Gaining Insight into Parallel Program Performance using HPCToolkit

John Mellor-Crummey
Department of Computer Science
Rice University

http://hpctoolkit.org

Acknowledgments

- **Current funding**
  - Argonne National Laboratory Subcontract 4F-30241
  - DOE Office of Science ASCR X-Stack "PIPER" Award
  - Intel
- **Project team**
  - Research Staff
    - Laksono Adhianto, Mike Fagan, Mark Krentel
  - Students
    - Milind Chabbi, Karthik Murthy
  - Recent Alumni
    - Xu Liu (William and Mary, 2014)
    - Nathan Tallent (PNNL, 2010)

Challenges for Computational Scientists

- Rapidly evolving platforms and applications
  - architecture
    - rapidly changing multicore microprocessor designs
    - increasing architectural diversity
      - multicore, manycore, accelerators
    - increasing scale of parallel systems
  - applications
    - transition from MPI everywhere to threaded implementations
    - enhance vector parallelism
    - augment computational capabilities
- Computational scientists needs
  - adapt to changes in emerging architectures
  - improve scalability within and across nodes
  - assess weaknesses in algorithms and their implementations

Performance tools can play an important role as a guide

Performance Analysis Challenges

- **Complex node architectures are hard to use efficiently**
  - multi-level parallelism: multiple cores, ILP, SIMD, accelerators
  - multi-level memory hierarchy
  - result: gap between typical and peak performance is huge
- **Complex applications present challenges**
  - measurement and analysis
  - understanding behaviors and tuning performance
- **Supercomputer platforms compound the complexity**
  - unique hardware & microkernel-based operating systems
  - multifaceted performance concerns
    - computation
    - data movement
    - communication
    - I/O

What Users Want

- **Multi-platform, programming model independent tools**
- **Accurate measurement of complex parallel codes**
  - large, multi-lingual programs
  - (heterogeneous) parallelism within and across nodes
  - optimized code: loop optimization, templates, inlining
  - binary-only libraries, sometimes partially stripped
  - complex execution environments
    - dynamic binaries on clusters; static binaries on supercomputers
    - batch jobs
- **Effective performance analysis**
  - insightful analysis that pinpoints and explains problems
    - correlate measurements with code for actionable results
    - support analysis at the desired level
      - intuitive enough for application scientists and engineers
      - detailed enough for library developers and compiler writers
- **Scalable to petascale and beyond**

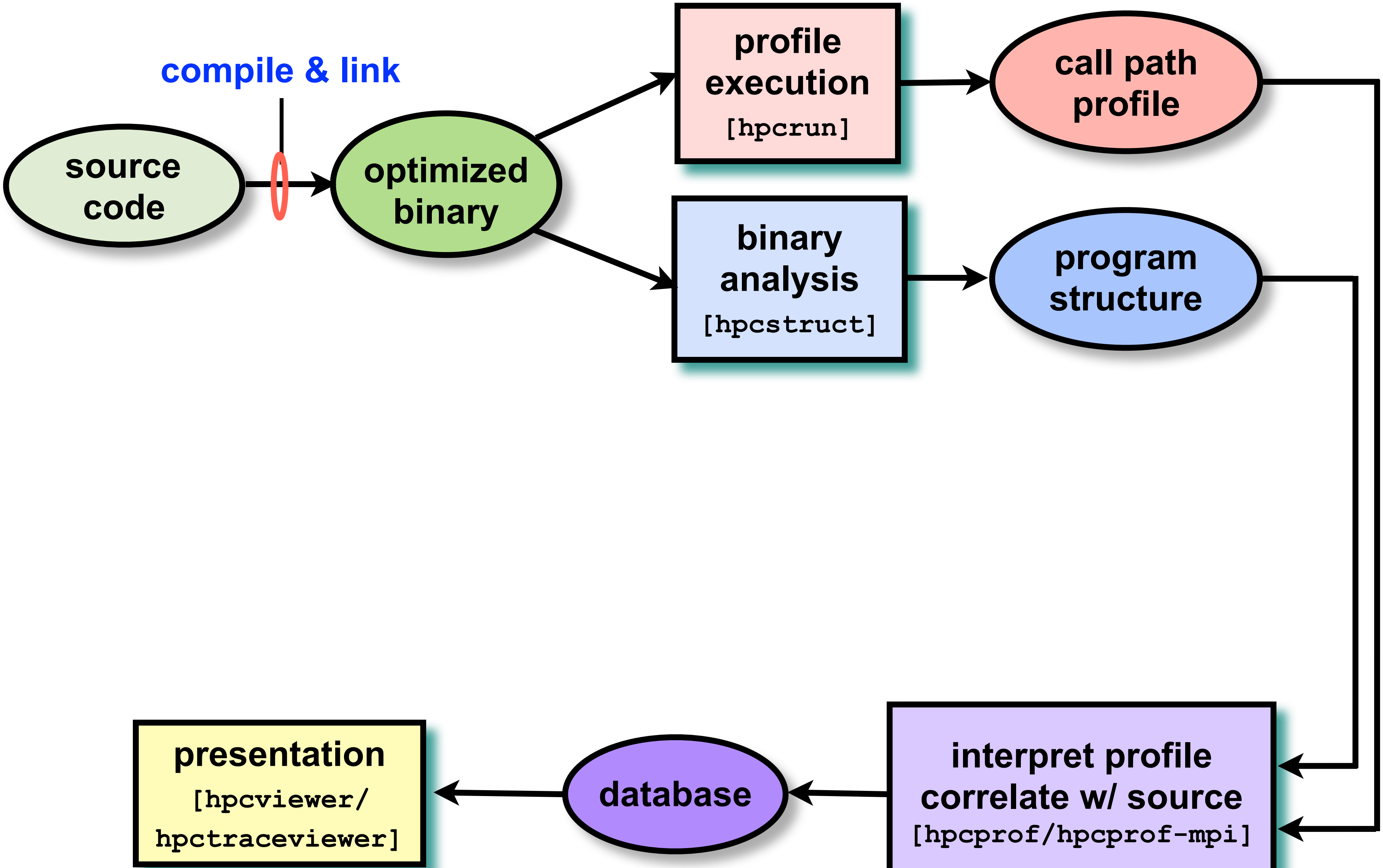
Outline

- **Overview of Rice's HPCToolkit**
- **Pinpointing scalability bottlenecks**
  - — scalability bottlenecks on large-scale parallel systems
  - — scaling on multicore processors
- **Understanding temporal behavior**
- **Assessing process variability**
- **Understanding threading performance**
  - — blame shifting
- **Today and the future**

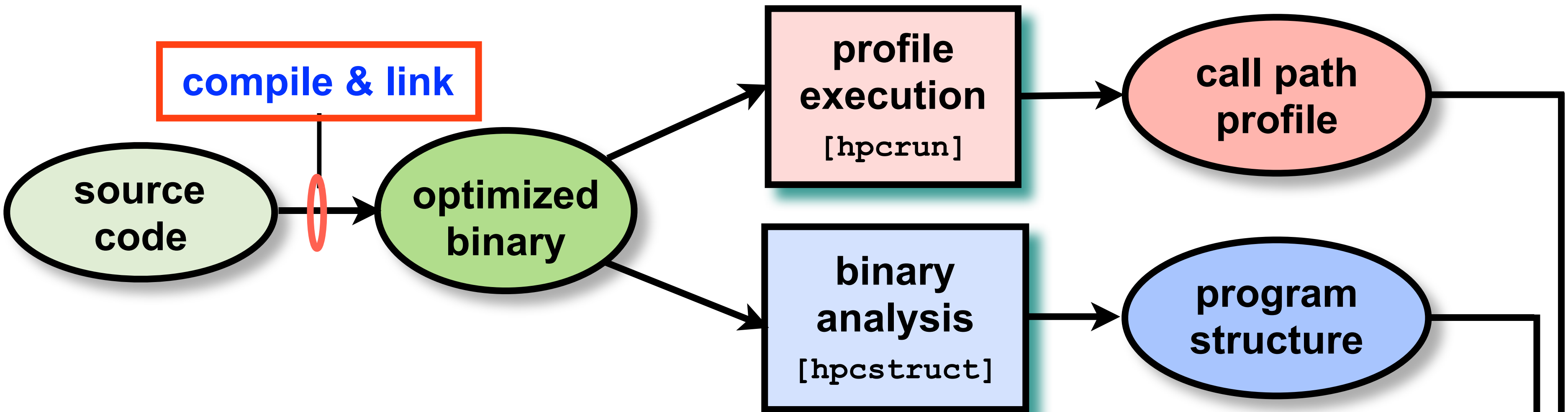
Rice University's HPCToolkit

- 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 typically 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

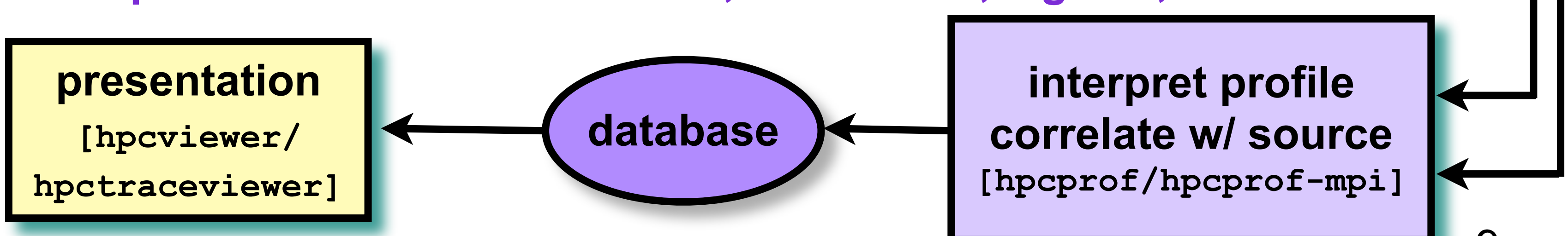
HPCToolkit Workflow



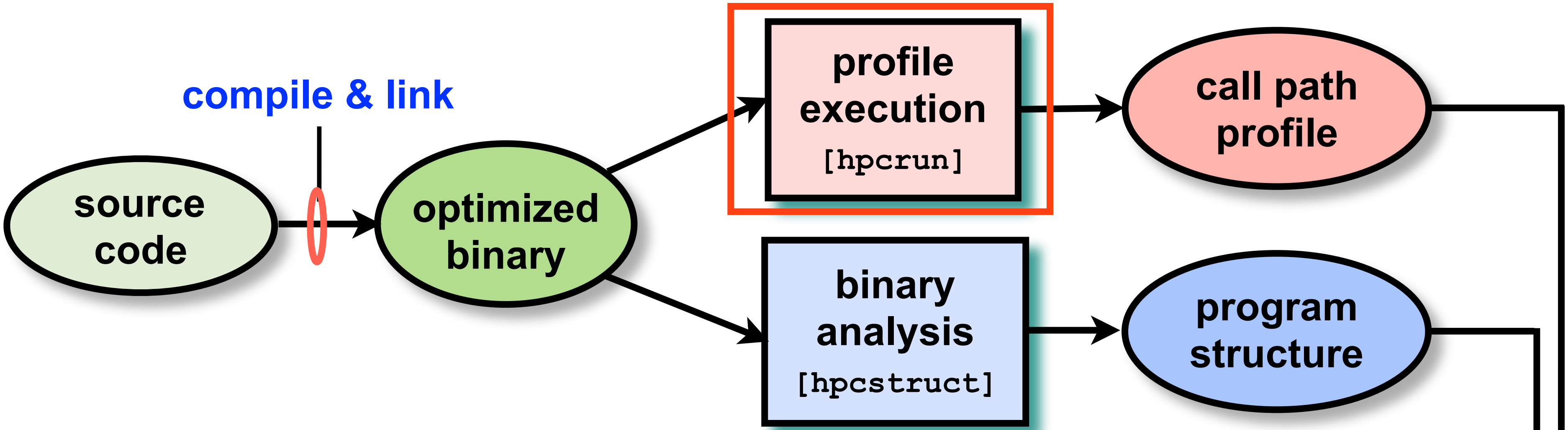
HPCToolkit Workflow



- For dynamically-linked executables, e.g., Linux
  - compile and link as you usually do: nothing special needed
- For statically-linked executables, e.g., Blue Gene/Q
  - add monitoring by using `hpclink` as prefix to your link line
    - uses "linker wrapping" to catch "control" operations
      process and thread creation, finalization, signals, ...

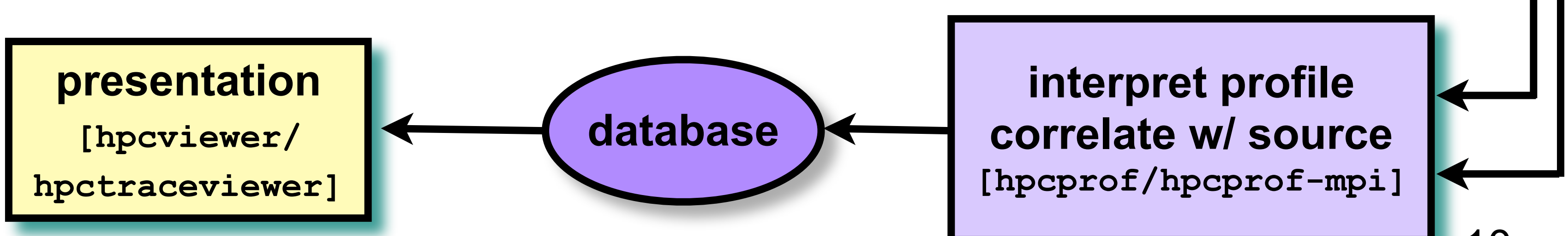


HPCToolkit Workflow



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

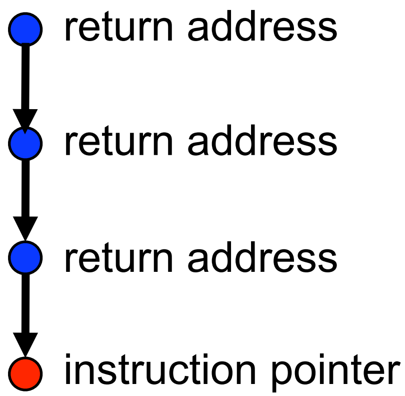


Call Path Profiling

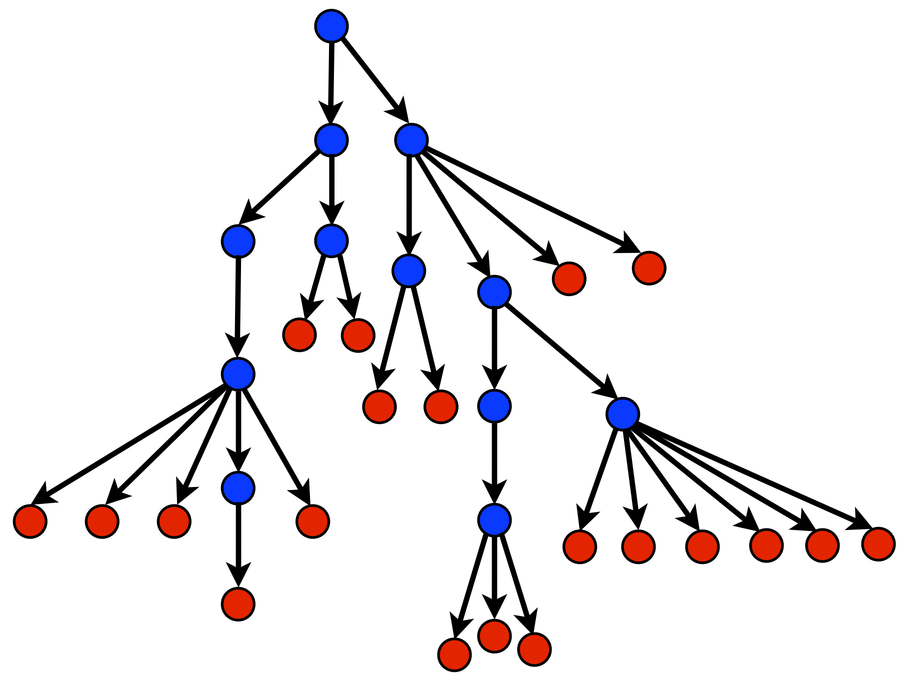
Measure and attribute costs in context

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

Call path sample

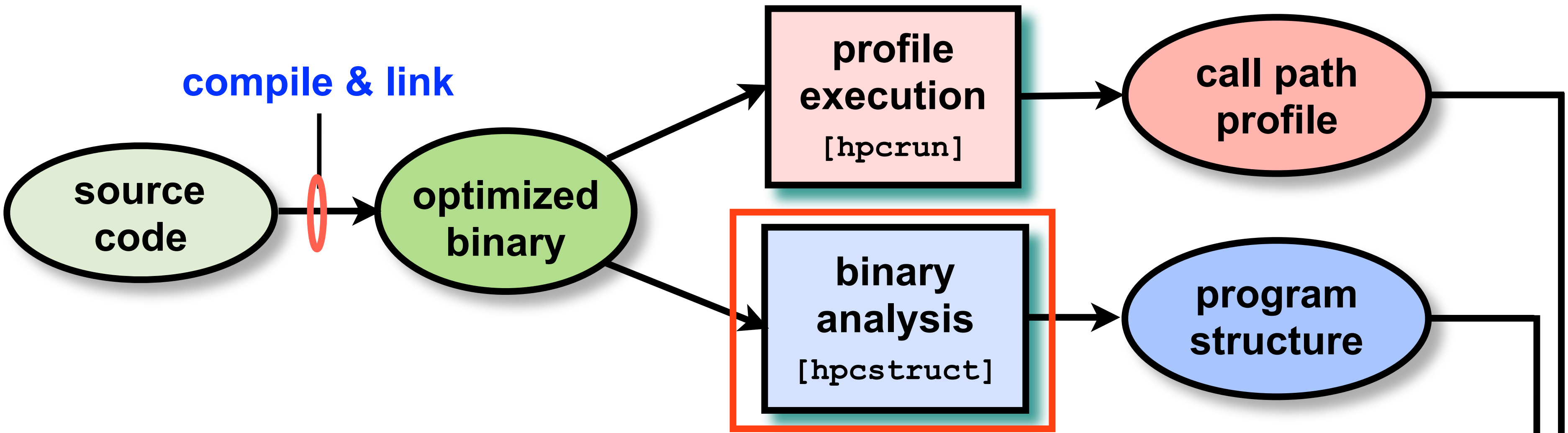


Calling context tree

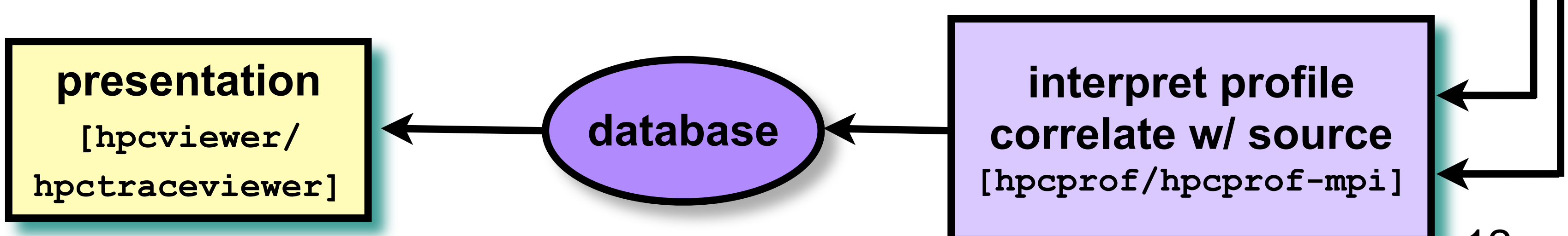


Overhead proportional to sampling frequency...
...not call frequency

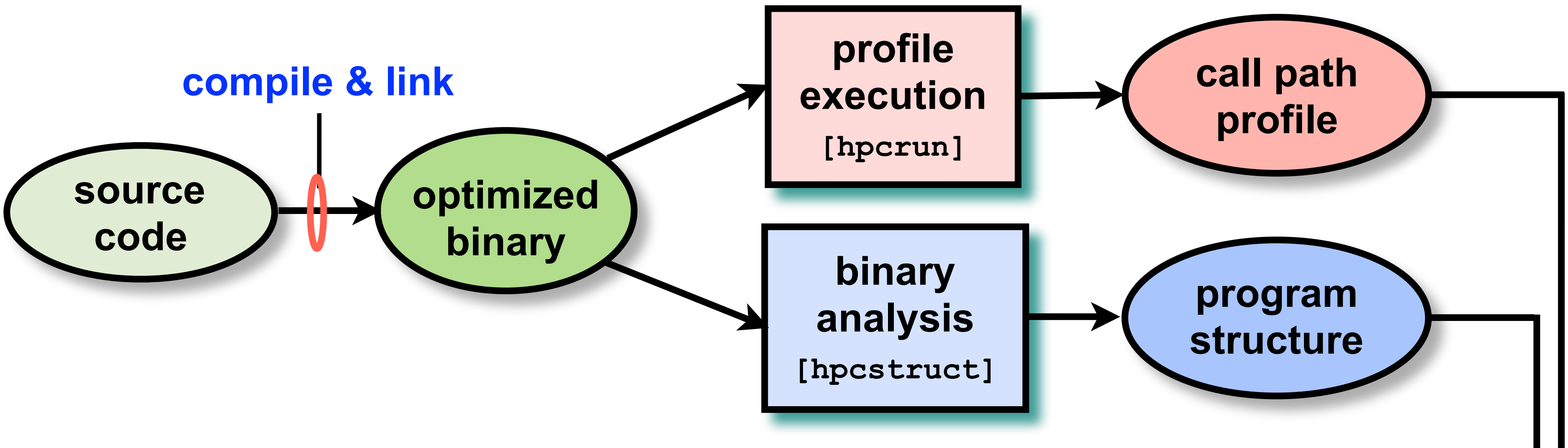
HPCToolkit Workflow



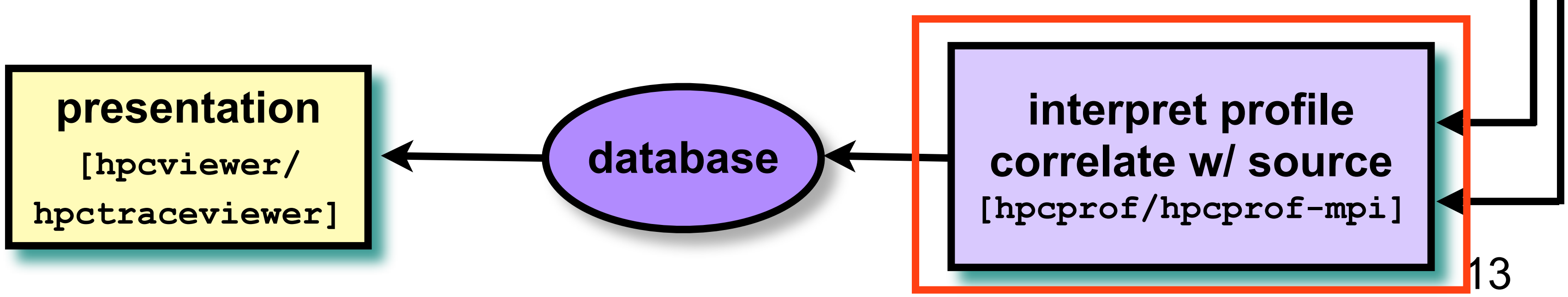
- Analyze binary with `hpcstruct`: recover program structure
  - — analyze machine code, line map, debugging information
  - — extract loop nesting & identify inlined procedures
  - — map transformed loops and procedures to source



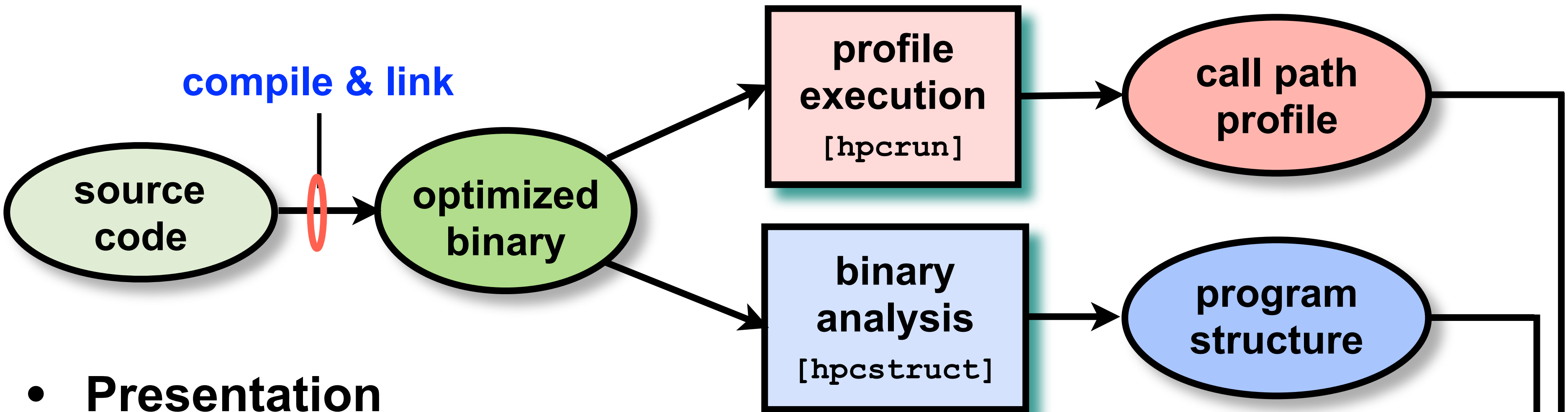
HPCToolkit Workflow



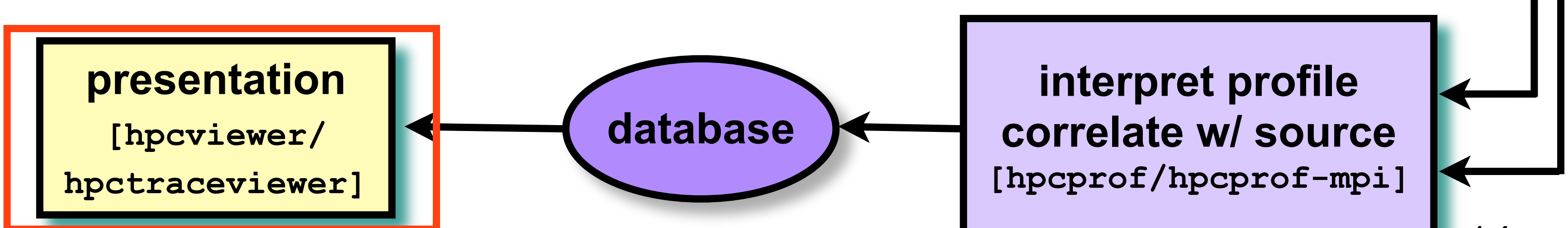
- Combine multiple profiles
  — multiple threads; multiple processes; multiple executions
- Correlate metrics to static & dynamic program structure



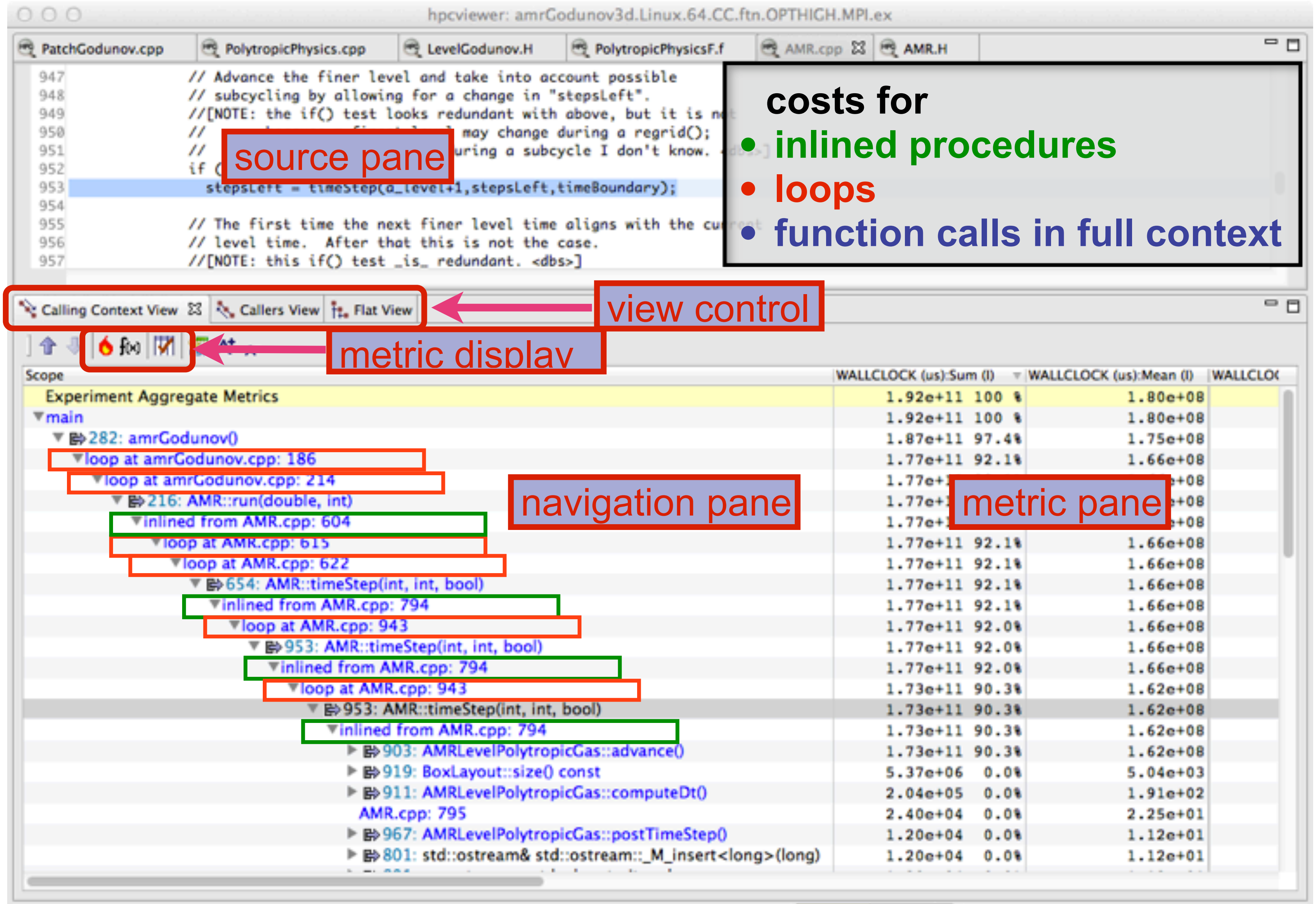
HPCToolkit Workflow



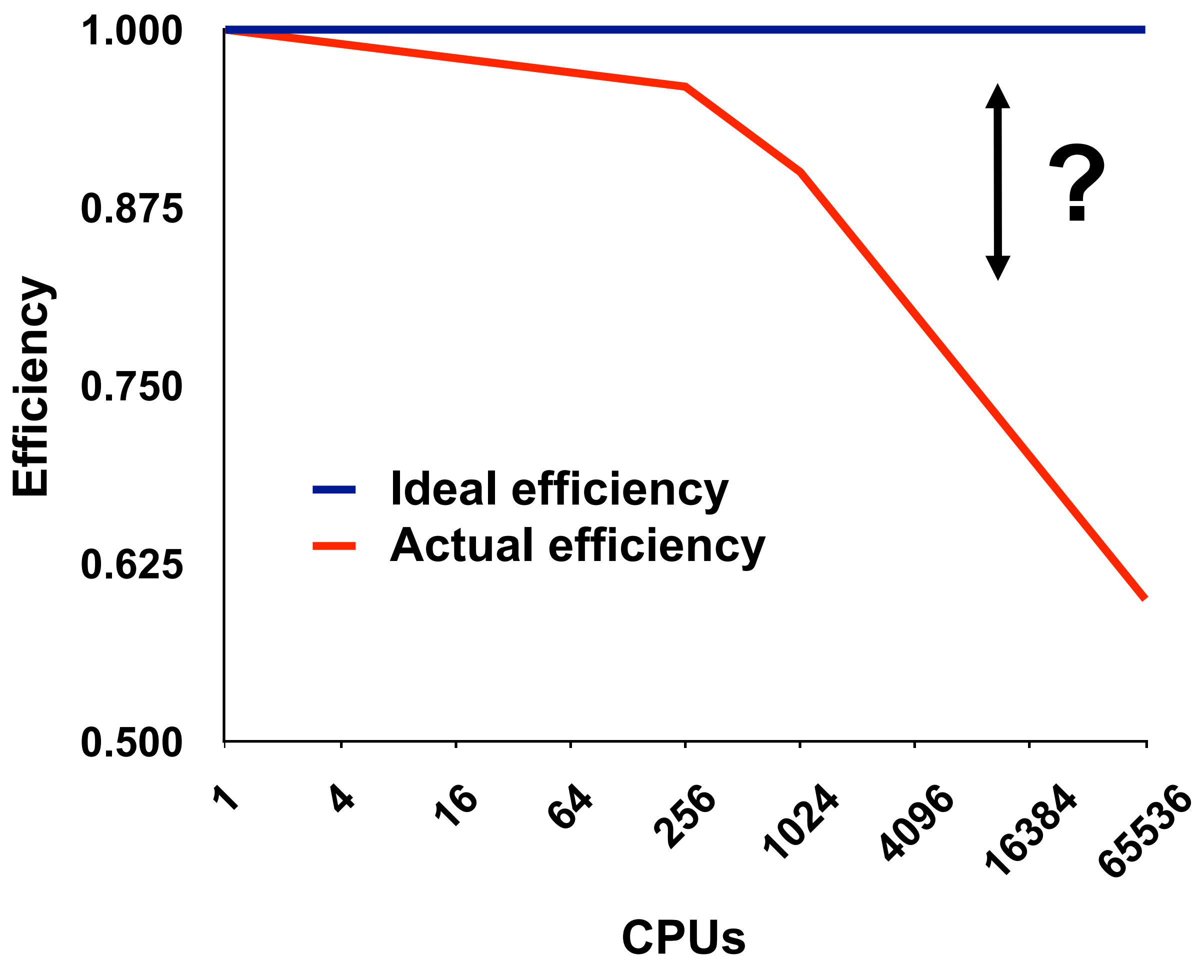
- **Presentation**
  - **explore performance data from multiple perspectives**
    - **rank order by metrics to focus on what's important**
    - **compute derived metrics to help gain insight**
      - **e.g. scalability losses, waste, CPI, bandwidth**
  - **graph thread-level metrics for contexts**
  - **explore evolution of behavior over time**



Code-centric Analysis with hpcviewer



The Problem of Scaling



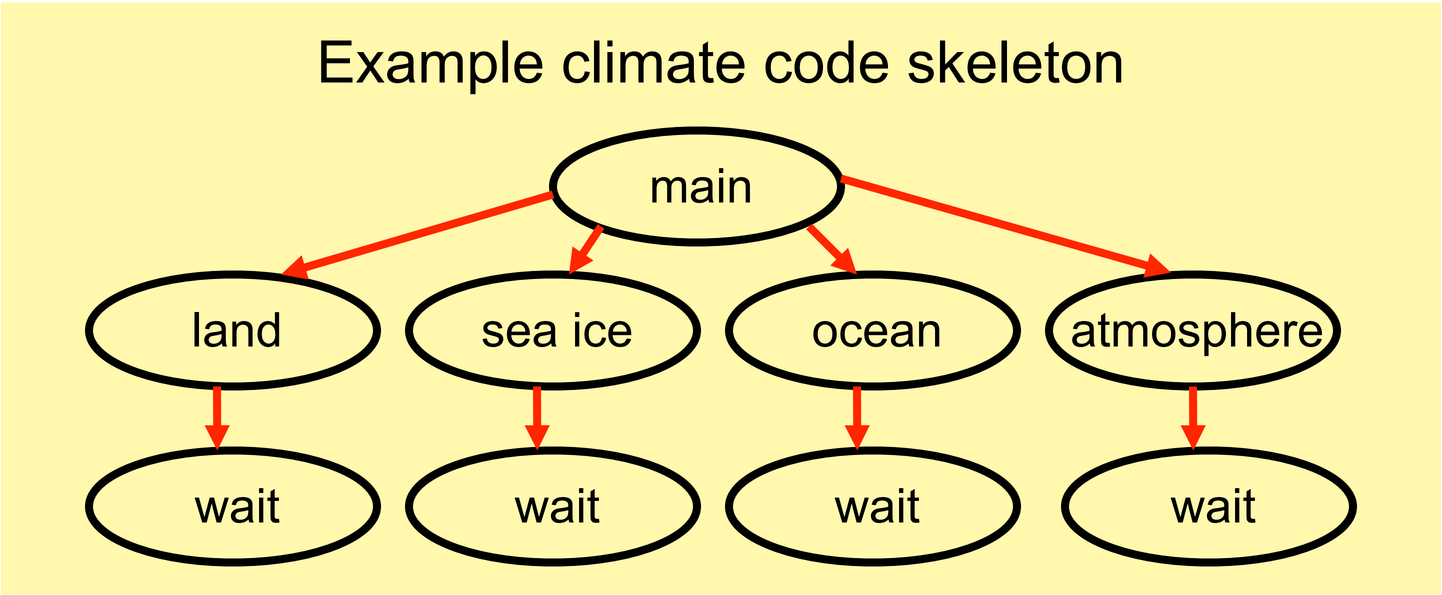
Note: higher is better

Goal: Automatic Scaling Analysis

- Pinpoint scalability bottlenecks
- Guide user to problems
- Quantify the magnitude of each problem
- Diagnose the nature of the problem

Challenges for Pinpointing Scalability Bottlenecks

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



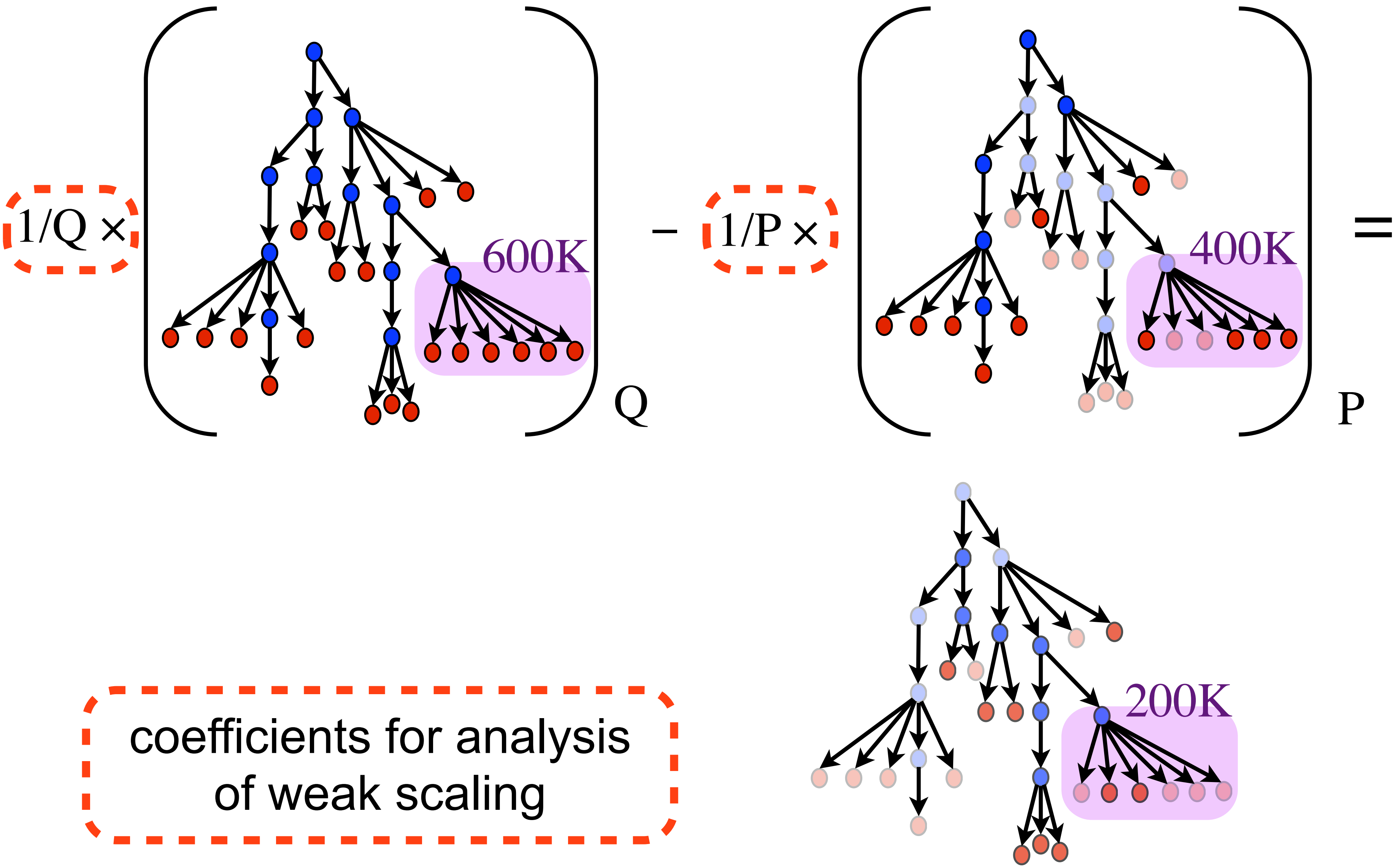
- **Monitoring**
  - — **bottleneck nature: computation, data movement, synchronization?**
  - — **2 pragmatic constraints**
    - – **acceptable data volume**
    - – **low perturbation for use in production runs**

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 or different inputs
  - 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

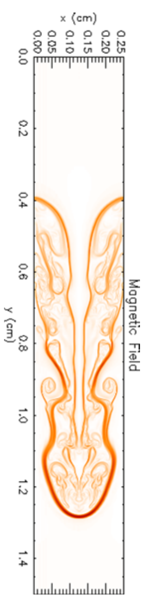
Pinpointing and Quantifying Scalability Bottlenecks



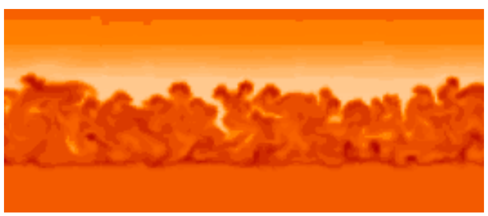
coefficients for analysis of weak scaling

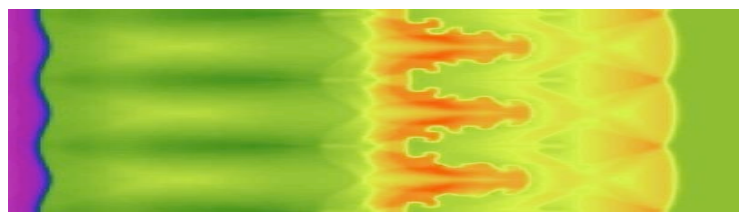
Scalability Analysis Demo

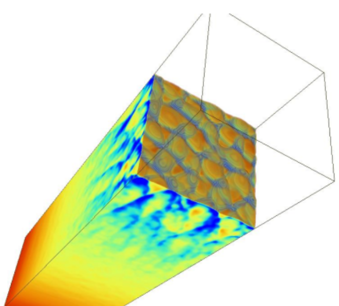
Code:	**University of Chicago FLASH**
Simulation:	**white dwarf detonation**
Platform:	**Blue Gene/P**
Experiment:	**8192 vs. 256 processors**
Scaling type:	**weak**

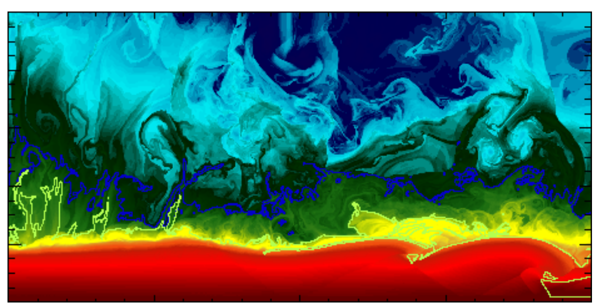


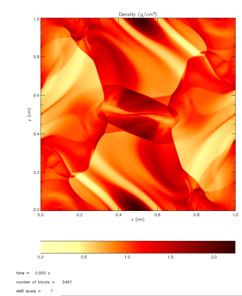
Magnetic Rayleigh-Taylor

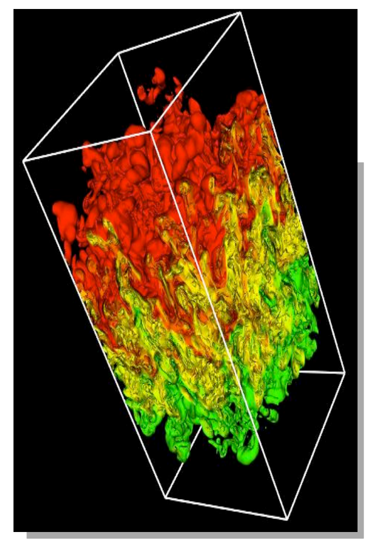

Nova outbursts on white dwarfs


Laser-driven shock instabilities


Cellular detonation

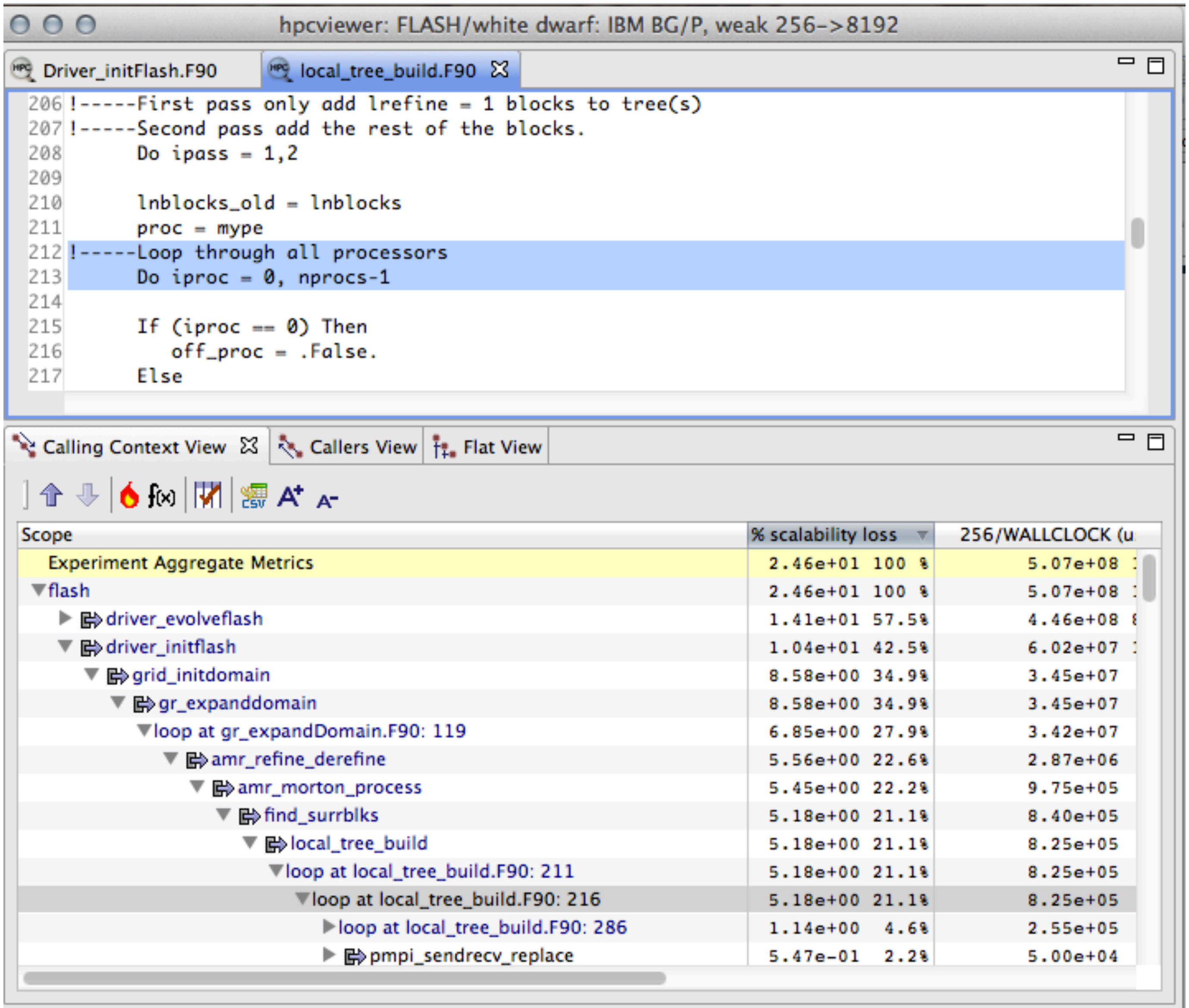

Helium burning on neutron stars


Orzag/Tang MHD vortex


Rayleigh-Taylor instability

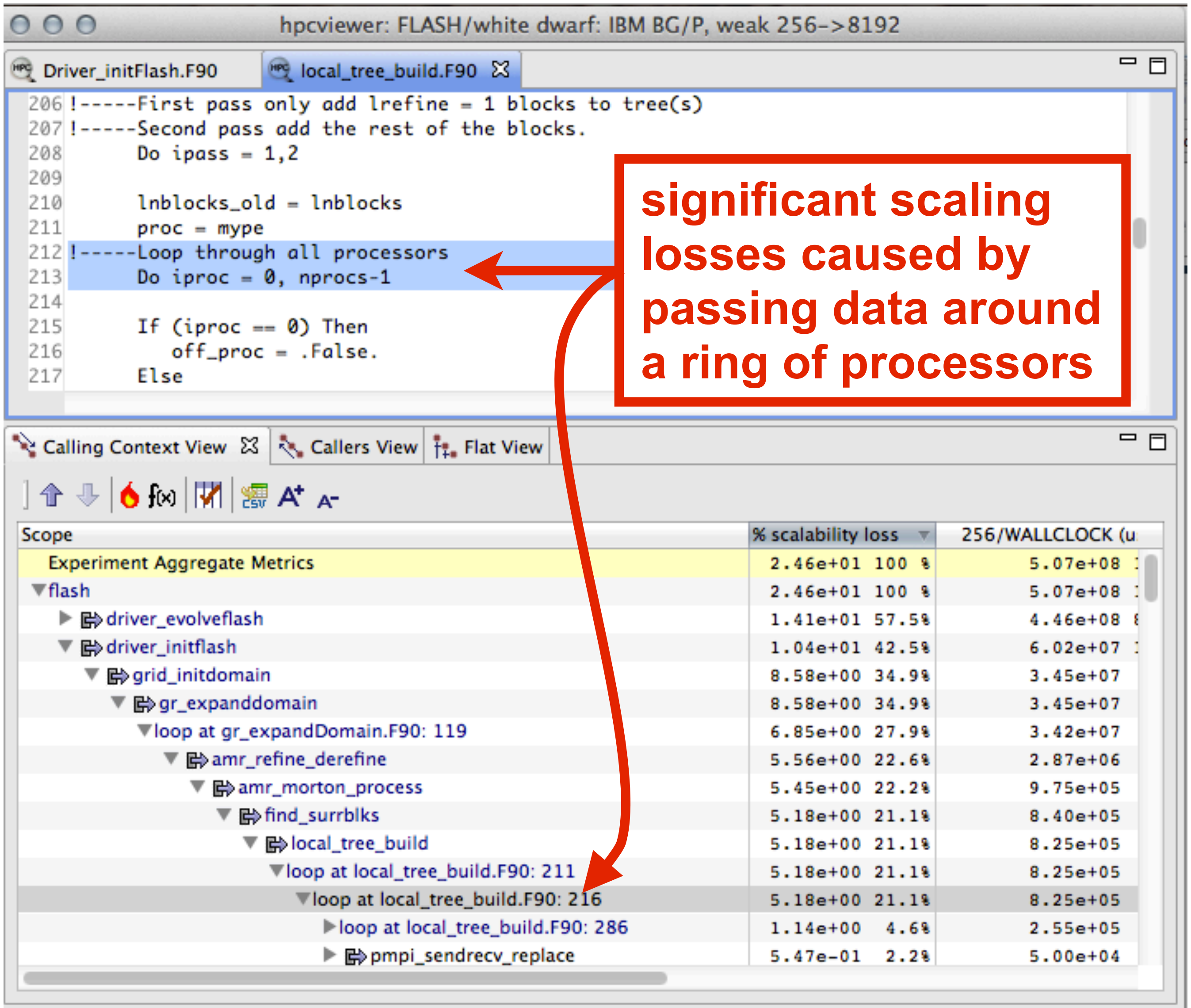
Figures courtesy of FLASH Team, University of Chicago

Scalability Analysis of Flash (Demo)

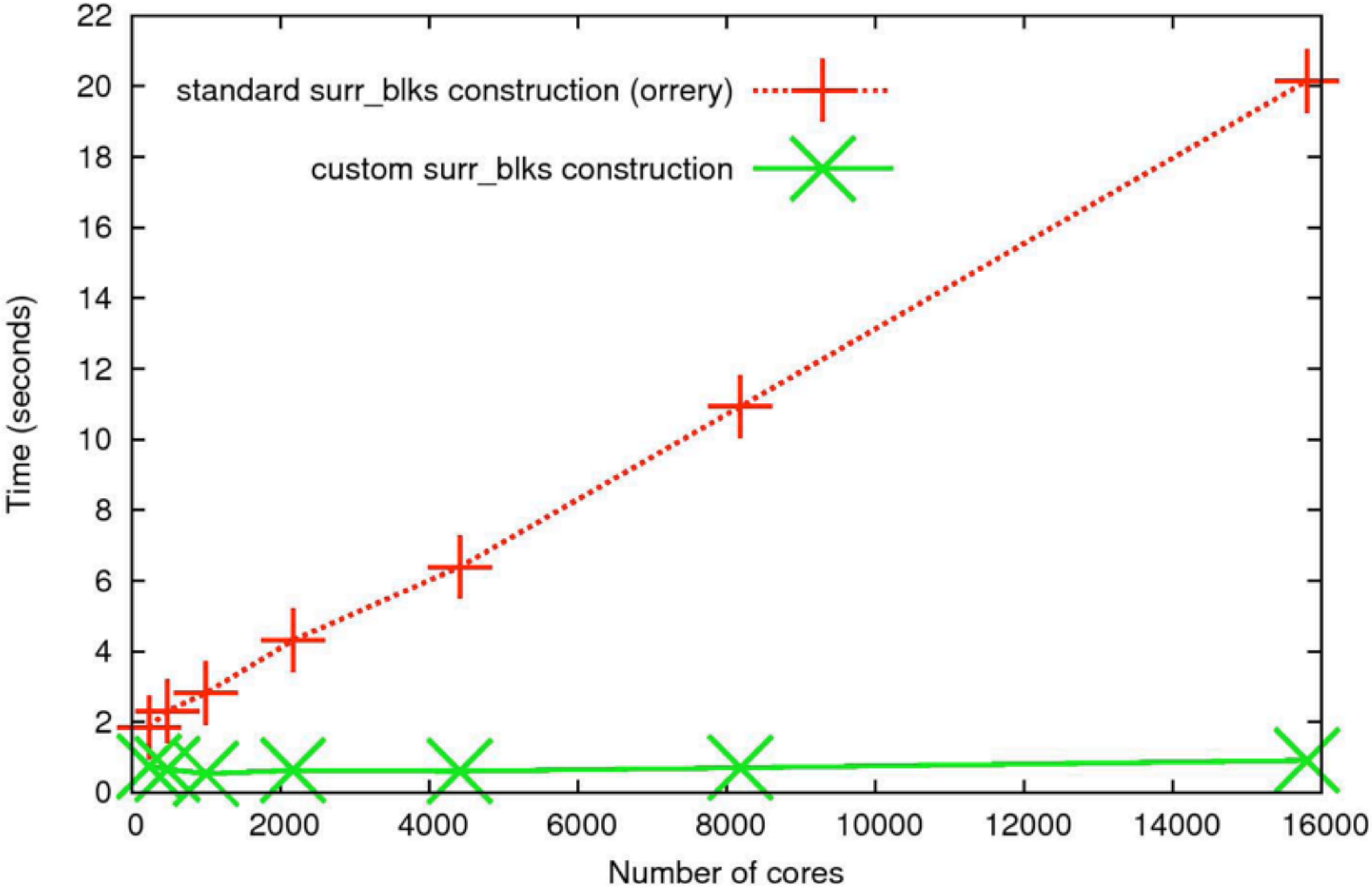


Scalability Analysis

- **Difference call path profile from two executions**
  - — ***different number of nodes***
  - — ***different number of threads***
- **Pinpoint and quantify scalability bottlenecks within and across nodes**



Improved Flash Scaling of AMR Setup

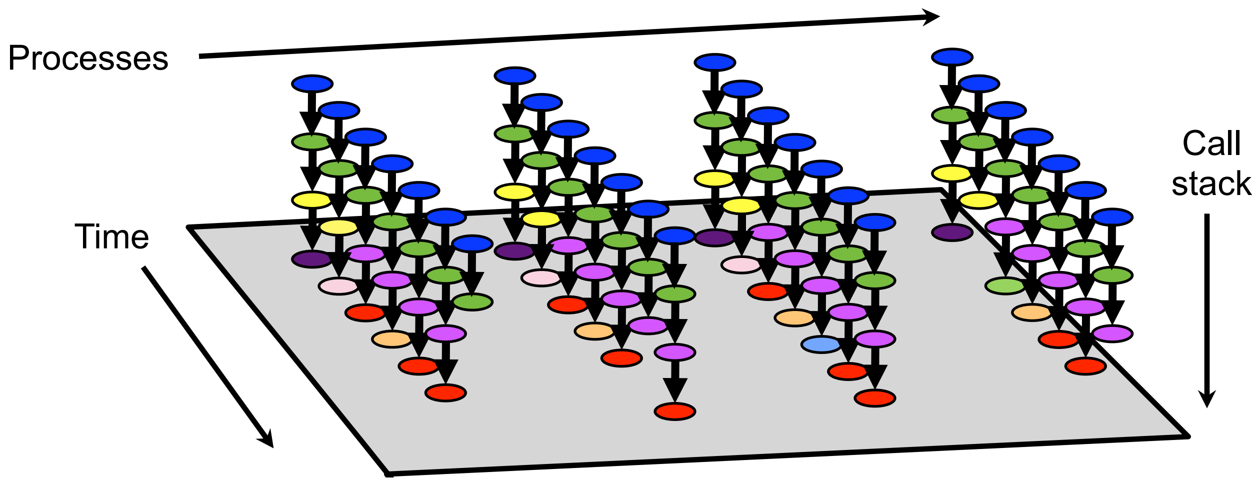


Graph courtesy of Anshu Dubey, U Chicago

Understanding Temporal Behavior

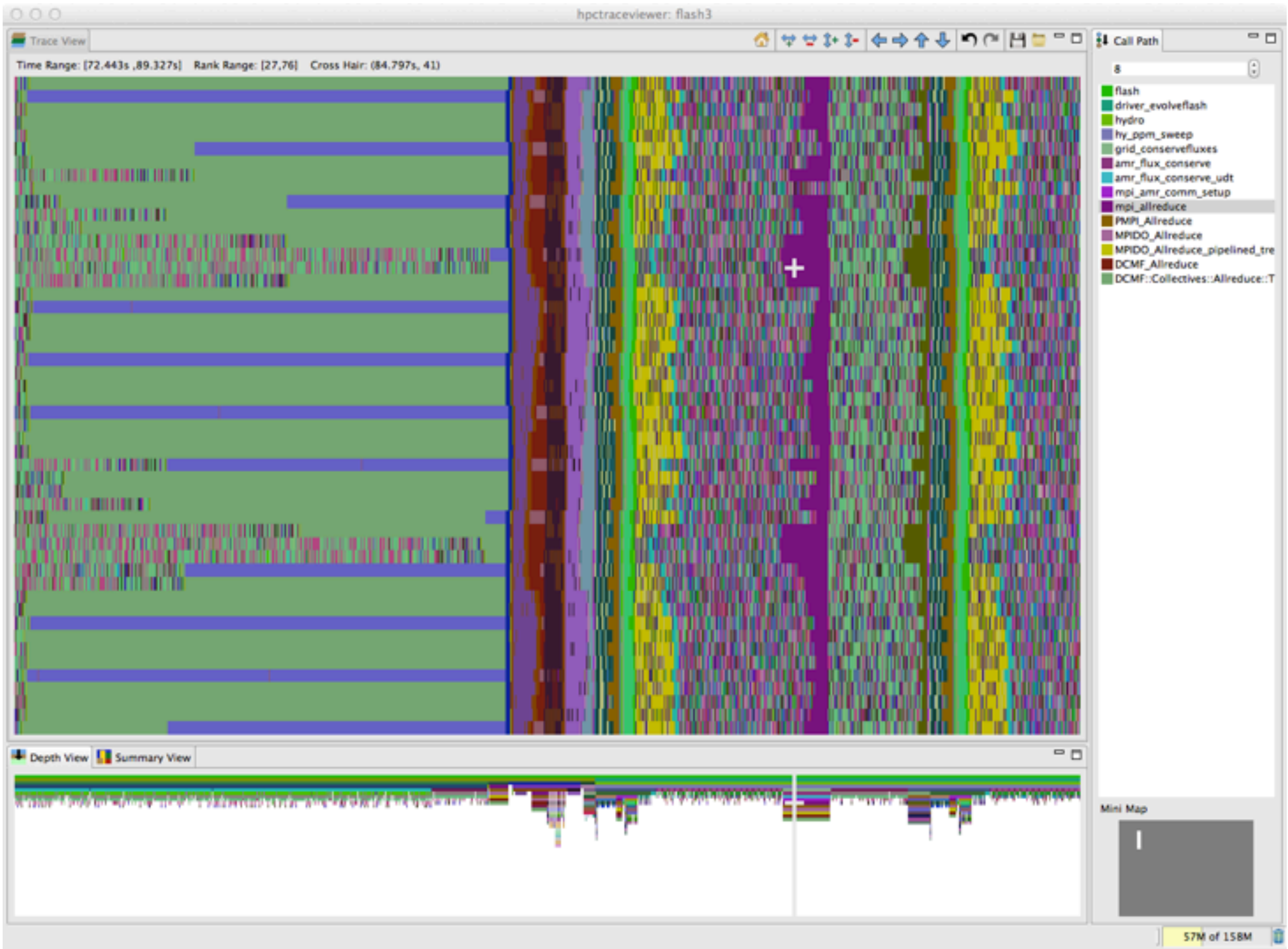
- **Profiling compresses out the temporal dimension**
  - —**temporal patterns, e.g. serialization, are invisible in profiles**
- **What can we do? Trace call path samples**
  - —**sketch:**
    - **N times per second, take a call path sample of each thread**
    - **organize the samples for each thread 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 FLASH3@256PE

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

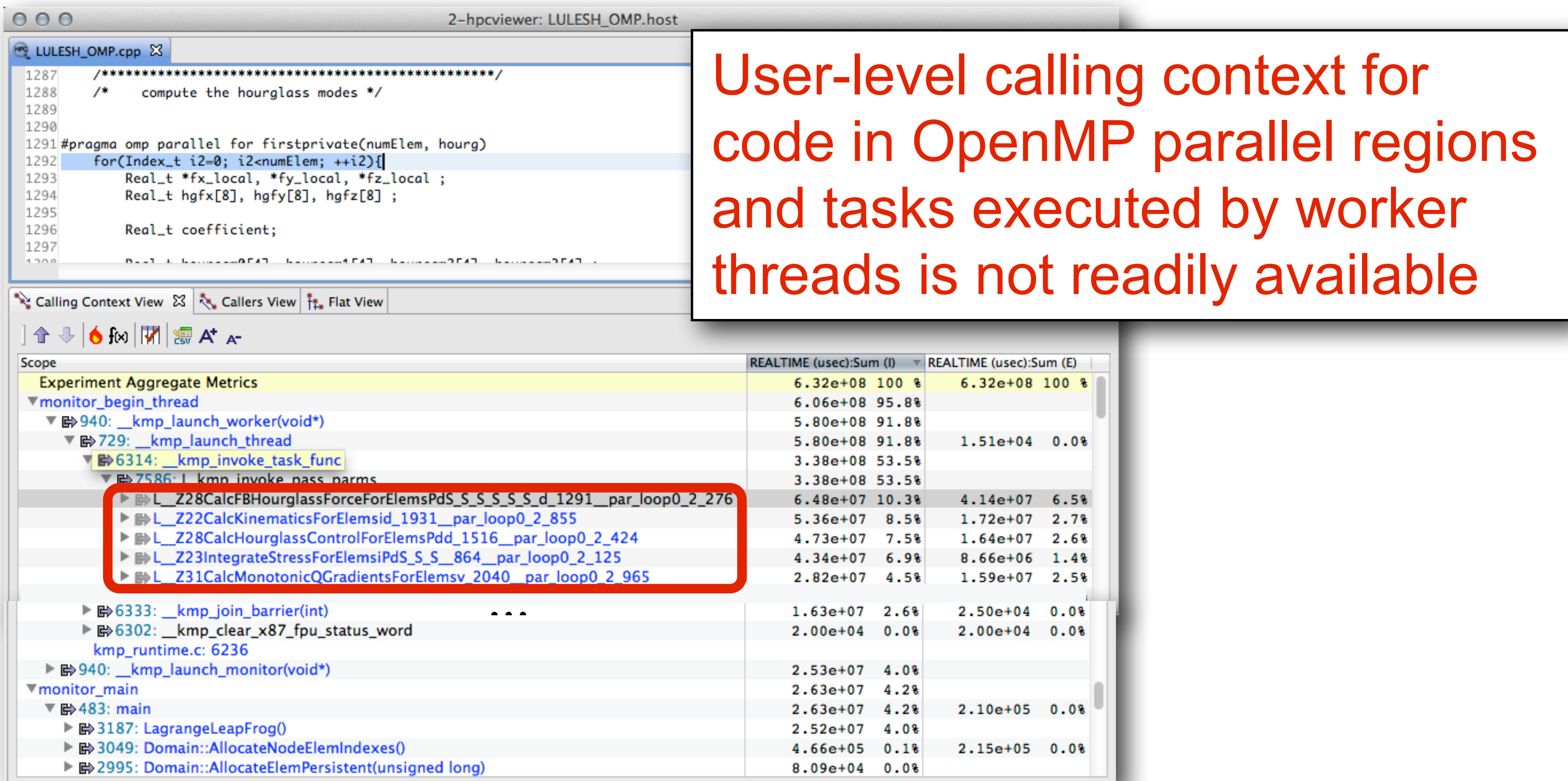


Measurement & Attribution of L2 Activity

- **L2Unit measurement capabilities**
  - e.g., counts load/store activity
  - node-wide counting; not thread-centric
  - global or per slice counting
  - supports threshold-based sampling
    - samples delivered late: about 800 cycles after threshold reached
    - each sample delivered to ALL threads/cores
- **HPCToolkit approach**
  - attribute a share of L2Unit activity to each thread context for each sample
    - e.g., when using a threshold of 1M loads and T threads, attribute 1M/T events to the active context in each thread when each sample event occurs
  - best effort attribution
    - strength: correlate L2Unit activity with regions of your code
    - weakness: some threads may get blamed for activity of others

OpenMP: A Challenge for Tools

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



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

Challenges for OpenMP Node Programs

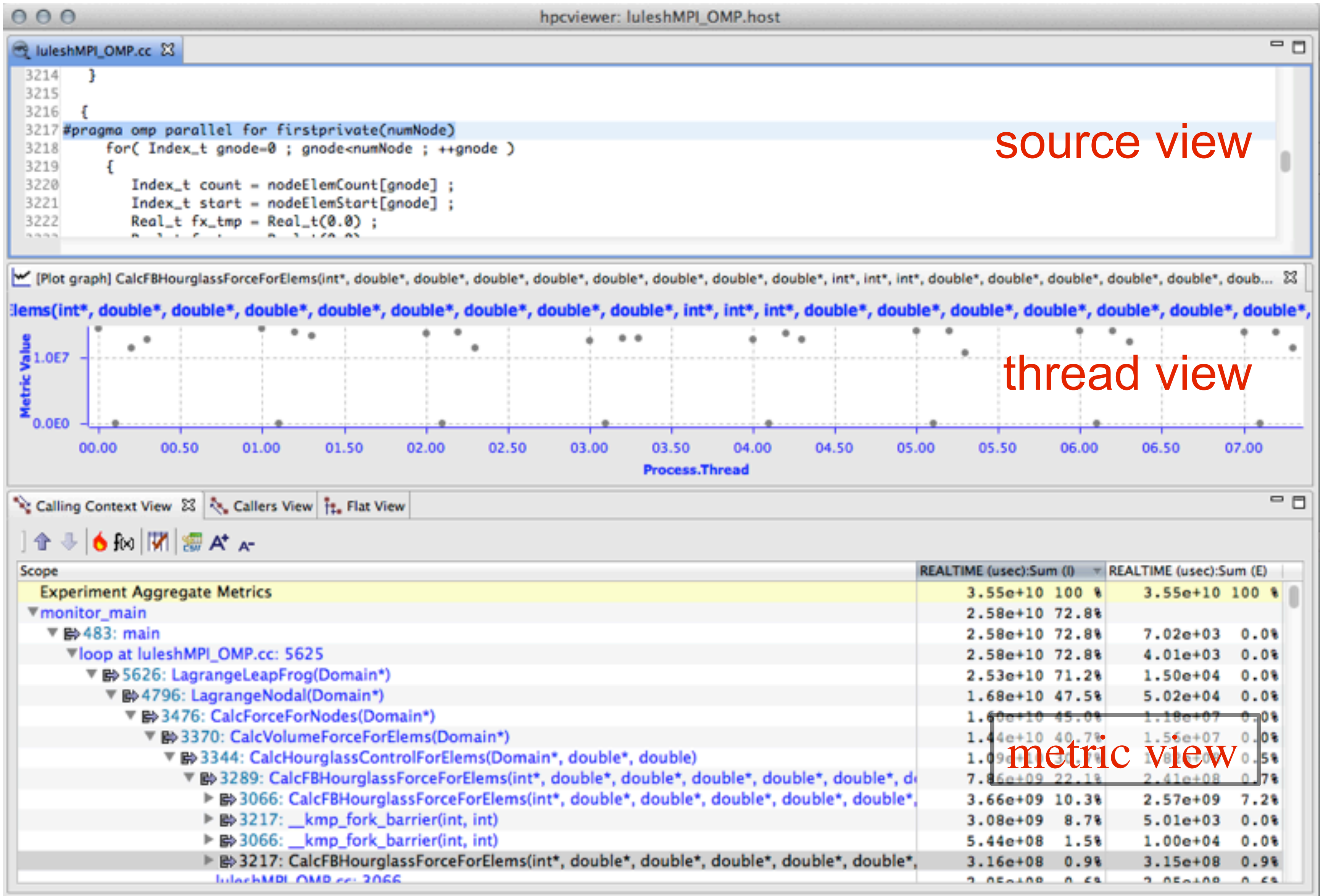
- Tools provide implementation-level view of OpenMP threads
  - asymmetric threads
    - master thread
    - worker thread
  - run-time frames are interspersed with user code
- Hard to understand causes of idleness
  - long serial sections
  - load imbalance in parallel regions
  - waiting for critical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



Integrated View of MPI+OpenMP with OMPT

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

Blame-shifting: Analyze Thread Performance

	Problem	Approach
Undirected Blame Shifting[1,3]	A thread is idle waiting for work	Apportion blame among working threads for not shedding enough parallelism to keep all threads busy
Directed Blame Shifting[2,3]	A thread is idle waiting for a mutex	Blame the thread holding the mutex for idleness of threads waiting for the mutex

[1]Tallent & Mellor-Crummey: PPoPP 2009

[2]Tallent, Mellor-Crummey, Porterfield: PPoPP 2010

[3]Liu, Mellor-Crummey, Fagan: ICS 2013

OpenMP Tools API Status

- **April 2014: OpenMP TR2**
  - —OMPT: An OpenMP Tools Application Programming Interface for Performance Analysis
    - – Alexandre Eichenberger (IBM), John Mellor-Crummey (Rice), Martin Schulz (LLNL), Nawal Copty (Oracle), Jim Cownie (Intel), Robert Dietrich (TU Dresden), Xu Liu (Rice), Eugene Loh (Oracle), Daniel Lorenz (Juelich), and other members of the OpenMP tools subcommittee
  - —<u>major</u> step toward having a tools API added to OpenMP standard
- **OMPT implementations: IBM, Intel (prototype), GOMP (partial), LLVM (soon)**
- **Next steps**
  - —transition OMPT prototype into Intel for use with production OpenMP runtime
    - – http://code.google.com/p/ompt-intel-openmp
    - – contributors: Rice, University of Oregon, RWTH Aachen, TU Dresden
    - – status: finalizing code in preparation for merge into Intel's LLVM OpenMP
      - ongoing dialog with Jim Cownie (Intel SSG/DPD/TCAR)
  - —propose OMPT additions to the language standard
  - —refine HPCToolkit OMPT support for production use

Ongoing Work and Future Plans

- **Argonne**
  - — deploy OMPT support for OpenMP on Blue Gene/Q
  - — scale I/O strategy
    - – one file per node rather than one file per thread
  - — scale traceviewer
    - – split traceviewer into client server
      - server runs as a parallel program on vis cluster
      - client runs on your laptop
- **Other work**
  - — data-centric analysis: associate costs with variables
  - — analysis and attribution of performance to optimized code
- **Future plans**
  - — resource-centric performance analysis
    - – within and across nodes
  - — scale measurement and analysis for exascale
  - — automated analysis to deliver performance insights

HPCToolkit at ALCF

- **ALCF systems (vesta, mira, cetus)**
  - — in your .soft file, add one of the following lines below
    - – +hpctoolkit-devel
    - – (this package is always the most up-to-date)
- **Man pages**
  - — automatically added to MANPATH by the aforementioned softenv command
- **ALCF guide to HPCToolkit**
  - — http://www.alcf.anl.gov/user-guides/hpctoolkit
- **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**

Using HPCToolkit

- **Add hpctoolkit's bin directory to your path using softenv**
- **Adjust your compiler flags (if you want full attribution to src)**
  - add -g flag after any optimization flags
- **Add hpclink as a prefix to your Makefile's link line**
  - e.g. `hpclink mpixlf -o myapp foo.o ... lib.a -lm ...`
- **See what sampling triggers are available on BG/Q**
  - 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

Collecting Performance Data on BG/Q

- **Collecting traces on BG/Q**
  - set environment variable HPCRUN_TRACE=1
  - use WALLCLOCK or PAPI_TOT_CYC as one of your sample sources when collecting a trace
- **Launching your job on BG/Q using hpctoolkit**
  - qsub -A ... -t 10 -n 1024 --mode c1 --proccount 16384 \
    --cwd `pwd` \
    --env OMP_NUM_THREADS=2:\
    HPCRUN_EVENT_LIST=WALLCLOCK@5000:\
    HPCRUN_TRACE=1\
    your_executable

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 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
- **Hint: you can run hpcprof-mpi on the x86_64 vis cluster**

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