linux clusters
  - compile and link as you usually do: nothing special needed
- For statically-linked executables (Cray default)
  - add monitoring by using `hpclink` as prefix to your link line

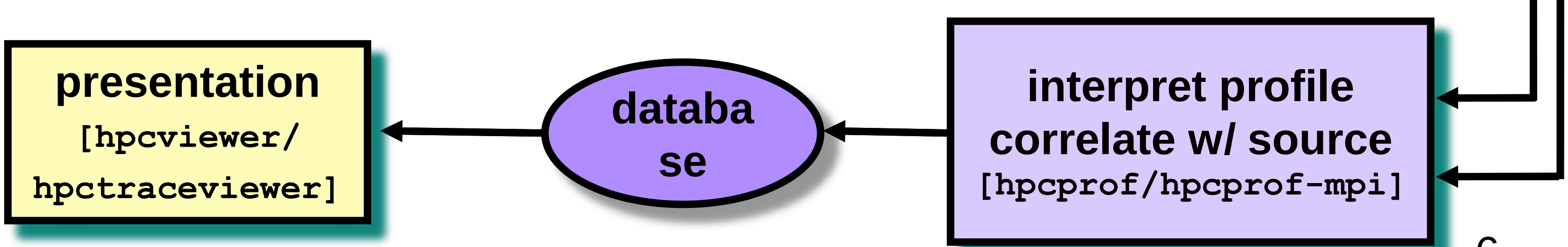


HPCToolkit Workflow



Measure execution unobtrusively

- launch optimized application binaries
  - dynamically-linked: launch with `hpcrun`, arguments control monitoring
  - statically-linked: environment variables control monitoring
- collect statistical call path profiles of events of interest

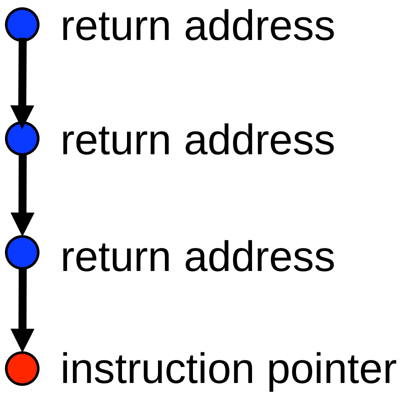


Call Path Profiling

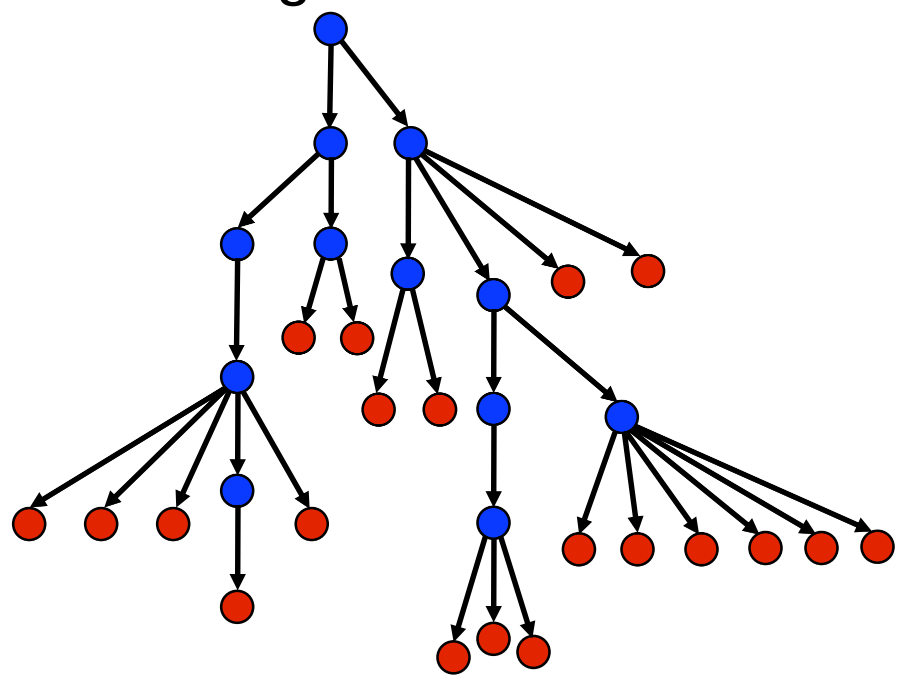
Measure and attribute costs in context

- **sample timer or hardware counter overflows**
- **gather calling context using stack unwinding**

Call path sample

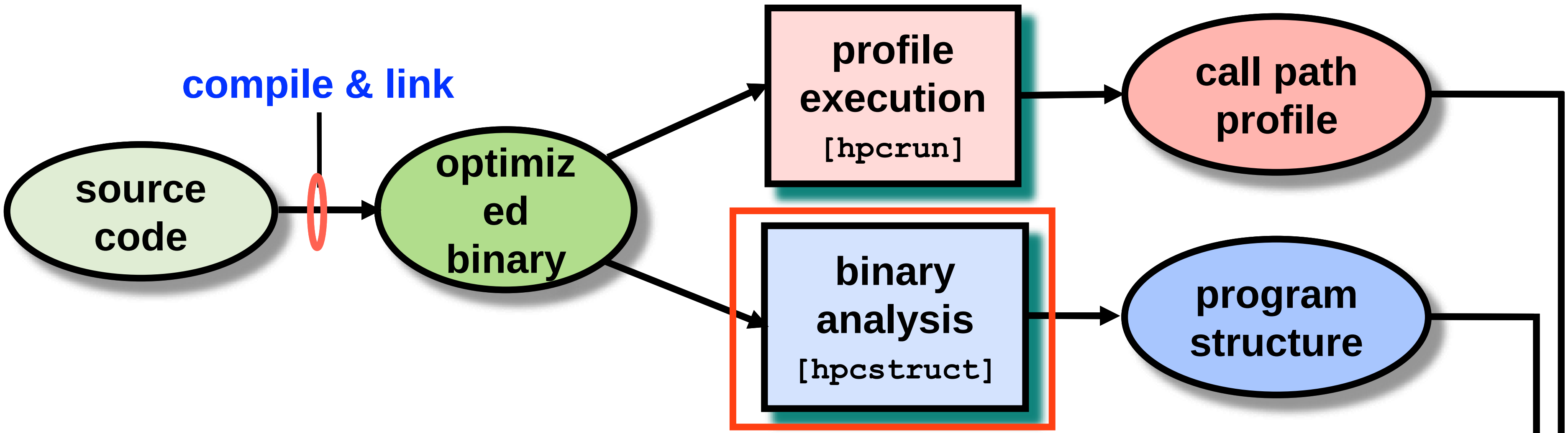


Calling context tree

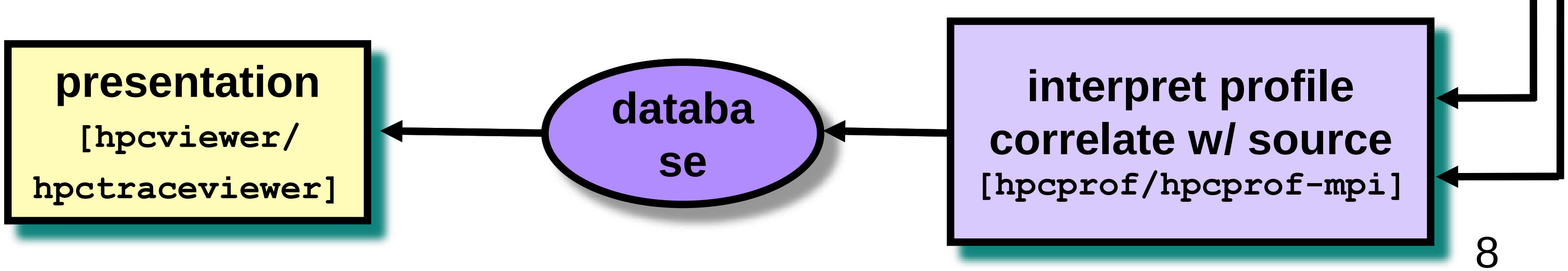


Overhead proportional to sampling frequency...
...not call frequency

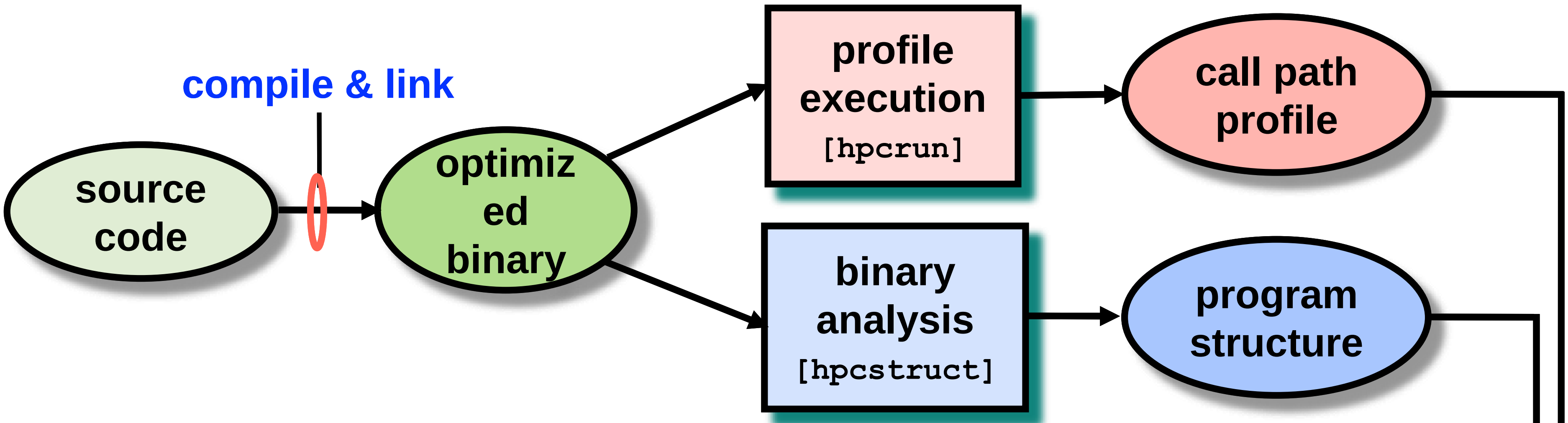
HPCToolkit Workflow



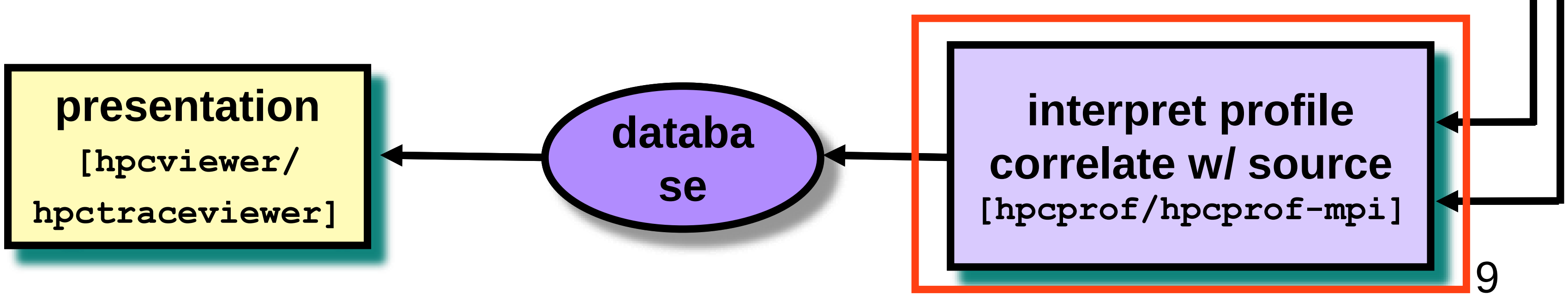
- Analyze binary with `hpcstruct`: recover program structure
  - analyze machine code, line map, debugging information
  - extract loop nests & identify inlined procedures
  - map transformed loops and procedures to source



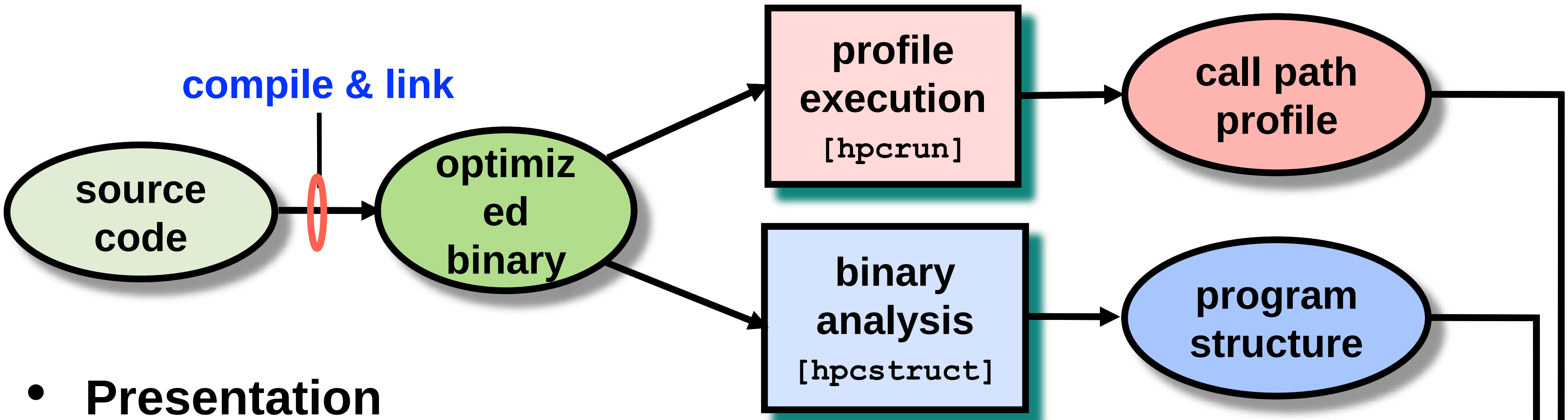
HPCToolkit Workflow



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

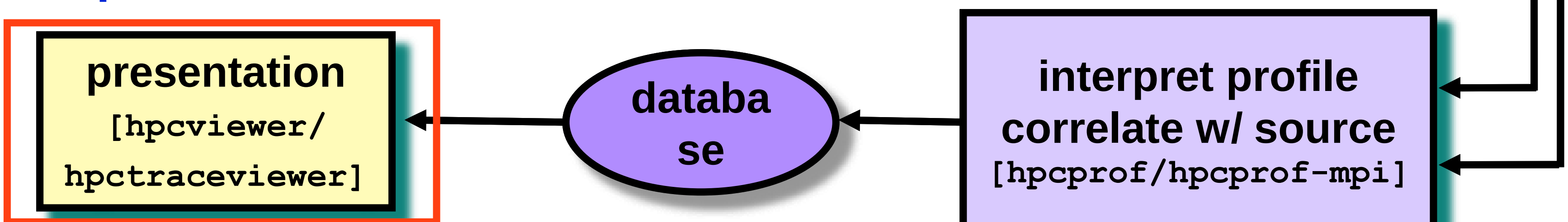


HPCToolkit Workflow

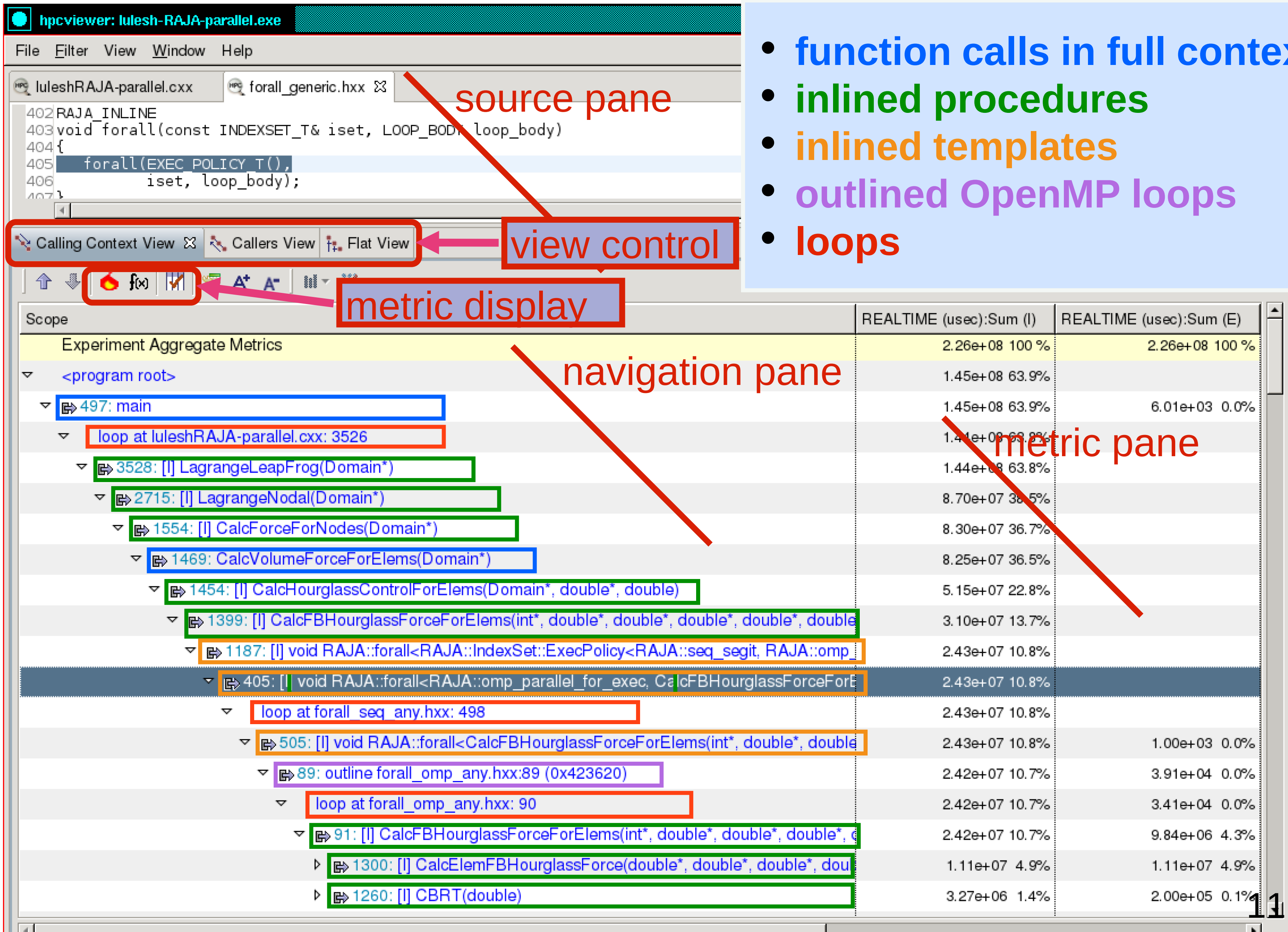


- **Presentation**
  - — explore performance data from multiple perspectives
    - – rank order by metrics to focus on what's important
    - – compute derived metrics to help gain insight

      e.g. scalability losses, waste, CPI, bandwidth
  - — graph thread-level metrics for contexts
  - — explore evolution of behavior over time



Code-centric Analysis with hpcviewer

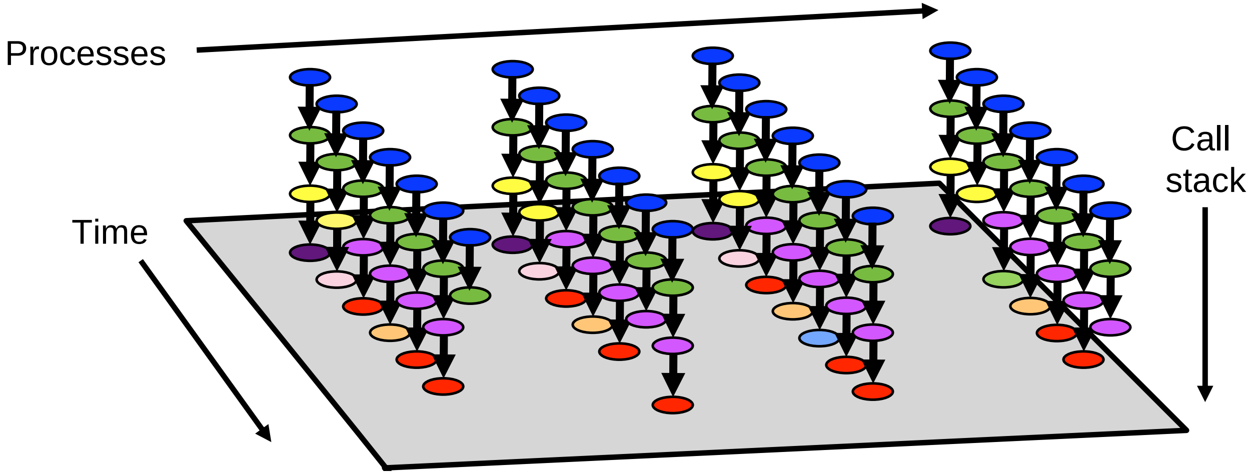


- function calls in full context
- inlined procedures
- inlined templates
- outlined OpenMP loops
- loops

Understanding Temporal Behavior

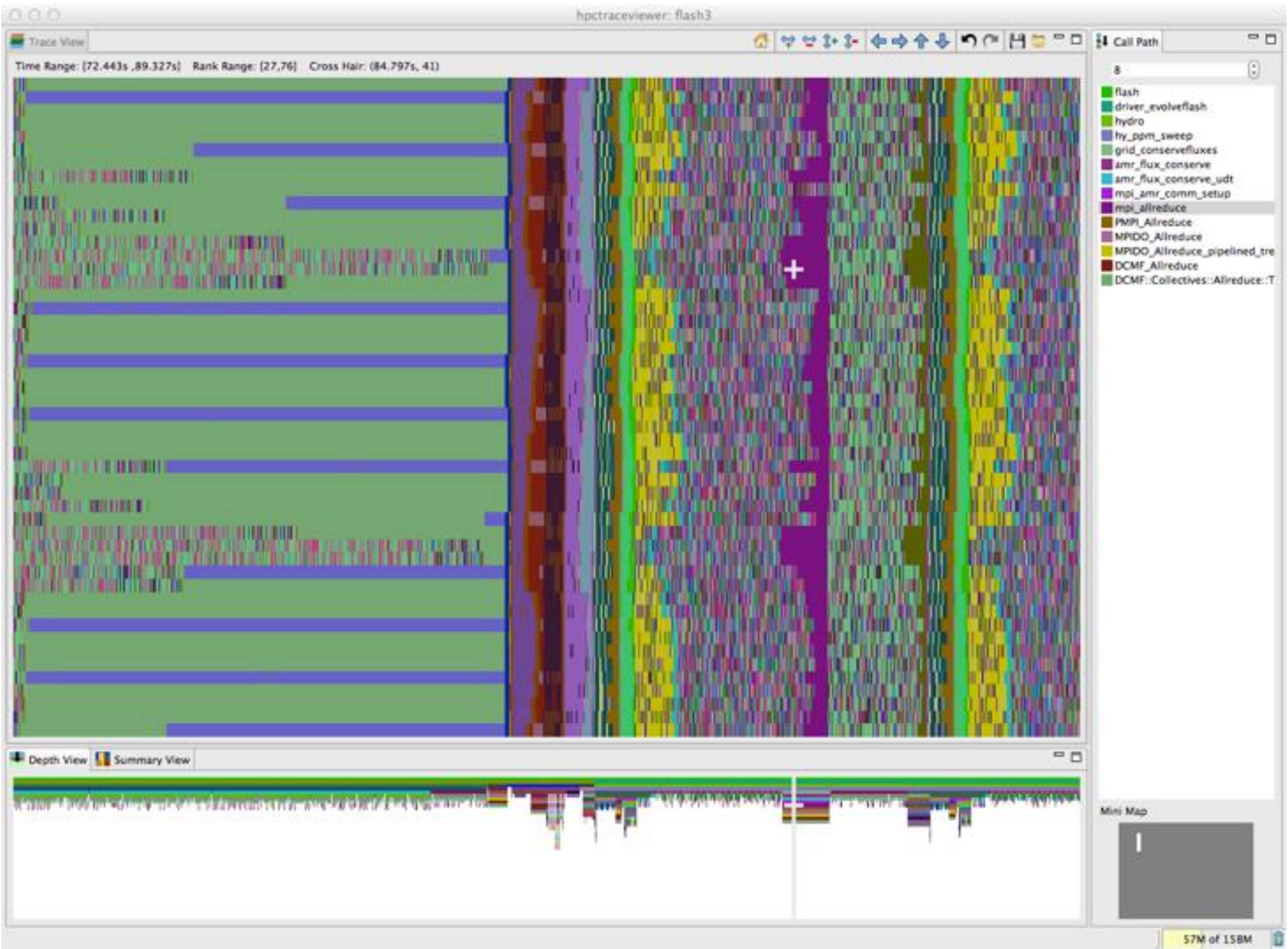
- **Profiling compresses out the temporal dimension**
  - **—temporal patterns, e.g. serialization, are invisible in profiles**
- **What can we do? Trace call path samples**
  - **—sketch:**
    - **N times per second, take a call path sample of each thread**
    - **organize the samples for each thread 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



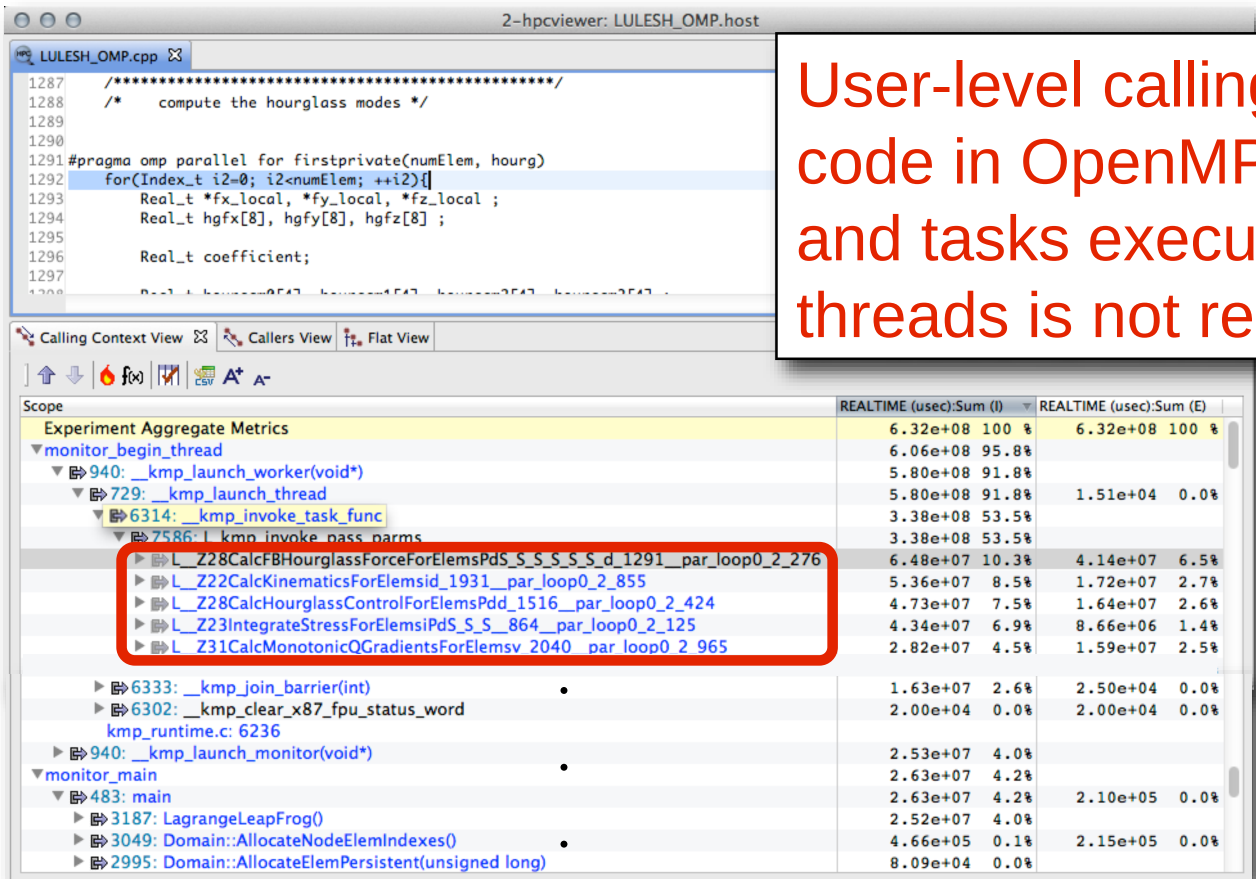
hpctraceviewer: detail of FLASH@256PE

Time-centric analysis: load imbalance among threads appears as different lengths of colored bands along the x axis



OpenMP: A Challenge for Tools

- **Large gap between between threaded programming models and their implementations**



User-level calling context for code in OpenMP parallel regions and tasks executed by worker threads is not readily available

- **Runtime support is necessary for tools to bridge the gap**

Challenges for OpenMP Node Programs

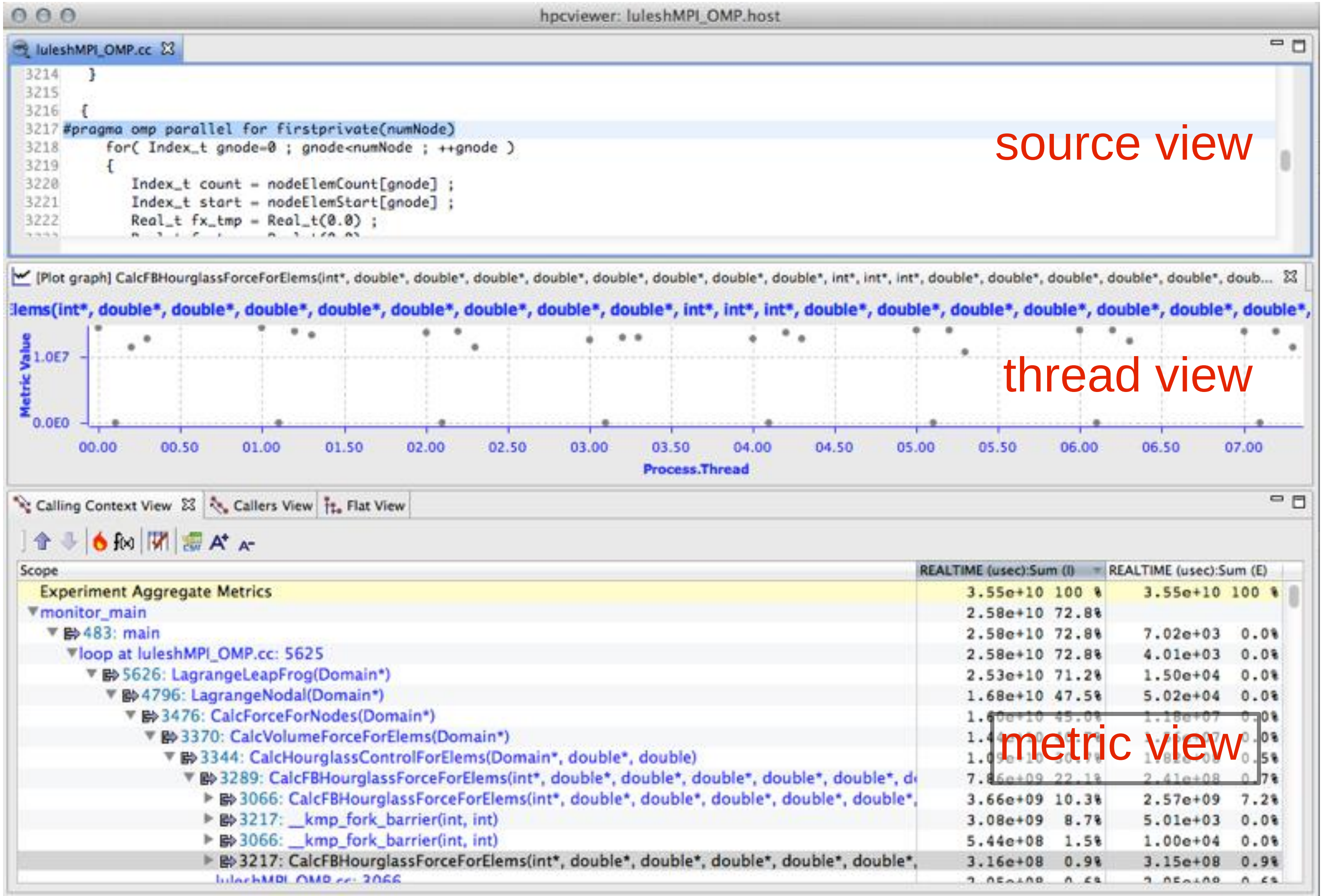
- **Tools provide implementation-level view of OpenMP threads**
  - asymmetric threads
    - master thread
    - worker thread
  - run-time frames are interspersed with user code
- **Hard to understand causes of idleness**
  - long serial sections
  - load imbalance in parallel regions
  - waiting for critical sections or locks

OMPT: An OpenMP Tools API

- **Goal: a standardized tool interface for OpenMP**
  - — prerequisite for portable tools
  - — missing piece of the OpenMP language standard
- **Design objectives**
  - — enable tools to measure and attribute costs to application source and runtime system
    - support low-overhead tools based on asynchronous sampling
    - attribute to user-level calling contexts
    - associate a thread’s activity at any point with a descriptive state
  - — minimize overhead if OMPT interface is not in use
    - features that may increase overhead are optional
  - — define interface for trace-based performance tools
  - — don’t impose an unreasonable development burden
    - runtime implementers
    - tool developers

Integrated View of MPI+OpenMP with OMPT

LLNL’s luleshMPI_OMP (8 MPI x 3 OMP), 30, REALTIME@1000

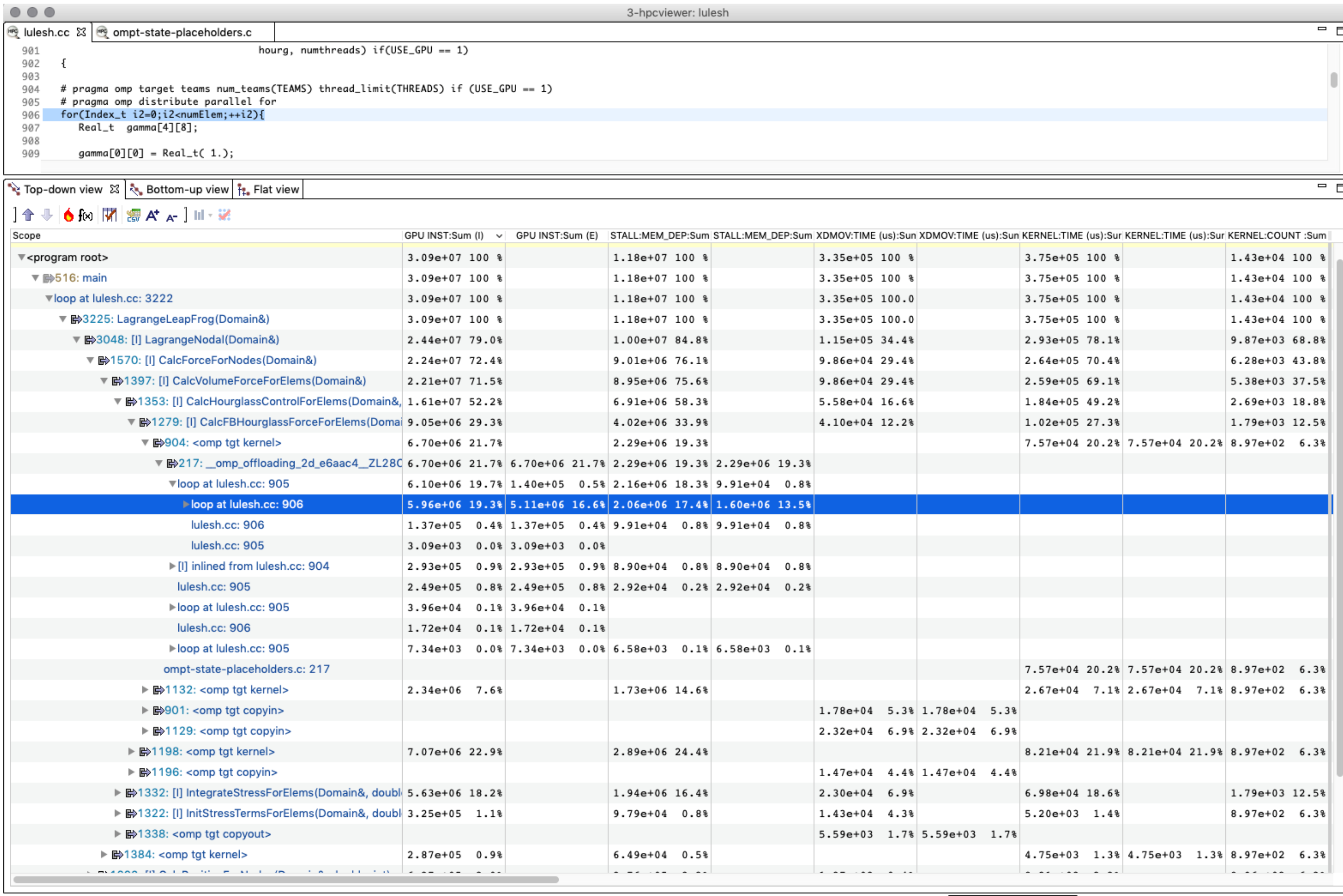


OpenMP Tool API Status

- HPCToolkit supports OpenMP 5.0 OMPT
- OMPT prototype implementations
  - LLVM (emerging: OpenMP 5.0)
    - interoperable with GNU, Intel compilers
  - IBM LOMP (currently targets OpenMP 4.5)
- Ongoing work
  - refining OpenMP 5.0 OMPT support in LLVM OpenMP
  - refining OpenMP 5.0 OMPT support in HPCToolkit
    - asynchronous call stack assembly for lightweight monitoring

HPCToolkit Capabilities for GPU Code

MPI + OpenMP 4.5 or CUDA GPU accelerated applications



3-hpcviewer: lulesh

lulesh.cc | ompt-state-placeholders.c

```
901                                hourg, numthreads) if(USE_GPU == 1)
902    {
903
904    # pragma omp target teams num_teams(TEAMS) thread_limit(THREADS) if (USE_GPU == 1)
905    # pragma omp distribute parallel for
906    for(Index_t i2=0;i2<numElem;++i2){
907       Real_t  gamma[4][8];
908
909       gamma[0][0] = Real_t( 1.);
```

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

Scope	GPU INST:Sum (I)	GPU INST:Sum (E)	STALL:MEM_DEP:Sum	STALL:MEM_DEP:Sum	XDMOV:TIME (us):Sun	XDMOV:TIME (us):Sun	KERNEL:TIME (us):Sur	KERNEL:TIME (us):Sur	KERNEL:COUNT :Sum
<program root>	3.09e+07 100 %		1.18e+07 100 %		3.35e+05 100 %		3.75e+05 100 %		1.43e+04 100 %
516: main	3.09e+07 100 %		1.18e+07 100 %		3.35e+05 100 %		3.75e+05 100 %		1.43e+04 100 %
loop at lulesh.cc: 3222	3.09e+07 100 %		1.18e+07 100 %		3.35e+05 100.0		3.75e+05 100 %		1.43e+04 100 %
3225: LagrangeLeapFrog(Domain&)	3.09e+07 100 %		1.18e+07 100 %		3.35e+05 100.0		3.75e+05 100 %		1.43e+04 100 %
3048: [I] LagrangeNodal(Domain&)	2.44e+07 79.0%		1.00e+07 84.8%		1.15e+05 34.4%		2.93e+05 78.1%		9.87e+03 68.8%
1570: [I] CalcForceForNodes(Domain&)	2.24e+07 72.4%		9.01e+06 76.1%		9.86e+04 29.4%		2.64e+05 70.4%		6.28e+03 43.8%
1397: [I] CalcVolumeForceForElems(Domain&)	2.21e+07 71.5%		8.95e+06 75.6%		9.86e+04 29.4%		2.59e+05 69.1%		5.38e+03 37.5%
1353: [I] CalcHourglassControlForElems(Domain&,	1.61e+07 52.2%		6.91e+06 58.3%		5.58e+04 16.6%		1.84e+05 49.2%		2.69e+03 18.8%
1279: [I] CalcFBHourglassForceForElems(Domai	9.05e+06 29.3%		4.02e+06 33.9%		4.10e+04 12.2%		1.02e+05 27.3%		1.79e+03 12.5%
904: <omp tgt kernel>	6.70e+06 21.7%		2.29e+06 19.3%				7.57e+04 20.2%	7.57e+04 20.2%	8.97e+02 6.3%
217: __omp_offloading_2d_e6aac4__ZL28C	6.70e+06 21.7%	6.70e+06 21.7%	2.29e+06 19.3%	2.29e+06 19.3%					
loop at lulesh.cc: 905	6.10e+06 19.7%	1.40e+05 0.5%	2.16e+06 18.3%	9.91e+04 0.8%					
loop at lulesh.cc: 906	5.96e+06 19.3%	5.11e+06 16.6%	2.06e+06 17.4%	1.60e+06 13.5%					
lulesh.cc: 906	1.37e+05 0.4%	1.37e+05 0.4%	9.91e+04 0.8%	9.91e+04 0.8%					
lulesh.cc: 905	3.09e+03 0.0%	3.09e+03 0.0%							
[I] inlined from lulesh.cc: 904	2.93e+05 0.9%	2.93e+05 0.9%	8.90e+04 0.8%	8.90e+04 0.8%					
lulesh.cc: 905	2.49e+05 0.8%	2.49e+05 0.8%	2.92e+04 0.2%	2.92e+04 0.2%					
loop at lulesh.cc: 905	3.96e+04 0.1%	3.96e+04 0.1%							
lulesh.cc: 906	1.72e+04 0.1%	1.72e+04 0.1%							
loop at lulesh.cc: 905	7.34e+03 0.0%	7.34e+03 0.0%	6.58e+03 0.1%	6.58e+03 0.1%					
ompt-state-placeholders.c: 217							7.57e+04 20.2%	7.57e+04 20.2%	8.97e+02 6.3%
1132: <omp tgt kernel>	2.34e+06 7.6%		1.73e+06 14.6%				2.67e+04 7.1%	2.67e+04 7.1%	8.97e+02 6.3%
901: <omp tgt copyin>					1.78e+04 5.3%	1.78e+04 5.3%			
1129: <omp tgt copyin>					2.32e+04 6.9%	2.32e+04 6.9%			
1198: <omp tgt kernel>	7.07e+06 22.9%		2.89e+06 24.4%				8.21e+04 21.9%	8.21e+04 21.9%	8.97e+02 6.3%
1196: <omp tgt copyin>					1.47e+04 4.4%	1.47e+04 4.4%			
1332: [I] IntegrateStressForElems(Domain&, doubl	5.63e+06 18.2%		1.94e+06 16.4%		2.30e+04 6.9%		6.98e+04 18.6%		1.79e+03 12.5%
1322: [I] InitStressTermsForElems(Domain&, doubl	3.25e+05 1.1%		9.79e+04 0.8%		1.43e+04 4.3%		5.20e+03 1.4%		8.97e+02 6.3%
1338: <omp tgt copyout>					5.59e+03 1.7%	5.59e+03 1.7%			
1384: <omp tgt kernel>	2.87e+05 0.9%		6.49e+04 0.5%				4.75e+03 1.3%	4.75e+03 1.3%	8.97e+02 6.3%

Other Capabilities

- **Measure hardware counters using Linux perf_events**
  - —available events can be listed with
    - – hpcrun -L
    - – launching a binary created by hpclink with environment setting
      - HPCRUN_EVENT_LIST=LIST
  - —frequency based sampling: 300/s per thread or machine max
    - – no need to set periods or frequencies unless you want precise control
  - —hardware event multiplexing
    - – measure more events than hardware counters
- **Kernel sampling**
  - —measure activity in the Linux kernel in addition to your program
    - – e.g., allocating and clearing memory pages
  - —not available on BG/Q
  - —measurement and attribution subject to system permissions
    - – detailed attribution not available on NERSC or ANL systems

HPCToolkit at NERSC

- **NERSC cori**
  - a setup script or a set of module loads
  - source /global/cscratch1/sd/kz21/env-static.sh
  - source /global/cscratch1/sd/kz21/env-shared.sh
- **Man pages**
  - automatically added to MANPATH by the aforementioned command

HPCToolkit at ORNL

- **On Summit**
  - **module use /gpfs/alpine/csc322/world-shared/modulefiles**
  - **module load hpctoolkit**
- **Man pages**
  - **automatically added to MANPATH by the aforementioned command**

GUIs for your Laptop

- **Download binary packages for HPCToolkit's user interfaces on your laptop**
  - http://hpctoolkit.org/download/hpcviewer

Detailed HPCToolkit Documentation

http://hpctoolkit.org/documentation.html

- **Comprehensive user manual:**
  http://hpctoolkit.org/manual/HPCToolkit-users-manual.pdf
  - **Quick start guide**
    - **essential overview that almost fits on one page**
  - **Using HPCToolkit with statically linked programs**
    - **a guide for using hpctoolkit on BG/Q and Cray platforms**
  - **The hpcviewer and hpctraceviewer user interfaces**
  - **Effective strategies for analyzing program performance with HPCToolkit**
    - **analyzing scalability, waste, multicore performance ...**
  - **HPCToolkit and MPI**
  - **HPCToolkit Troubleshooting**
    - **why don’t I have any source code in the viewer?**
    - **hpcviewer isn’t working well over the network ... what can I do?**
- **Installation guide**

Advice for Using HPCToolkit

Using HPCToolkit

- **Add hpctoolkit's bin directory to your path using softenv**
- **Adjust your compiler flags (if you want <u>full</u> attribution to src)**
  - — **add -g flag after any optimization flags**
- **Add hpclink as a prefix to your Makefile's link line**
  - — **e.g.** `hpclink CC -o myapp foo.o ... lib.a -lm ...`
- **See what sampling triggers are available on Cray**
  - — **use hpclink to link your executable**
  - — **launch executable with environment variable HPCRUN_EVENT_LIST=LIST**
    - – **you can launch this on 1 core of 1 node**
    - – **no need to provide arguments or input files for your program**

      they will be ignored

Monitoring Large Executions

- **Collecting performance data on every node is typically not necessary**
- **Can improve scalability of data collection by recording data for only a fraction of processes**
  - **set environment variable HPCRUN_PROCESS_FRACTION**
  - **e.g. collect data for 10% of your processes**
    - **set environment variable HPCRUN_PROCESS_FRACTION=0.10**

Digesting your Performance Data

- **Use hpcstruct to reconstruct program structure**
  - **e.g. `hpcstruct your_app`**
    - **creates your_app.hpcstruct**
- **Correlate measurements to source code with hpcprof and hpcprof-mpi**
  - **run hpcprof on the front-end to analyze data from small runs**
  - **run hpcprof-mpi on the compute nodes to analyze data from lots of nodes/threads in parallel**
    - **notes**
      - **much faster to do this on an x86_64 vis cluster (cooley) than on BG/Q**
      - **avoid expensive per-thread profiles with --metric-db no**
- **Digesting performance data in parallel with hpcprof-mpi**
  - **qsub -A ... -t 20 -n 32 --mode c1 --proccount 32 --cwd \`pwd\` **
    **/projects/Tools/hpctoolkit/pkgs-vesta/hpctoolkit/bin/hpcprof-mpi **
    **-S your_app.hpcstruct **
    **-I /path/to/your_app/src/+ **
    hpctoolkit-your_app-measurements.jobid

Analysis and Visualization

- **Use hpcviewer to open resulting database**
  - warning: first time you graph any data, it will pause to combine info from all threads into one file
- **Use hpctraceviewer to explore traces**
  - warning: first time you open a trace database, the viewer will pause to combine info from all threads into one file
- **Try our our user interfaces before collecting your own data**
  - example performance data
    http://hpctoolkit.org/examples.html

Installing HPCToolkit GUIs on your Laptop

- **See http://hpctoolkit.org/download/hpcviewer**
- **Download the latest for your laptop (Linux, Mac, Windows)**
  - **hpctraceviewer**
  - **hpcviewer**

A Note for Mac Users

When installing HPCToolkit GUIs on your Mac laptop, don’t simply download and double click on the zip file and have Finder unpack them. Follow the Terminal-based installation directions on the website to avoid interference by Mac Security.