



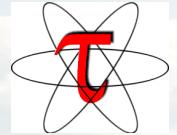
Geant4 Instrumentation-Based Measurements with TAU

Boyana Norris

University of Oregon

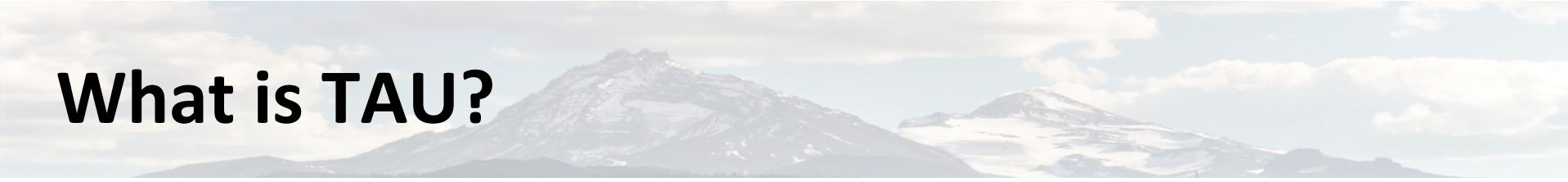
(based on materials by Sameer Shende, ParaTools)

TAU Performance System®



- Tuning and Analysis Utilities (18+ year project)
- Comprehensive performance profiling and tracing
 - Integrated, scalable, flexible, portable
 - Targets all parallel programming/execution paradigms
- Integrated performance toolkit
 - Instrumentation, measurement, analysis, visualization
 - Widely-ported performance profiling / tracing system
 - Performance data management and data mining
 - Open source (BSD-style license)
- Easy to integrate in application frameworks

<http://tau.uoregon.edu>



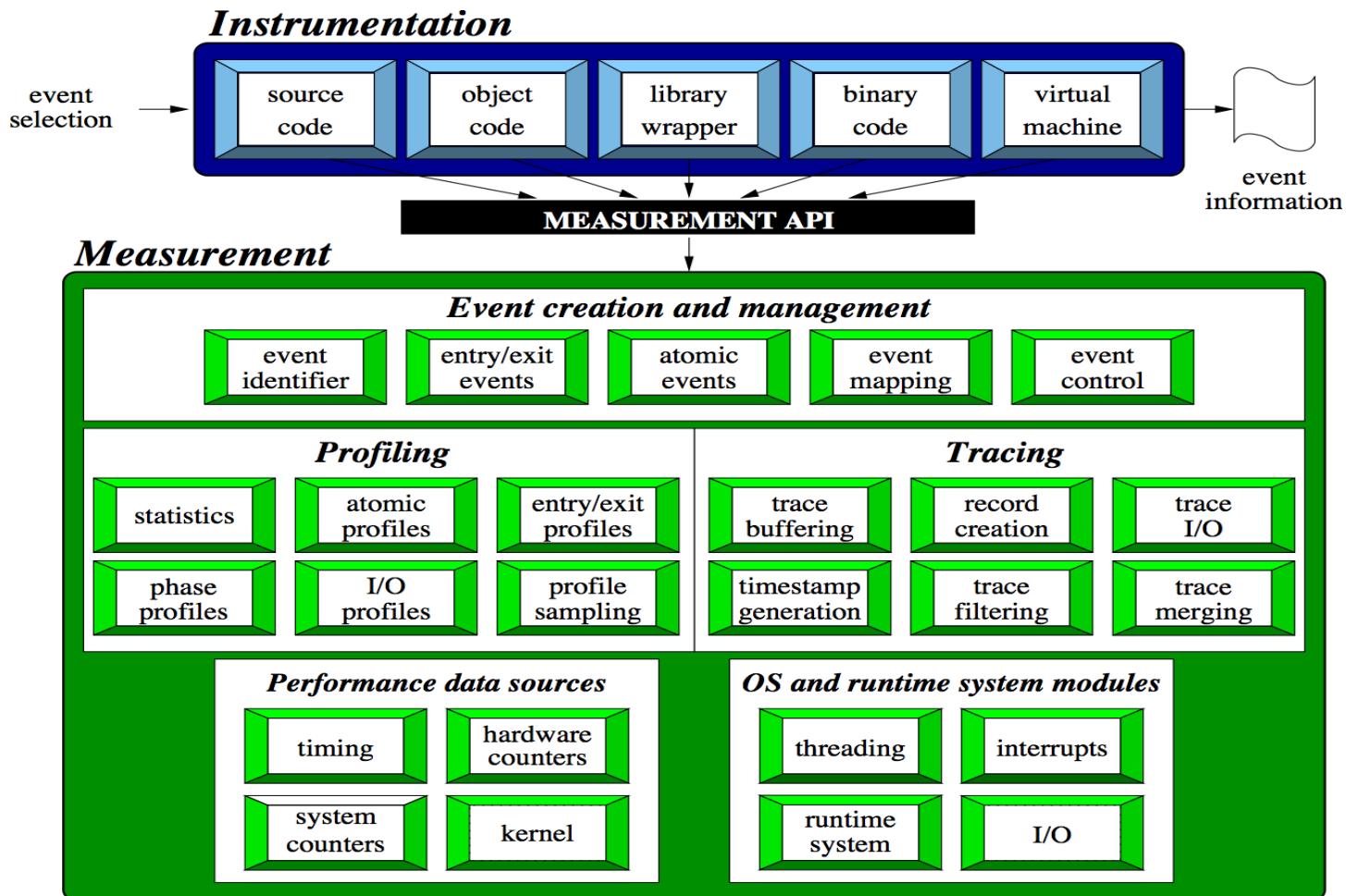
What is TAU?

- TAU is a performance evaluation tool
- It supports parallel profiling and tracing
- **Profiling:** how much (total) time was spent in each routine
- **Tracing:** when events take place in each process along a timeline
- Profiling and tracing can measure **time** as well as **hardware performance counters** (cache misses, instructions) from your CPU
- TAU can **automatically instrument** your source code using a package called PDT for routines, loops, I/O, memory, phases, etc.
- TAU runs on most HPC platforms and is free (BSD style license)
- TAU has instrumentation, measurement and analysis tools
 - paraprof is TAU's 2D and 3D profile browser
- TAU's automatic instrumentation works with most build systems

Understanding Application Performance

- ❑ How much time is spent in each application routine and outer loops? Within loops, what is the contribution of each statement?
- ❑ How many instructions are executed in these code regions? Floating point, Level 1 and 2 data cache misses, hits, branches taken?
- ❑ What is the peak heap memory usage of the code? When and where is memory allocated/de-allocated? Are there any memory leaks?
- ❑ How much time does the application spend performing I/O? What is the peak read and write *bandwidth* of individual calls, total volume?
- ❑ What is the contribution of different *phases* of the program?

TAU Instrumentation / Measurement



What does TAU support?

C/C++

Fortran

pthreads

Intel GNU

MinGW

Insert
yours
here

CUDA

OpenACC

Intel MIC

LLVM

Linux

BlueGene

NVIDIA Kepler

UPC

PGI

Windows

Fujitsu

OS X

OpenCL

GPI

Java

Python

MPI

OpenMP

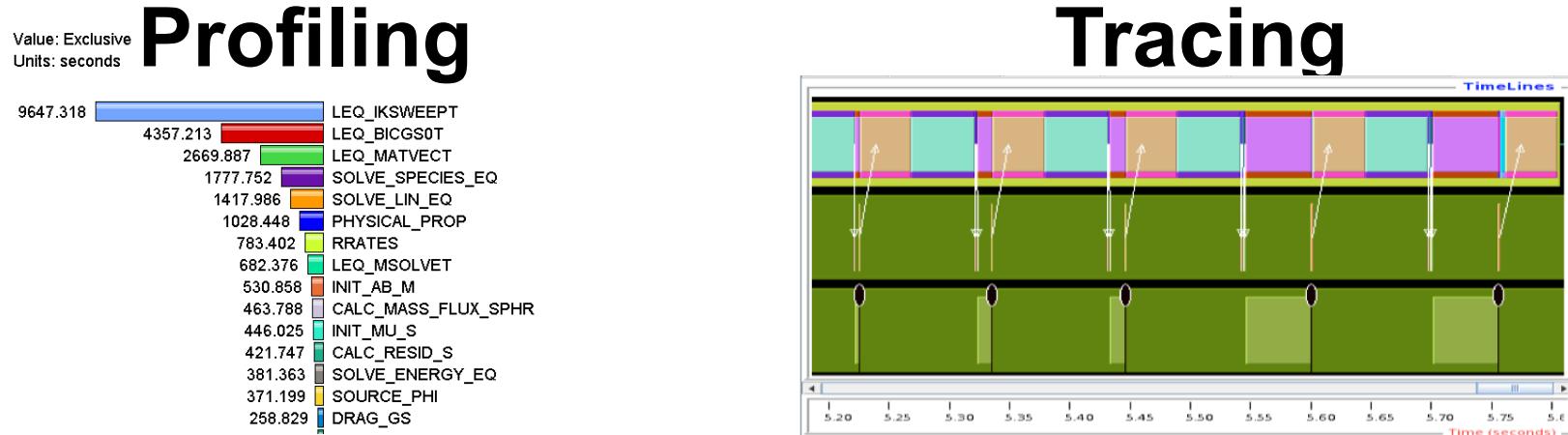
Cray

Sun

AIX

ARM

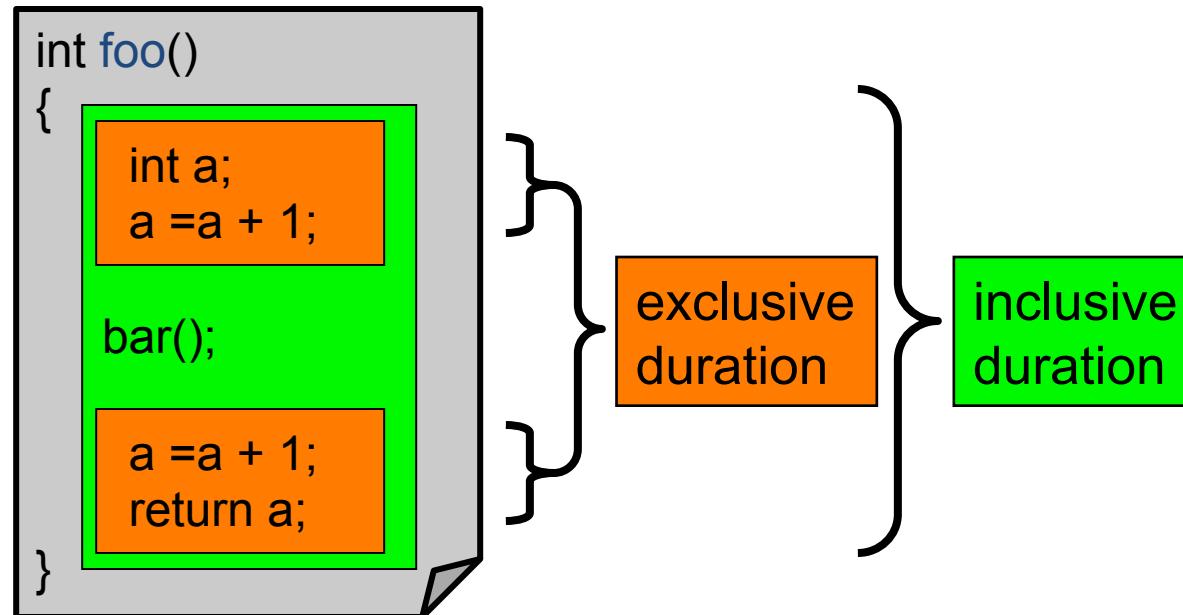
Profiling and Tracing



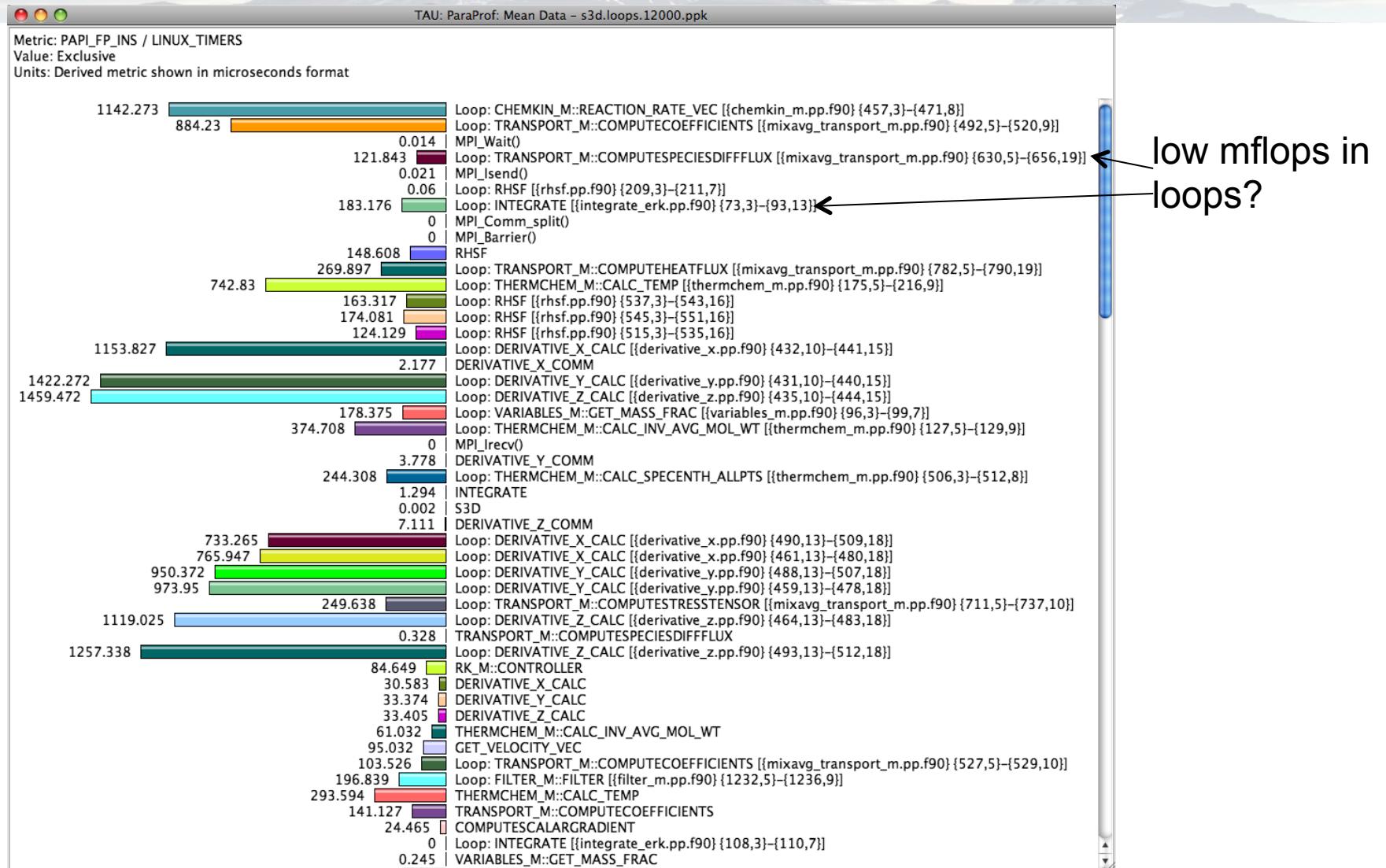
- Profiling** shows how much (total) time was spent in each routine
- Tracing** shows when the events take place on a timeline
- Metrics can be time or hardware performance counters (cache misses, instructions)
- TAU can automatically instrument your source code using a package called PDT for routines, loops, I/O, memory, phases, etc.

Inclusive vs. Exclusive Measurements

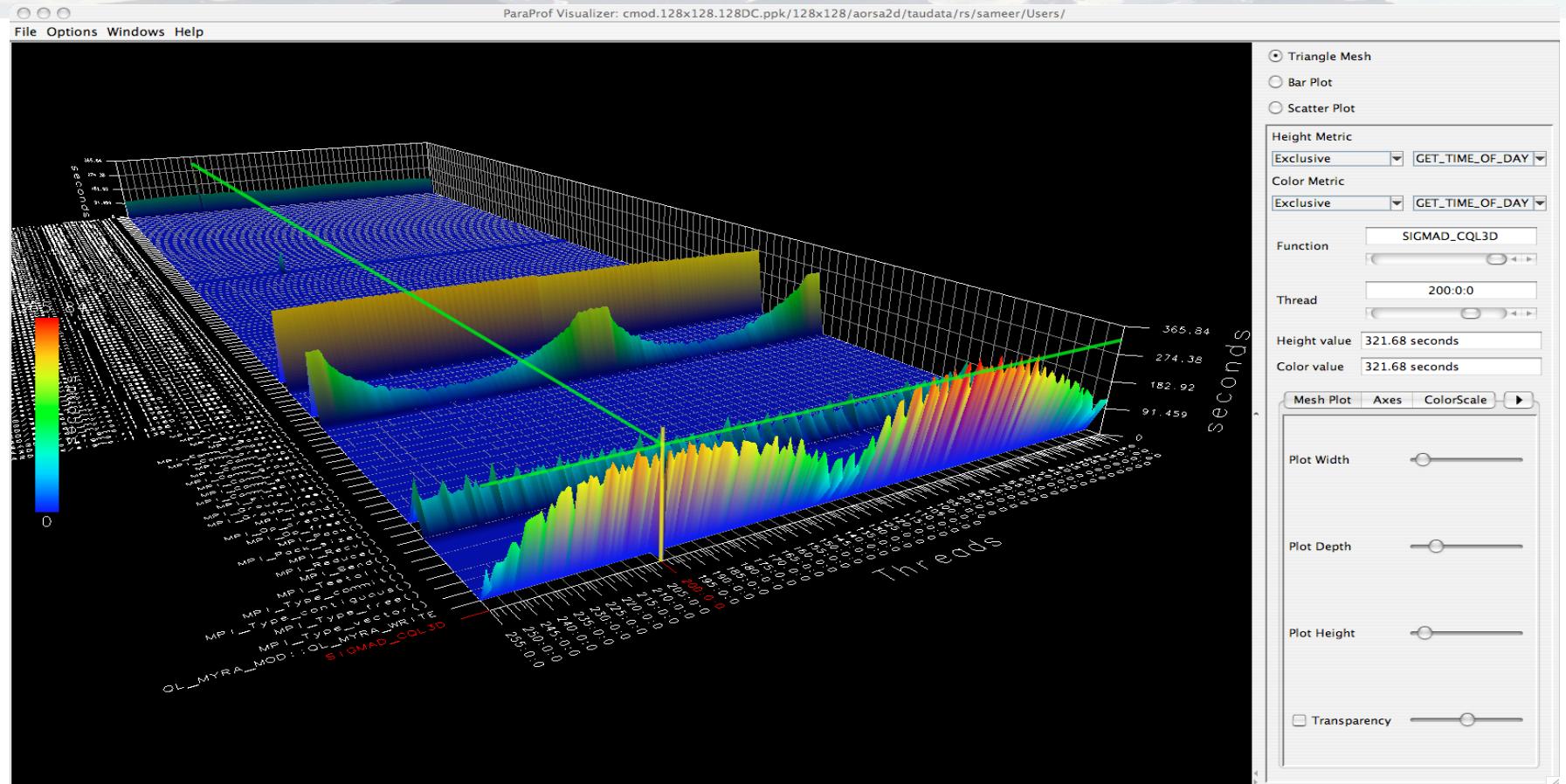
- ❑ Performance with respect to code regions
- ❑ Exclusive measurements for region only
- ❑ Inclusive measurements includes child regions



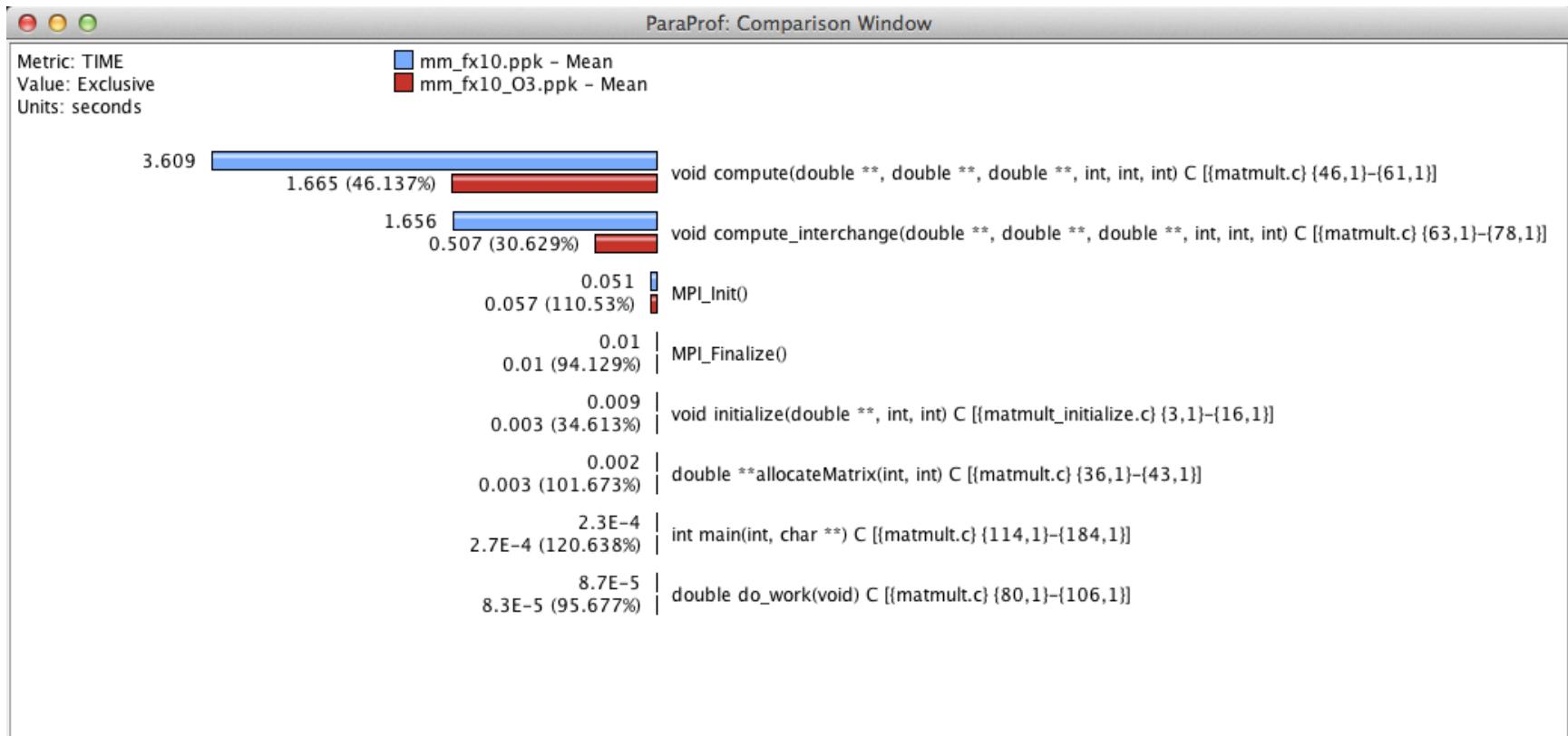
Identifying Potential Bottlenecks



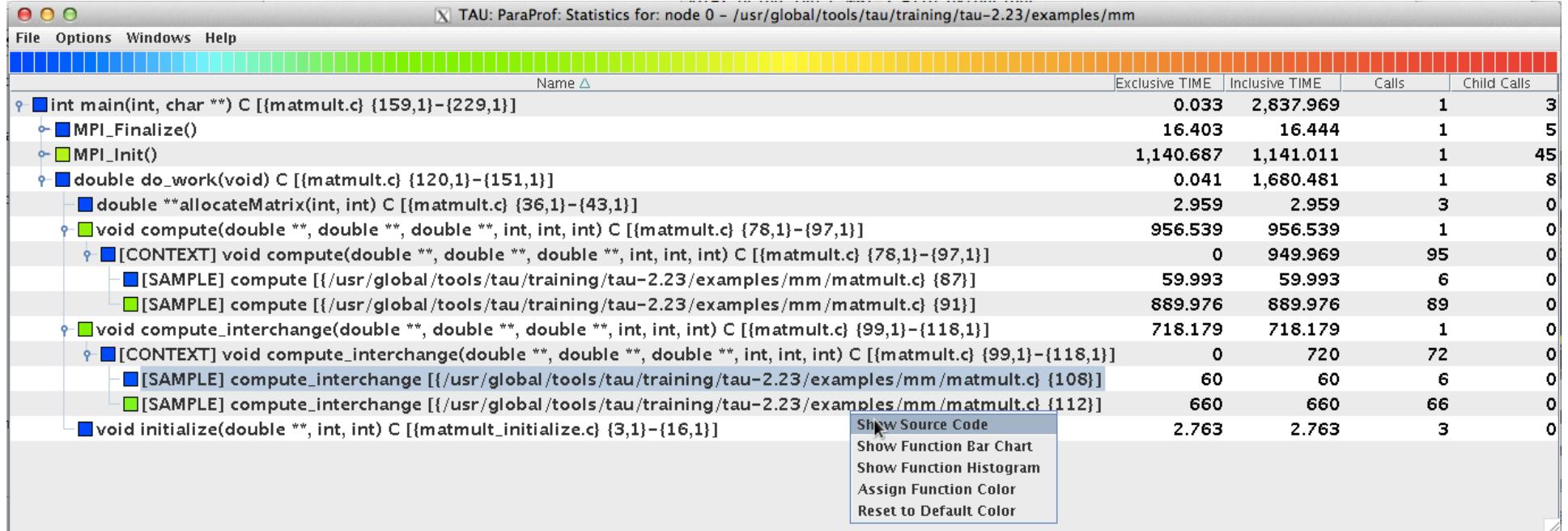
ParaProf 3D Profile Browser



TAU's ParaProf Comparison Window



Event-Based Sampling in TAU (not the focus of this talk)



```
% export TAU_MAKEFILE=$TAU/Makefile.tau-icpc-papi-mpi-pdt
% make CC=tau_cc.sh CXX=tau_cxx.sh
% export TAU_SAMPLING=1
% mpirun -np 256 ./a.out
% paraprof
```

TAU Instrumentation Approach

- Supports both direct and indirect performance observation
 - Direct instrumentation of program (system) code (probes)
 - Instrumentation invokes performance measurement
 - Event measurement: performance data, meta-data, context
 - Indirect mode supports sampling based on periodic timer or hardware performance counter overflow based interrupts
- Support for user-defined events
 - **Interval** (Start/Stop) events to measure exclusive & inclusive duration
 - **Atomic events** (Trigger at a single point with data, e.g., heap memory)
 - ◆ Measures total, samples, min/max/mean/std. deviation statistics
 - **Context events** (are atomic events with executing context)
 - ◆ Measures above statistics for a given calling path3

Direct Instrumentation Options in TAU

- ❑ Source Code Instrumentation
 - Automatic instrumentation using pre-processor based on static analysis of source code (PDT), creating an instrumented copy
 - Compiler generates instrumented object code
 - Manual instrumentation
- ❑ Library-level instrumentation
 - Statically or dynamically linked wrapper libraries: MPI, I/O, memory, ...
 - Wrapping external libraries where source is not available
- ❑ Runtime pre-loading and interception of library calls
- ❑ Binary Code instrumentation
 - Rewrite the binary, runtime instrumentation
- ❑ Virtual Machine, Interpreter, OS level instrumentation

Direct Observation: Events

- Event types
 - Interval events (begin/end events)
 - ◆ Measures exclusive & inclusive durations between events
 - ◆ Metrics monotonically increase
 - Atomic events (trigger with data value)
 - ◆ Used to capture performance data state
 - ◆ Shows extent of variation of triggered values (min/max/mean)

- Code events
 - Routines, classes, templates
 - Statement-level blocks, loops

Initial Experiences with Geant4

- Performed automated instrumentation of Geant4 v. 10.0
 - Complete: all functions
 - Selective: only explicitly specified functions
- Collected two types of profiles
 - Flat: per-function information without any context
 - Callpath: per-function information for each calling context
- Types of data collected
 - Wall-clock time
 - CPU and GPU hardware performance counters, e.g., instruction counts, cache misses
- Performance database hosted at UO
 - Host: taudb.nic.uoregon.edu, DB: geant4

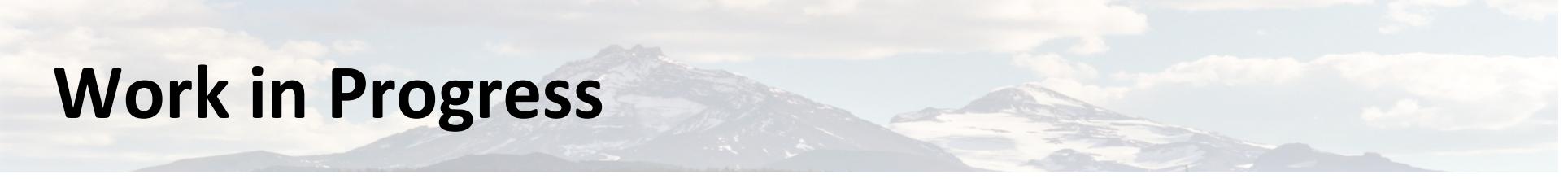
Steps in Using TAU to Profile Geant4

- Compile Geant4 with instrumentation:
 - For full instrumentation, simply configure with:

```
cmake -DCMAKE_CXX_COMPILER=tau_cxx.sh \
       -DCMAKE_CC_COMPILER=tau_cc.sh ...
```

- For selective instrumentation, create a **select.tau** file, then define the **TAU_OPTIONS** environment variable accordingly, for example:

```
export TAU_OPTIONS=' -optVerbose -optRevert \
                   -optTauSelectFile="$HOME/Geant4/select.tau" '
```



Work in Progress

- Automate *selective instrumentation* and performance experiments process
 - Integration with Fast – use it as input to decide what to instrument, generate select.tau
- Integrate PerfExplorer analysis scripts for
 - More detailed analysis of individual experiments
 - Comparison across Geant4 source versions
 - More efficient analysis of subsystems, e.g., EM physics or portions of the class hierarchy (no performance tools support this well)