Center for Scalable Application Development Software

A Slice of CScADS: Performance Tools for Petascale Platforms

John Mellor-Crummey

Laksono Adhianto, Mike Fagan, Michael Franco, Mark Krentel, Reed Landrum, Xu Liu, Nathan Tallent

Department of Computer Science
Rice University

CScADS Co-PIs and Senior Personnel

John Mellor-Crummey, Keith Cooper

Peter Beckman, Ewing Lusk

Jack Dongarra

Bart Miller

Katherine Yelick

CScADS Mission

- Provide open source software systems, tools, and components that address a spectrum of needs
  - directly usable by application experts
  - support development of enabling technologies by the CS community
- Target architectures of critical interest to DOE
  - Cray XT
  - Blue Gene/P
  - multicore processors in general
- Engage DOE application teams and vendors
- Engage the research community in SciDAC challenges

SciDAC-2 Mission

- Develop comprehensive scientific computing software infrastructure to enable petascale science
- Develop new generation of data management and knowledge discovery tools for large data sets

CScADS Research and Development

Vertical integration across the petascale software stack

- System software for leadership computing platforms
- Communication libraries
- Math libraries
- Open source compilers
- Performance tools and infrastructure
- Application engagement: analysis and tuning
  - e.g., Annual CScADS Workshop on Leadership Computing
    - experts worked with approximately 200 INCITE and SciDAC code developers to help them scale to DOE's largest systems

Key Performance Questions

- Why doesn't my application scale as well as I hoped?
- How can I identify bottlenecks in multithreaded node programs?
- How is my code performing relative to peak performance?
  - if my code is not performing well, what is the nature of its problems?

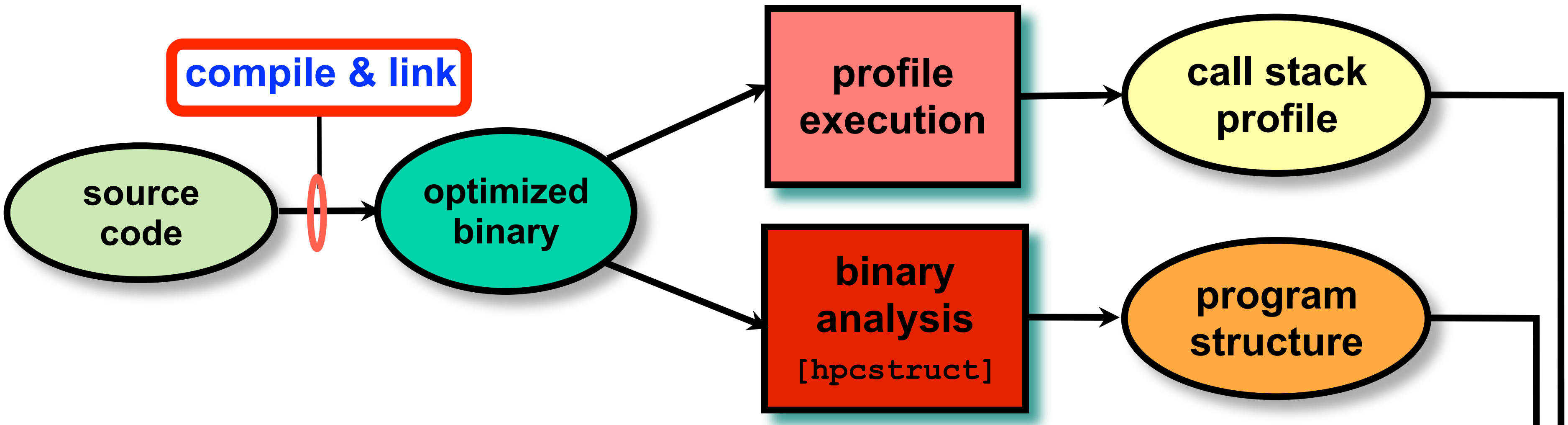
Performance Tool Requirements

- Cope with complex application characteristics
  - large, multi-lingual programs
  - fully optimized code: loop optimization, templates, inlining
  - binary-only libraries, sometimes partially stripped
  - hybrid programs: MPI + OpenMP
- Cope with complex execution environments
  - static or dynamic binaries
  - batch jobs
- Provide effective performance analysis
  - pinpoint and quantify problems
  - yield actionable results
- Scale to leadership computing platforms

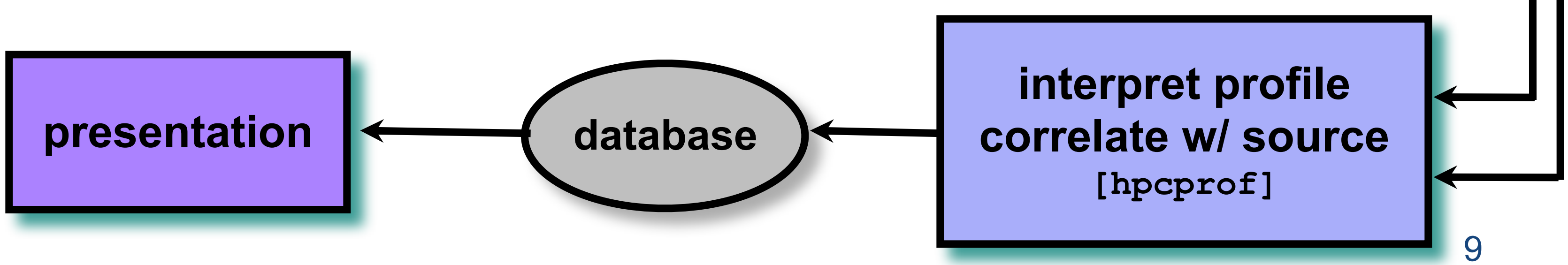
Outline

- Introduction to HPCToolkit
- Five new approaches for analyzing parallel program performance
  - scalability analysis using call path profiles [SC09]
  - blame shifting to analyze lock contention in threaded codes [PPoPP10]
  - pinpointing load imbalance in parallel codes [SC10]
  - understanding temporal dynamics of parallel codes
  - data centric analysis of program performance
- Conclusions

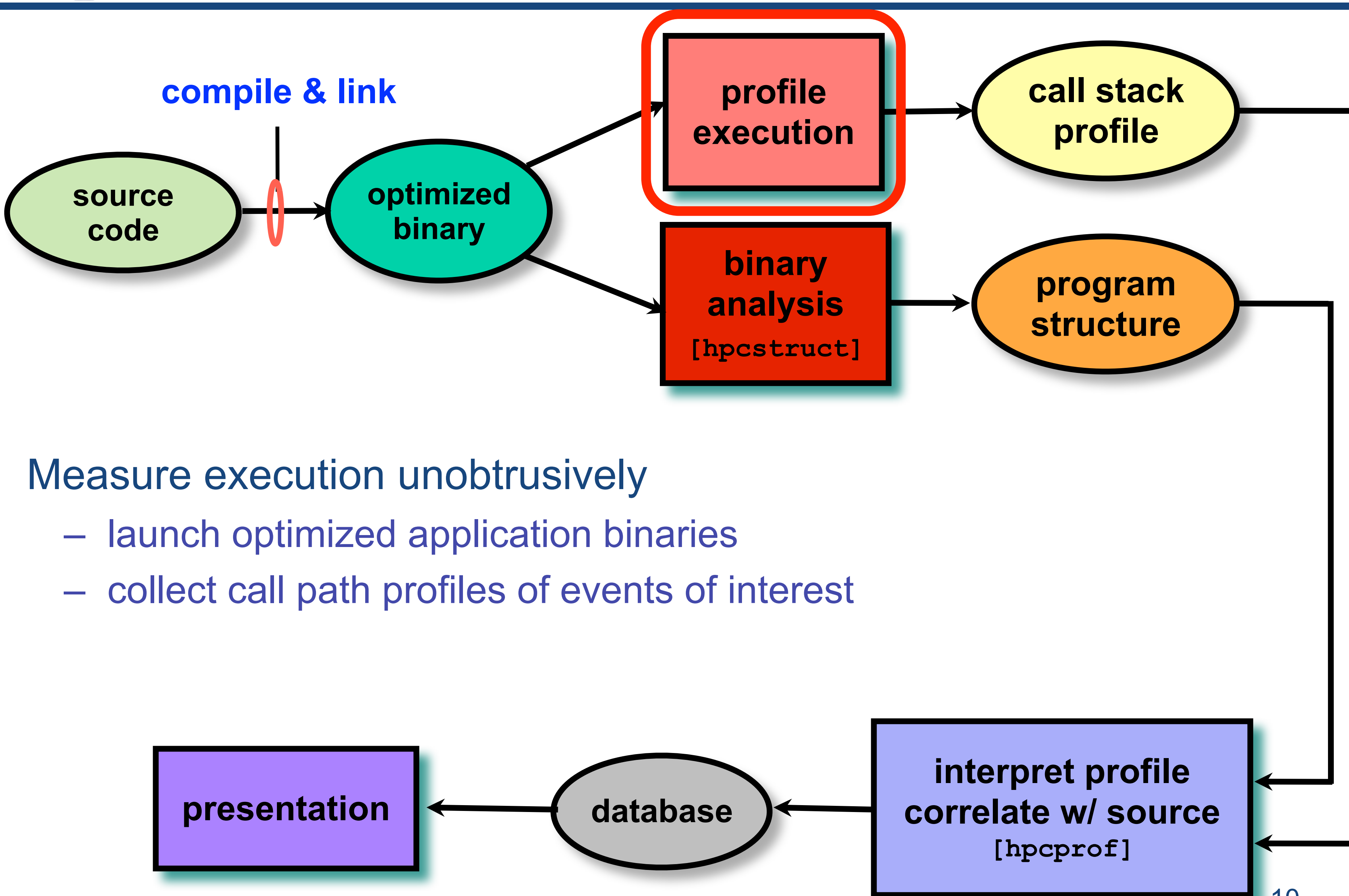
HPCToolkit Performance Tools



- Compile and link for production
  - with full optimization
- For statically-linked executables (e.g. for Cray XT or BG/P)
  - use `hpclink` script to incorporate our monitoring library



HPCToolkit Performance Tools



Measure execution unobtrusively

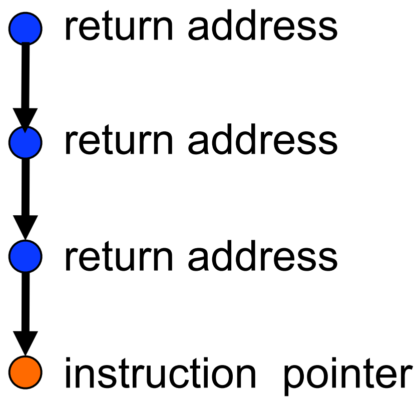
- launch optimized application binaries
- collect call path profiles of events of interest

Call Path Profiling

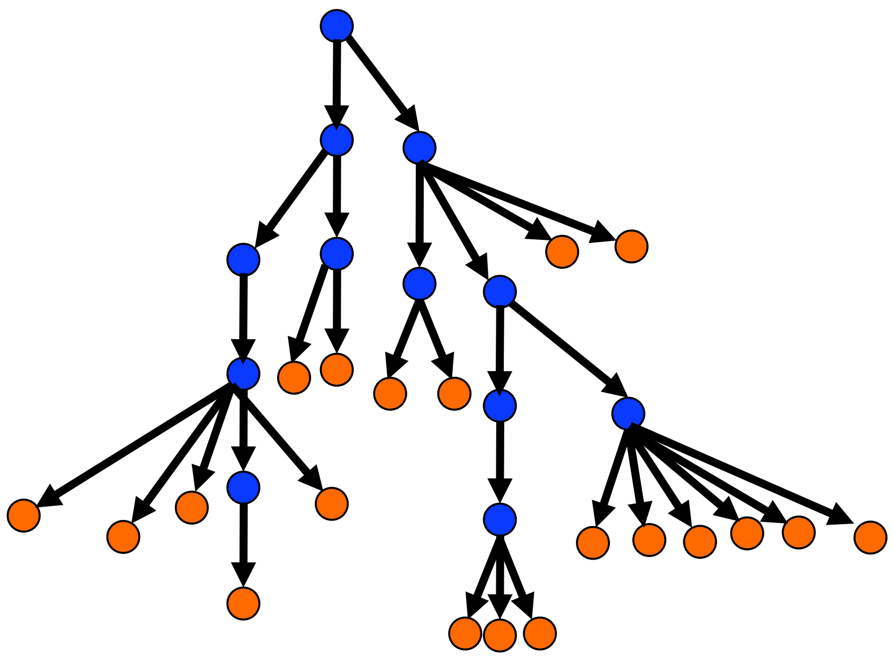
Measure and attribute costs in context

- Sample timer or hardware counter overflows
- Gather calling context using stack unwinding

Call path sample

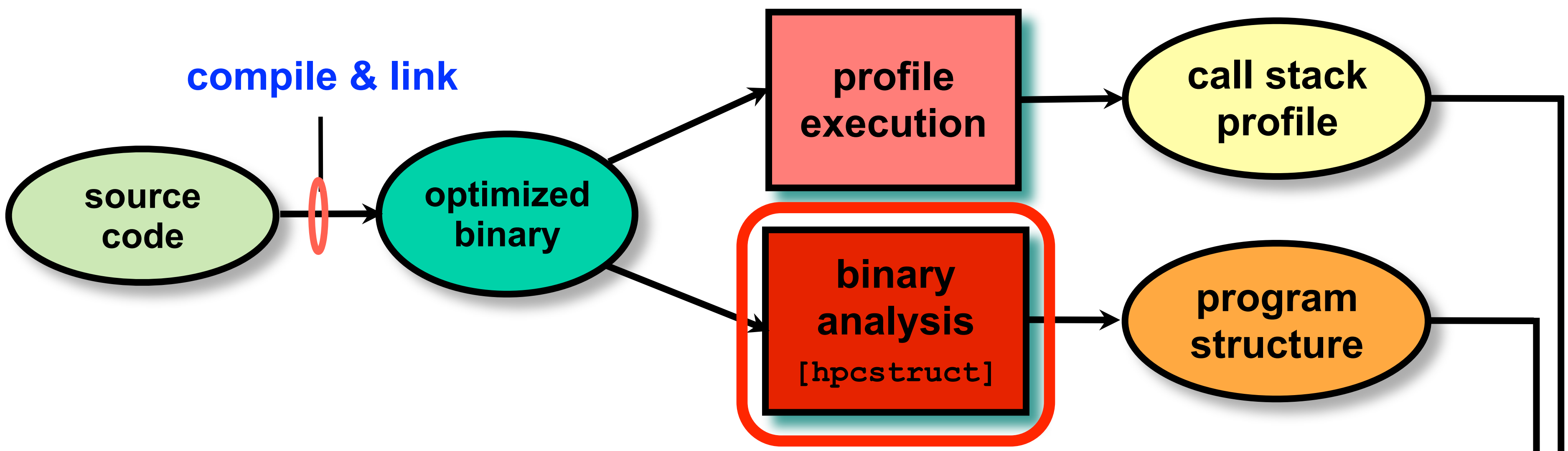


Calling Context Tree (CCT)



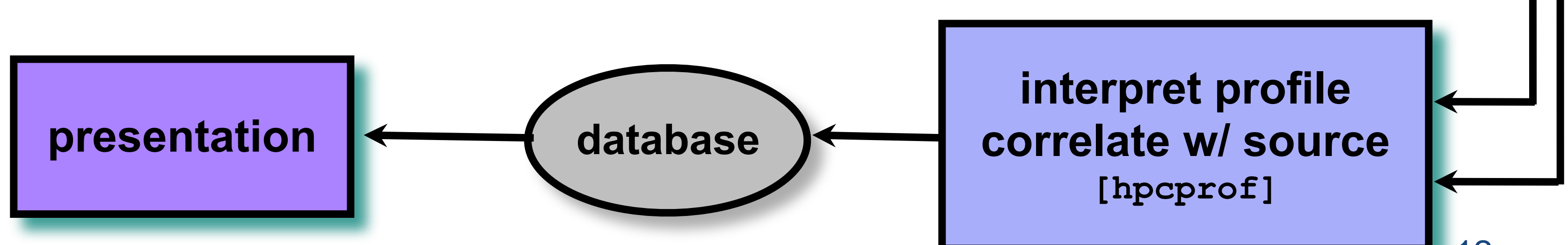
Overhead proportional to sampling frequency ...
... not call frequency

HPCToolkit Performance Tools

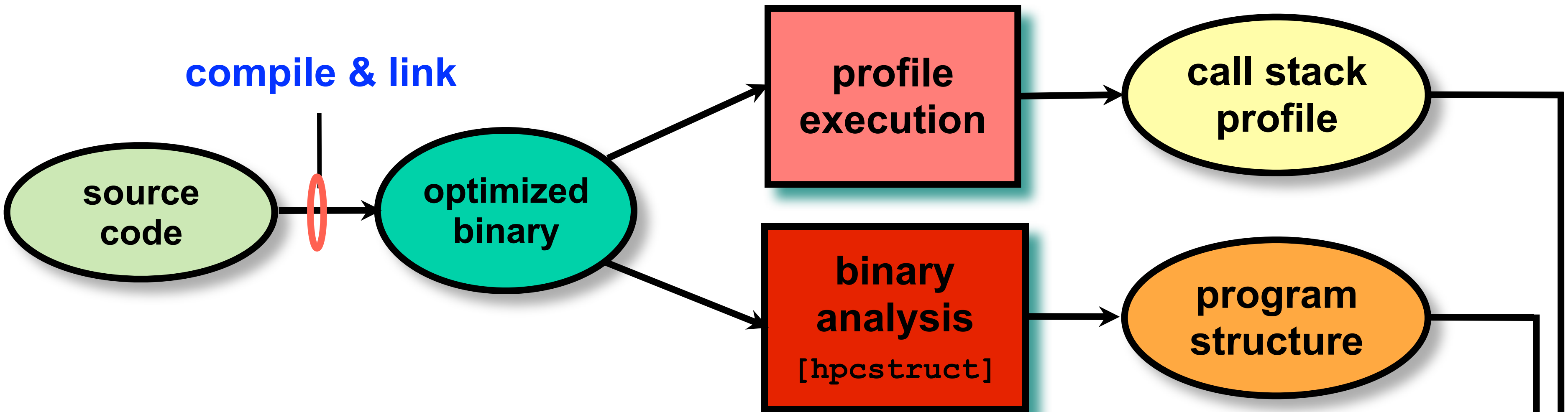


Analyze binary to recover program structure

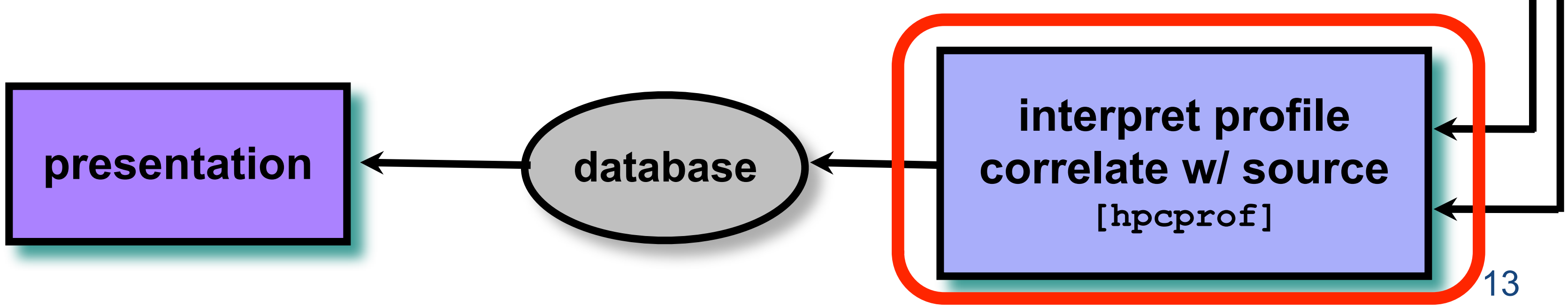
- analyze machine code, line map, and debugging information
- extract loop nesting information and identify inlined procedures
- map transformed loops and procedures back to source



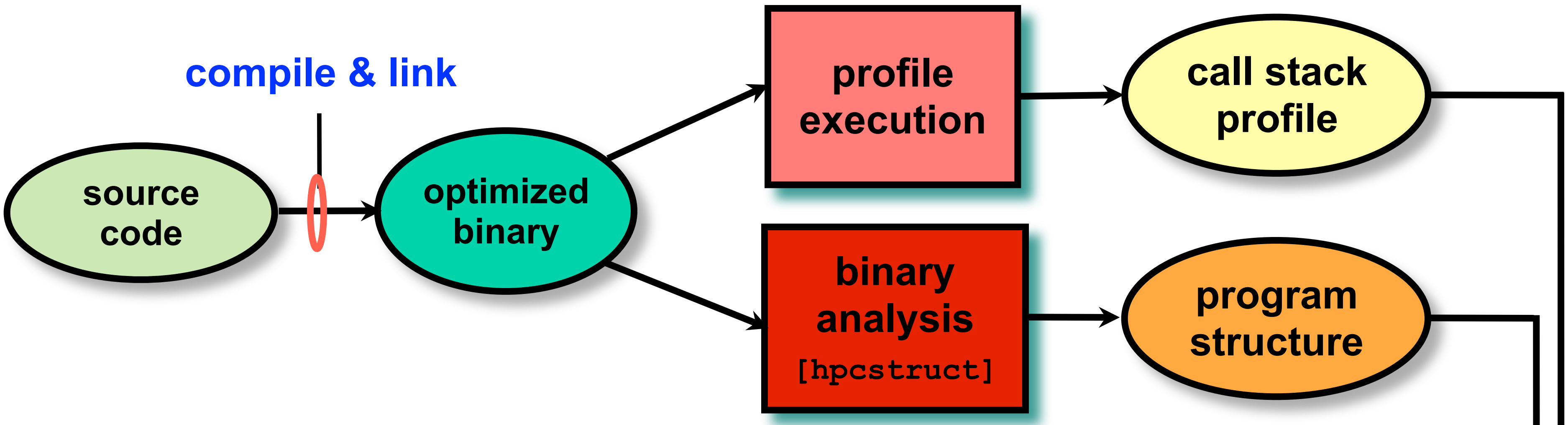
HPCToolkit Performance Tools



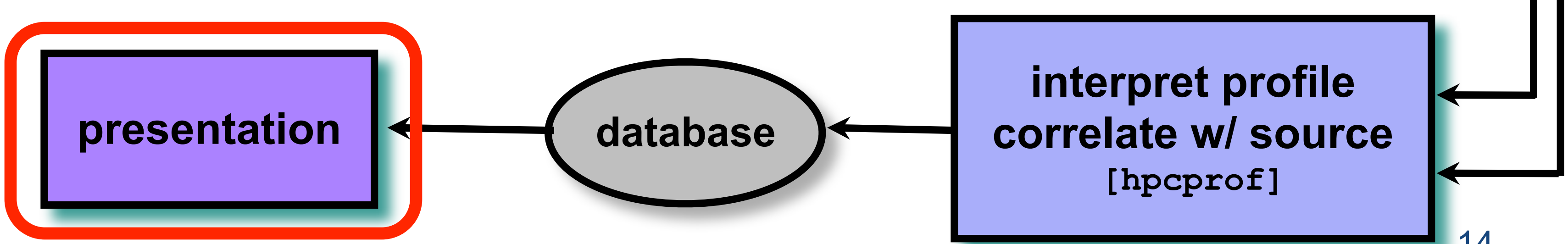
- Combine multiple profiles
  - multiple threads; multiple processes; multiple executions
- Correlate measurements to static and dynamic program structure



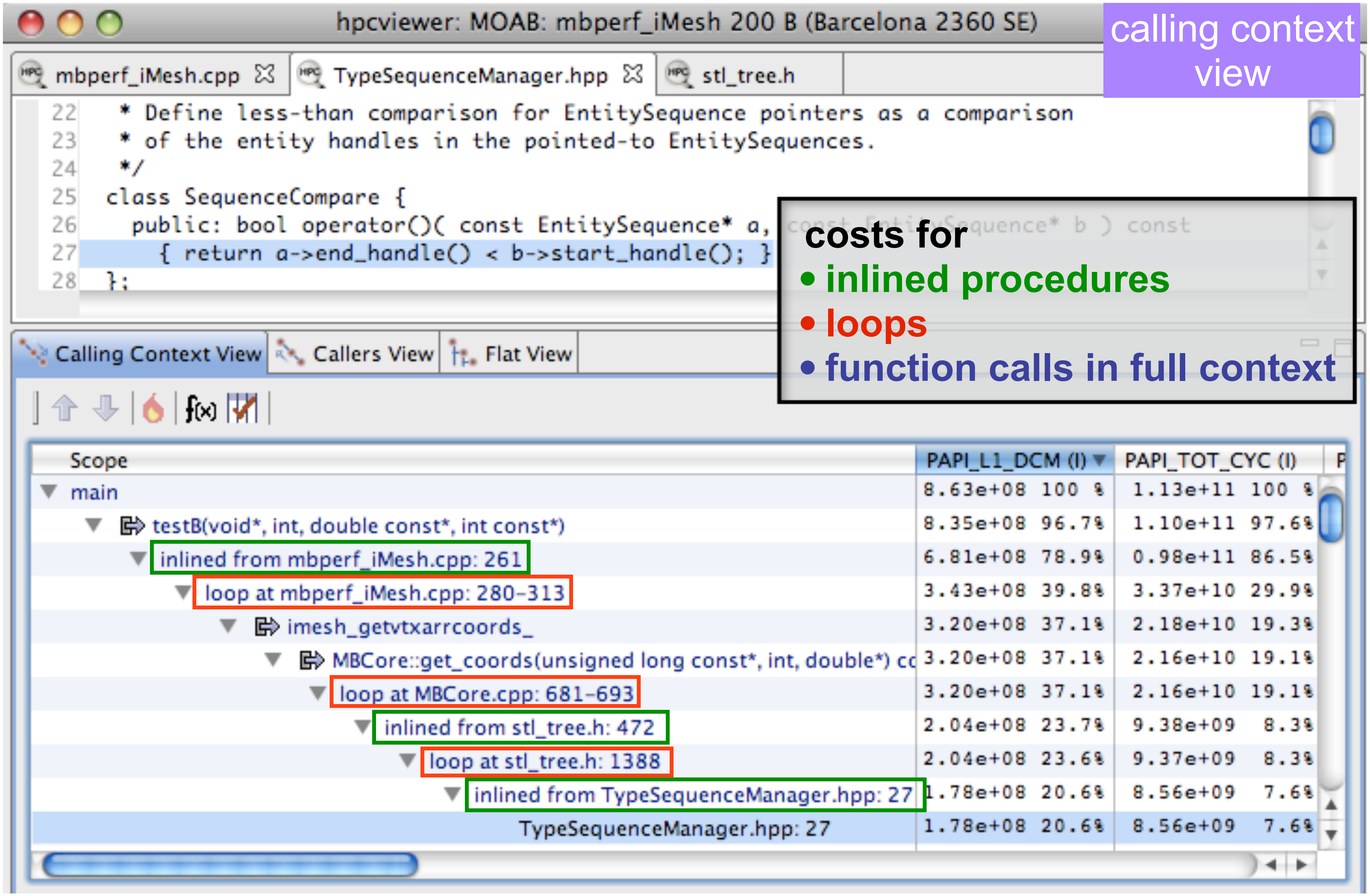
HPCToolkit Performance Tools



- Explore performance data from multiple perspectives
- Rank order by metrics to focus on what’s important
- Compute derived metrics to gain insight
- Explore call stack traces to understand transient behavior



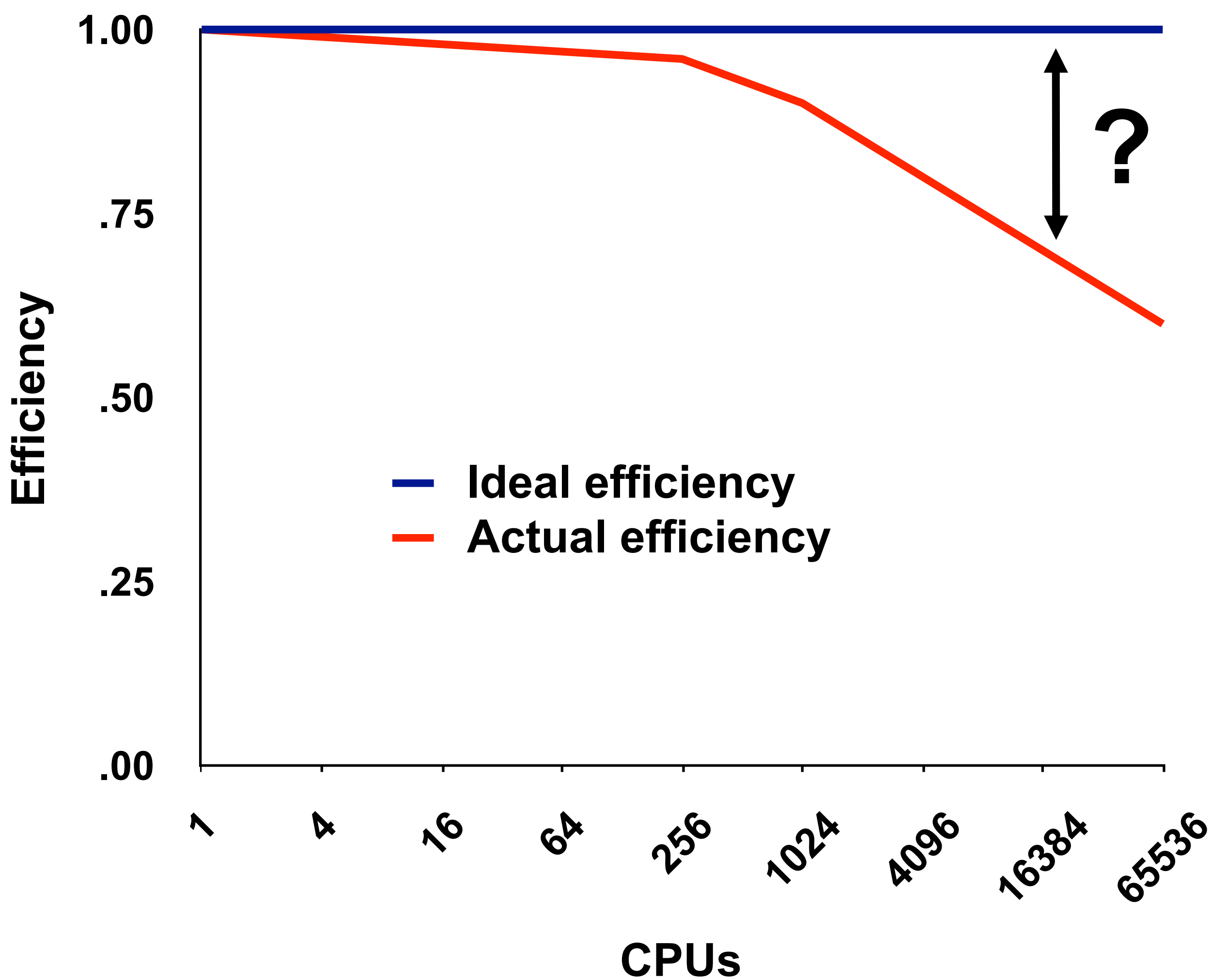
MOAB Mesh Library from ITAPS



Outline

- Introduction to HPCToolkit
- Five new approaches for analyzing parallel program performance
  - scalability analysis using call path profiles [SC09]
  - blame shifting to analyze lock contention in threaded codes [PPoPP10]
  - pinpointing load imbalance in parallel codes [SC10]
  - understanding temporal dynamics of parallel codes
  - data centric analysis of program performance
- Conclusions

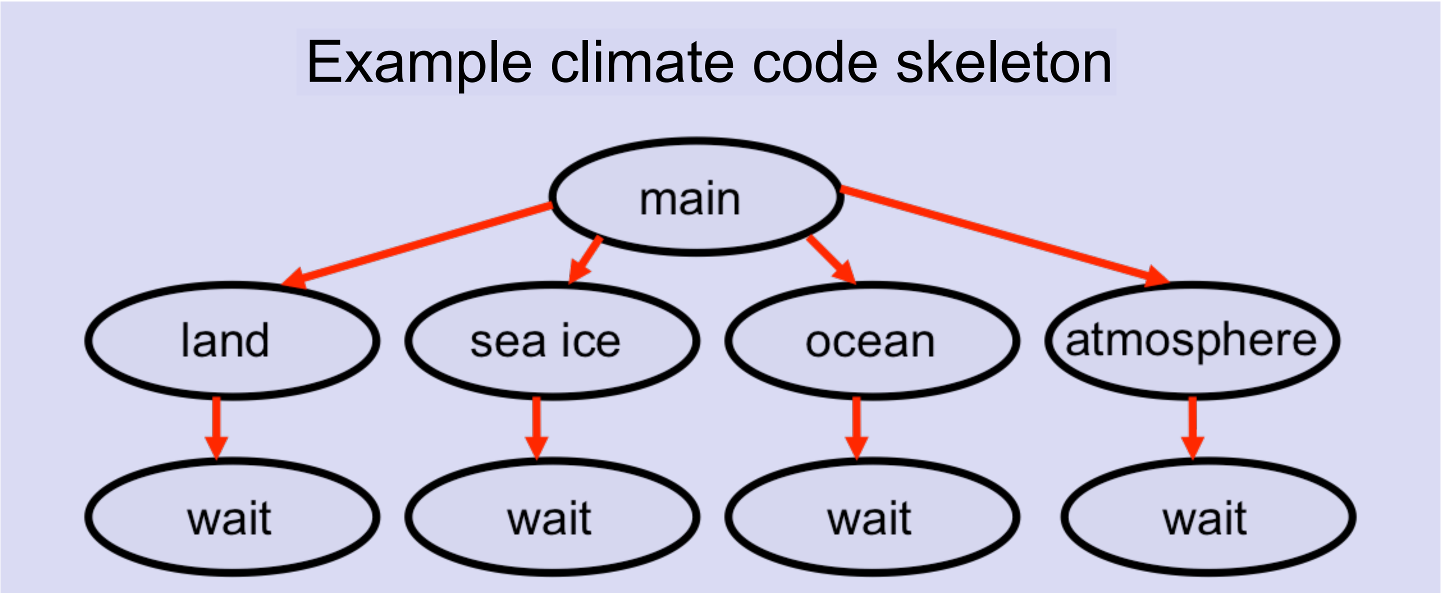
Pinpointing Scalability Bottlenecks



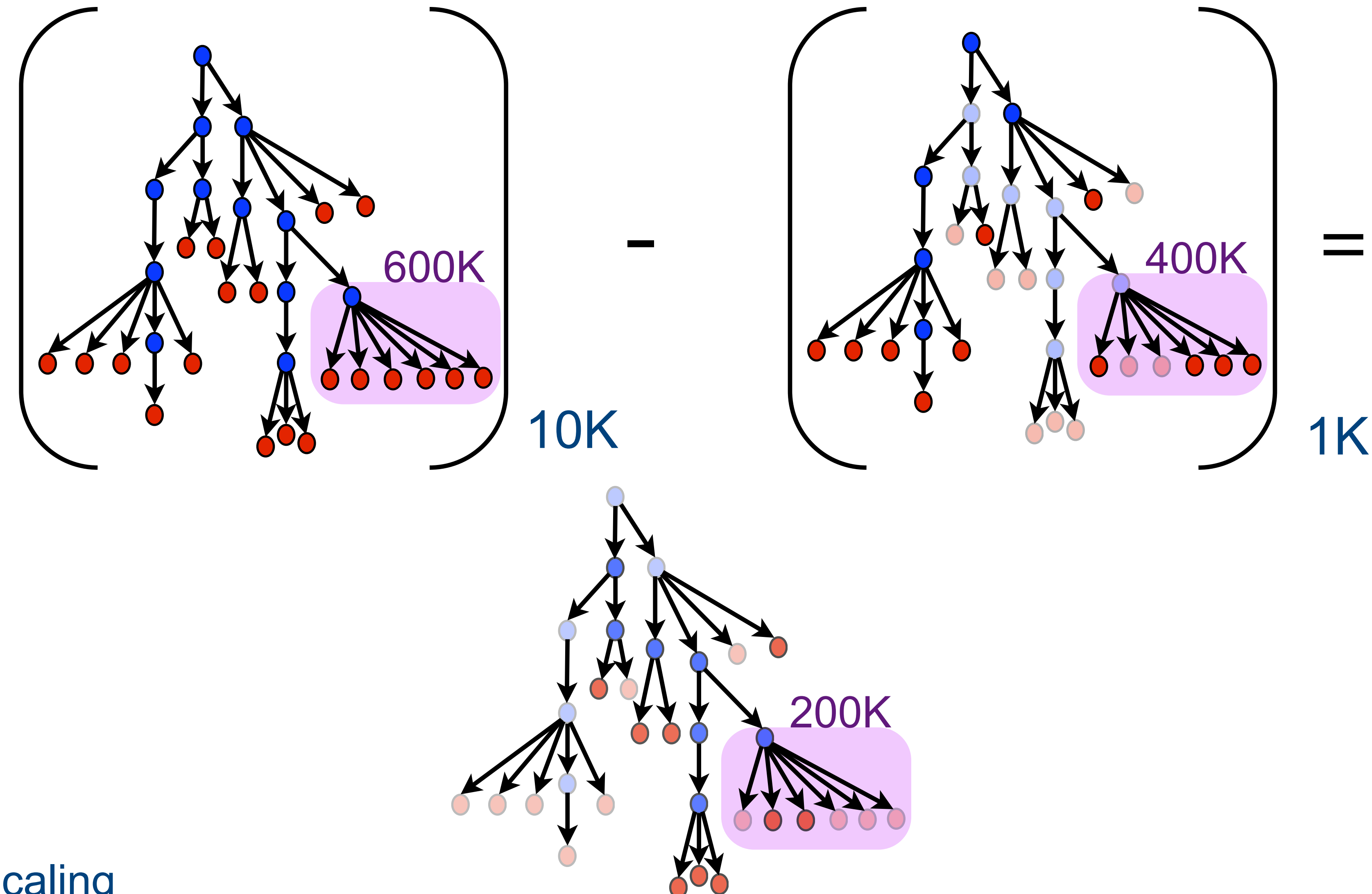
Note: higher is better

Scalability Analysis Challenges

- Parallel applications
  - modern software uses layers of libraries
  - performance is often context dependent
- Monitoring
  - bottleneck nature: computation, data movement, synchronization?
  - pragmatics: need low data volume and low perturbation



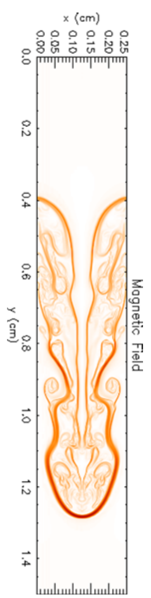
Analyzing Weak Scaling: 1K to 10K processors



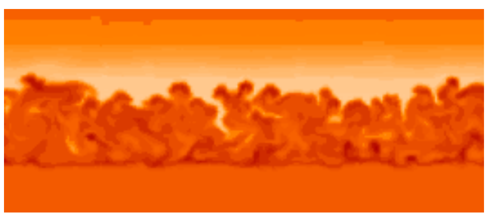
Weak scaling

Scalability Analysis Demo: FLASH

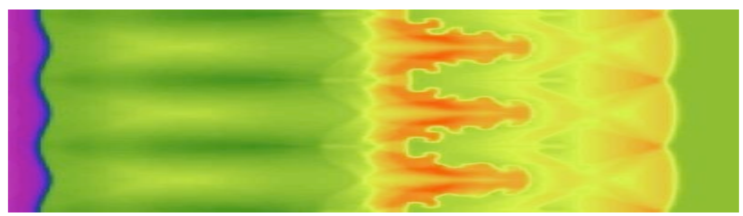
Code:	University of Chicago FLASH
Simulation:	white dwarf collapse
Platform:	Blue Gene/P
Experiment:	8192 vs. 256 processors
Scaling type:	weak



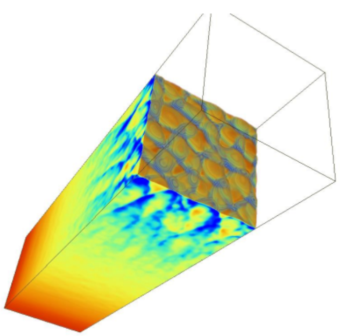
Magnetic Rayleigh-Taylor



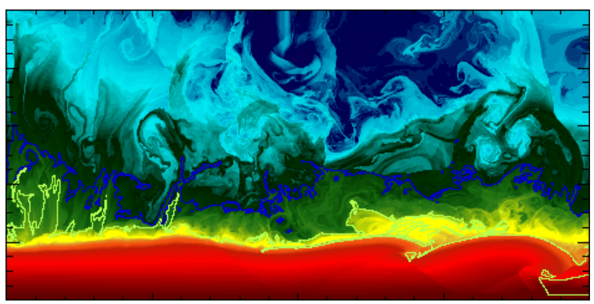
Nova outbursts on white dwarfs



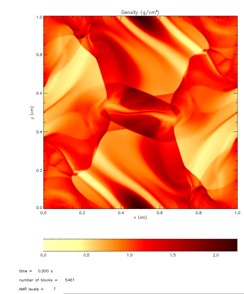
Laser-driven shock instabilities



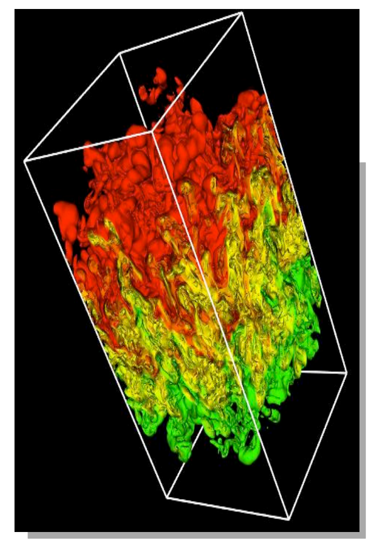
Cellular detonation



Helium burning on neutron stars



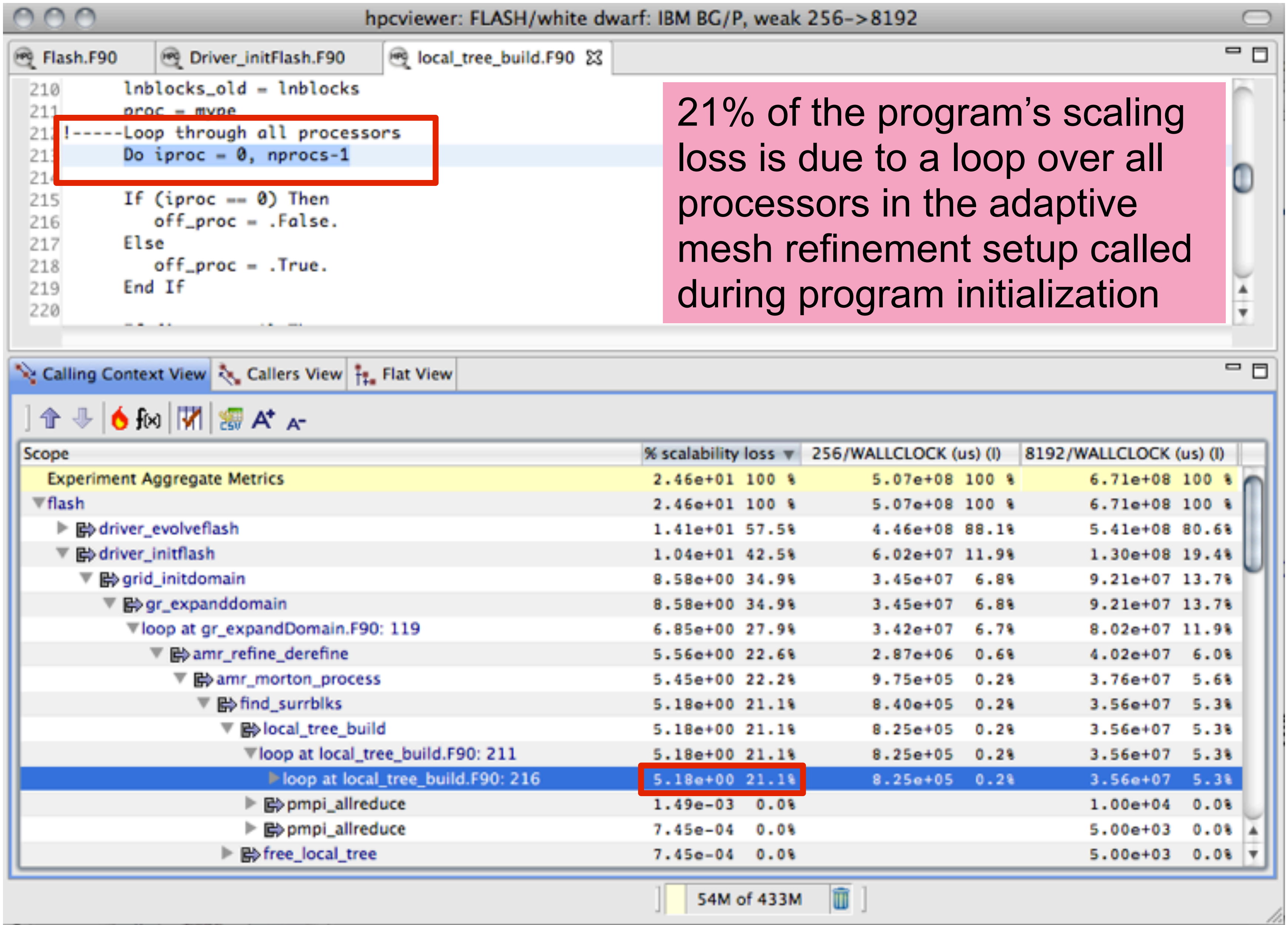
Orzag/Tang MHD vortex



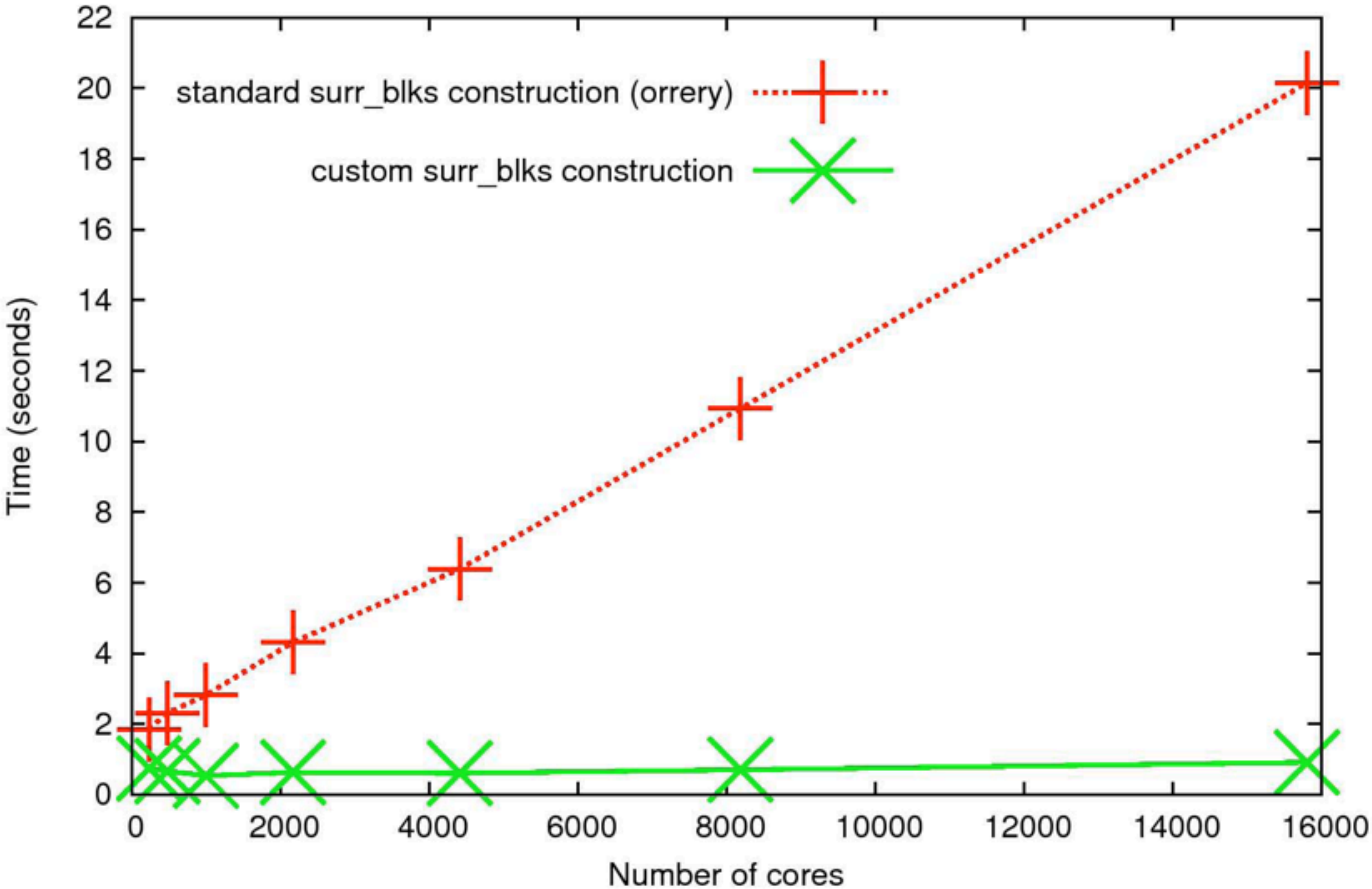
Rayleigh-Taylor instability

Figures courtesy of FLASH Team, University of Chicago

Pinpointing a Scalability Loss in Flash



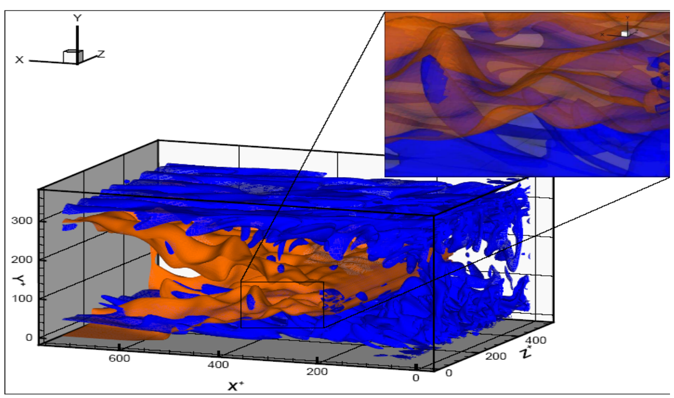
Improved Flash Scaling of AMR Setup



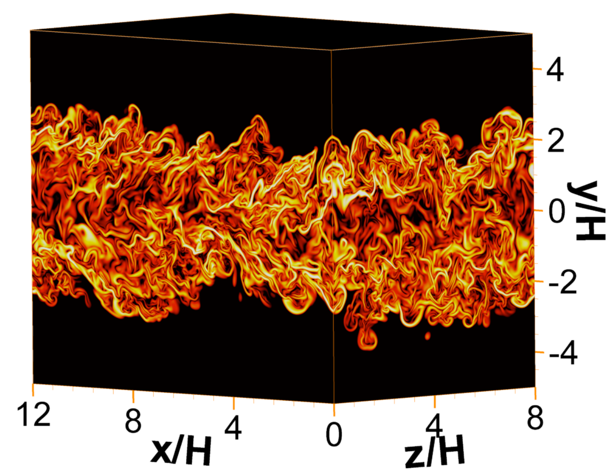
Graph courtesy of Anshu Dubey, U Chicago

S3D - DNS Solver

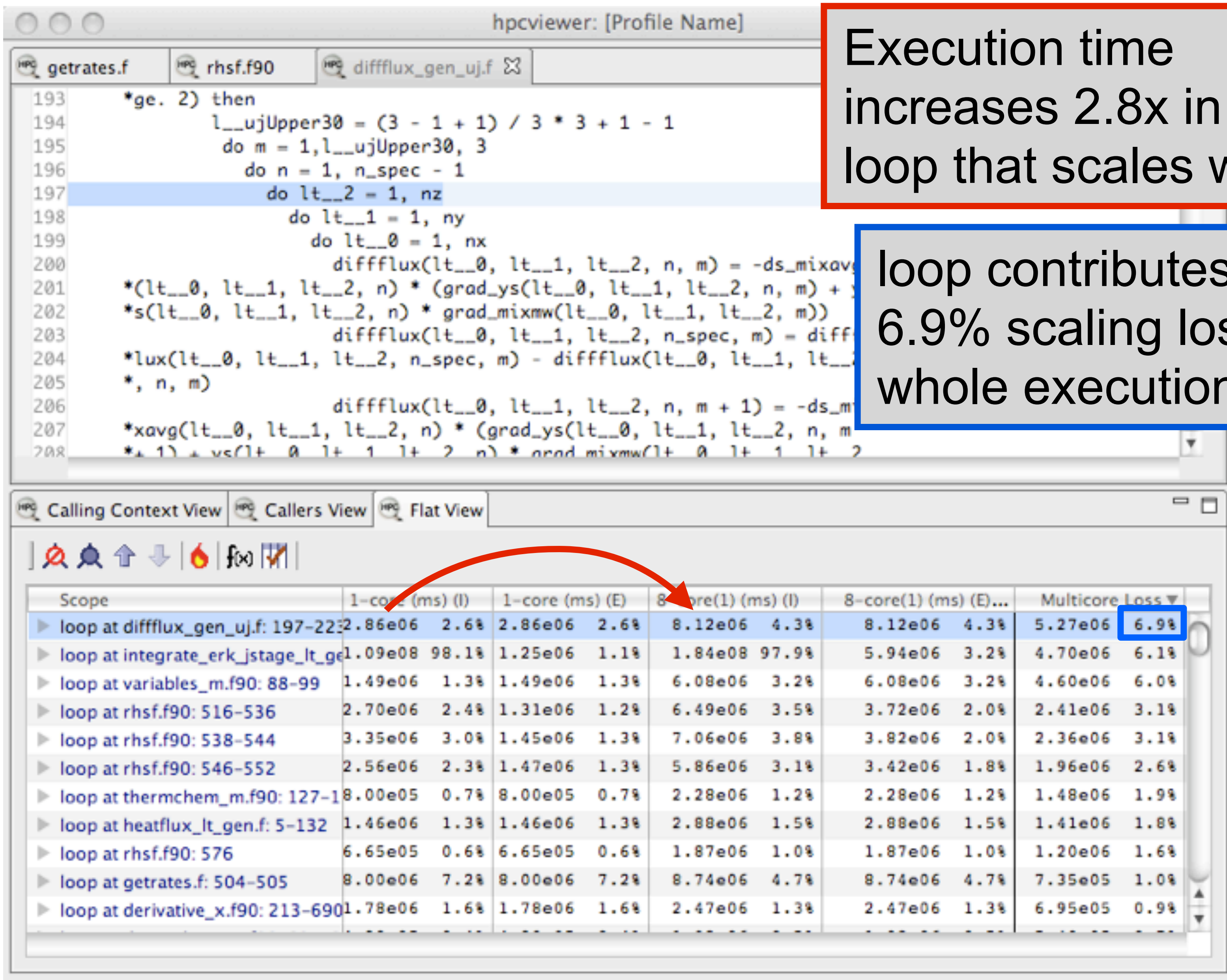
- Solves compressible reacting Navier-Stokes equations
- High fidelity numerical methods
  - 8th order finite-difference
  - 4th order explicit RK integrator
- Hierarchy of molecular transport models
- Detailed chemistry
- Multi-physics (sprays, radiation and soot)
  - from SciDAC-TSTC (Terascale Simulation of Turbulent Combustion)



Text and figures courtesy of Jacqueline H. Chen, SNL



S3D: Multicore Losses at the Loop Level



Scope	1-core (ms) (I)		1-core (ms) (E)		8-core(1) (ms) (I)		8-core(1) (ms) (E)...		Multicore Loss	
loop at diffflux_gen_uj.f: 197-223	2.86e06	2.6%	2.86e06	2.6%	8.12e06	4.3%	8.12e06	4.3%	5.27e06	6.9%
loop at integrate_erk_jstage_lt_ge	1.09e08	98.1%	1.25e06	1.1%	1.84e08	97.9%	5.94e06	3.2%	4.70e06	6.1%
loop at variables_m.f90: 88-99	1.49e06	1.3%	1.49e06	1.3%	6.08e06	3.2%	6.08e06	3.2%	4.60e06	6.0%
loop at rhsf.f90: 516-536	2.70e06	2.4%	1.31e06	1.2%	6.49e06	3.5%	3.72e06	2.0%	2.41e06	3.1%
loop at rhsf.f90: 538-544	3.35e06	3.0%	1.45e06	1.3%	7.06e06	3.8%	3.82e06	2.0%	2.36e06	3.1%
loop at rhsf.f90: 546-552	2.56e06	2.3%	1.47e06	1.3%	5.86e06	3.1%	3.42e06	1.8%	1.96e06	2.6%
loop at thermchem_m.f90: 127-1	8.00e05	0.7%	8.00e05	0.7%	2.28e06	1.2%	2.28e06	1.2%	1.48e06	1.9%
loop at heatflux_lt_gen.f: 5-132	1.46e06	1.3%	1.46e06	1.3%	2.88e06	1.5%	2.88e06	1.5%	1.41e06	1.8%
loop at rhsf.f90: 576	6.65e05	0.6%	6.65e05	0.6%	1.87e06	1.0%	1.87e06	1.0%	1.20e06	1.6%
loop at getrates.f: 504-505	8.00e06	7.2%	8.00e06	7.2%	8.74e06	4.7%	8.74e06	4.7%	7.35e05	1.0%
loop at derivative_x.f90: 213-690	1.78e06	1.6%	1.78e06	1.6%	2.47e06	1.3%	2.47e06	1.3%	6.95e05	0.9%

Execution time increases 2.8x in the loop that scales worst

loop contributes a 6.9% scaling loss to whole execution

Outline

- Introduction to HPCToolkit
- Five new approaches for analyzing parallel program performance
  – scalability analysis using call path profiles [SC09]
  – blame shifting to analyze lock contention in threaded codes [PPoPP10]
  – pinpointing load imbalance in parallel codes [SC10]
  – understanding temporal dynamics of parallel codes
  – data centric analysis of program performance
- Conclusions

Understanding Lock Contention in Threaded Code

- Lock contention => idleness
  - explicitly threaded programs (Pthreads, etc)
  - implicitly threaded programs (critical sections in OpenMP, ...)
- Strategy: “blame-shifting” of contention from victim to perpetrator
  - use shared state (locks) to communicate blame
- How it works
  - consider spin-waiting
  - sample a working thread:
    - charge to ‘work’ metric
  - sample an idle thread
    - accumulate in idleness counter associated with a lock (atomic add)
  - working thread releases a lock
    - atomically swap 0 with lock’s idleness counter
    - exactly represents contention while that thread held the lock
    - unwind the call stack to attribute lock contention to a calling context

Lock Contention in MADNESS

```
add(MEMFUN_OBJT(memfunT)& obj,
    memfunT memfun,
    const arg1T& arg1, const arg2T& arg2, const a
    Future<REMFUTURE(MEMFUN_RETURNT(memfunT))> re
    add(new TaskMemfun<memfunT>(result,obj,memfun
    return result;
}
```

Quantum chemistry; MPI + pthreads
- **65M distinct locks**
- **max. of 340K live locks**
- **30K lock acquisitions/sec/thread**

1-5% overhead

Calling Context View | Callers View | Flat View

16 cores; 1 thread/core (4 x Barcelona)

μs

Scope	% idleness (all/E)	idleness (all/E)
Experiment Aggregate Metrics	2.35e+01 100 %	1.57e+09 100 %
pthread_spin_unlock	2.35e+01 100.0	
madness::Spinlock::unlock() const	2.35e+01 100.0	
inlined from worldmutex.h: 142	1.78e+01 75.6%	
madness::ThreadPool::add(madness::PoolTaskInterface*)	1.78e+01 75.6%	
inlined from worldtask.h: 581	7.35e+00 31.2%	4.92e+08 31.2%
madness::Future<> madness::WorldObject<>::task<>(	7.35e+00 31.2%	4.92
inlined from worldtask.h: 569	4.56e+00 19.4%	3.05
madness::Future<> madness::WorldObject<>::task<>(	4.56e+00 19.4%	3.05
inlined from worlddep.h: 68	1.53e+00 6.5%	1.02
inlined from worldtask.h: 570	1.49e+00 6.3%	9.97e+07 6.3%
madness::Future<> madness::WorldObject<>::task<>(	1.49e+00 6.3%	9.97e+07 6.3%
inlined from worldtask.h: 558	1.38e+00 5.9%	9.26e+07 5.9%
madness::Future<> madness::WorldTaskQueue::add<>(ma	6.72e-01 2.9%	4.49e+07 2.9%

lock contention accounts for **23.5**% of execution time.

Adding futures to shared global work queue.

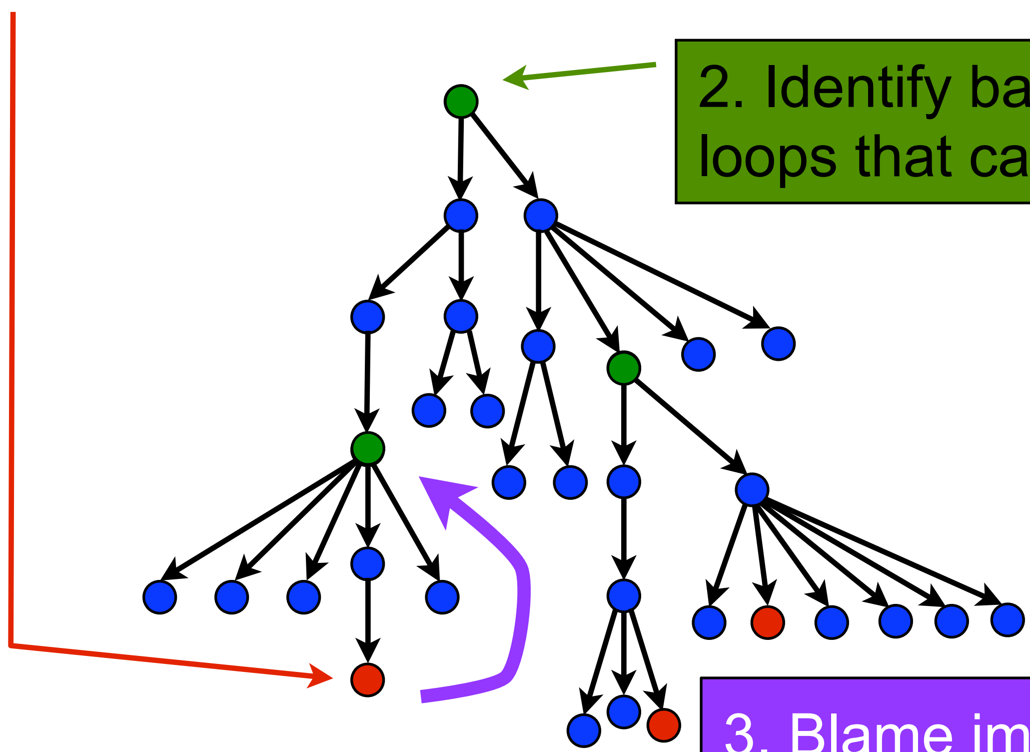
Outline

- Introduction to HPCToolkit
- Five new approaches for analyzing parallel program performance
  - scalability analysis using call path profiles [SC09]
  - blame shifting to analyze lock contention in threaded codes [PPoPP10]
  - pinpointing load imbalance in parallel codes [SC10]
  - understanding temporal dynamics of parallel codes
  - data centric analysis of program performance
- Conclusions

Identifying Load Imbalance Post Mortem

1. Identify exposed waiting: all imbalance is manifested in waiting

2. Identify balance points (procedures or loops that cannot contribute to imbalance)



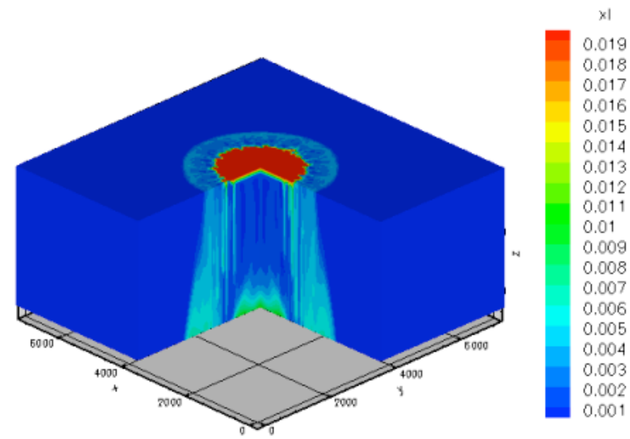
3. Blame imbalance on the computation subtree in which it originates

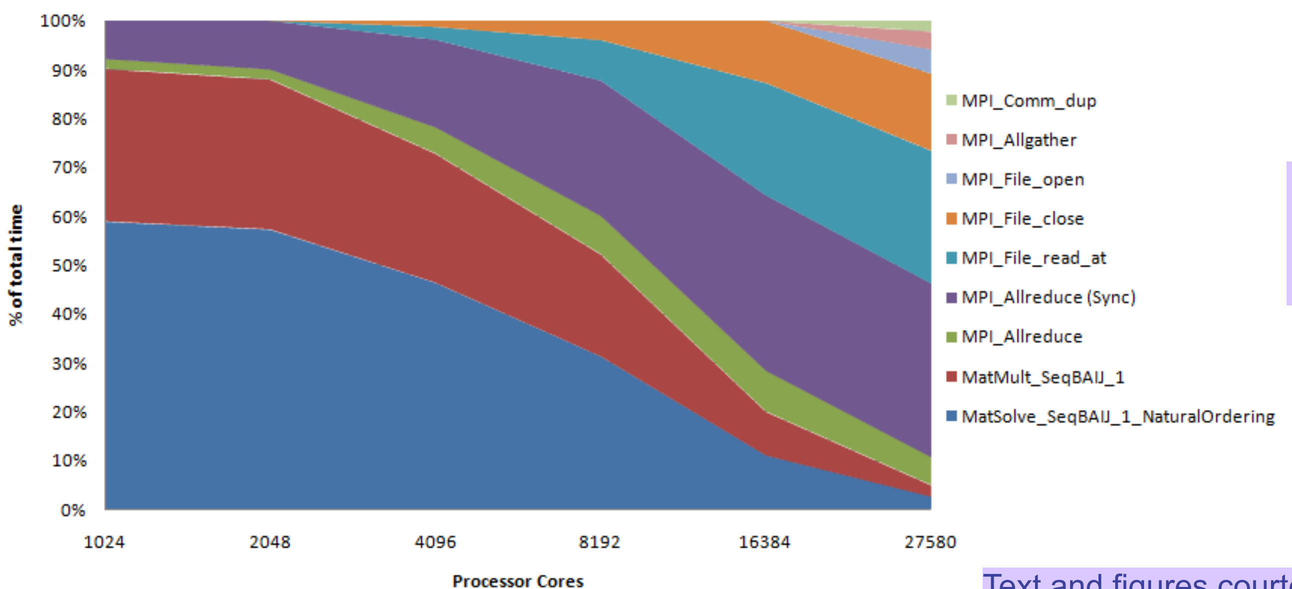
4. Associate each (summary) node with thread-level metric values

Load Imbalance Analysis Example

PFLOTRAN: modeling multi-scale, multiphase, multi-component subsurface reactive flows

Example use: modeling sequestration of CO_2 in deep geologic formations, where resolving density-driven fingering patterns is necessary to accurately describe the rate of dissipation of the CO_2 plume





Strong scaling study on Cray XT

Text and figures courtesy of PFLOTRAN Team

PFLOTRAN

8K cores, Cray XT5

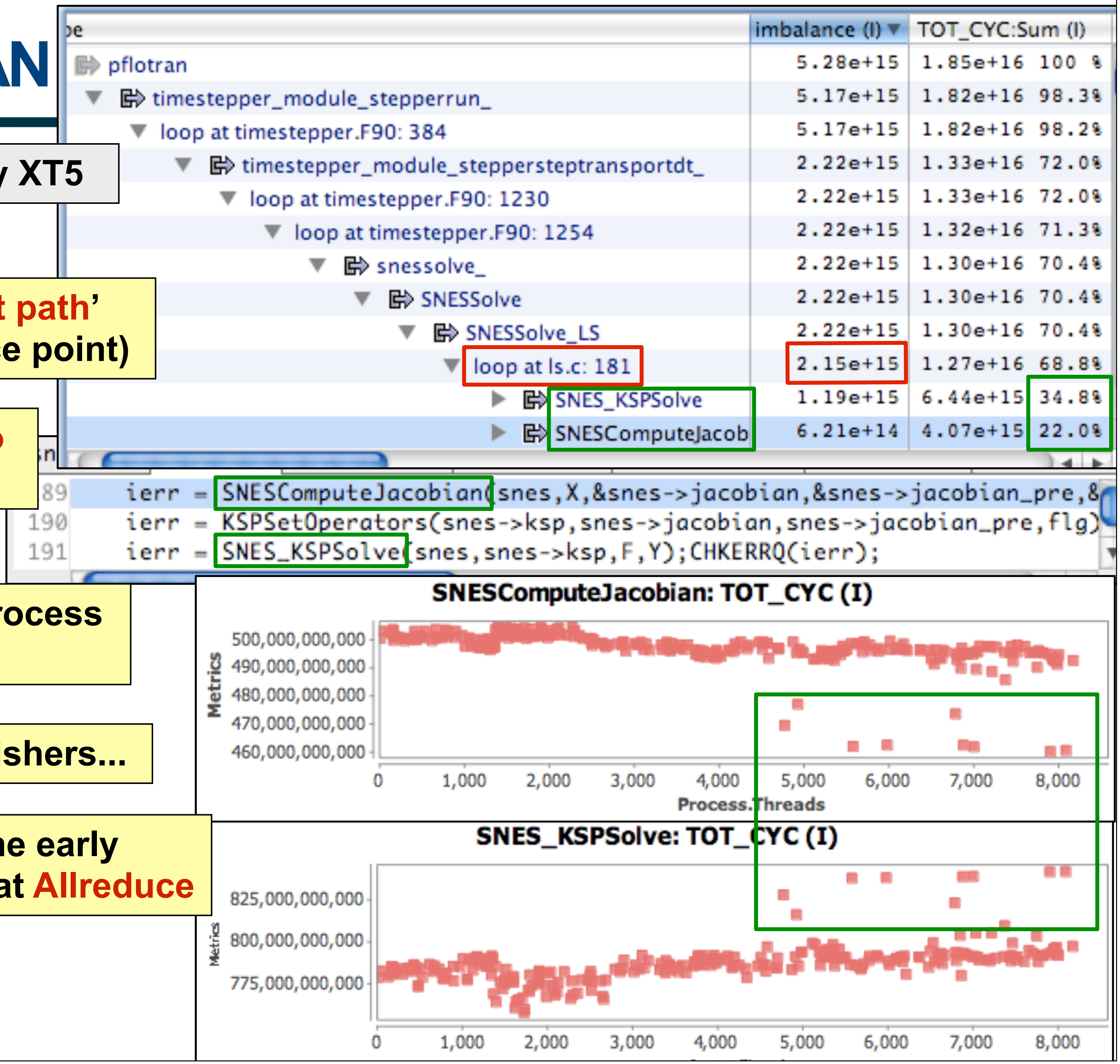
1. Drill down 'hot path' to loop (a balance point)

2. Notice top two call sites...

3. Plot the per-process values:

Early finishers...

... become early arrivers at Allreduce

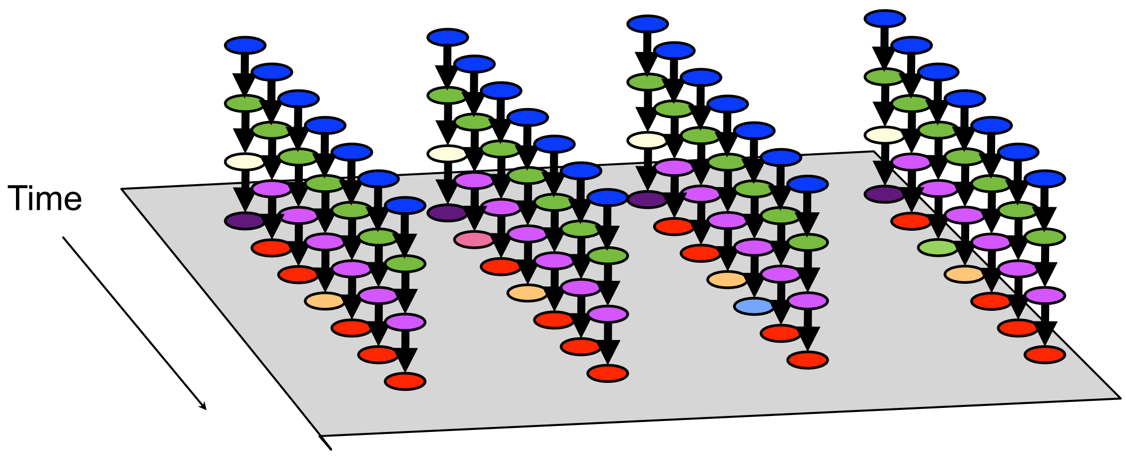


Outline

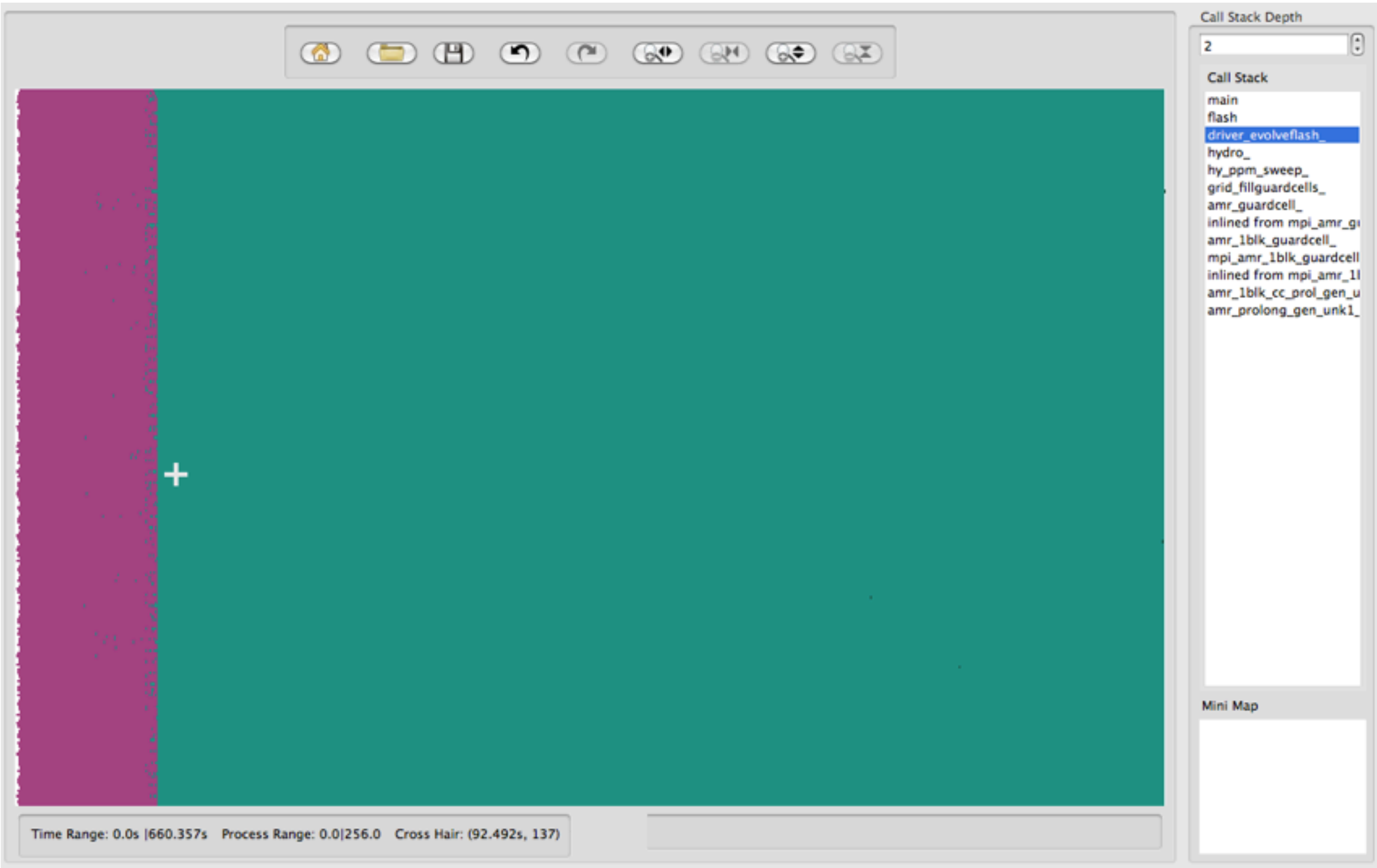
- Introduction to HPCToolkit
- Five new approaches for analyzing parallel program performance
  - scalability analysis using call path profiles [SC09]
  - blame shifting to analyze lock contention in threaded codes [PPoPP10]
  - pinpointing load imbalance in parallel codes [SC10]
  - understanding temporal dynamics of parallel codes
  - data centric analysis of program performance
- Conclusions

Understanding Temporal Behavior

- Profiling compresses out the temporal dimension
  - that's why serialization is 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 execution with a depth slice

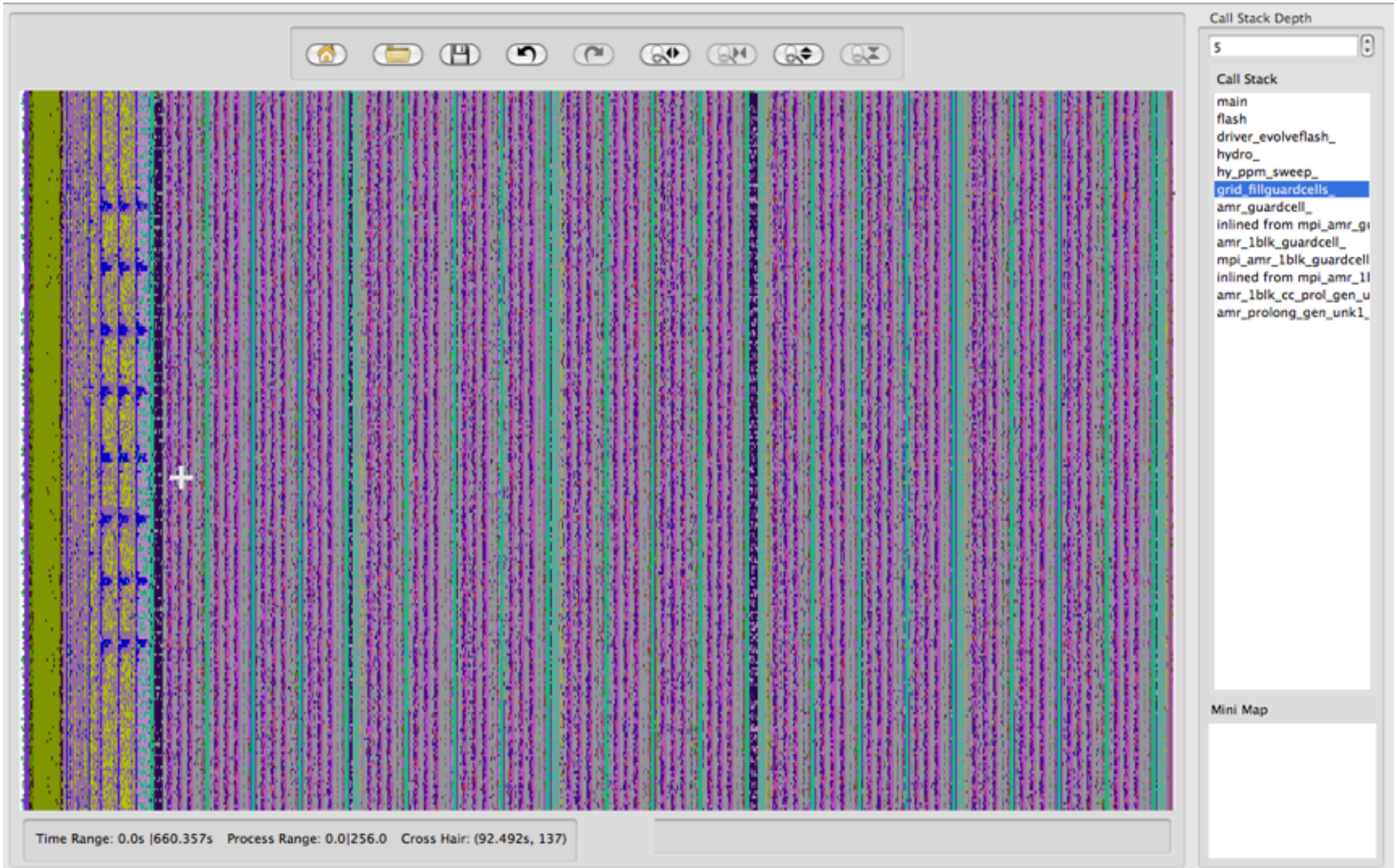


Flash White Dwarf Collapse on 256 Cores



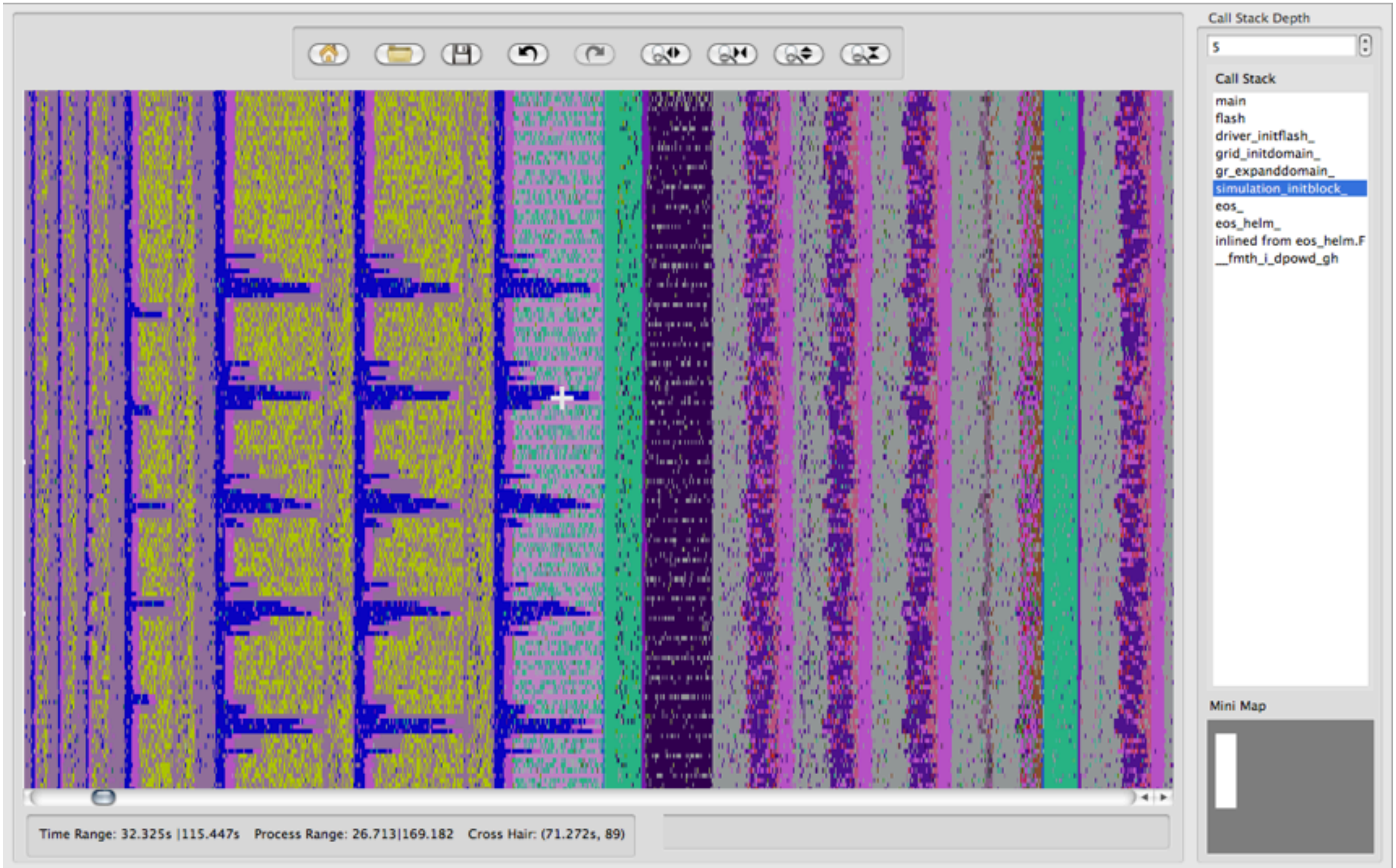
Full execution at call stack depth 2

Flash White Dwarf Collapse on 256 Cores



Full execution at call stack depth 5

Flash White Dwarf Collapse on 256 Cores



Execution detail at call stack depth 5

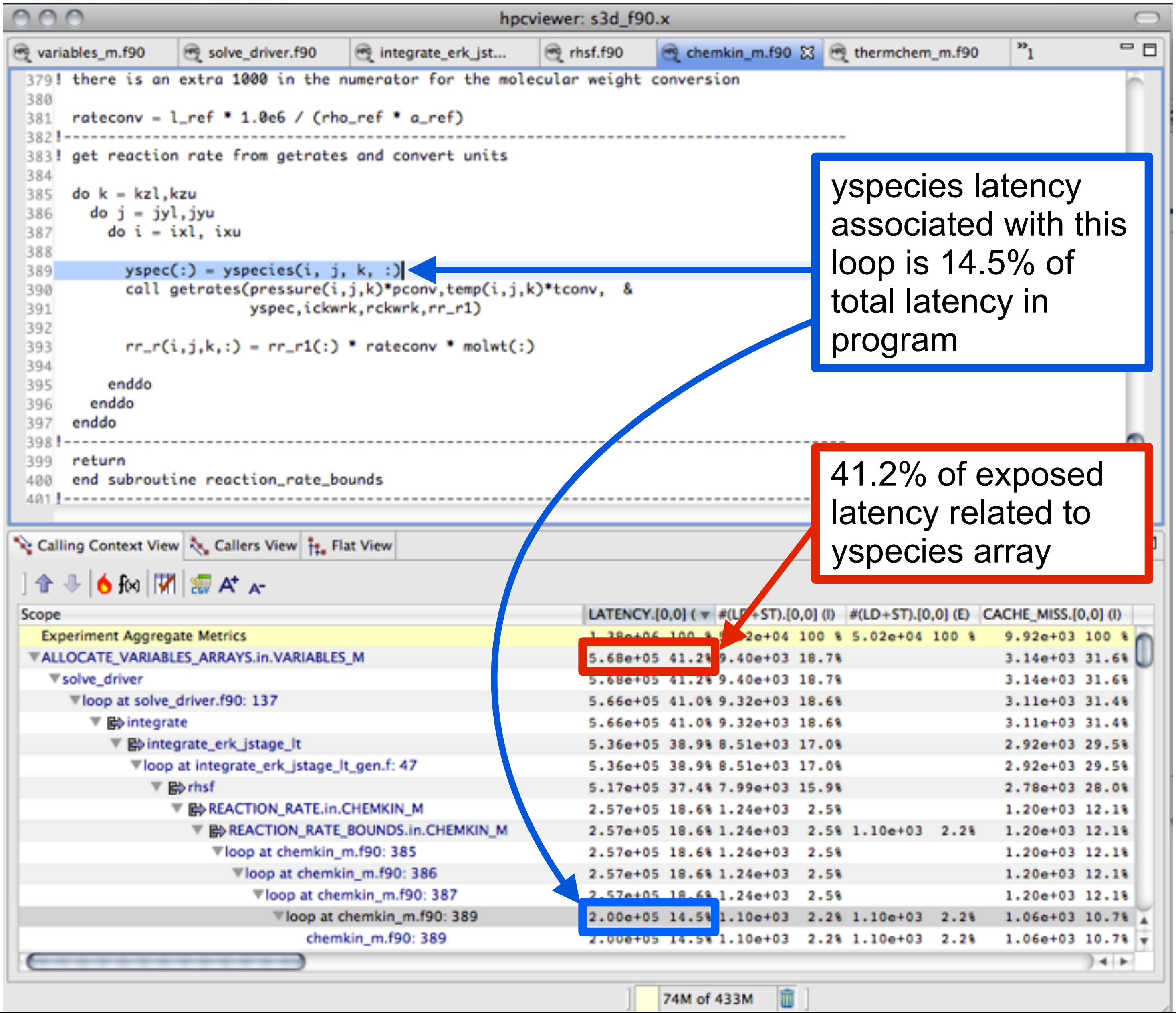
Outline

- Introduction to HPCToolkit
- Five new approaches for analyzing parallel program performance
  - scalability analysis using call path profiles [SC09]
  - blame shifting to analyze lock contention in threaded codes [PPoPP10]
  - pinpointing load imbalance in parallel codes [SC10]
  - understanding temporal dynamics of parallel codes
  - data centric analysis of program performance
- Conclusions

Data Centric Analysis

- Goal: associate memory hierarchy locality problems with particular data structures
- Approach
  - intercept memory allocations to associate data range with allocation
  - associate latency with data structures using “instruction based sampling” capability of AMD Opteron CPUs
    - identify instances of loads and store instructions
    - identify the data structure an access touches based on L/S address
    - measure the total latency associated with each L/S
  - present results in hpcviewer

Data Centric Analysis of S3D



yspecies latency associated with this loop is 14.5% of total latency in program

41.2% of exposed latency related to yspecies array

Conclusions

- Obtain insight, accuracy & precision by combining call path profiling, binary analysis, and blame shifting
- Show surprisingly effective measurement and source-level attribution for fully optimized code (1-3% overhead)
  - statements in their full static and dynamic context
  - project low-level measurements to much higher levels
- Sampling-based measurements can deliver insight into a range of phenomena
  - scalability bottlenecks
  - sources of lock contention
  - load imbalance
  - temporal dynamics
  - problematic data structures

Some Challenges Ahead

- Data management for scalable measurement and analysis
- Moving from descriptive to prescriptive feedback
- Increasing importance of threading as core counts increase
- Heterogeneous architectures, e.g. GPU accelerators