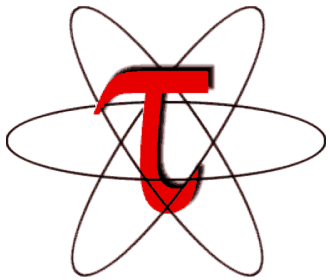
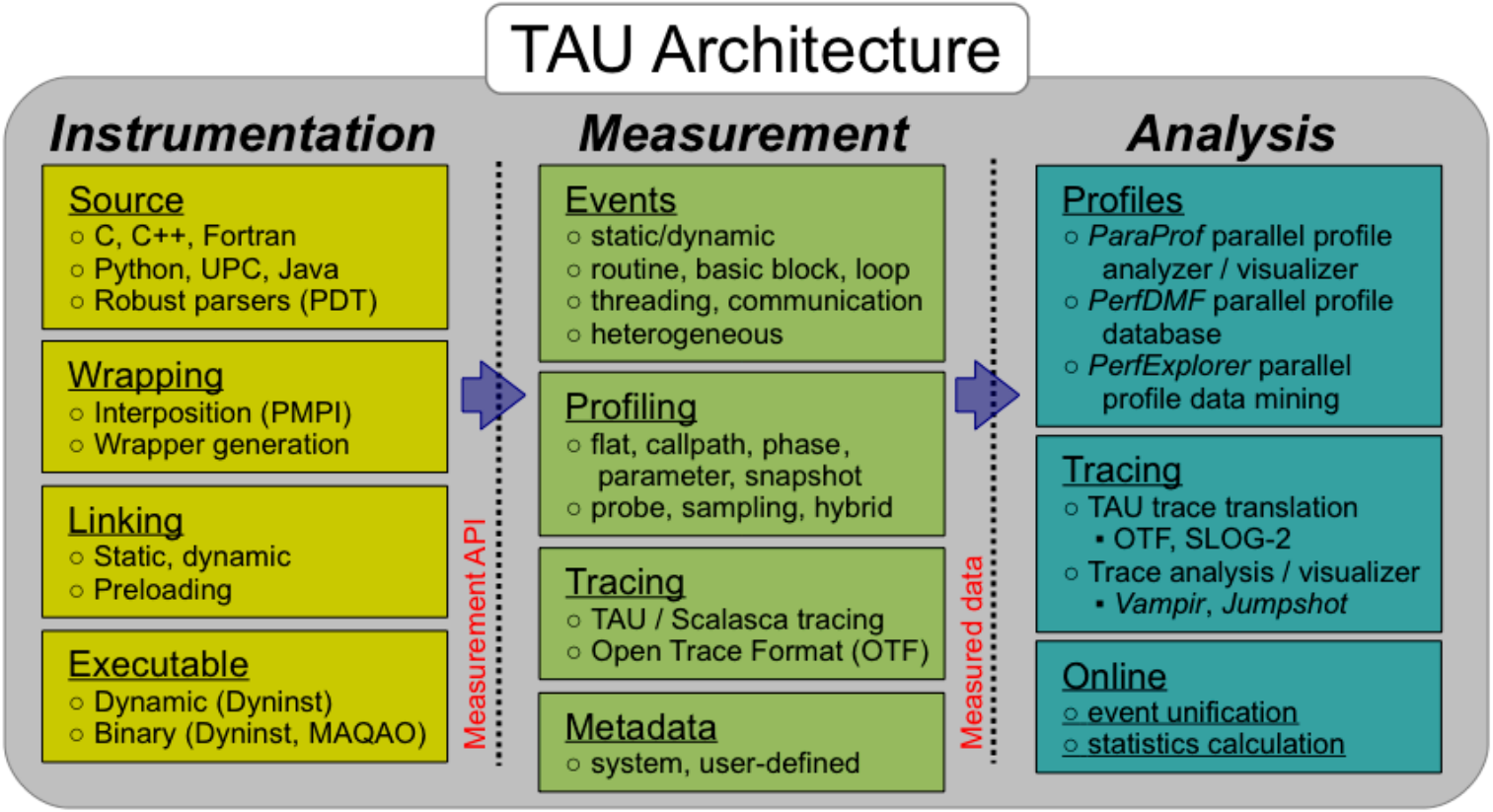
VI-HPS

Parallel Profile Analysis with TAU

Prof. Allen D. Malony
University of Oregon



- Parallel performance framework and toolkit
  - Supports all HPC platforms, compilers, runtime system
  - Provides portable instrumentation, measurement, analysis



TAU Observation Methodology and Workflow

- TAU's (primary) methodology for parallel performance observation is based on the insertion of measurement *probes* into application, library, and runtime system
  - Code is instrumented to make visible certain events
  - Performance measurements occur when events are triggered
  - Known as probe-based (direct) measurement
- Performance experimentation workflow
  - Instrument application and other code components
  - Link / load TAU measurement library
  - Execute program to gather performance data
  - Analysis performance data with respect to events
  - Analyze multiple performance experiments
- Extend methodology with sampling-based techniques

Understanding Application Performance

- How much time is spent in each routine and outer loops?
  - Within loops, what is the contribution of each statement?
- How many instructions are executed in code regions?
  - Floating point, Level 1/2 data cache misses, hits, branches, …
- What is the memory usage of the code?
  - When and where is memory allocated/de-allocated?
  - Are there any memory leaks?
- What are the I/O characteristics of the code?
  - What is the peak read/write BW of individual calls, total volume?
- What is the contribution of each phase of the program
  - What is the time wasted/spent waiting for collectives, and I/O operations in initialization, computation, I/O phases?
- How does the application scale?
  - What is the efficiency, runtime breakdown of performance across different core counts?

TAU Components

- Instrumentation
  - Fortran, C/C++, OpenMP, UPC, Java, Python, Chapel
  - Source, compiler, library wrapping, binary rewriting
  - Automatic instrumentation
- Measurement and analysis support
  - Internode: MPI, OpenSHMEM, ARMCI, PGAS, DMAPP
  - Intranode: pthreads, OpenMP, hybrid, other thread models
  - Heterogeneous: GPU, MIC, CUDA, OpenCL, OpenACC, …
  - Performance data (timing, counters) and metadata
  - Parallel profiling and tracing (with Score-P integration)
- Analysis
  - Parallel profile analysis and visualization (*ParaProf*)
  - Performance data mining / machine learning (*PerfExplorer*)
  - Performance database technology (*TAUdb*)
  - Empirical autotuning (integration with Active Harmony, CHiLL, Orio)

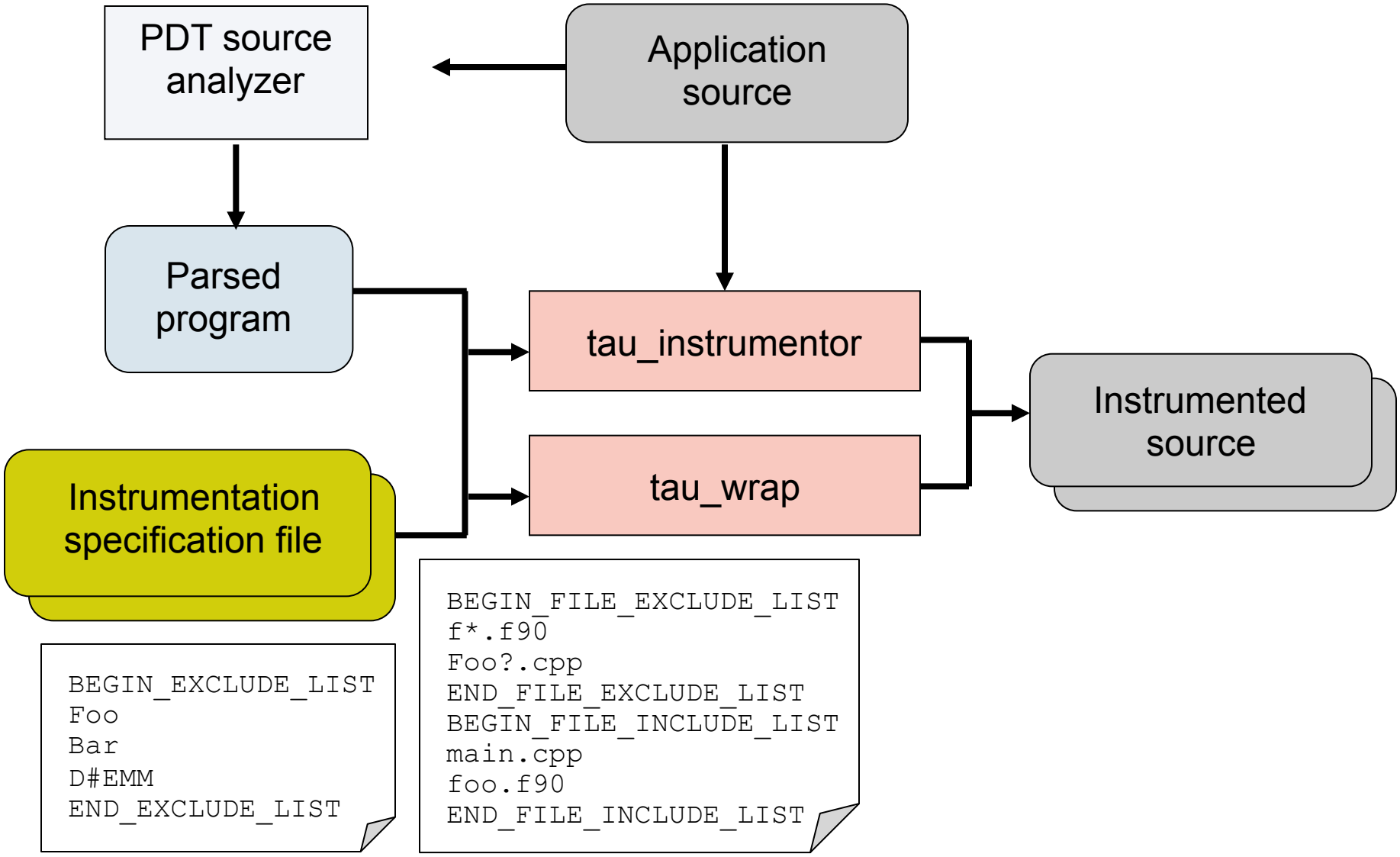
TAU Instrumentation Approach

- Direct and indirect performance instrumentation
  - *Direct* instrumentation of program (system) code (probes)
  - *Indirect* support via sampling or interrupts
- Support for standard program code events
  - Routines, classes and templates
  - Statement-level blocks, loops
  - *Interval events* (start/stop)
- Support for user-defined events
  - Interval events specified by user
  - *Atomic events* (statistical measurement at a single point)
  - *Context events* (atomic events with calling path context)
- Provides static events and dynamic events

TAU Instrumentation Mechanisms

- Source code
  - Manual (TAU API, TAU component API)
  - Automatic (robust)
    - C, C++, F77/90/95, OpenMP (POMP/OPARI), UPC
  - Compiler (GNU, IBM, NAG, Intel, PGI, Pathscale, Cray, …)
- Object code (library-level)
  - Statically- and dynamically-linked wrapper libraries
    - MPI, I/O, memory, …
  - Powerful library wrapping of external libraries without source
- Executable code / runtime
  - Runtime preloading and interception of library calls (*tau_exec*)
  - Binary instrumentation (Dyninst, MAQAO, PEBIL)
  - OpenMP Tools Interface (OMPT, tau_exec –T ompt)
  - Sampling (TAU_SAMPLING=1, TAU_EBS_UNWIND=1)
  - CUDA CUPTI, OpenCL (tau_exec -T cupti -cupti)

Automatic Source-level / Wrapper Instrumentation



TAU Measurement Approach

- Portable and scalable parallel profiling solution
  - Multiple profiling types and options
  - Event selection and control (enabling/disabling, throttling)
  - Online profile access and sampling
  - Online performance profile overhead compensation
- Portable and scalable parallel tracing solution
  - Trace translation to OTF, EPILOG, Paraver, and SLOG2
  - Trace streams (OTF) and hierarchical trace merging
- Robust timing and hardware performance support
- Multiple counters (hardware, user-defined, system)
- Metadata (hardware/system, application, …)
- Measurement approach also integrates with Score-P

TAU Measurement Mechanisms

- Parallel profiling
  - Function-level, block-level, statement-level
  - Supports user-defined events and mapping events
  - Support for flat, callgraph/callpath, phase profiling
  - Support for parameter and context profiling
  - Support for tracking I/O and memory (library wrappers)
  - Parallel profile stored (dumped, shapshot) during execution
- Tracing
  - All profile-level events
  - Inter-process communication events
  - Inclusion of multiple counter data in traced events
- Utilizes Score-P for efficient callpath profiling and tracing

- Flat profiles
  - Metric (e.g., time) spent in an event (callgraph nodes)
  - Exclusive/inclusive, # of calls, child calls
- Callpath profiles (Calldepth profiles)
  - Time spent along a calling path (edges in callgraph)
  - “main=> f1 => f2 => MPI_Send” (event name)
  - TAU_CALLPATH_DEPTH environment variable
- Phase profiles
  - Flat profiles under a phase (nested phases are allowed)
  - Default “main” phase
  - Supports static or dynamic (per-iteration) phases
- Parameter and context profiling

Using TAU

- TAU supports several measurement and thread options
  - Each measurement configuration of TAU corresponds to a unique stub makefile (configuration file) and library that is generated when you configure it
- To instrument source code automatically using PDT
  - Choose an appropriate TAU stub makefile in <arch>/lib:
  - % export TAU_MAKEFILE=$TAU/Makefile.tau-icpc-mpi-pdt
  - % export TAU_OPTIONS= '-optVerbose …' (see tau_compiler.sh)
  - % export PATH=$TAU_ROOT/x86_64/bin:$PATH
  - % export TAU=$TAU_ROOT/x86_64/lib
  - Use *tau_f90.sh*, *tau_cxx.sh*, *tau_upc.sh*, or *tau_cc.sh* as compilers
  - % mpif90 foo.f90 ➡ % tau_f90.sh foo.f90
- Set environment variables, execute application, and analyze with *pprof* (text based profile display) or *paraprof* (GUI)

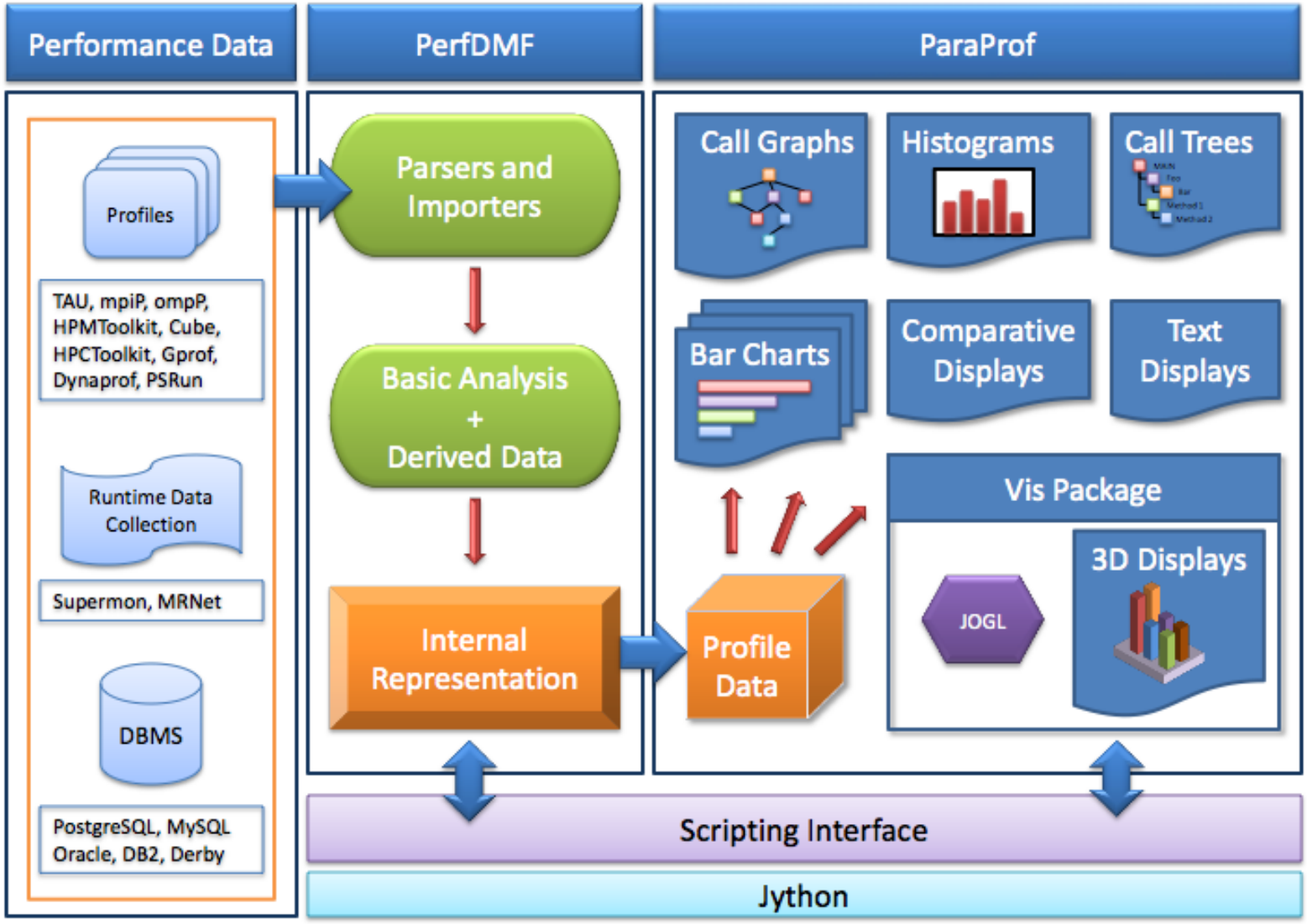
Using TAU with Score-P

```
% export TAU=$TAU_ROOT/x86_64/lib
% export TAU_MAKEFILE=
  $TAU/Makefile.tau-icpc-papi-mpi-pdt-openmp-opari-scorep
% export OMP_NUM_THREADS=10
% make CC=tau_cc.sh CXX=tau_cxx.sh F90=tau_f90.sh
% mpirun -np 4  ./matmult
% cd score*; paraprof profile.cubex &
```

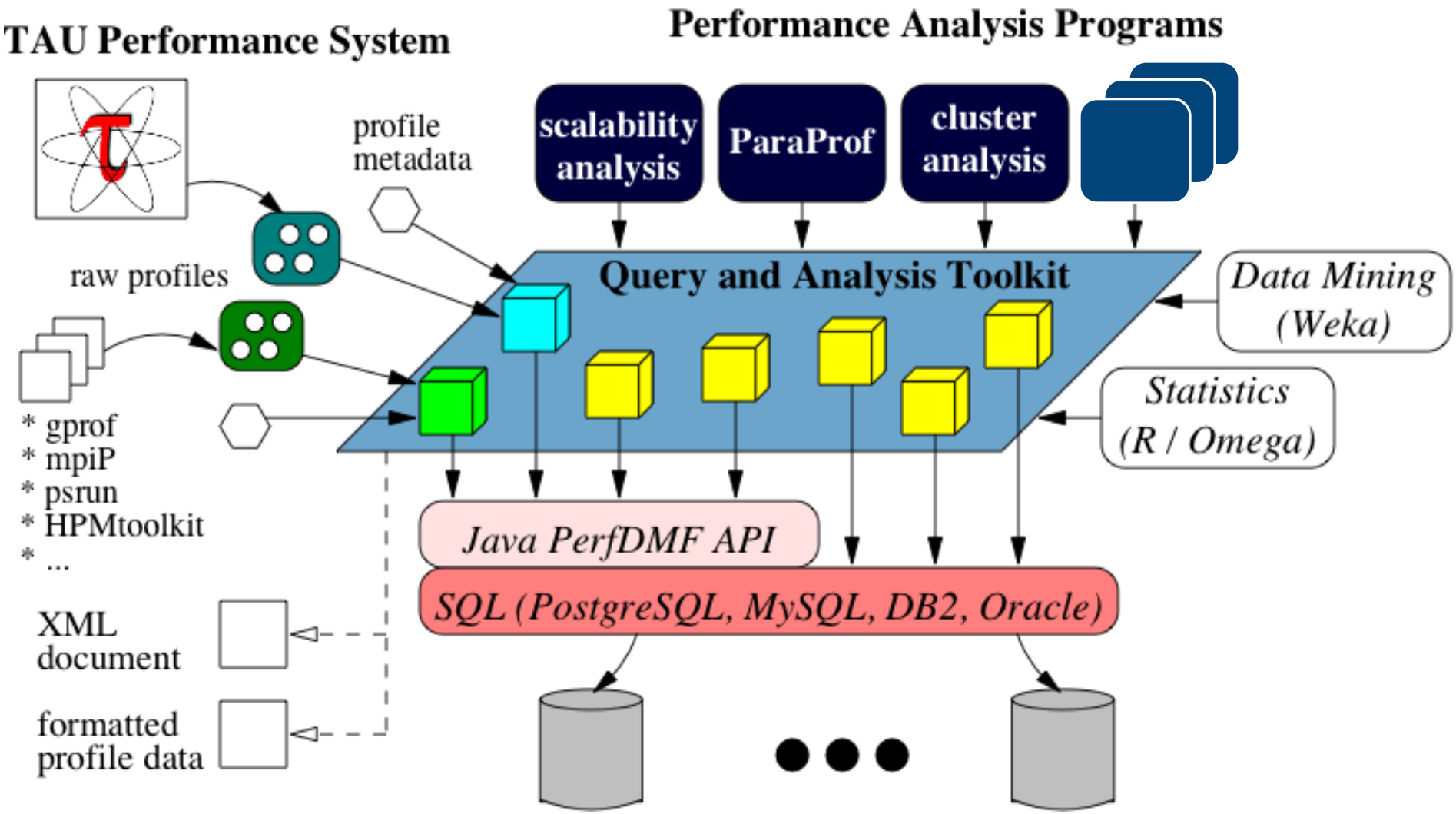
TAU Performance Analysis

- Analysis of parallel profile and trace measurement
- Parallel profile analysis (*ParaProf*)
  - Java-based analysis and visualization tool
  - Support for large-scale parallel profiles
- Performance data management (*TAUdb*)
- Performance data mining (PerfExplorer)
- Parallel trace analysis
  - Translation to VTF (V3.0), EPILOG, OTF formats
  - Integration with Vampir / Vampir Server (TU Dresden)
- Integration with CUBE browser (Scalasca, UTK / FZJ)
- Scalable runtime fault isolation with callstack debugging
- Efficient parallel runtime bounds checking

TAU Performance Analysis Framework

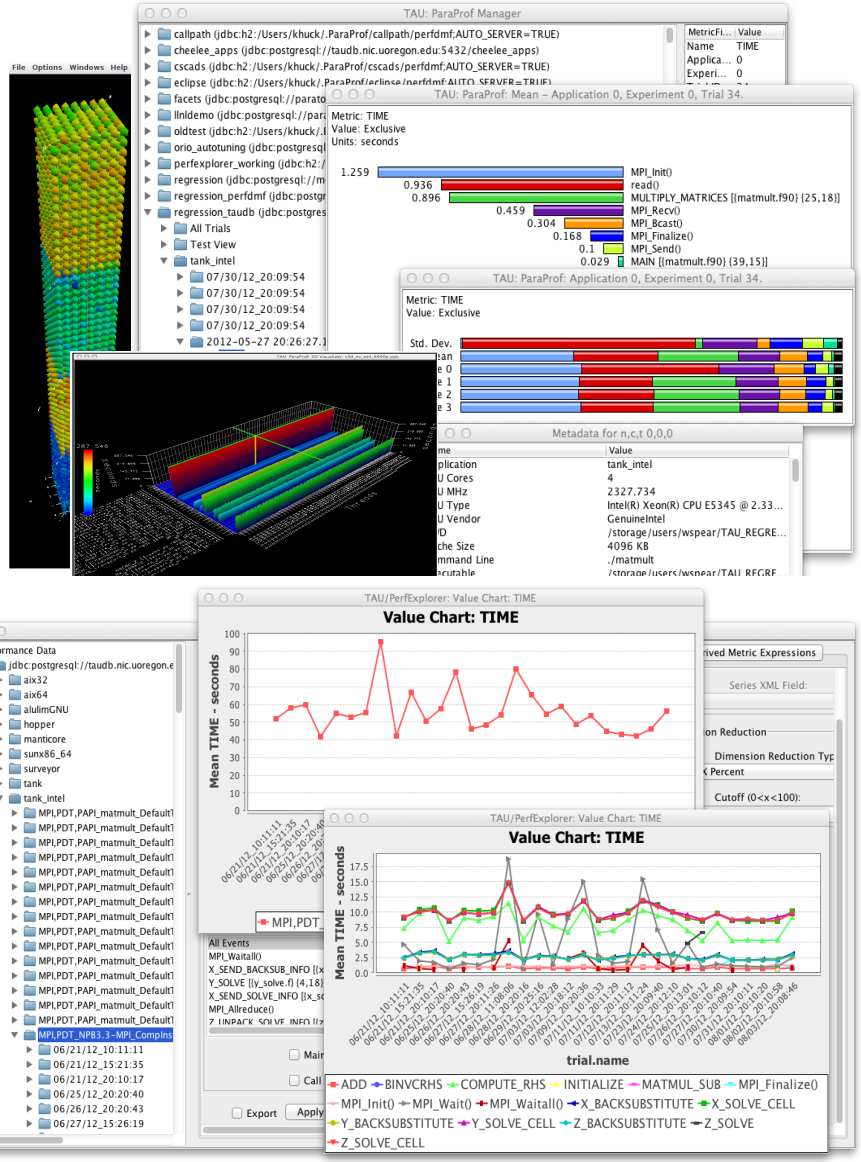


TAUdb: Performance Database Framework



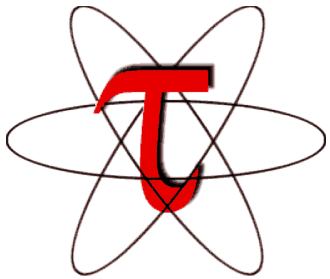
TAUdb: Performance Analysis Tool Support

- ParaProf
  - Parallel profile analyzer
    - visual pprof
  - 2, 3+D visualizations
  - Single and comparative experiment analysis
- PerfExplorer
  - Data mining framework
    - Clustering, correlation
  - Multi-experiment analysis
  - Scripting engine
  - Expert system



VI-HPS

Demo:
NPB-MZ-MPI / BT



Hands-on: Profile report exploration

- Tutorial contains Score-P experiments of BT-MZ
  - MPI + OpenMP
  - Class “B“, 4 processes with 4 OpenMP threads each
  - Collected on SuperMUC HPC system at LRZ, Munich, Germany

```
% wget http://tau.uoregon.edu/data.tgz
% tar zxf data.tgz
% cd data/scorep
```

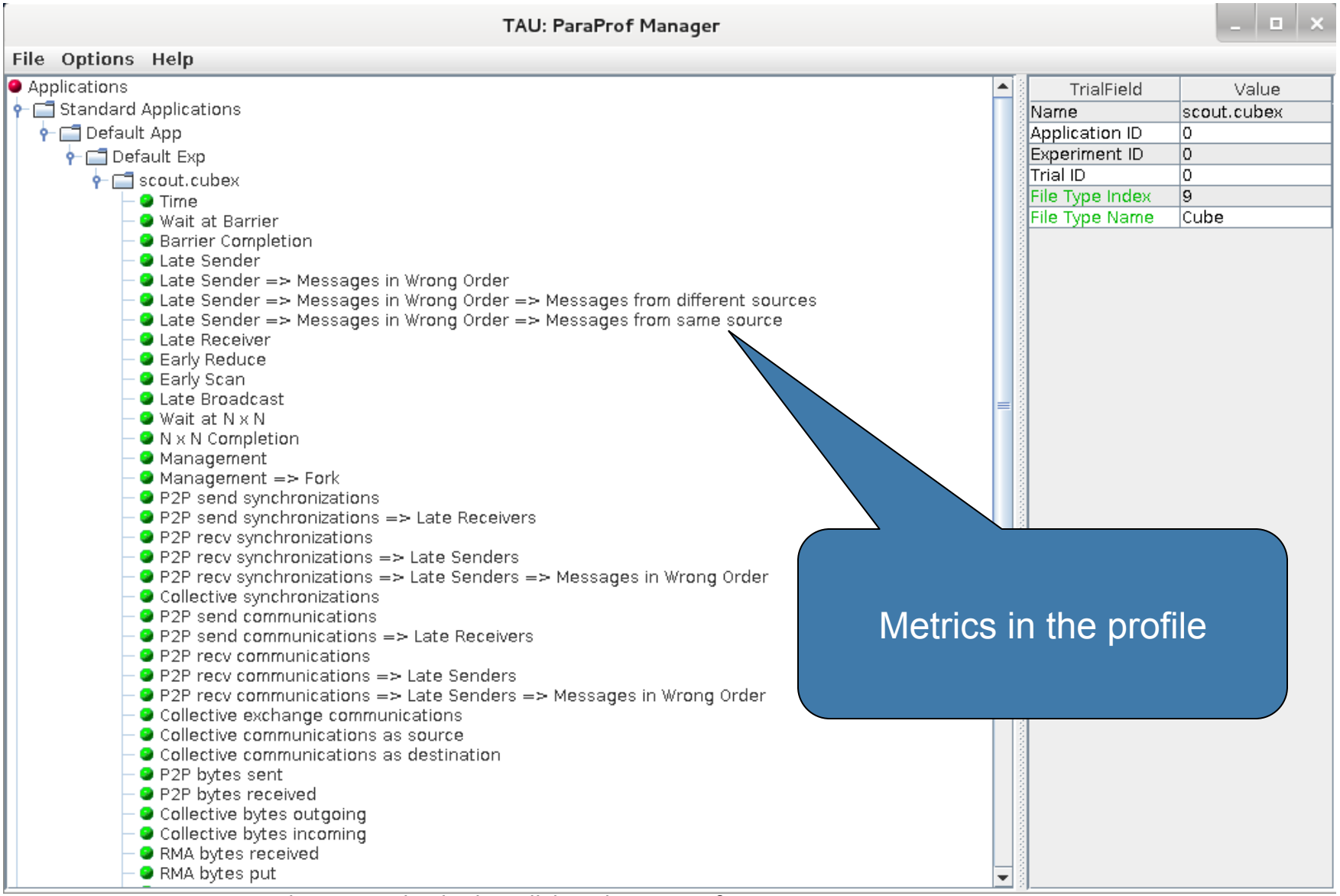
- Load trials into TAUdb

```
% taudb_loadtrial -a BT_MZ -x "Class_B" -n ”16p" -f cube 2p/profile.cubex
% taudb_loadtrial -a BT_MZ -x "Class_B" -n ”16p" -f cube 4p/profile.cubex
% taudb_loadtrial -a BT_MZ -x "Class_B" -n ”16p" -f cube 8p/profile.cubex
% taudb_loadtrial -a BT_MZ -x "Class_B" -n ”16p" -f cube 16p/profile.cubex
```

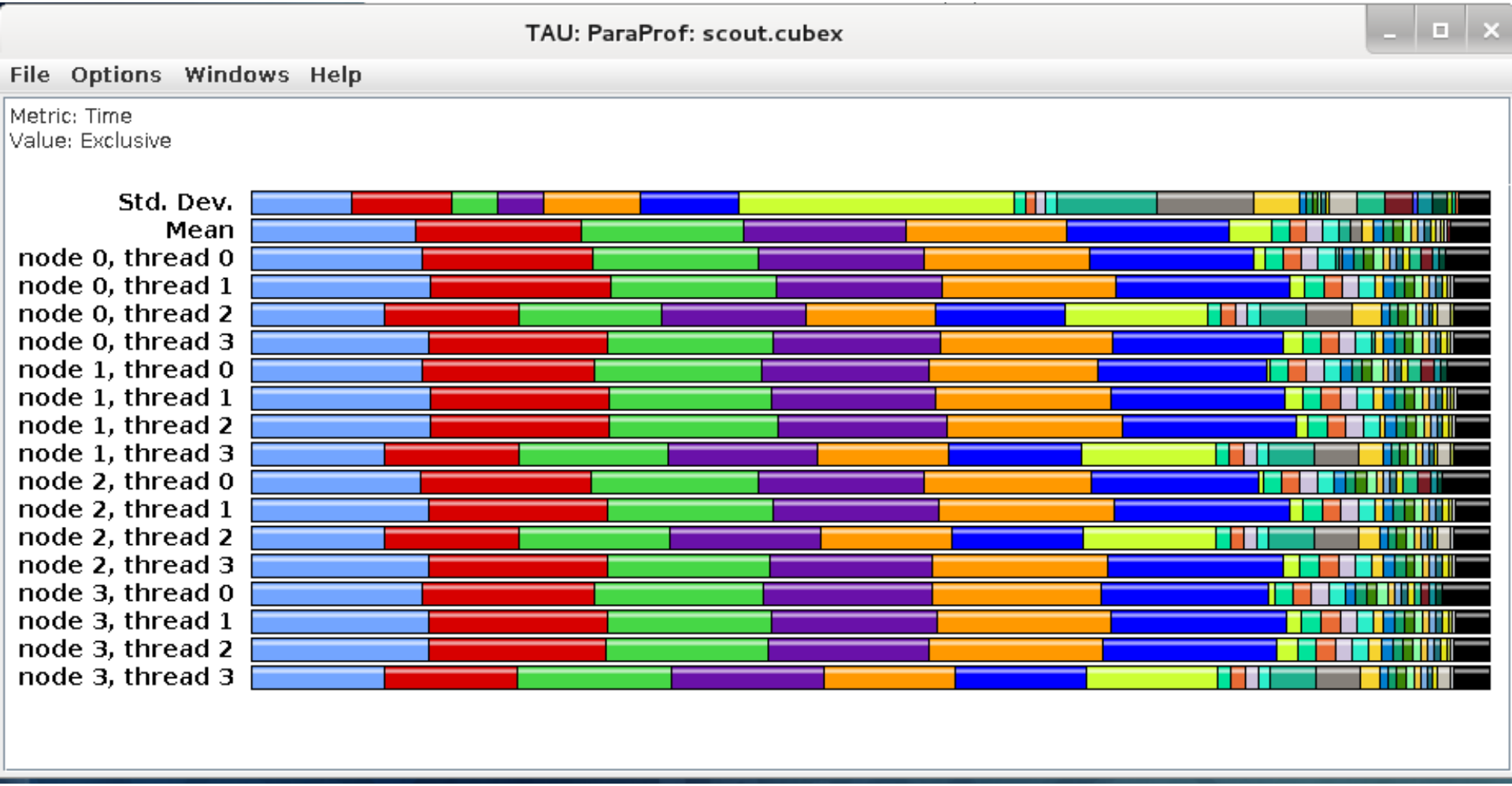
- Start TAU‘s paraprof and perfexplorer GUI

```
% paraprof &
% perfexplorer &
```

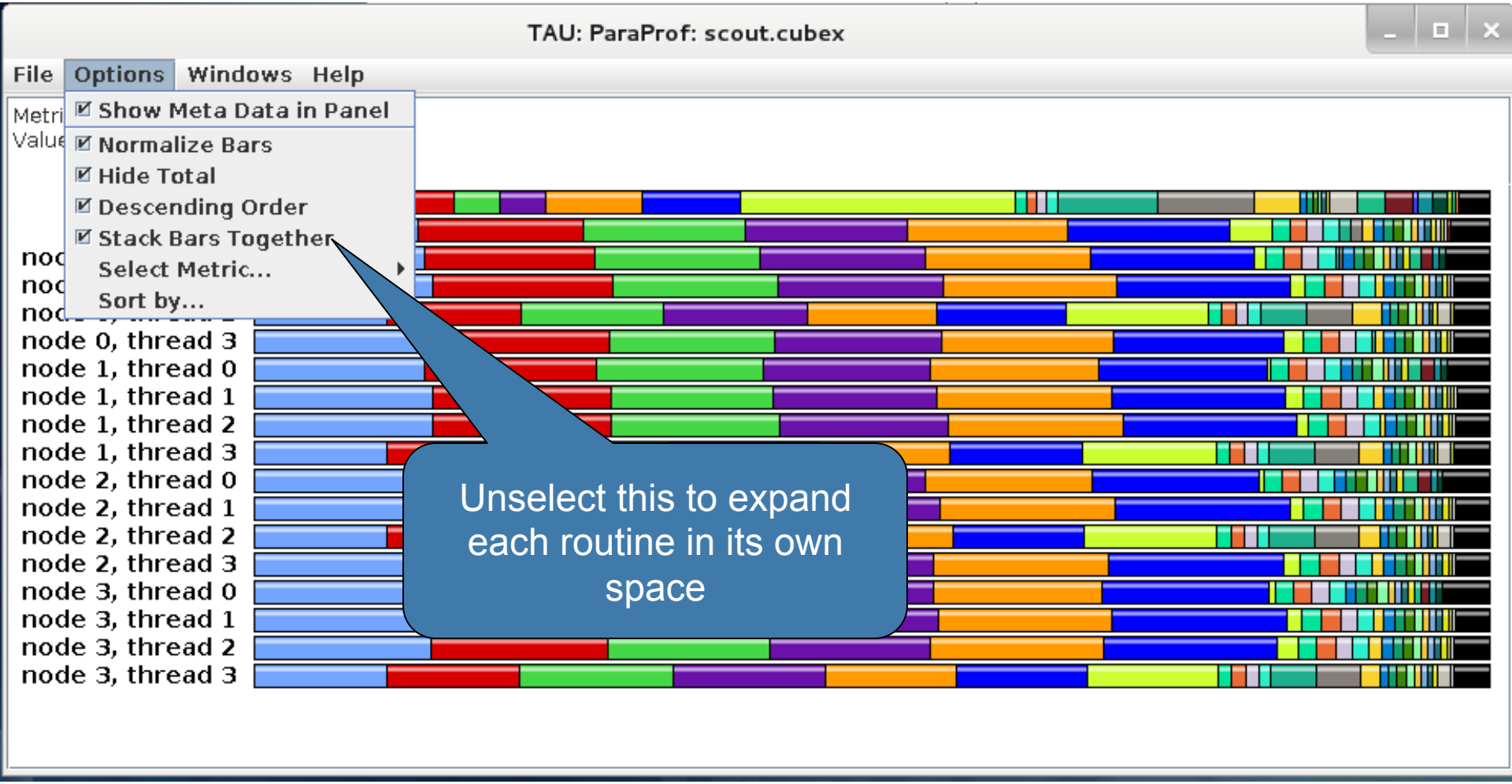
ParaProf: Manager Window: scout.cubex



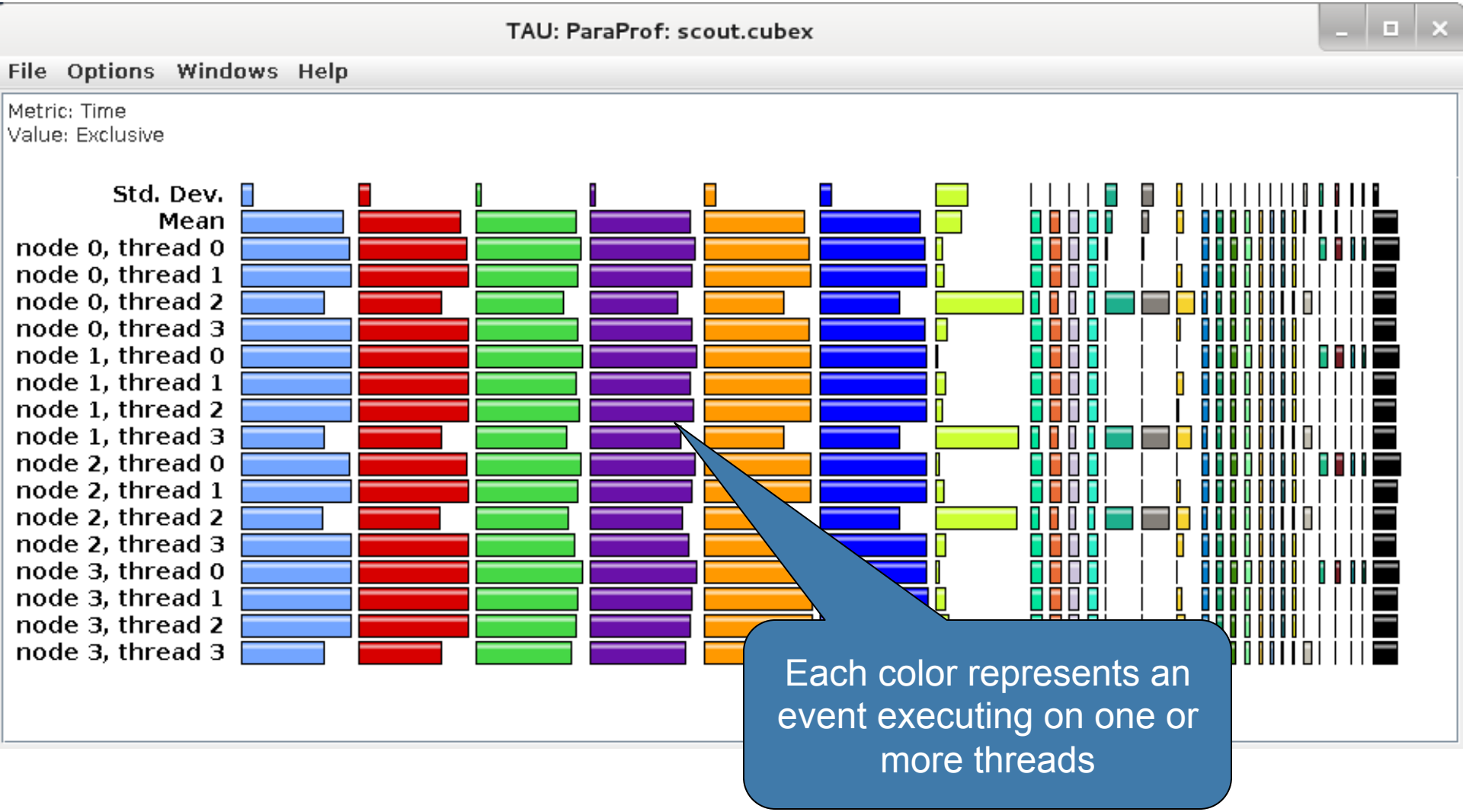
ParaProf: Main window



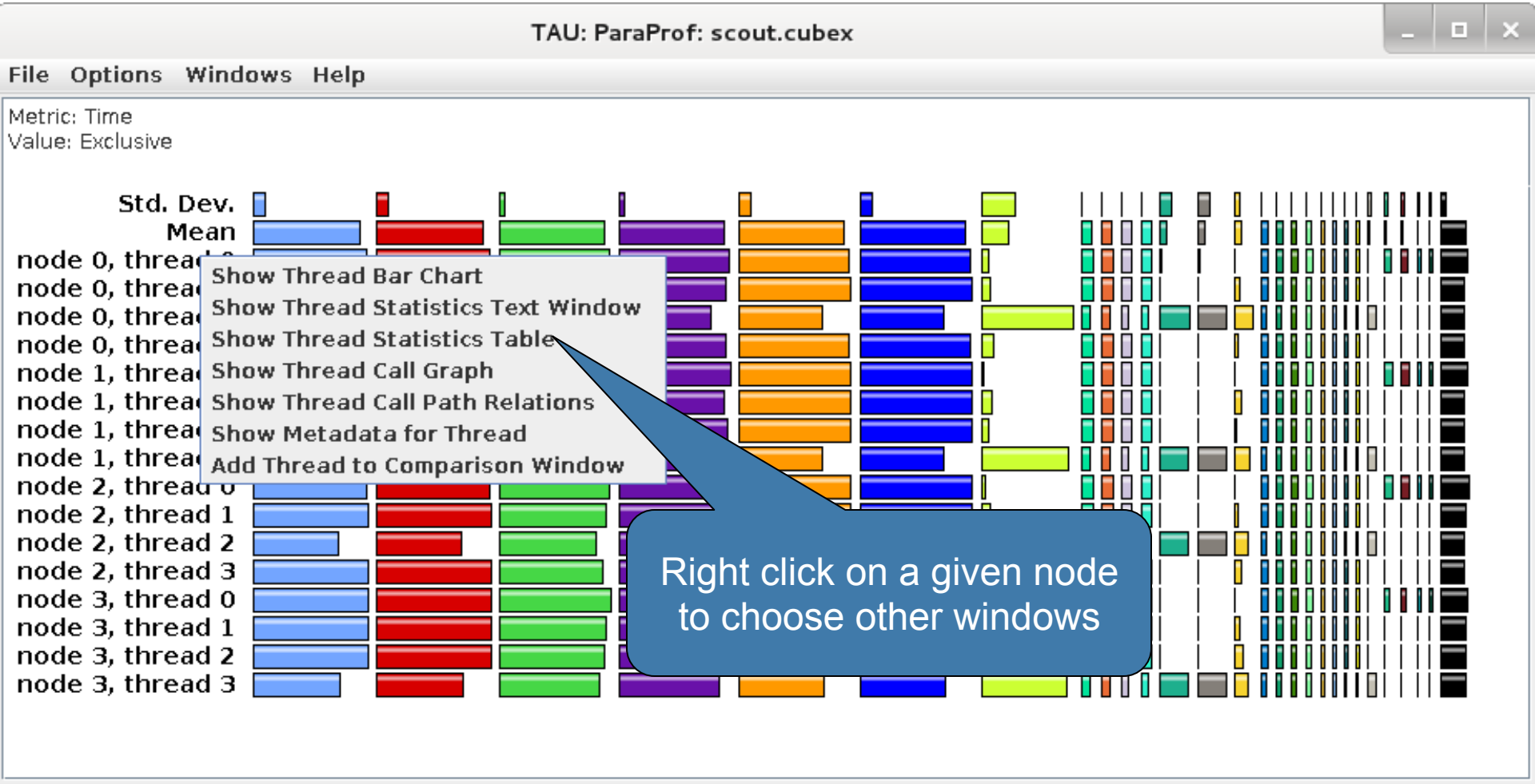
ParaProf: Options



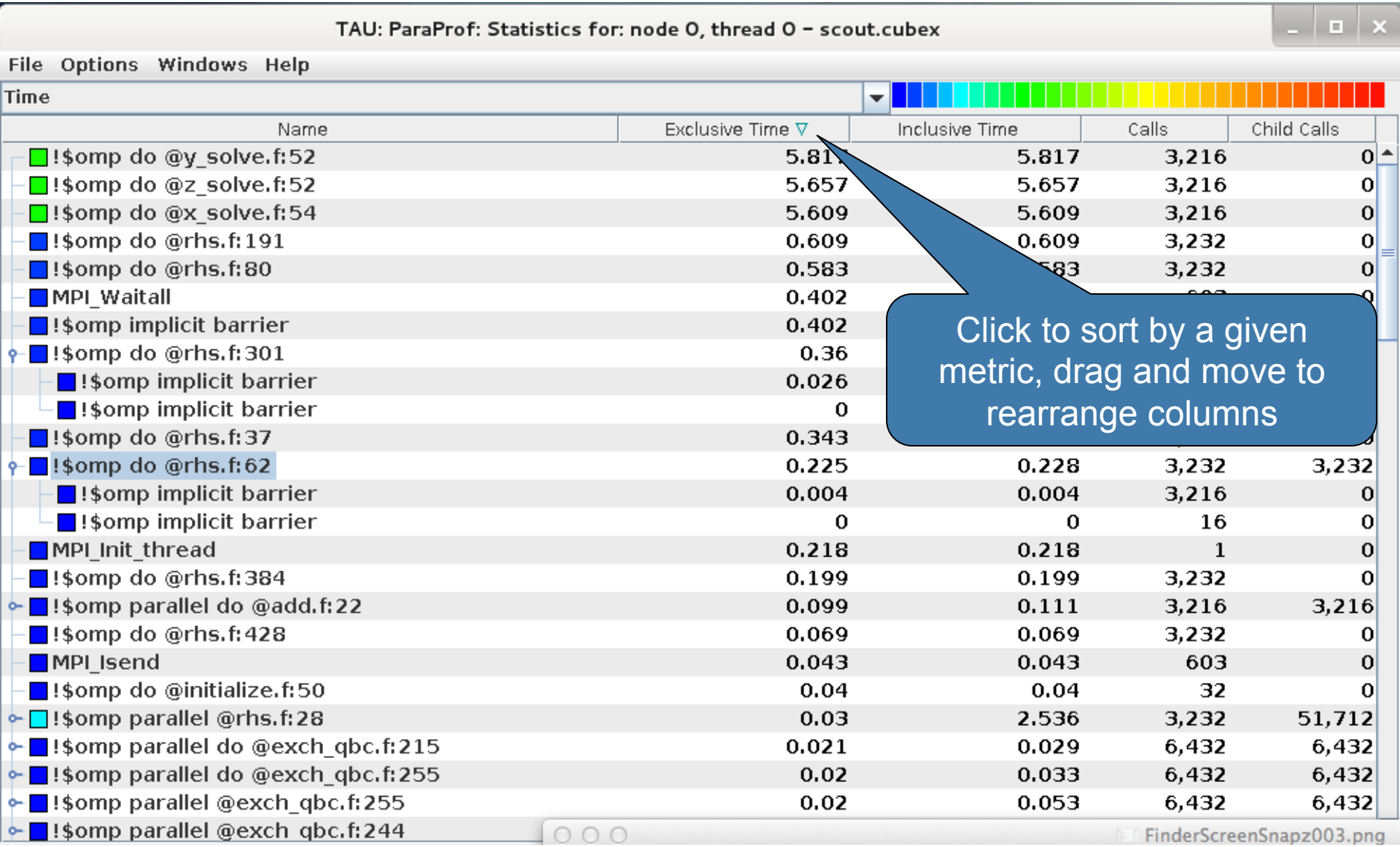
ParaProf: Stacked view



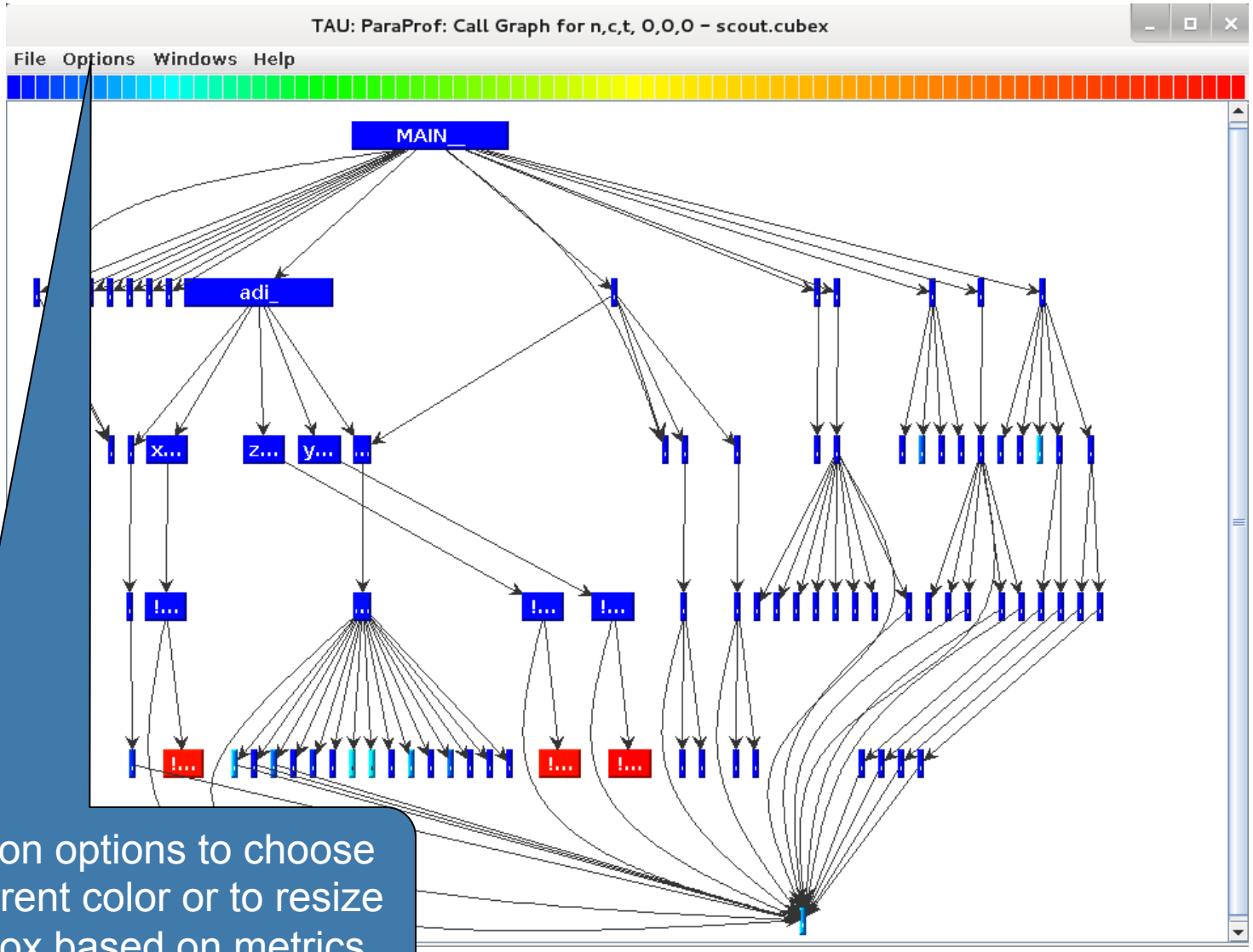
ParaProf: Windows



ParaProf: Thread Statistics Table

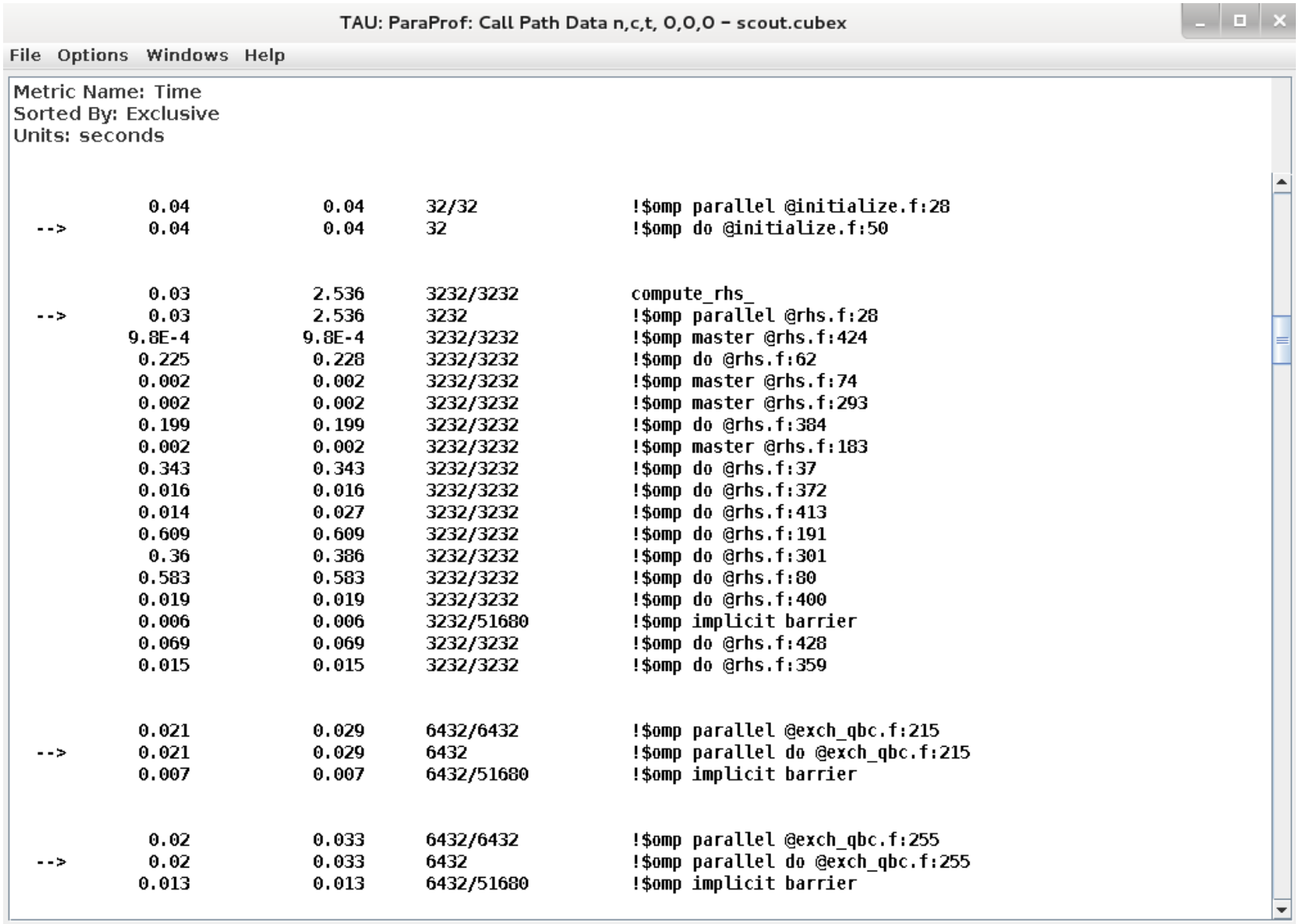


ParaProf: Thread Callgraph Window

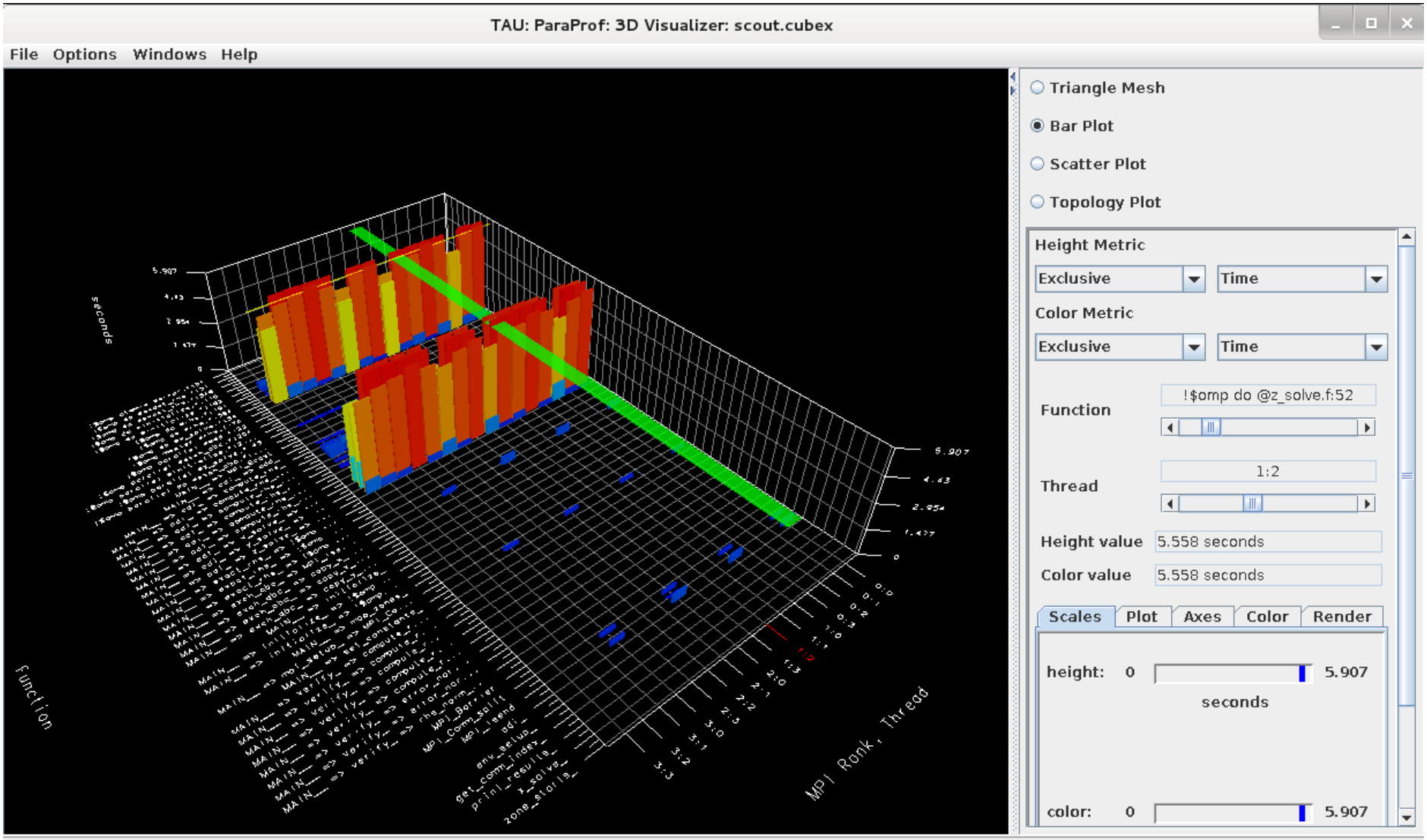


Click on options to choose a different color or to resize the box based on metrics

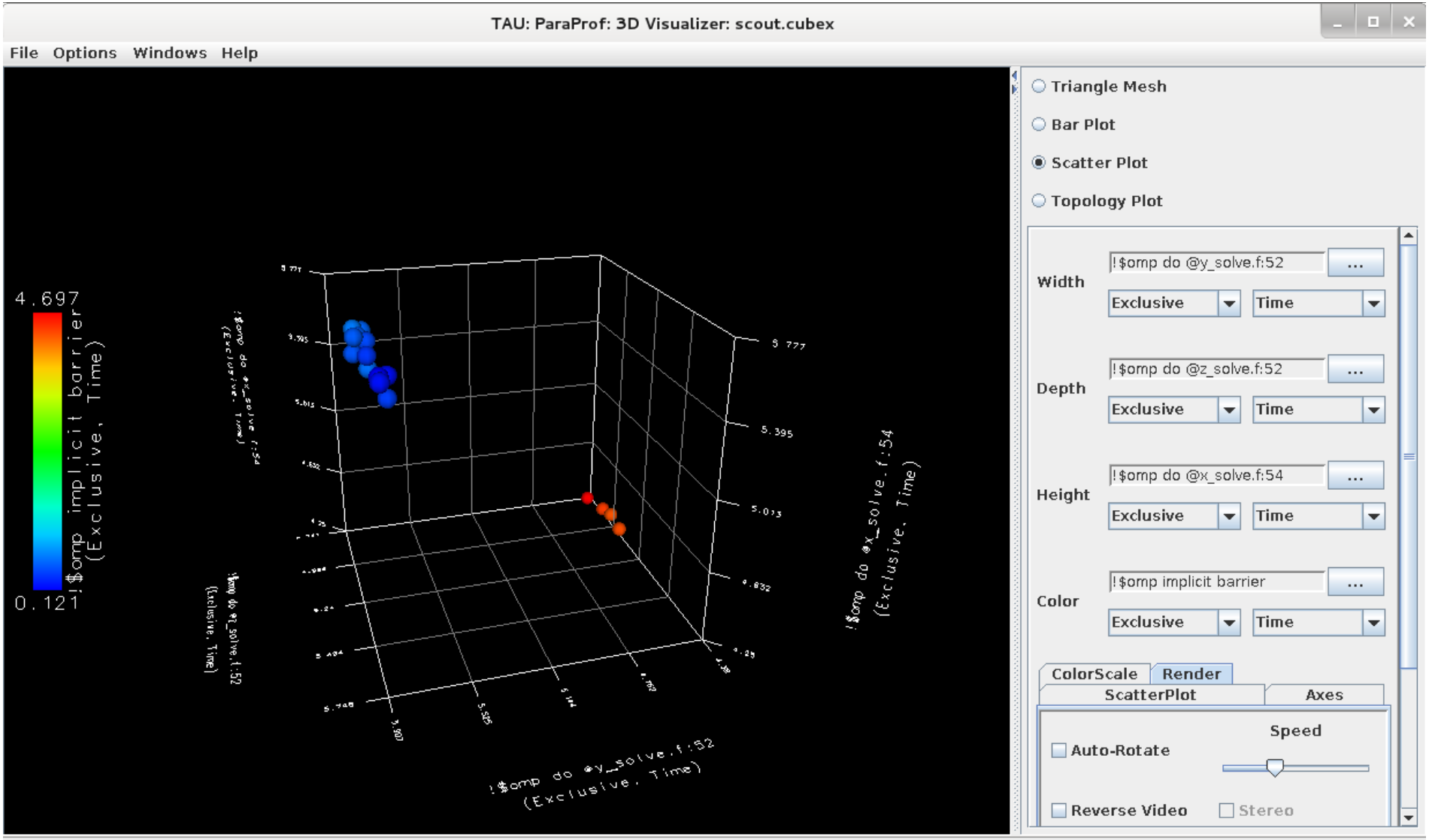
ParaProf: Callpath Thread Relations Window



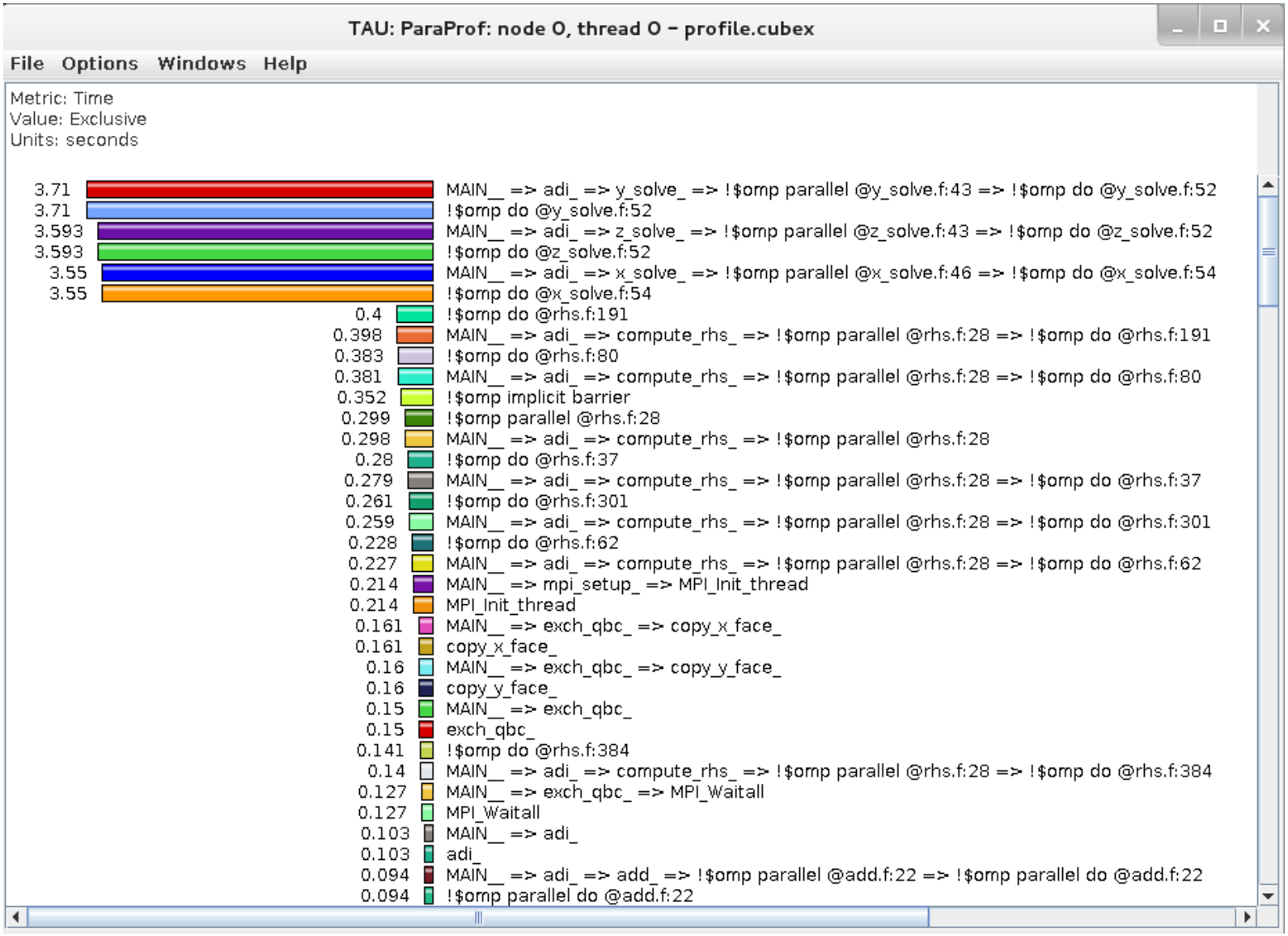
ParaProf:Windows -> 3D Visualization -> Bar Plot



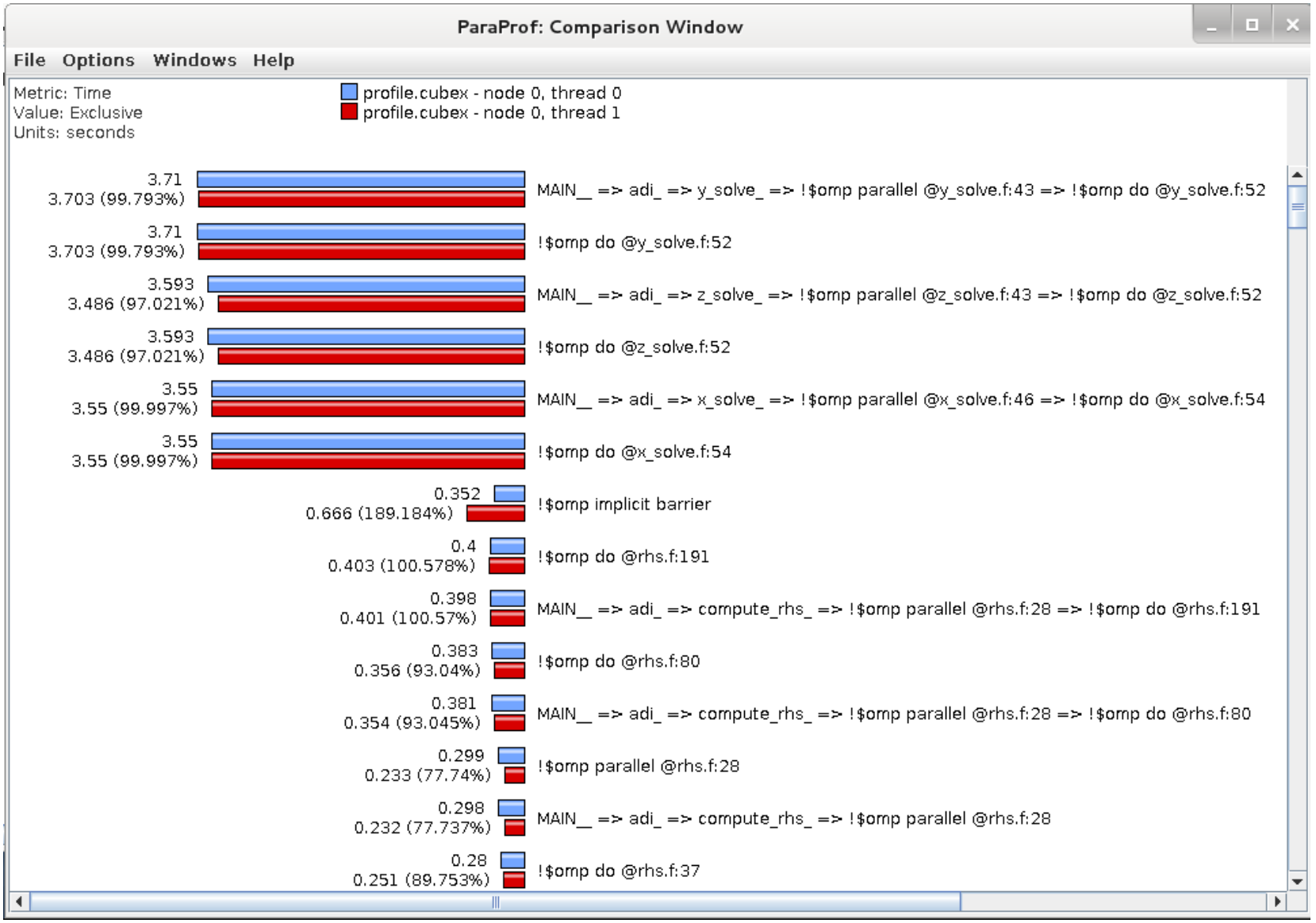
ParaProf: 3D Scatter Plot



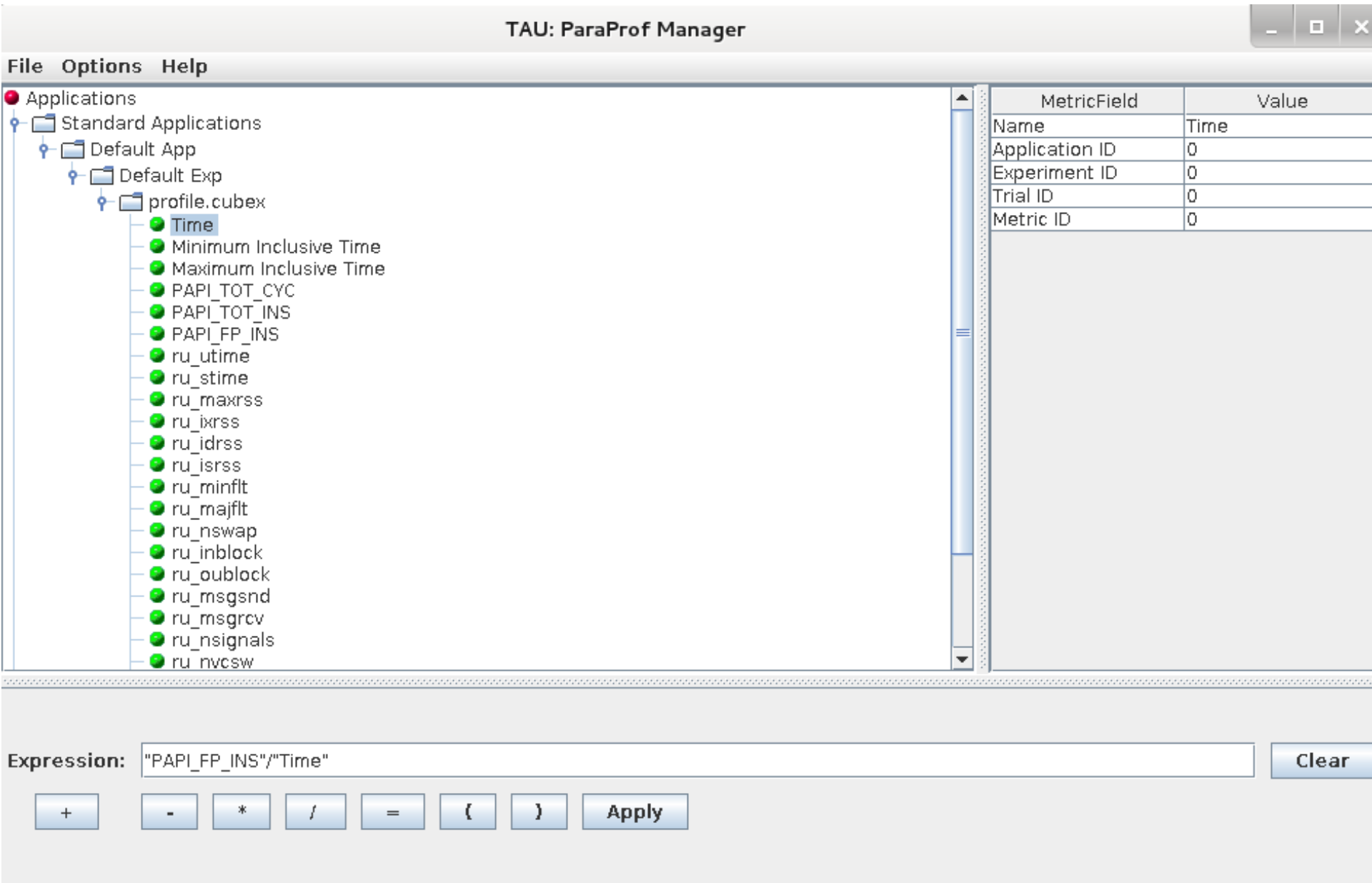
ParaProf: Node View



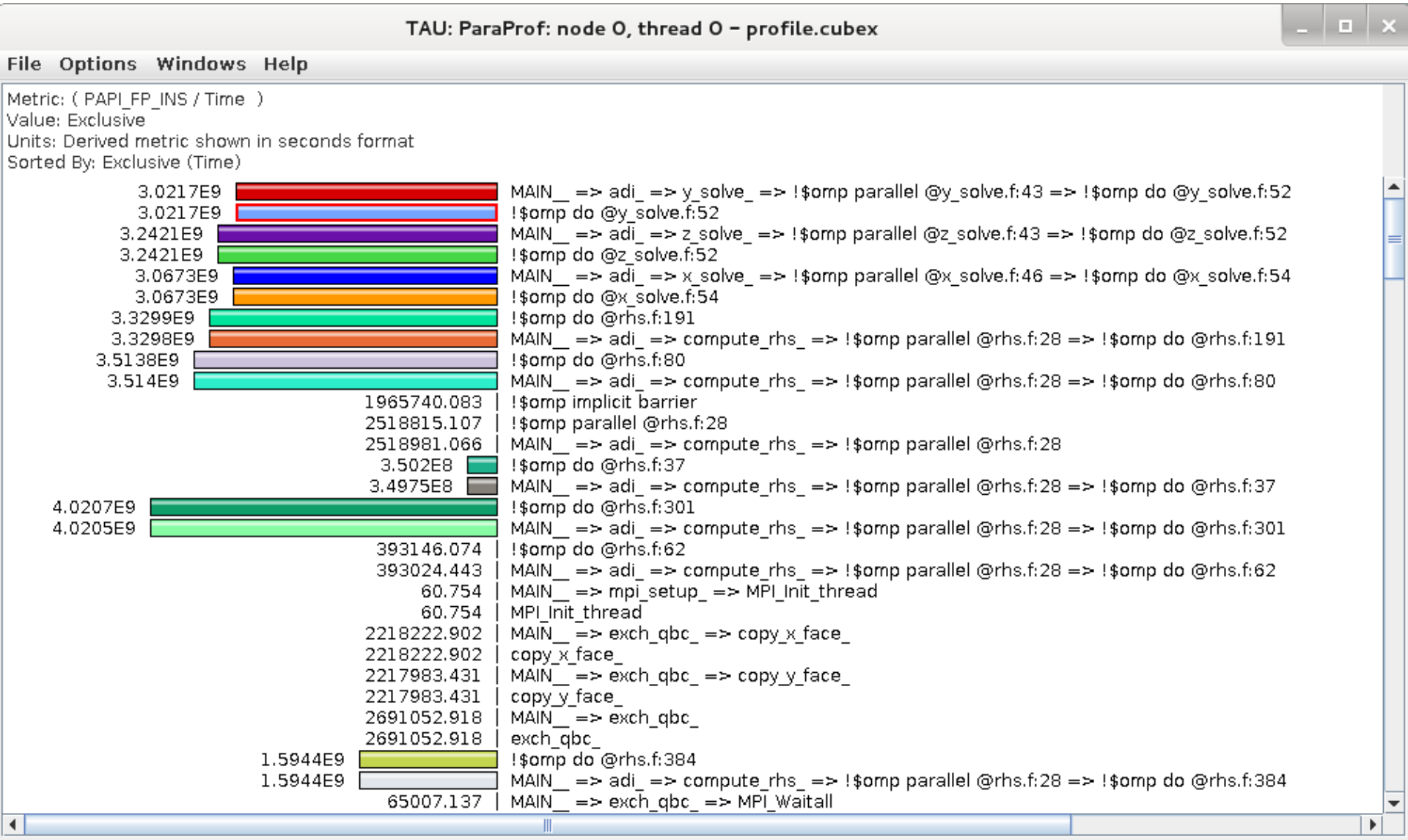
ParaProf: Add Thread to Comparison Window



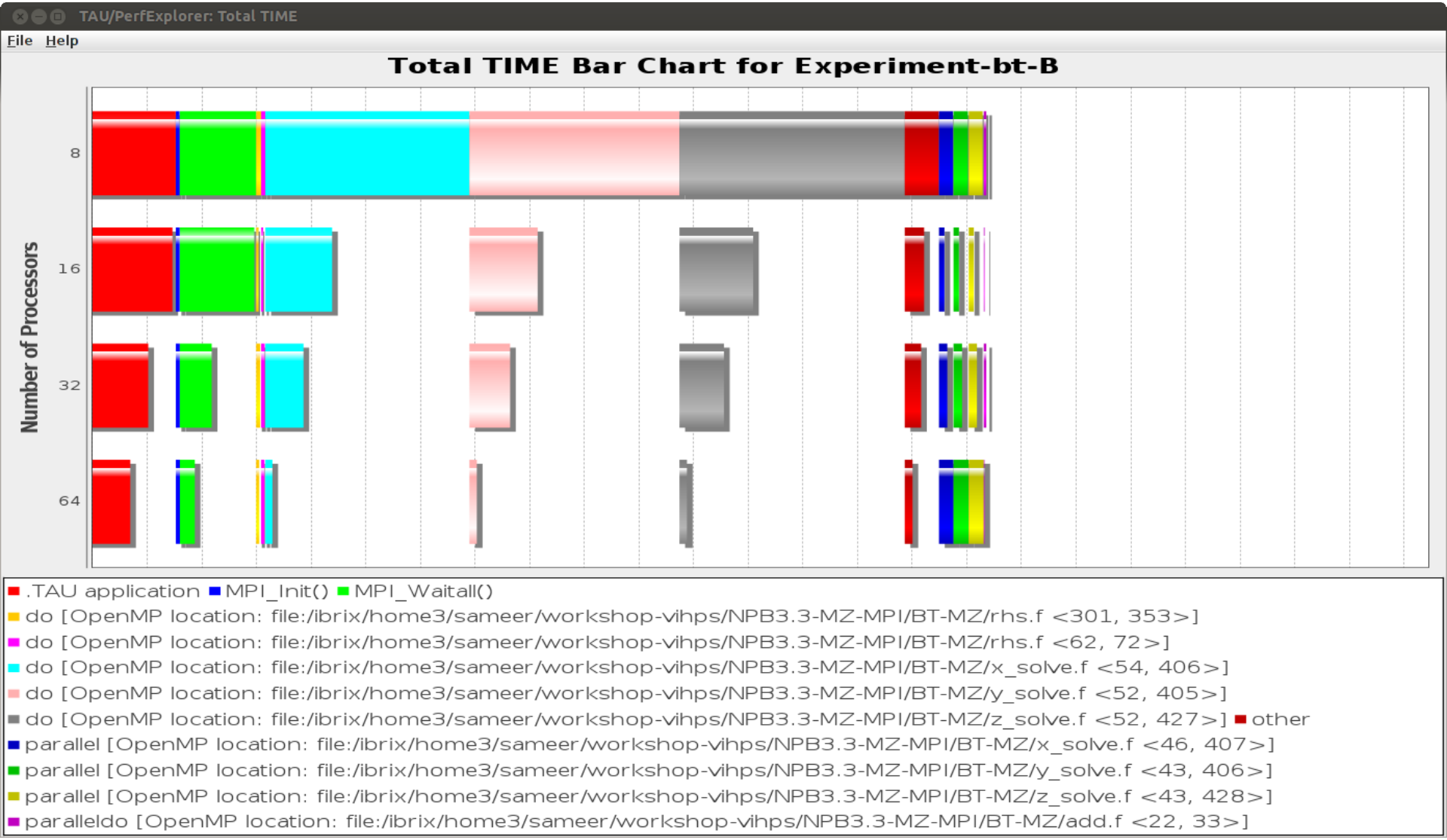
ParaProf: Options -> Derived Metric Panel



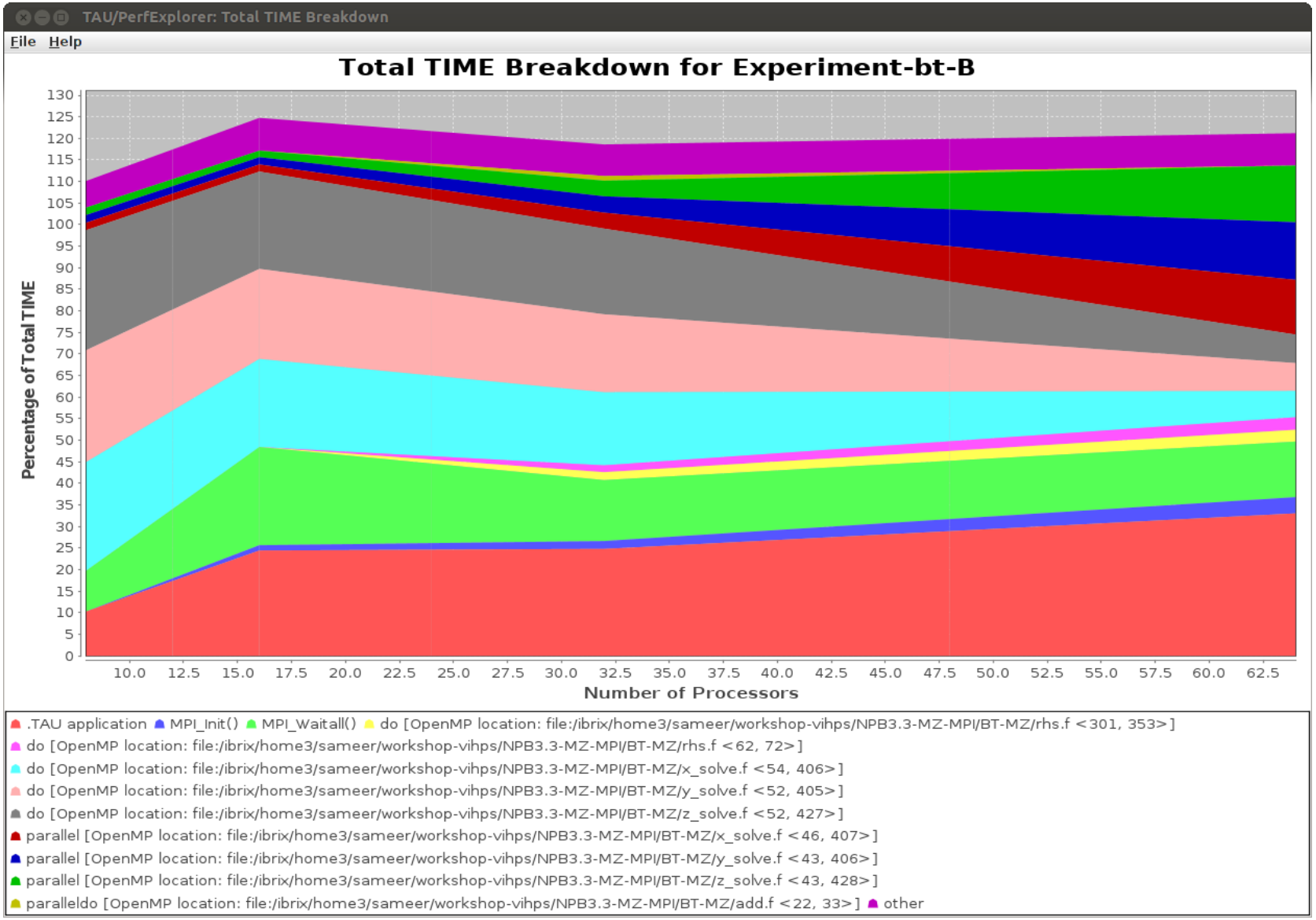
Sorting Derived Flops Metric by Exclusive Time



PerfExplorer: Scalability Analysis



PerfExplorer: Scalability Analysis



Support Acknowledgments

- U.S. Department of Energy (DOE)
  - Advanced Scientific Computing Research
  - Office of Science
  - ASC/NNSA, Tri-labs (LLNL,LANL, SNL)
- U.S. Department of Defense (DoD)
  - HPC Modernization Office (HPCMO)
- NSF Office of Cyberinfrastructure
- Juelich Supercomputing Center, NIC
- Argonne National Laboratory
- Technical University Dresden
- ParaTools, Inc.

ParaTools