TAUg: Runtime Global Performance Data Access using MPI

Kevin A. Huck, Allen D, Malony,
Sameer Shende and Alan Morris

{khuck,malony,sameer,amorris}@cs.uoregon.edu

Performance Research Laboratory
Department of Computer and Information Science
University of Oregon
http://www.cs.uoregon.edu/research/tau/

Motivation

- **Growing interest in adaptive computations**
  - Decisions at runtime based on dynamic state
  - Application adapts to current execution conditions
    - problem state and performance
- **Scalable parallel performance measurements**
  - Good solutions exist for parallel profiling and tracing
    - profiling: Dynaprof, mpiP, HPMToolkit, TAU
    - tracing: KOJAK, Paraver, TAU
- **Use of performance tools in adaptive applications**
  - Integrate with performance measurement infrastructure
  - Requires means to access performance data *online*

Need For A Runtime Global Performance View

- **Performance data collected locally and concurrently**
  - Scalable efficiency dictates “local” measurements
    - save data in processes or threads (“local context”)
  - Done without synchronization or central control
- ***Parallel performance state* is globally distributed**
  - Logically part of application’s global data space
  - *Offline* tools aggregate data after execution
  - *Online* use requires access to *global performance state*
- **How does an application access performance data?**
  - Abstract interface (*view*) to global performance state
  - Extend performance measurement system to collect data

Related Work

- **Performance monitoring**
  - OCM / OMIS [Wismuller1998]
  - Peridot [Gerndt2002]
  - Distributed performance consultant [Miller1995, Roth2006]
- **Computational steering**
  - Falcon [Gu1995]
  - Echo [Eisenhauer1998]
- **Online performance adaption**
  - Autopilot [Ribler2001]
  - Active harmony [Tapus2002]
  - Computational Quality of Service [Norris2004, Ray2004]

Solutions Approaches

- **Any solution needs to take into consideration:**
  - Granularity of global performance data to be accessed
  - Cost of observation
- **Approach 1: Build on global file system**
  - Tool *dumps* performance data to the filesystem
  - Application processes read from file system
- **Approach 2: Build on a *collection network***
  - Additional threads, processes or daemons (external)
  - Gather and communicate performance data
  - MRNet (Paradyn) [Arnold2005] and Supermon [Sottile2005]
- **Our approach: Build on MPI and TAU (TAUg)**

Tarzan and Taug

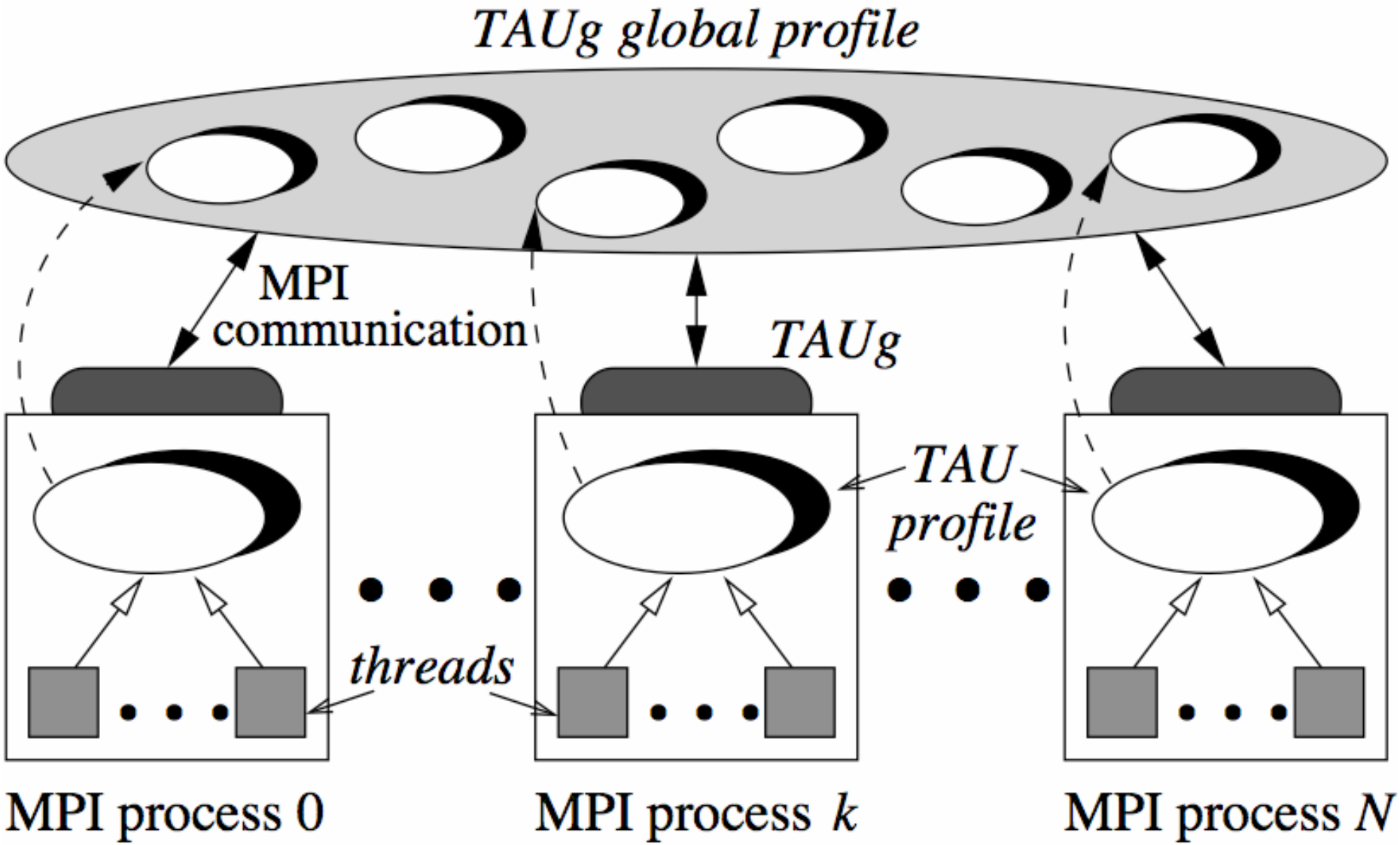
"The ape-man's eyes fell upon Taug, the playmate of his childhood, … Tarzan knew, Taug was courageous, and he was young and agile and wonderfully muscled."

Jungle Tales of Tarzan
Edgar Rice Burroughs

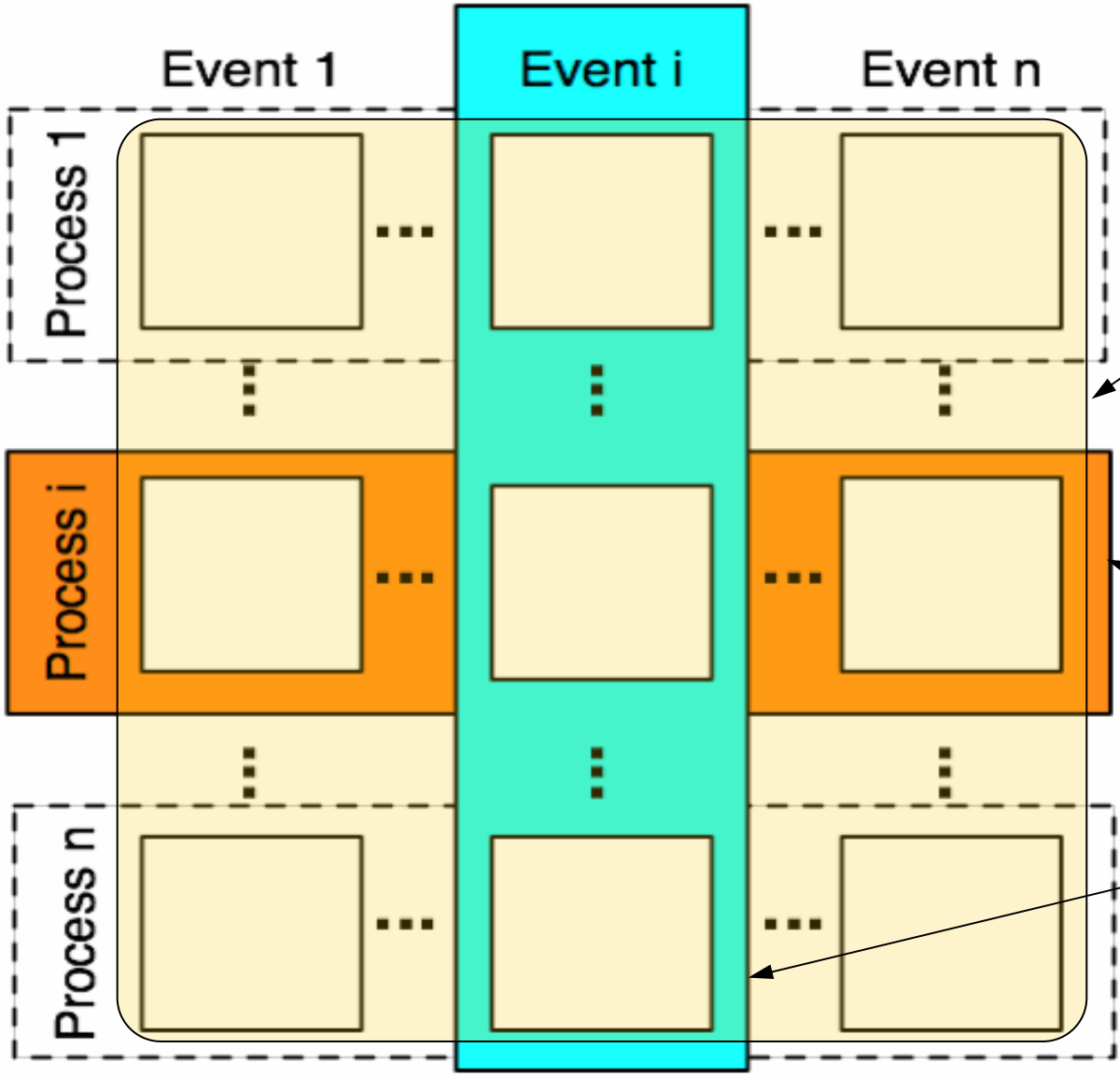
TAUg (TAU global)

- **TAU global performance space**
  - TAU aggregates local parallel performance measurements
  - Uses application communication infrastructure (MPI)
- **Application-level access to global performance data**
  - TAU API lets application create performance views
  - Application abstractions as basis for performance view
  - TAU aggregates performance data for a view (internal)
- **TAUg infrastructure focuses on:**
  - Portability
  - Scalability
  - Application-level support

TAUg – System Design



Views and Communicators



Global Performance Space

Global performance communicator

- **subset of MPI processes**

Global performance view

- **subset of global profile data**

TAUg Programming Interface

- **TAUg uses both TAU and MPI.**
- **TAUg method interfaces**
  - Designed in MPI style
  - Callable from C, C++, and Fortran.
- **View and communicator definition**

  void static TAU_REGISTER_VIEW()

  void static TAU_REGISTER_COMMUNICATOR()
  - Called after MPI_Init()
- **Get global performance view data**

  void static TAU_GET_VIEW()

View and Communicator Registration

```
void static TAU_REGISTER_VIEW(
  char* tau_event_name,
  int*  new_view_ID
);

void static TAU_REGISTER_COMMUNICATOR(
  int[] member_processes,
  int   number_of_processes,
  int*  new_communicator_id
);
```

Registration – Sample Code

```
int viewID = 0, commID = 0, numprocs = 0;

/* register a view for the calc method */
TAU_REGISTER_VIEW("calc()", &viewID);

MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
int members[numprocs];
for (int i = 0 ; i < numprocs ; i++)
    members[i] = i;

/* register a communicator for all processes */
TAU_REGISTER_COMMUNICATOR(
    members, numprocs, &commID);
```

Registration – Implementation Details

- **Registering a communicator**
  - Creates a new MPI communicator
  - Uses process ranks from MPI_COMM_WORLD
  - Used by TAUg only
- **TAUg maintains a map**
  - MPI_COMM_WORLD → new communicator rank
- **Returns a TAUg communicator ID**
  - Not MPI_Comm object reference
  - Ensures TAUg can support other communication libraries

Getting Global Performance Data View

```
void static TAU_GET_VIEW(
  int      viewID,
  int      communicatorID,
  int      type,
  int      sink,
  double** data,
    int*     outSize
);
```

- **Choices for type:**
  - TAU_ALL_TO_ONE (requires sink process)
  - TAU_ONE_TO_ALL (requires source process)
  - TAU_ALL_TO_ALL (sink parameter ignored)

Getting View – Sample Code

```
double *loopTime;
int size = 0, myid = 0, sink = 0;
MPI_Comm_rank(MPI_COMM_WORLD,&myid);

TAU_GET_VIEW(viewID, commID,
    TAU_ALL_TO_ONE, sink, &loopTime, &size);
if (myid == sink) {
    /* do something with the result... */
}
```

Getting View – Implementation Details

- **TAUg gathers local performance data**
  - Calls TAU_GET_FUNC_VALS()
  - Stops all performance measurements for process
  - Updates profile to current values
- **Package view data as custom MPI_Datatype**
- **Send to other processors according to type**
  - TAU_ALL_TO_ONE (uses MPI_Gather())
  - TAU_ONE_TO_ALL (uses MPI_Scatter())
  - TAU_ALL_TO_ALL (uses MPI_Allgather())

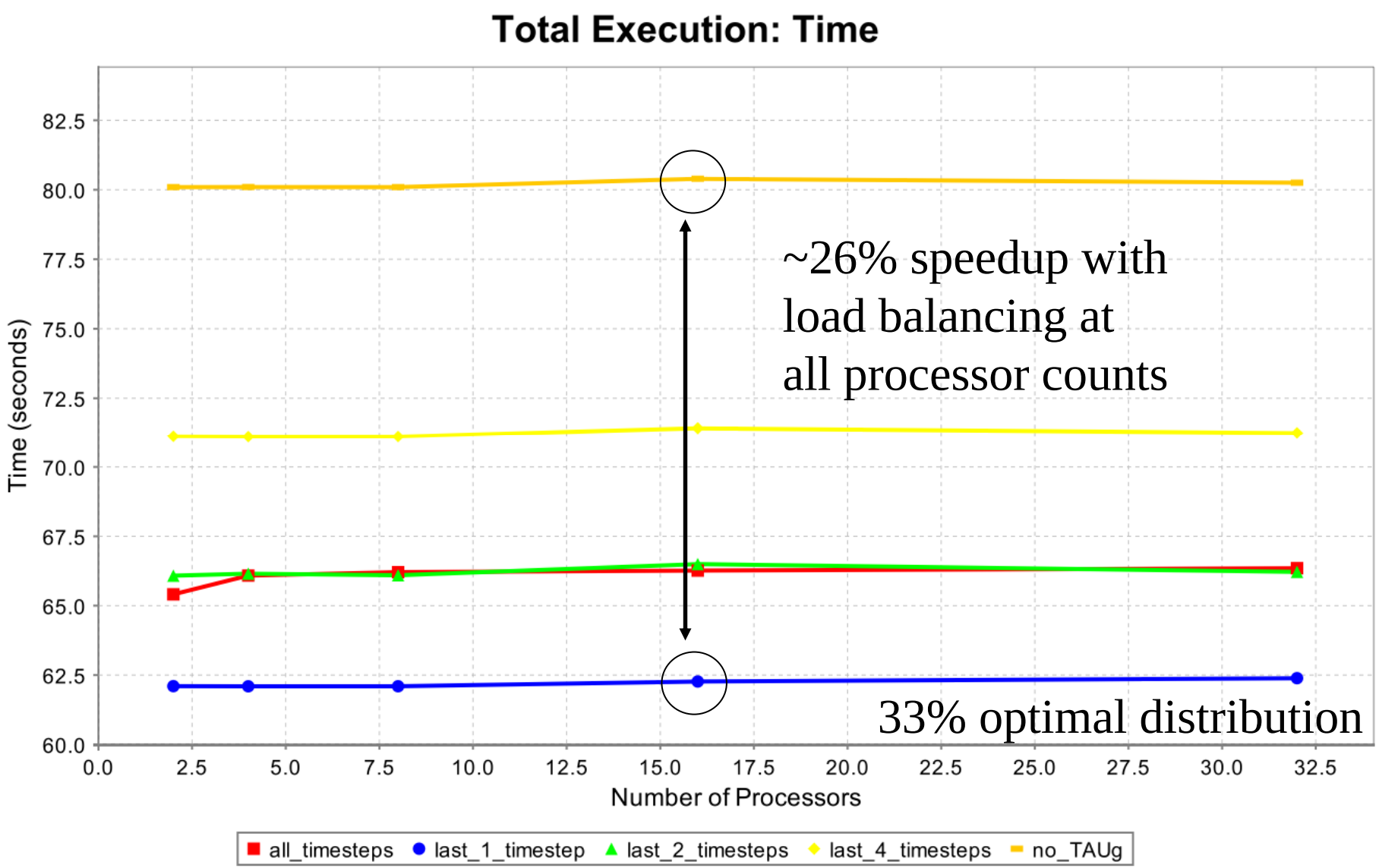
Experiment 1: Load Balancing Application

- **Simulate a heterogeneous cluster of *n* processors**
  - *n*/2 are 2x as fast as other *n*/2 nodes
  - Situation where external factors affect performance
- **Application wants to load balance work**
  1. Load is initially divided evenly
  2. After each timestep

     - application requests a TAUg performance view

     - redistributes load based on global performance state
- **Only one TAUg performance view**
  - Application-specific event
  - Execution time for work

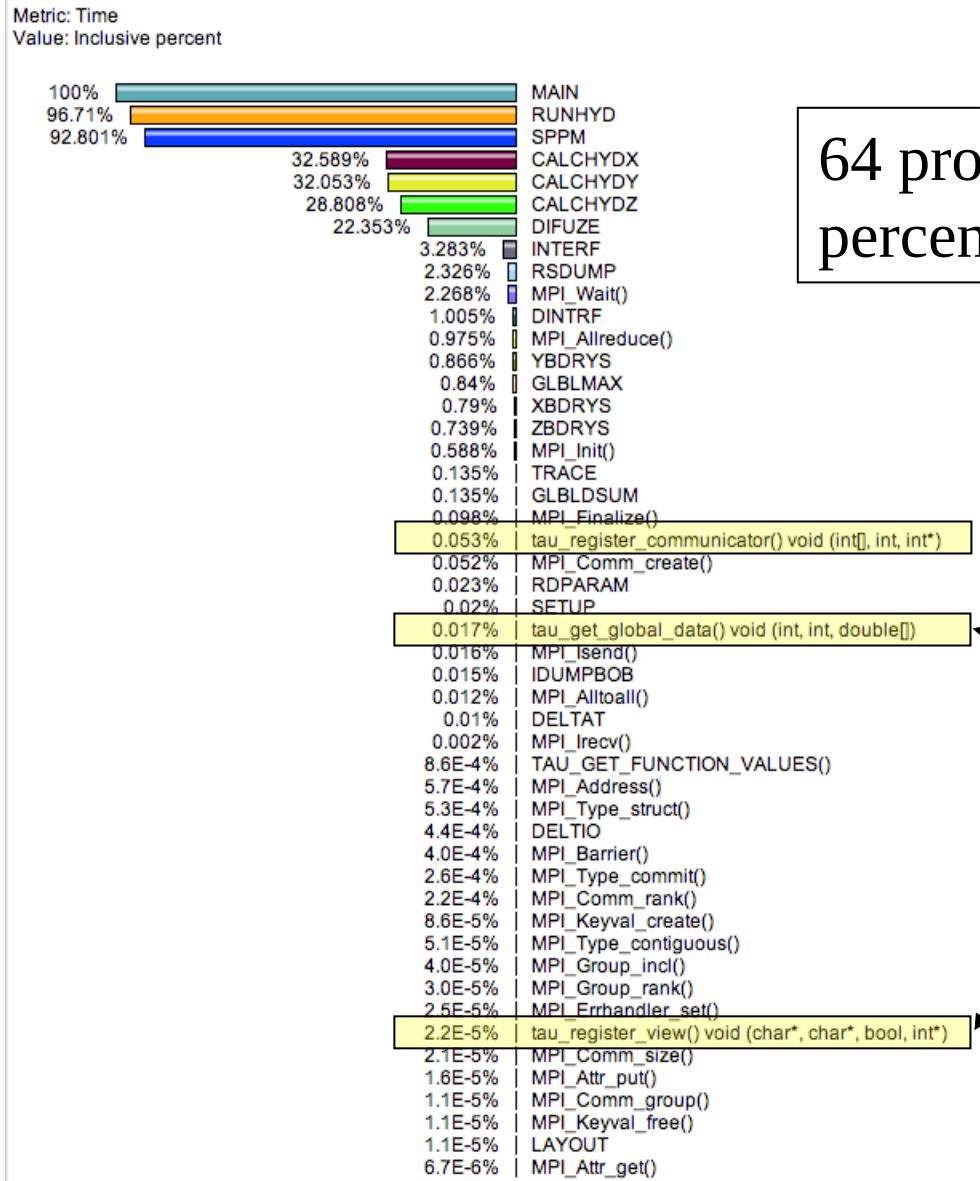
Load Balancing Simulation Results



Experiment 2: Overhead in Weak Scaling Study

- **sPPM benchmark (3D gas dynamics)**
  - 22 global performance views monitored
  - One for each application function (not including MPI)
  - 8 equal-sized communicator groups per experiment
    - 1, 2, 4 and 8 processes per group (4 experiments)
  - Get performance data for all view at each iteration
  - Run on MCR @ LLNL - 1152 node dual P4 2.4 Ghz
- **Weak scaling study up to 64 processors**
- **Investigate TAUg overhead**
  - Never exceeds 0.1% of the total runtime
    - 0.026 seconds of total 111 seconds with 8 processors
    - 0.081 seconds of total 115 seconds with 64 processors

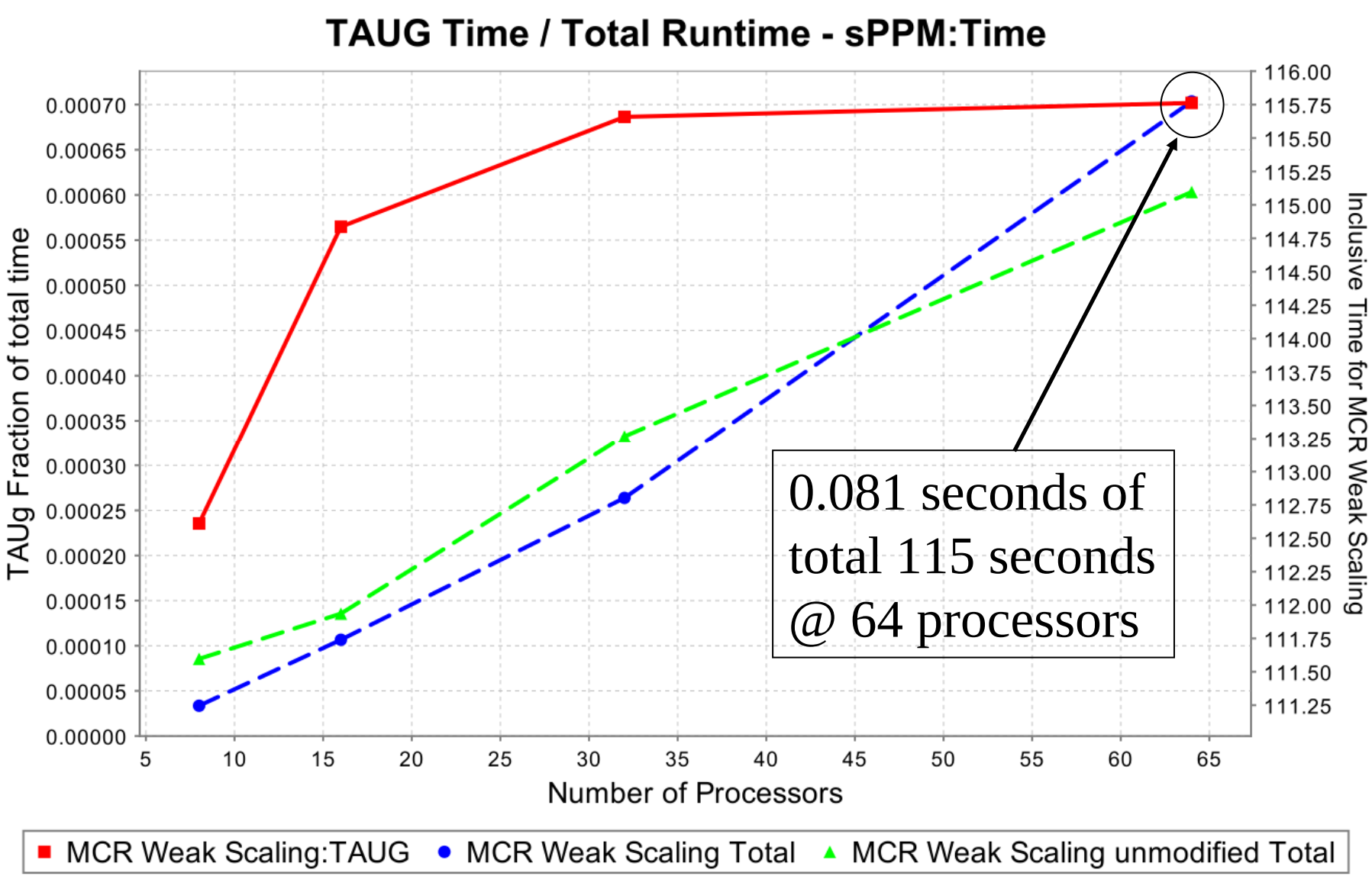
TAUg Overhead in sPPM Weak Scaling Study



64 processors
percent inclusive times

TAUg events are
not major contributors
to total runtime

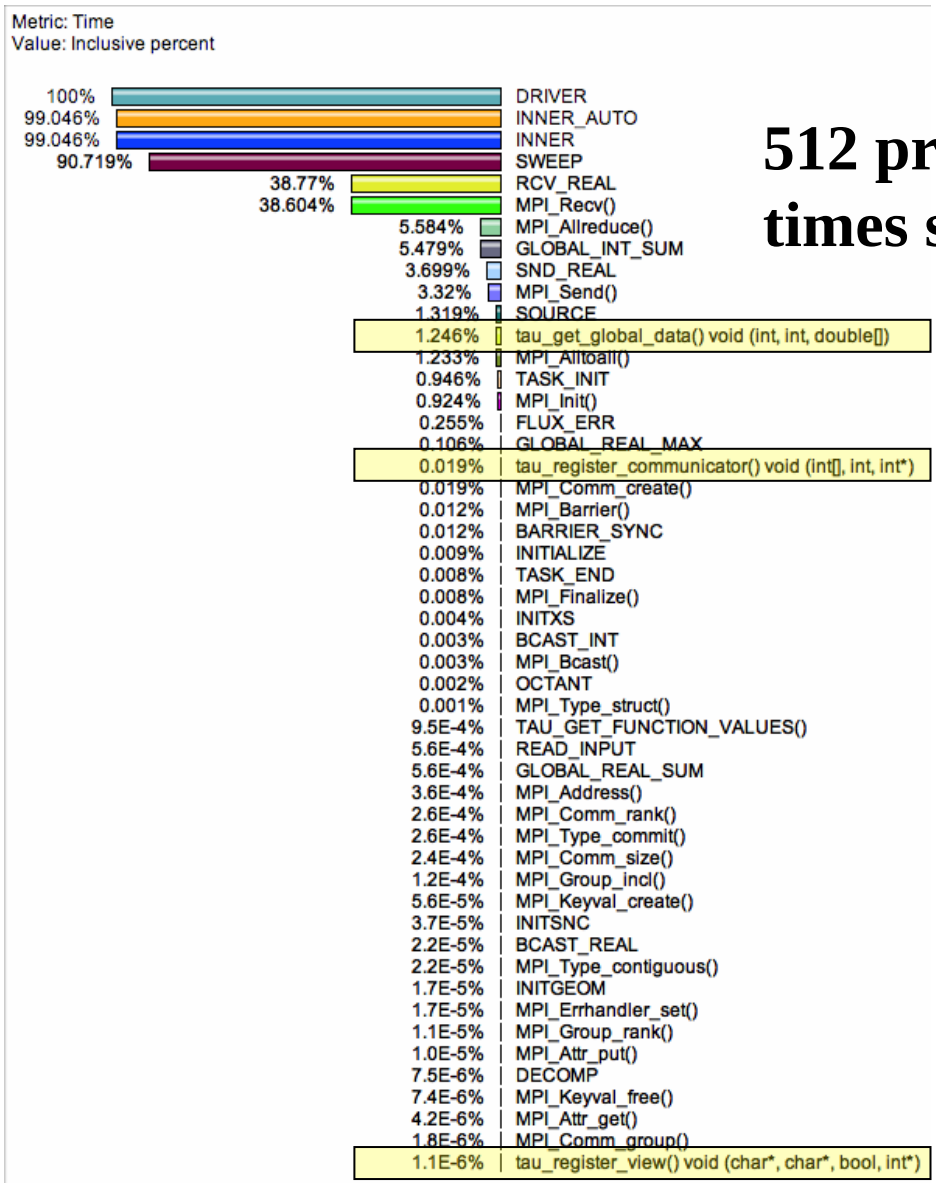
TAUg Overhead in sPPM



Experiment 3: Overhead in Strong Scaling Study

- **ASCI Sweep3D benchmark (3D neutron transport)**
  - One global performance view exchanged
    - one event: “SWEEP” (the main calculation routine)
  - One communicator consisting of all processes
    - all processes exchange performance data every timestep
  - Run on ALC @ LLNL - 960 node dual P4 2.4 Ghz
- **Strong scaling up to 512 processors**
- **Investigate TAUg overhead**
  - Never exceeds 1.3% of the total runtime
    - 0.094 seconds of total 77 minutes with 16 processors
    - 2.05 seconds of total 162 seconds with 512 processors

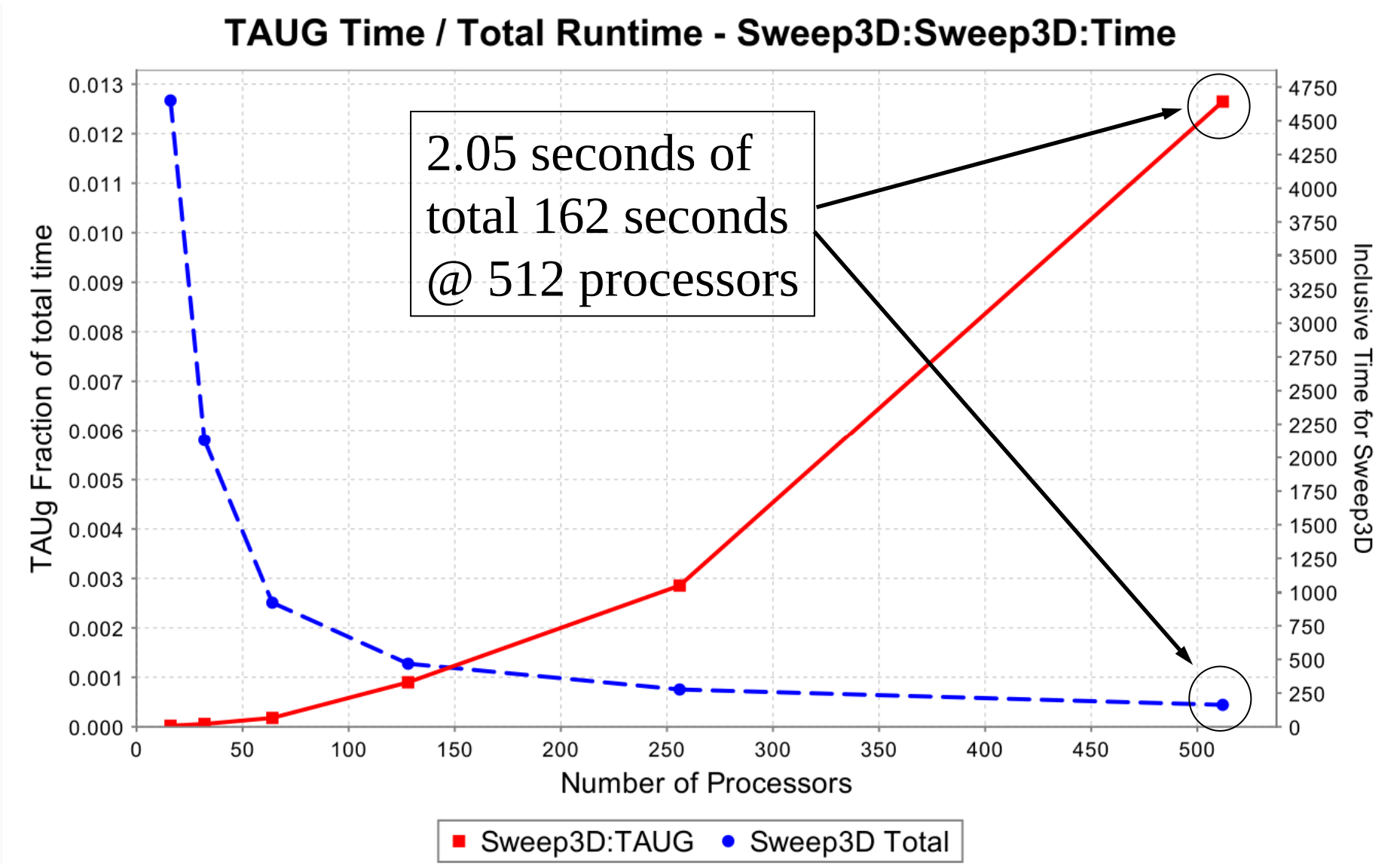
TAUg Overhead in Sweep3D



512 processors, strong scaling example, times shown are inclusive percentage

TAUg events are not major contributors to total runtime, even when exchanging with 512 processors each timestep

TAUg Overhead in Sweep3D



Discussion

- **TAUg currently limits global view access**
  - Single event and single metric (exclusive)
  - Simplified implementation for view data type
  - Need flexible view data specification
  - Multi-view access
- **Limited view communication types**
  - Only three communication methods
    - collective with barrier
  - Can imagine more sophisticated patterns
    - send and receive with pairwise and non-blocking
- **No performance data processing**

Future Work

- **Provide full access to TAU performance data**
- **Provide better view specific and communication**
- **Develop helper functions**
  - Basic statistics: average, minimum, maximum, …
  - Relative differentials: between timesteps and processes
- **Examine one-sided communication in MPI-2**
  - Support non-synchronous exchange of performance data
- **Integrate in real applications**

Conclusion

- **An application may desire to query its runtime performance state to make decisions that direct how the computation will proceed**
- **TAUg designed as an abstraction layer for MPI-based applications to retrieve performance views from the global performance space**
- **TAUg insulates the application developer from complexity involved in exchanging the global performance space**
- **TAUg is as portable as MPI and TAU, and is as scalable as the local MPI implementation.**
- **Overhead is inevitable, but minimized and under the application's control**

Support Acknowledgements


- **Department of Energy (DOE)**
  - Office of Science contracts
  - University of Utah ASCI Level 1
  - ASC/NNSA Level 3 contract
    - Lawrence Livermore National Laboratory
- **Department of Defense (DoD)**
  - HPC Modernization Office (HPCMO)
  - Programming Environment and Training (PET)
- **NSF Software and Tools for High-End Computing**
- **Research Centre Juelich**
- **Los Alamos National Laboratory**
- **www.cs.uoregon.edu/research/paracomp/tau**

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