TAU Performance System®

Tuning and Analysis Utilities
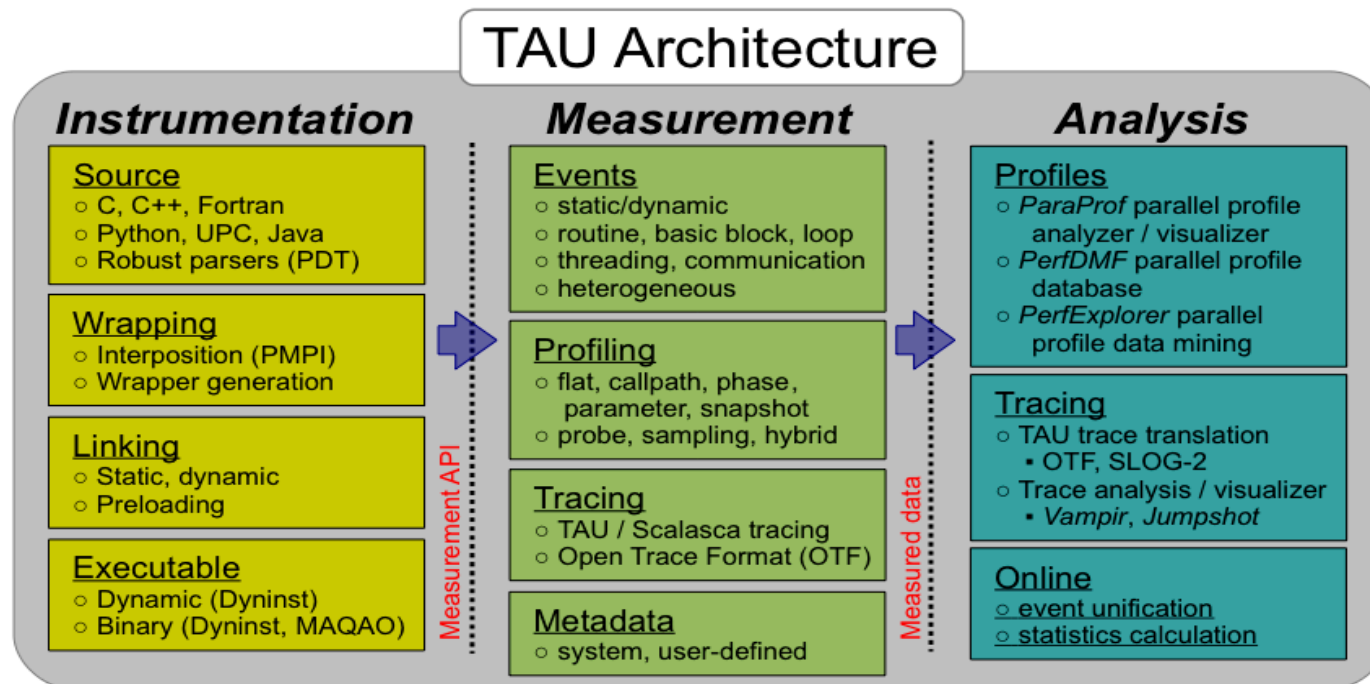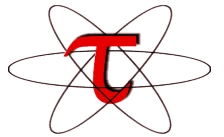
Sameer Shende
sameer@cs.uoregon.edu
University of Oregon
http://tau.uoregon.edu

TAU Performance System®

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

TAU Architecture

TAU Performance System

- Instrumentation
  - Fortran, C++, C, UPC, Java, Python, Chapel
  - Automatic instrumentation
- Measurement and analysis support
  - MPI, OpenSHMEM, ARMCI, PGAS, DMAPP
  - pthreads, OpenMP, OMPT interface, hybrid, other thread models
  - GPU, CUDA, OpenCL, OpenACC, ROCm, HIP
  - Parallel profiling and tracing
  - Use of Score-P for native OTF2 and CUBEX generation
  - Efficient callpath proflles and trace generation using Score-P
- Analysis
  - Parallel profile analysis (ParaProf), data mining (PerfExplorer)
  - Performance database technology (TAUdb)
  - 3D profile browser

TAU Performance System

- TAU supports both sampling and direct instrumentation
- Memory debugging as well as I/O performance evaluation
- Profiling as well as tracing
- Interfaces with Score-P for more efficient measurements
- TAU's instrumentation covers:
  - Runtime library interposition (tau_exec)
  - Compiler-based instrumentation
  - Native generation of OTF2 traces (TAU_TRACE=1, TAU_TRACE_FORMAT=otf2)
  - Callsite instrumentation with profiles and traces (TAU_CALLSITE=1)
  - PDT based Source level instrumentation: routine & loop
  - Event based sampling (TAU_SAMPLING=1 or tau_exec -ebs)
  - Callstack unwinding with sampling (TAU_EBS_UNWIND=1)
  - OpenMP Tools Interface TR6 (OMPT, tau_exec –T ompt,tr6)
  - CUDA CUPTI, OpenCL (tau_exec -T cupti -cupti)

Application Performance Engineering using TAU

- How much time is spent in each application routine and outer *loops*? Within loops, what is the contribution of each *statement*? What is the time spent in OpenMP loops?

- How many instructions are executed in these code regions?
  Floating point, Level 1 and 2 *data cache misses*, hits, branches taken? What is the extent of vectorization for loops on Intel MIC?

- What is the memory usage of the code? When and where is memory allocated/de-allocated? Are there any memory leaks? What is the memory footprint of the application? What is the memory high water mark?

- How much energy does the application use in Joules? What is the peak power usage?

- What are the I/O characteristics of the code? What is the peak read and write *bandwidth* 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?

Inclusive vs. Exclusive values

- Inclusive
  - Information of all sub-elements aggregated into single value
- Exclusive
  - Information cannot be subdivided further


```
int foo()
{
    int a;
    a = 1 + 1;

    bar();

    a = a + 1;
    return a;
}
```

Inclusive   Exclusive
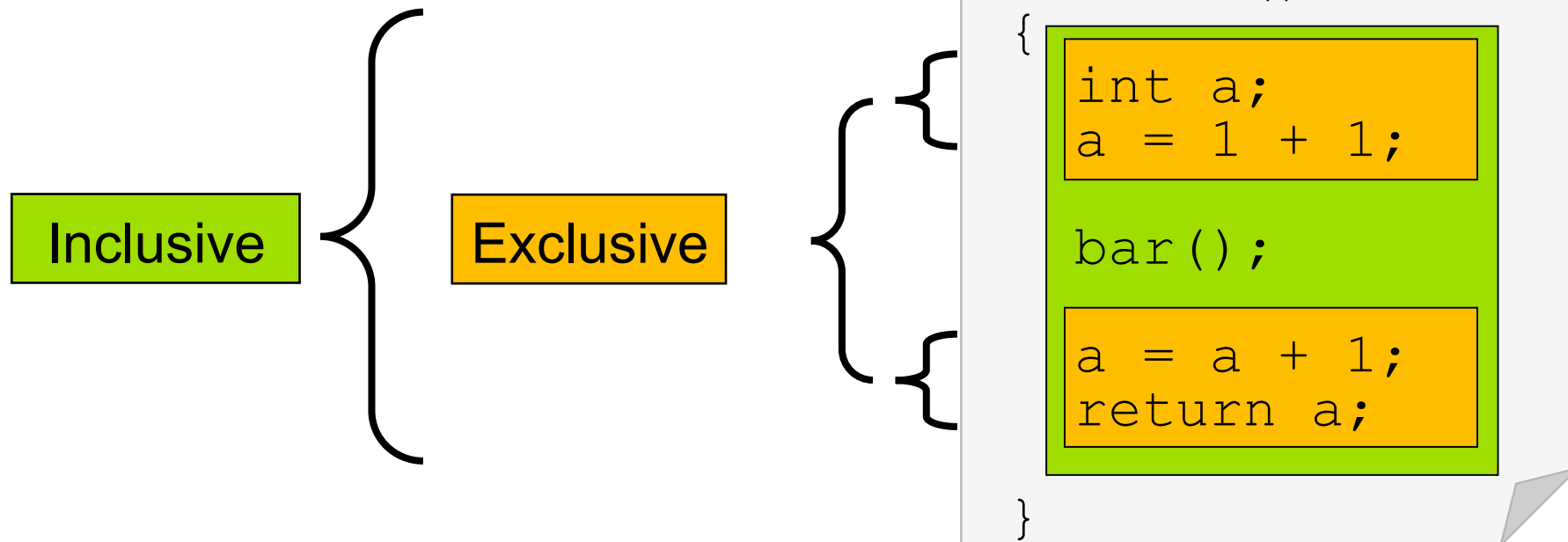
Performance Data Measurement

Direct via Probes

```
Call START('potential')
// code
Call STOP('potential')
```

- Exact measurement
- Fine-grain control
- Calls inserted into code

Indirect via Sampling
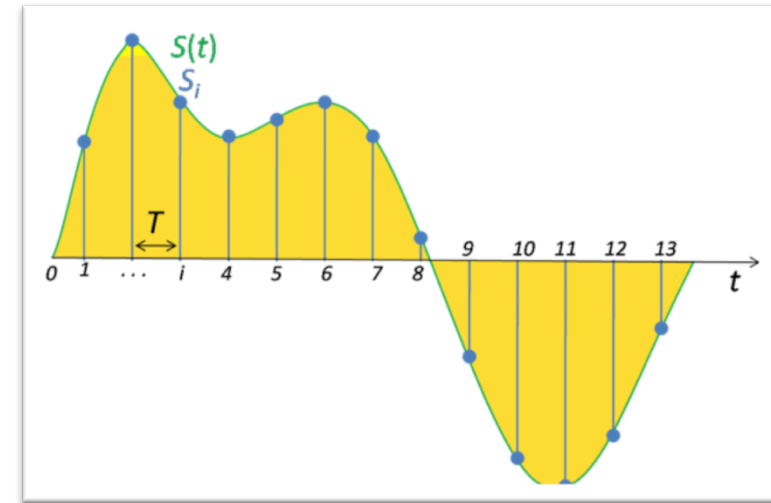
- No code modification
- Minimal effort
- Relies on debug symbols (**-g**)

Event-Based Sampling (EBS)

| main | foo(0) | foo(1) | foo(2) | Measurement |
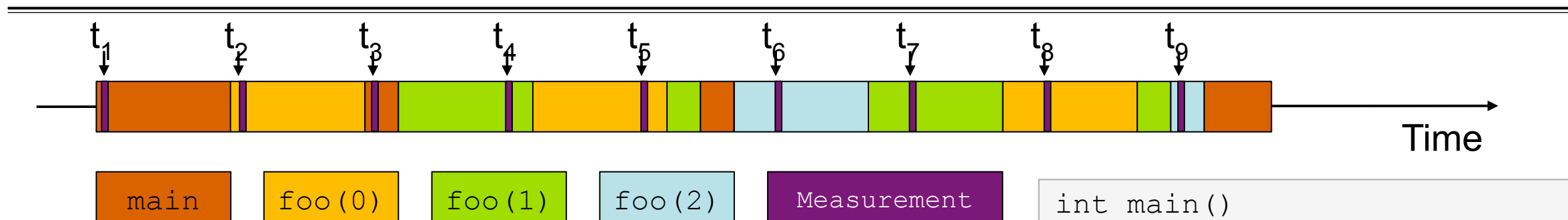
Time

- Running program is periodically interrupted to take measurement
  - Timer interrupt, OS signal, or HWC overflow
  - Service routine examines return-address stack
  - Addresses are mapped to routines using symbol table information
- Statistical inference of program behavior
  - Not very detailed information on highly volatile metrics
  - Requires long-running applications
- Works with unmodified executables (tau_exec –ebs)

```c
int main()
{
   int i;

   for (i=0; i < 3; i++)
      foo(i);

   return 0;
}

void foo(int i)
{

   if (i > 0)
      foo(i - 1);

}
```

Instrumentation

| main | foo(0) | foo(1) | foo(2) | Measurement |
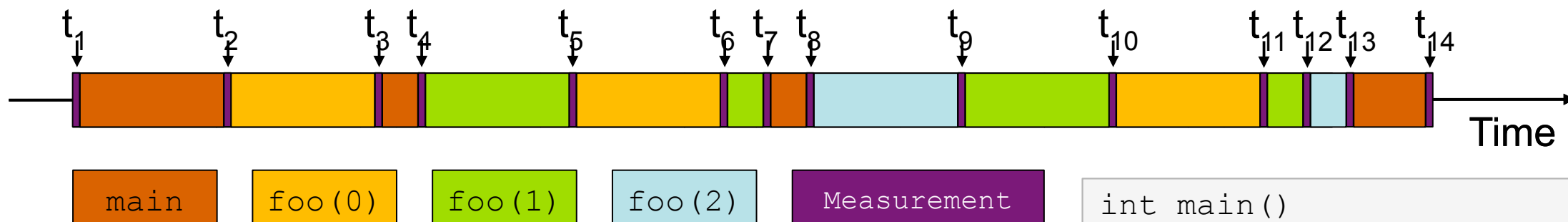
- Measurement code is inserted such that every event of interest is captured directly
  - Can be done in various ways
- Advantage:
  - Much more detailed information
- Disadvantage:
  - Processing of source-code / executable necessary
  - Large relative overheads for small functions

```
int main()
{
    int i;
    TAU_START("main");
    for (i=0; i < 3; i++)
        foo(i);
    TAU_STOP("main");
    return 0;
}

void foo(int i)
{
    TAU_START("foo");
    if (i > 0)
        foo(i - 1);
    TAU_STOP("foo");
}
```

How much data do you want?

Limited
Profile

Loop
Profile

Callpath
Profile

O(KB)

O(TB)

**Flat
Profile**

Callsite
Profile

Trace
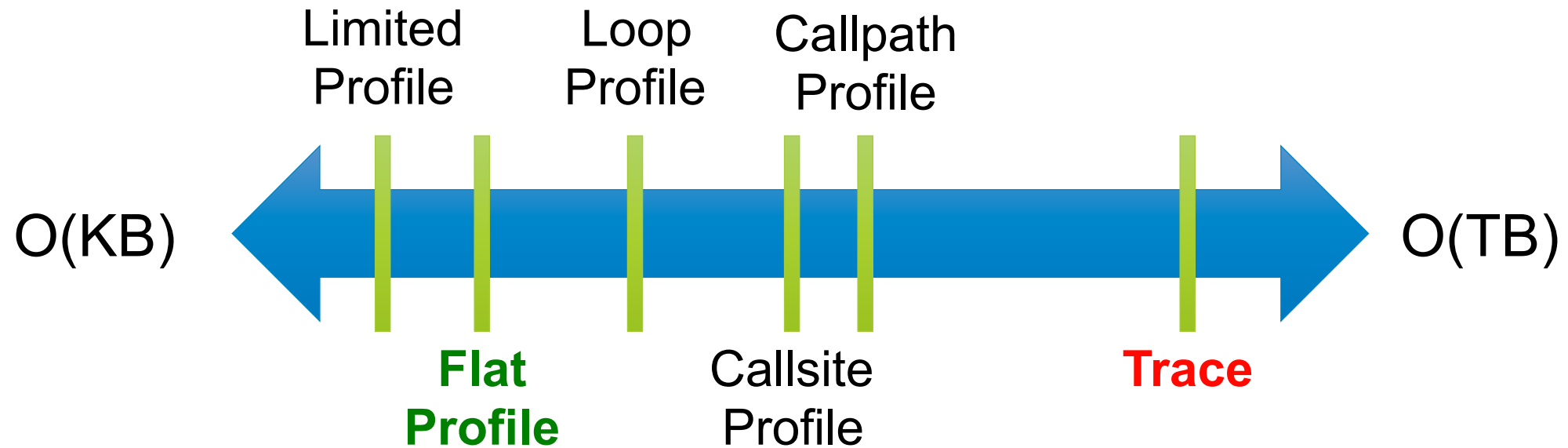
Types of Performance Profiles

- *Flat* profiles
  - Metric (e.g., time) spent in an event
  - Exclusive/inclusive, # of calls, child calls, …
- *Callpath* profiles
  - Time spent along a calling path (edges in callgraph)
  - "*main=> f1 => f2 => MPI_Send*"
  - Set the TAU_CALLPATH and TAU_CALLPATH_DEPTH environment variables
- *Callsite* profiles
  - Time spent along in an event at a given source location
  - Set the TAU_CALLSITE environment variable
- *Phase* profiles
  - Flat profiles under a phase (nested phases allowed)
  - Default "main" phase
  - Supports static or dynamic (e.g. per-iteration) phases

Using TAU's Runtime Preloading Tool: tau_exec

- Preload a wrapper that intercepts the runtime system call and substitutes with another

  - **MPI**

  - **OpenMP**

  - **POSIX I/O**

  - **Memory allocation/deallocation routines**

  - **Wrapper library for an external package**

- No modification to the binary executable!

- Enable other TAU options (communication matrix, OTF2, event-based sampling)

- Add tau_exec before the name of the binary

  - srun tau_exec ./a.out

  - srun tau_exec −T ompt,tr6,mpi,papi -ompt ./a.out

tau_exec

```
$ tau_exec

Usage: tau_exec [options] [--] <exe> <exe options>

Options:
        -v              Verbose mode
        -s              Show what will be done but don't actually do anything (dryrun)
        -qsub           Use qsub mode (BG/P only, see below)
        -io             Track I/O
        -memory         Track memory allocation/deallocation
        -memory_debug   Enable memory debugger
        -cuda           Track GPU events via CUDA
        -cupti          Track GPU events via CUPTI (Also see env. variable TAU_CUPTI_API)
        -opencl         Track GPU events via OpenCL
        -openacc        Track GPU events via OpenACC (currently PGI only)
        -ompt           Track OpenMP events via OMPT interface
        -armci          Track ARMCI events via PARMCI
        -ebs            Enable event-based sampling
        -ebs_period=<count> Sampling period (default 1000)
        -ebs_source=<counter> Counter (default itimer)
        -um             Enable Unified Memory events via CUPTI
        -T <DISABLE,GNU,ICPC,MPI,OMPT,OPENMP,PAPI,PDT,PROFILE,PTHREAD,SCOREP,SERIAL> : Specify TAU tags
        -loadlib=<file.so>   : Specify additional load library
        -XrunTAUsh-<options> : Specify TAU library directly
        -gdb            Run program in the gdb debugger

Notes:
        Defaults if unspecified: -T MPI
        MPI is assumed unless SERIAL is specified
```

- Tau_exec preloads the TAU wrapper libraries and performs measurements.

No need to recompile the application!

tau_exec Example (continued)

```
Example:
    mpirun –np 2 tau_exec -T icpc,ompt,mpi  -ompt ./a.out
    aprun -n 2 tau_exec -io ./a.out
Example - event-based sampling with samples taken every 1,000,000 FP instructions
    aprun -n 8 tau_exec -ebs -ebs_period=1000000 -ebs_source=PAPI_FP_INS ./ring
Examples - GPU:
    tau_exec -T serial,cupti -cupti ./matmult (Preferred for CUDA 4.1 or later)
    tau_exec -openacc ./a.out
    tau_exec -T serial –opencl ./a.out (OPENCL)
    mpirun –np 2 tau_exec -T mpi,cupti,papi -cupti -um ./a.out (Unified Virtual Memory in CUDA 6.0+)

qsub mode (IBM BG/Q only):
    Original:
      qsub –n 1 --mode smp -t 10 ./a.out
    With TAU:
      tau_exec -qsub -io -memory -- qsub -n 1 … -t 10 ./a.out

Memory Debugging:
    -memory option:
      Tracks heap allocation/deallocation and memory leaks.
    -memory_debug option:
      Detects memory leaks, checks for invalid alignment, and checks for
      array overflow.  This is exactly like setting TAU_TRACK_MEMORY_LEAKS=1
      and TAU_MEMDBG_PROTECT_ABOVE=1 and running with -memory
```

- tau_exec can enable event based sampling while launching the executable using the –ebs flag!

TAU's Source Instrumentation

TAU's Static Analysis System: Program Database Toolkit (PDT)
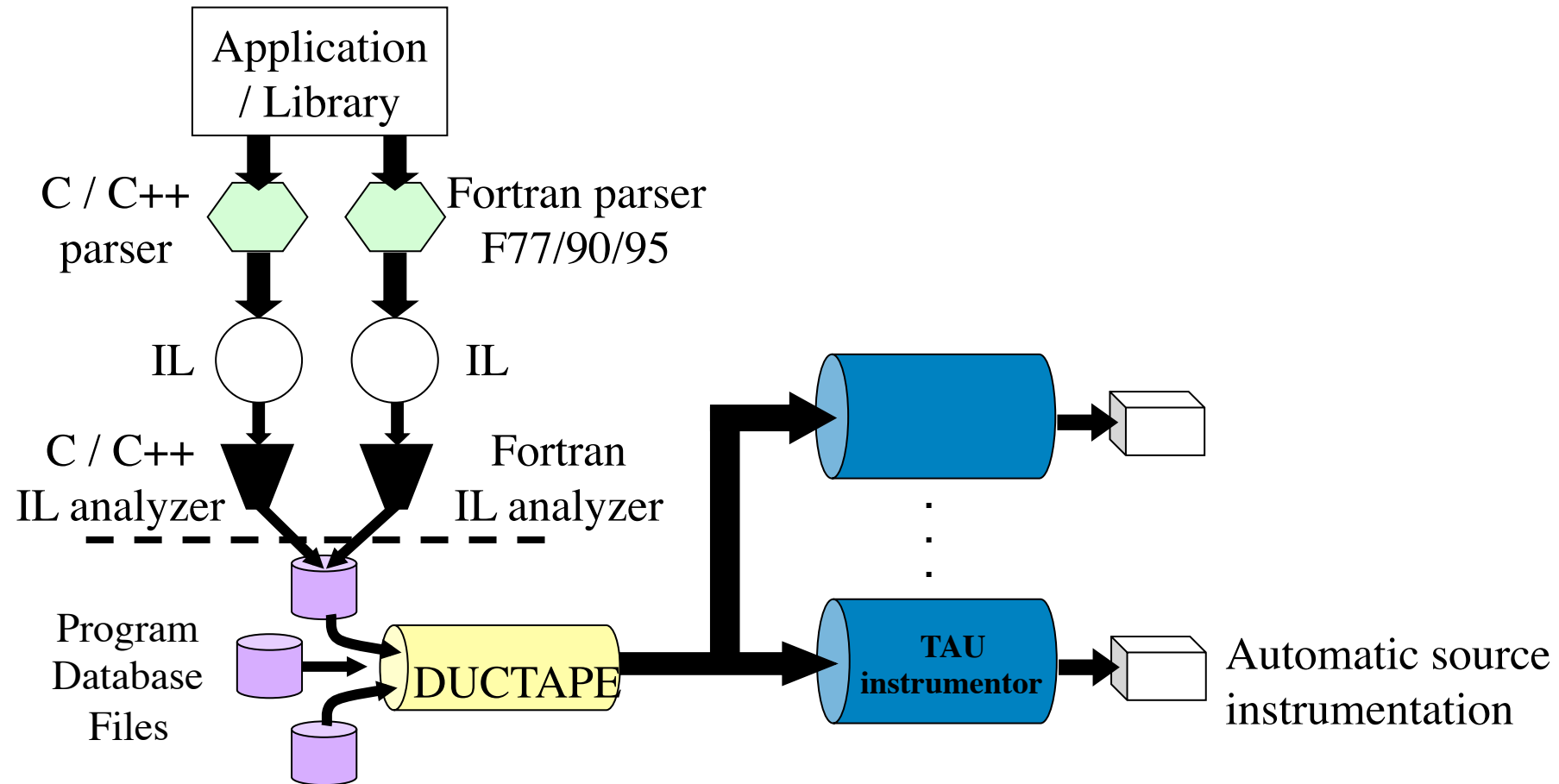
Automatic Source Instrumentation using PDT
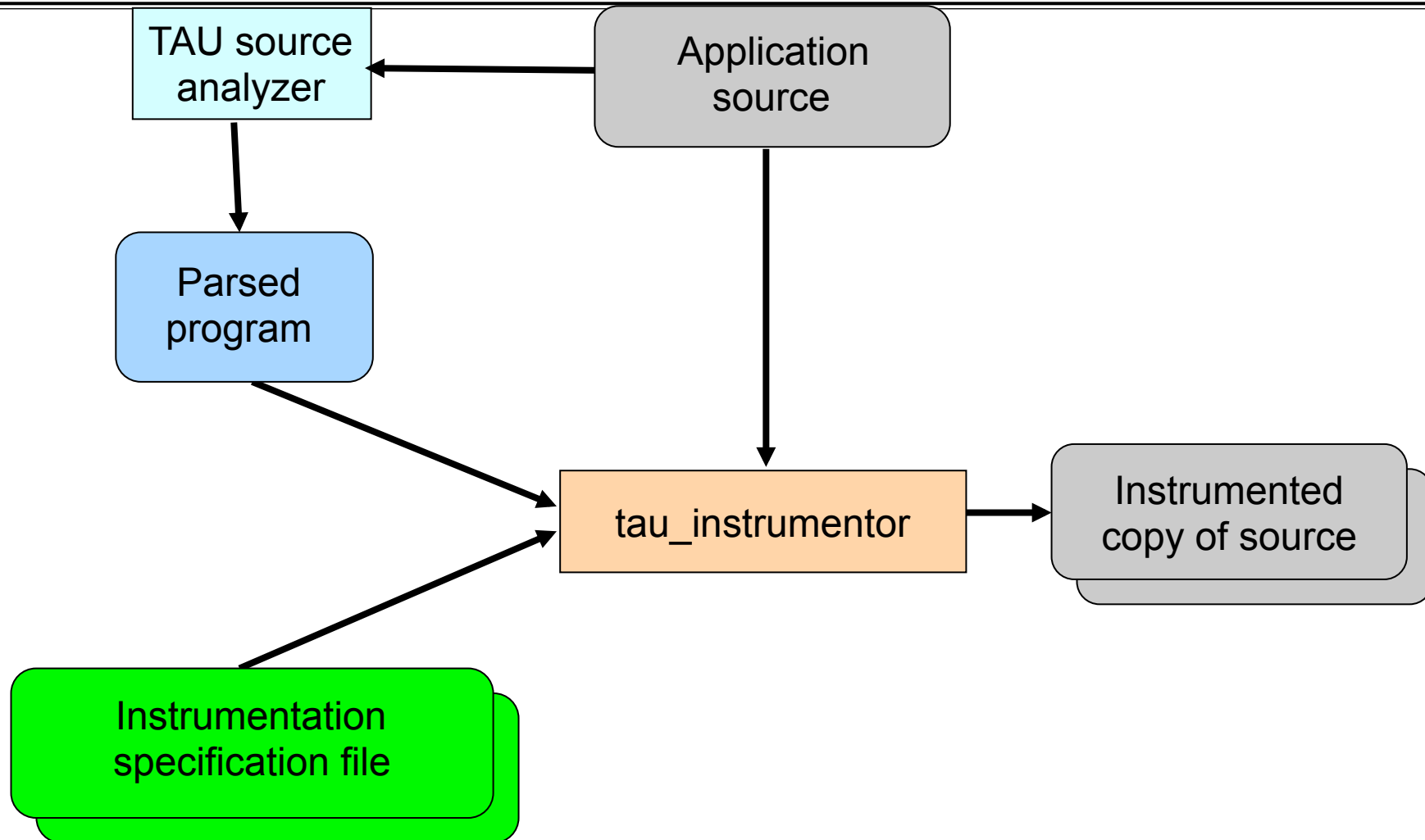
Installing and Configuring TAU

- Installing PDT:
  - wget http://tau.uoregon.edu/pdt.tgz
  - ./configure; make ; make install

- Installing TAU:
  - wget http://tau.uoregon.edu/tau.tgz
  - ./configure –ompt=download-tr6 –c++=mpiicpc –cc=mpiicc –fortran=mpiifort -mpi -bfd=download -pdt=<dir> -papi=<dir> ...
  - make install; export PATH=<taudir>/x86_64/bin:$PATH

- Using TAU for source instrumentation:
  - export TAU_MAKEFILE=<taudir>/x86_64/lib/Makefile.tau-<TAGS>
  - make CC=tau_cc.sh   CXX=tau_cxx.sh   F90=tau_f90.sh

Using TAU's Source Code Instrumentation

- TAU supports several compilers, measurement, and thread options

  Intel compilers, profiling with hardware counters using PAPI, MPI library, CUDA…

  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:

- ```
  % jutil env activate -p cjzam11 -A jzam11
  ```

- ```
  % module use /p/scratch/share/VI-HPS/JURECA/mf
  ```

- ```
  % module load tau
  ```

- ```
  % export TAU_MAKEFILE=$TAU/Makefile.tau-icpc-papi-mpi-pdt-openmp-opari
  % export TAU_OPTIONS='-optVerbose …'  (see tau_compiler.sh )
  ```

  Use tau_f90.sh, tau_f77.sh , tau_cxx.sh, tau_upc.sh, or tau_cc.sh as F90, F77, C++, UPC, or C compilers respectively:

  % mpif90 foo.f90   changes to

  % tau_f90.sh foo.f90

- Set runtime environment variables, execute application and analyze performance data:

  % pprof   (for text based profile display)

  % paraprof (for GUI)

Makefiles for TAU Compiler and Runtime Options (Jureca)

```
% module use /p/scratch/share/VI-HPS/JURECA/mf; module load tau
% echo $TAU
/p/scratch/share/VI-HPS/JURECA/packages/tau2.28.1-intel-cluster/x86_64/lib
% ls $TAU/Makefile.*
```

/p/scratch/share/VI-HPS/JURECA/packages/tau2.28.1-intel-cluster/x86_64/lib/Makefile.tau-icpc-papi-mpi-pdt-openmp-opari

/p/scratch/share/VI-HPS/JURECA/packages/tau2.28.1-intel-cluster/x86_64/lib/Makefile.tau-icpc-papi-mpi-pthread-pdt

/p/scratch/share/VI-HPS/JURECA/packages/tau2.28.1-intel-cluster/x86_64/lib/Makefile.tau-icpc-papi-ompt-tr6-mpi-pdt-openmp

/p/scratch/share/VI-HPS/JURECA/packages/tau2.28.1-intel-cluster/x86_64/lib/Makefile.tau-icpc-papi-ompt-tr6-pdt-openmp

/p/scratch/share/VI-HPS/JURECA/packages/tau2.28.1-intel-cluster/x86_64/lib/Makefile.tau-icpc-papi-pdt

/p/scratch/share/VI-HPS/JURECA/packages/tau2.28.1-intel-cluster/x86_64/lib/Makefile.tau-icpc-papi-pthread-pdt

For an MPI+OpenMP+F90 application with Intel MPI, you may choose

Makefile.tau-icpc-papi-mpi-pdt-openmp-opari
- Supports MPI instrumentation & PDT for automatic source instrumentation

```
% export TAU_MAKEFILE=$TAU/Makefile.tau-icpc-papi-mpi-pdt-openmp-opari
% tau_f90.sh matmult.f90 -o matmult
% aprun -n 16 ./matmult
% paraprof
```

Configuration tags for tau_exec

```
% ./configure –pdt=<dir> -mpi –papi=<dir>; make install
Creates in $TAU:
Makefile.tau-papi-mpi-pdt(Configuration parameters in stub makefile)
shared-papi-mpi-pdt/libTAU.so

% ./configure –pdt=<dir> -mpi; make install   creates
Makefile.tau-mpi-pdt
shared-mpi-pdt/libTAU.so

To explicitly choose preloading of shared-<options>/libTAU.so change:
% aprun –n 256 ./a.out      to
% aprun –n 256  tau_exec –T <comma_separated_options> ./a.out

% aprun –n 256  tau_exec –T papi,mpi,pdt ./a.out
Preloads $TAU/shared-papi-mpi-pdt/libTAU.so
% aprun –n 256  tau_exec –T papi ./a.out
Preloads $TAU/shared-papi-mpi-pdt/libTAU.so by matching.
% aprun –n 256 tau_exec –T papi,mpi,pdt -s ./a.out
Does not execute the program. Just displays the library that it will preload if executed without the –s option.
NOTE: -mpi configuration is selected by default. Use –T serial for
Sequential programs.
```

Simplifying TAU's usage (tau_exec)

- Uninstrumented execution
  - % aprun -n 16 ./a.out
- Track MPI performance
  - % aprun -n 16 tau_exec ./a.out
- Track OpenMP, and MPI performance (MPI enabled by default)
  - % export TAU_OMPT_SUPPORT_LEVEL=full; export TAU_OMPT_RESOLVE_ADDRESS_EAGERLY=1
  - % aprun -n 16 tau_exec –T mpi,pdt,ompt,papi   –ompt ./a.out
- Track memory operations
  - % export TAU_TRACK_MEMORY_LEAKS=1
  - % mpirun –np 16 tau_exec –memory_debug ./a.out (bounds check)
- Use event based sampling (compile with –g)
  - % mpirun –np 16 tau_exec –ebs ./a.out
  - Also –ebs_source=<PAPI_COUNTER> -ebs_period=<overflow_count> -ebs_resolution=<file|function|line>
- Load wrapper interposition library
  - % mpirun –np 16 tau_exec –loadlib=<path/libwrapper.so> ./a.out
- Track GPGPU operations (-rocm, -opencl, -cupti, -cupti –um, -openacc):
  - % mpirun –np 16 tau_exec –cupti ./a.out

TAU's Analysis Tools: ParaProf

TAU Analysis
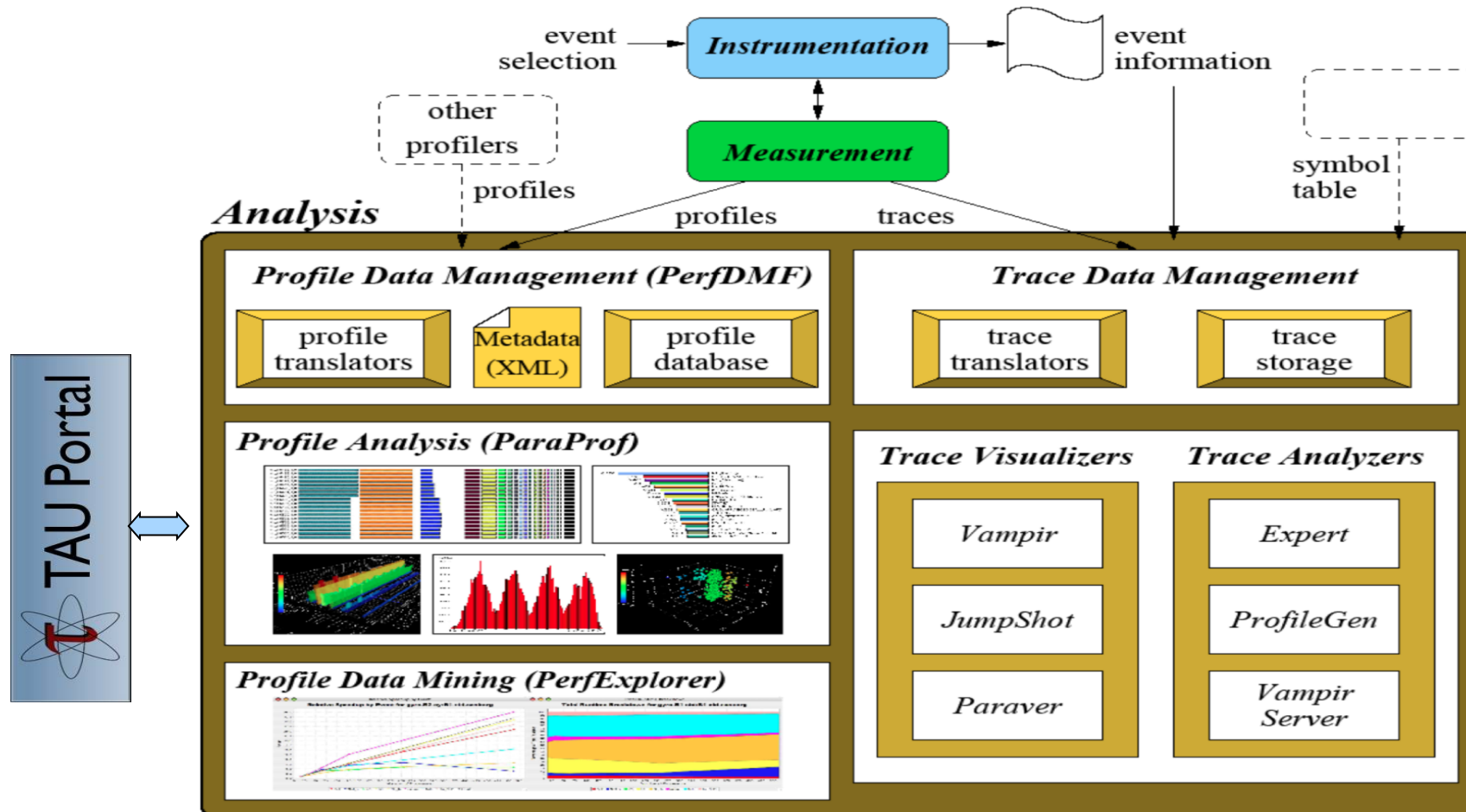
ParaProf Profile Analysis Framework
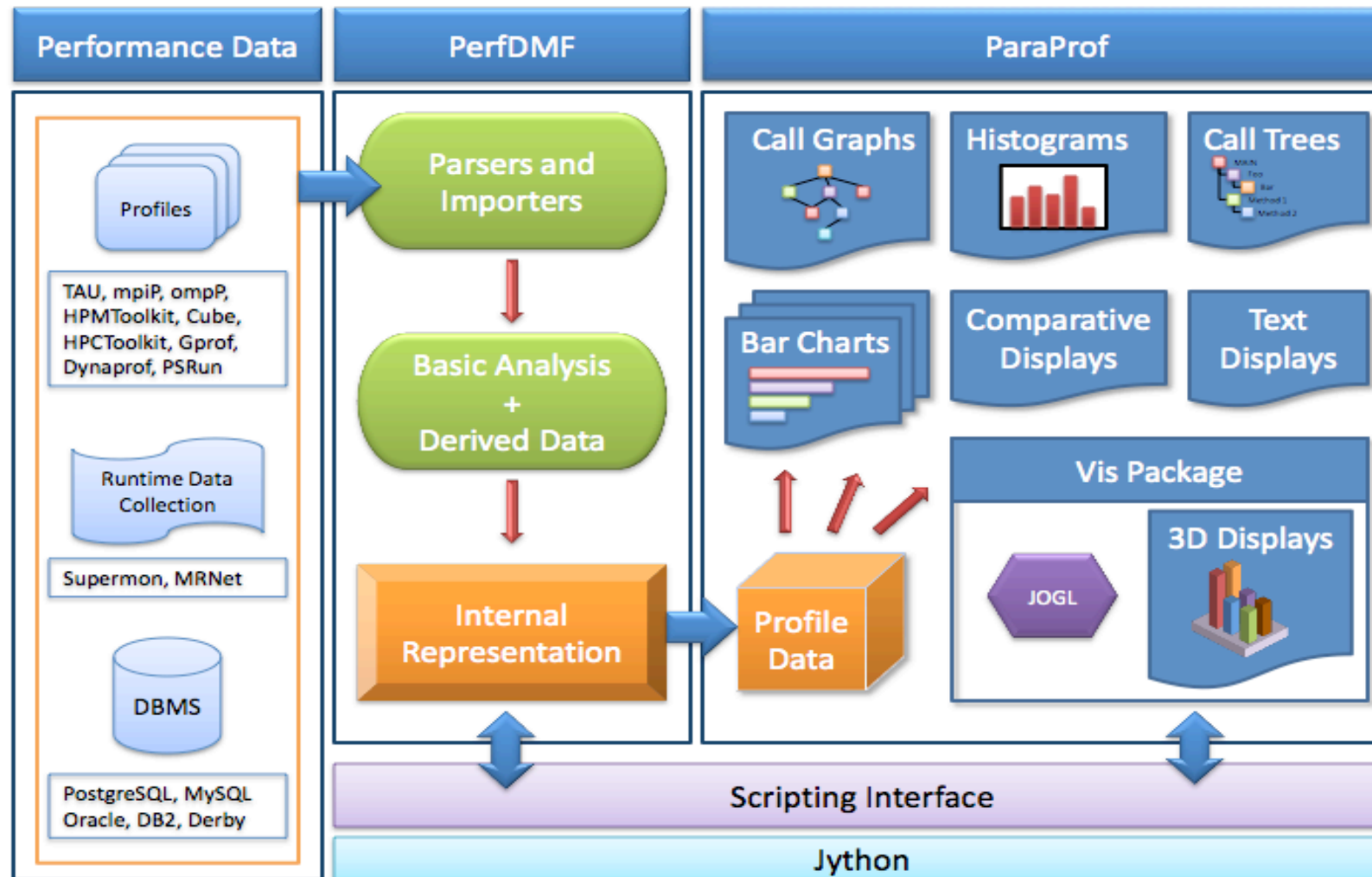
TAU Analysis Tools: paraprof

- Launch paraprof

```
% paraprof
```

Metric

ParaProf Manager Widow: scout.cubex

Paraprof main window

Colors represent code regions

Options -> uncheck Stack Bars Together

Paraprof main window
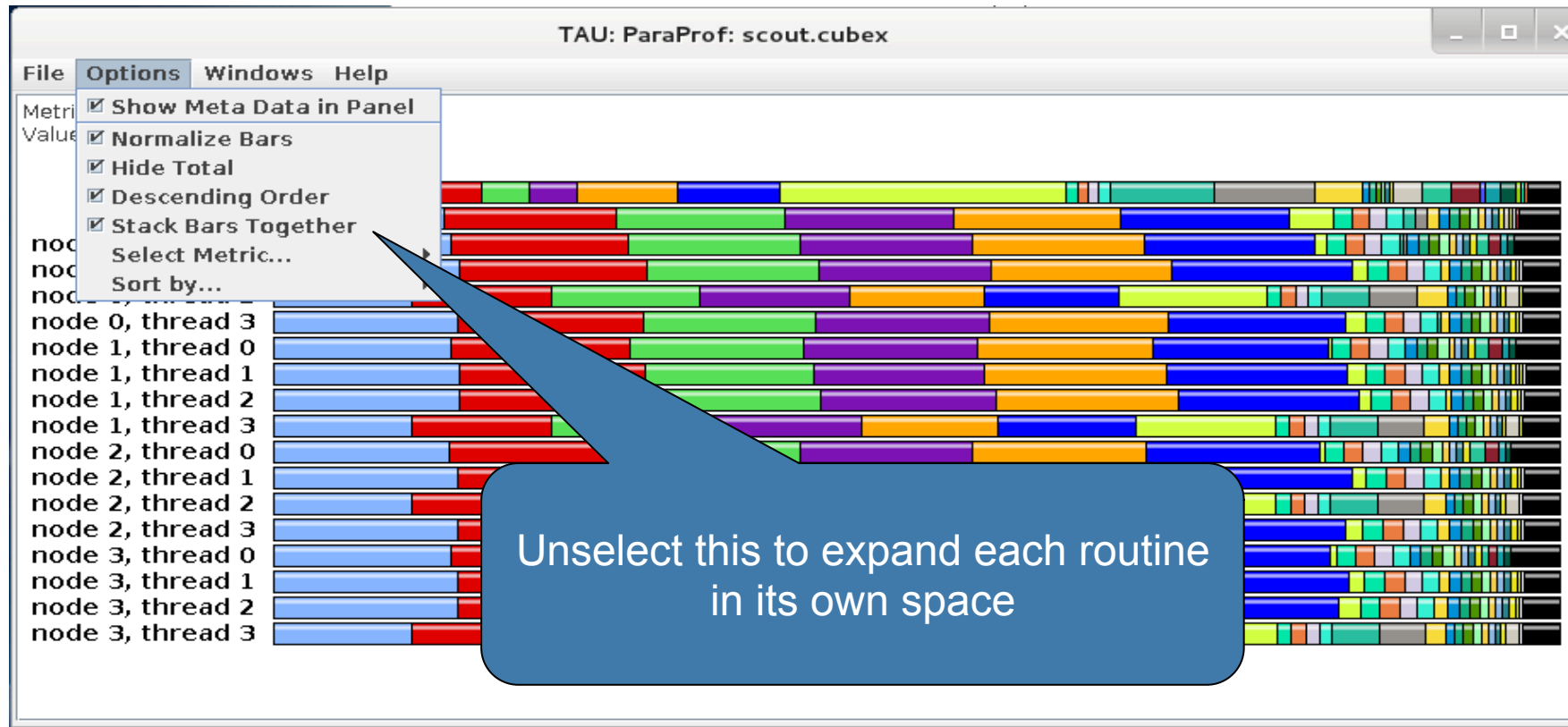
ParaProf Profile Browser
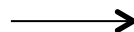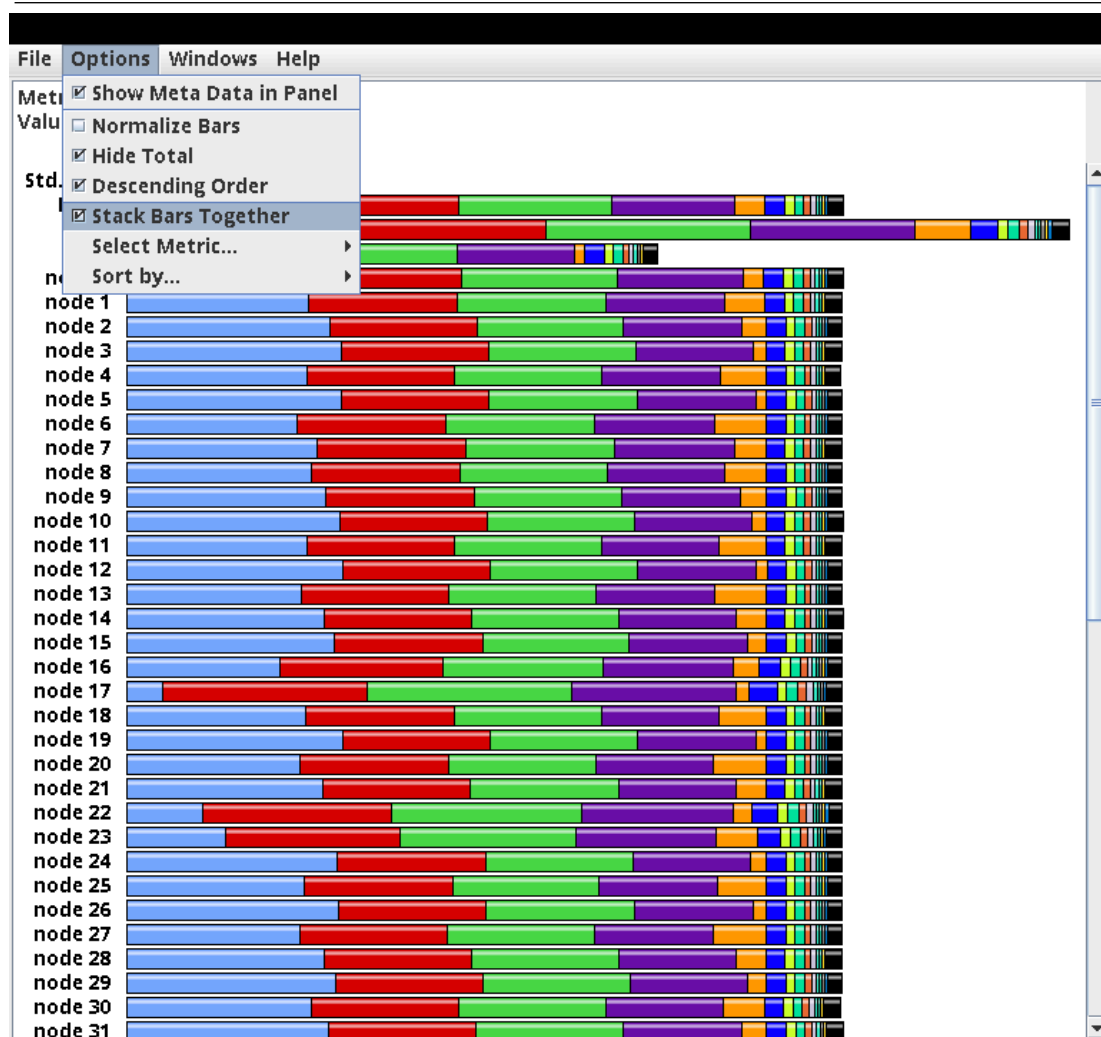
Paraprof main window
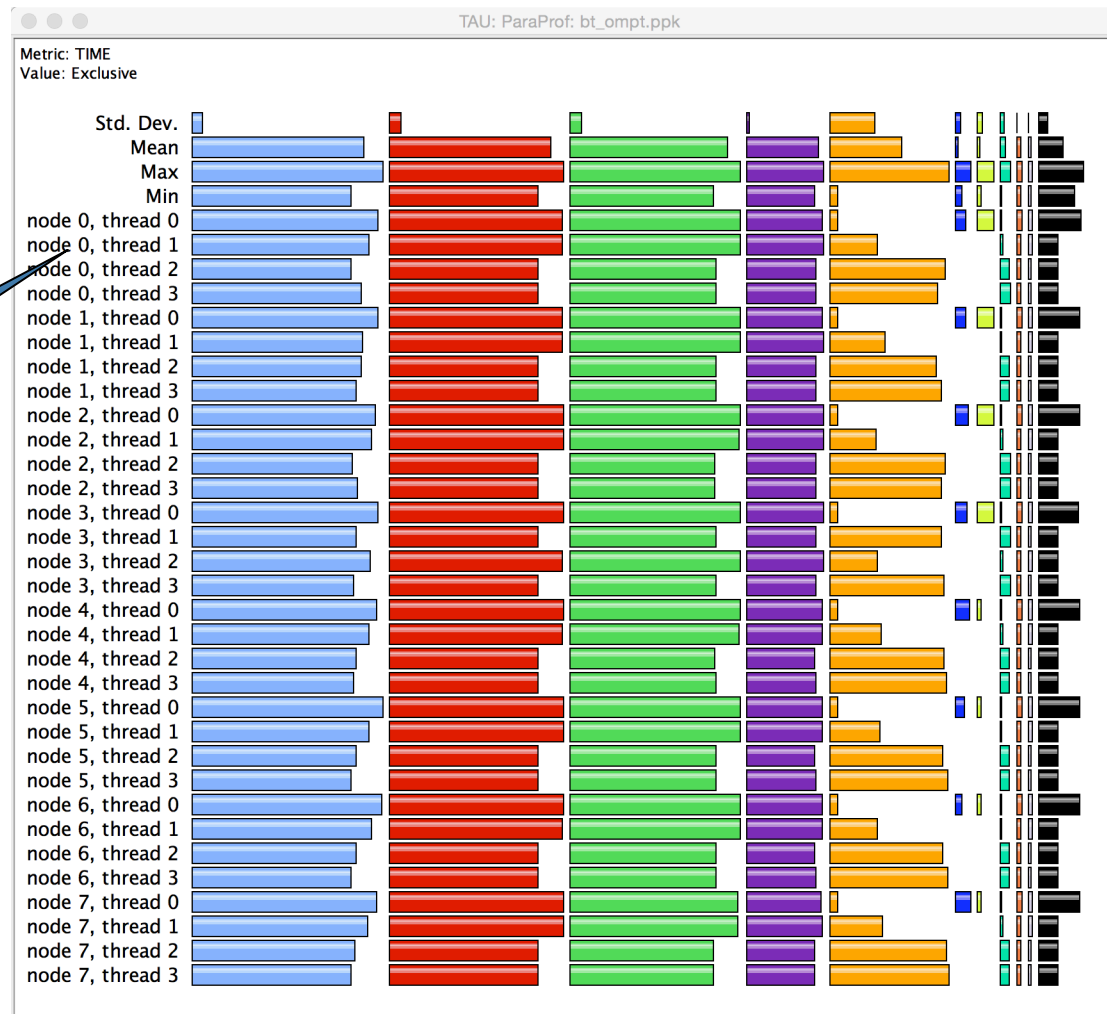
Paraprof node window (function barchart window)
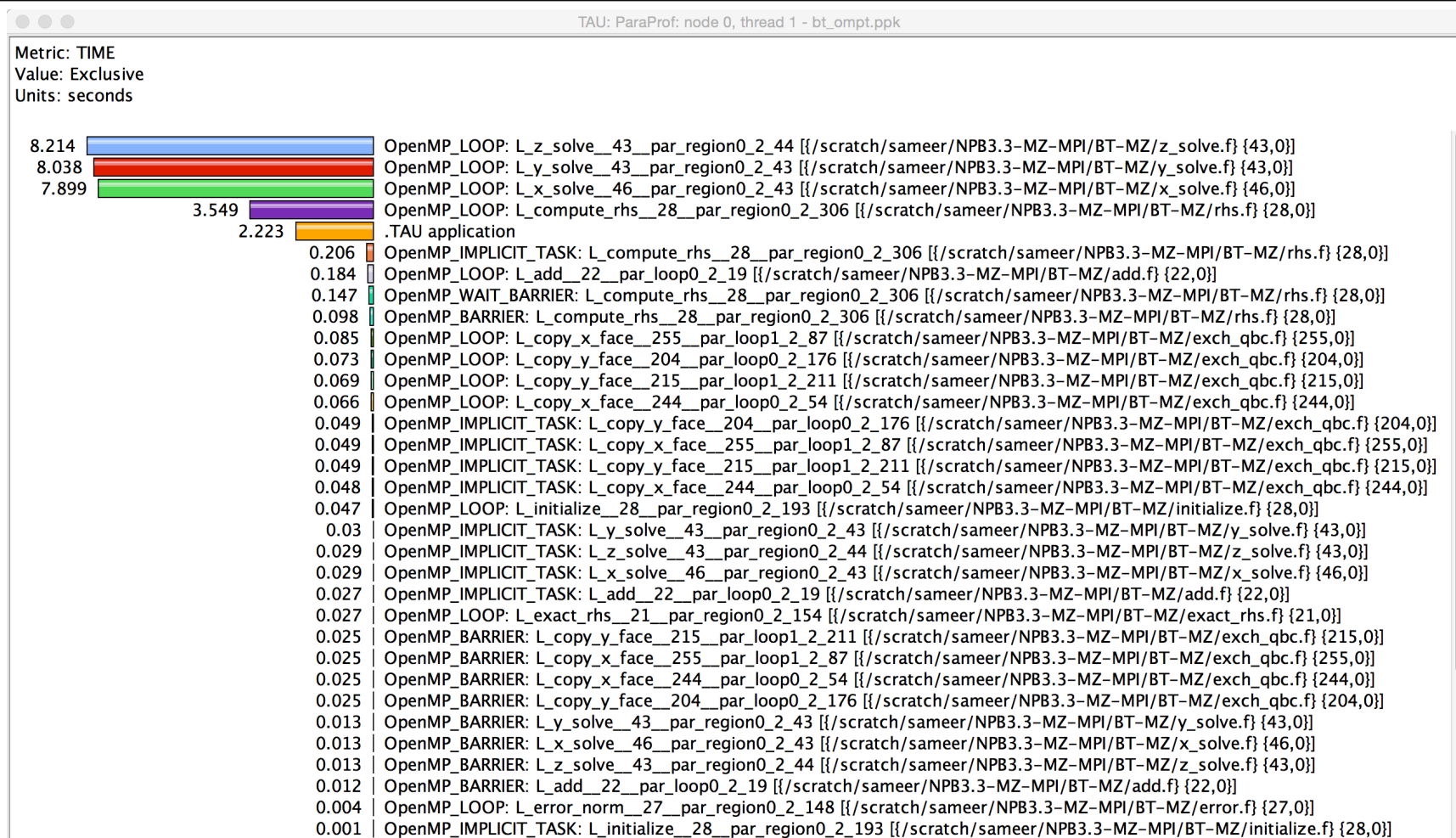
Exclusive time spent in each code region (OpenMP loop) is shown here for MPI rank 0 thread 1

Instrumenting Source Code with PDT and Opari
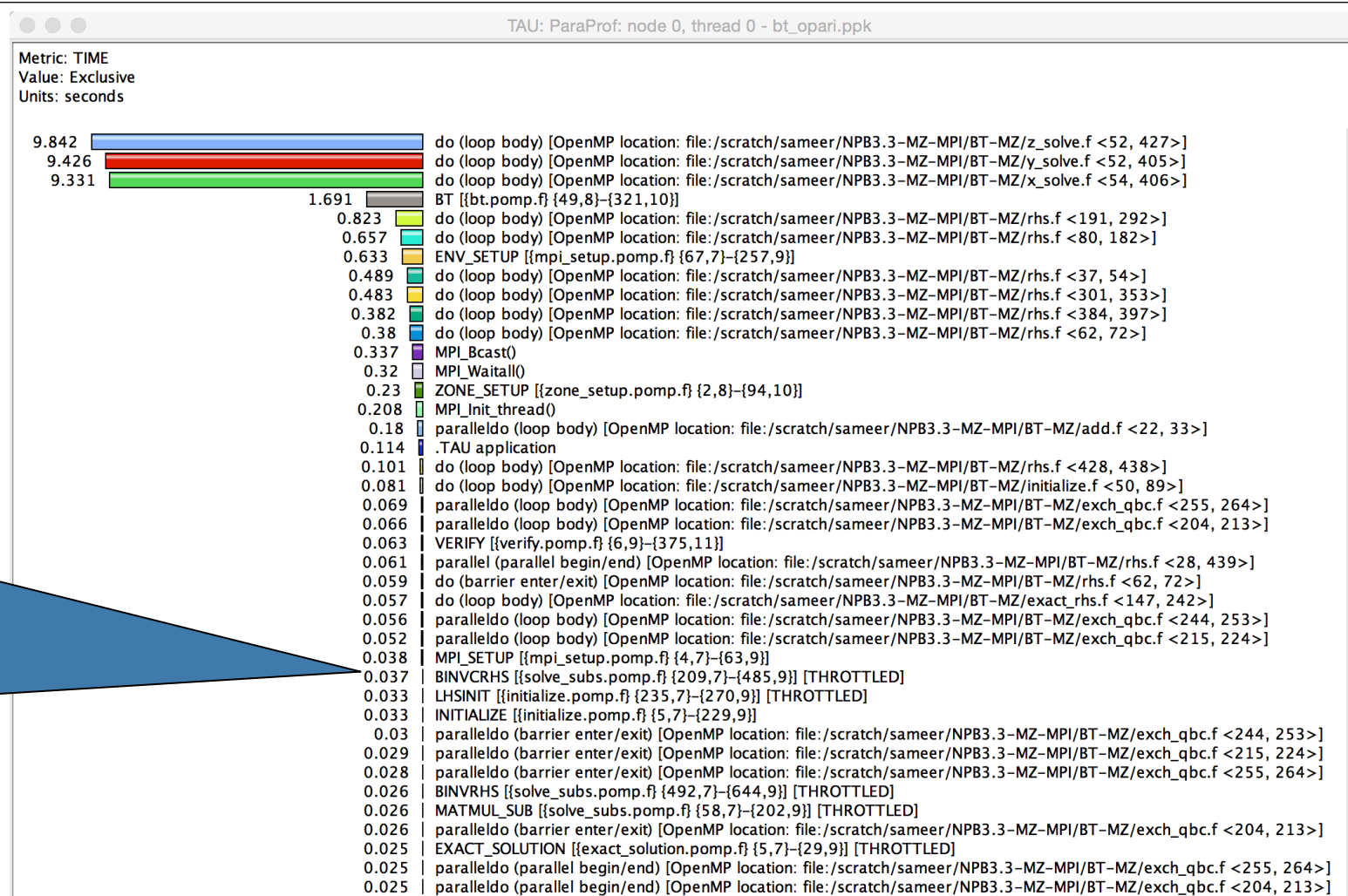
ParaProf: Node view in a callpath profile
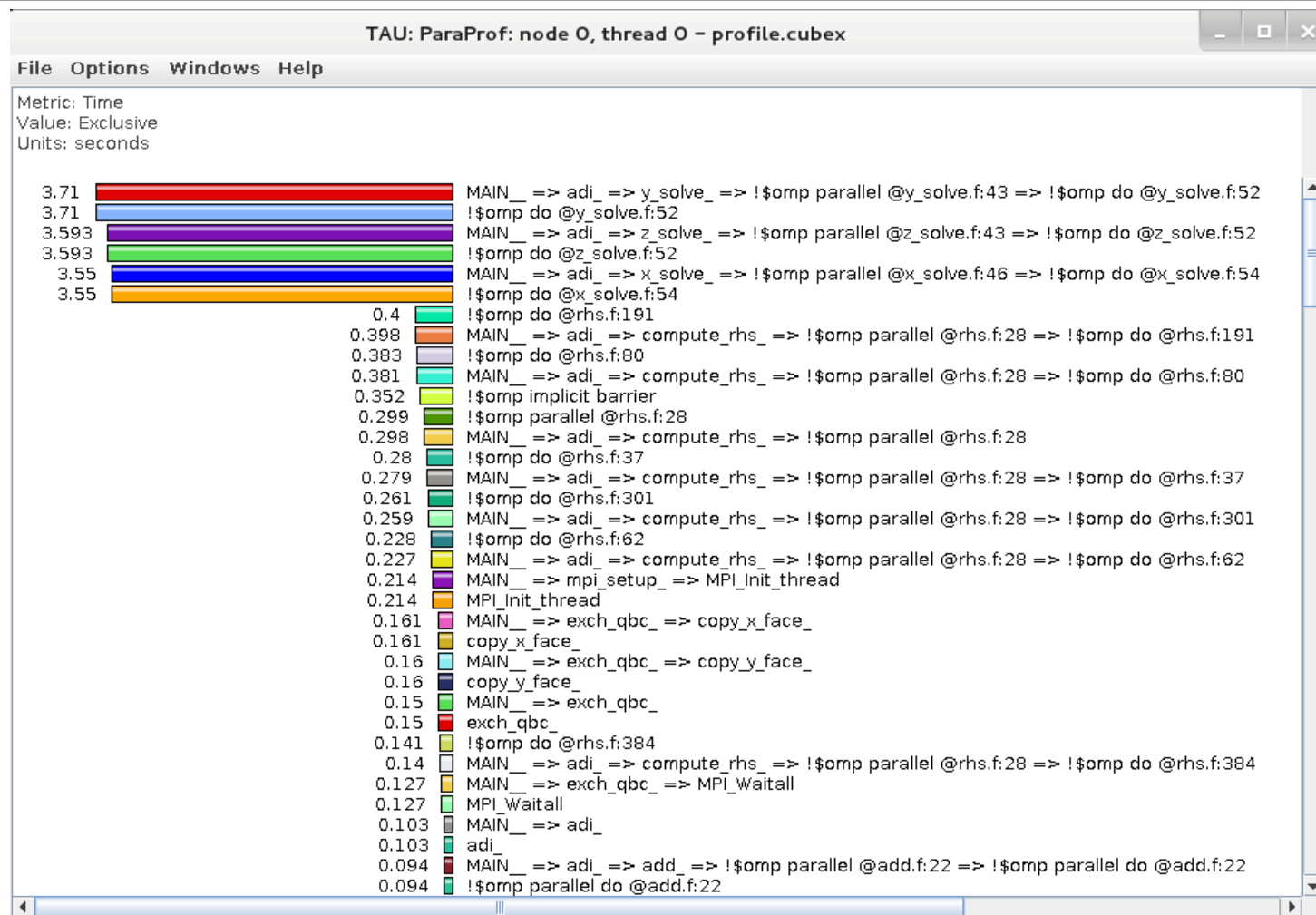
ParaProf: Add thread to comparison window
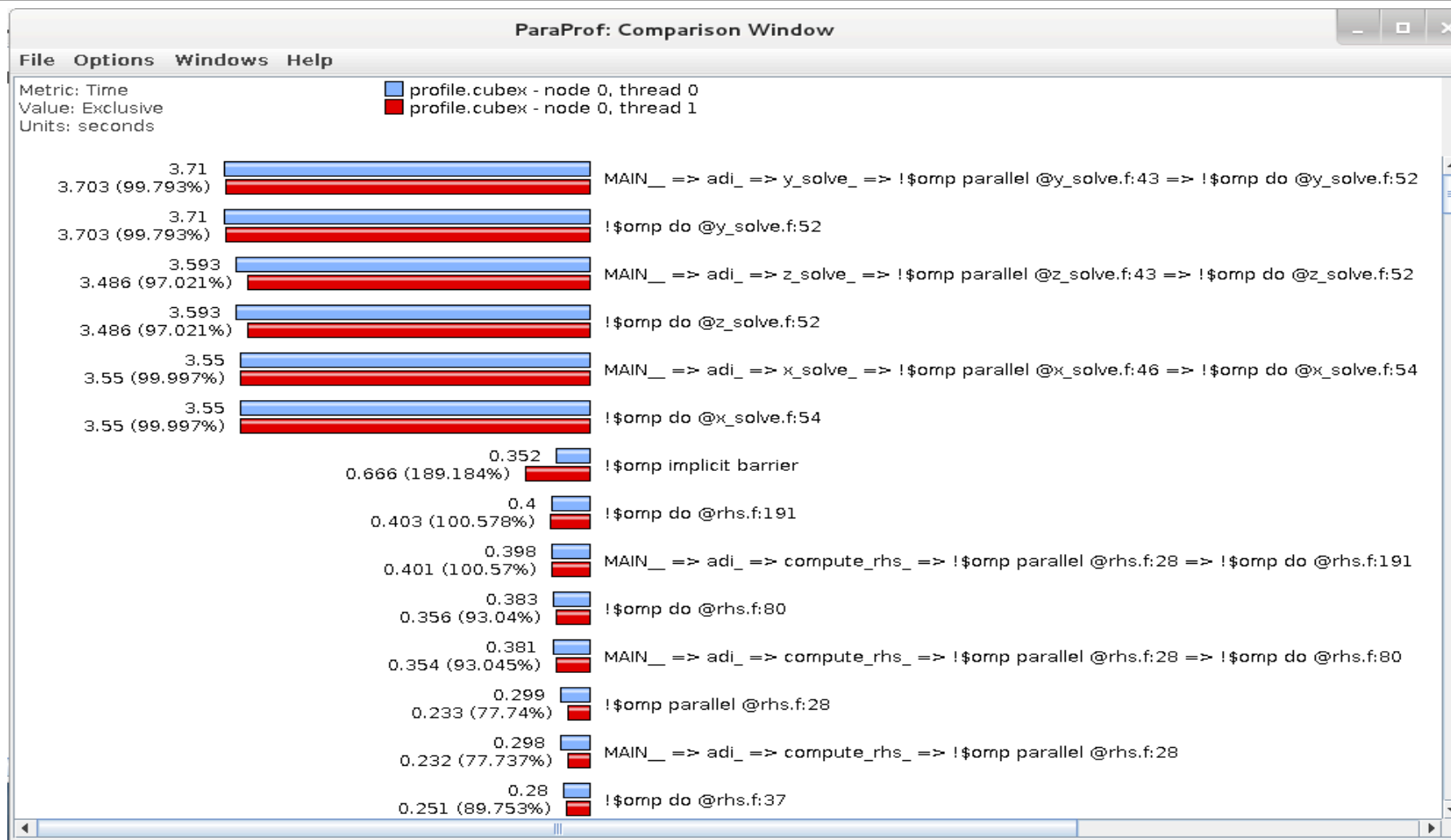
Paraprof Thread Statistics Table with TAU_SAMPLING=1
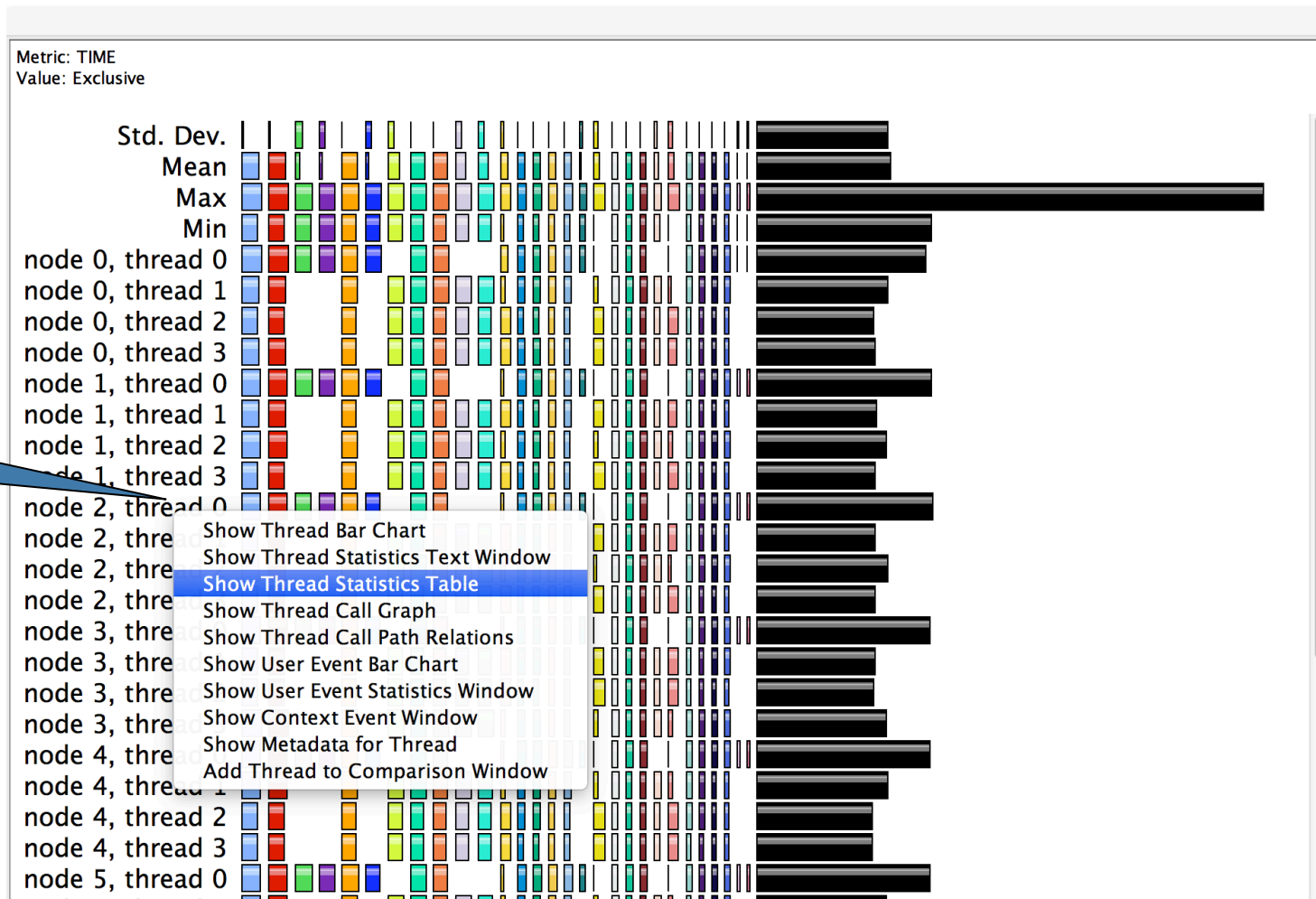
ParaProf: Thread Statistics Table

TAU: ParaProf: Statistics for: node 0, thread 0 – scout.cubex

File   Options   Windows   Help

Time

Name	Exclusive Time ▽	Inclusive Time	Calls	Child Calls
!$omp do @y_solve.f:52	5.817	5.817	3,216	0
!$omp do @z_solve.f:52	5.657	5.657	3,216	0
!$omp do @x_solve.f:54	5.609	5.609	3,216	0
!$omp do @rhs.f:191	0.609		3,232	0
!$omp do @rhs.f:80	0.583		3,232	0
MPI_Waitall	0.402		603	0
!$omp implicit barrier	0.402			
!$omp do @rhs.f:301	0.36			
!$omp implicit barrier	0.026			
!$omp implicit barrier	0			
!$omp do @rhs.f:37	0.343			
!$omp do @rhs.f:62	0.225			
!$omp implicit barrier	0.004	0.004	3,216	0
!$omp implicit barrier	0	0	16	0
MPI_Init_thread	0.218	0.218	1	0
!$omp do @rhs.f:384	0.199	0.199	3,232	0
!$omp parallel do @add.f:22	0.099	0.111	3,216	3,216
!$omp do @rhs.f:428	0.069	0.069	3,232	0
MPI_Isend	0.043	0.043	603	0
!$omp do @initialize.f:50	0.04	0.04	32	0
!$omp parallel @rhs.f:28	0.03	2.536	3,232	51,712
!$omp parallel do @exch_qbc.f:215	0.021	0.029	6,432	6,432
!$omp parallel do @exch_qbc.f:255	0.02	0.033	6,432	6,432
!$omp parallel @exch_qbc.f:255	0.02	0.053	6,432	6,432
!$omp parallel @exch_qbc.f:244				

FinderScreenSnapz003.png
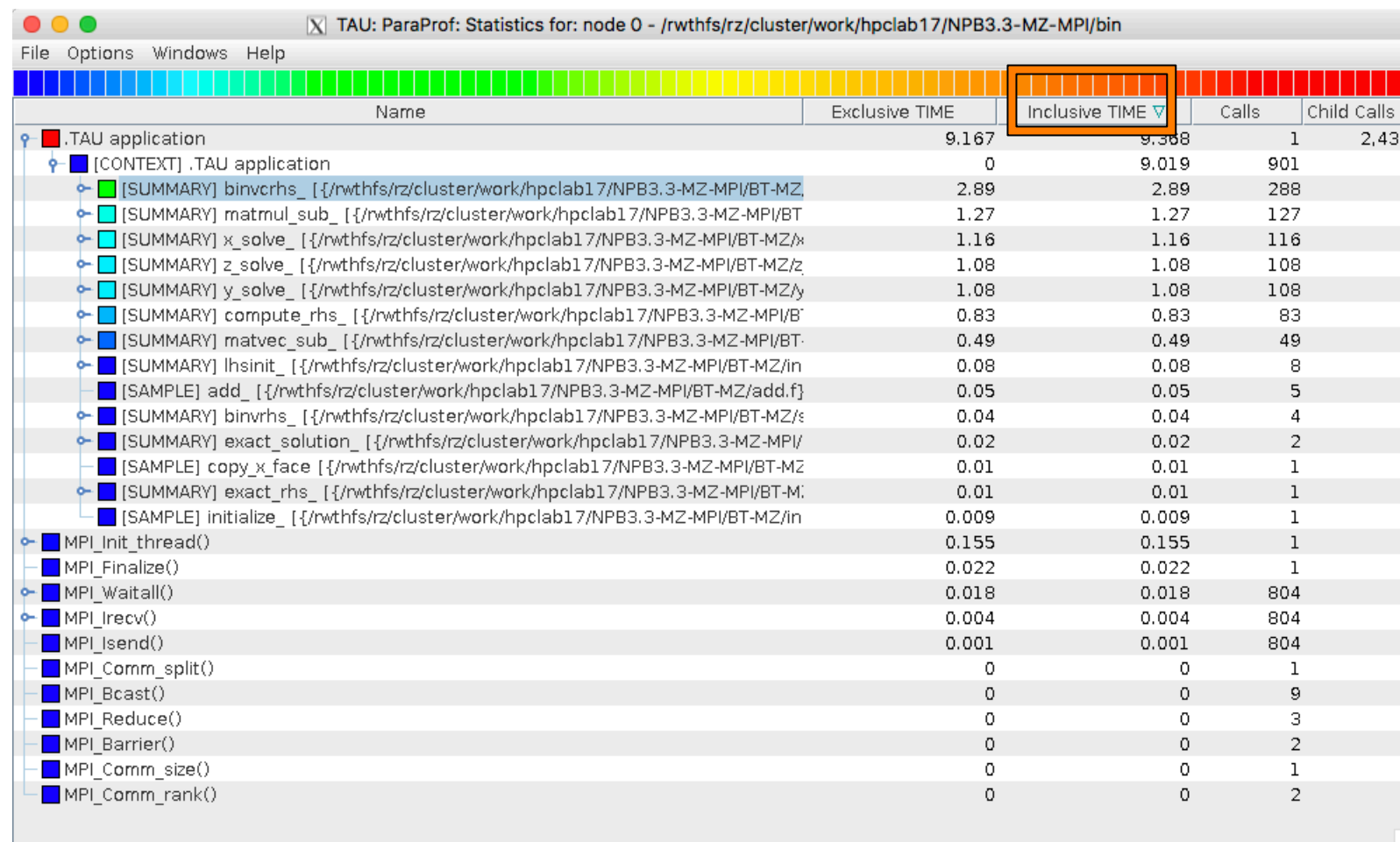
> Click to sort by a given metric, drag and move to rearrange columns

ParaProf

- Click on Columns:
- to sort by incl time

- Open binvcrhs
- Click on Sample

Paraprof Thread Statistics Table

ParaProf

Statement Level Profiling with TAU

Source location where samples are taken. Compute intensive region.

ParaProf Comparison Window
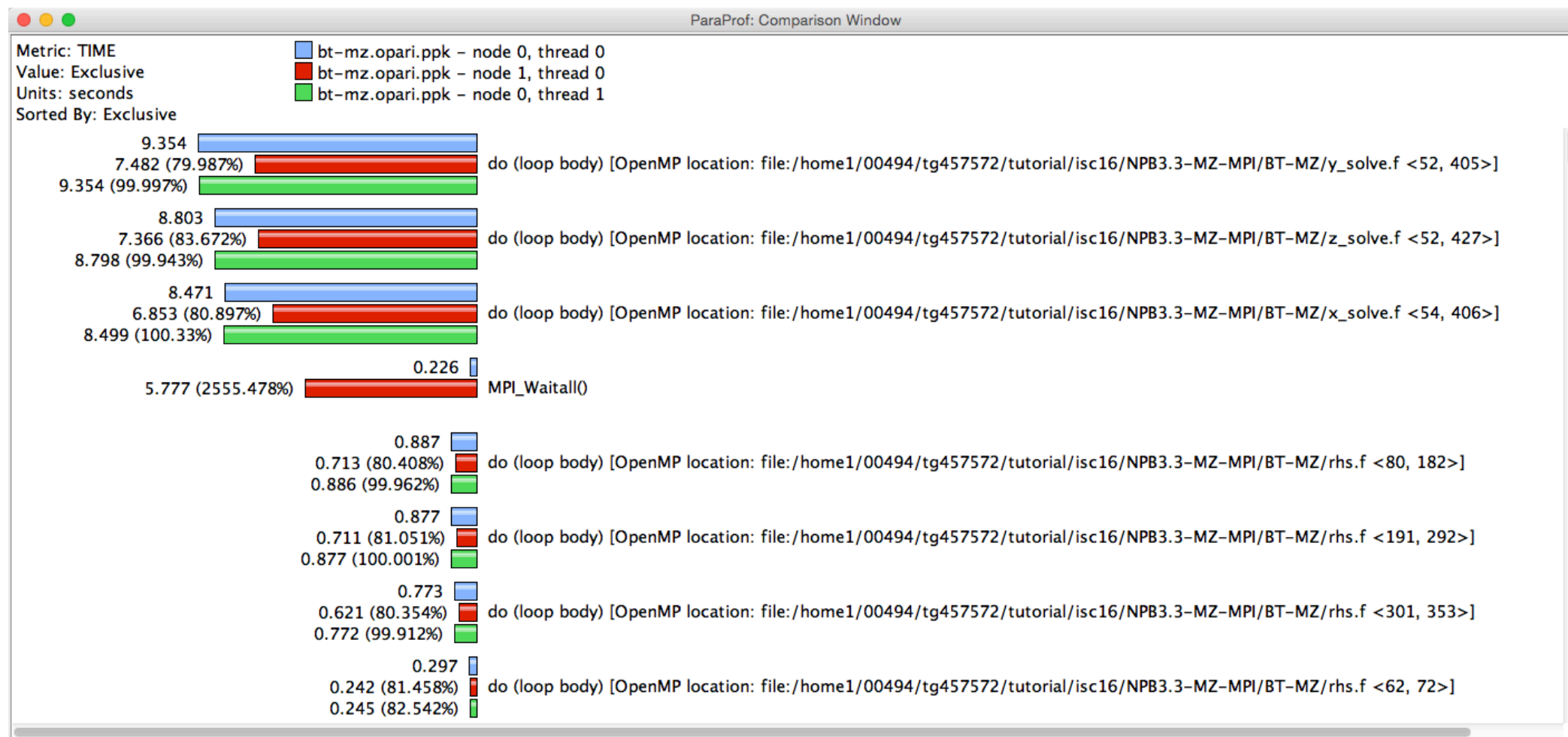
TAU – Event Based Sampling (EBS)
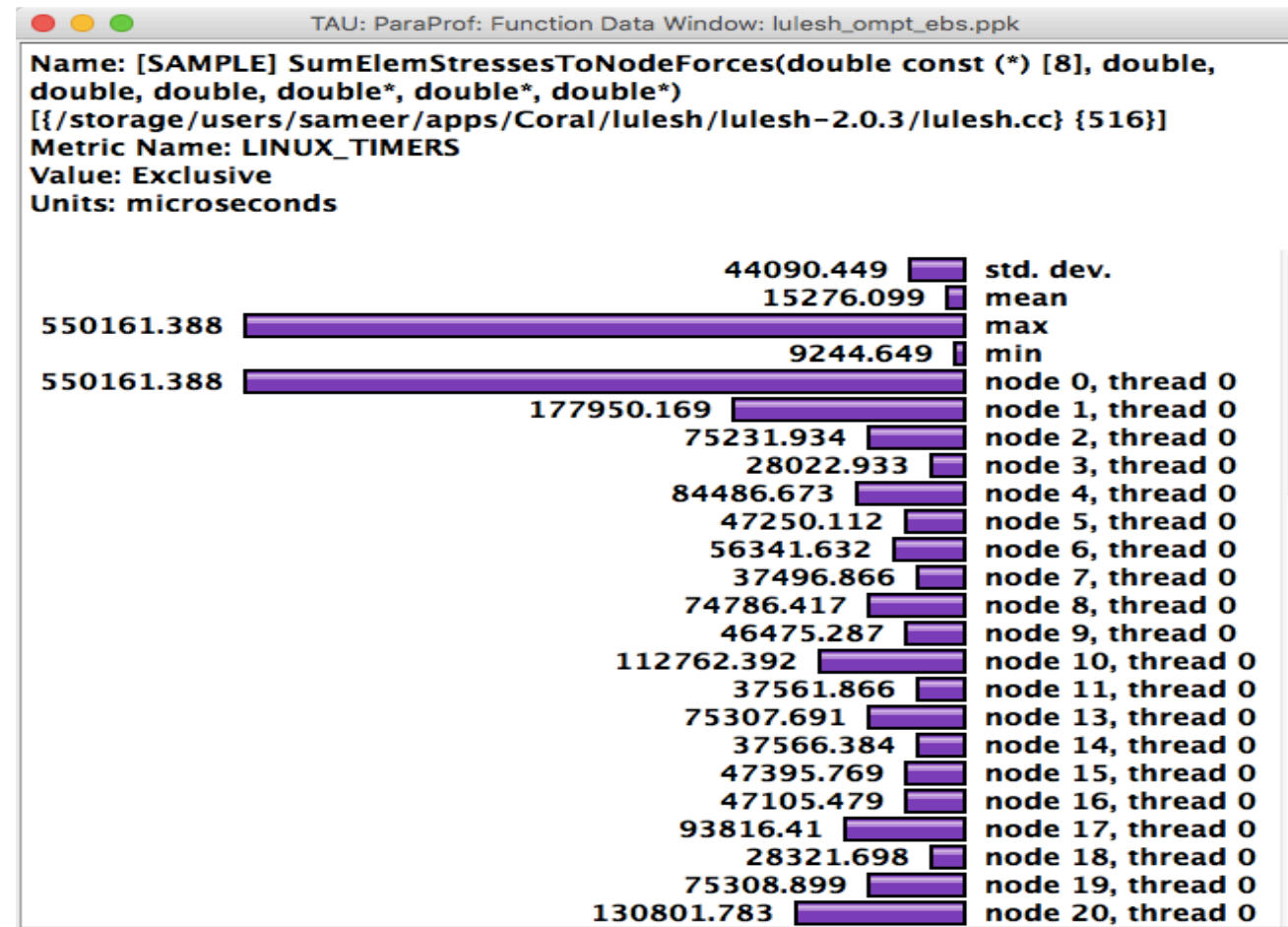
% export TAU_SAMPLING=1

Examples: Callstack Sampling in TAU

% export TAU_SAMPLING=1; export TAU_EBS_UNWIND=1

UNWINDING CALLSTACKS

% export TAU_SAMPLING=1; export TAU_EBS_UNWIND=1

UNWINDING CALLSTACKS

Deep Learning: Tensorflow

% **`tau_python`** **`—ebs`** **`nt3_baseline_keras2.py (CANDLE)`**
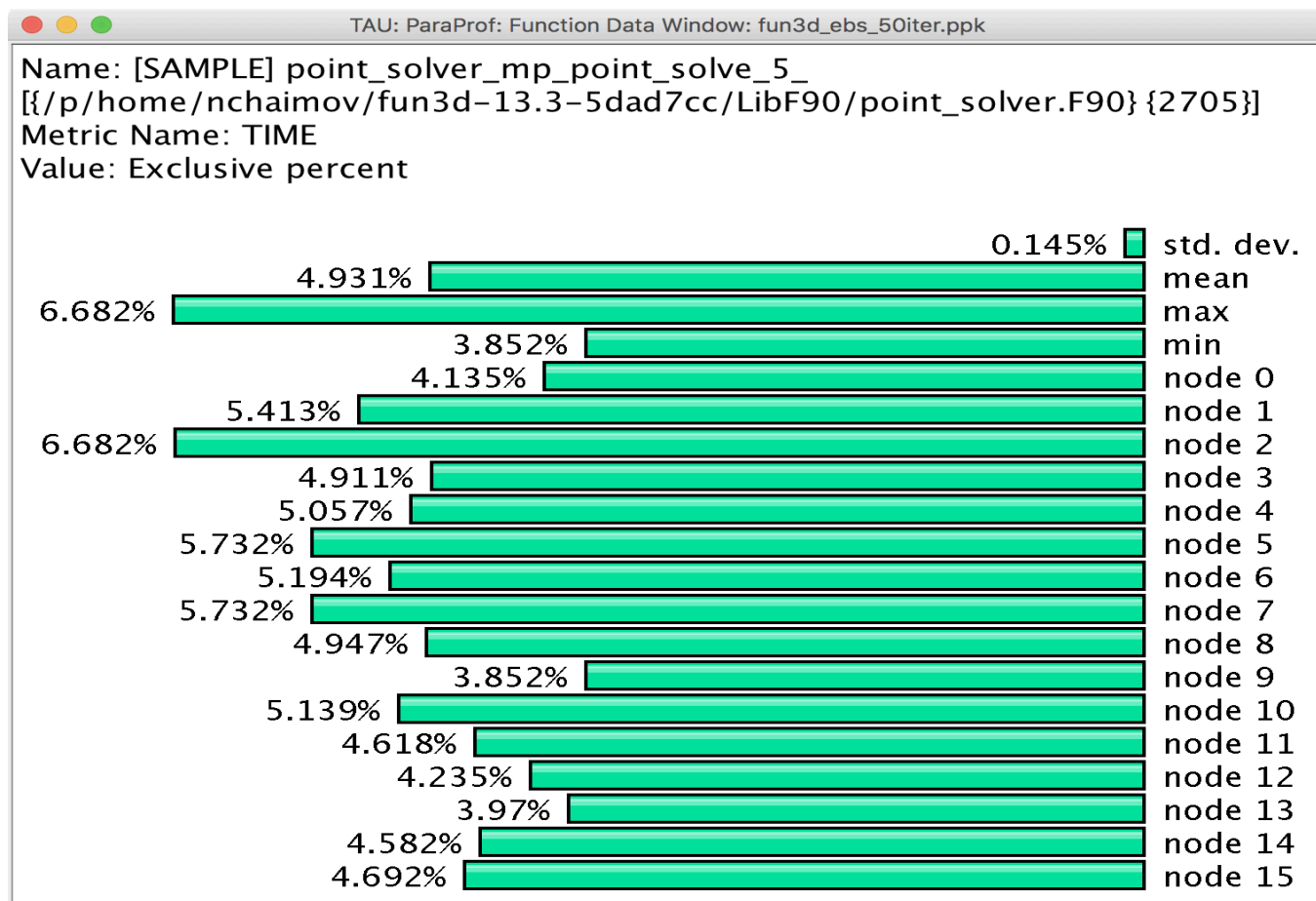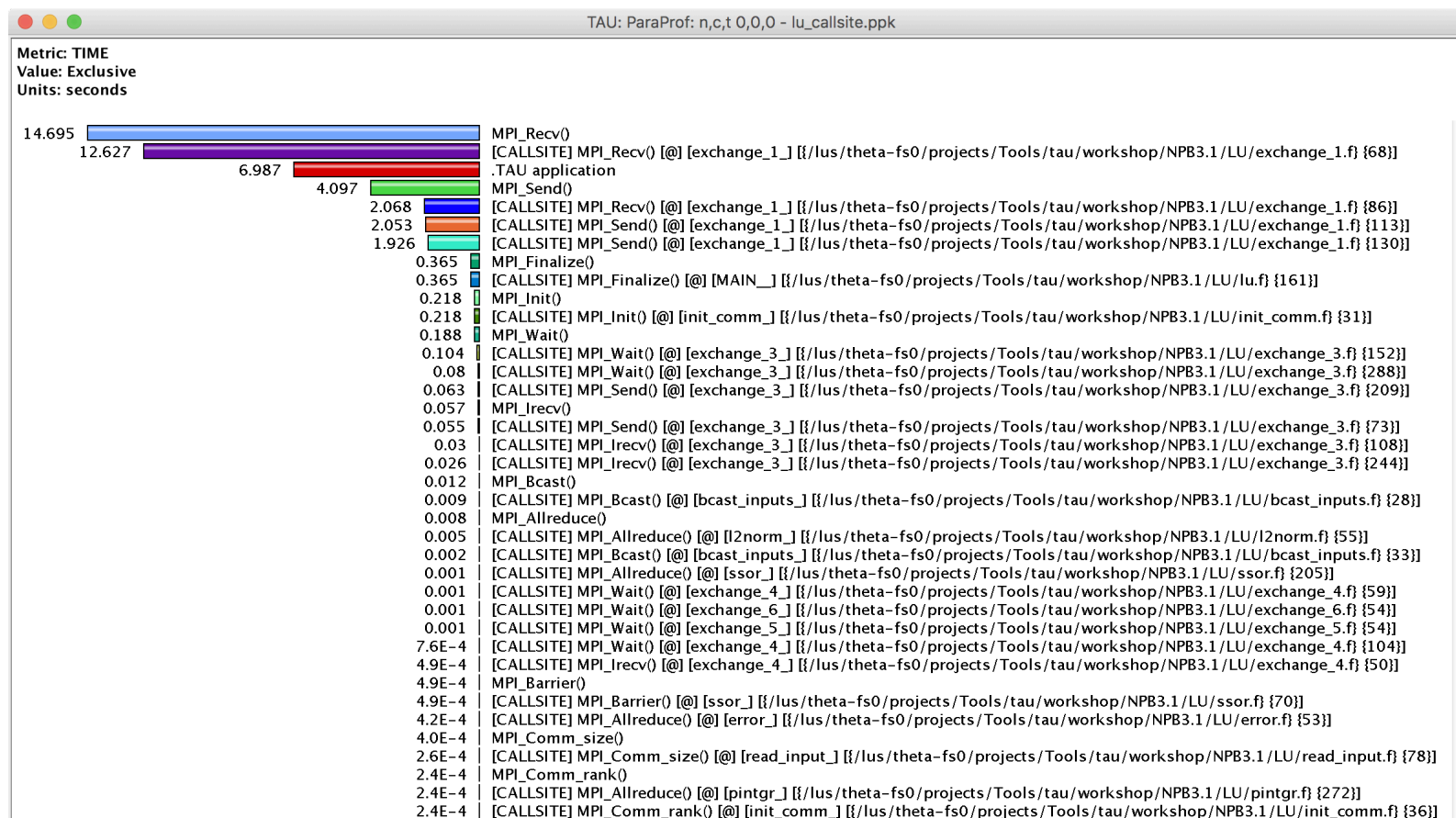
Sampling Tensorflow
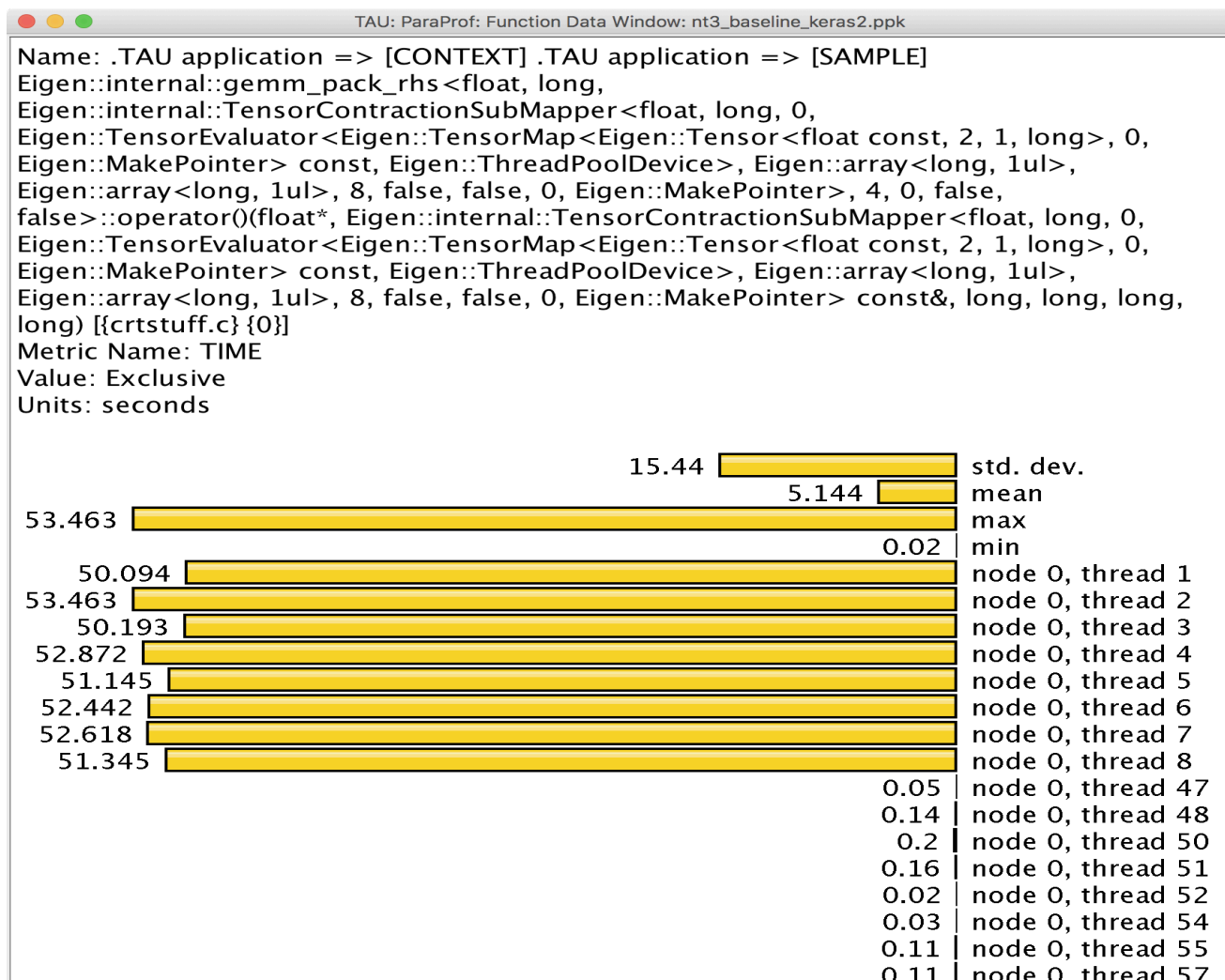
Event Based Sampling (EBS)

Uninstrumented!

```
% aprun –n 16 tau_exec —ebs a.out
```

Callsite Profiling and Tracing

% export TAU_CALLSITE=1

CALLPATH THREAD RELATIONS WINDOW

TAU: ParaProf: Call Path Data n,c,t, 2,0,0 - gamess_unw_call_ebs.ppk

Metric Name: TIME
Sorted By: Inclusive
Units: seconds

```
        Exclusive        Inclusive        Calls/Tot.Calls     Name[id]
        ----------------------------------------------------------------------------
  -->      0.121          79.592           1                   .TAU application
           0.002           0.002           1/1                 MPI_Gather()
           0.004           0.004           3/3                 MPI_Allgather()
           0.122           0.122           6866/6866           MPI_Send()
           0.002           0.002           1/1                 MPI_Comm_split()
           8.9E-5          8.9E-5          2/2                 MPI_Comm_size()
           4.6E-4          4.6E-4          3/3                 MPI_Group_incl()
          75.607          75.607           6870/6870           MPI_Recv()
           0.002           0.002           4/4                 MPI_Comm_create()
           9.5E-5          9.5E-5          6/6                 MPI_Comm_rank()
           5.4E-4          5.4E-4          1/1                 MPI_Comm_group()
           0.003           0.003           7/7                 MPI_Barrier()
           0.112           0.112           1/1                 MPI_Init_thread()
           6.3E-4          6.3E-4          1/1                 MPI_Group_intersection()
           0               0.05            1/1                 [CONTEXT] .TAU application
           3.601           3.601           1/1                 MPI_Finalize()
           0.014           0.014           6/6                 MPI_Bcast()


          75.607          75.607           6870/6870           .TAU application
  -->     75.607          75.607           6870                MPI_Recv()
           0              74.848           1497/1497           [CONTEXT] MPI_Recv()


           0              74.848           1497/1497           MPI_Recv()
  -->      0              74.848           1497                [CONTEXT] MPI_Recv()
           0               8.701           174/1371            [UNWIND] /gpfs/mira-home/yuri/dist/Github/gamess-theta-tau/ddi/src/ddi_init.c.113 [@] ddi_i
           0              26.196           524/763             [UNWIND] /gpfs/mira-home/sameer/gamess-theta-tau/object/unport.f.410 [@] MAIN__ [{/gpfs/mira
           0.2             0.2             4/138               [SAMPLE] GNII_DlaProgress [{/opt/cray/ugni/6.0.14-6.0.4.0_14.1__ge7db4a2.ari/lib64/libugni.
           0               5.75            115/1484            [UNWIND] /gpfs/mira-home/yuri/dist/Github/gamess-theta-tau/ddi/src/ddi_server.c.99 [@] DDI_
           0               0.2             4/5                 [UNWIND] /lib64/libc-2.22.so.0 [@] _start [{/home/abuild/rpmbuild/BUILD/glibc-2.22/csu/../sy
           0              11.85            237/239             [UNWIND] /gpfs/mira-home/sameer/gamess-theta-tau/object/gamess.f.538 [@] main [{/gpfs/mira-l
           0.051           0.051           1/273               [SAMPLE] GNI_CqGetEvent [{/opt/cray/ugni/6.0.14-6.0.4.0_14.1__ge7db4a2.ari/lib64/libugni.so
           0               0.05            1/1197              [UNWIND] /opt/cray/pe/mpt/7.6.3/gni/mpich-intel/16.0/lib/libmpich_intel.so.3.0.1.0 [@] MPID
           0               0.15            3/7                 [UNWIND] [/opt/cray/ugni/6.0.14-6.0.4.0_14.1__ge7db4a2.ari/lib64/libugni.so.0.6.0.0] [@] UN
           0              21.7             434/1197            [UNWIND] /gpfs/mira-home/yuri/dist/Github/gamess-theta-tau/ddi/src/ddi_fortran.c.67 [@] beg
```

CALLPATH THREAD RELATIONS WINDOW

ParaProf: Callpath Thread Relations Window
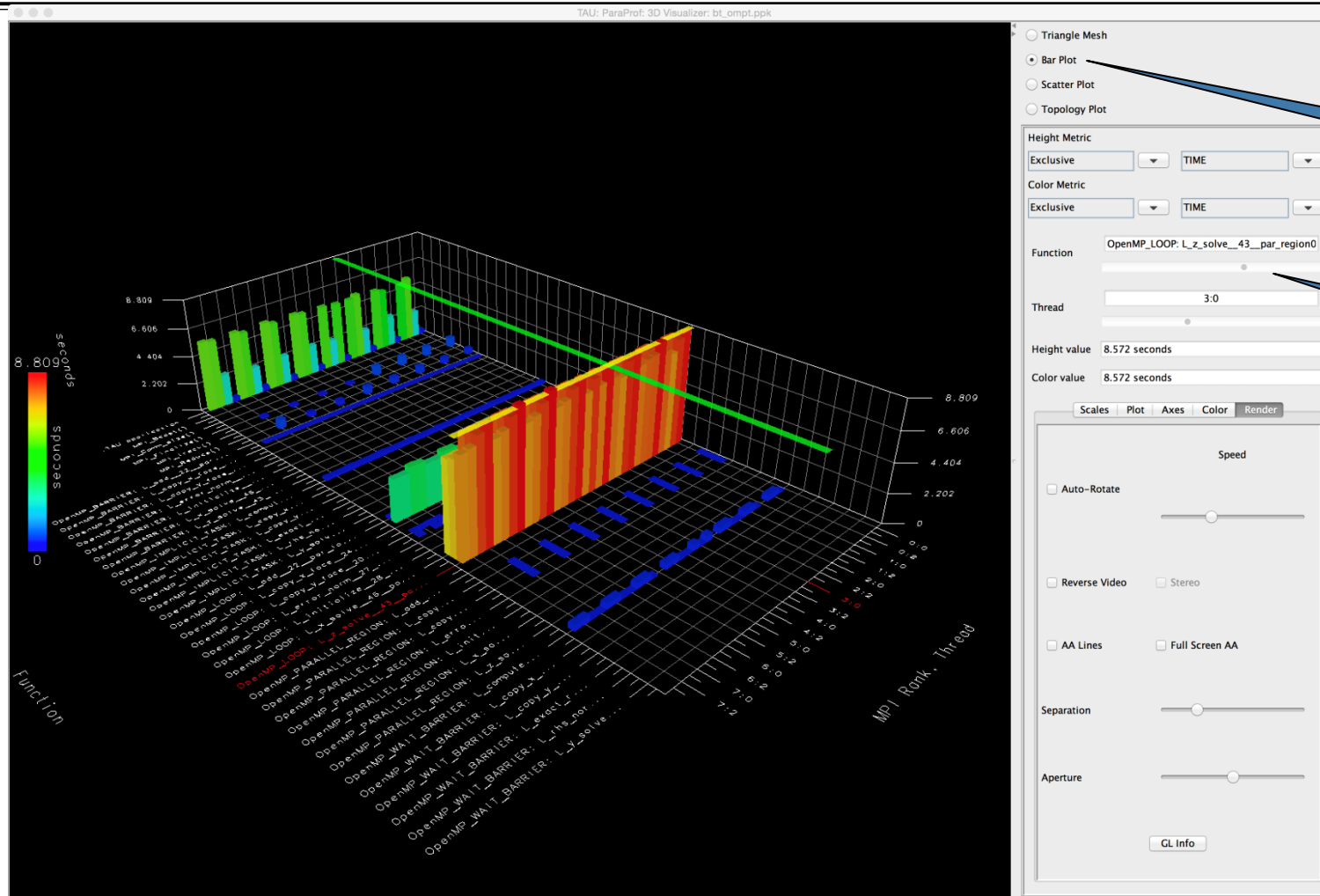
Callsite Profiling and Tracing (TAU_CALLSITE=1)

TAU – Context Events

ParaProf with Optimized Instrumentation
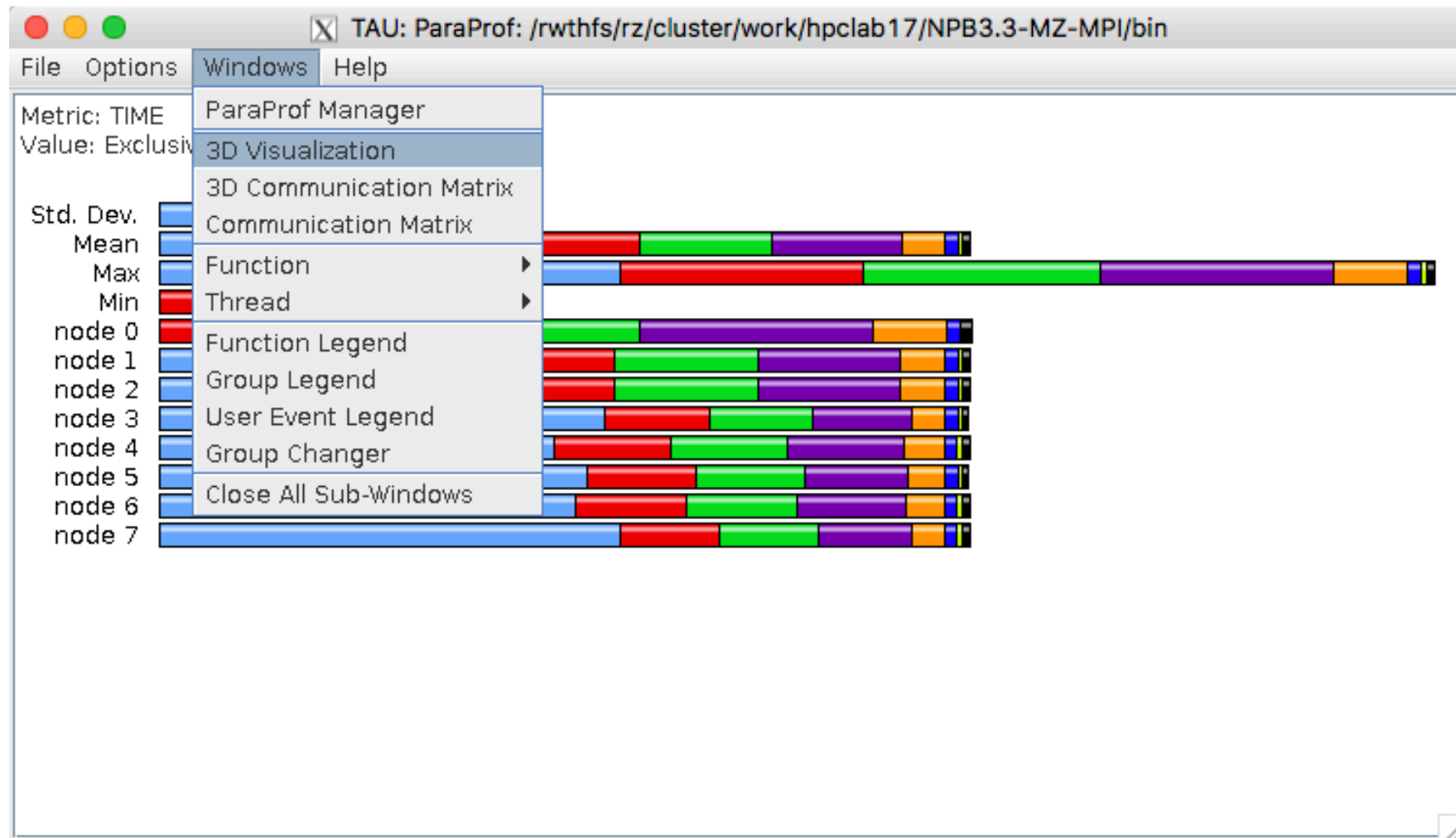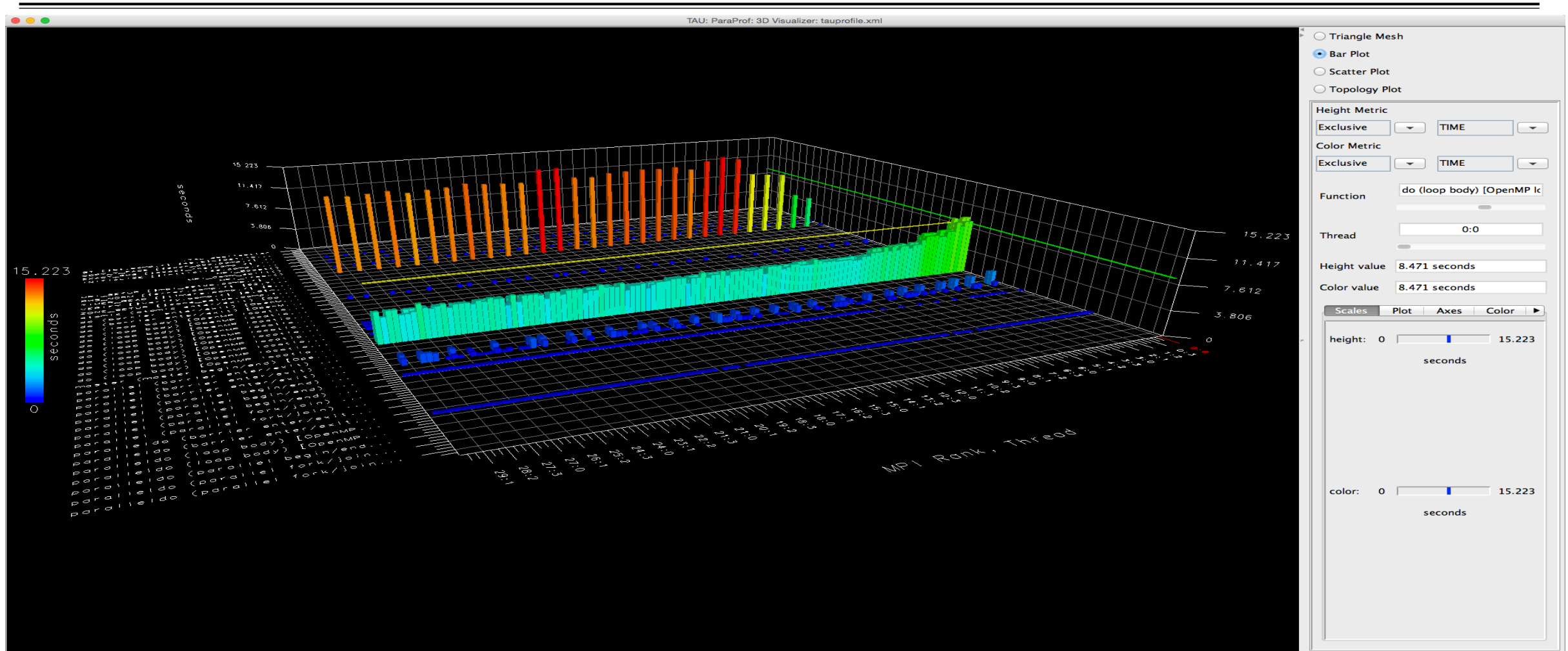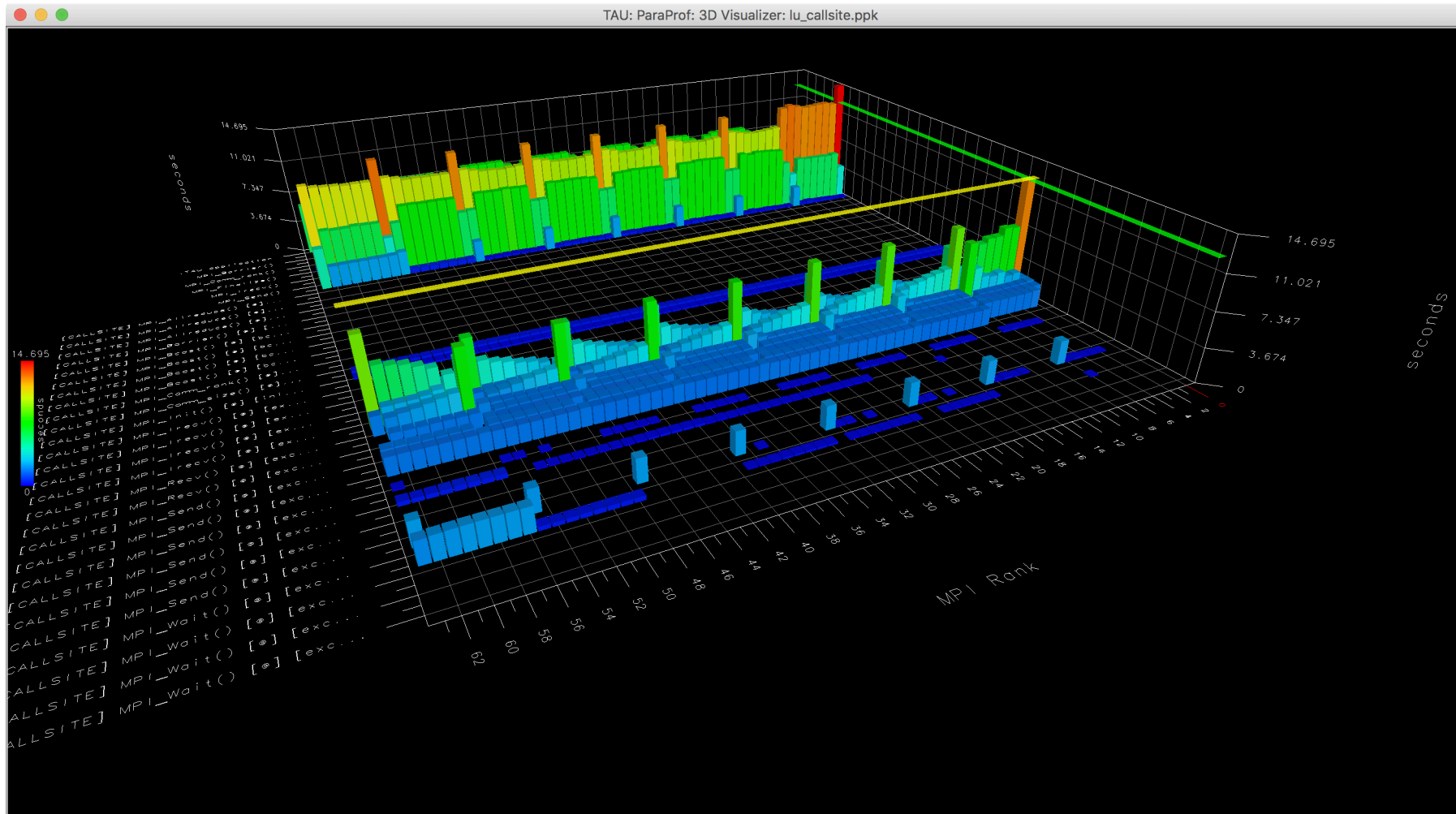
Paraprof 3D visualization window

Click Bar Plot

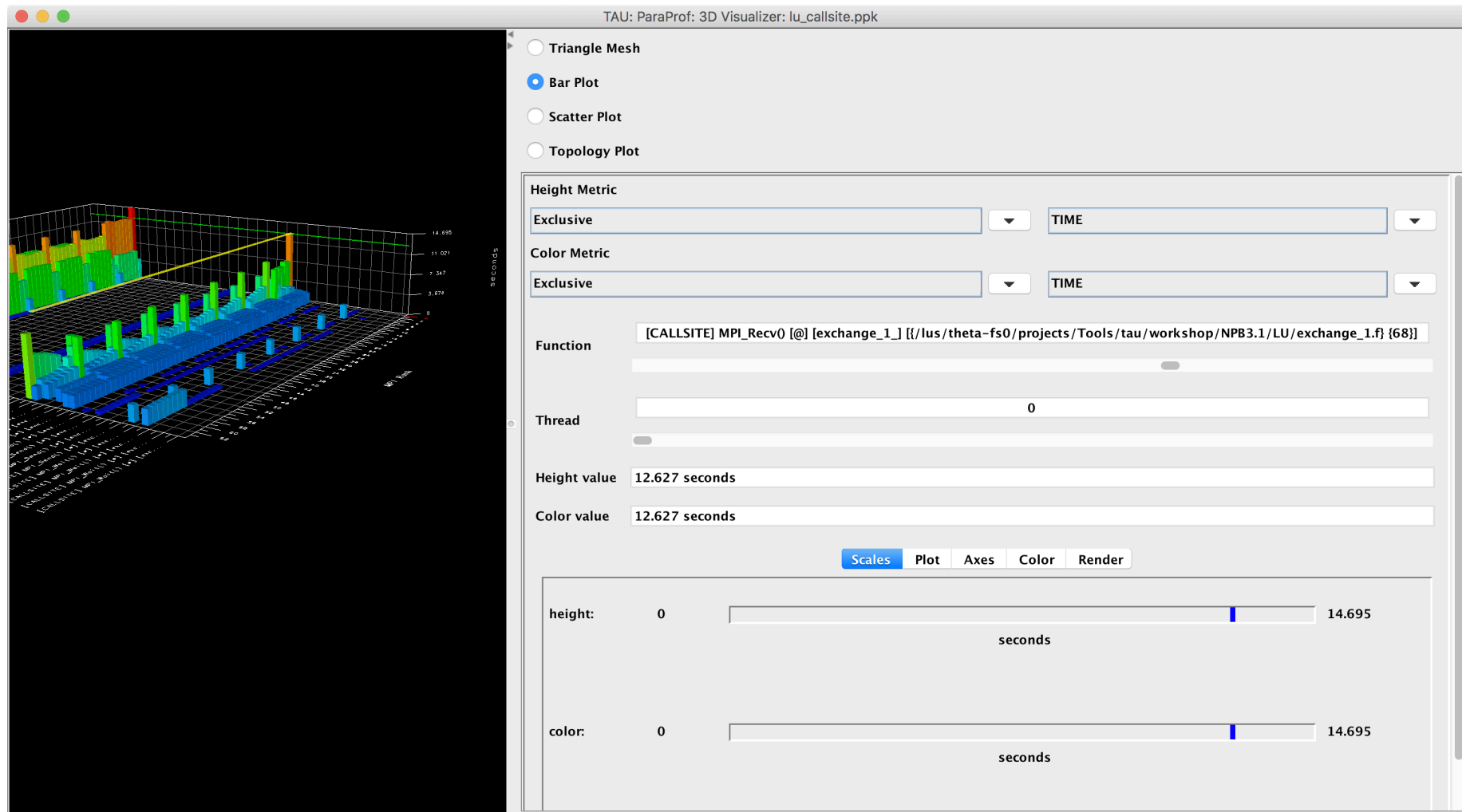Move Function and Thread Sliders

Windows ->
3D visualization

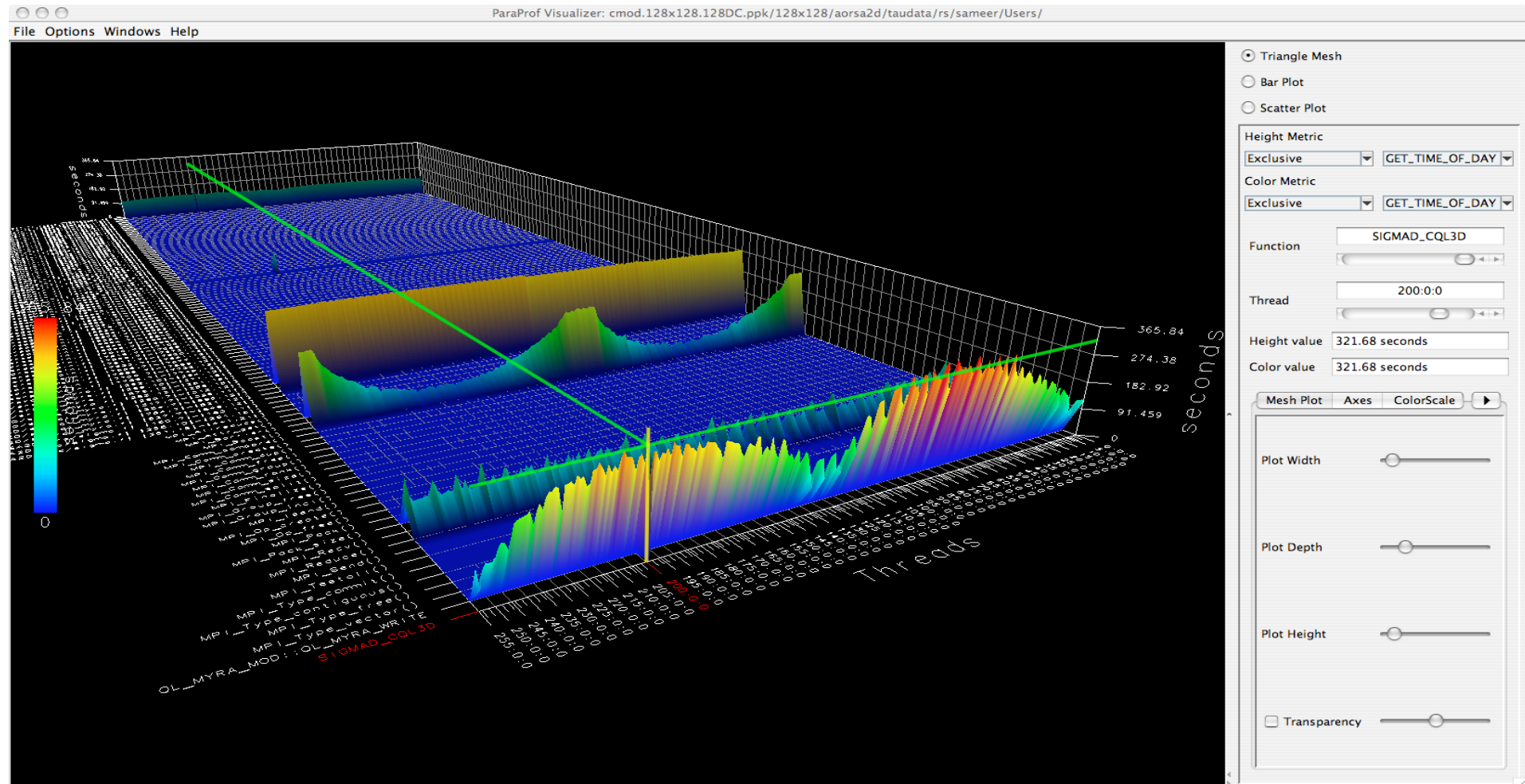# ParaProf: 3D Visualization Window Showing Entire Profile

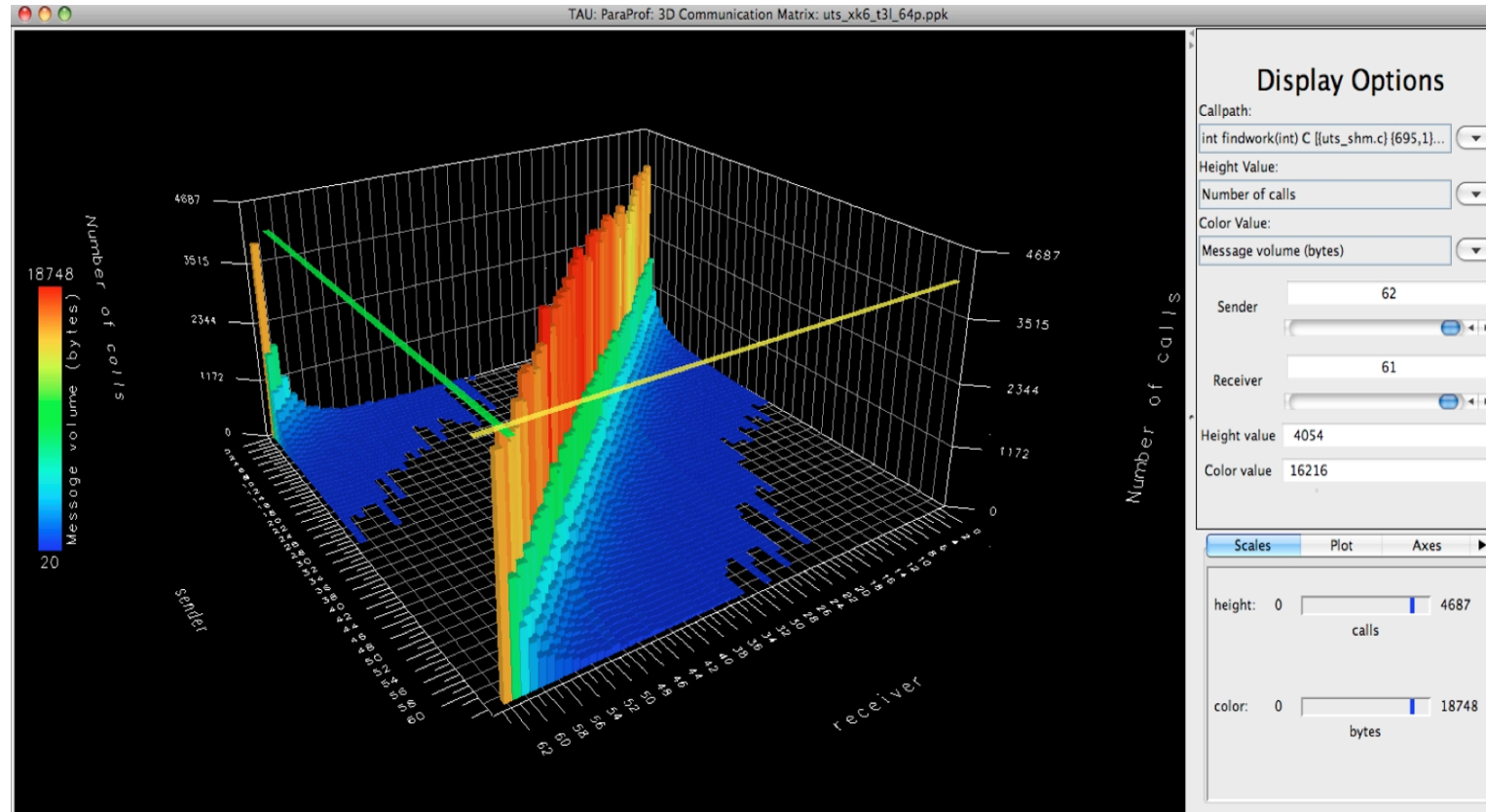# Callsite Profiling and Tracing

Callsite Profiling and Tracing
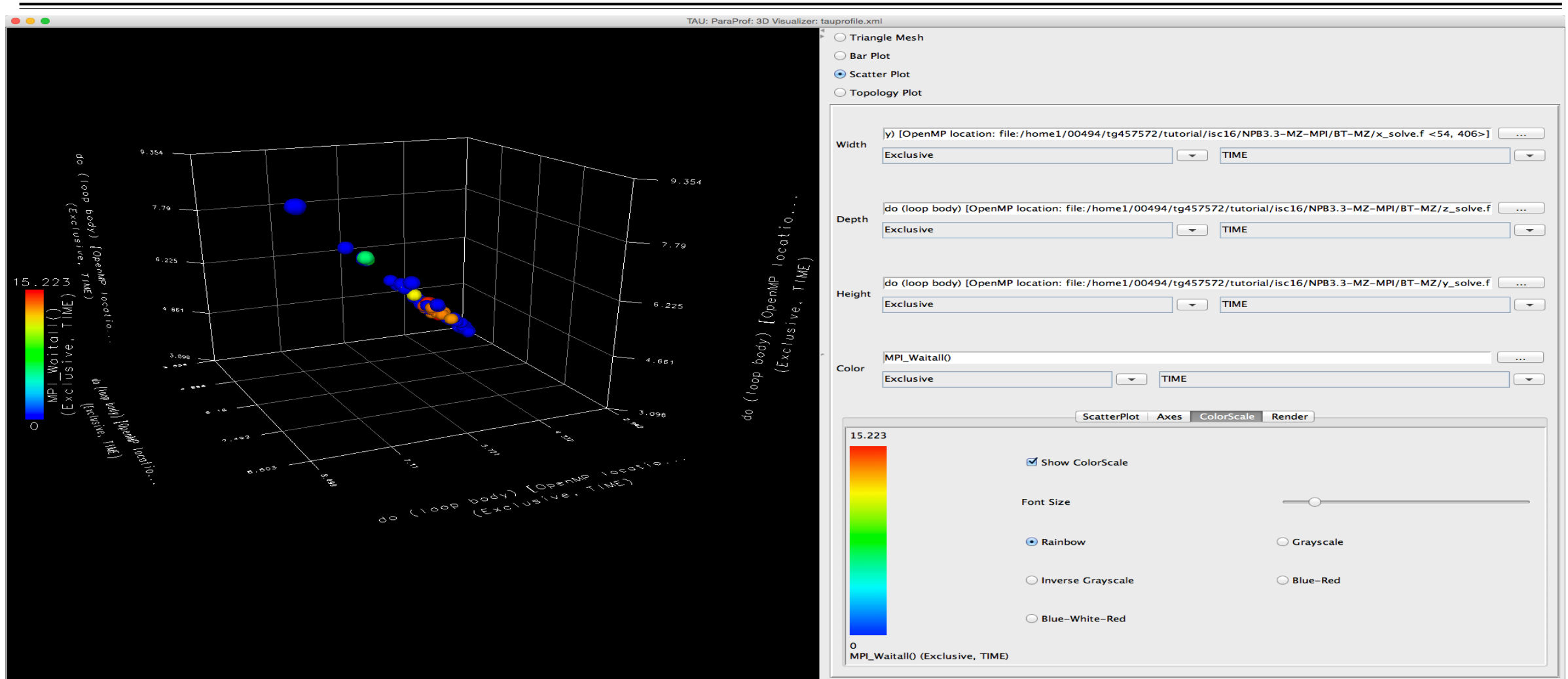
Parallel Profile Visualization: ParaProf

ParaProf 3D Communication Matrix

% export TAU_COMM_MATRIX=1

ParaProf: 3D Scatter Plot

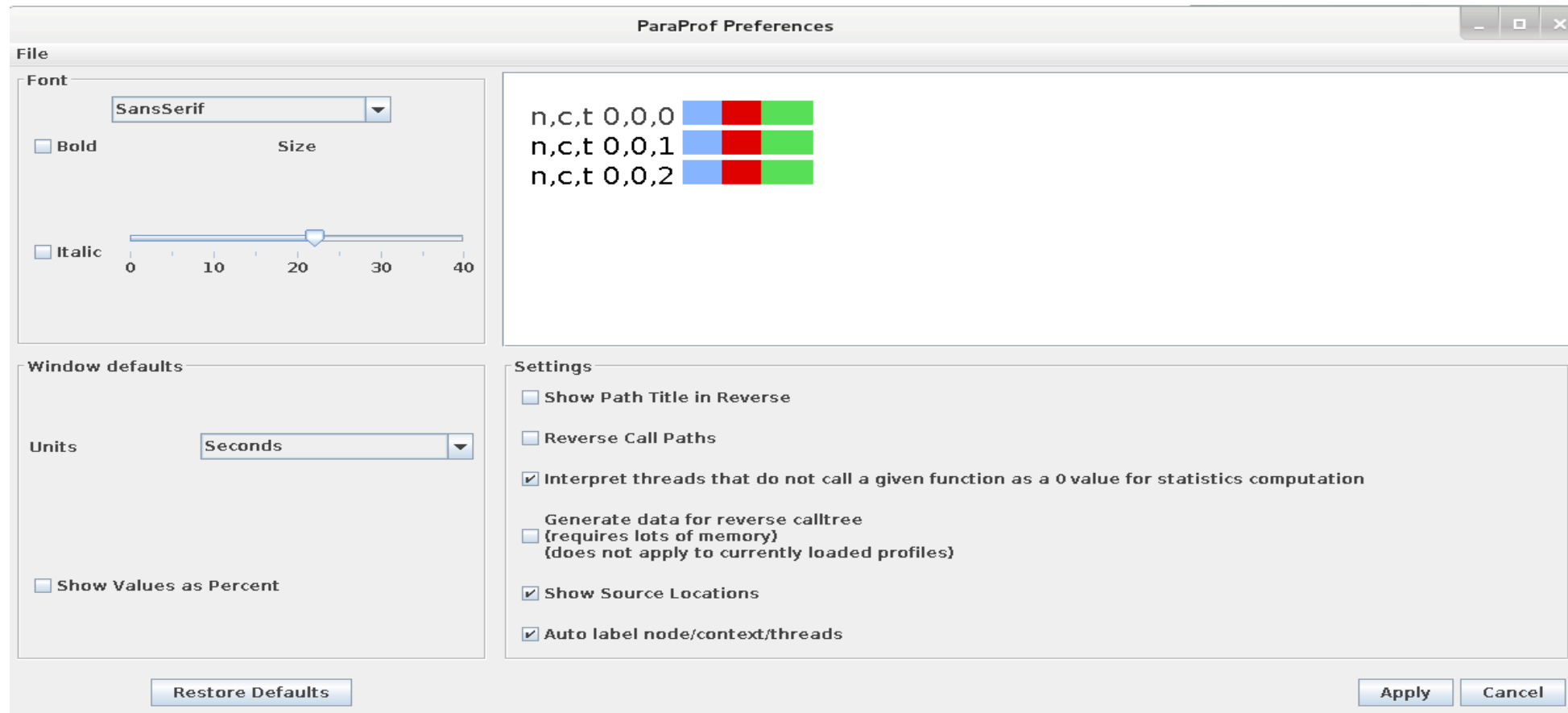# ParaProf: Score-P Profile Files, Database

ParaProf: File Preferences Window

ParaProf: Group Changer Window
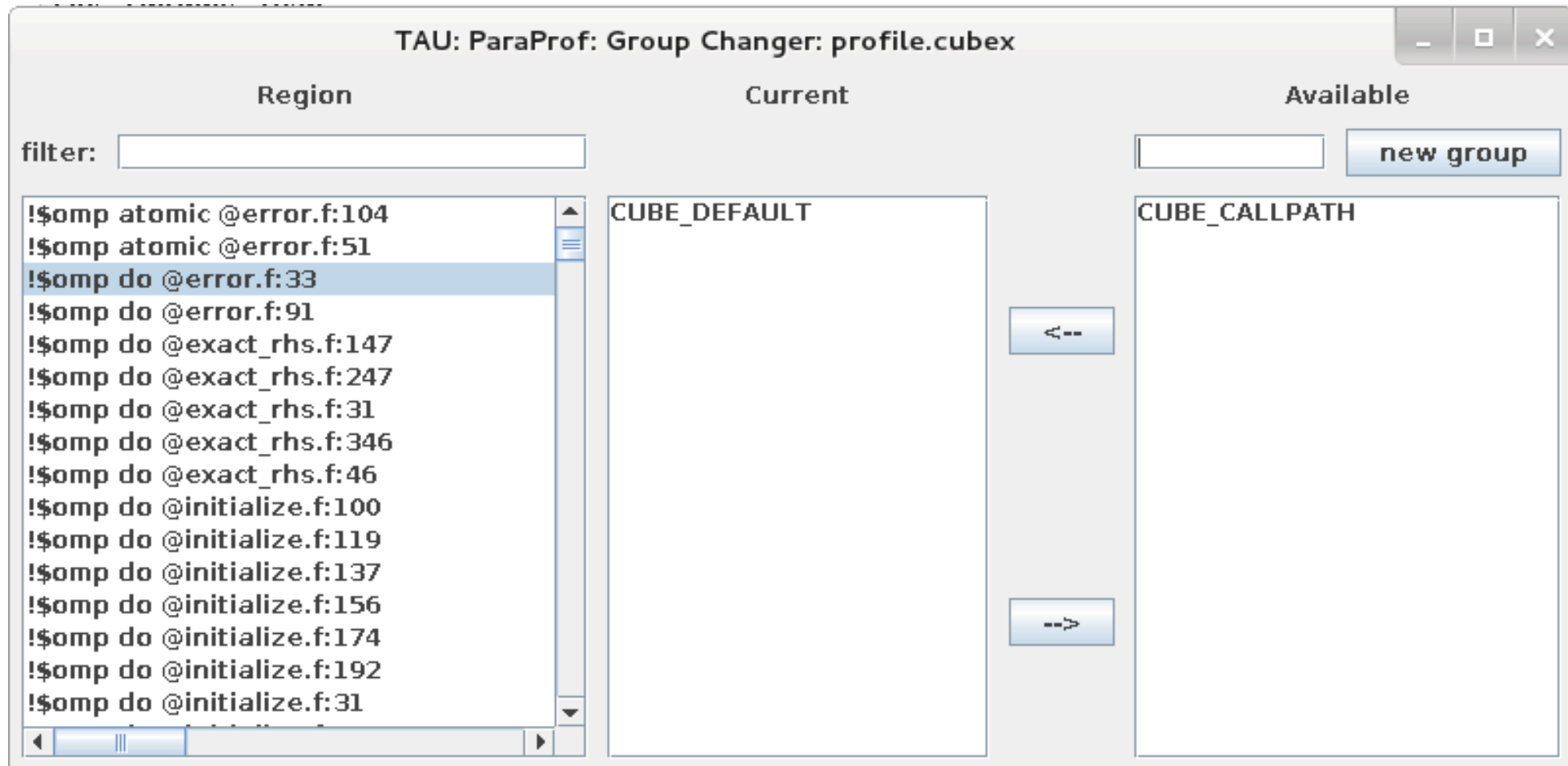
ParaProf: Derived Metric Panel in Manager Window

Sorting Derived FLOPS metric by Exclusive Time
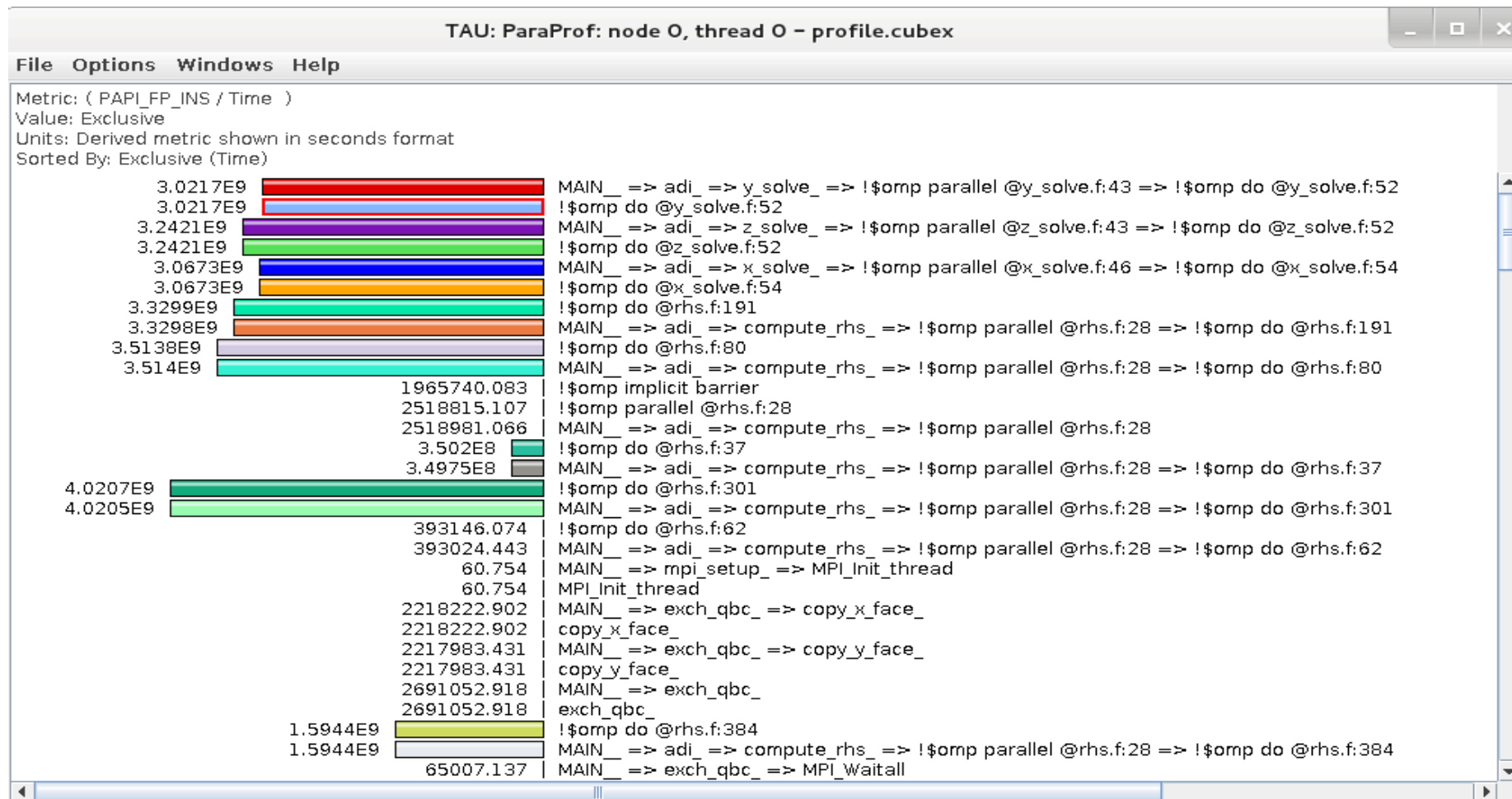
TAU hands-on exercises

Performance Research Lab, University of Oregon, Eugene, USA

Support Acknowledgments

- US Department of Energy (DOE)
  - Office of Science contracts, ECP
  - SciDAC, LBL contracts
  - LLNL-LANL-SNL ASC/NNSA contract
  - Battelle, PNNL contract
  - ANL, ORNL contract
- Department of Defense (DoD)
  - PETTT, HPCMP
- National Science Foundation (NSF)
  - Glassbox, SI-2
- NASA
- CEA, France
- Partners:
  - University of Oregon
  - ParaTools, Inc., ParaTools, SAS
  - The Ohio State University
  - University of Tennessee, Knoxville
  - T.U. Dresden, GWT
 - Juelich Supercomputing Center

Acknowledgement

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration) responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering, and early testbed platforms, in support of the nation's exascale computing imperative.

Download TAU from U. Oregon

http://tau.uoregon.edu
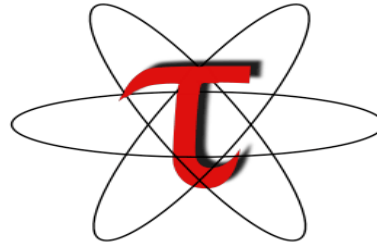
http://www.hpclinux.com [LiveDVD, OVA]
https://e4s.io [Containers for Extreme-Scale Scientific Software Stack]

Free download, open source, BSD license