Targeting the TAU Performance System for Extreme Scale

Allen D. Malony

Department of Computer and Information Science
University of Oregon

Acknowledgements

Performance Research Lab

- Sameer Shende, Senior Research Associate, Director
- Alan Morris, Senior software engineer
- Wyatt Spear, Software engineer
- Scott Biersdorff, Software engineer

Department Computer and Information Science

- Matt Sottile, Research Faculty, CIS Department
- Kevin Huck, Ph.D. student
- Aroon Nataraj, Ph.D. student
- Shangkar Mayanglambam, Ph.D. student

Outline

- Viewpoints
  - Scalability, productivity, and performance technology
  - Expert performance analysis and automation
  - Whole performance evaluation
  - Usability, integration, and performance tool design
- Some current work
  - Scalable performance analysis and problem solving
  - Parallel performance dynamics and monitoring
  - Integration in parallel programming systems
  - Performance data mining
- Concluding remarks

Challenges in Performance Problem Solving

- Scalable, optimized applications deliver HPC promise
- Optimization through *performance engineering* process
  - Understand performance complexity and inefficiencies
  - Tune application to run optimally at scale
- How to make the process more *effective* and *productive*?
- Performance technology part of larger environment
  - Programmability, reusability, portability, robustness
  - Application development and optimization productivity
- Process, performance technology, and use will change as parallel systems evolve
- Goal is to deliver effective performance with high productivity value now and in the future
  - *scientific performance return*

Performance Technology and Scale

- What does it mean for performance technology to scale?
  - Instrumentation, measurement, analysis, tuning
- Scaling complicates observation and analysis
  - Performance data size and value
    - standard approaches deliver a lot of data with little value
  - Measurement overhead, intrusion, perturbation, noise
    - measurement overhead → intrusion → perturbation
    - tradeoff with analysis accuracy
    - “noise” in the system distorts performance
  - Analysis complexity increases
    - more events, larger data, more parallelism
- Tuning needs to be less dependent on full-scale runs

Performance Process and Scale

- What will enhance productive application development?
- Address application (developer) requirements at scale
- Nature of application development may change
- What are the important events and performance metrics?
  - Tied to application structure and computational model
  - Tied to application domain and algorithms
  - Process and tools must be more *application-aware*
- Petascale will also need more online, adaptive methods
  - Complement static, offline assessment and adjustment
  - Introspection of execution (performance) dynamics
  - Requires scalable performance monitoring
  - Integration with runtime infrastructure

Whole Performance Evaluation

- Petascale performance requires optimized orchestration
  - Applicaton processor, memory, network, I/O
- Reductionist approaches to performance will be unable to support optimization and productivity objectives
- Application-level only performance view is myopic
  - Interplay of hardware, software, and system components
  - Ultimately determines how performance is delivered
- Performance should be evaluated *in toto*
  - Application and system components
  - Understand effects of performance interactions
  - Identify opportunities for optimization across levels
- Need *whole performance evaluation practice*

Role of Intelligence, Automation, and Knowledge

- Scale forces the process to become more intelligent
- Even with intelligent and application-specific tools, the decisions of what to analyze is difficult
- More automation and knowledge-based decision making
- Build these capabilities into performance tools
  - Support broader experimentation methods and refinement
  - Access and correlate data from several sources
  - Automate performance data analysis / mining / learning
  - Include predictive features and experiment refinement
- Knowledge-driven adaptation and optimization guidance
- Address scale issues through increased expertise

Usability, Integration, and Performance Tool Design

- What are usable performance tools?
  - Support the performance problem solving process
  - Does not make overall environment less productive
- Usable should not mean to make simple
  - Usability should not be at the cost of functionality
- Robust performance technology needs integration
- Integration in performance system does not imply
  - Performance system is monolithic
  - Performance system is a “Swiss army knife”
- Deconstruction or refactoring
  - Should lead to opportunities for integration

TAU Performance System® Project

- Tuning and Analysis Utilities (15+ year project effort)
- Performance system framework for HPC systems
  - Integrated, scalable, and flexible
  - Target parallel programming paradigms
- Integrated toolkit for performance problem solving
  - Instrumentation, measurement, analysis, visualization
  - Portable performance profiling and tracing facility
  - Performance data management and data mining
- Evolve TAU to address new requirements
  - Usability, automation, …
  - Scale, heterogeneity, …

TAU Measurement Approach

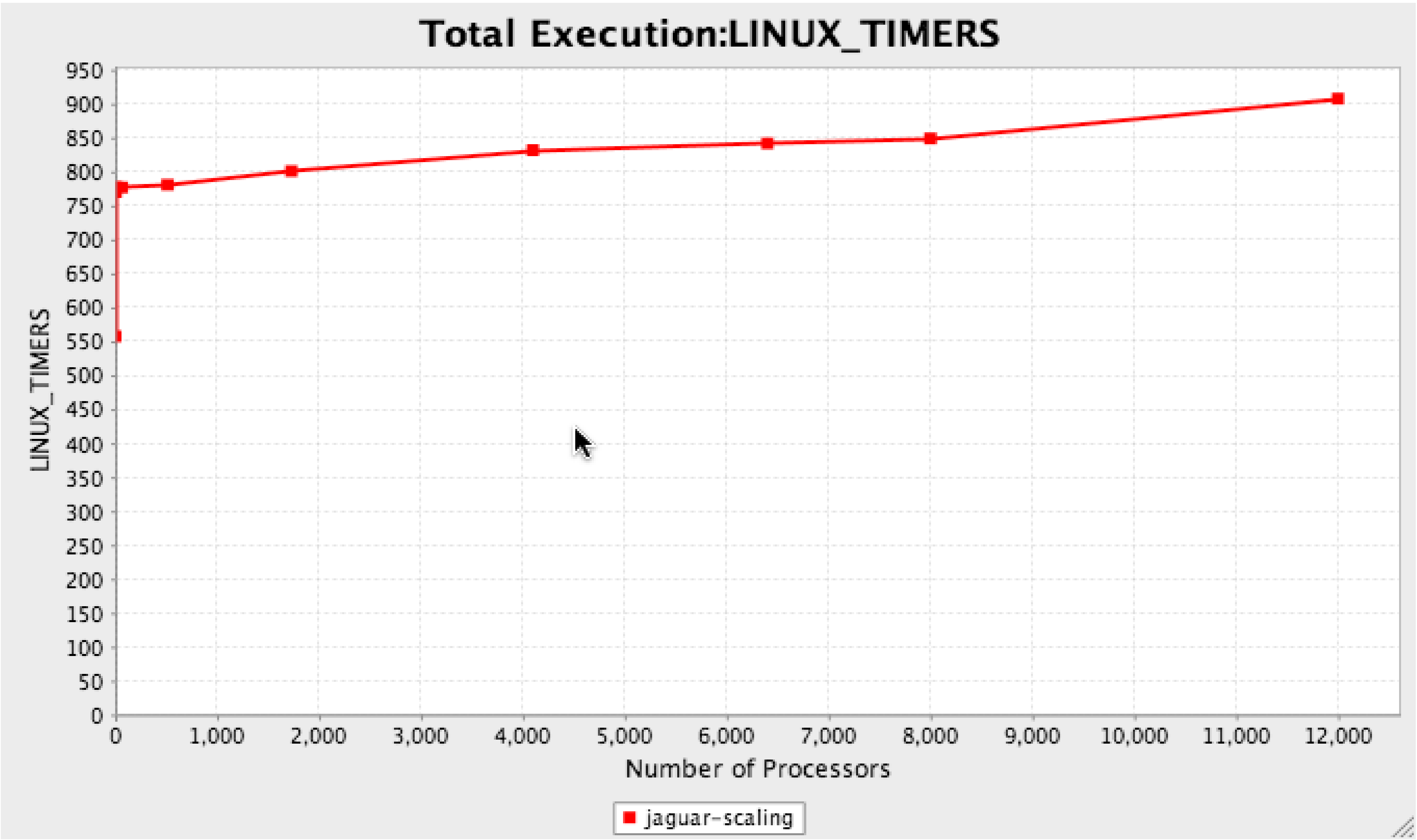
- *Direct performance measurement*
- Portable and scalable parallel profiling solution
  - Multiple profiling types and options
  - Event selection and control (enabling/disabling, throttling)
  - Online profile access and sampling
- Portable and scalable parallel tracing solution
  - Trace translation to OTF, EPILOG, Paraver, and SLOG2
  - Native EPILOG and Scalasca tracing library
  - Trace streams (OTF) and hierarchical trace merging
- Robust timing and hardware performance support
- Multiple counters (hardware, user-defined, system)

Performance Problem Solving: S3D Scalability Study

- S3D flow solver for simulation of turbulent combustion
  - Targeted application by DOE SciDAC PERI tiger team
- Performance scaling study (C2H4 benchmark)
  - Cray XT4 (ORNL, Jaguar)
  - IBM BG/P (ANL, Intrepid)
  - Weak scaling (1 to 12000 cores)
  - Evaluate scaling of code regions and MPI
- Demonstration of scalable performance measurement, analysis, and visualization
- Understanding scalability
  - Requires environment for performance investigation
  - Multi-experiment analysis, database, data mining, …

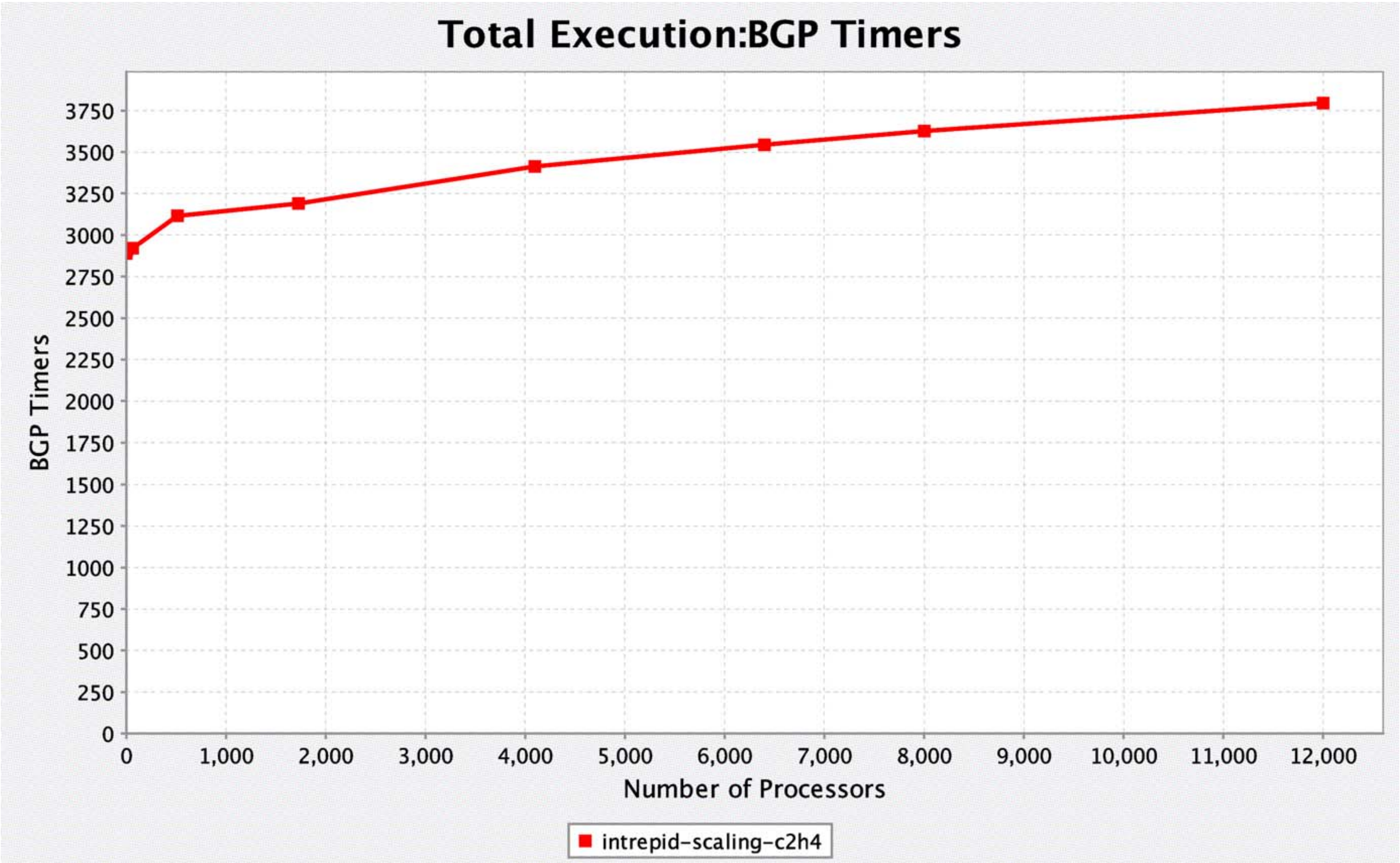
S3D Weak Scaling (Jaguar)

C2H4 benchmark

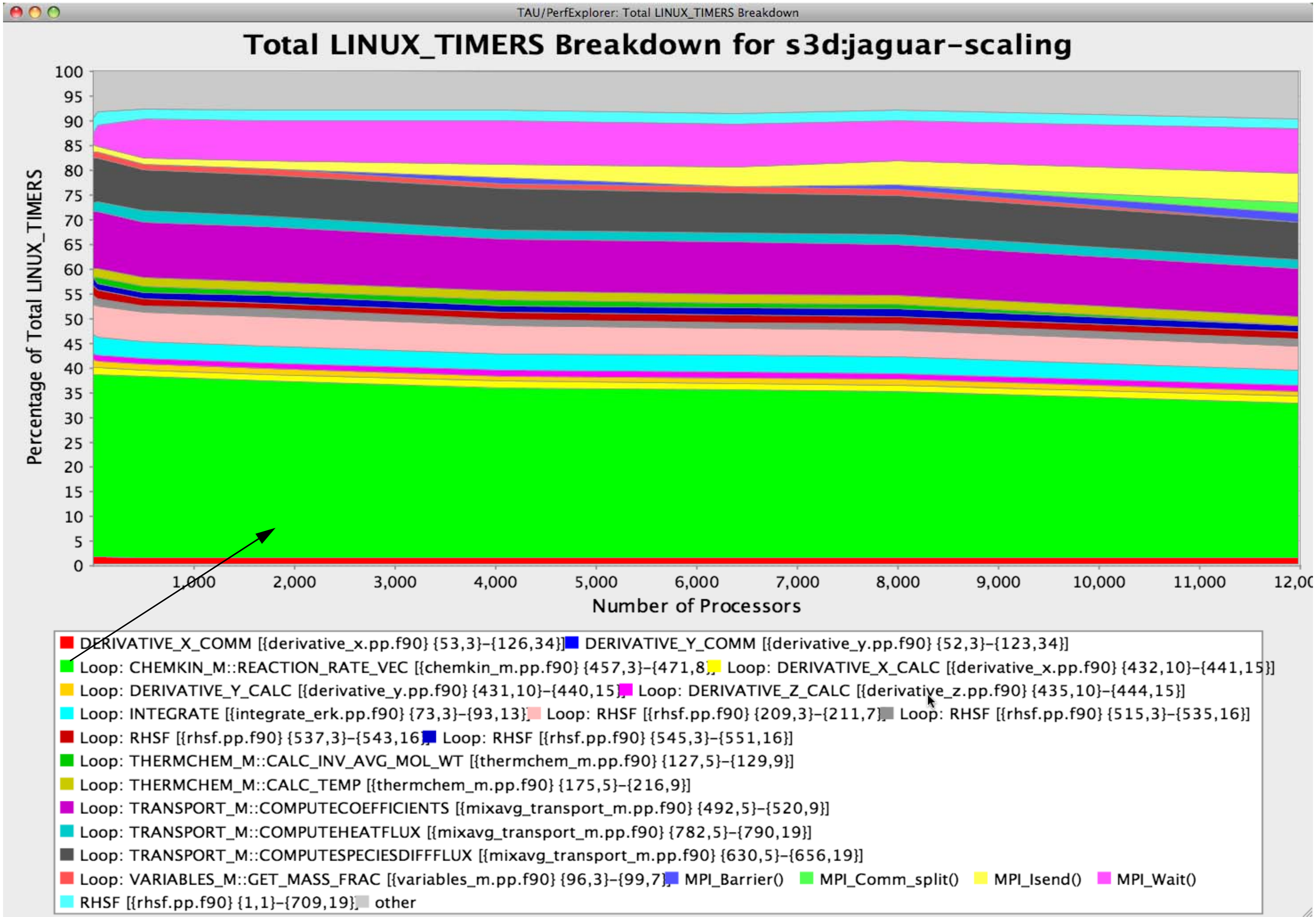


S3D Weak Scaling (Intrepid)

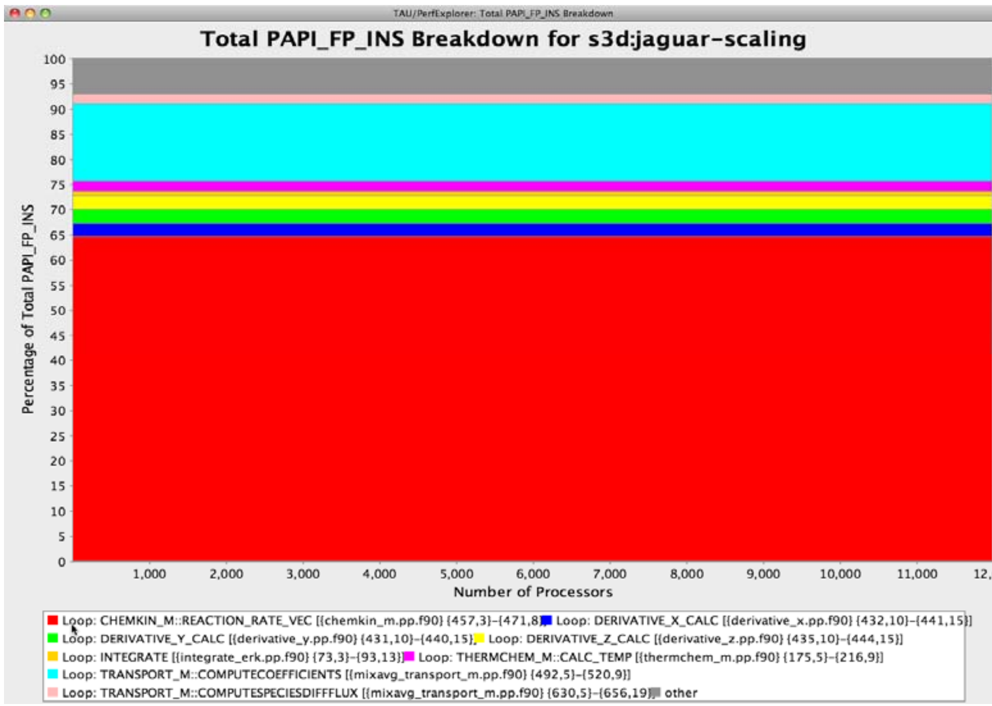
C2H4 benchmark

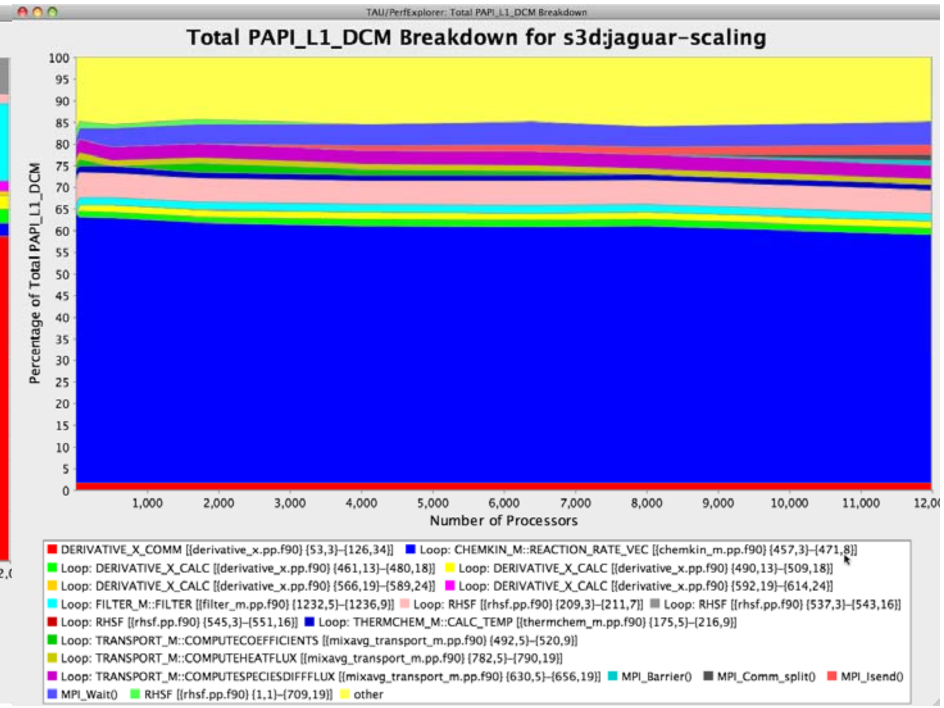


S3D Total Runtime Breakdown by Events (Jaguar)

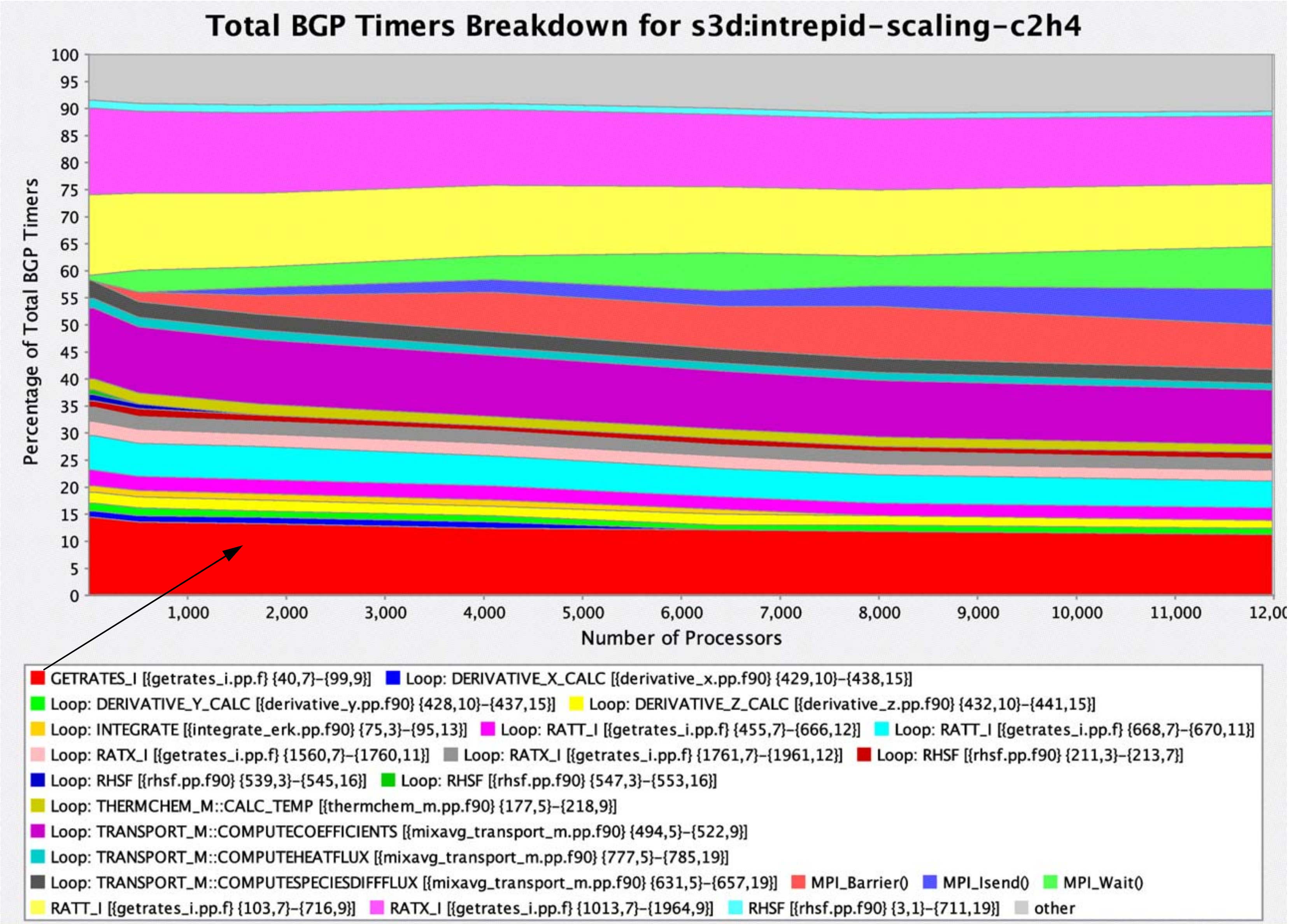


S3D FP Instructions / L1 Data Cache Miss (Jaguar)

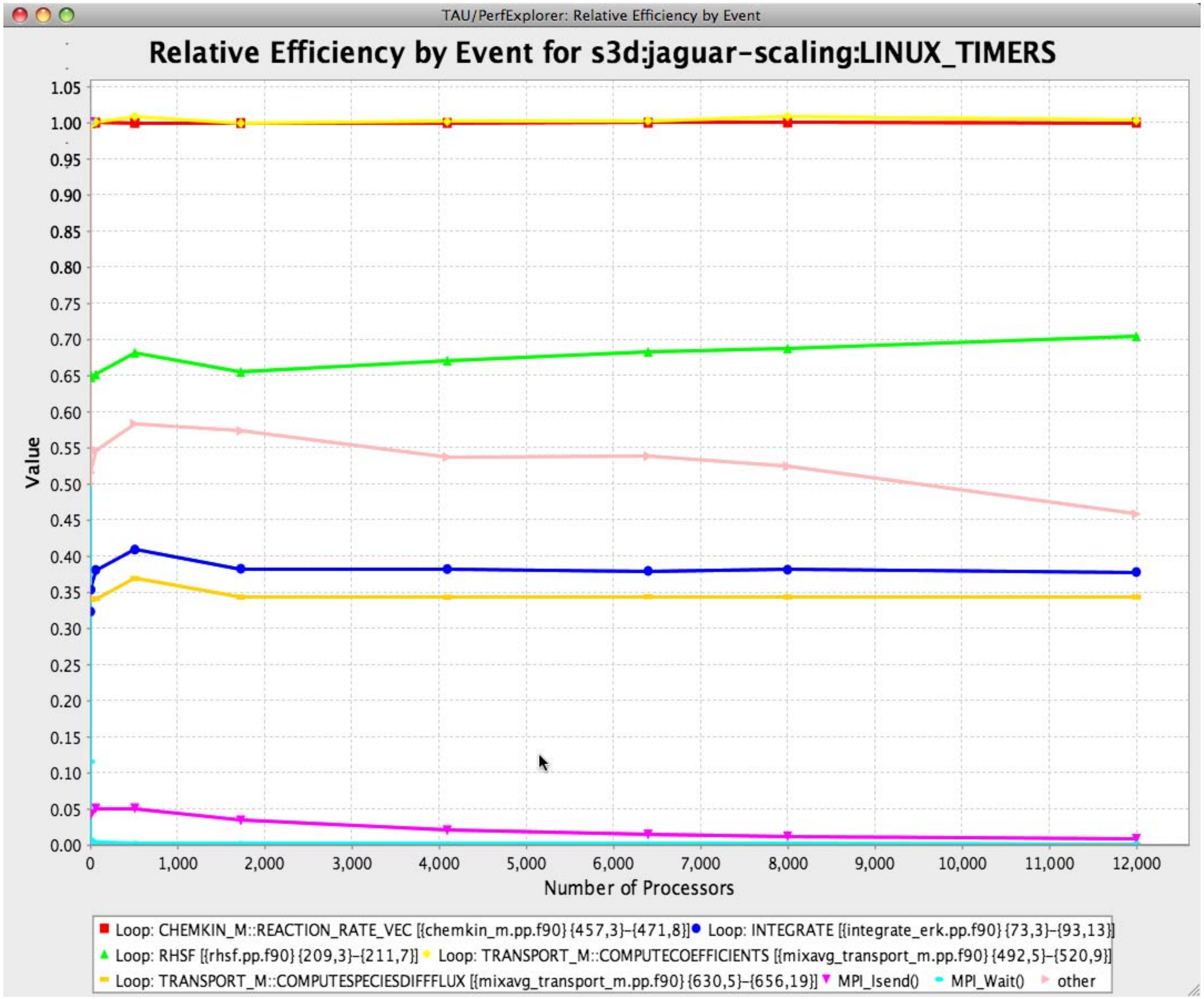




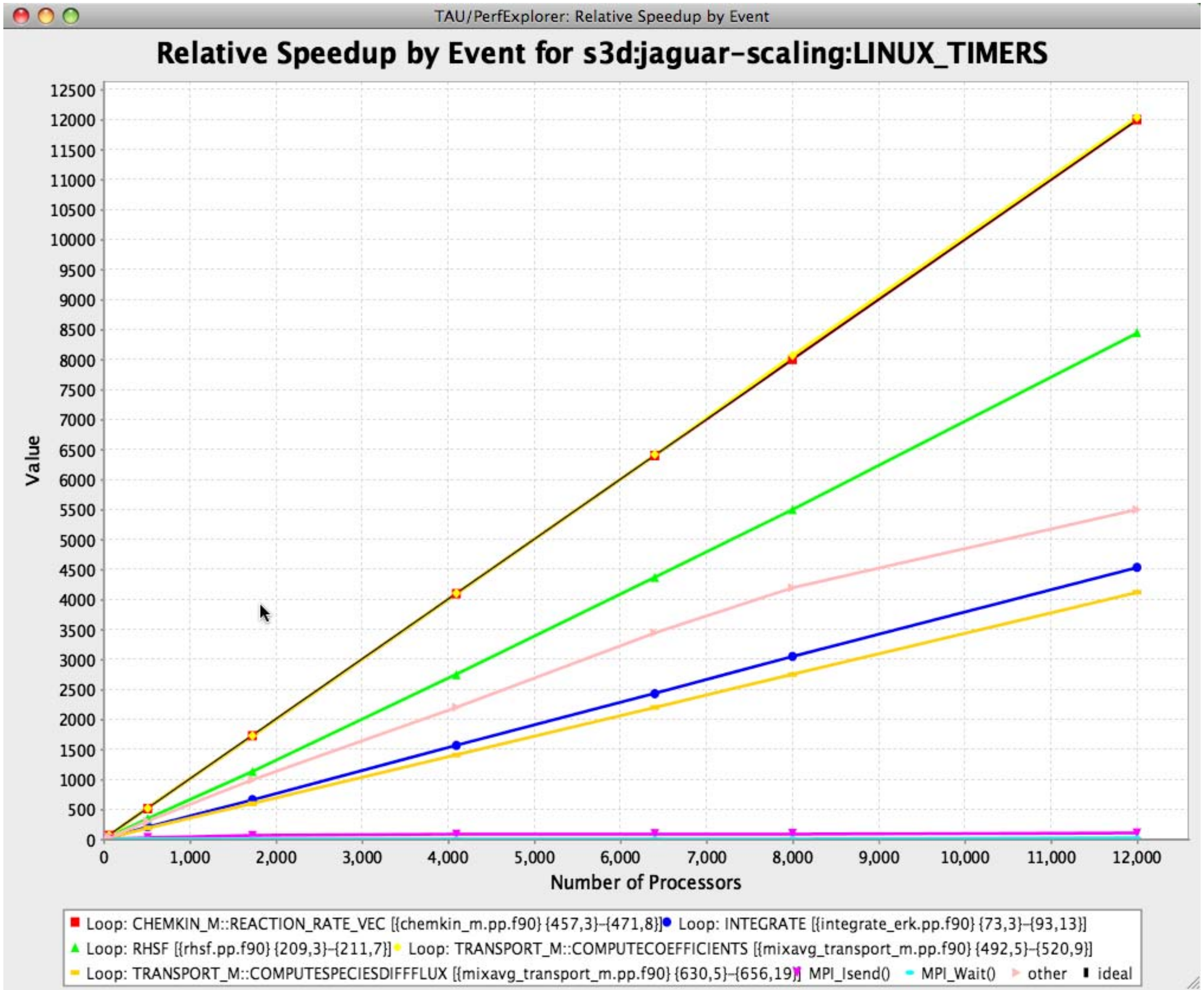
S3D Total Runtime Breakdown by Events (Intrepid)



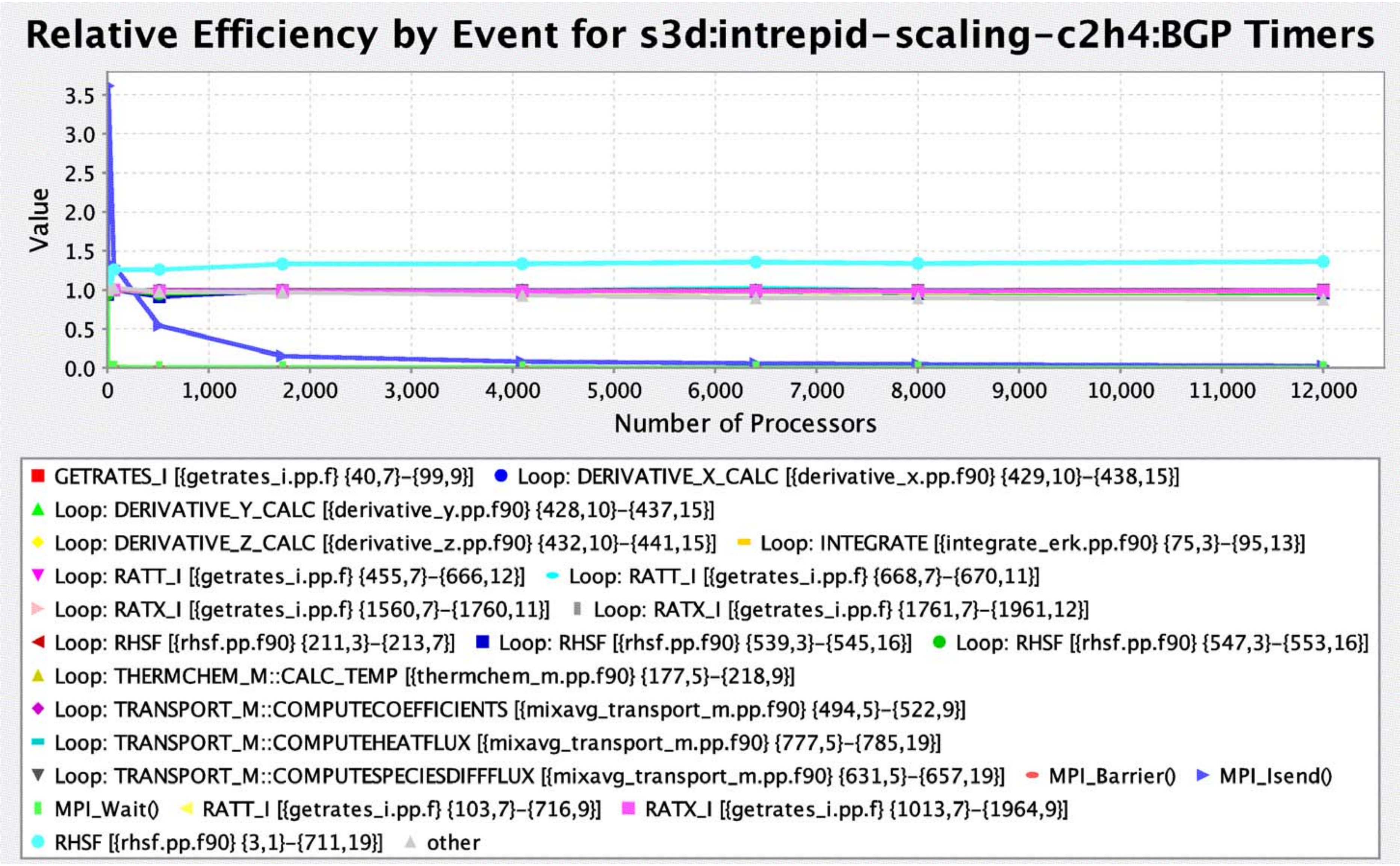
S3D Relative Efficiency by Event (Jaguar)



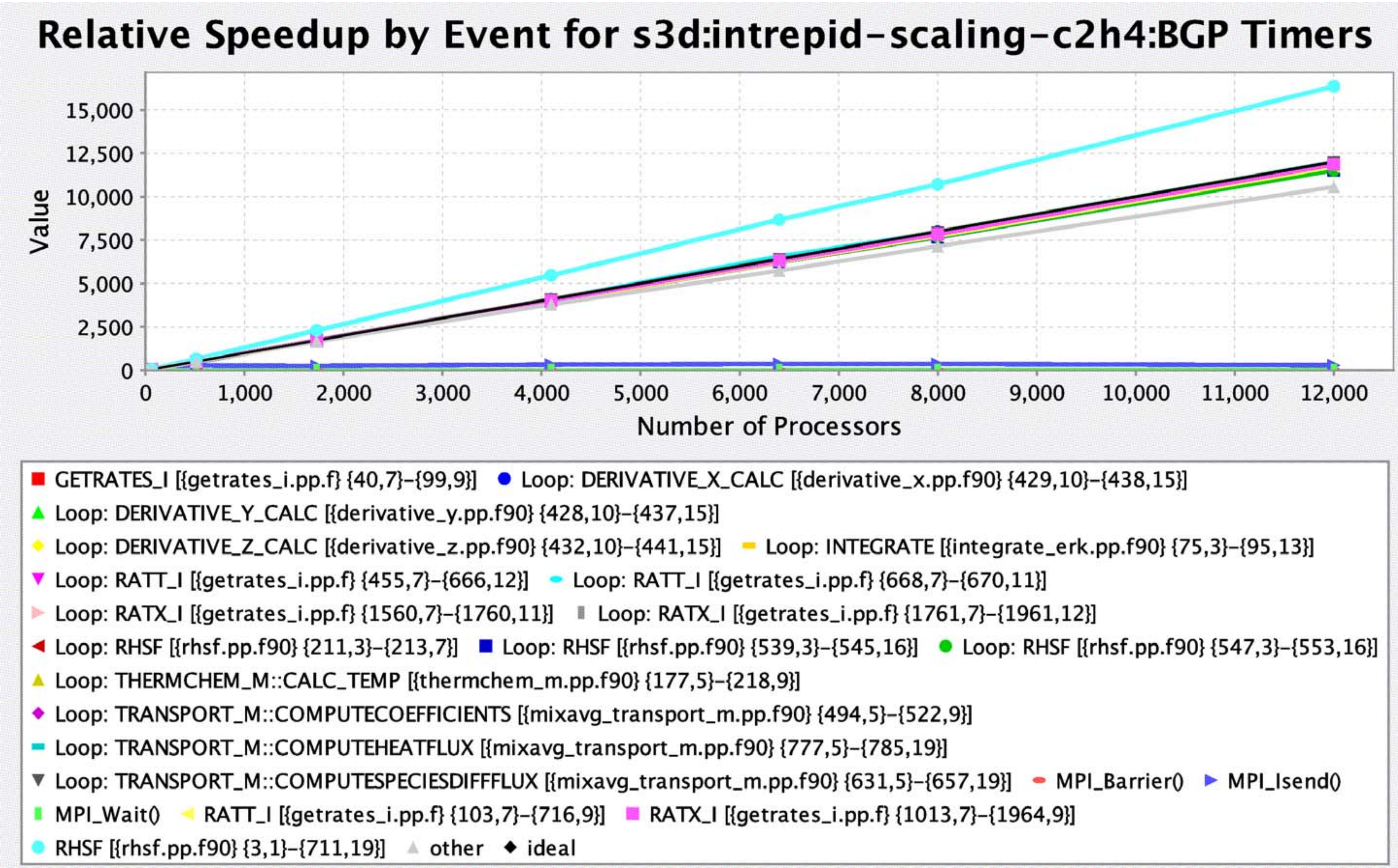
S3D Relative Speedup by Event (Jaguar)



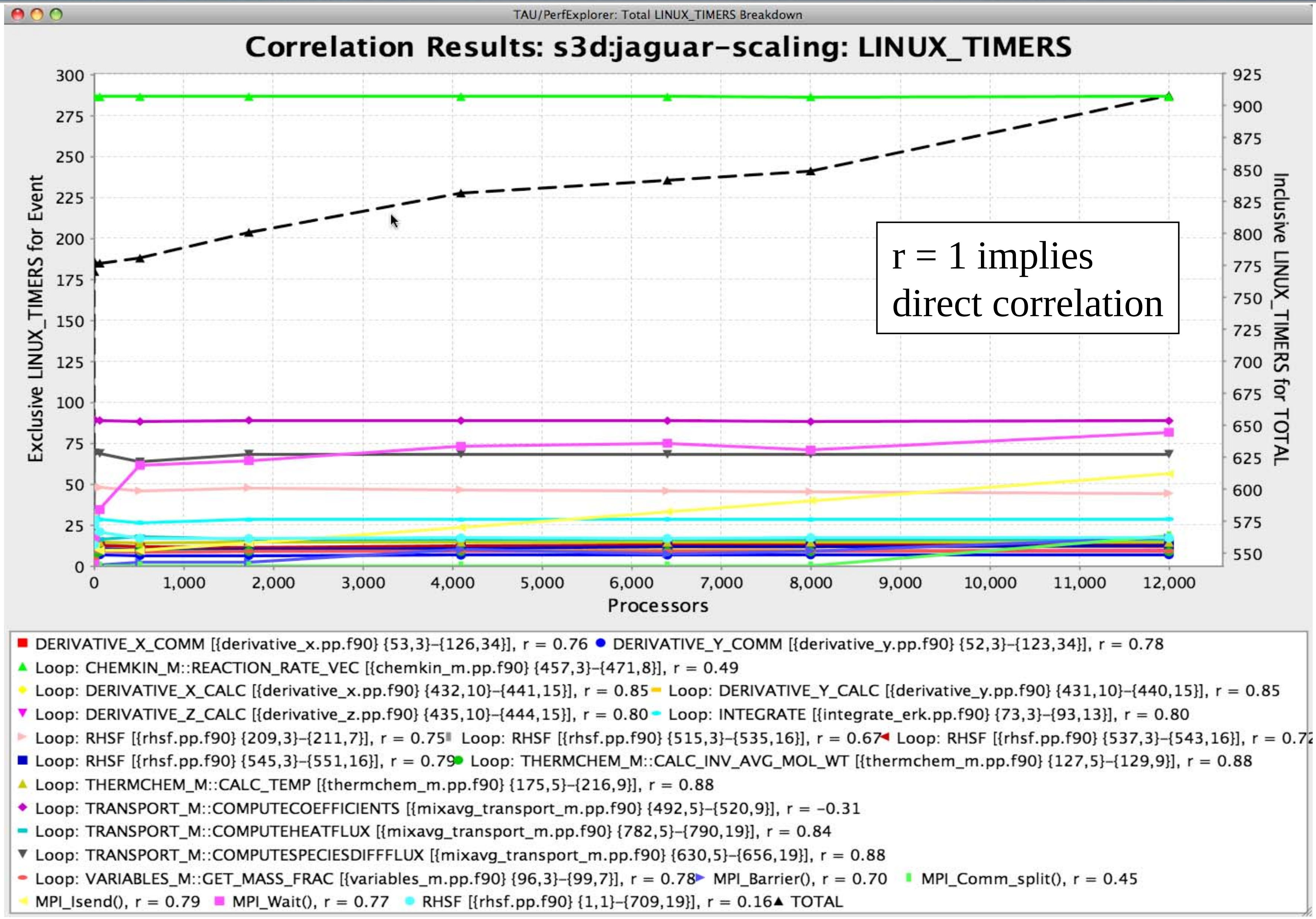
S3D Relative Efficiency by Event (Intrepid)



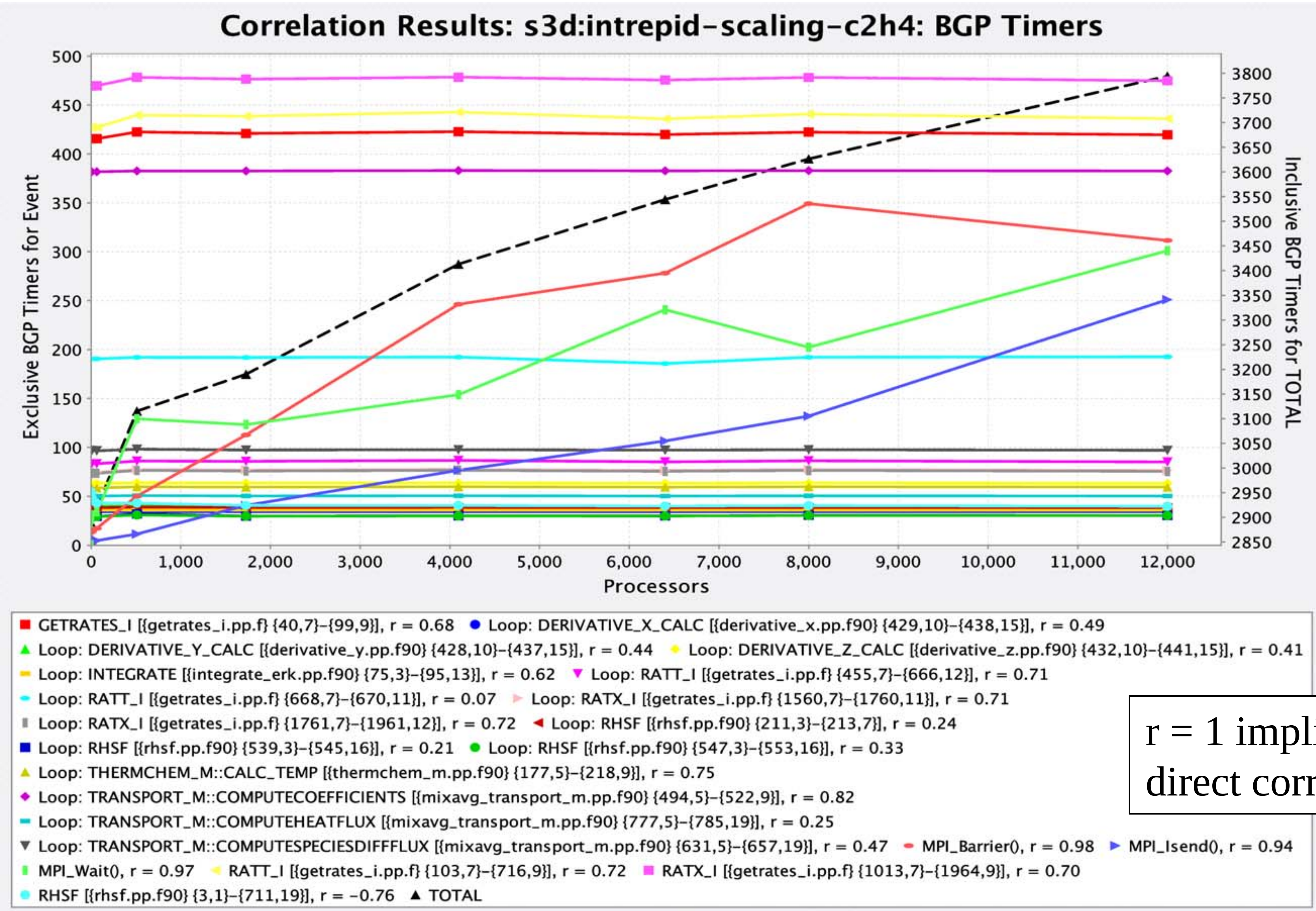
Relative Speedup by Event (Intrepid)



S3D Event Correlation to Total Time (Jaguar)

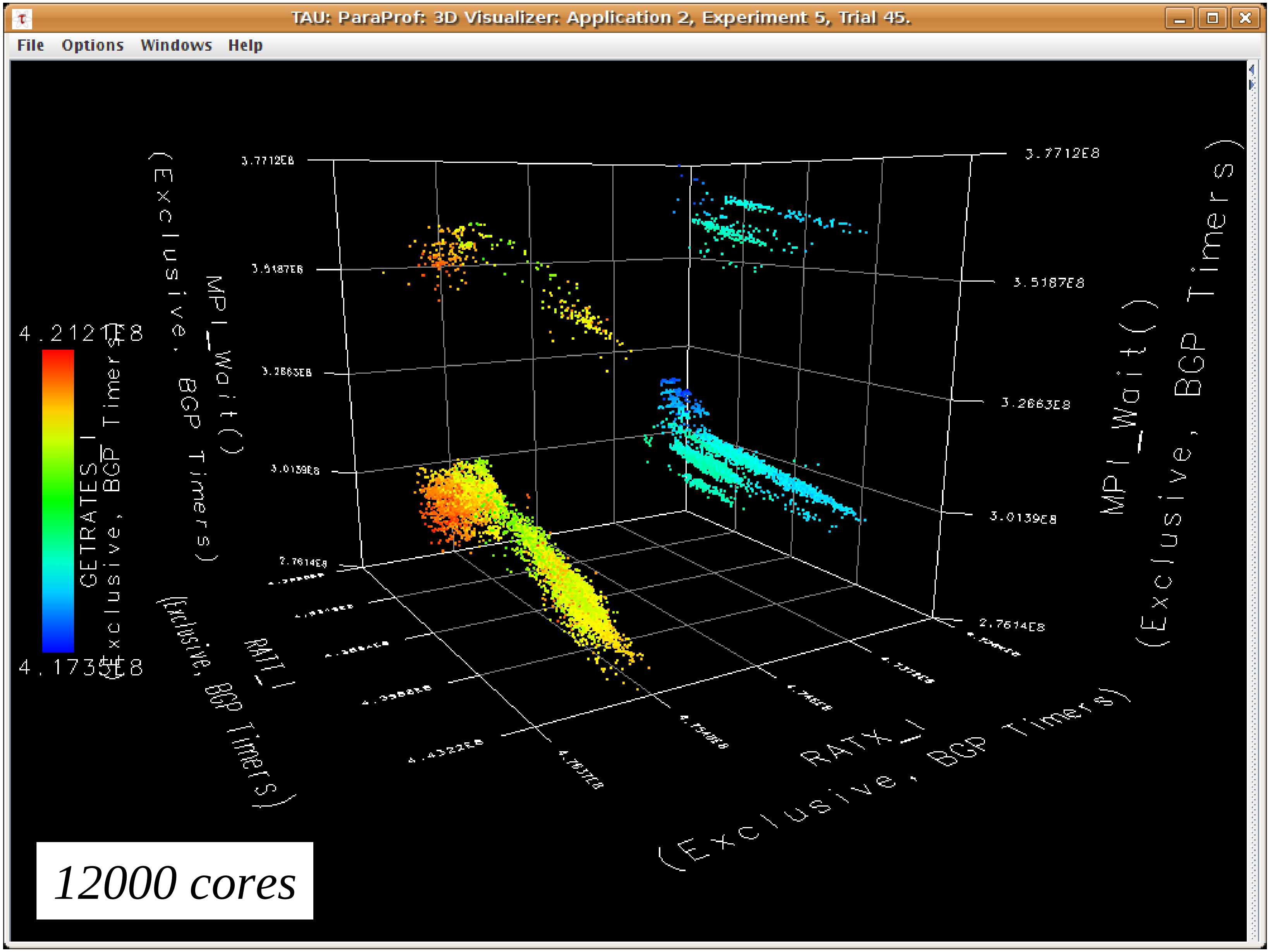


S3D Event Correlation to Total Time (Intrepid)



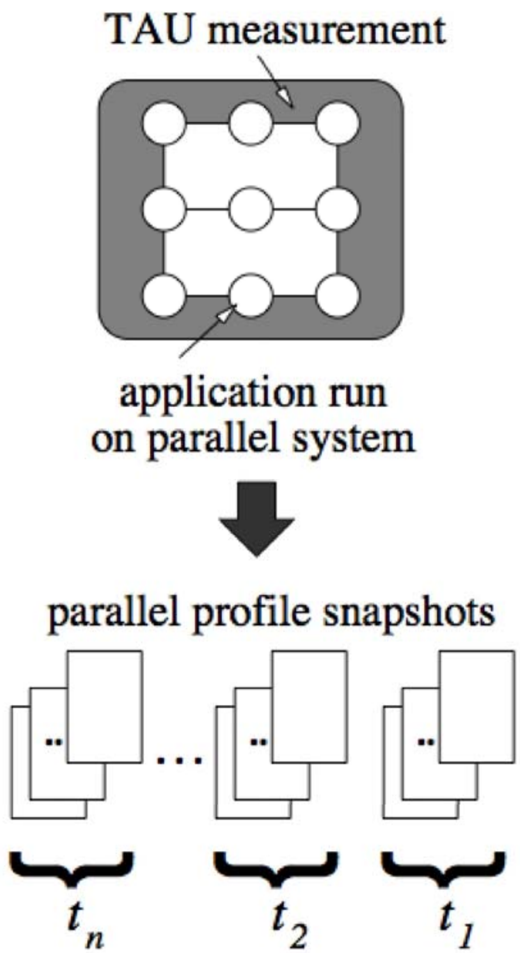
r = 1 implies direct correlation

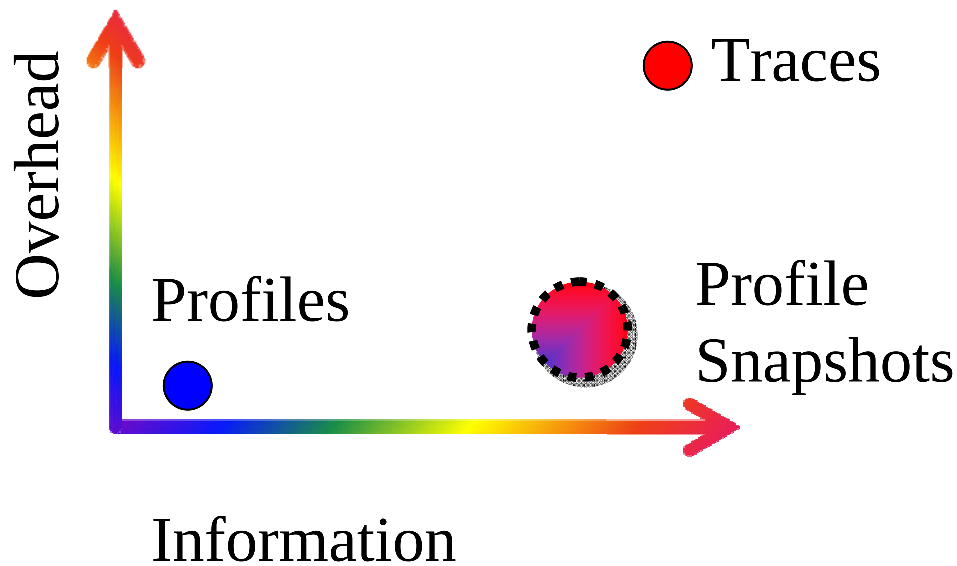
S3D 3D Correlation Cube (Intrepid, MPI_Wait)



Performance Dynamics: Parallel Profile Snapshots

- Profile snapshots are parallel profiles recorded at runtime
- Shows performance profile dynamics (all types allowed)

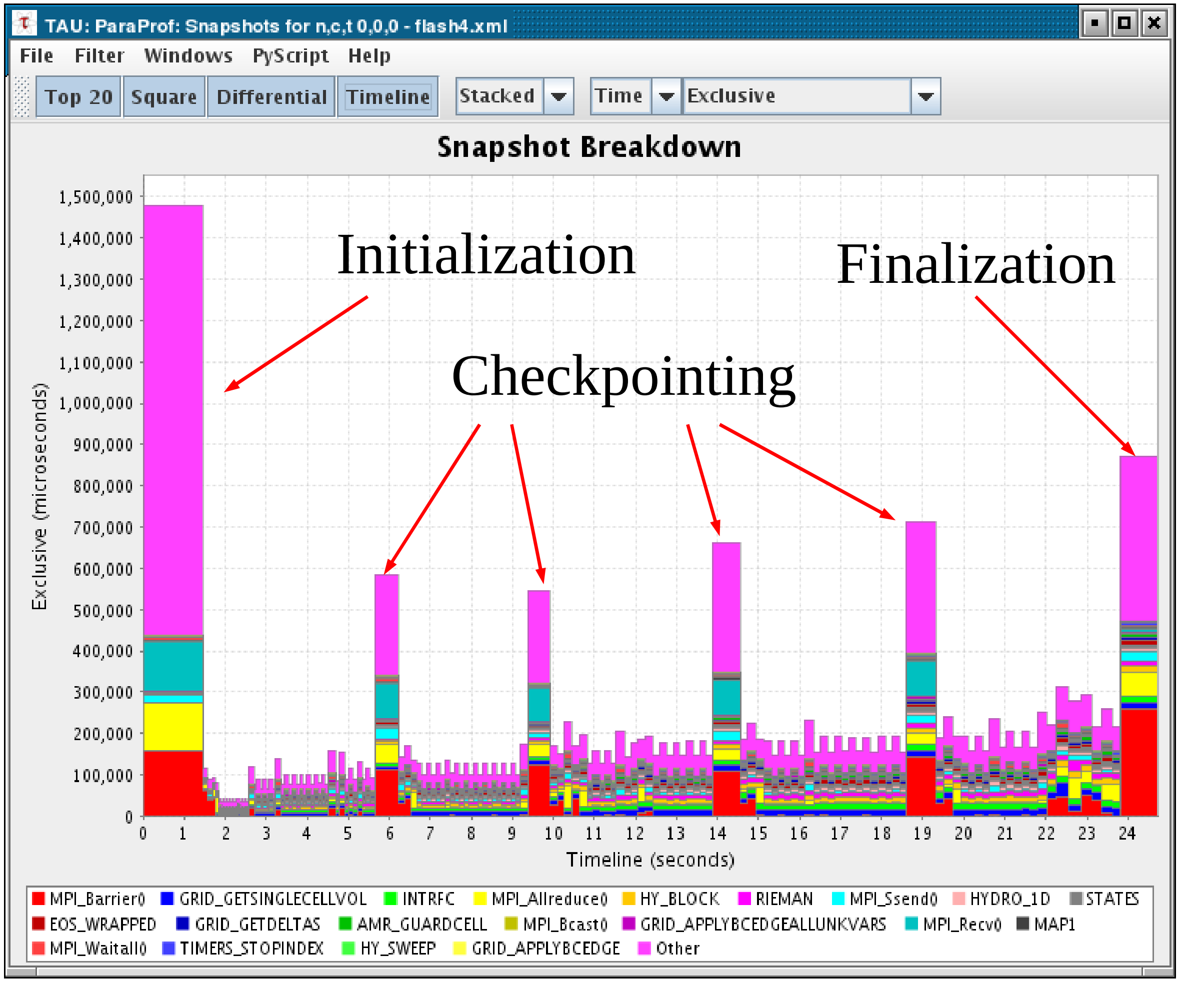




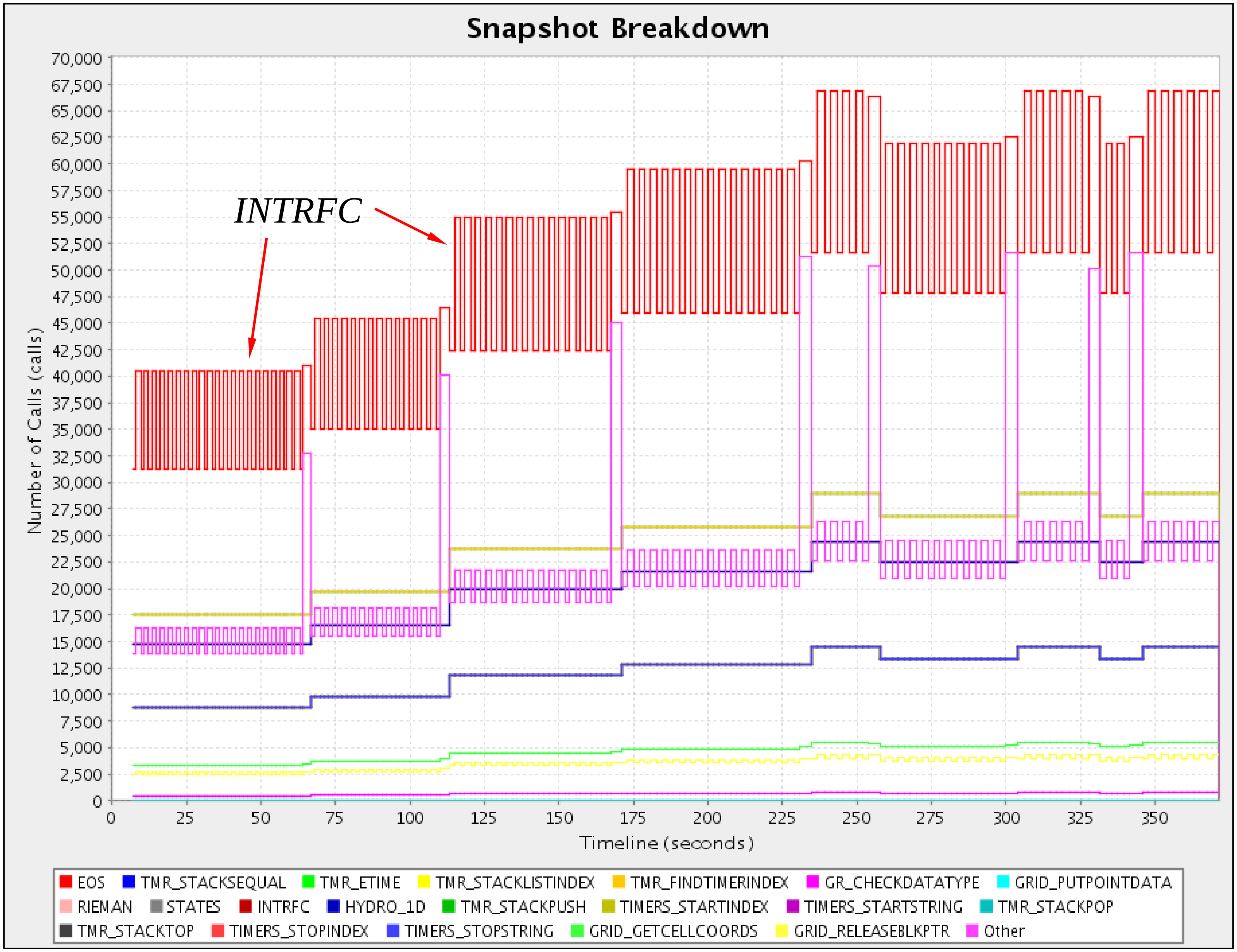
A. Morris, W. Spear, A. Malony, and S. Shende, "**Observing Performance Dynamics using Parallel Profile Snapshots**," *European Conference on Parallel Processing (EuroPar)*, 2008.

Parallel Profile Snapshots of FLASH 3.0 (UIC)

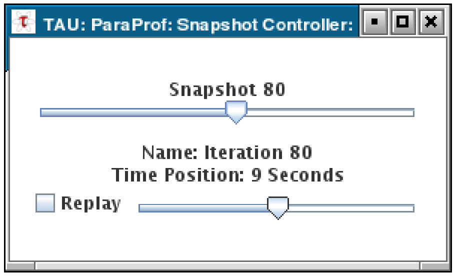
- Simulation of astrophysical thermonuclear flashes
- Highlight performance evolution and checkpointing



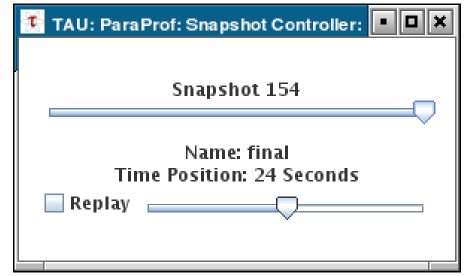
FLASH 3.0 Performance Dynamics

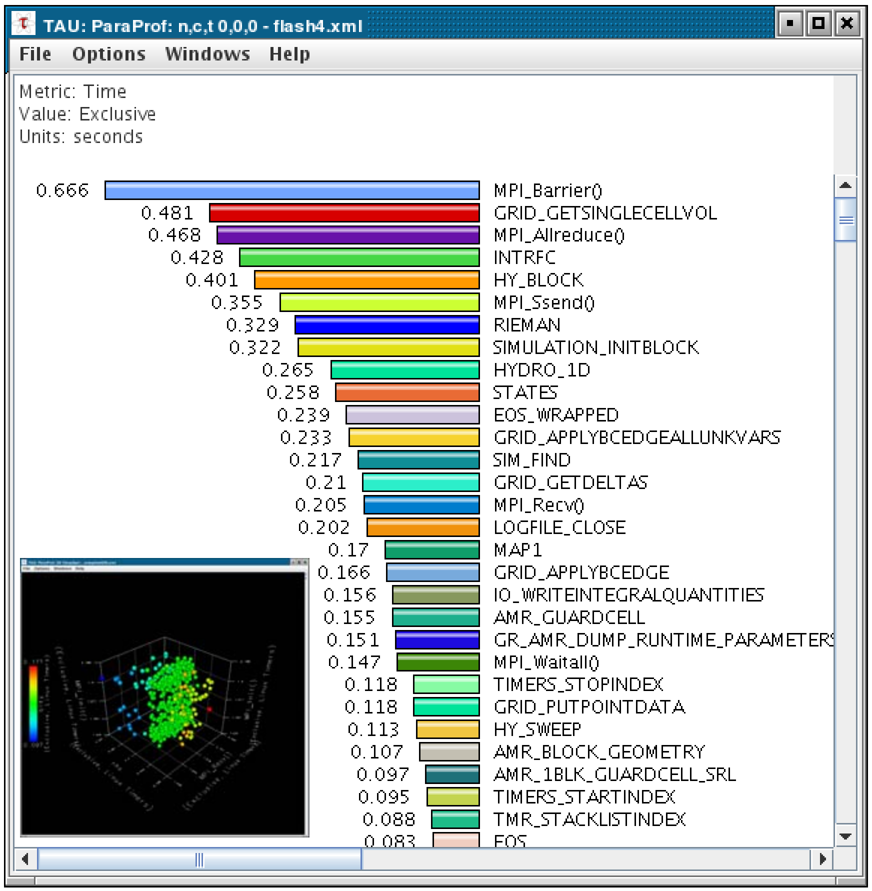


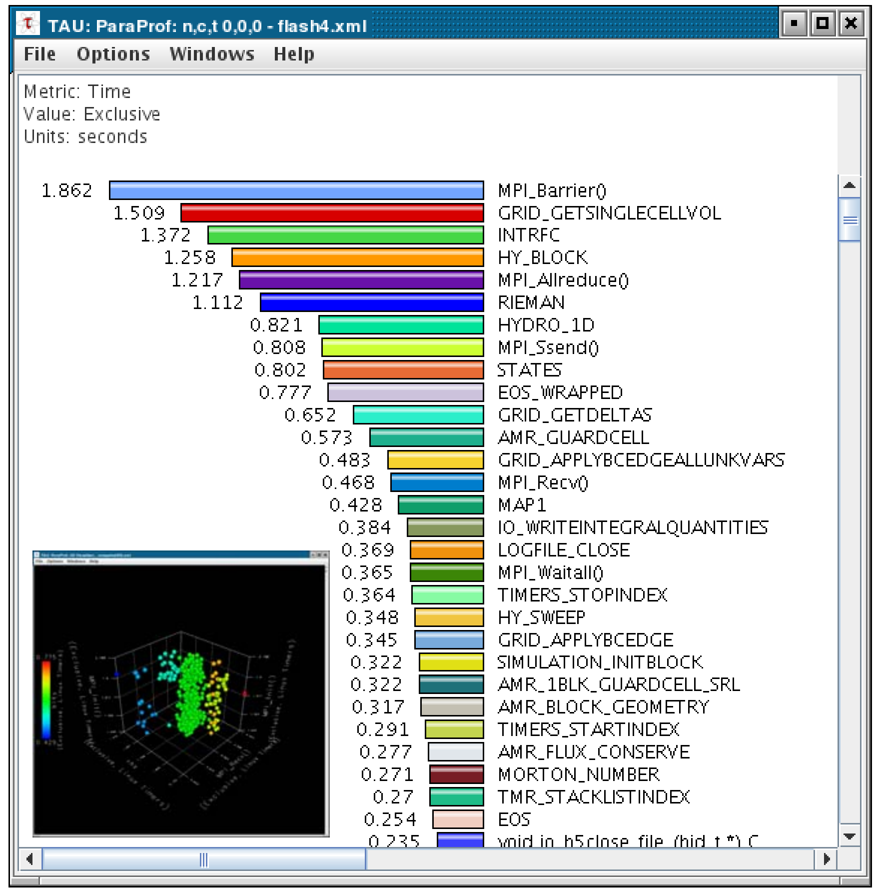
Snapshot Replay in ParaProf (FLASH 3.0)



All windows dynamically update







Profile Snapshot Performance

- Six minute run of FLASH on 128 processors
  - 321 instrumented events
  - Approximately 71 million event invocations per process
- Profiling only
  - 486KB (compressed)
  - 4.6% overhead
- Profiling with snapshots (~200 snapshots)
  - 320MB (2.6MB) uncompressed (58MB compressed)
  - 6.3% overhead
- Tracing (TAU)
  - 397GB uncompressed (3.1GB)
  - 130.3% overhead

Monitoring for Performance Dynamics

- Runtime access to parallel performance data
- Support for performance-adaptive, dynamic applications
- Monitoring modes
  - Online observation
  - Online observation with feedback into application
- Couple measurement with monitoring infrastructure
- TAUoverSupermon (UO, LANL)

> A. Nataraj, M. Sottile, A. Morris, A. Malony, and S. Shende, "**TAUoverSupermon: Low-overhead Online Parallel Performance Monitoring**," *Euro-Par*, 2007.

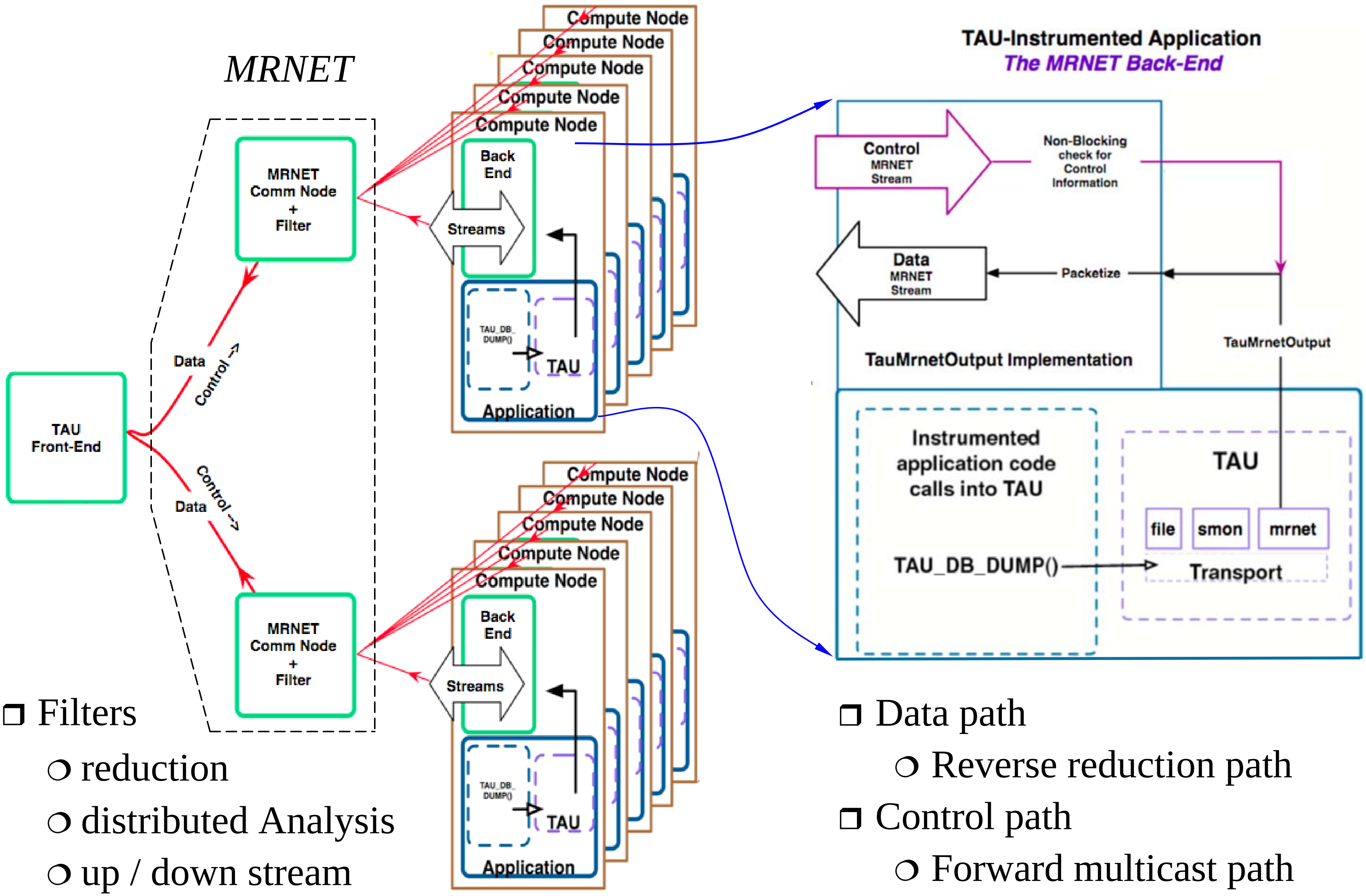
- TAUoverMRNET (UO, UWisconsin)

> A. Nataraj, A. Malony, A. Morris, D. Arnold, and B. Miller, "**A Framework for Scalable, Parallel Performance Monitoring using TAU and MRNet**," to appear in *Concurrency and Computation: Practice and Experience*, 2008.
>
> A. Nataraj, A. Malony, A. Morris, D. Arnold, and B. Miller, "**In Search of *Sweet-Spots* in Parallel Performance Monitoring**," Conference on Cluster Computing (Cluster 2008).

What is MRNet?

- ***M***ulticast ***R***eduction ***Net***work
  - Software infrastructure, API, utilities (written in C++)
  - Create and manage network overlay trees (TBON model)
  - Efficient control through root-to-leaf multicast path
  - Reductions (transformations) on leaf-to-root data path
  - Packed binary data representation
- Uses *thread-per-connection* model
  - Supports multiple concurrent “streams”
- Filters on intermediate nodes
  - Default filters (e.g., sum, average)
  - Loads custom filters through shared-object interface
- MRNet-base tools (Paradyn, STAT debugger, ToM)

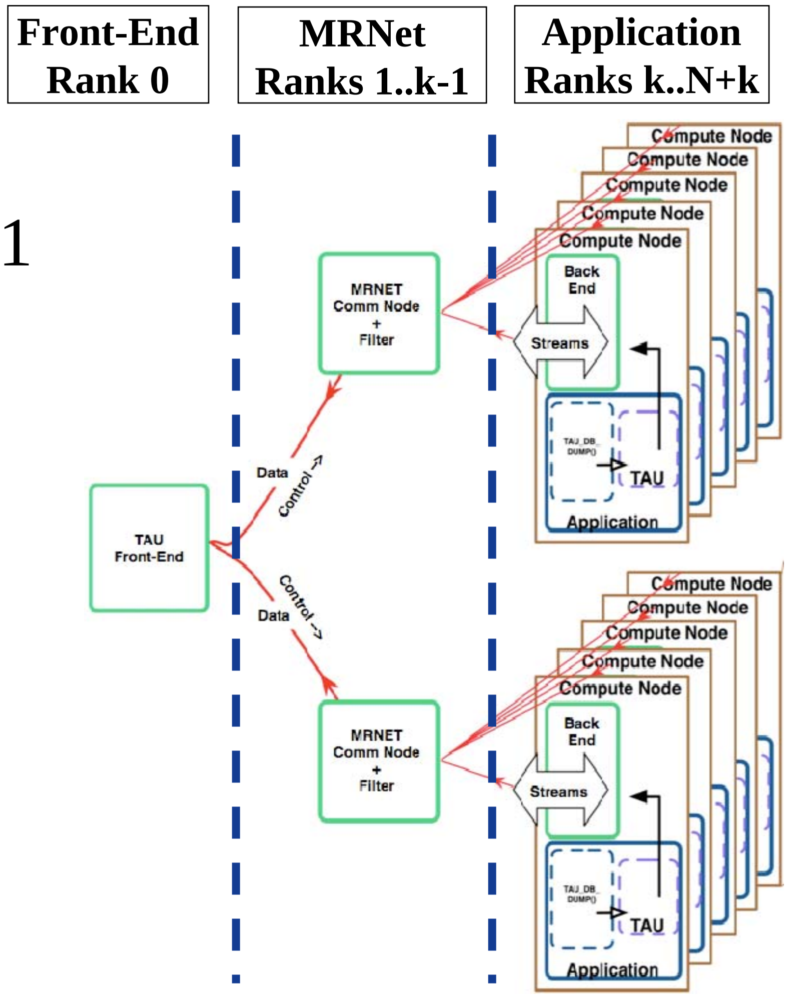
TAU over MRNet (ToM)



- Filters
  - reduction
  - distributed Analysis
  - up / down stream
- Data path
  - Reverse reduction path
- Control path
  - Forward multicast path

Transparent Monitor Instantiation

- Solution for MPI Applications
- Based on interception of MPI Calls
  - PMPI interface
- Separate roles
  - Tree: Rank-0 and Ranks 1..k-1
  - Application: Ranks k..N+k
- Three parts to method:
  - Initialization
  - Application execution
  - Finalization

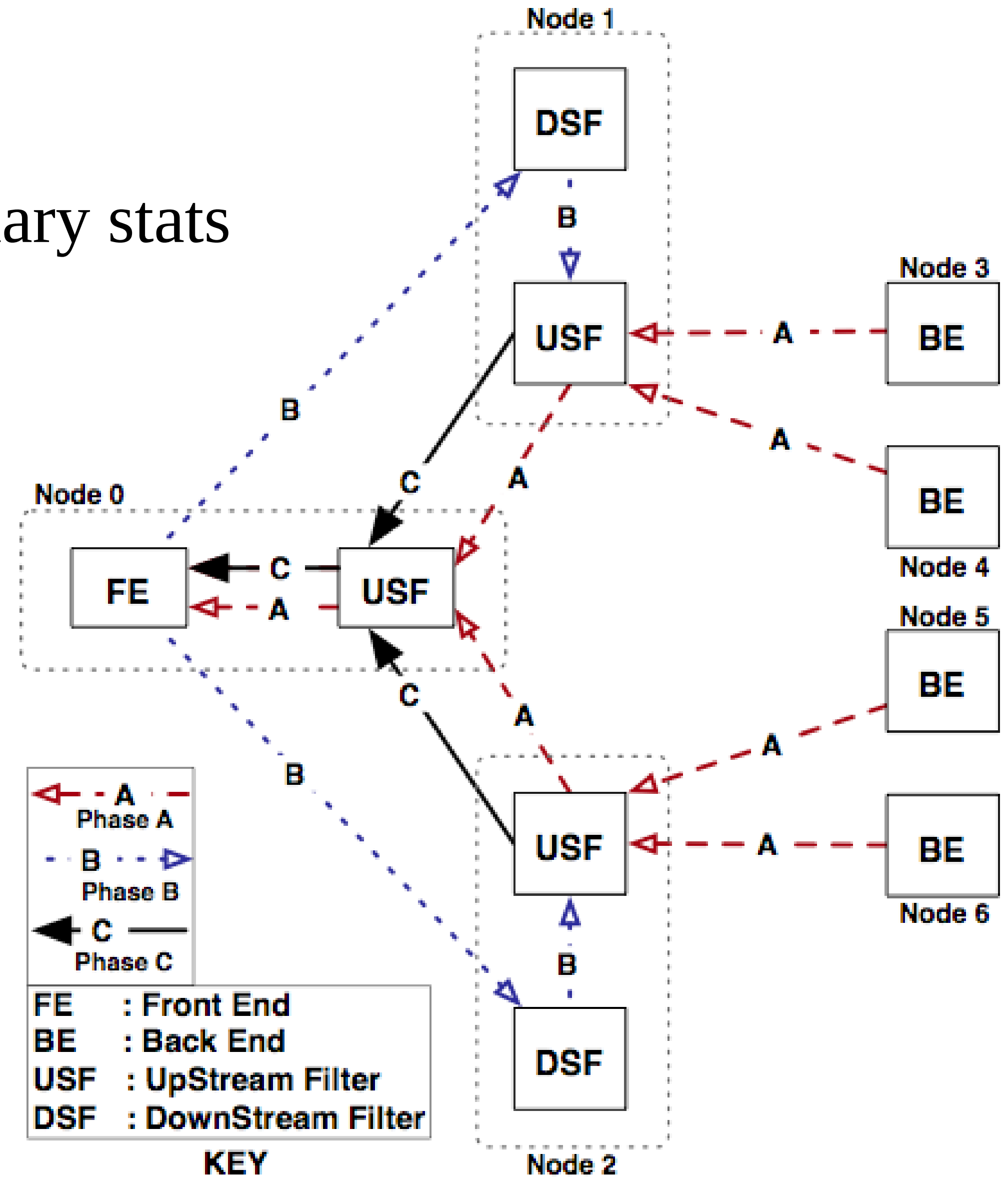


ToM Filters

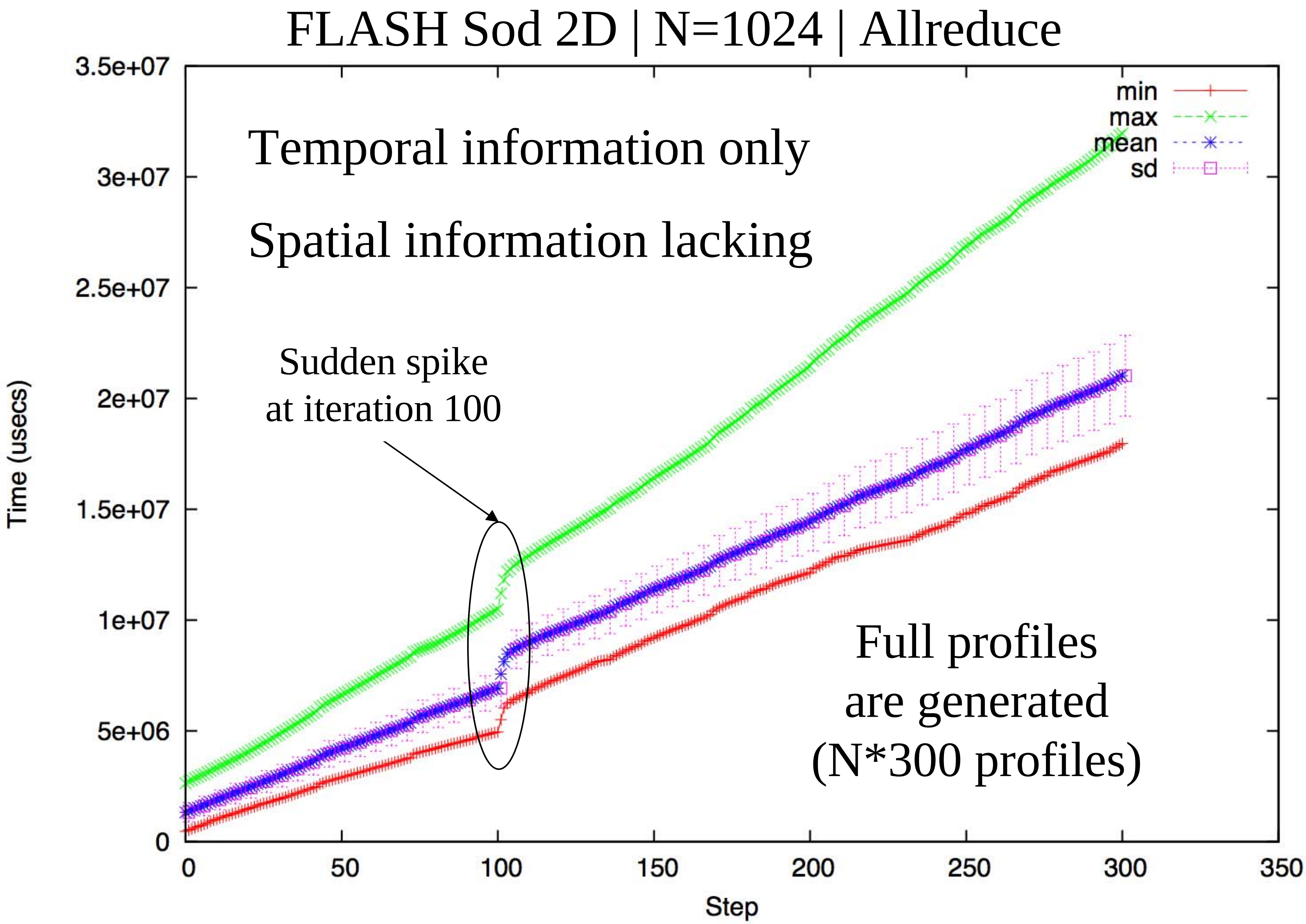
- Ideally there would be no need for filtering
  - Retrieve and store *all* performance data provided
  - Acceptability depends on performance monitor use
- High application intrusion, transport and storage costs
  - Need to trade-off queried performance data granularity
  - Which events, time intervals, application ranks?
- Reduce performance data as it flows through transport
  - Distribute FE analysis out to intermediate filters
- Three filtering schemes
  - *1-phase*: summary
  - *3-phase*: histogram, classification histogram
  - Progressive temporal/spatial detail with added complexity

ToM Distributed Analysis and Reduction

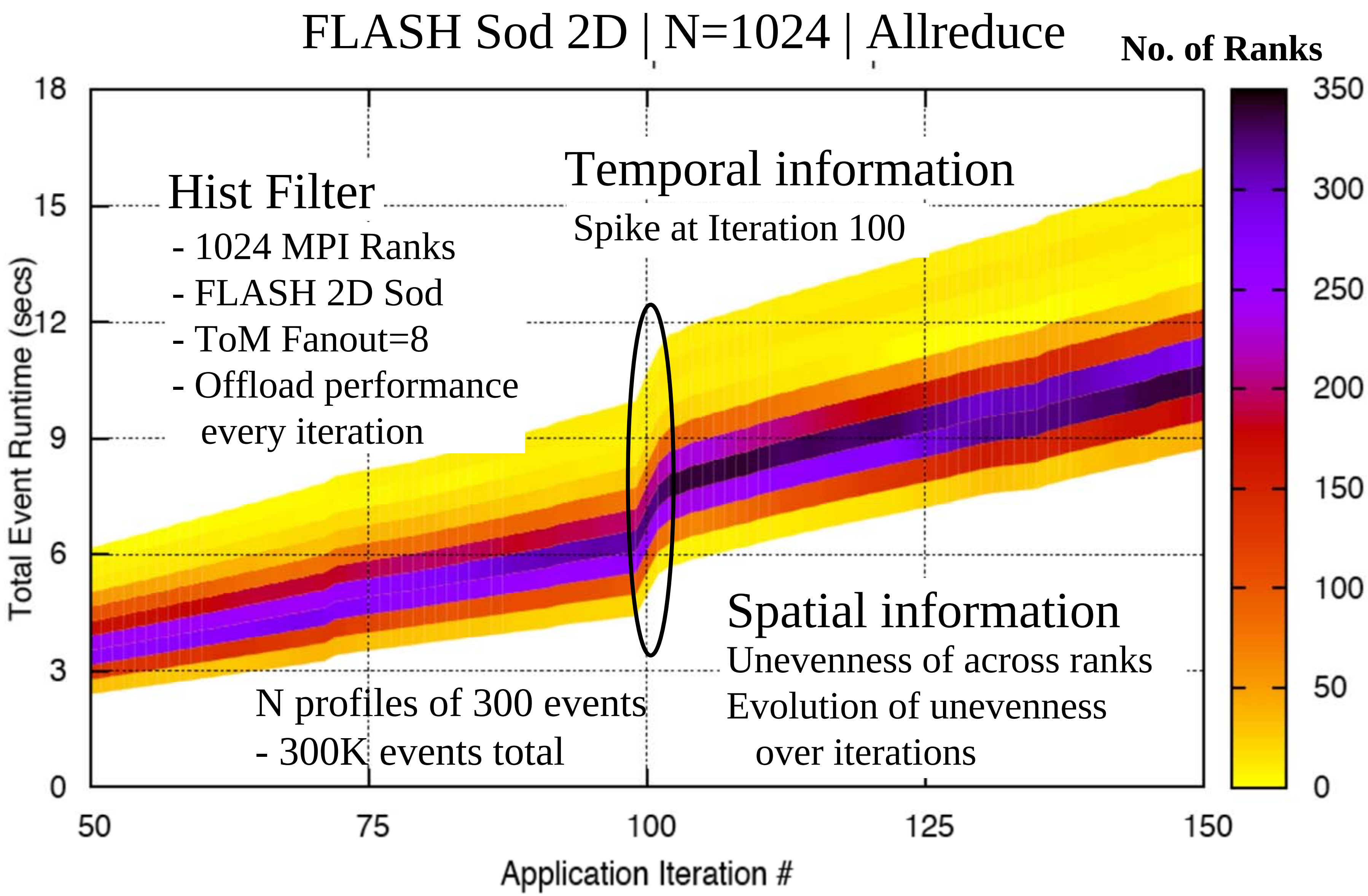
- *StatsFilter*
  - Upstream filter
  - Calculates global summary stats
  - Produces single profile
- *HistFilter*
  - 3-phase (up-down-up)
  - Global min/max needed
  - Variable # bins
- *ClassFilter*
  - Groups ranks into functional classes
  - Compute histograms



ToM Statistics Filter (FLASH, MPI_Allreduce)



ToM Histogram Filter (FLASH) Projection View

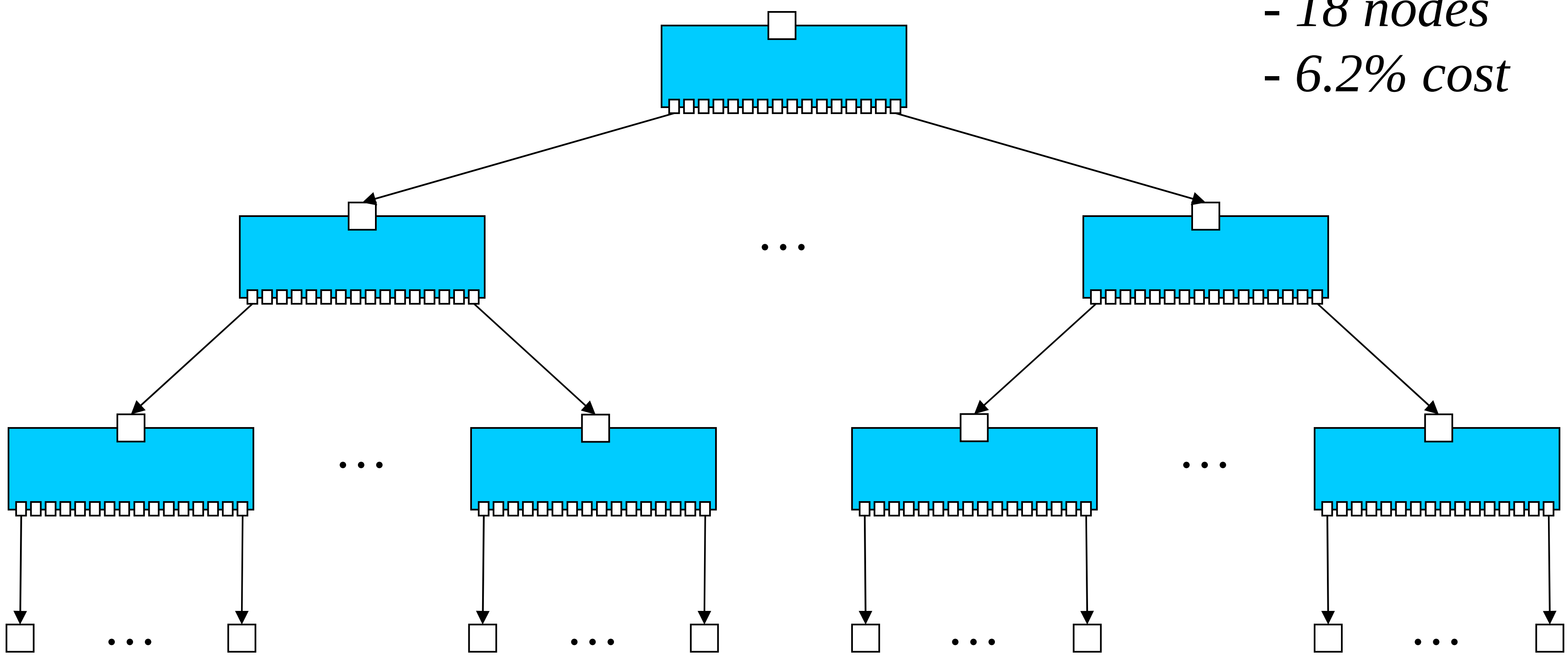


ToM on TACC Ranger (4096 cores)

ToM fanout = 16

273 cores for TBON
- 18 nodes
- 6.2% cost

Front-End

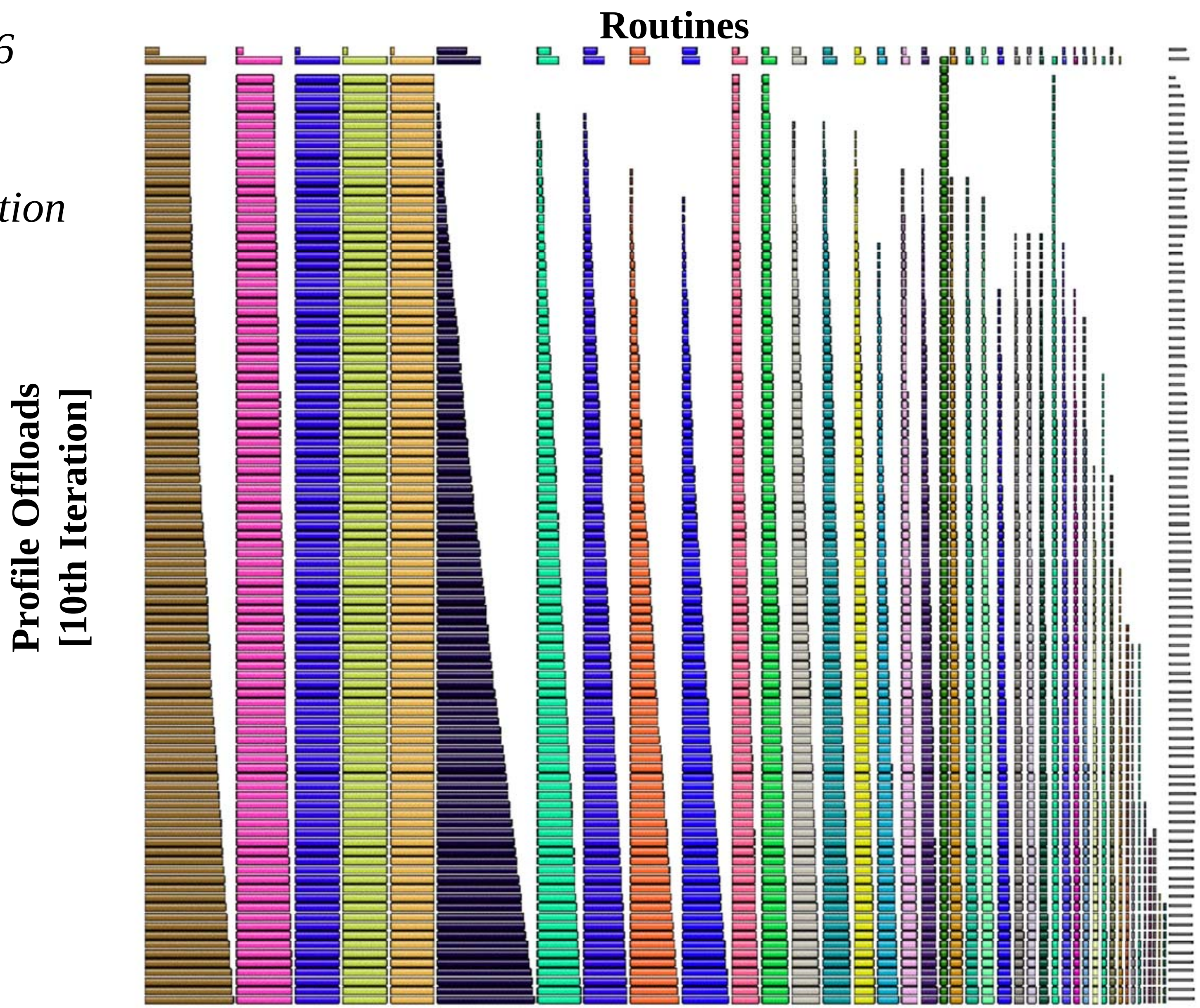


Back-End cores (4096)

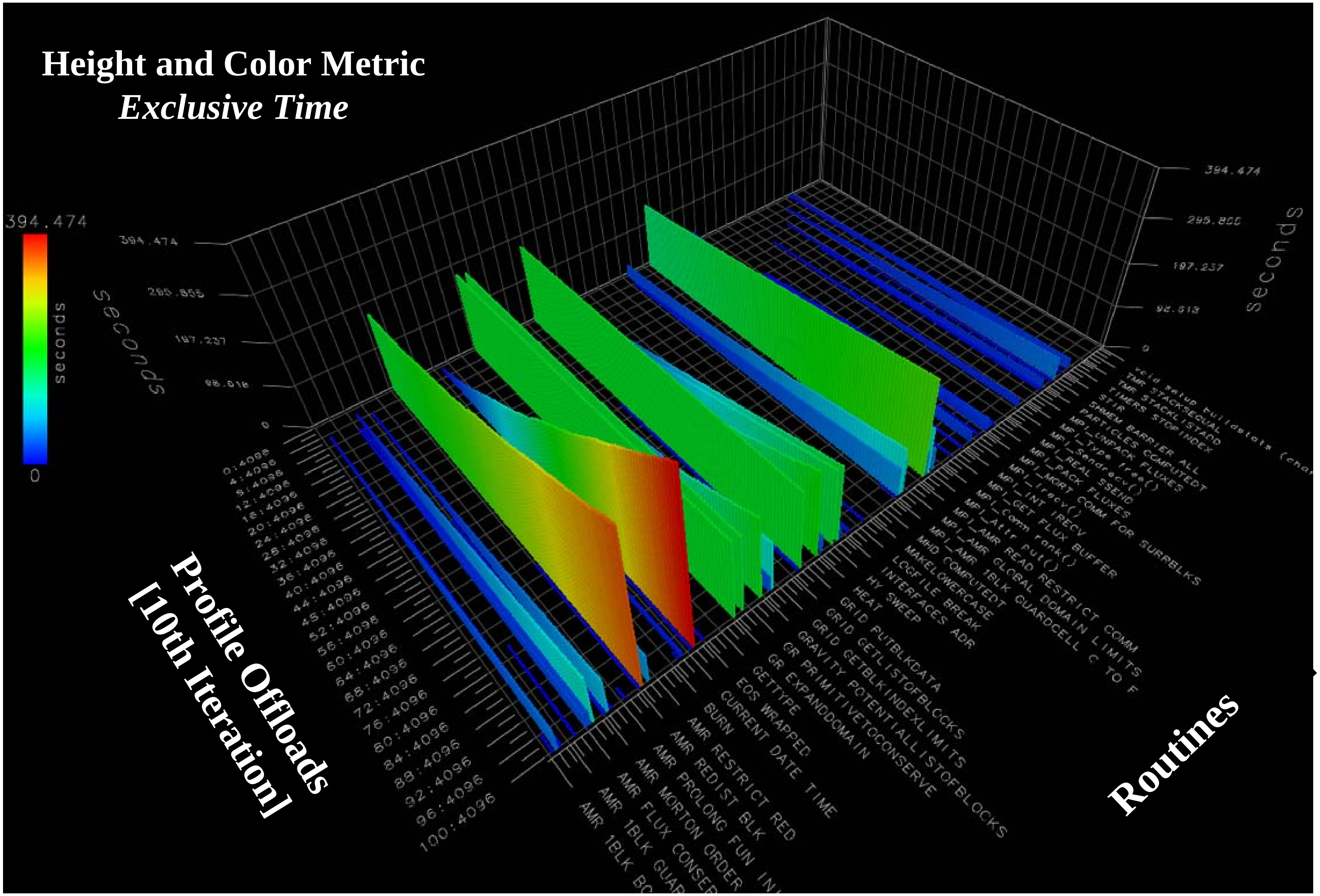
16 cores per Ranger node

Iteration Profile View (FLASH, Ranger, 4096 cores)

ToM fanout = 16
Statistics filter
200 events
Every 10th iteration



Iteration Profile 3D (FLASH, Ranger, 4096 cores)



Classified Histogram Filter

- Are there “classes” of ranks performing specific roles?
- Can we identify them from the performance profile?
- Definition of class
  - Class-id: hash of concatenated event-names
  - Ranks with same class-id belong to same class
  - Application-specific or tailored to observer’s wishes
  - Class-id generated based on call-depth or for MPI events
- Histograms generated within class
  - Output: set of histograms per-event, one for each class
- More detail than simple histograms
  - Trade-off detail from classification against the costs

Classification Filter : Paraprof Main View

Pick Single Offload #18 / Application Iteration 180

22 Functional Classes seen from 4096 ranks

Naming Classes: Offload# . Class# . # of Ranks in Class

8.18.2507 Example: Class 8 contains most (2507) ranks

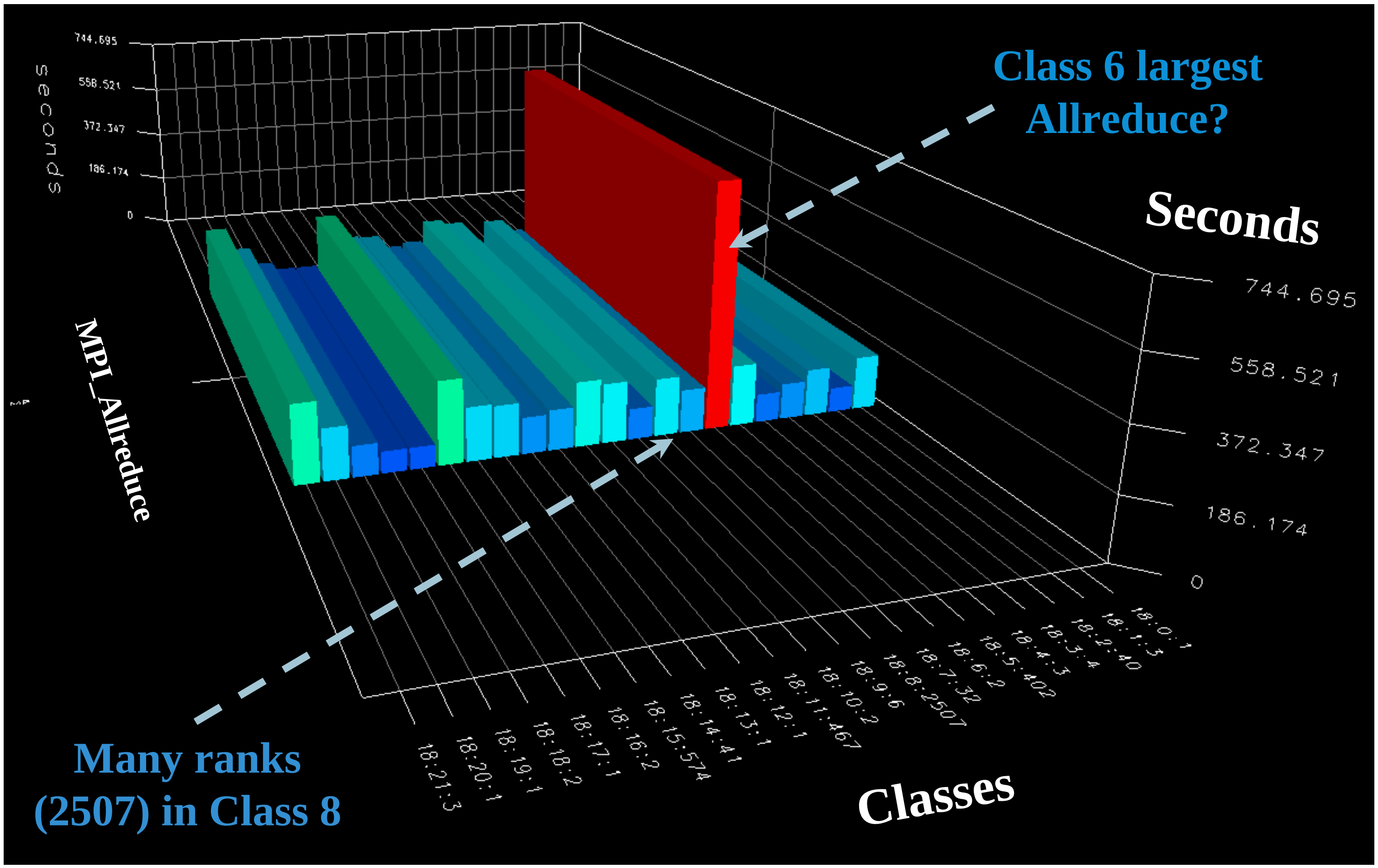
QuickTime™ and a
decompressor
are needed to see this picture.

Differing Routines / Counters

Classification Filter : Paraprof 3D View

Pick MPI_Allreduce() From Single Offload #18

Allreduce Time across Classes

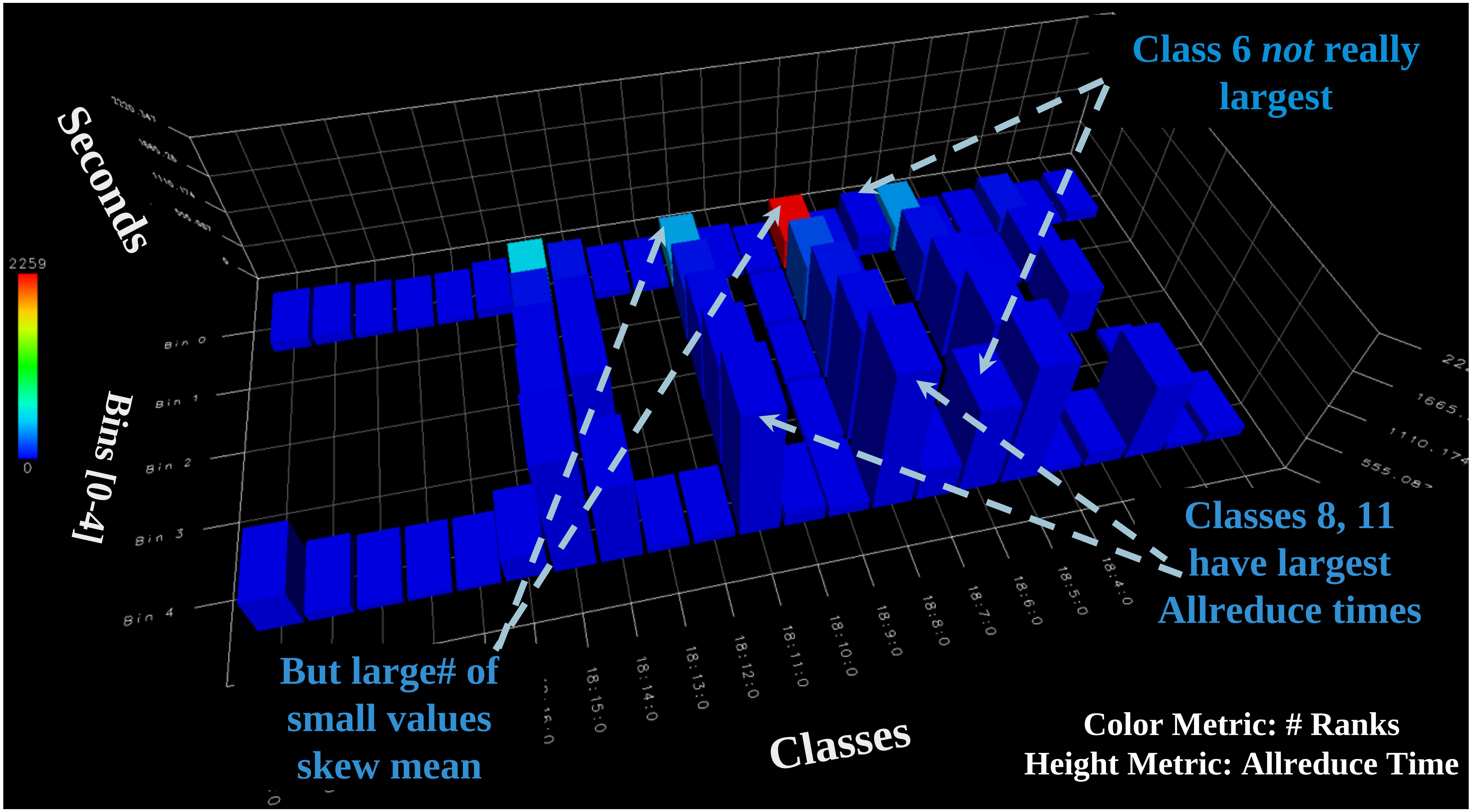


Classification Filter : Classified Histograms

Allreduce Time

5 Bin Histogram / Class

[Same Offload# 18 as before]

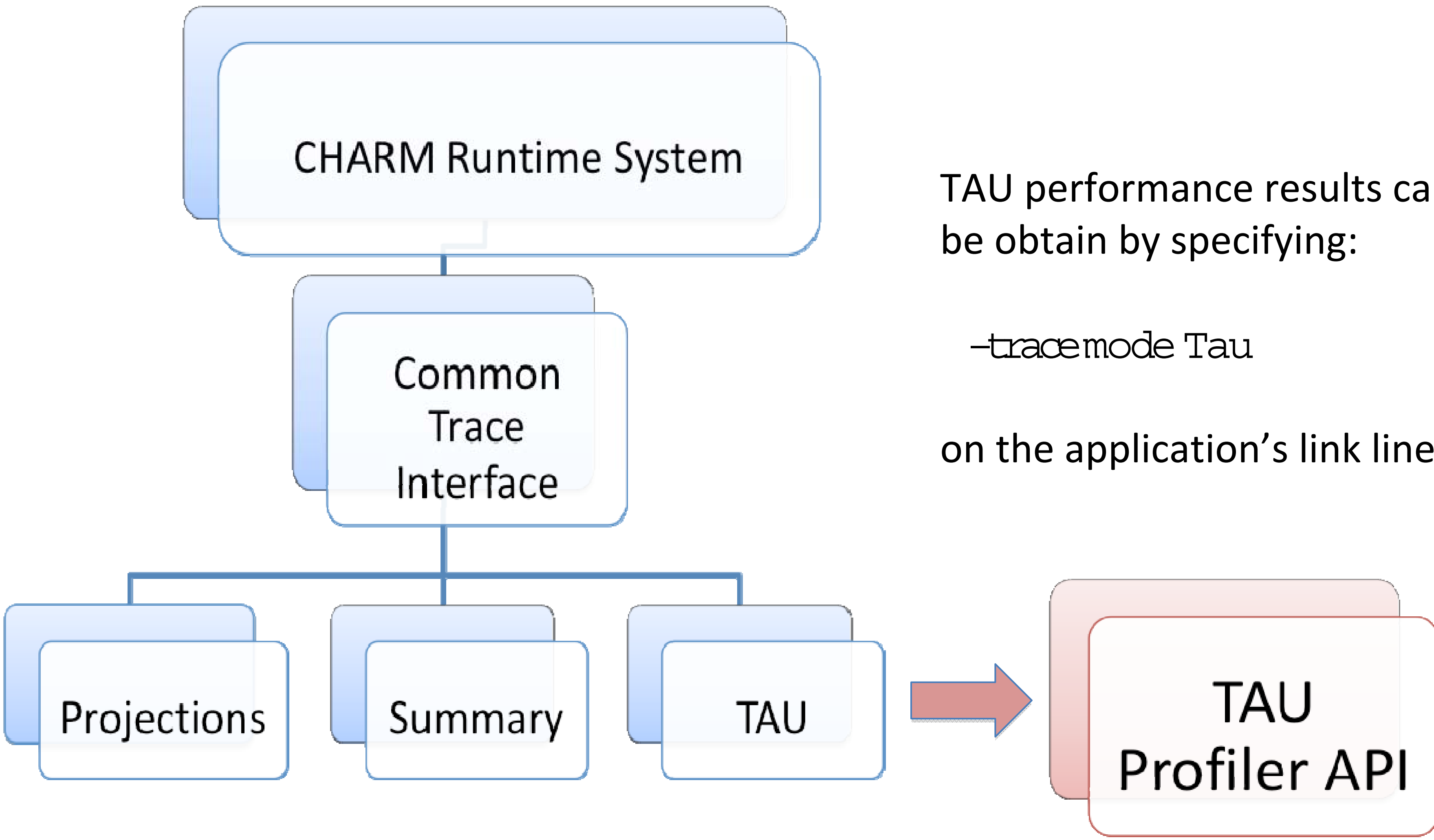


Parallel Programming Systems: Charm++

- Important to support performance measurement in the context of a parallel programming model and system
  - TAU working with DOE CCA project
  - Now want to look at Charm++
- Charm++ is a object-oriented parallel programming environment for scalable system based on C++
  - Medium-grained processes (called chares)
  - Interact with each other via messages
  - Program entry methods run as user-level threads
- Charm++ implements the *Projections* performance tool
  - Trace-based measurement and analysis
  - Uses a performance (tracing) callback interface
  - Trace buffer overflow can cause measurement intrusion

Charm++ Tracing Structure

- ❒ Can performance callback interface be used for TAU?



TAU performance results can be obtain by specifying:

–tracemode Tau

on the application’s link line

Charm Performance Call-back API

- Runtime performance (trace) class and methods
  - Trace class

    class TraceTau : public Trace { … }   void traceClose();
  - Begin/end Computation

    void beginComputation(void);

    void endComputation(void);
  - Begin/end Idle

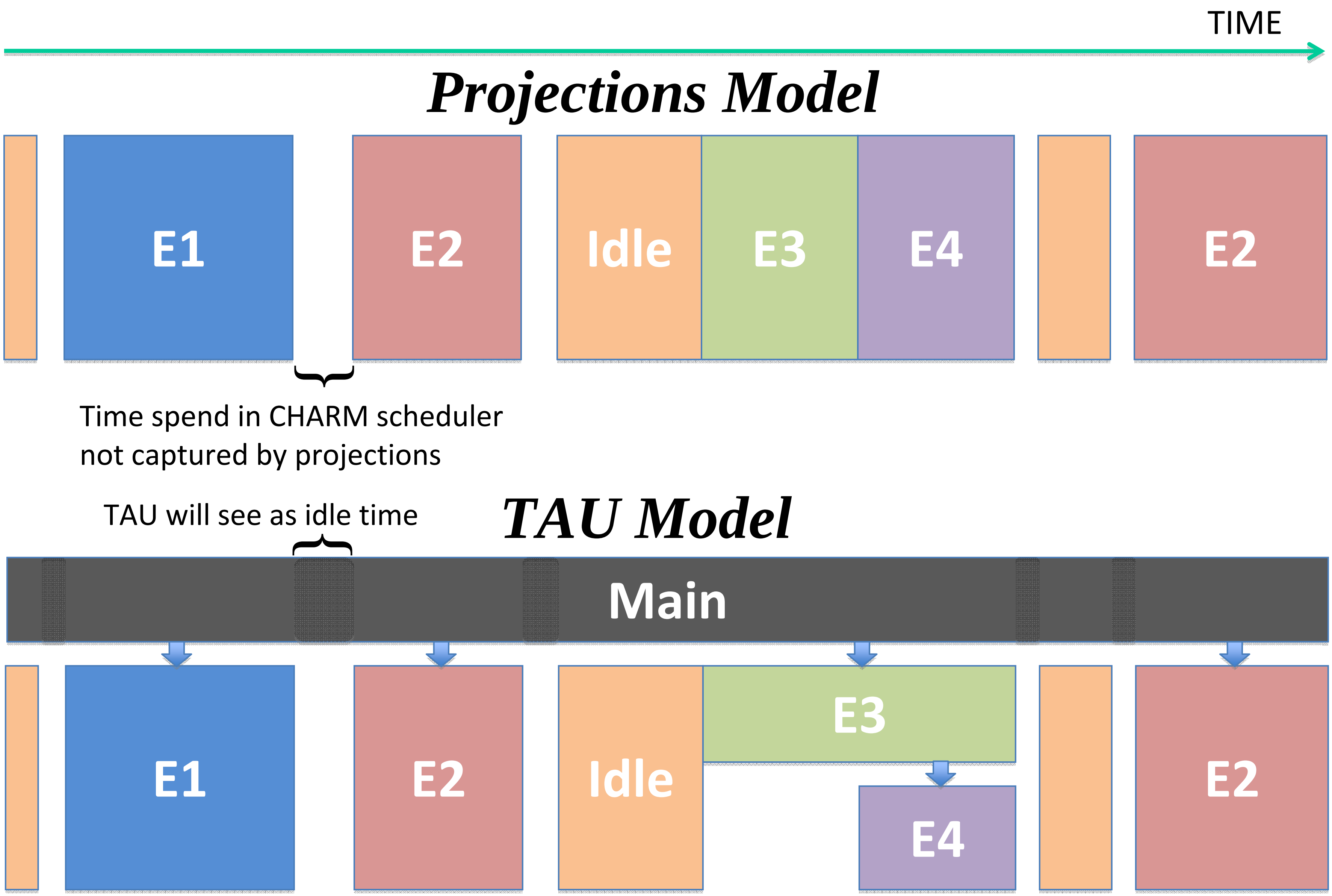
    void beginIdle();

    void endIdle();
  - Begin/end entry event (with user or charm specified IDs)

    void beginExecute();

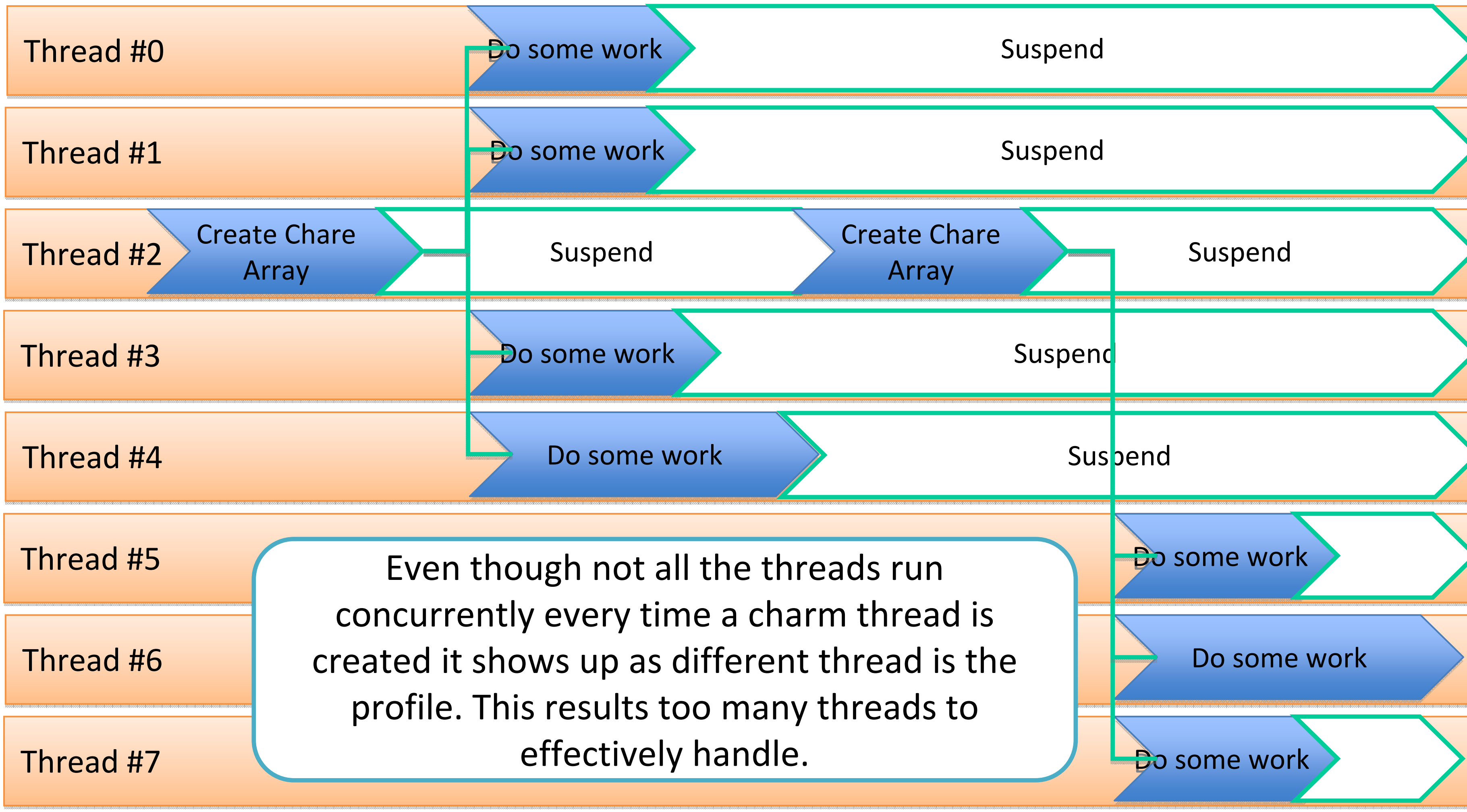
    void endExecute();
- Simple stack keeps track of the events

Event Model Comparison



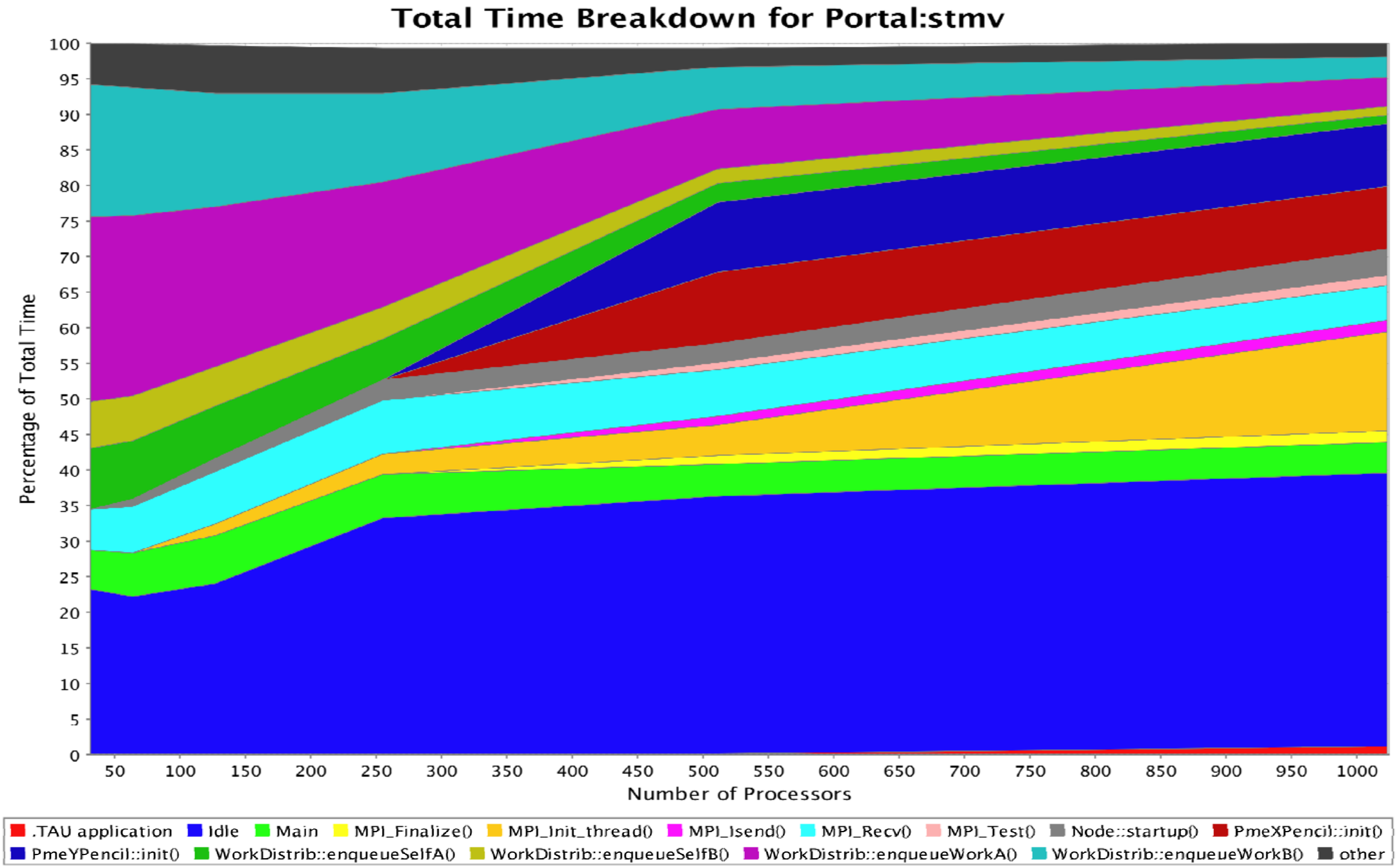
Charm's User-level Threads

- Entry events run as Charm user level threads
- TAU sees each user-level thread as a separate profile

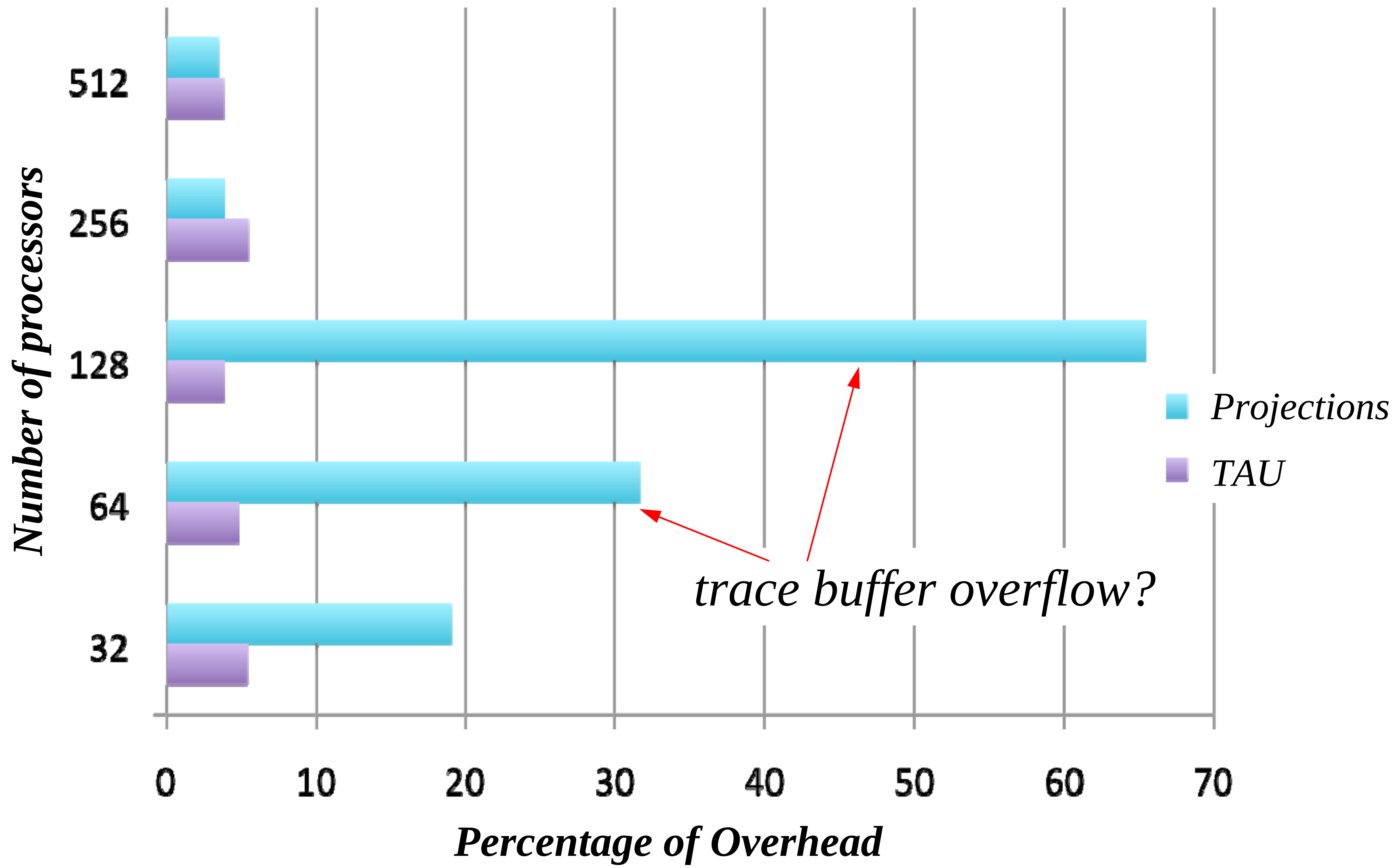


NAMD Runtime Breakdown

- Parallel MD for large biomolecular systems

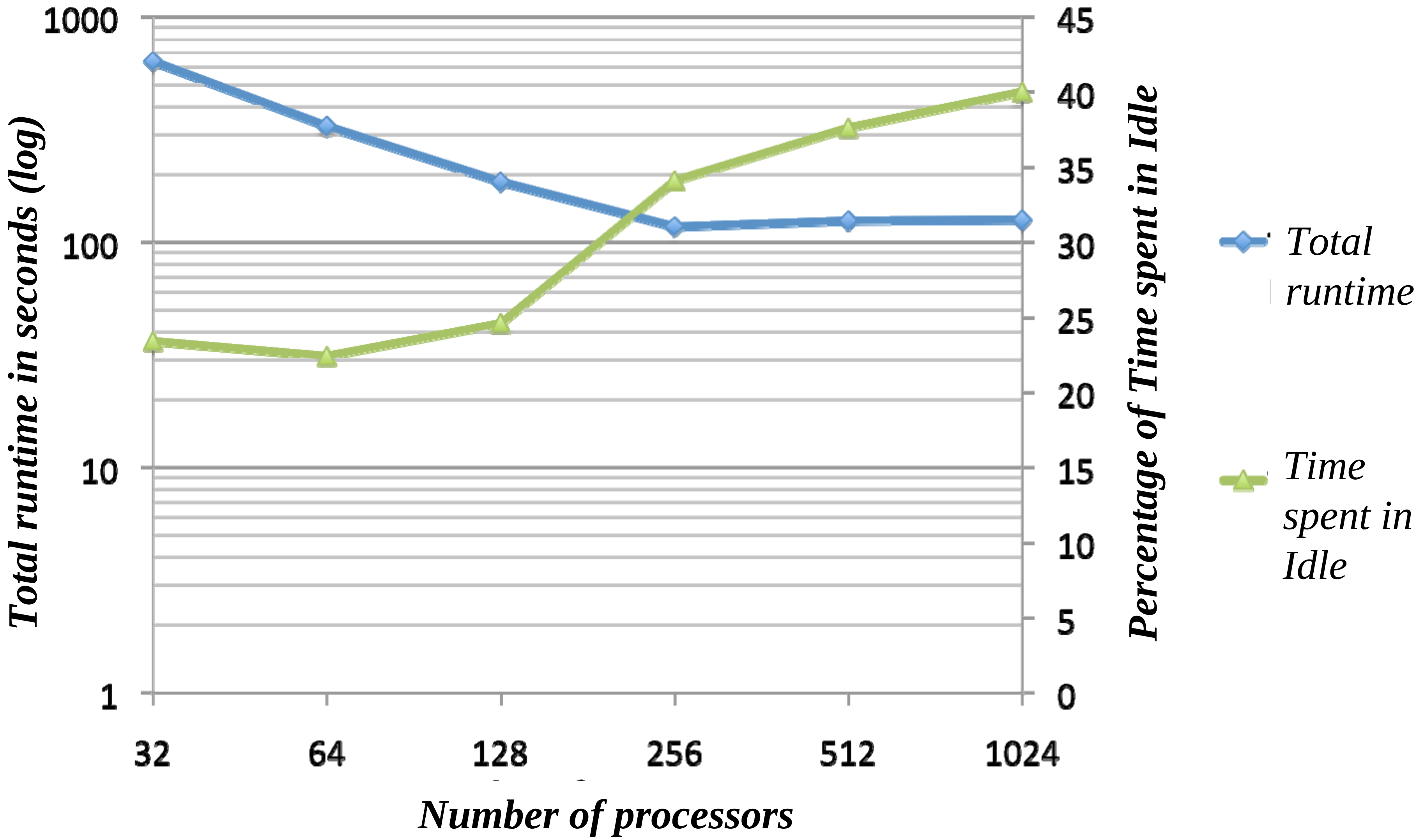


NAMD Overhead Comparison

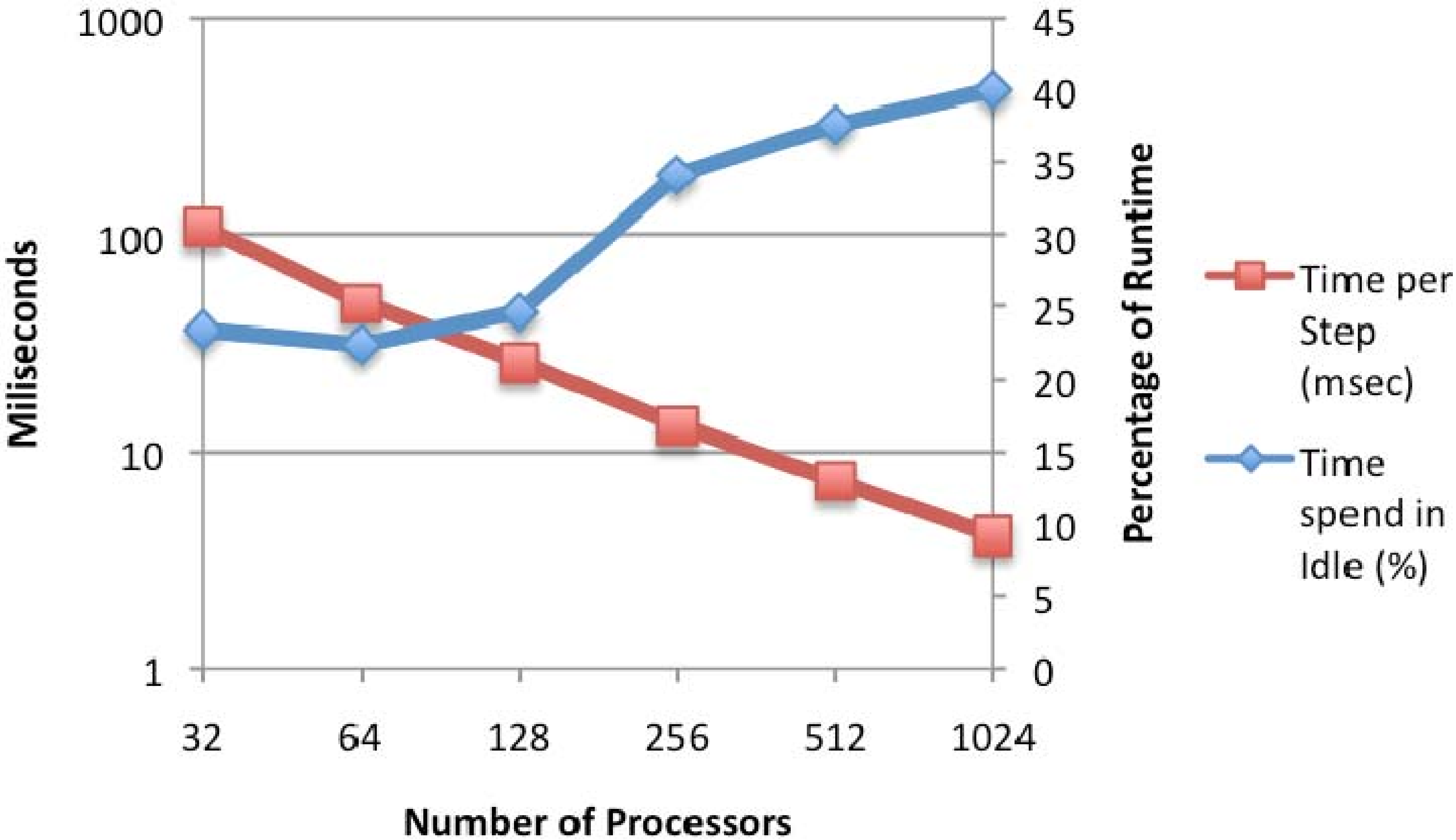


On Ranger (x86_64) stmv benchmark ~1 million atoms with PME

NAMD Scaling



NAMD Scaling

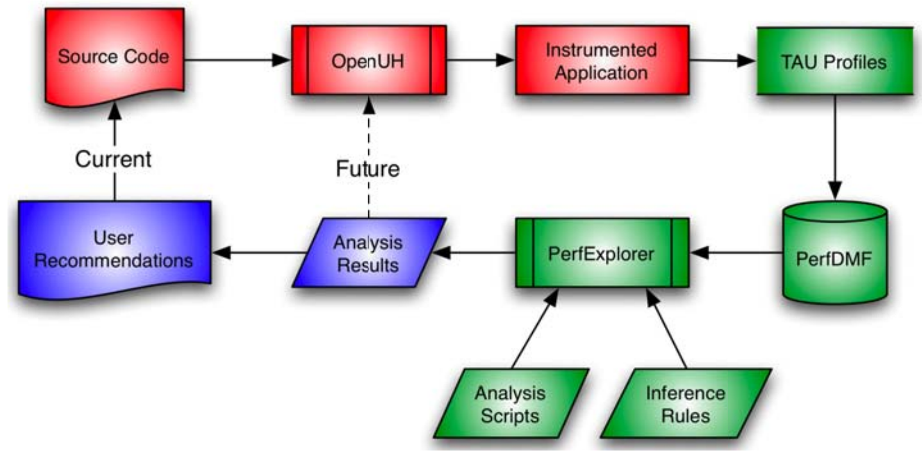


Application-level Profiling with Charm++

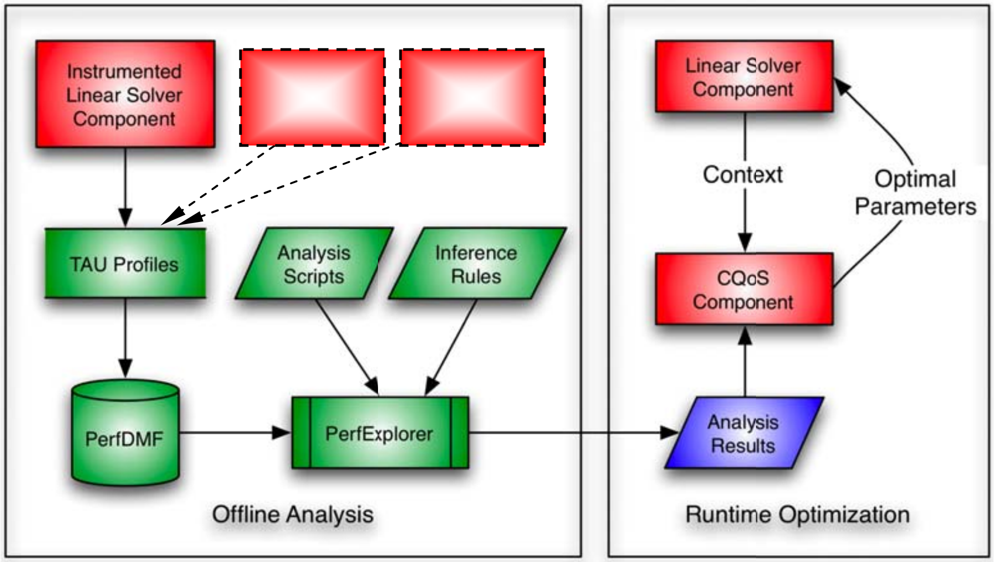
- Application-level instrumentation is possible with TAU
  - Application plus Charm++ runtime system events
- TAU sees Charm++ user-level threads
  - User-level threads have own profile in TAU
  - Number of user-level threads depends on chare parallelism
  - Unfortunately, multiplies the number of profiles
    - NAMD: 1024 * 150 user-level profile
- Tested TAU module with other Charm++ applications
  - ChaNGa
  - OpenAtom

PerfExplorer Applications

- OpenUH (Uhouston)
  - Feedback based compiler optimizations
  - Validate, improve cost model for loop optimization for OpenMP



- DOE SciDAC TASCS
  - CCA computational quality of service
  - Linear system solvers
  - Offline analysis
  - Online choice



Performance Data Mining: CQoS Recommender

- Component based scientific applications can have many algorithmic and hardware configuration options
- *CQoS Recommender System* can help eliminate educated guesswork or trial-and-error configuration for *optimal* (or near optimal) performance
  - Time to solution
  - Accuracy
  - Other quality measures
- Two types of applications we are working with:
  - Quantum chemistry (GAMESS, NWChem)
  - Iterative, non-linear solvers (PETSc)
- DOE TASCS SciDAC project (CCA)

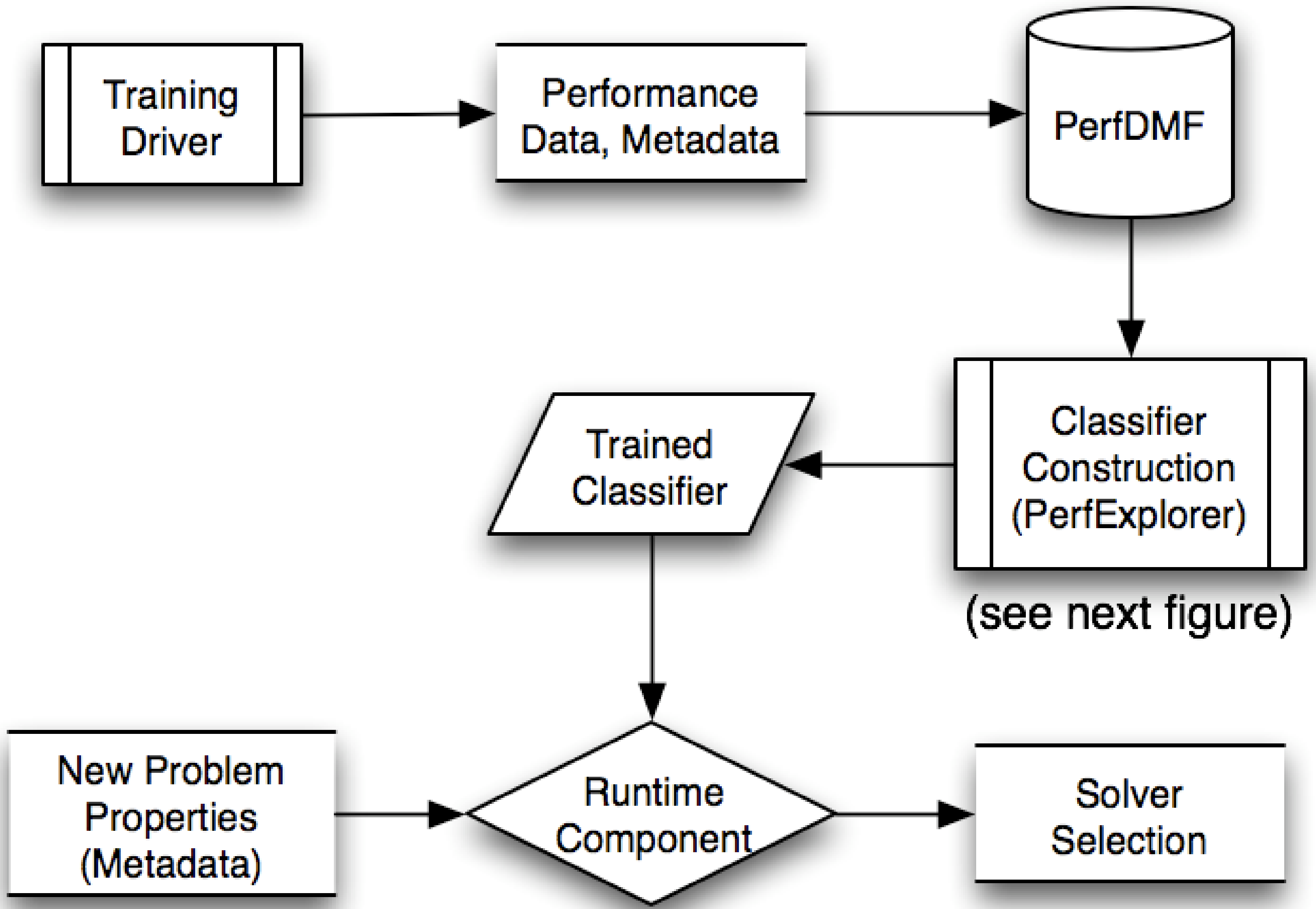
Iterative, Non-Linear Solvers (PETSc)

- Problem: sparse matrix of non-linear equations
- Solution: PETSc iterative, non-linear solvers
  - Iteratively call linear solvers
- Configuration problem
  - Convergence performance of the solvers
  - Factors:
    - properties of the matrix
    - properties of the hardware
    - type of linear solver used and options
    - type of pre-conditioner used and options
- Some combinations may never converge!

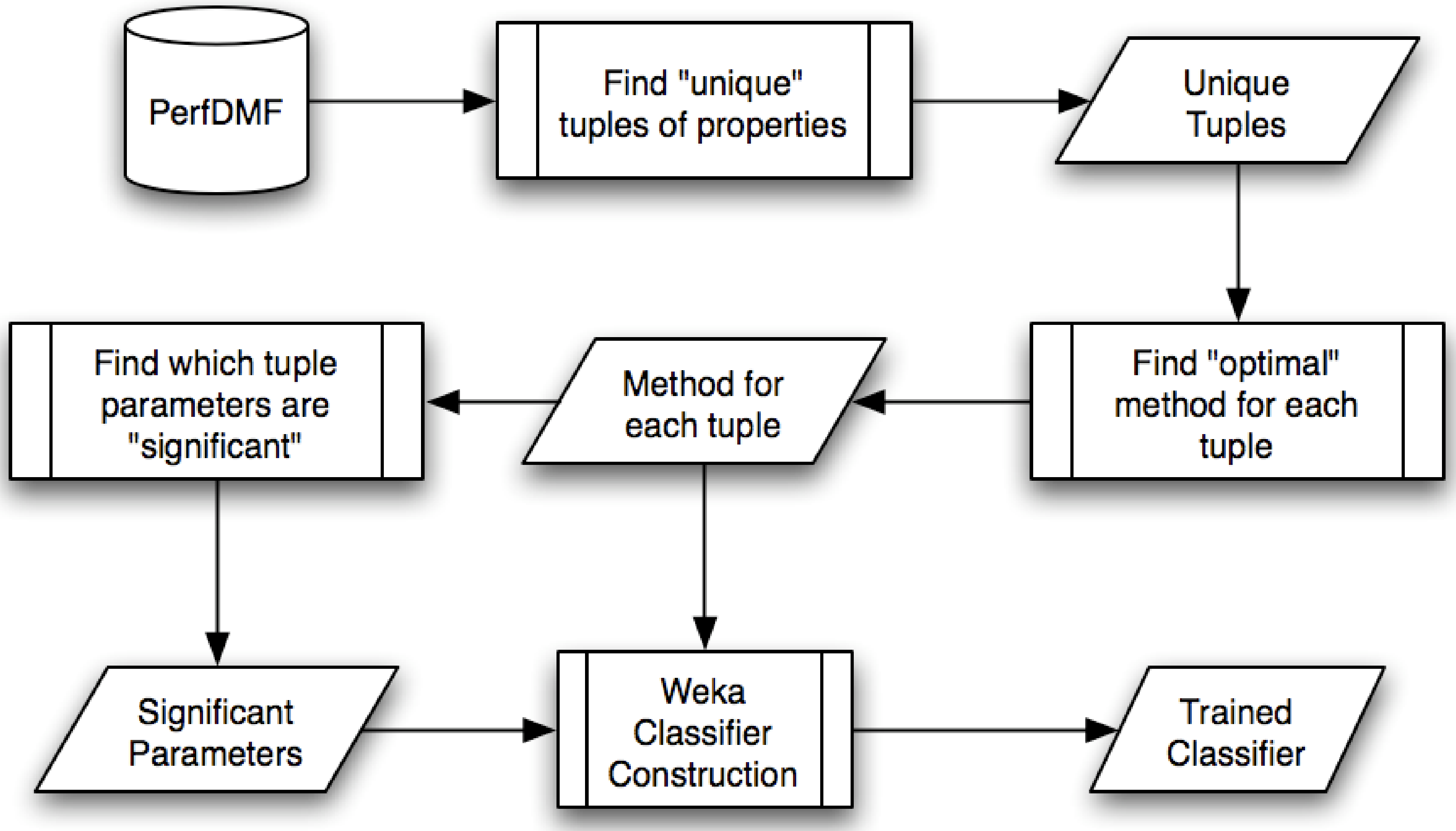
Quantum Chemistry (GAMESS)

- Goal: conduct performance evaluation for quantum chemistry packages, to learn optimal configurations
  - 7 test molecules: bz (benzine), bz-dimer, AT (a DNA base pair), np (naphthalene), np-dimer, CG (another DNA base pair), C60
  - 3 run types: energy, gradient, hessian
  - 2 scf types: rhf, uhf
  - 2 MP levels: 0, 2
  - 4 basis sets: CCD, N31-6-1-1, N31-6-1, N31-6
  - 2 methods: direct or conventional
  - 672 combinations, without even considering node/core counts, machine parameters, architectures…

Construction of Recommender System



Construction of Classifier (detail)



Implemented Classifiers:

Naive Bayes

Support Vector Machine (SVM)

Multilayer Perceptron

Preliminary Recommendation Results - GAMESS

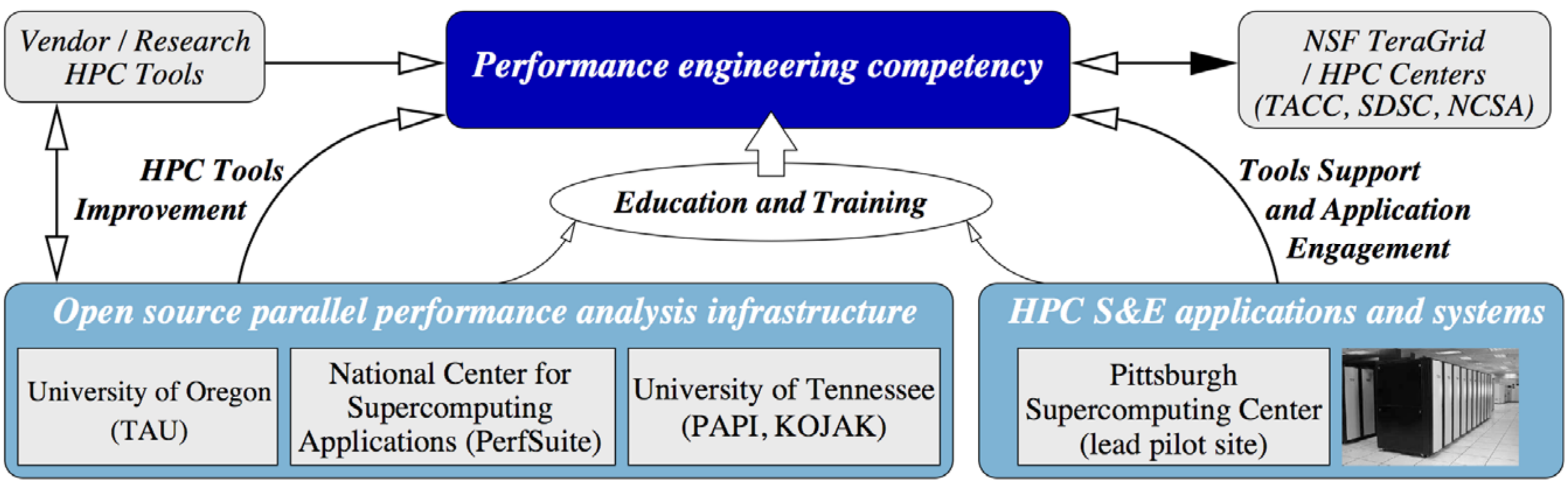
- With basis set = ccd, scf type = rhf, run type = energy
- Vary molecule, mp level, method and node counts
- Test machine: bassi.nersc.gov: IBM POWER5, 111 compute nodes with 8 cores, 32GB memory per node
- Direct method is faster than conventional for some molecules at higher node counts, but it varies

Run type = energy, core count = 8, scf type = rhf, basis set = ccd

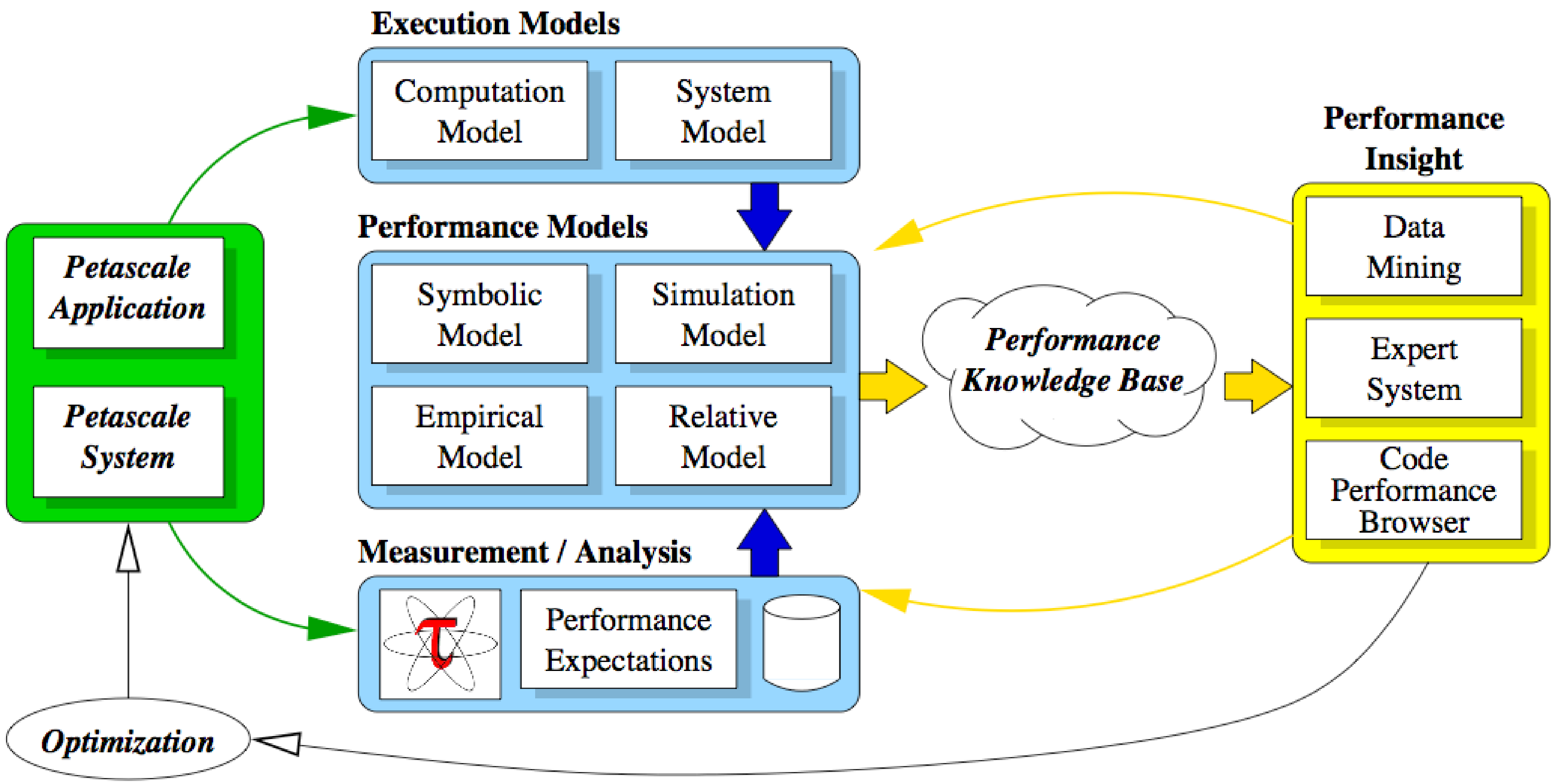
	MP level = 0						MP level = 2					
Molecule / Nodes	**1**	**2**	**4**	**8**	**16**	**32**	**1**	**2**	**4**	**8**	**16**	**32**
AT	con	con	con	con	dir	dir	con	con	con	con	dir	dir
bz	con	con	dir	dir	dir	dir	con	con	con	dir	dir	dir
bz-dimer	con	con	con	con	dir	dir	con	con	con	con	dir	dir
C60	con	con	con	dir	dir	dir	con	con	con	dir	dir	dir
GC	con	con	con	con	dir	dir	con	con	con	con	dir	dir
np	con	con	con	con	dir	dir	con	con	con	con	dir	dir
np-dimer	con	con	con	con	dir	dir	con	con	con	con	dir	dir

NSF POINT Project

- "High-Productivity Performance Engineering (Tools, Methods, Training) for NSF HPC Applications"
  - NSF SDCI program (Software Improvement and Support)
  - Productivity from Open, Integrated Tools (POINT)
- Robust performance technology
- Performance engineering best practices



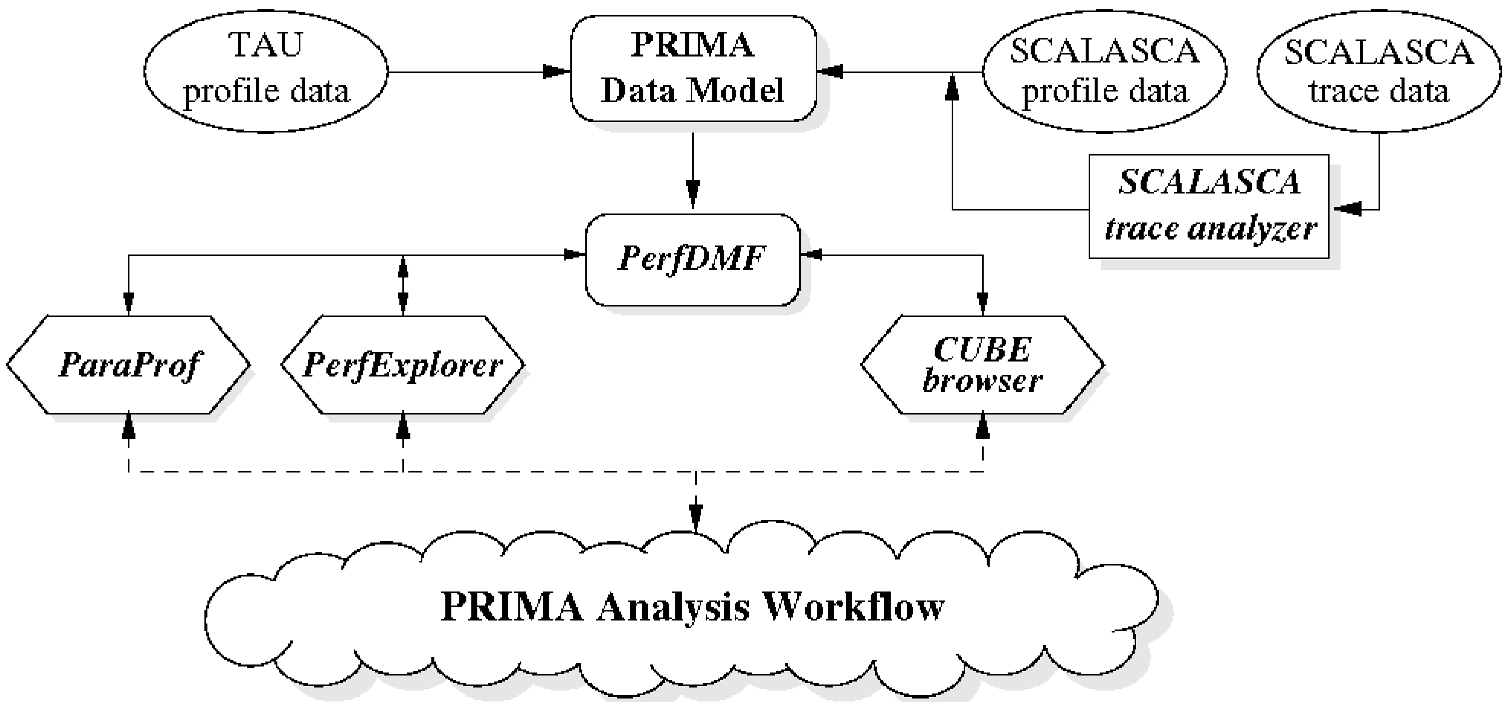
Model Oriented Global Optimization (MOGO)



- Proposal to DOE Software Development Tools for Improved Ease-of-Use of Petascale Systems

DOE PRIMA Project

- Performance Refactoring of Instrumentation, Measurement, and Analysis (PRIMA) Technologies
- Integration of leading direct measurement tools
  - TAU performance system
  - Scalasca



Conclusion

- Performance problem solving (process, technology)
- Evolve process and technology to meet scaling challenges
- Closer integration with application development and execution environment
- Raise the level of performance problem analysis
  - Scalable performance monitoring
  - Performance data mining and expert analysis
  - Whole performance evaluation
- The parallel performance tools community is small
  - Refactoring of core components and integration

Support Acknowledgements

- Department of Energy (DOE)
  - Office of Science
    - MICS, Argonne National Lab
  - ASC/NNSA
    - University of Utah ASC/NNSA Level 1
    - ASC/NNSA, Lawrence Livermore National Lab
- Department of Defense (DoD)
  - HPC Modernization Office (HPCMO)
- NSF Software Development for Cyberinfrastructure (SDCI)
- Research Centre Juelich
- Los Alamos National Laboratory
- Technical University Dresden
- ParaTools, Inc.

QuickTime™ and a TIFF (Uncompressed) decompressor are needed to see this picture.