Analyzing Parallel Program Performance using HPCToolkit

John Mellor-Crummey
Department of Computer Science
Rice University

`http://hpctoolkit.org`

RICE

Acknowledgments

- Current funding
  - DOE Exascale Computing Project (Subcontract 400015182)
  - NSF Software Infrastructure for Sustained Innovation (Collaborative Agreement 1450273)
  - ANL (Subcontract 4F-30241)
  - LLNL (Subcontracts B609118, B614178)
  - Intel gift funds
- Project team
  - Research Staff
    - Laksono Adhianto, Mark Krentel, Scott Warren, Doug Moore
  - Students
    - Lai Wei, Keren Zhou
  - Recent Alumni
    - Xu Liu (William and Mary)
    - Milind Chabbi (Baidu Research)
    - Mike Fagan (Rice)

Challenges for Computational Scientists

- **Rapidly evolving platforms and applications**
  - architecture
    - rapidly changing designs for compute nodes
    - significant architectural diversity

      multicore, manycore, accelerators
    - increasing parallelism within nodes
  - applications
    - exploit threaded parallelism in addition to MPI
    - leverage vector parallelism
    - augment computational capabilities
- **Computational scientists need to**
  - adapt codes to changes in emerging architectures
  - improve code 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

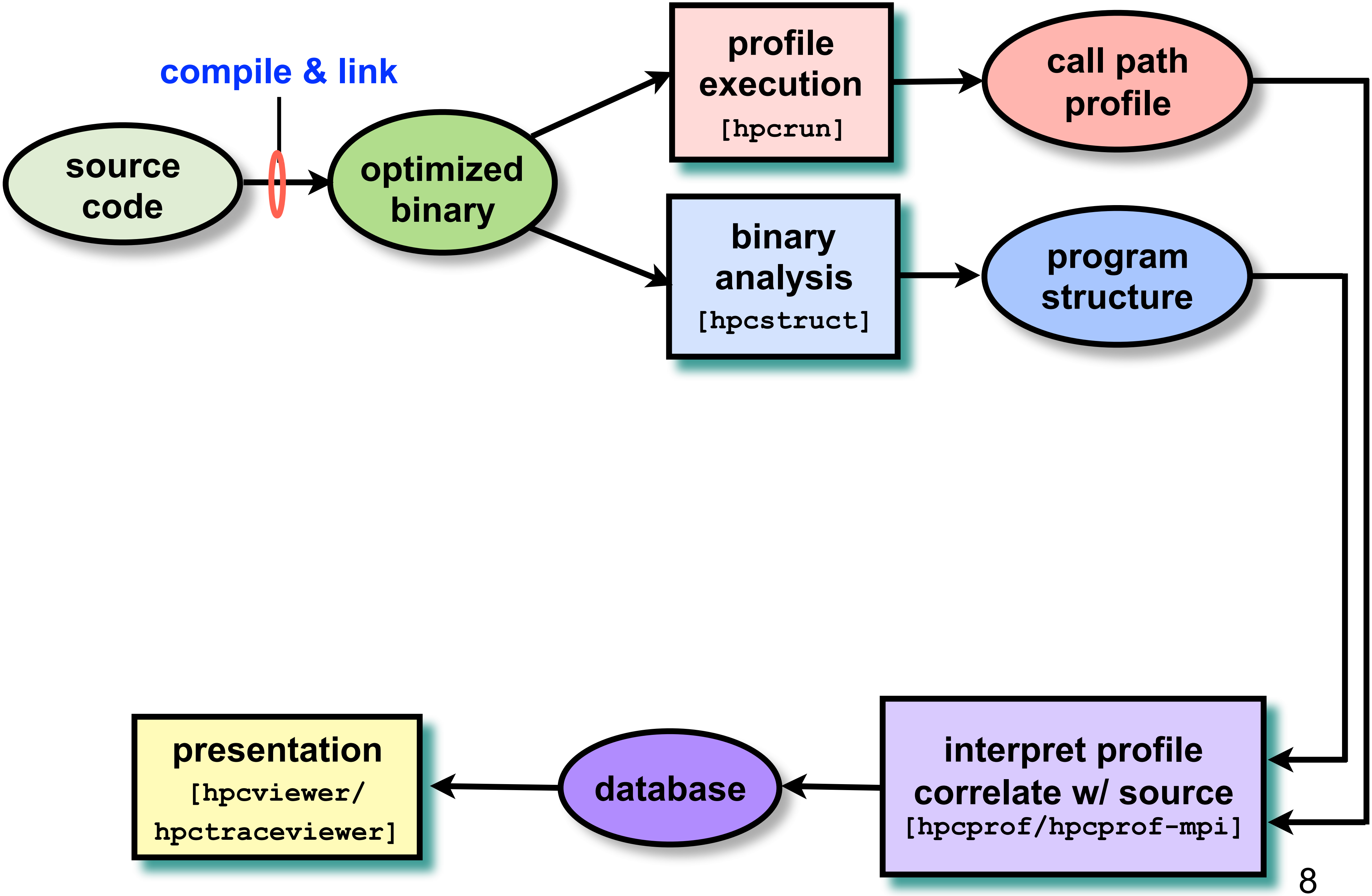
Outline

- **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 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



compile & link

source code

optimized binary

profile execution [hpcrun]

call path profile

binary analysis [hpcstruct]

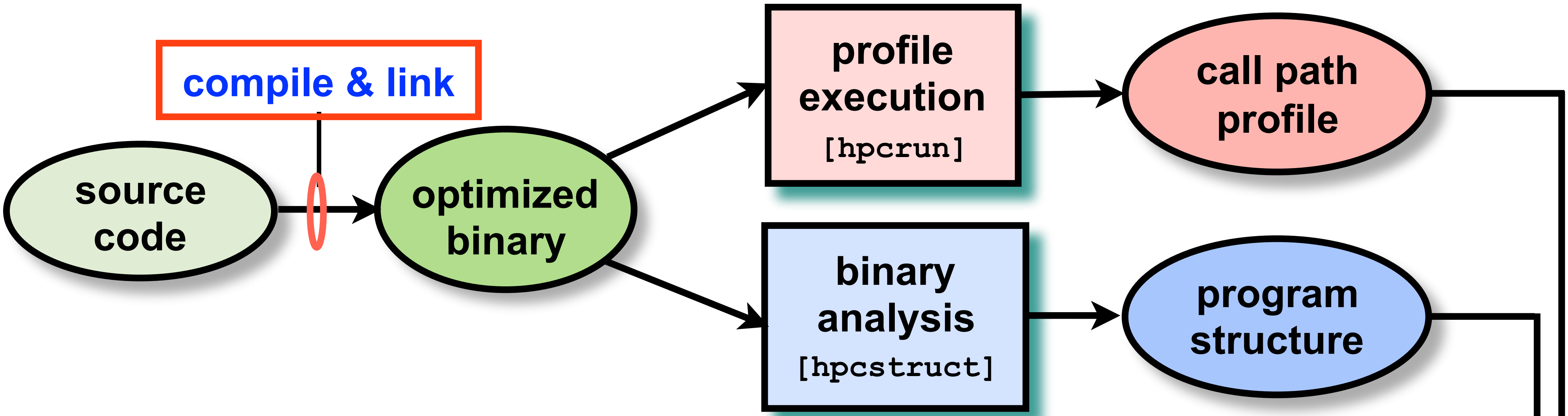
program structure

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

database

presentation [hpcviewer/ hpctraceviewer]

HPCToolkit Workflow



compile & link

source code

optimized binary

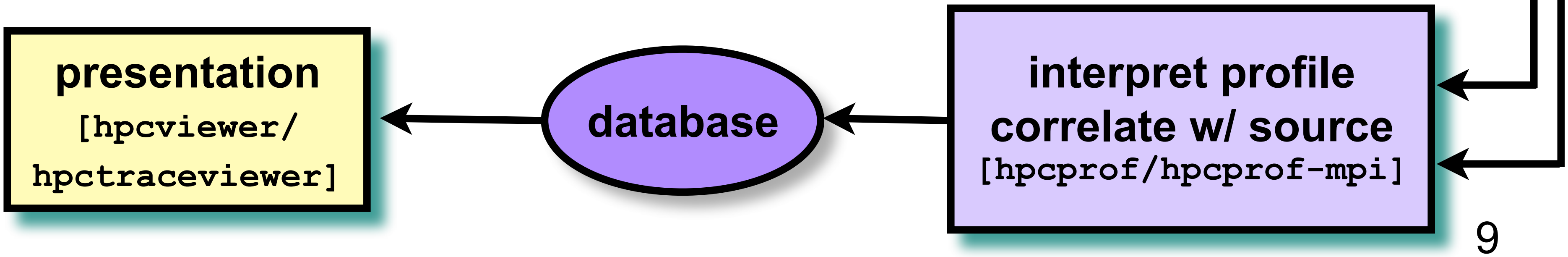
profile execution
`[hpcrun]`

call path profile

binary analysis
`[hpcstruct]`

program structure

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

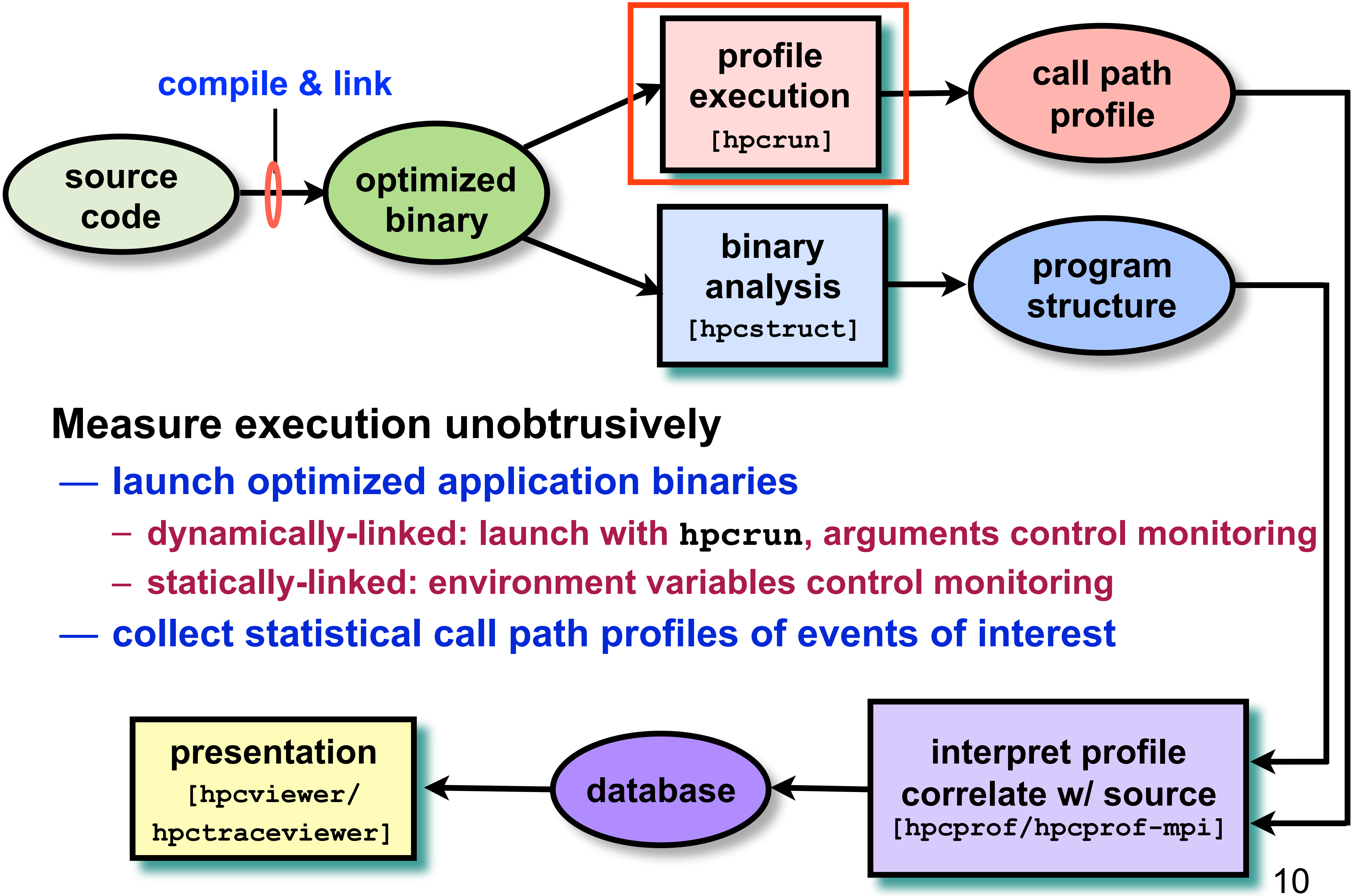


presentation
`[hpcviewer/`
`hpctraceviewer]`

database

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

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]

Measure execution unobtrusively

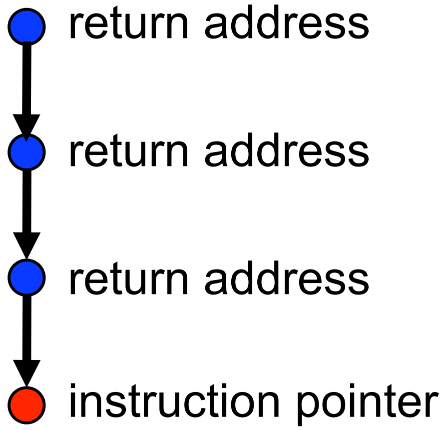
- launch optimized application binaries
  - dynamically-linked: launch with **`hpcrun`**, arguments control monitoring
  - statically-linked: environment variables control monitoring
- collect statistical call path profiles of events of interest

Call Path Profiling

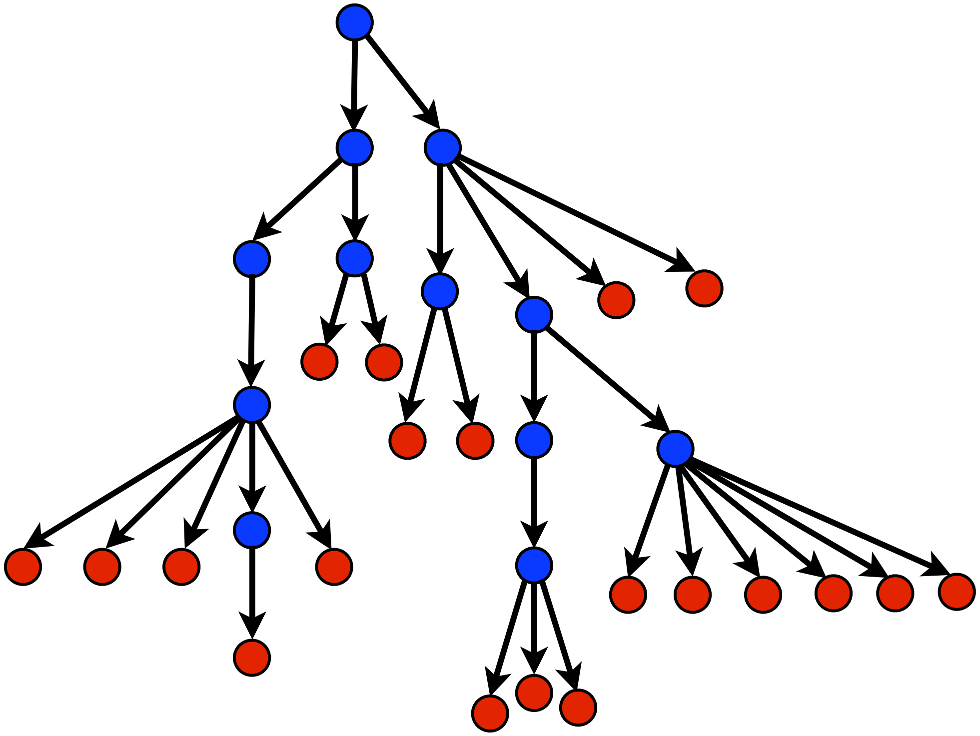
Measure and attribute costs in context

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

Call path sample

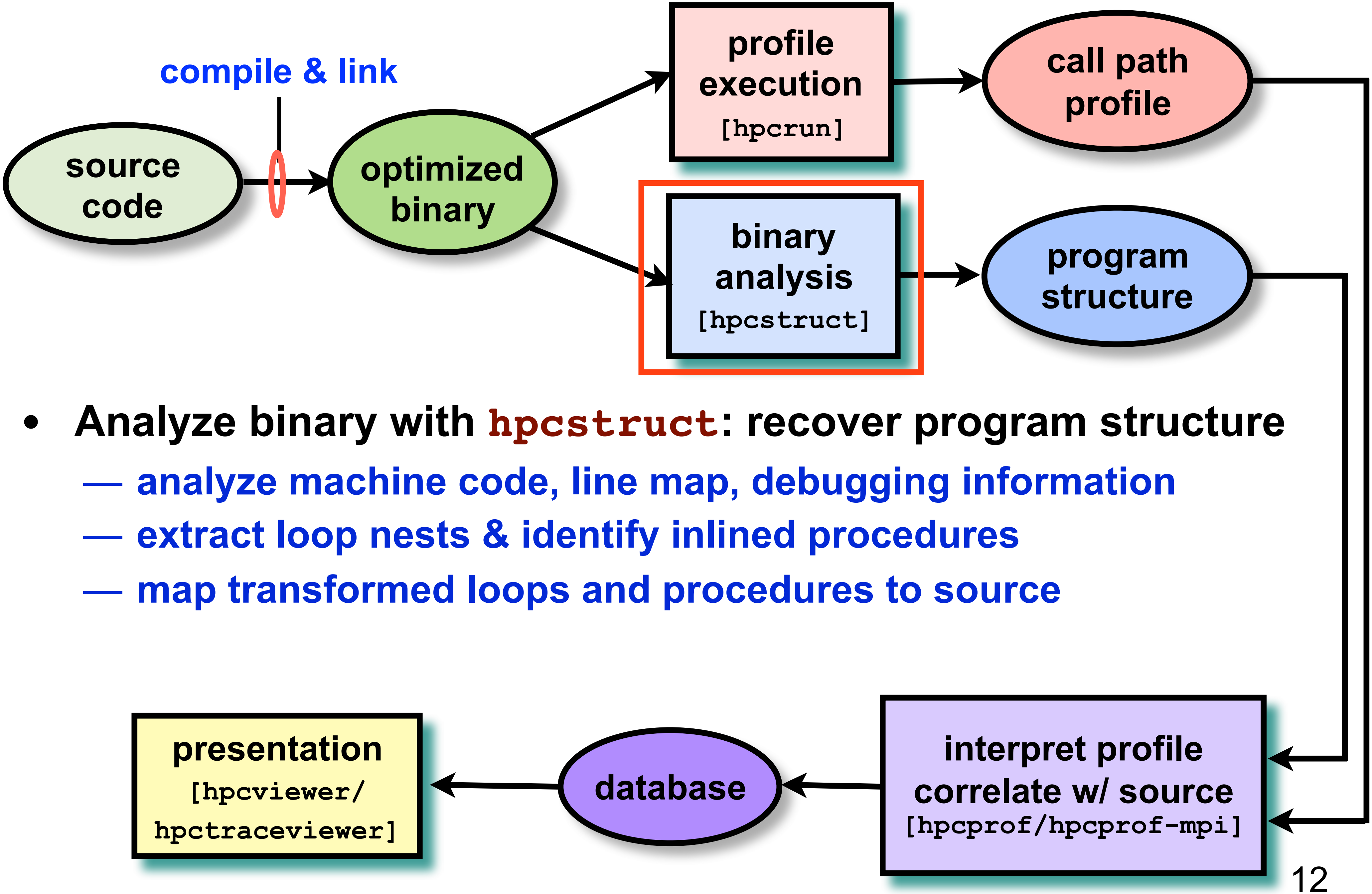


Calling context tree



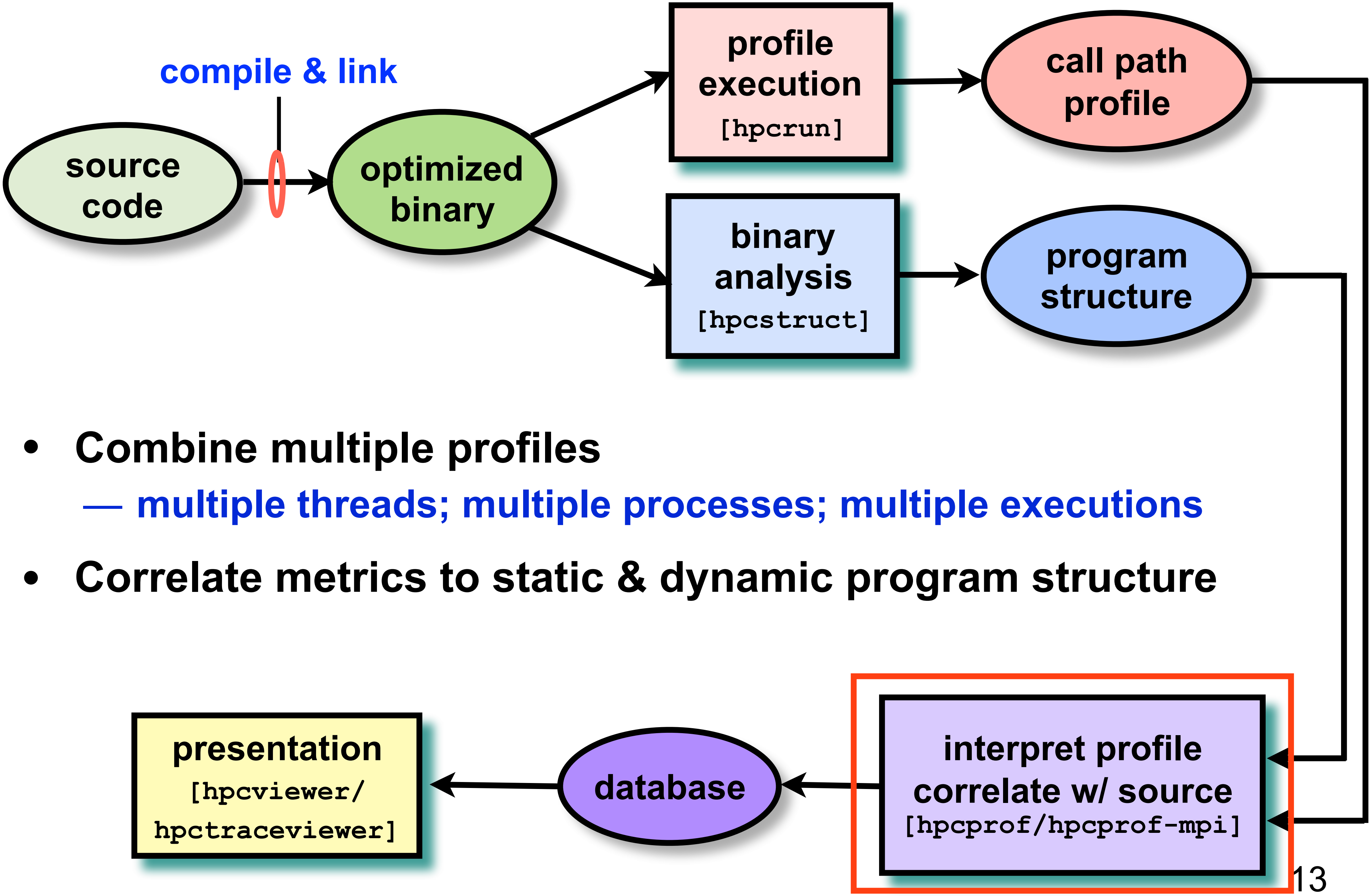
Overhead proportional to sampling frequency...
...not call frequency

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



compile & link

source code

optimized binary

profile execution [hpcrun]

call path profile

binary analysis [hpcstruct]

program structure

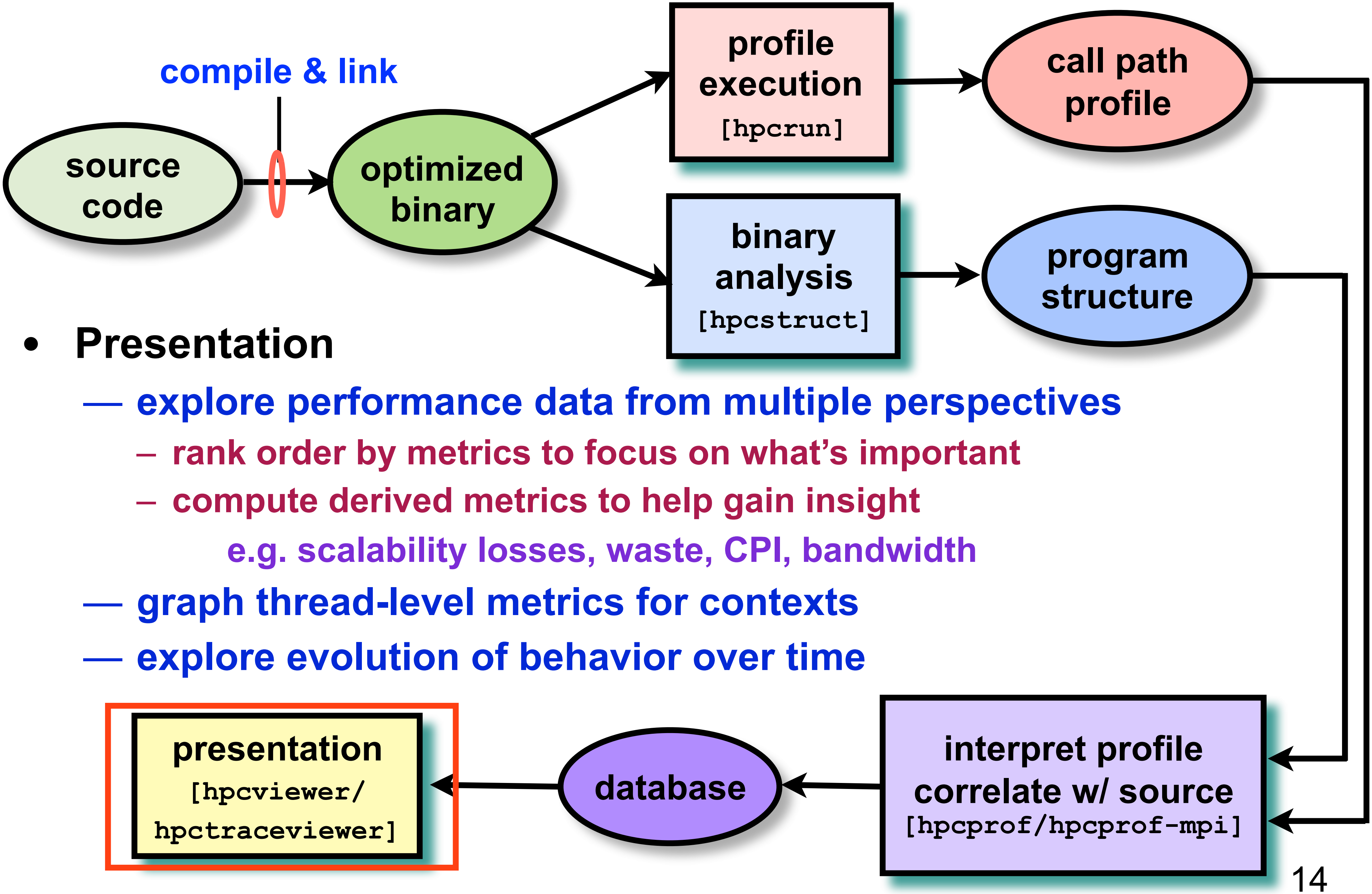
interpret profile correlate w/ source [hpcprof/hpcprof-mpi]

database

presentation [hpcviewer/ hpctraceviewer]

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

HPCToolkit Workflow



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]

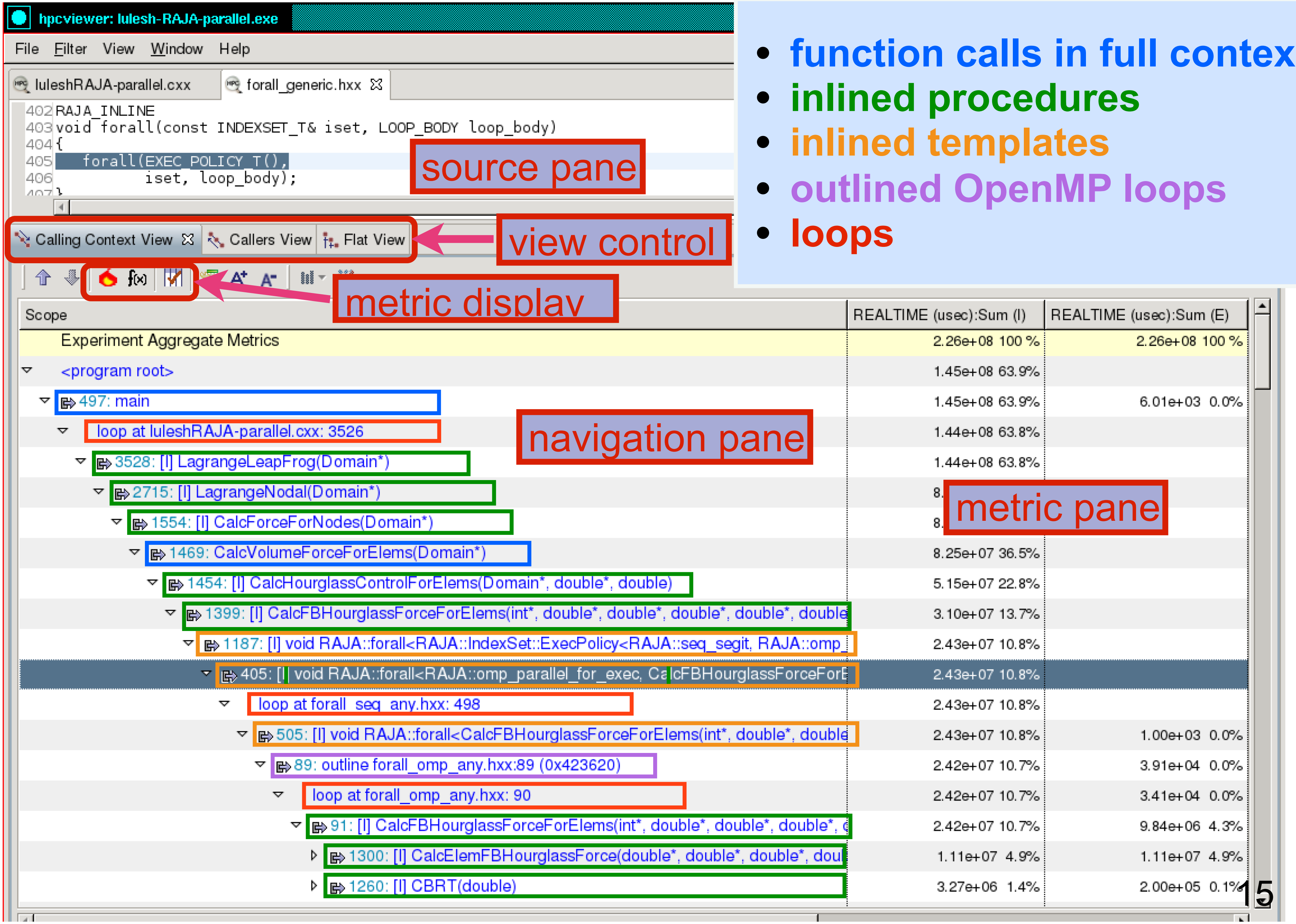
database

presentation
[hpcviewer/
hpctraceviewer]

- **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



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

source pane

view control

metric display

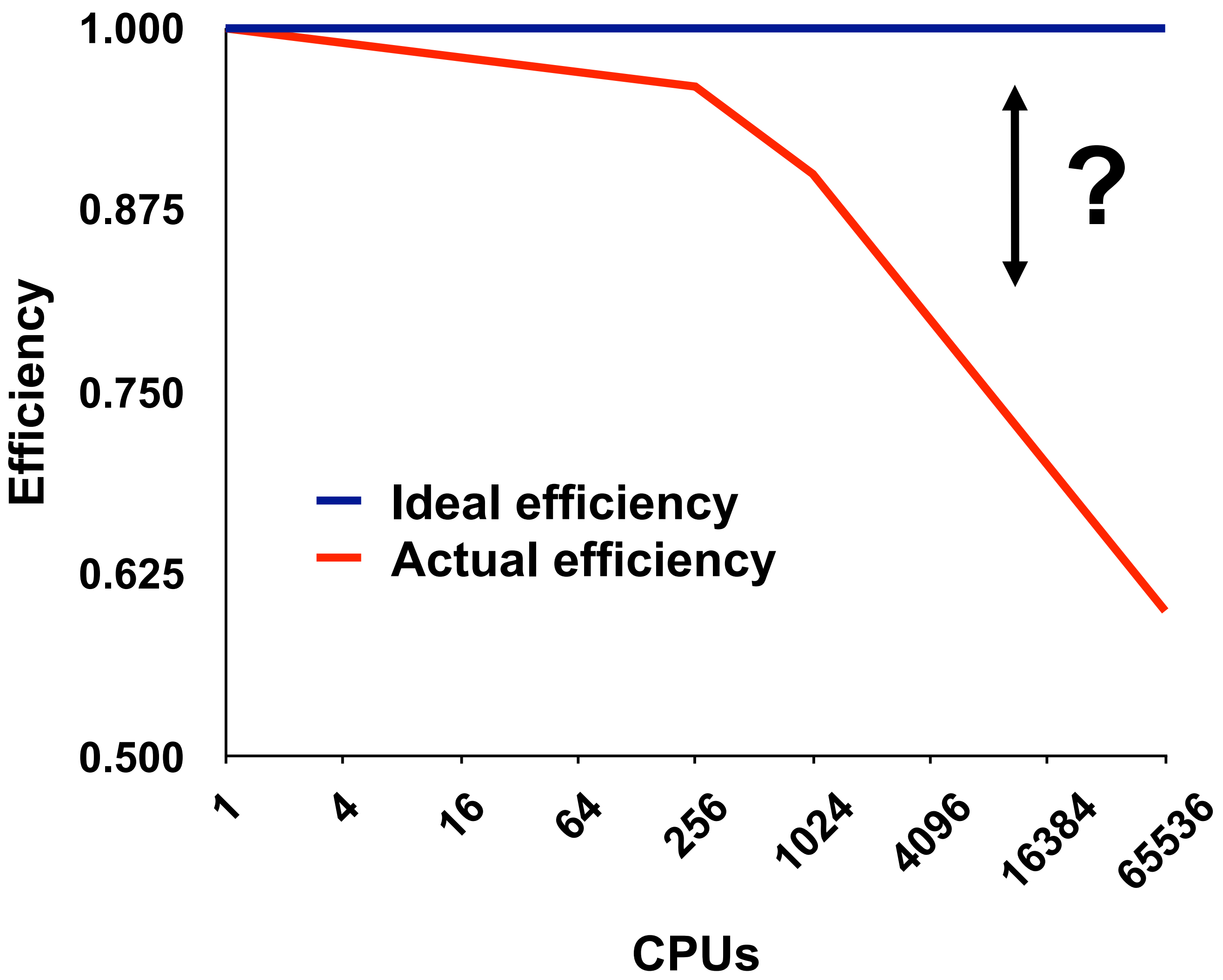
navigation pane

metric pane

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.	
1554: [I] CalcForceForNodes(Domain*)	8.	
1469: CalcVolumeForceForElems(Domain*)	8.25e+07 36.5%	
1454: [I] CalcHourglassControlForElems(Domain*, double*, double)	5.15e+07 22.8%	
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*, c	2.42e+07 10.7%	9.84e+06 4.3%
1300: [I] CalcElemFBHourglassForce(double*, double*, double*, doul	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

The Problem of Scaling



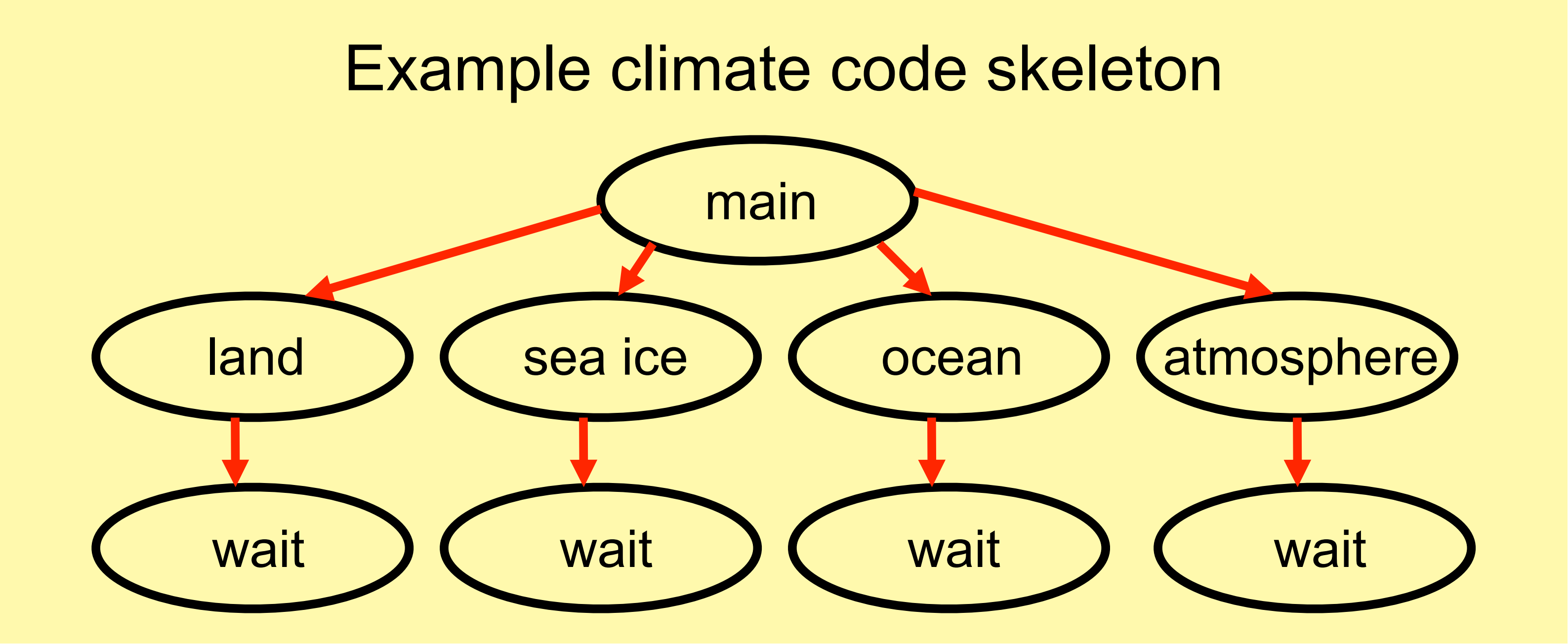
Note: higher is better

Goal: Automatic Scalability 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



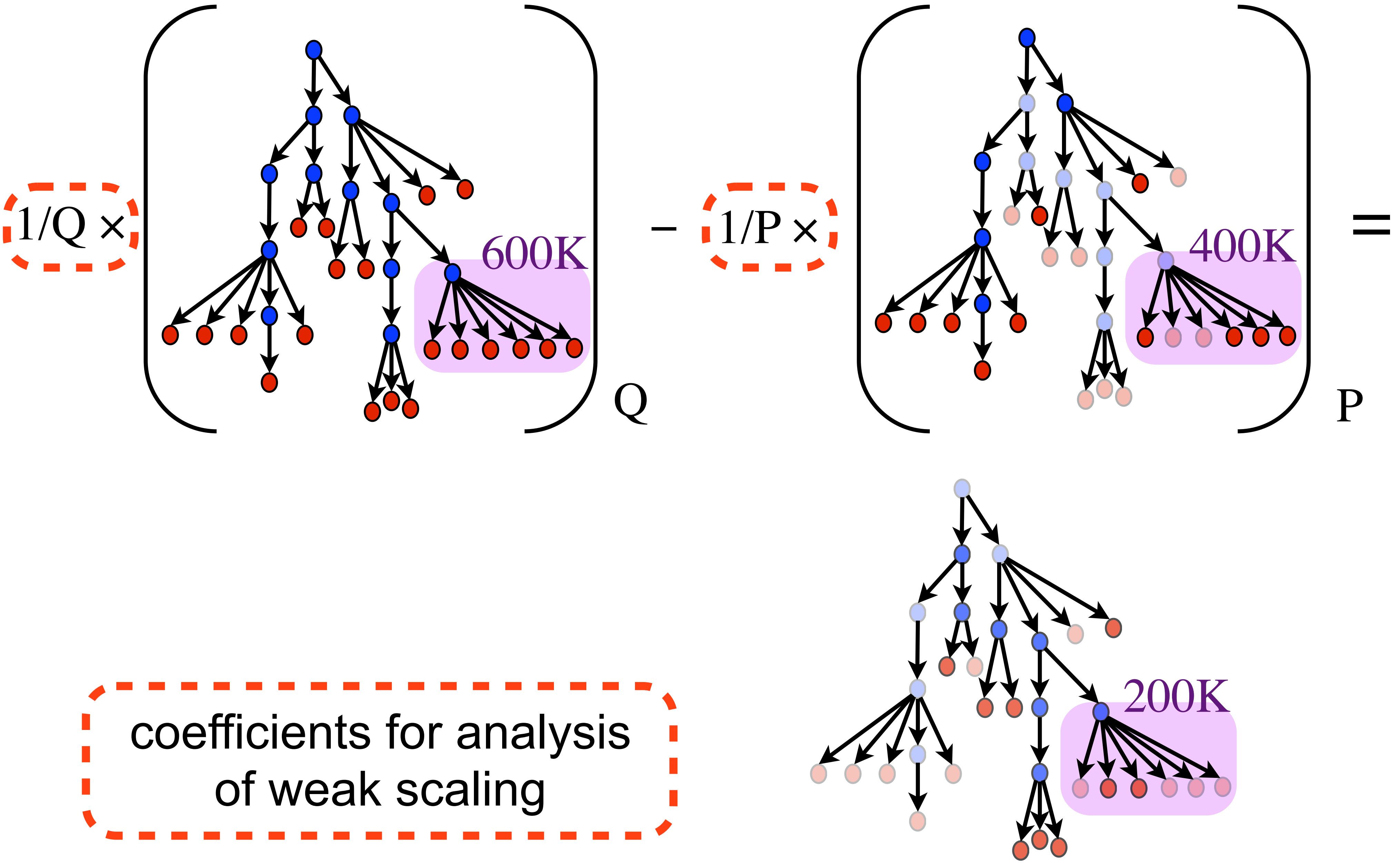
- **Monitoring**
  - bottleneck nature: computation, data movement, synchronization?
  - 2 pragmatic constraints
    - acceptable data volume
    - low perturbation for use in production runs

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

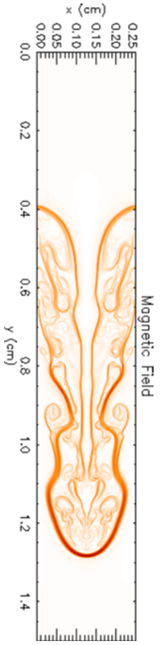
Pinpointing and Quantifying Scalability Bottlenecks



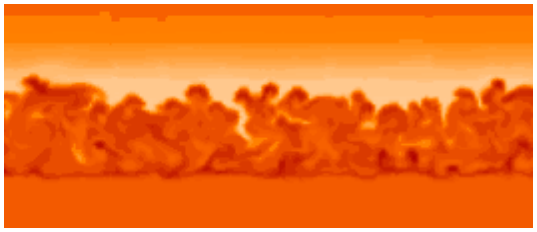
coefficients for analysis of weak scaling

Scalability Analysis Demo

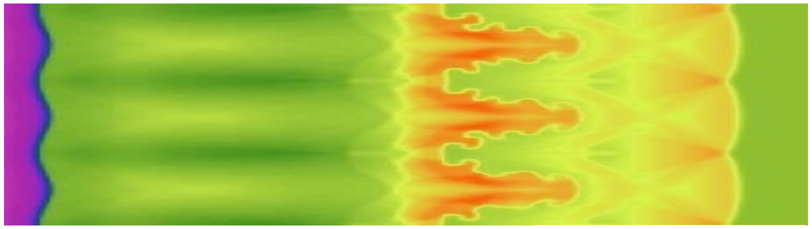
Code: University of Chicago FLASH
Simulation: white dwarf detonation
Platform: Blue Gene/P
Experiment: 8192 vs. 256 processors
Scaling type: weak



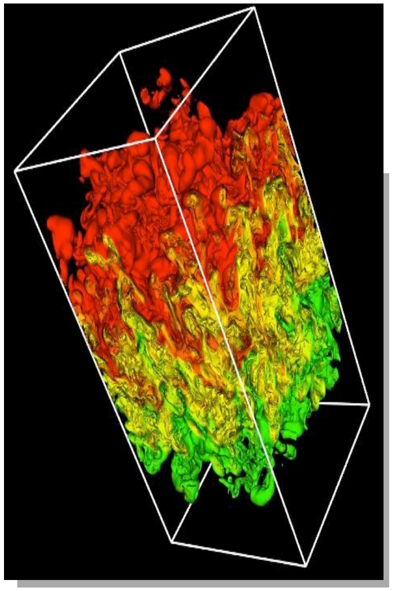
Magnetic Rayleigh-Taylor



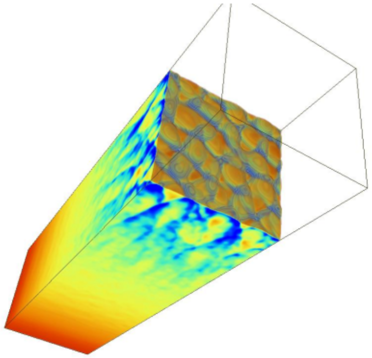
Nova outbursts on white dwarfs



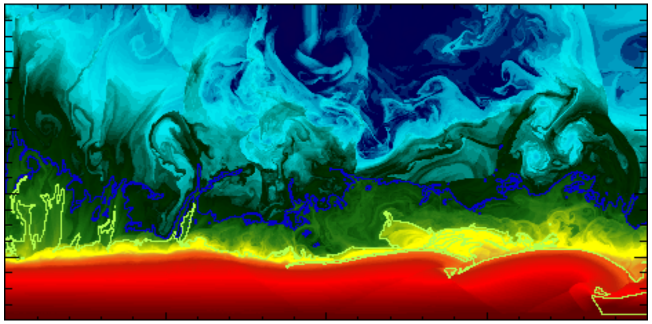
Laser-driven shock instabilities



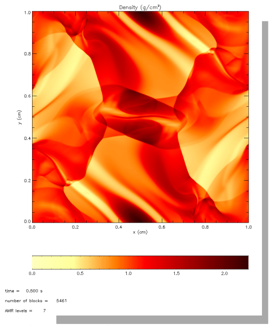
Rayleigh-Taylor instability



Cellular detonation



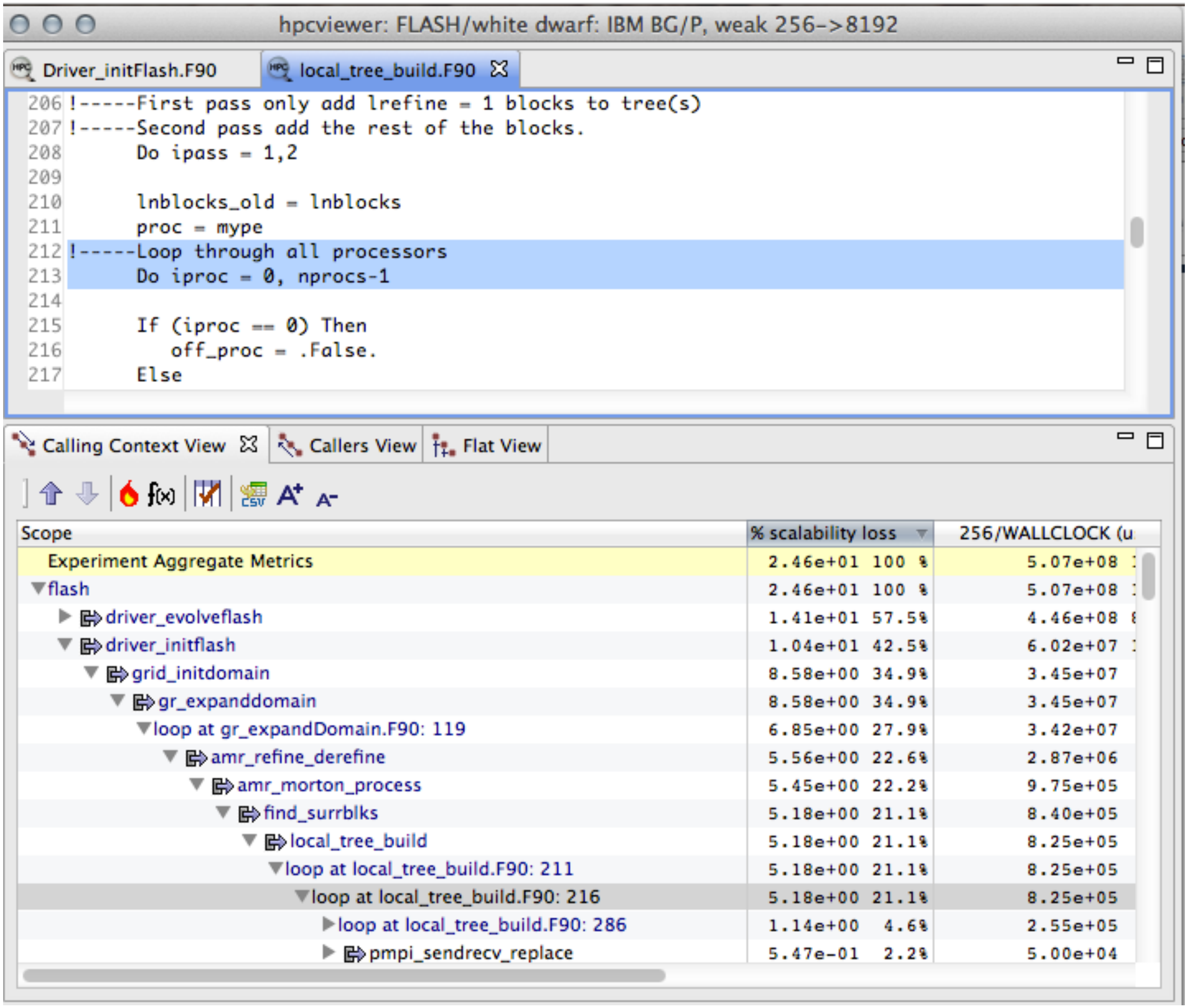
Helium burning on neutron stars



Orzag/Tang MHD vortex

Figures courtesy of FLASH Team, University of Chicago

Scalability Analysis of Flash (Demo)

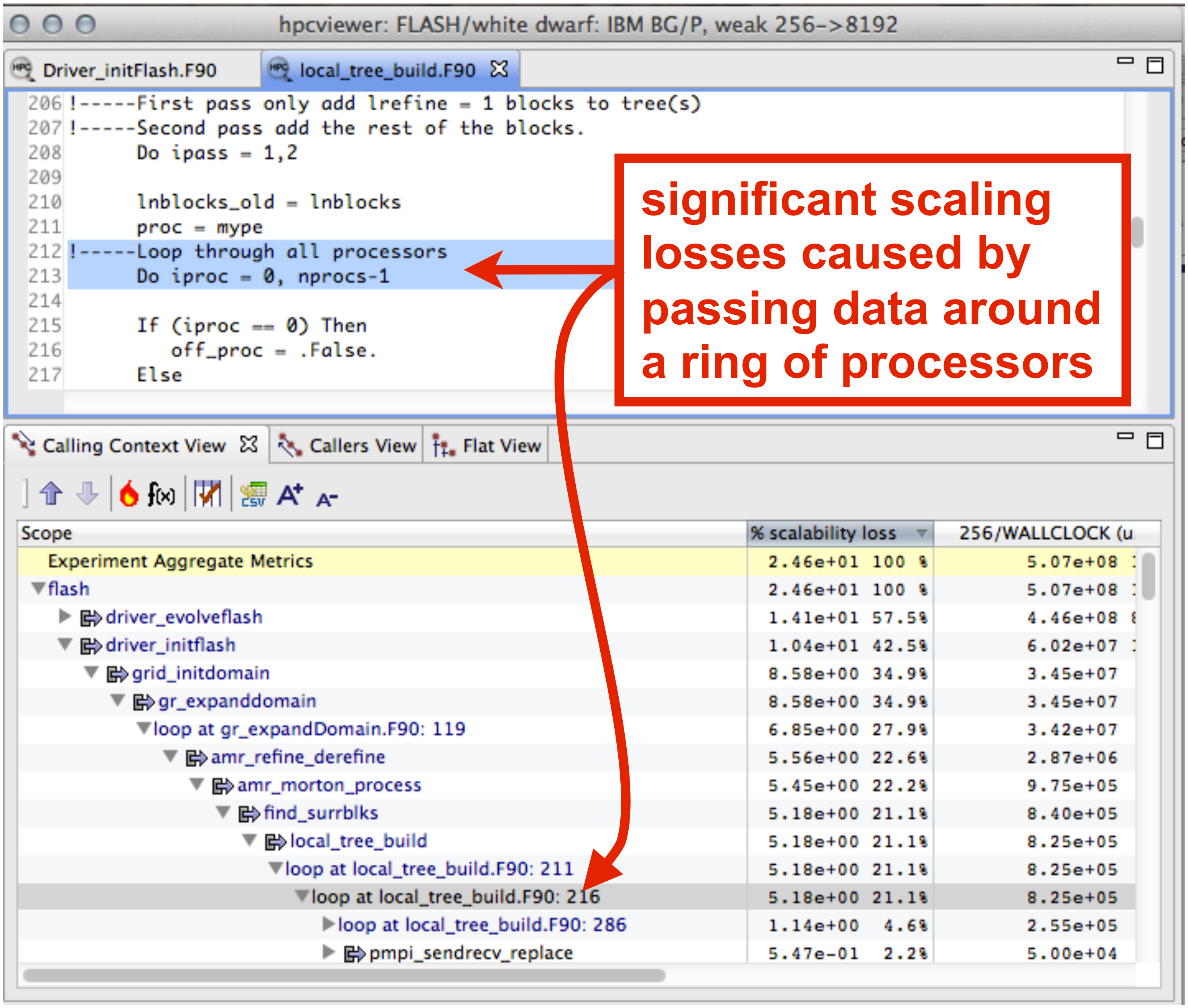


```
!-----First pass only add lrefine = 1 blocks to tree(s)
!-----Second pass add the rest of the blocks.
      Do ipass = 1,2

      lnblocks_old = lnblocks
      proc = mype
!-----Loop through all processors
      Do iproc = 0, nprocs-1

      If (iproc == 0) Then
         off_proc = .False.
      Else
```

Scope	% scalability loss		256/WALLCLOCK (u
Experiment Aggregate Metrics	2.46e+01	100 %	5.07e+08
flash	2.46e+01	100 %	5.07e+08
driver_evolveflash	1.41e+01	57.5%	4.46e+08
driver_initflash	1.04e+01	42.5%	6.02e+07
grid_initdomain	8.58e+00	34.9%	3.45e+07
gr_expanddomain	8.58e+00	34.9%	3.45e+07
loop at gr_expandDomain.F90: 119	6.85e+00	27.9%	3.42e+07
amr_refine_derefine	5.56e+00	22.6%	2.87e+06
amr_morton_process	5.45e+00	22.2%	9.75e+05
find_surrblks	5.18e+00	21.1%	8.40e+05
local_tree_build	5.18e+00	21.1%	8.25e+05
loop at local_tree_build.F90: 211	5.18e+00	21.1%	8.25e+05
loop at local_tree_build.F90: 216	5.18e+00	21.1%	8.25e+05
loop at local_tree_build.F90: 286	1.14e+00	4.6%	2.55e+05
pmpi_sendrecv_replace	5.47e-01	2.2%	5.00e+04

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**

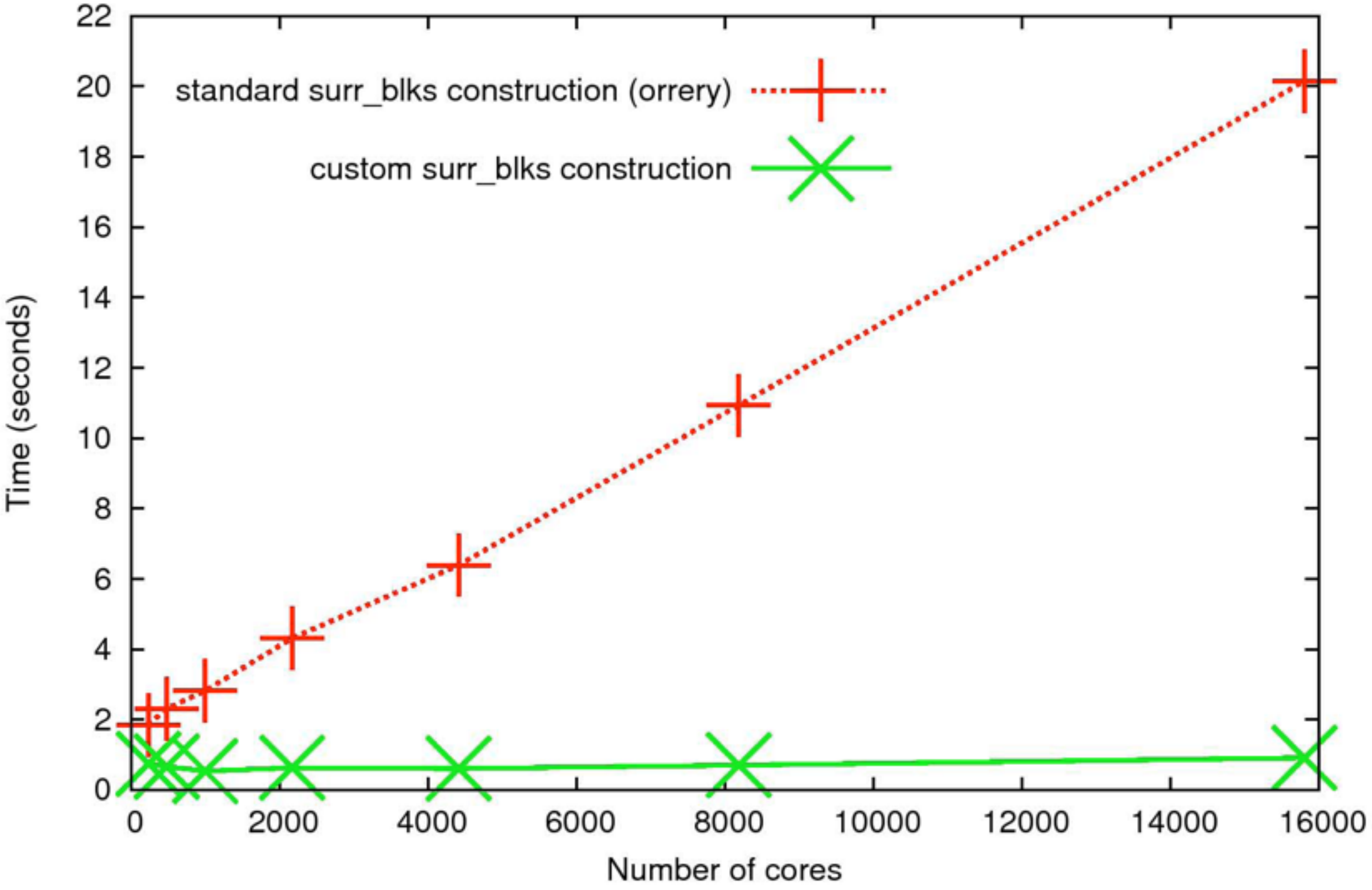


```
!-----First pass only add lrefine = 1 blocks to tree(s)
!-----Second pass add the rest of the blocks.
      Do ipass = 1,2

      lnblocks_old = lnblocks
      proc = mype
!-----Loop through all processors
      Do iproc = 0, nprocs-1

      If (iproc == 0) Then
         off_proc = .False.
      Else
```

Scope	% scalability loss		256/WALLCLOCK (u
Experiment Aggregate Metrics	2.46e+01	100 %	5.07e+08
flash	2.46e+01	100 %	5.07e+08
driver_evolveflash	1.41e+01	57.5%	4.46e+08
driver_initflash	1.04e+01	42.5%	6.02e+07
grid_initdomain	8.58e+00	34.9%	3.45e+07
gr_expanddomain	8.58e+00	34.9%	3.45e+07
loop at gr_expandDomain.F90: 119	6.85e+00	27.9%	3.42e+07
amr_refine_derefine	5.56e+00	22.6%	2.87e+06
amr_morton_process	5.45e+00	22.2%	9.75e+05
find_surrblks	5.18e+00	21.1%	8.40e+05
local_tree_build	5.18e+00	21.1%	8.25e+05
loop at local_tree_build.F90: 211	5.18e+00	21.1%	8.25e+05
loop at local_tree_build.F90: 216	5.18e+00	21.1%	8.25e+05
loop at local_tree_build.F90: 286	1.14e+00	4.6%	2.55e+05
pmpi_sendrecv_replace	5.47e-01	2.2%	5.00e+04

significant scaling losses caused by passing data around a ring of processors

Improved Flash Scaling of AMR Setup

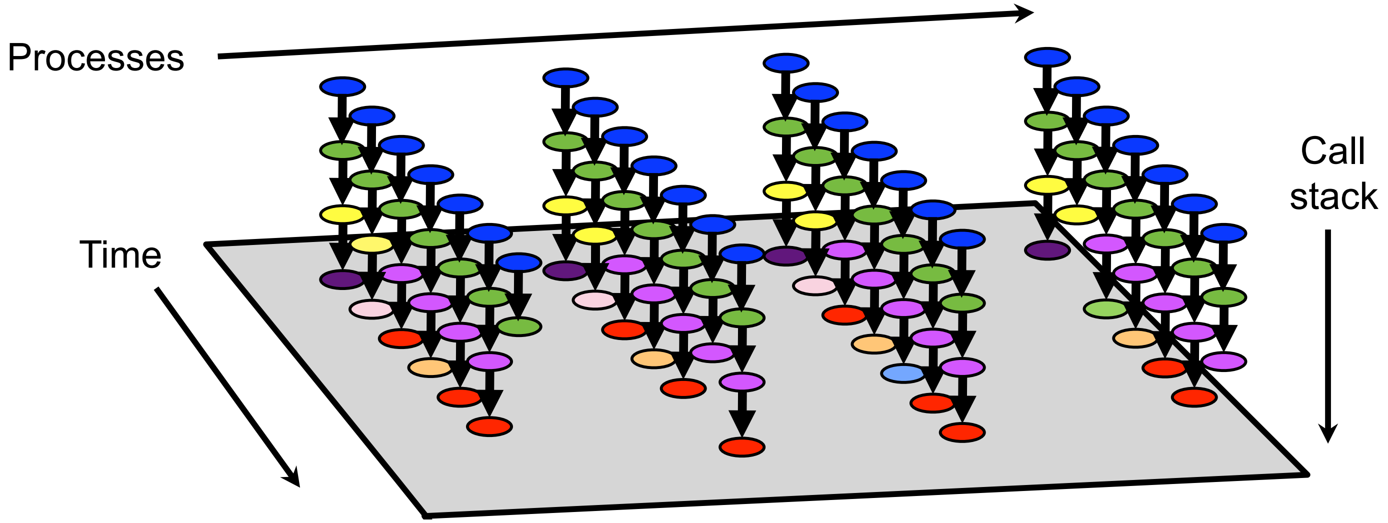


Graph courtesy of Anshu Dubey, U Chicago

Understanding Temporal Behavior

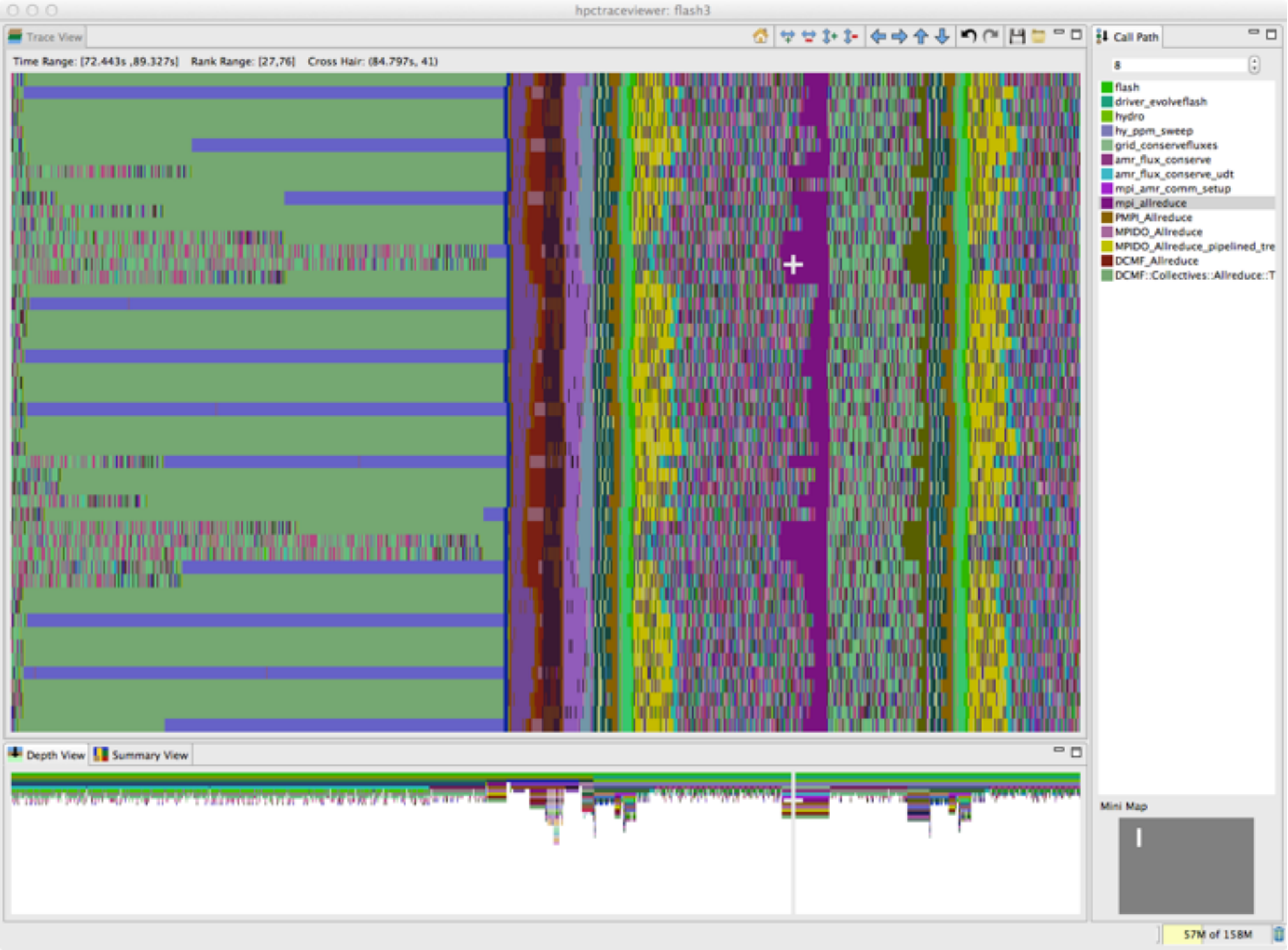
- **Profiling compresses out the temporal dimension**
  - —temporal patterns, e.g. serialization, are invisible in profiles
- **What can we do? Trace call path samples**
  - —sketch:
    - N times per second, take a call path sample of each thread
    - organize the samples for each thread along a time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

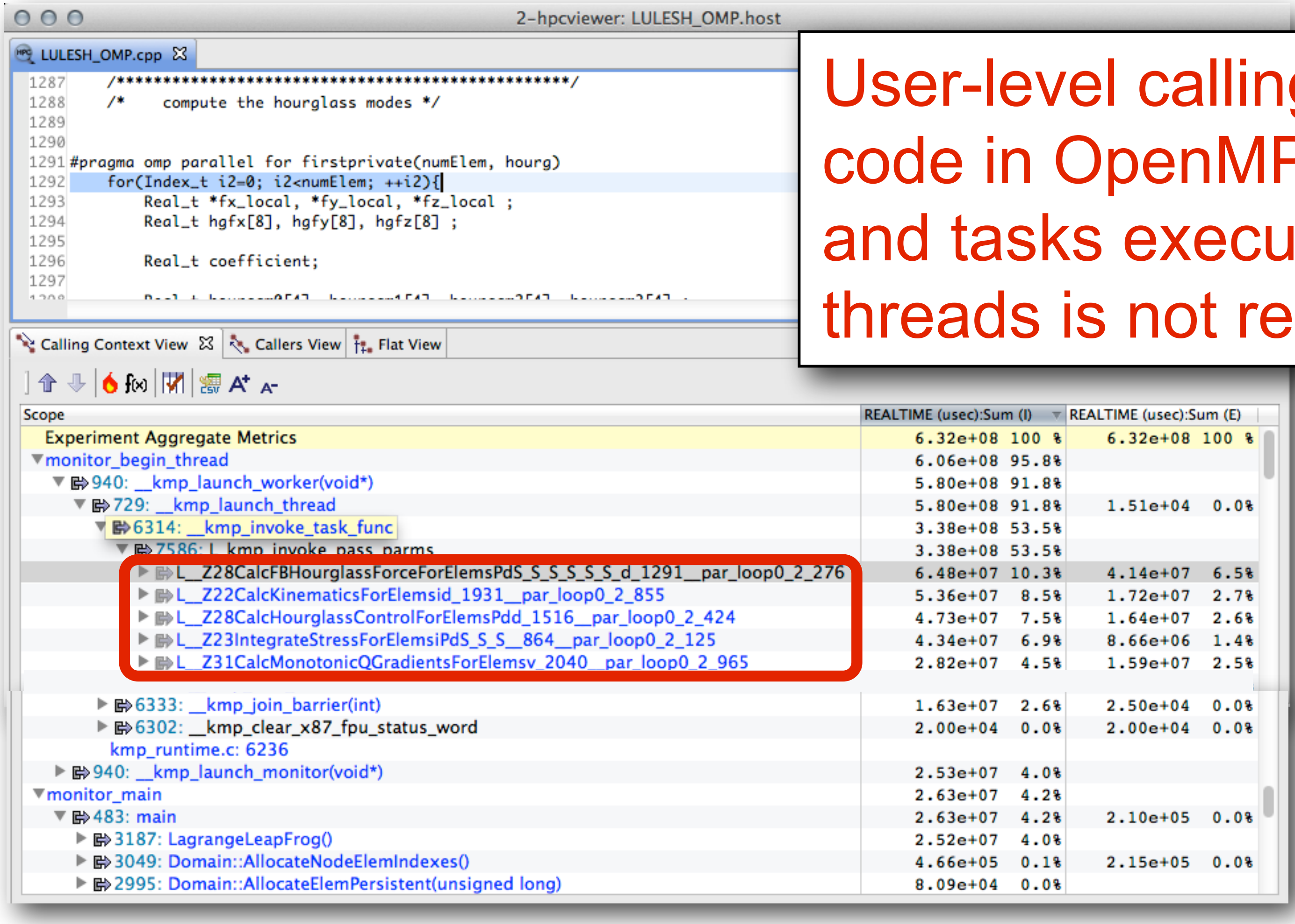
Time-centric analysis: load imbalance among threads appears as different lengths of colored bands along the x axis



Time Range: [72.443s ,89.327s] Rank Range: [27,76] Cross Hair: (84.797s, 41)

OpenMP: A Challenge for Tools

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



```
    /*****************************************************/
    /*    compute the hourglass modes */


#pragma omp parallel for firstprivate(numElem, hourg)
    for(Index_t i2=0; i2<numElem; ++i2){
        Real_t *fx_local, *fy_local, *fz_local ;
        Real_t hgfx[8], hgfy[8], hgfz[8] ;

        Real_t coefficient;

```

Scope	REALTIME (usec):Sum (I)	REALTIME (usec):Sum (E)
Experiment Aggregate Metrics	6.32e+08 100 %	6.32e+08 100 %
monitor_begin_thread	6.06e+08 95.8%	
940: __kmp_launch_worker(void*)	5.80e+08 91.8%	
729: __kmp_launch_thread	5.80e+08 91.8%	1.51e+04 0.0%
6314: __kmp_invoke_task_func	3.38e+08 53.5%	
7586: L_kmp_invoke_pass_parms	3.38e+08 53.5%	
L__Z28CalcFBHourglassForceForElemsPdS_S_S_S_S_S_d_1291__par_loop0_2_276	6.48e+07 10.3%	4.14e+07 6.5%
L__Z22CalcKinematicsForElemsid_1931__par_loop0_2_855	5.36e+07 8.5%	1.72e+07 2.7%
L__Z28CalcHourglassControlForElemsPdd_1516__par_loop0_2_424	4.73e+07 7.5%	1.64e+07 2.6%
L__Z23IntegrateStressForElemsiPdS_S_S__864__par_loop0_2_125	4.34e+07 6.9%	8.66e+06 1.4%
L__Z31CalcMonotonicQGradientsForElemsv_2040__par_loop0_2_965	2.82e+07 4.5%	1.59e+07 2.5%
6333: __kmp_join_barrier(int)	1.63e+07 2.6%	2.50e+04 0.0%
6302: __kmp_clear_x87_fpu_status_word	2.00e+04 0.0%	2.00e+04 0.0%
kmp_runtime.c: 6236		
940: __kmp_launch_monitor(void*)	2.53e+07 4.0%	
monitor_main	2.63e+07 4.2%	
483: main	2.63e+07 4.2%	2.10e+05 0.0%
3187: LagrangeLeapFrog()	2.52e+07 4.0%	
3049: Domain::AllocateNodeElemIndexes()	4.66e+05 0.1%	2.15e+05 0.0%
2995: Domain::AllocateElemPersistent(unsigned long)	8.09e+04 0.0%	

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

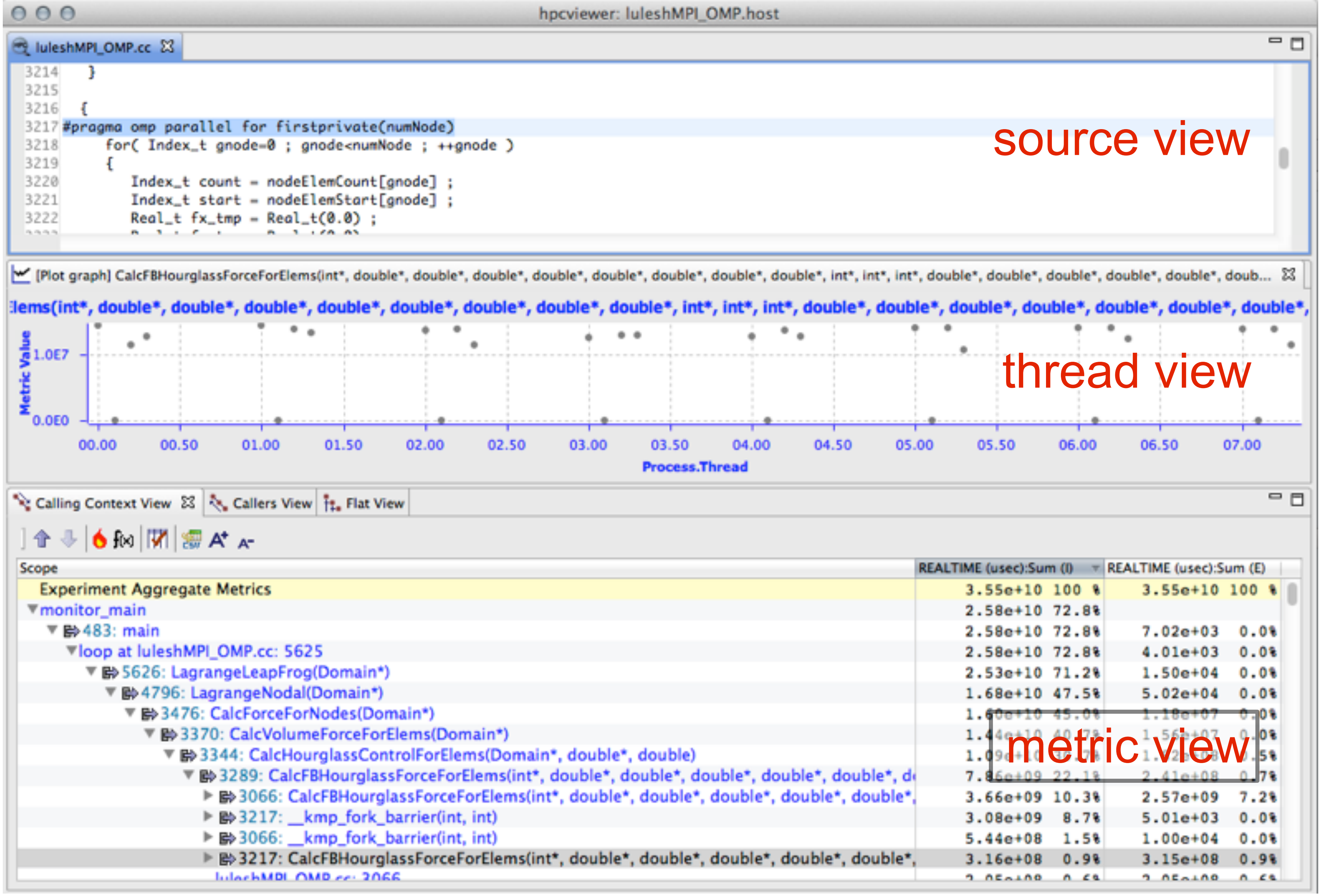
- **Tools provide implementation-level view of OpenMP threads**
  - asymmetric threads
    - master thread
    - worker thread
  - run-time frames are interspersed with user code
- **Hard to understand causes of idleness**
  - long serial sections
  - load imbalance in parallel regions
  - waiting for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



```
3214    }
3215
3216   {
3217 #pragma omp parallel for firstprivate(numNode)
3218      for( Index_t gnode=0 ; gnode<numNode ; ++gnode )
3219      {
3220         Index_t count = nodeElemCount[gnode] ;
3221         Index_t start = nodeElemStart[gnode] ;
3222         Real_t fx_tmp = Real_t(0.0) ;
```

source view

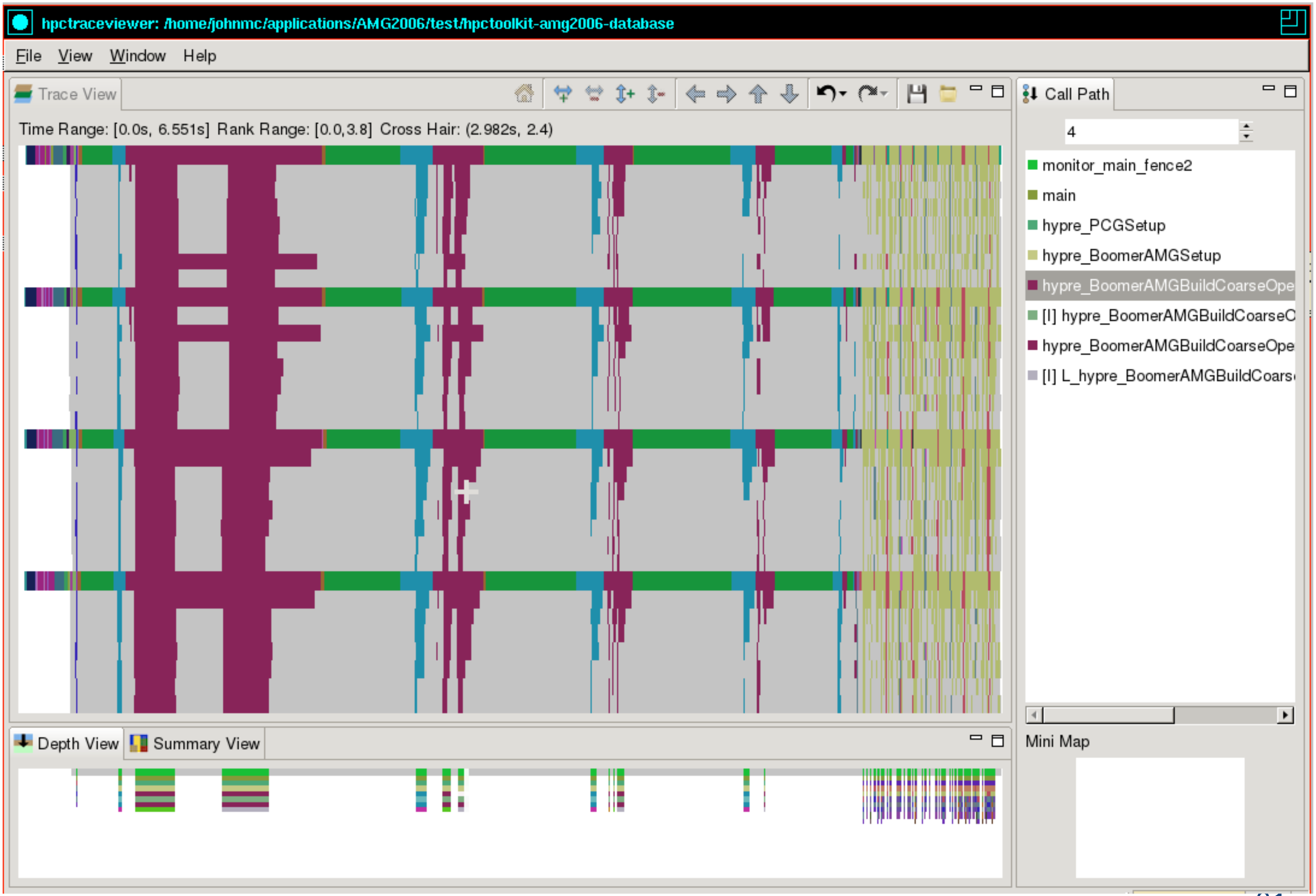
thread view

Scope	REALTIME (usec):Sum (I)	REALTIME (usec):Sum (E)
Experiment Aggregate Metrics	3.55e+10 100 %	3.55e+10 100 %
monitor_main	2.58e+10 72.8%	
483: main	2.58e+10 72.8%	7.02e+03 0.0%
loop at luleshMPI_OMP.cc: 5625	2.58e+10 72.8%	4.01e+03 0.0%
5626: LagrangeLeapFrog(Domain*)	2.53e+10 71.2%	1.50e+04 0.0%
4796: LagrangeNodal(Domain*)	1.68e+10 47.5%	5.02e+04 0.0%
3476: CalcForceForNodes(Domain*)	1.60e+10 45.0%	1.18e+07 0.0%
3370: CalcVolumeForceForElems(Domain*)	1.44e+10 40.7%	1.56e+07 0.0%
3344: CalcHourglassControlForElems(Domain*, double*, double)	1.09e+10 30.7%	[illegible] 0.5%
3289: CalcFBHourglassForceForElems(int*, double*, double*, double*, double*, double*, d	7.86e+09 22.1%	2.41e+08 0.7%
3066: CalcFBHourglassForceForElems(int*, double*, double*, double*, double*, double*,	3.66e+09 10.3%	2.57e+09 7.2%
3217: __kmp_fork_barrier(int, int)	3.08e+09 8.7%	5.01e+03 0.0%
3066: __kmp_fork_barrier(int, int)	5.44e+08 1.5%	1.00e+04 0.0%
3217: CalcFBHourglassForceForElems(int*, double*, double*, double*, double*, double*,	3.16e+08 0.9%	3.15e+08 0.9%
luleshMPI_OMP.cc: 3066	2.05e+08 0.6%	2.05e+08 0.6%

metric view

2 18-core Haswell
4 MPI ranks
6+3 threads per rank

Case Study: AMG2006



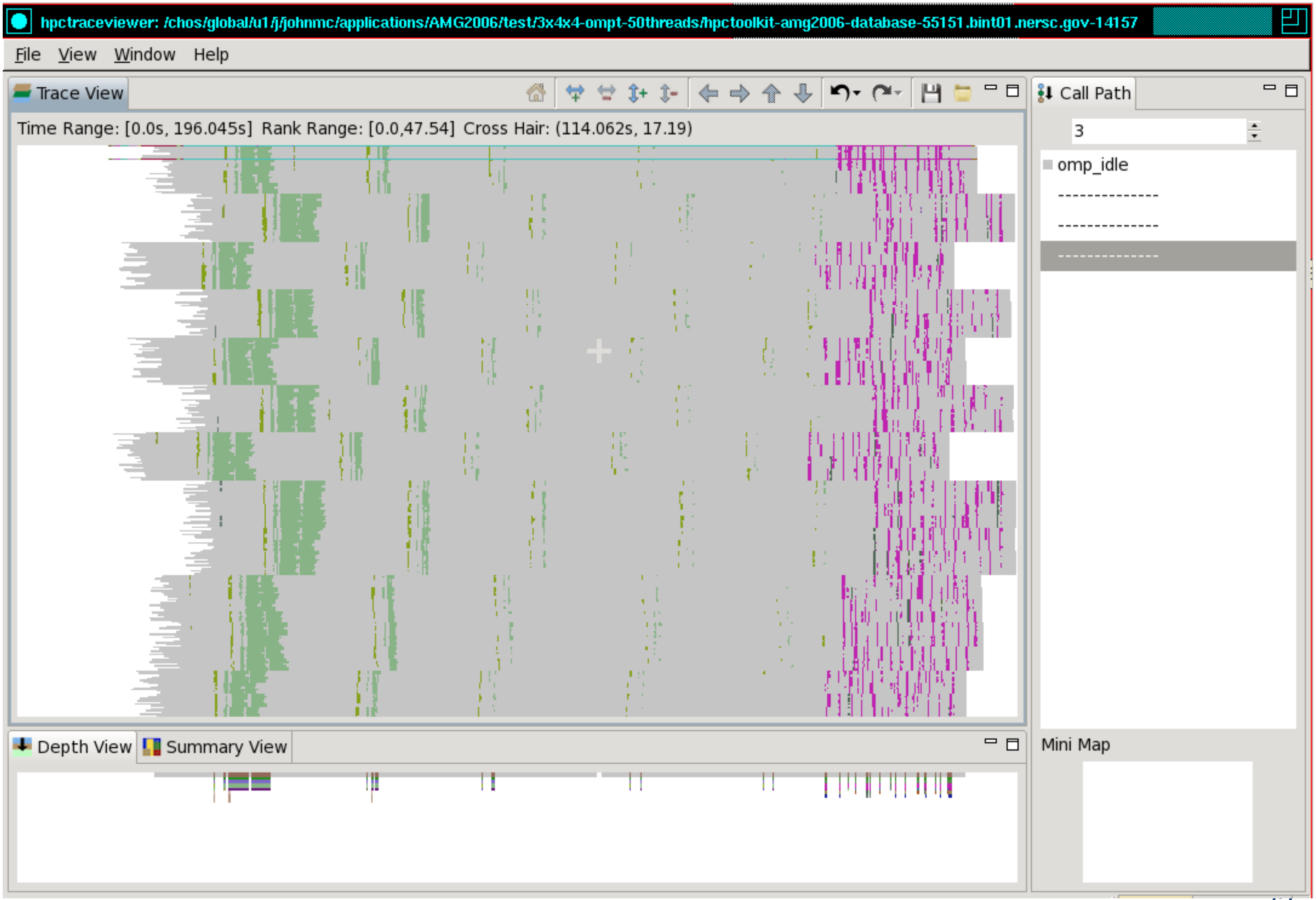
Time Range: [0.0s, 6.551s] Rank Range: [0.0,3.8] Cross Hair: (2.982s, 2.4)

4

- 🟩 monitor_main_fence2
- 🟩 main
- 🟩 hypre_PCGSetup
- 🟨 hypre_BoomerAMGSetup
- 🟪 hypre_BoomerAMGBuildCoarseOpe
- 🟩 [I] hypre_BoomerAMGBuildCoarseC
- 🟪 hypre_BoomerAMGBuildCoarseOpe
- ⬜ [I] L_hypre_BoomerAMGBuildCoars

12 nodes on Babbage@NERSC
24 Xeon Phi
48 MPI ranks
50+5 threads per rank

Case Study: AMG2006


hpctraceviewer: /chos/global/u1/j/johnmc/applications/AMG2006/test/3x4x4-ompt-50threads/hpctoolkit-amg2006-database-55151.bint01.nersc.gov-14157

Time Range: [0.0s, 196.045s] Rank Range: [0.0,47.54] Cross Hair: (114.062s, 17.19)

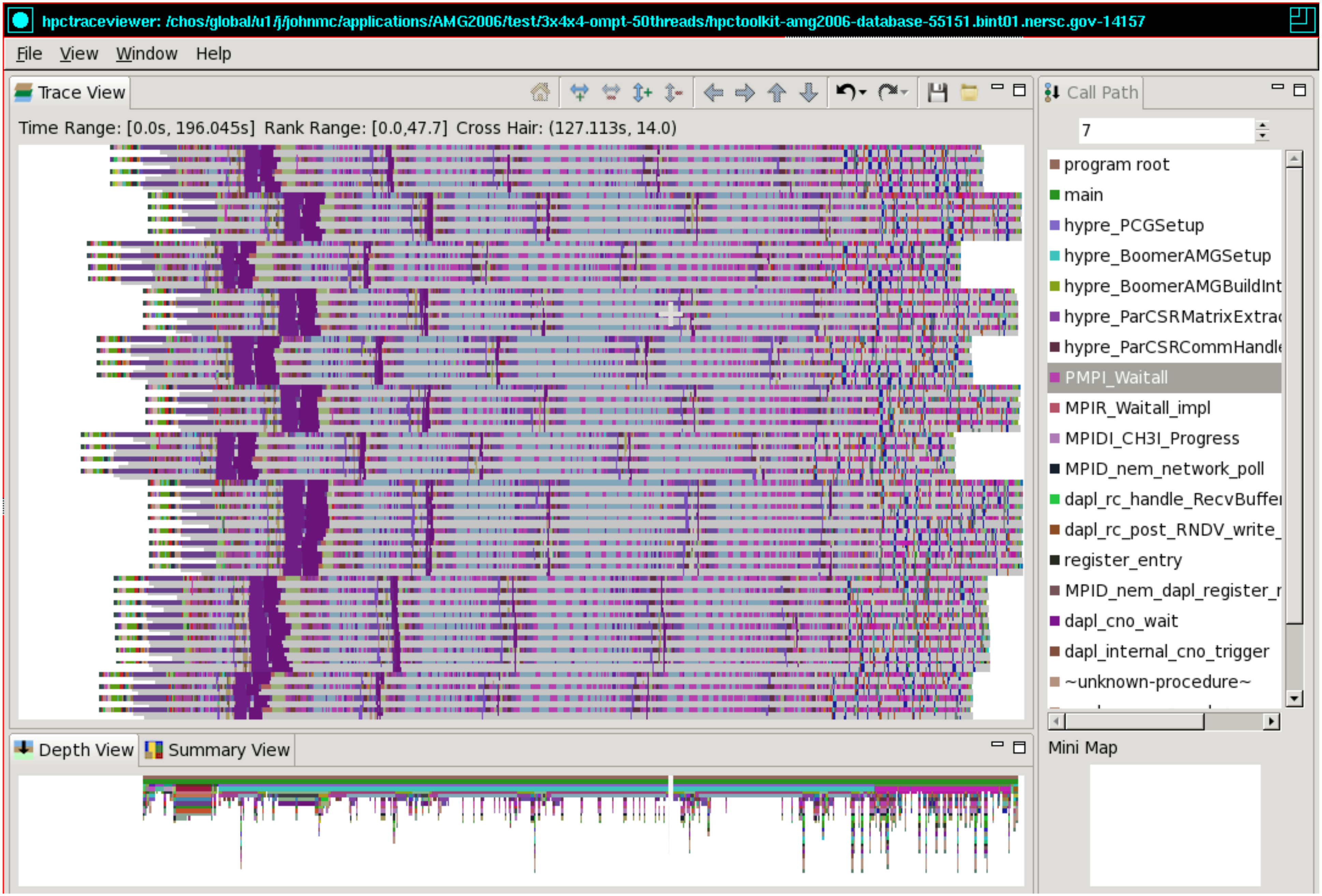
Call Path: 3

omp_idle

12 nodes on Babbage@NERSC
24 Xeon Phi
48 MPI ranks
50+5 threads per rank

Case Study: AMG2006

Slice
Thread 0 from each MPI rank
First two OpenMP workers


hpctraceviewer: /chos/global/u1/j/johnmc/applications/AMG2006/test/3x4x4-ompt-50threads/hpctoolkit-amg2006-database-55151.bint01.nersc.gov-14157

Time Range: [0.0s, 196.045s] Rank Range: [0.0,47.7] Cross Hair: (127.113s, 14.0)

7

- program root
- main
- hypre_PCGSetup
- hypre_BoomerAMGSetup
- hypre_BoomerAMGBuildInt
- hypre_ParCSRMatrixExtra
- hypre_ParCSRCommHandle
- PMPI_Waitall
- MPIR_Waitall_impl
- MPIDI_CH3I_Progress
- MPID_nem_network_poll
- dapl_rc_handle_RecvBuffer
- dapl_rc_post_RNDV_write_
- register_entry
- MPID_nem_dapl_register_r
- dapl_cno_wait
- dapl_internal_cno_trigger
- ~unknown-procedure~

Blame-shifting: Analyze Thread Performance

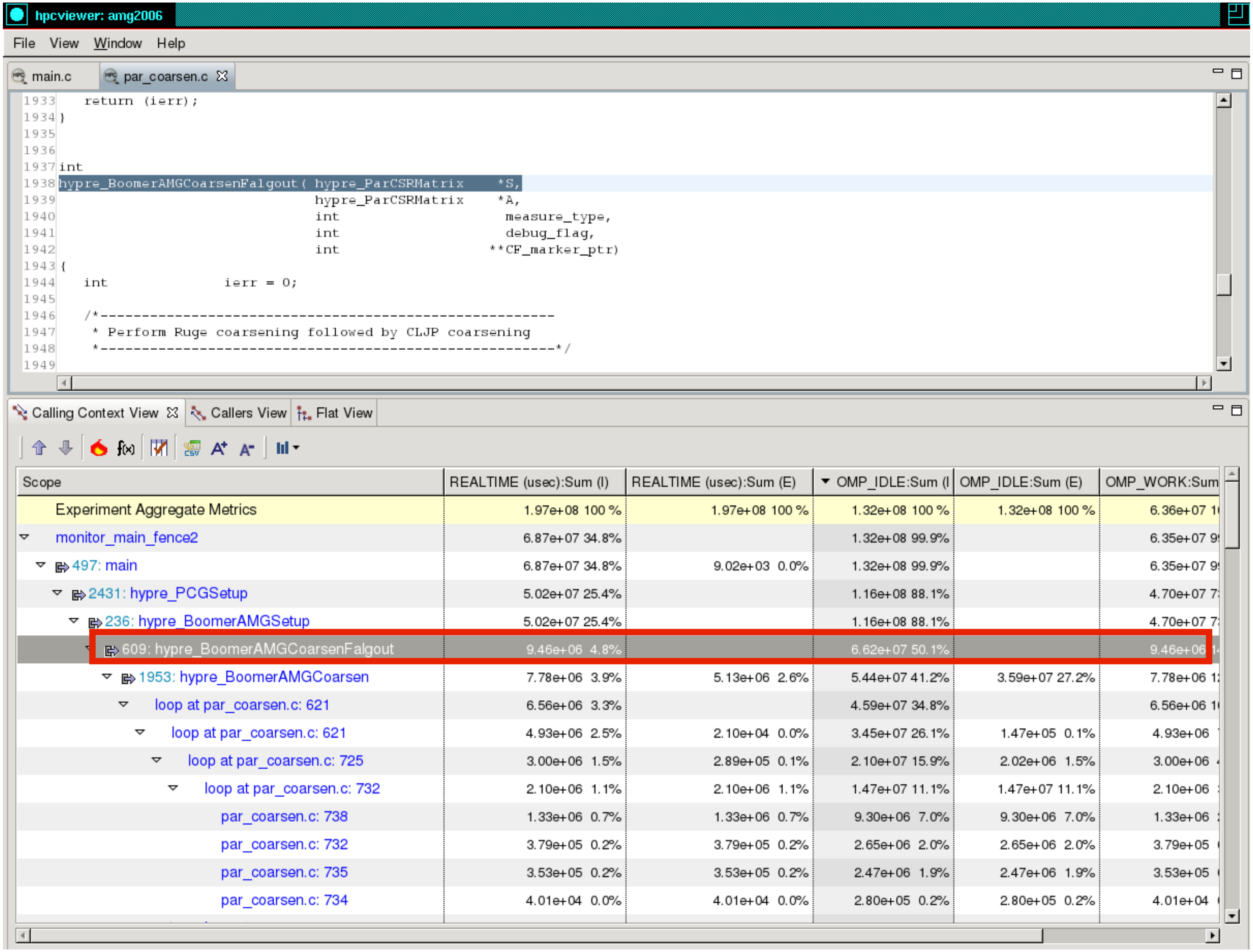
	Problem	Approach
Undirected Blame Shifting[1,3]	A thread is idle waiting for work	Apportion blame among working threads for not shedding enough parallelism to keep all threads busy
Directed Blame Shifting[2,3]	A thread is idle waiting for a mutex	Blame the thread holding the mutex for idleness of threads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

[1] Tallent & Mellor-Crummey: PPoPP 2009

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

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

Blame Shifting: Idleness in AMG2006



```
   return (ierr);
}


int
hypre_BoomerAMGCoarsenFalgout( hypre_ParCSRMatrix    *S,
                               hypre_ParCSRMatrix    *A,
                               int                    measure_type,
                               int                    debug_flag,
                               int                  **CF_marker_ptr)
{
   int              ierr = 0;

   /*------------------------------------------------------
    * Perform Ruge coarsening followed by CLJP coarsening
    *------------------------------------------------------*/

```

Scope	REALTIME (usec):Sum (I)	REALTIME (usec):Sum (E)	OMP_IDLE:Sum (I	OMP_IDLE:Sum (E)	OMP_WORK:Sum
Experiment Aggregate Metrics	1.97e+08 100 %	1.97e+08 100 %	1.32e+08 100 %	1.32e+08 100 %	6.36e+07
monitor_main_fence2	6.87e+07 34.8%		1.32e+08 99.9%		6.35e+07
497: main	6.87e+07 34.8%	9.02e+03 0.0%	1.32e+08 99.9%		6.35e+07
2431: hypre_PCGSetup	5.02e+07 25.4%		1.16e+08 88.1%		4.70e+07
236: hypre_BoomerAMGSetup	5.02e+07 25.4%		1.16e+08 88.1%		4.70e+07
609: hypre_BoomerAMGCoarsenFalgout	9.46e+06 4.8%		6.62e+07 50.1%		9.46e+06
1953: hypre_BoomerAMGCoarsen	7.78e+06 3.9%	5.13e+06 2.6%	5.44e+07 41.2%	3.59e+07 27.2%	7.78e+06
loop at par_coarsen.c: 621	6.56e+06 3.3%		4.59e+07 34.8%		6.56e+06
loop at par_coarsen.c: 621	4.93e+06 2.5%	2.10e+04 0.0%	3.45e+07 26.1%	1.47e+05 0.1%	4.93e+06
loop at par_coarsen.c: 725	3.00e+06 1.5%	2.89e+05 0.1%	2.10e+07 15.9%	2.02e+06 1.5%	3.00e+06
loop at par_coarsen.c: 732	2.10e+06 1.1%	2.10e+06 1.1%	1.47e+07 11.1%	1.47e+07 11.1%	2.10e+06
par_coarsen.c: 738	1.33e+06 0.7%	1.33e+06 0.7%	9.30e+06 7.0%	9.30e+06 7.0%	1.33e+06
par_coarsen.c: 732	3.79e+05 0.2%	3.79e+05 0.2%	2.65e+06 2.0%	2.65e+06 2.0%	3.79e+05
par_coarsen.c: 735	3.53e+05 0.2%	3.53e+05 0.2%	2.47e+06 1.9%	2.47e+06 1.9%	3.53e+05
par_coarsen.c: 734	4.01e+04 0.0%	4.01e+04 0.0%	2.80e+05 0.2%	2.80e+05 0.2%	4.01e+04

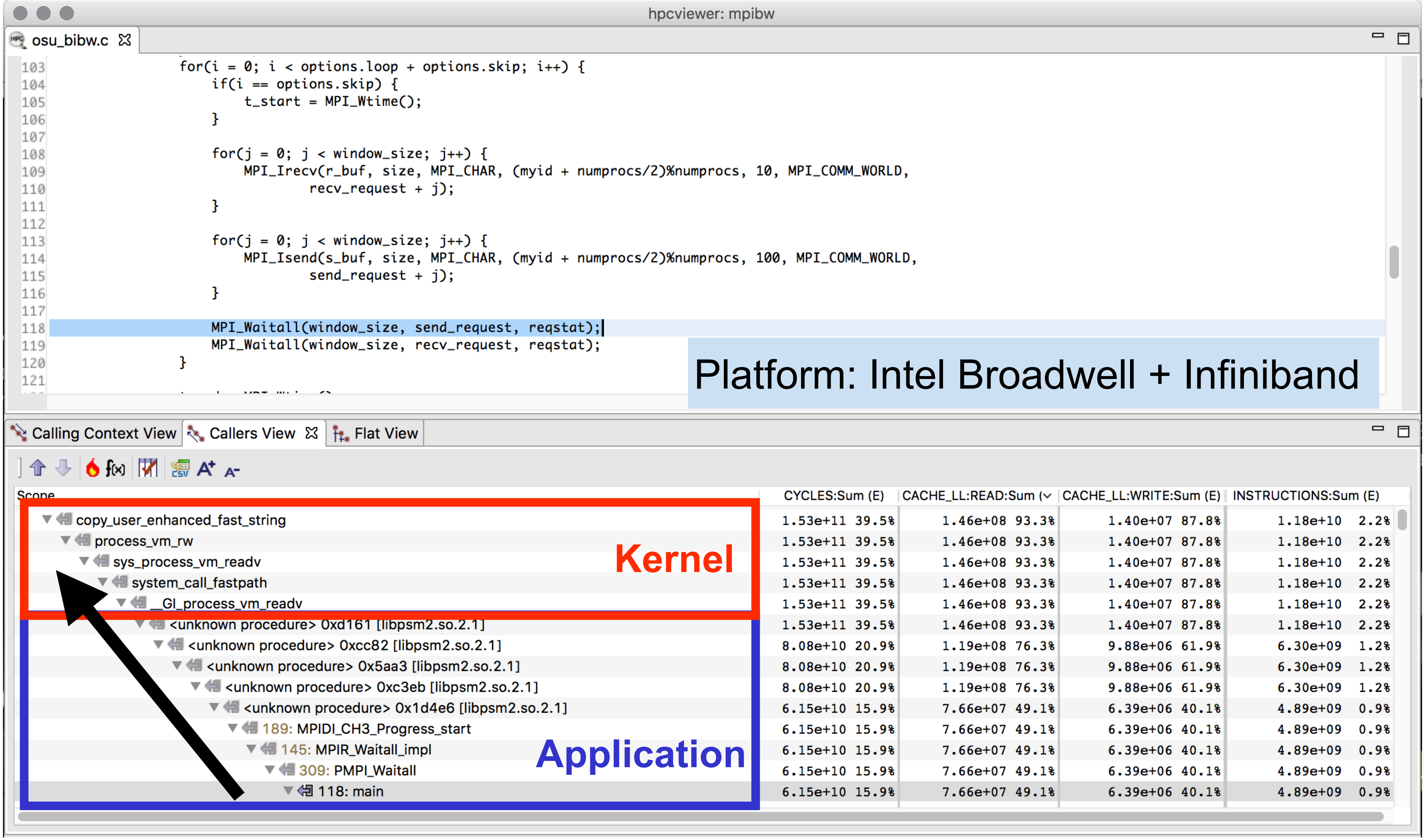
OpenMP Tool API Status

- **Currently HPCToolkit supports OMPT interface based on OpenMP TR2 (April 2014)**
- **Migrating to emerging OpenMP 5.0 (preview, Nov 2016)**
- **OMPT prototype implementations**
  - **LLVM (current: OpenMP 5)**
    - **interoperable with GNU, Intel compilers**
  - **IBM LOMP (currently targets OpenMP 5)**
- **Ongoing work**
  - **refining OpenMP 5.0 definition of OMPT**
  - **refining OpenMP 5.0 OMPT support in LLVM**
  - **refining HPCToolkit OMPT to match emerging standard**

Emerging Capabilities in Brief

Monitoring Application + Kernel

Sampling call stacks into the kernel



```
103                for(i = 0; i < options.loop + options.skip; i++) {
104                    if(i == options.skip) {
105                        t_start = MPI_Wtime();
106                    }
107
108                    for(j = 0; j < window_size; j++) {
109                        MPI_Irecv(r_buf, size, MPI_CHAR, (myid + numprocs/2)%numprocs, 10, MPI_COMM_WORLD,
110                                recv_request + j);
111                    }
112
113                    for(j = 0; j < window_size; j++) {
114                        MPI_Isend(s_buf, size, MPI_CHAR, (myid + numprocs/2)%numprocs, 100, MPI_COMM_WORLD,
115                                send_request + j);
116                    }
117
118                    MPI_Waitall(window_size, send_request, reqstat);
119                    MPI_Waitall(window_size, recv_request, reqstat);
120                }
121
```

Platform: Intel Broadwell + Infiniband

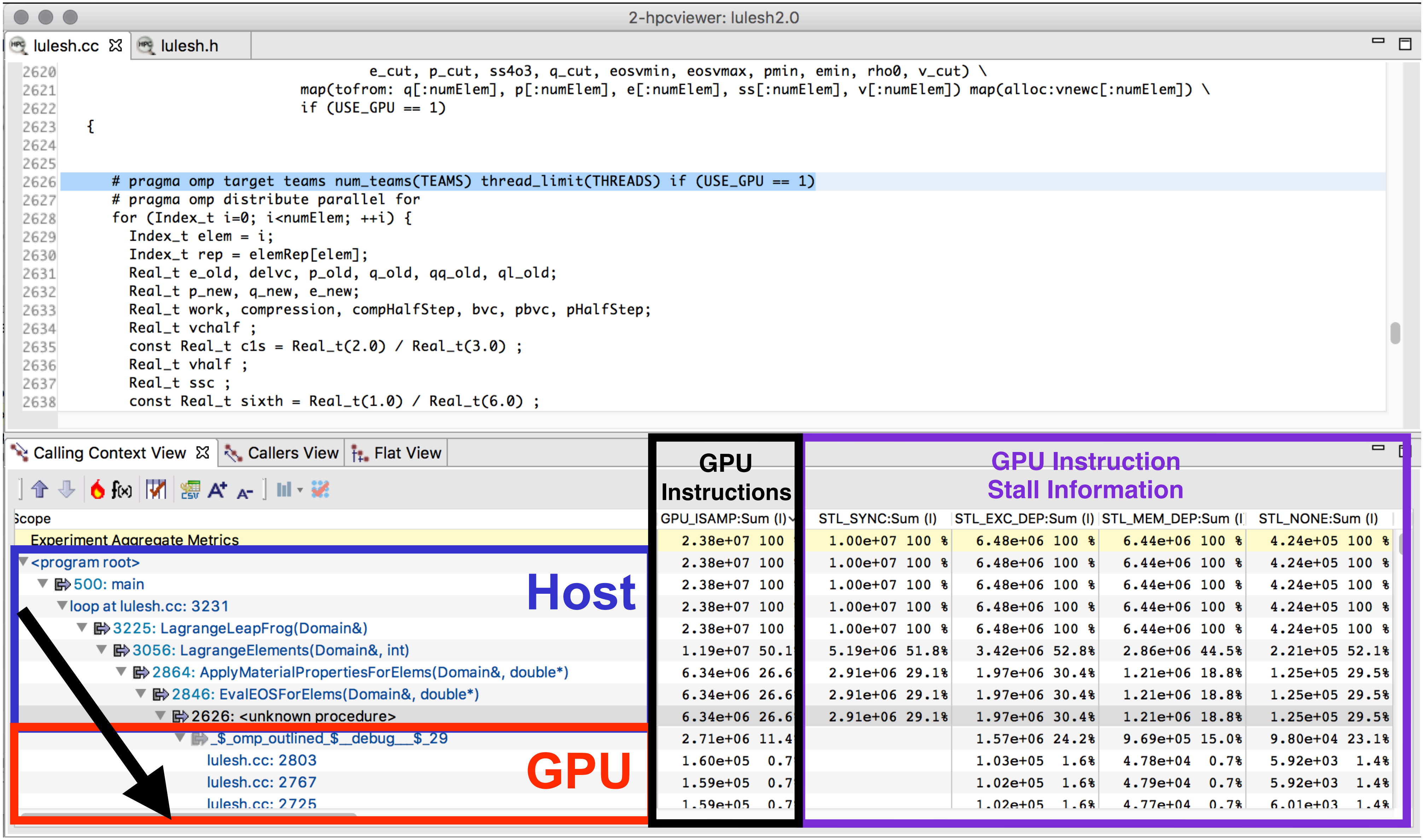
Scope	CYCLES:Sum (E)	CACHE_LL:READ:Sum (	CACHE_LL:WRITE:Sum (E)	INSTRUCTIONS:Sum (E)
copy_user_enhanced_fast_string	1.53e+11 39.5%	1.46e+08 93.3%	1.40e+07 87.8%	1.18e+10 2.2%
process_vm_rw	1.53e+11 39.5%	1.46e+08 93.3%	1.40e+07 87.8%	1.18e+10 2.2%
sys_process_vm_readv	1.53e+11 39.5%	1.46e+08 93.3%	1.40e+07 87.8%	1.18e+10 2.2%
system_call_fastpath	1.53e+11 39.5%	1.46e+08 93.3%	1.40e+07 87.8%	1.18e+10 2.2%
__GI_process_vm_readv	1.53e+11 39.5%	1.46e+08 93.3%	1.40e+07 87.8%	1.18e+10 2.2%
<unknown procedure> 0xd161 [libpsm2.so.2.1]	1.53e+11 39.5%	1.46e+08 93.3%	1.40e+07 87.8%	1.18e+10 2.2%
<unknown procedure> 0xcc82 [libpsm2.so.2.1]	8.08e+10 20.9%	1.19e+08 76.3%	9.88e+06 61.9%	6.30e+09 1.2%
<unknown procedure> 0x5aa3 [libpsm2.so.2.1]	8.08e+10 20.9%	1.19e+08 76.3%	9.88e+06 61.9%	6.30e+09 1.2%
<unknown procedure> 0xc3eb [libpsm2.so.2.1]	8.08e+10 20.9%	1.19e+08 76.3%	9.88e+06 61.9%	6.30e+09 1.2%
<unknown procedure> 0x1d4e6 [libpsm2.so.2.1]	6.15e+10 15.9%	7.66e+07 49.1%	6.39e+06 40.1%	4.89e+09 0.9%
189: MPIDI_CH3_Progress_start	6.15e+10 15.9%	7.66e+07 49.1%	6.39e+06 40.1%	4.89e+09 0.9%
145: MPIR_Waitall_impl	6.15e+10 15.9%	7.66e+07 49.1%	6.39e+06 40.1%	4.89e+09 0.9%
309: PMPI_Waitall	6.15e+10 15.9%	7.66e+07 49.1%	6.39e+06 40.1%	4.89e+09 0.9%
118: main	6.15e+10 15.9%	7.66e+07 49.1%	6.39e+06 40.1%	4.89e+09 0.9%

Kernel

Application

Monitoring Accelerated OpenMP 5

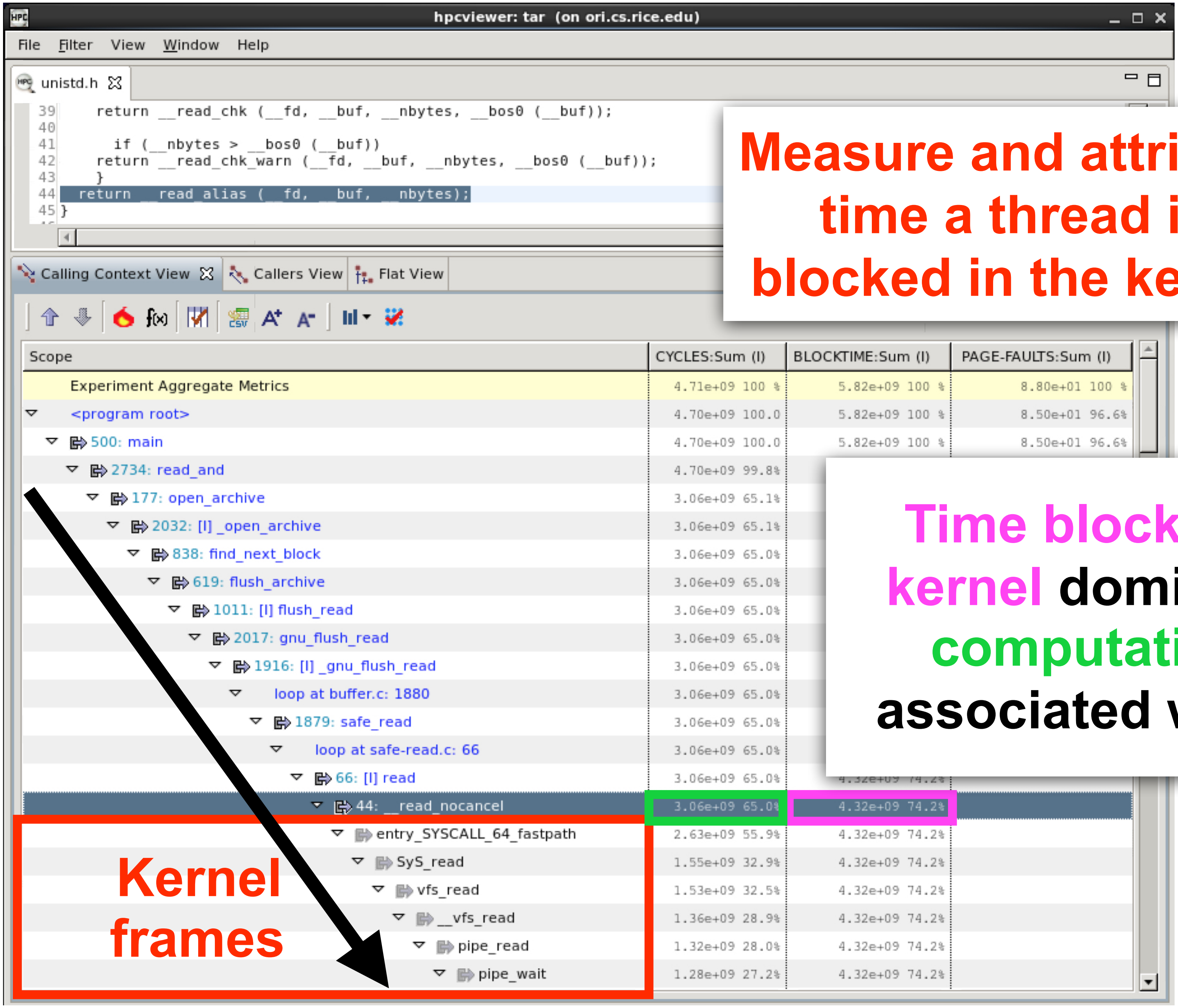
Sampling calling contexts spanning CPU + GPU



```
                                   e_cut, p_cut, ss4o3, q_cut, eosvmin, eosvmax, pmin, emin, rho0, v_cut) \
                           map(tofrom: q[:numElem], p[:numElem], e[:numElem], ss[:numElem], v[:numElem]) map(alloc:vnewc[:numElem]) \
                           if (USE_GPU == 1)
  {


     # pragma omp target teams num_teams(TEAMS) thread_limit(THREADS) if (USE_GPU == 1)
     # pragma omp distribute parallel for
     for (Index_t i=0; i<numElem; ++i) {
       Index_t elem = i;
       Index_t rep = elemRep[elem];
       Real_t e_old, delvc, p_old, q_old, qq_old, ql_old;
       Real_t p_new, q_new, e_new;
       Real_t work, compression, compHalfStep, bvc, pbvc, pHalfStep;
       Real_t vchalf ;
       const Real_t c1s = Real_t(2.0) / Real_t(3.0) ;
       Real_t vhalf ;
       Real_t ssc ;
       const Real_t sixth = Real_t(1.0) / Real_t(6.0) ;
```

Scope	GPU_ISAMP:Sum (I)	STL_SYNC:Sum (I)	STL_EXC_DEP:Sum (I)	STL_MEM_DEP:Sum (I	STL_NONE:Sum (I)
Experiment Aggregate Metrics	2.38e+07 100	1.00e+07 100 %	6.48e+06 100 %	6.44e+06 100 %	4.24e+05 100 %
<program root>	2.38e+07 100	1.00e+07 100 %	6.48e+06 100 %	6.44e+06 100 %	4.24e+05 100 %
500: main	2.38e+07 100	1.00e+07 100 %	6.48e+06 100 %	6.44e+06 100 %	4.24e+05 100 %
loop at lulesh.cc: 3231	2.38e+07 100	1.00e+07 100 %	6.48e+06 100 %	6.44e+06 100 %	4.24e+05 100 %
3225: LagrangeLeapFrog(Domain&)	2.38e+07 100	1.00e+07 100 %	6.48e+06 100 %	6.44e+06 100 %	4.24e+05 100 %
3056: LagrangeElements(Domain&, int)	1.19e+07 50.1	5.19e+06 51.8%	3.42e+06 52.8%	2.86e+06 44.5%	2.21e+05 52.1%
2864: ApplyMaterialPropertiesForElems(Domain&, double*)	6.34e+06 26.6	2.91e+06 29.1%	1.97e+06 30.4%	1.21e+06 18.8%	1.25e+05 29.5%
2846: EvalEOSForElems(Domain&, double*)	6.34e+06 26.6	2.91e+06 29.1%	1.97e+06 30.4%	1.21e+06 18.8%	1.25e+05 29.5%
2626: <unknown procedure>	6.34e+06 26.6	2.91e+06 29.1%	1.97e+06 30.4%	1.21e+06 18.8%	1.25e+05 29.5%
_$_omp_outlined_$__debug__$_29	2.71e+06 11.4		1.57e+06 24.2%	9.69e+05 15.0%	9.80e+04 23.1%
lulesh.cc: 2803	1.60e+05 0.7		1.03e+05 1.6%	4.78e+04 0.7%	5.92e+03 1.4%
lulesh.cc: 2767	1.59e+05 0.7		1.02e+05 1.6%	4.79e+04 0.7%	5.92e+03 1.4%
lulesh.cc: 2725	1.59e+05 0.7		1.02e+05 1.6%	4.77e+04 0.7%	6.01e+03 1.4%

Measuring Thread Blocking



```
    return __read_chk (__fd, __buf, __nbytes, __bos0 (__buf));

      if (__nbytes > __bos0 (__buf))
    return __read_chk_warn (__fd, __buf, __nbytes, __bos0 (__buf));
    }
  return __read_alias (__fd, __buf, __nbytes);
}
```

Scope	CYCLES:Sum (I)	BLOCKTIME:Sum (I)	PAGE-FAULTS:Sum (I)
Experiment Aggregate Metrics	4.71e+09 100 %	5.82e+09 100 %	8.80e+01 100 %
<program root>	4.70e+09 100.0	5.82e+09 100 %	8.50e+01 96.6%
500: main	4.70e+09 100.0	5.82e+09 100 %	8.50e+01 96.6%
2734: read_and	4.70e+09 99.8%		
177: open_archive	3.06e+09 65.1%		
2032: [I] _open_archive	3.06e+09 65.1%		
838: find_next_block	3.06e+09 65.0%		
619: flush_archive	3.06e+09 65.0%		
1011: [I] flush_read	3.06e+09 65.0%		
2017: gnu_flush_read	3.06e+09 65.0%		
1916: [I] _gnu_flush_read	3.06e+09 65.0%		
loop at buffer.c: 1880	3.06e+09 65.0%		
1879: safe_read	3.06e+09 65.0%		
loop at safe-read.c: 66	3.06e+09 65.0%		
66: [I] read	3.06e+09 65.0%	4.32e+09 74.2%	
44: __read_nocancel	3.06e+09 65.0%	4.32e+09 74.2%	
entry_SYSCALL_64_fastpath	2.63e+09 55.9%	4.32e+09 74.2%	
SyS_read	1.55e+09 32.9%	4.32e+09 74.2%	
vfs_read	1.53e+09 32.5%	4.32e+09 74.2%	
__vfs_read	1.36e+09 28.9%	4.32e+09 74.2%	
pipe_read	1.32e+09 28.0%	4.32e+09 74.2%	
pipe_wait	1.28e+09 27.2%	4.32e+09 74.2%	

Measure and attribute time a thread is blocked in the kernel

Time blocked in the kernel dominates the computation time associated with reads

Kernel frames

Other Ongoing Work and Future Plans

- Other ongoing work
  - data-centric analysis: associate costs with variables
    - analysis and attribution of performance to optimized code
  - adding OpenMP parallelism to hpcprof-mpi to accelerate data analysis
  - adding OpenMP parallelism to hpcstruct to accelerate binary analysis
  - automated analysis to deliver performance insights
- Future plans
  - support top-down analysis methods using hardware counters
  - resource-centric performance analysis
    - within and across nodes
  - scale measurement and analysis for exascale

Status

- New binary analyzer for better attribution of performance to source code merged into master this week
- Resolve conflict between Linux perf_events and Cray PAPI module
- Investigate issue measuring counter events related to SIMD performance
- Attribute kernel time to <vmlinux> if kernel symbols are not available
- Cherry-pick OMPT support for CPU and make it available
- We will update HPCToolkit modules on all ALCF systems once these issues are resolved
- We will email participants when new HPCToolkit installations are available

HPCToolkit at ALCF

- **ALCF systems (vesta, cetus)**
  - BG/Q: in your .soft file, add the following line
    - +hpctoolkit-devel
      (this package is always the most up-to-date)
  - Theta
    - module load hpctoolkit
- **Man pages**
  - available but not provided in module on theta
- **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**

Advice for Using HPCToolkit

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 (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
    ```
- **Hint: you can run hpcprof-mpi on the x86_64 vis cluster (cooley)**

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.