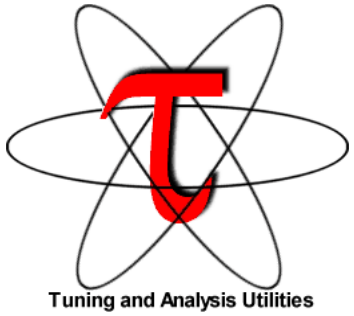
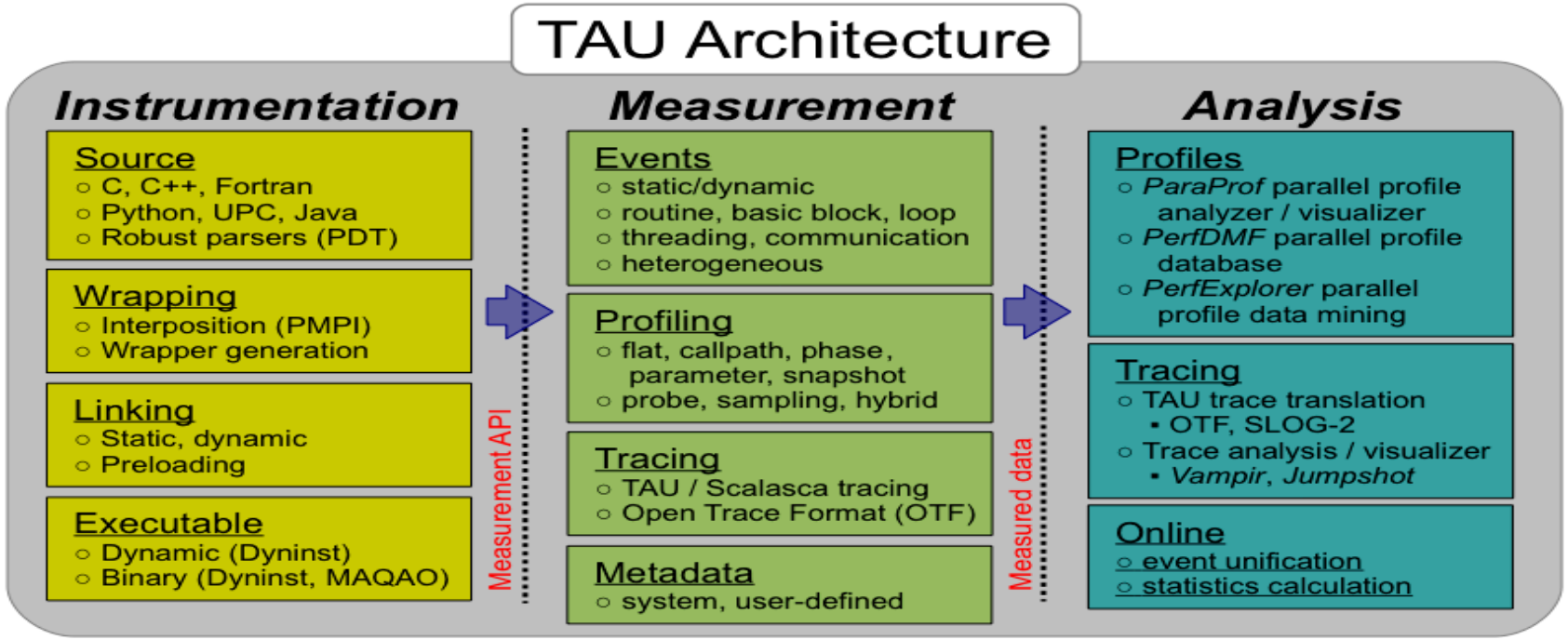
TAU Performance System®



Sameer Shende
sameer@cs.uoregon.edu
Performance Research Laboratory
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 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
  - 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

- 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
  - 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
  - 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 (OMPT, tau_exec –T ompt)
  - 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?

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 Program Database Toolkit (PDT) static tools
  - Choose an appropriate TAU stub makefile in <arch>/lib:

```
% source /home/S11505/shared/tools/setup.sh
% export TAU_MAKEFILE=$TAU/Makefile.tau-mpi-pdt-fujitsu
% export TAU_OPTIONS= '-optVerbose …'  (see tau_compiler.sh )
```

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

```
% mpifrtpx 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)
```

Automatic Source Instrumentation using PDT

```
% source /home/S11505/shared/tools/setup.sh
% cat /home/S11505/shared/tools/UO/README.TAU
% ls $TAU/Makefile.*
ls $TAU/Makefile.*
Makefile.tau-mpi-pdt-fujitsu
Makefile.tau-mpi-pdt-scorep-fujitsu
Makefile.tau-mpi-pdt-openmp-opari-fujitsu
Makefile.tau-mpi-pdt-openmp-opari-scorep-fujitsu
```

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

  Makefile.tau-mpi-pdt-openmp-opari-fujitsu for native TAU measurement library

  Makefile.tau-mpi-pdt-openmp-opari-scorep-fujitsu for Score-P library
  - Supports MPI instrumentation & PDT for automatic source instrumentation

```
% export TAU_MAKEFILE=$TAU/Makefile.tau-mpi-pdt-openmp-opari-fujitsu

% tau_f90.sh matmult.f90 -o matmult
% mpiexec -n 256 ./matmult
% paraprof
```

Compile-Time Options

Optional parameters for the TAU_OPTIONS environment variable:

% tau_compiler.sh

-optVerbose	Turn on verbose debugging messages
-optCompInst	Use compiler based instrumentation
-optNoCompInst	Do not revert to compiler instrumentation if source instrumentation fails.
-optTrackIO	Wrap POSIX I/O call and calculates vol/bw of I/O operations (configure TAU with *–iowrapper*)
-optTrackGOMP	Enable tracking GNU OpenMP runtime layer (used without –opari)
-optMemDbg	Enable runtime bounds checking (see TAU_MEMDBG_* env vars)
-optKeepFiles	Does not remove intermediate .pdb and .inst.* files
-optPreProcess	Preprocess sources (OpenMP, Fortran) before instrumentation
-optTauSelectFile=”<file>"	Specify selective instrumentation file for *tau_instrumentor*
-optTauWrapFile=”<file>"	Specify path to *link_options.tau* generated by *tau_gen_wrapper*
-optHeaderInst	Enable Instrumentation of headers
-optTrackUPCR	Track UPC runtime layer routines (used with tau_upc.sh)
-optLinking=""	Options passed to the linker. Typically $(TAU_MPI_FLIBS) $(TAU_LIBS) $(TAU_CXXLIBS)
-optCompile=""	Options passed to the compiler. Typically $(TAU_MPI_INCLUDE) $(TAU_INCLUDE) $(TAU_DEFS)
-optPdtF95Opts=""	Add options for Fortran parser in PDT (f95parse/gfparse) …

Compile-Time Options (contd.)

- Optional parameters for the TAU_OPTIONS environment variable:

% tau_compiler.sh

-optMICOffload	Links code for Intel MIC offloading, requires both host and MIC TAU libraries
-optShared	Use TAU's shared library (libTAU.so) instead of static library (default)
-optPdtCxxOpts=""	Options for C++ parser in PDT (cxxparse).
-optPdtF90Parser=""	Specify a different Fortran parser (e.g., -optPdtF90Parser=/home/S11505/shared/tools/JSC/pdt/REL-3.21/x86_64/bin/gfparse48)
-optPdtCleanscapeParser	Specify the Cleanscape Fortran parser instead of GNU gfparser
-optTau=""	Specify options to the tau_instrumentor
-optTrackDMAPP	Enable instrumentation of low-level DMAPP API calls on Cray
-optTrackPthread	Enable instrumentation of pthread calls
-optPdtCParser	Specify a different C++ parser (-optPdtCParser=cparse4101)

See tau_compiler.sh for a full list of TAU_OPTIONS.

…

Runtime Environment Variables

Environment Variable	Default	Description
TAU_TRACE	0	Setting to 1 turns on tracing
TAU_CALLPATH	0	Setting to 1 turns on callpath profiling
TAU_TRACK_MEMORY_FOOTPRINT	0	Setting to 1 turns on tracking memory usage by sampling periodically the resident set size and high water mark of memory usage
TAU_TRACK_POWER	0	Tracks power usage by sampling periodically.
TAU_CALLPATH_DEPTH	2	Specifies depth of callpath. Setting to 0 generates no callpath or routine information, setting to 1 generates flat profile and context events have just parent information (e.g., Heap Entry: foo)
TAU_SAMPLING	1	Setting to 1 enables event-based sampling.
TAU_TRACK_SIGNALS	0	Setting to 1 generate debugging callstack info when a program crashes
TAU_COMM_MATRIX	0	Setting to 1 generates communication matrix display using context events
TAU_THROTTLE	1	Setting to 0 turns off throttling. Throttles instrumentation in lightweight routines that are called frequently
TAU_THROTTLE_NUMCALLS	100000	Specifies the number of calls before testing for throttling
TAU_THROTTLE_PERCALL	10	Specifies value in microseconds. Throttle a routine if it is called over 100000 times and takes less than 10 usec of inclusive time per call
TAU_COMPENSATE	0	Setting to 1 enables runtime compensation of instrumentation overhead
TAU_PROFILE_FORMAT	Profile	Setting to “merged” generates a single file. “snapshot” generates xml format
TAU_METRICS	TIME	Setting to a comma separated list generates other metrics. (e.g., ENERGY,TIME,P_VIRTUAL_TIME,PAPI_FP_INS,PAPI_NATIVE_<event>:<subevent>)

Runtime Environment Variables (contd.)

Environment Variable	Default	Description
TAU_TRACK_MEMORY_LEAKS	0	Tracks allocates that were not de-allocated (needs –optMemDbg or tau_exec –memory)
TAU_EBS_SOURCE	TIME	Allows using PAPI hardware counters for periodic interrupts for EBS (e.g., TAU_EBS_SOURCE=PAPI_TOT_INS when TAU_SAMPLING=1)
TAU_EBS_PERIOD	100000	Specifies the overflow count for interrupts
TAU_MEMDBG_ALLOC_MIN/MAX	0	Byte size minimum and maximum subject to bounds checking (used with TAU_MEMDBG_PROTECT_*)
TAU_MEMDBG_OVERHEAD	0	Specifies the number of bytes for TAU's memory overhead for memory debugging.
TAU_MEMDBG_PROTECT_BELOW/ABOVE	0	Setting to 1 enables tracking runtime bounds checking below or above the array bounds (requires –optMemDbg while building or tau_exec –memory)
TAU_MEMDBG_ZERO_MALLOC	0	Setting to 1 enables tracking zero byte allocations as invalid memory allocations.
TAU_MEMDBG_PROTECT_FREE	0	Setting to 1 detects invalid accesses to deallocated memory that should not be referenced until it is reallocated (requires –optMemDbg or tau_exec –memory)
TAU_MEMDBG_ATTEMPT_CONTINUE	0	Setting to 1 allows TAU to record and continue execution when a memory error occurs at runtime.
TAU_MEMDBG_FILL_GAP	Undefined	Initial value for gap bytes
TAU_MEMDBG_ALINGMENT	Sizeof(int)	Byte alignment for memory allocations
TAU_EVENT_THRESHOLD	0.5	Define a threshold value (e.g., .25 is 25%) to trigger marker events for min/max

Simplifying TAU's usage (tau_exec)

- Uninstrumented execution
  - % mpiexec -n 4 ./a.out
- Track MPI performance
  - % mpiexec -n 4 tau_exec ./a.out
- Using TAU with Score-P and OTF2 with Vampir
  - % export SCOREP_ENABLE_TRACING=1
  - % mpiexec -n 4 tau_exec -T scorep,papi,mpi,pdt ./a.out
- Track memory operations
  - % export TAU_TRACK_MEMORY_LEAKS=1
  - % mpiexec -n 8 tau_exec –memory_debug ./a.out (bounds check)
- Use event based sampling (compile with –g)
  - % mpiexec -n 8 tau_exec –ebs ./a.out
  - Also –ebs_source=<PAPI_COUNTER> -ebs_period=<overflow_count>
- Load wrapper interposition library
  - % mpiexec -n 8 tau_exec –loadlib=<path/libwrapper.so> ./a.out
- Track GPGPU operations
  - % mpiexec -n 8 tau_exec –cupti ./a.out
  - % mpiexec -n 8 tau_exec –opencl ./a.out
  - % mpiexec -n 8 tau_exec –openacc ./a.out

Binary Rewriting Instrumentation

- Support for both static and dynamic executables
- Specify a list of routines to instrument
- Specify the TAU measurement library to be injected
- MAQAO [UVSQ, Intel Exascale Labs]:

```
% tau_rewrite -T [tags] a.out -o a.inst
```

- DyninstAPI [U. Maryland and U. Wisconsin, Madison]:

```
% tau_run -T [tags] a.out -o a.inst
```

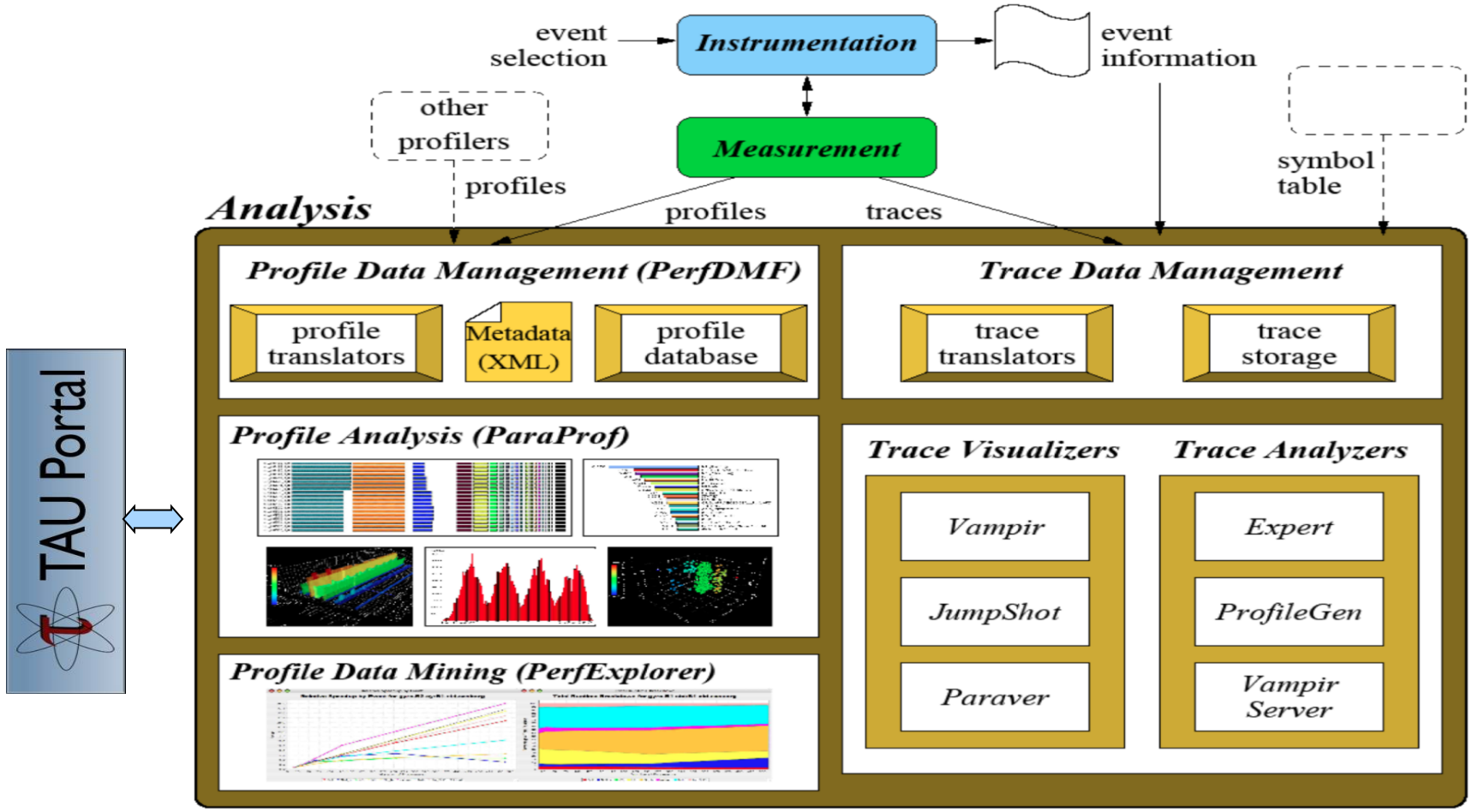
- Pebil [UC San Diego]:

```
% tau_pebil_rewrite -T [tags] a.out -o a.inst
```

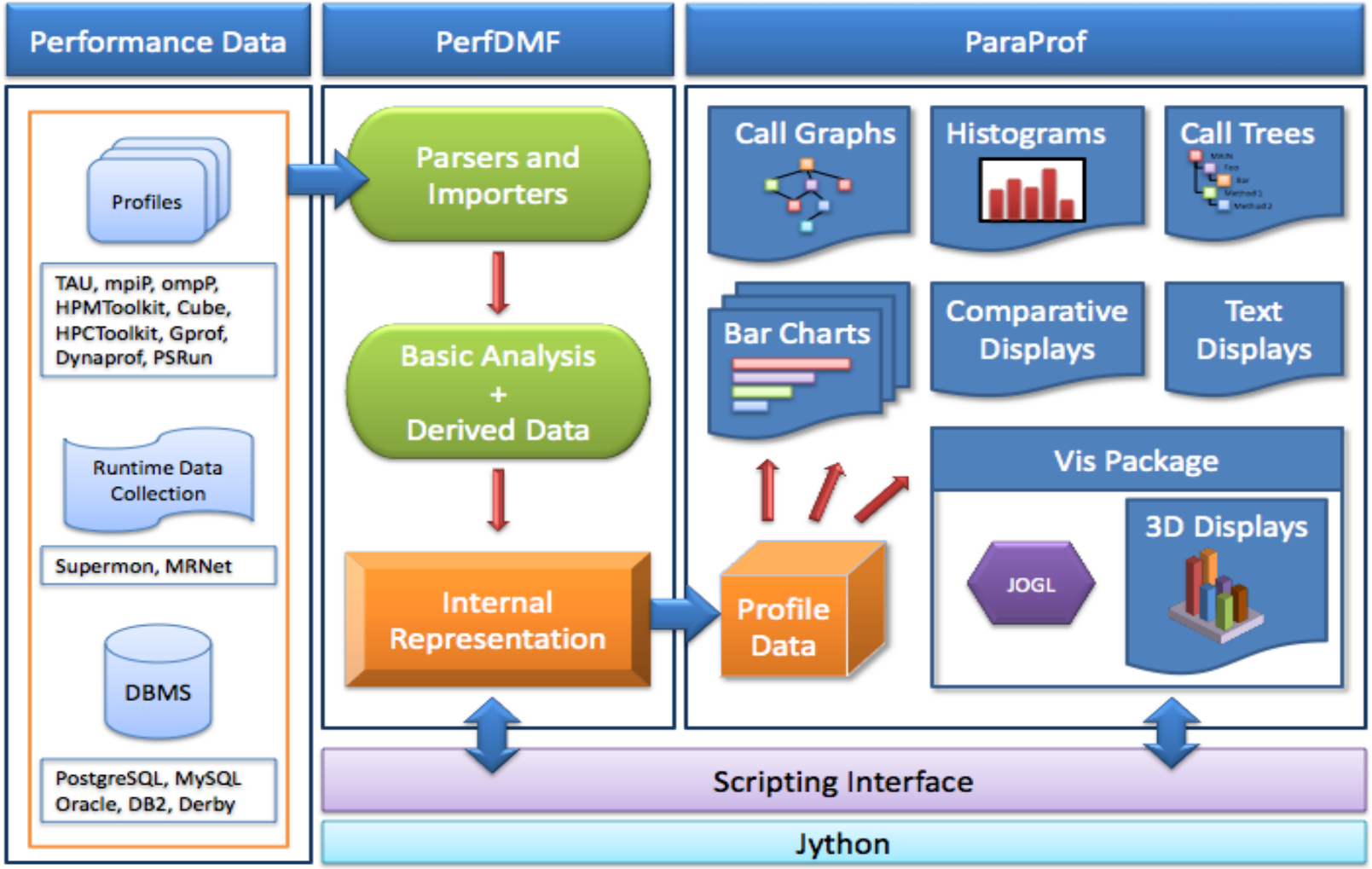
- Execute the application to get measurement data:

```
% mpiexec -n 256 ./a.inst
```

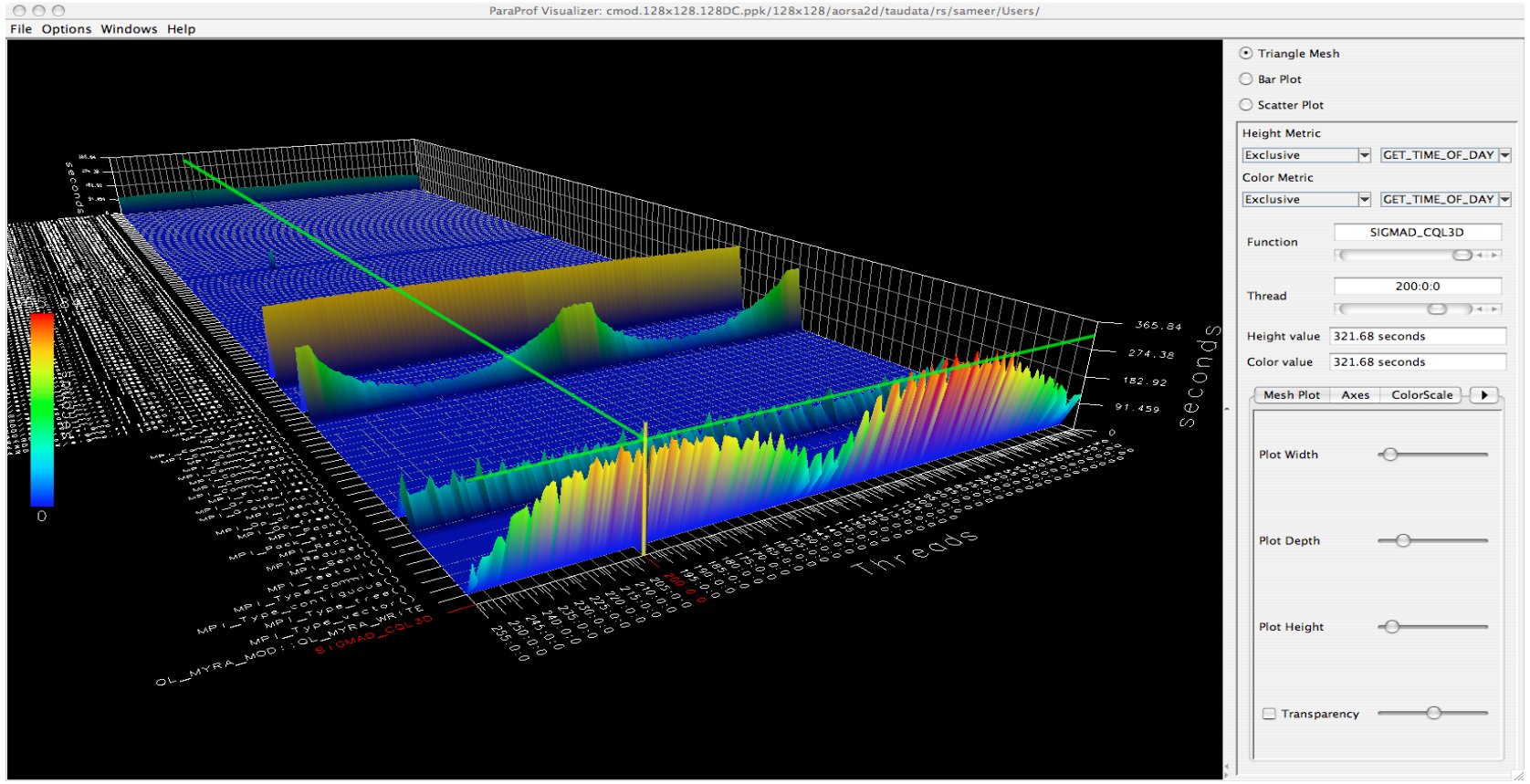
TAU Analysis



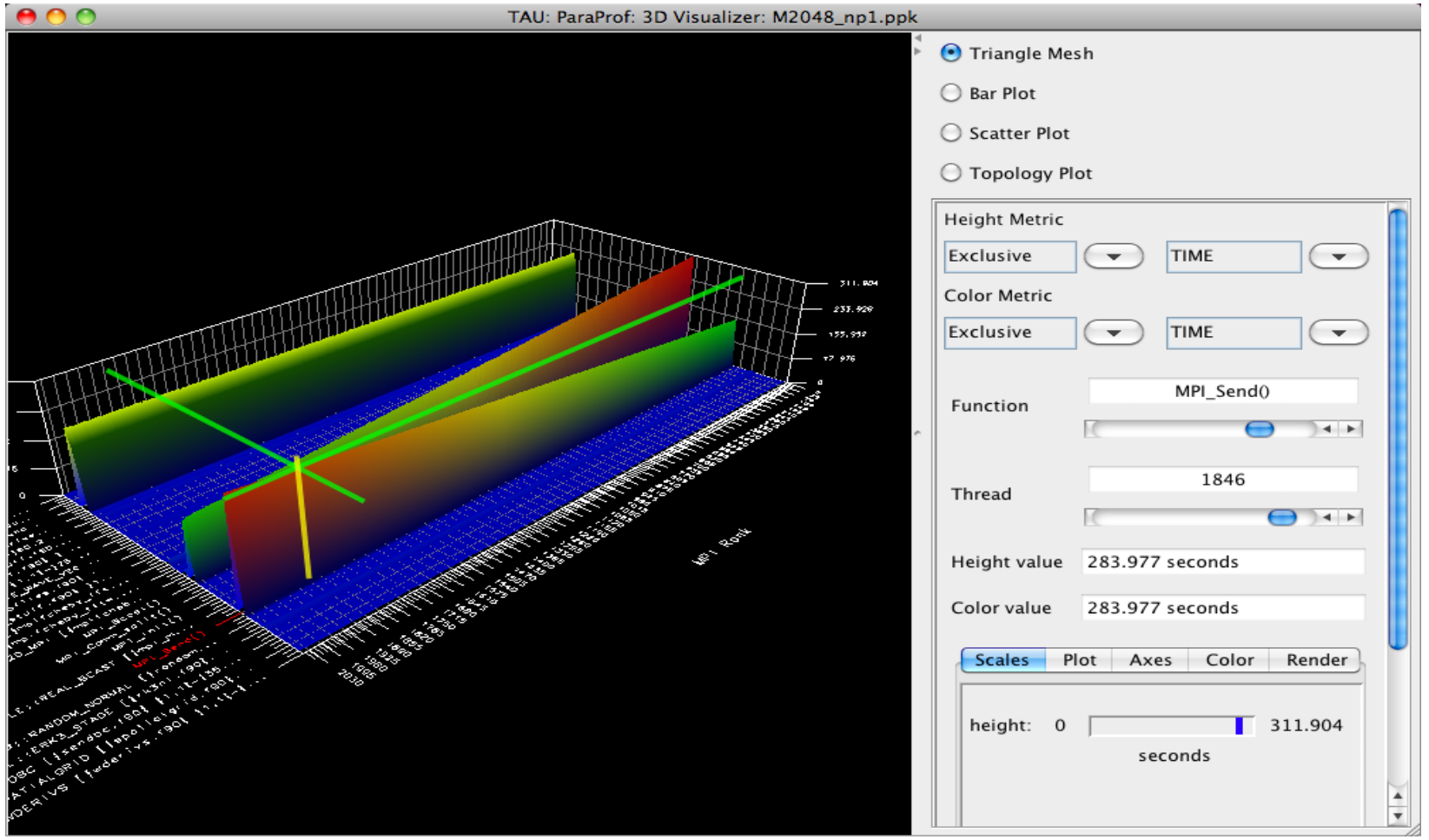
ParaProf Profile Analysis Framework



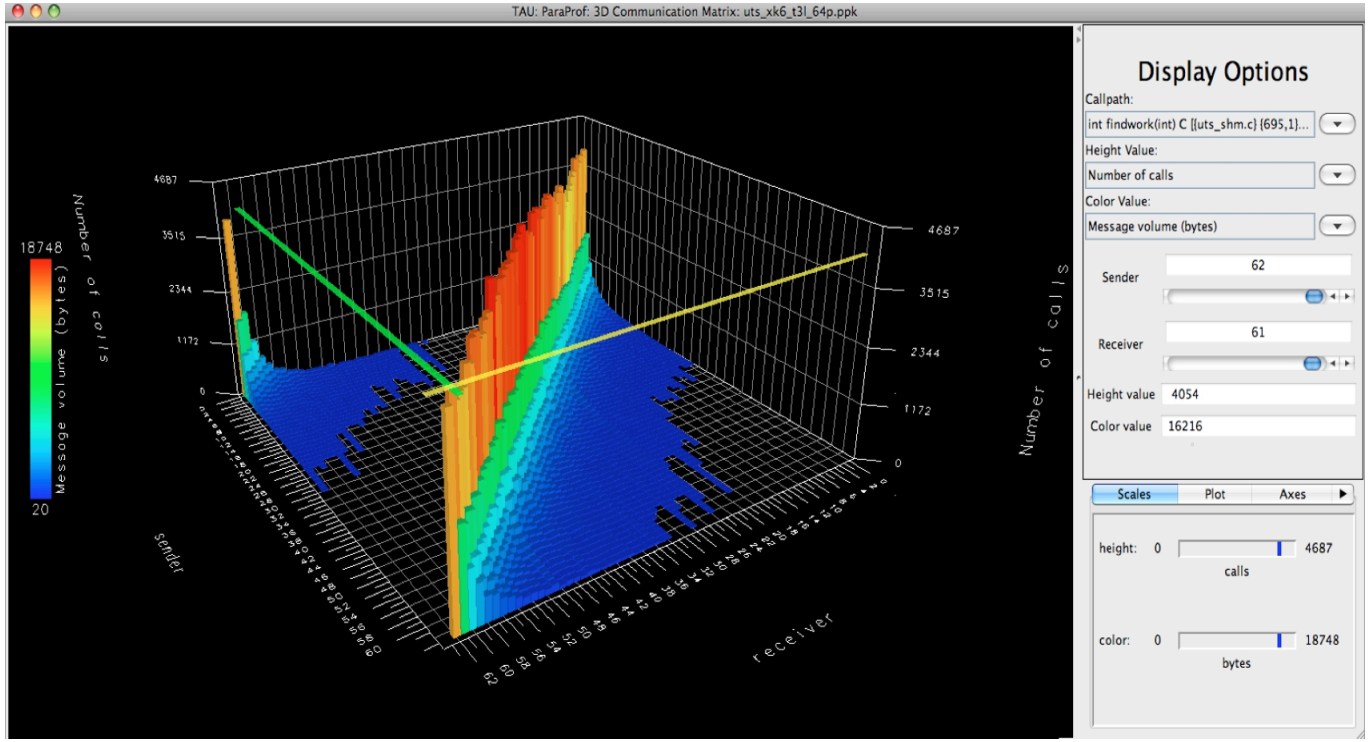
Parallel Profile Visualization: ParaProf



Parallel Profile Visualization: ParaProf

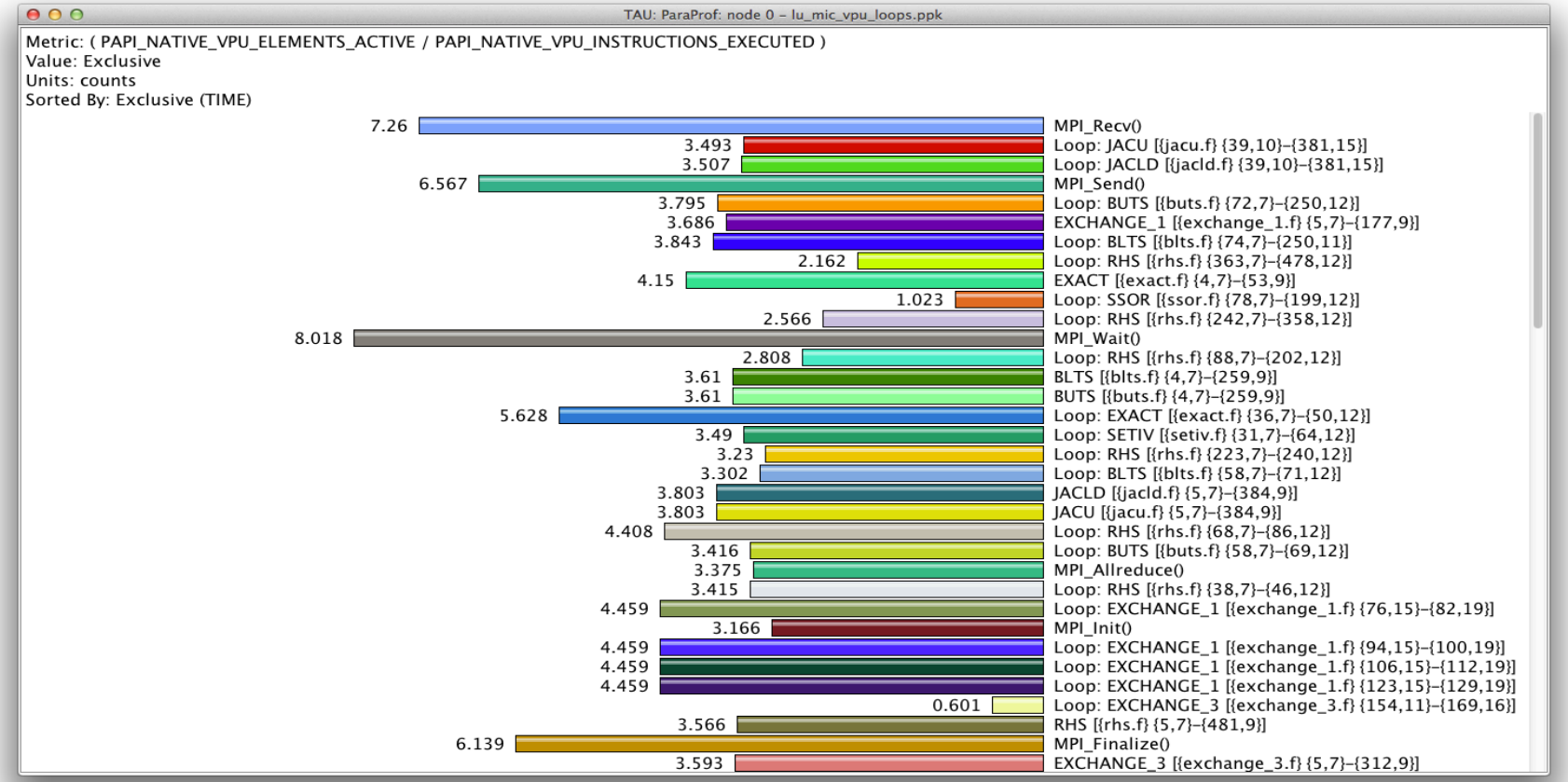


ParaProf 3D Communication Matrix



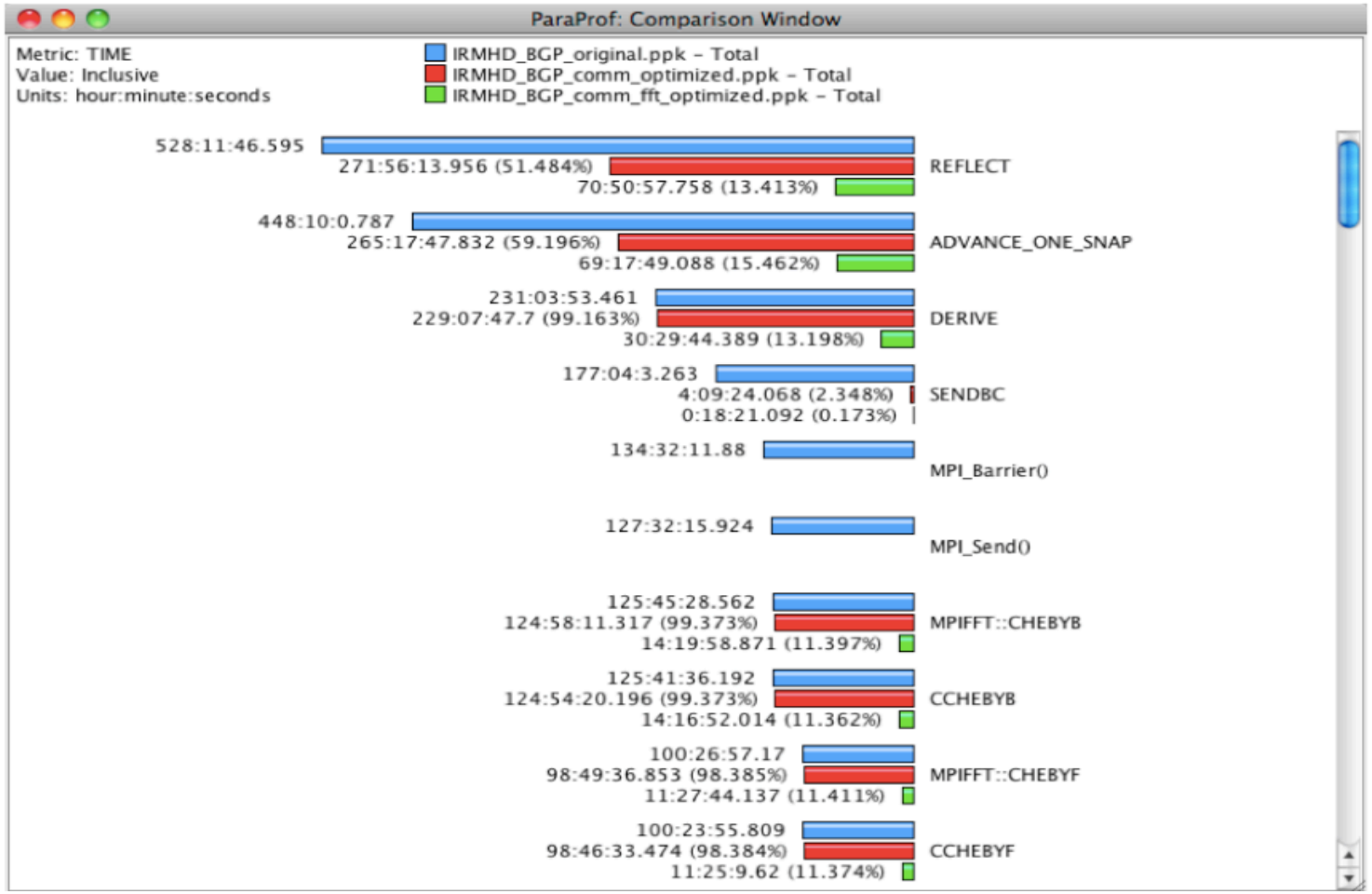
% export TAU_COMM_MATRIX=1

ParaProf Derived Metric Window: Intel MIC

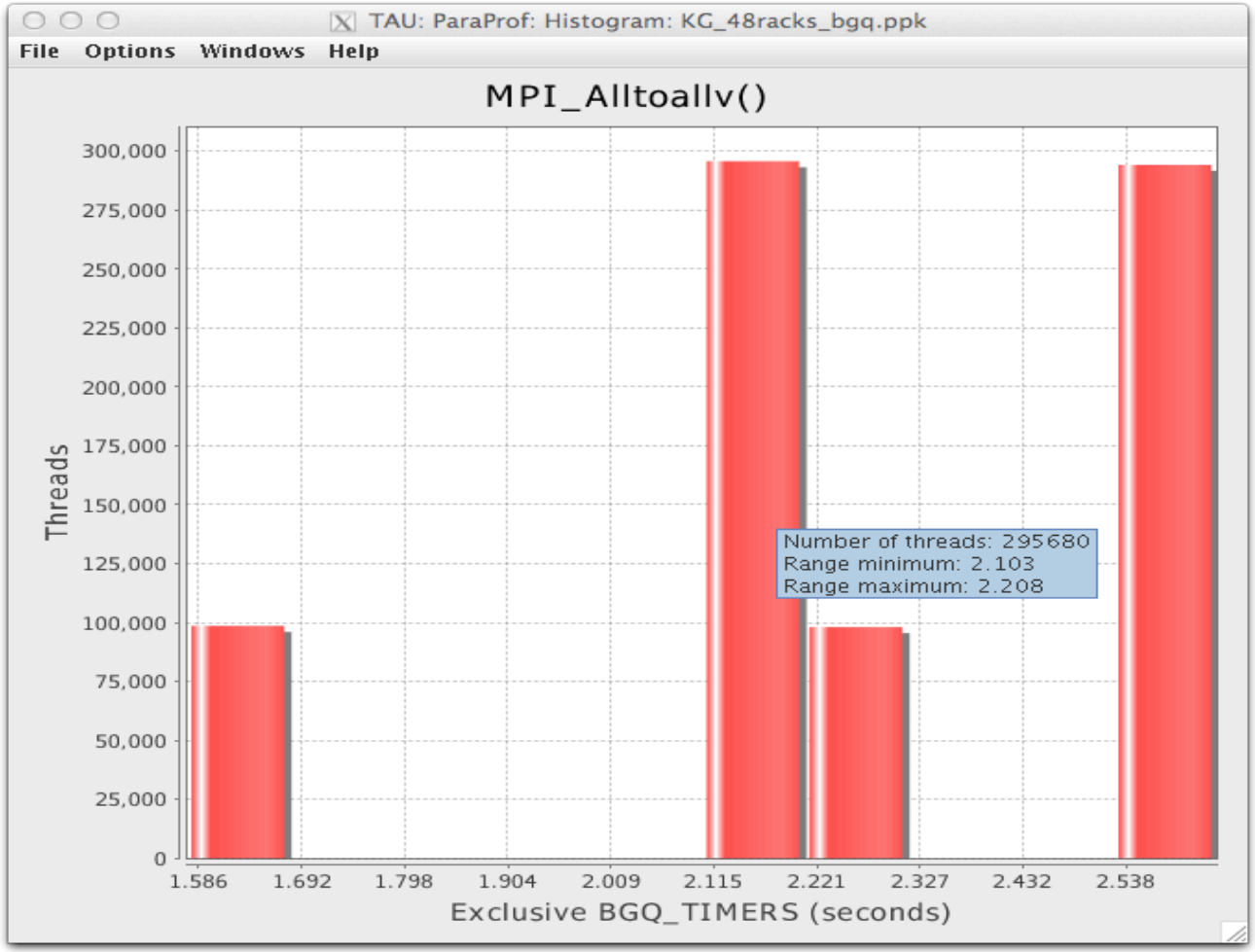


% export TAU_MAKEFILE=$TAUROOT/mic_linux/lib/Makefile.tau-icpc-papi-mpi-pdt
% export TAU_METRICS=TIME,PAPI_NATIVE_VPU_ELEMENTS_ACTIVE,PAPI_NATIVE_VPU_INSTRUCTIONS_EXECUTED

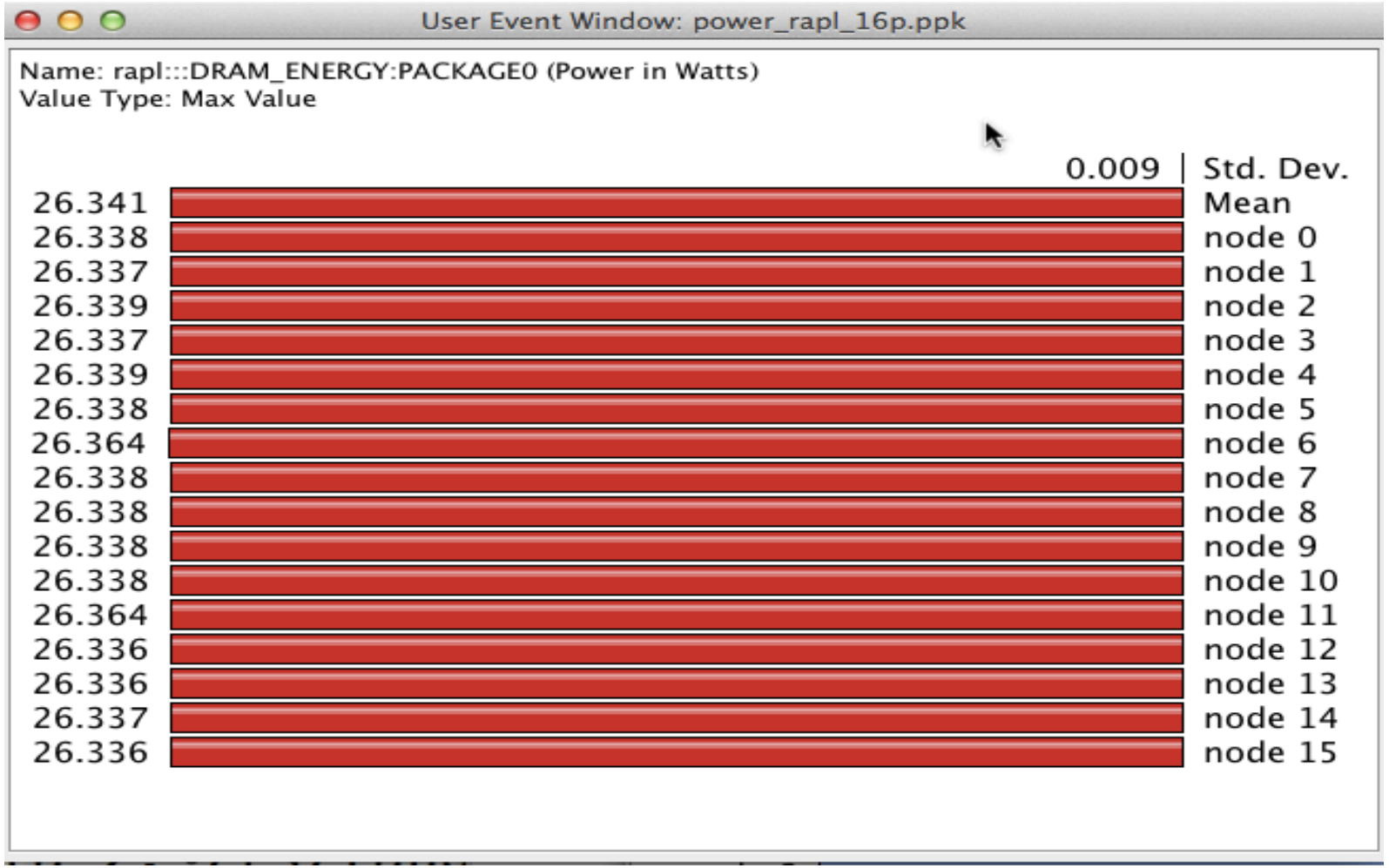
ParaProf Comparison Window



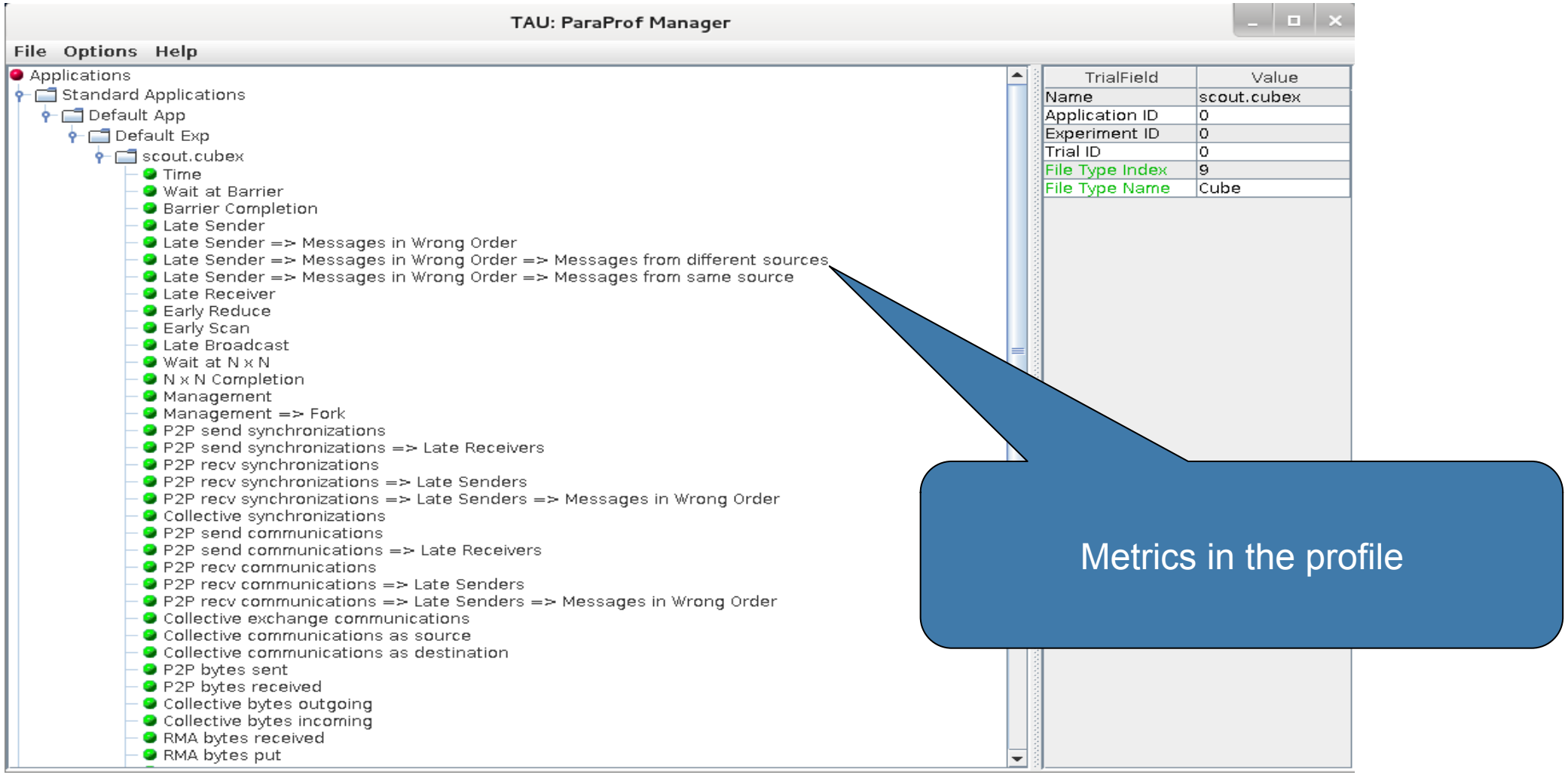
ParaProf Histogram Window



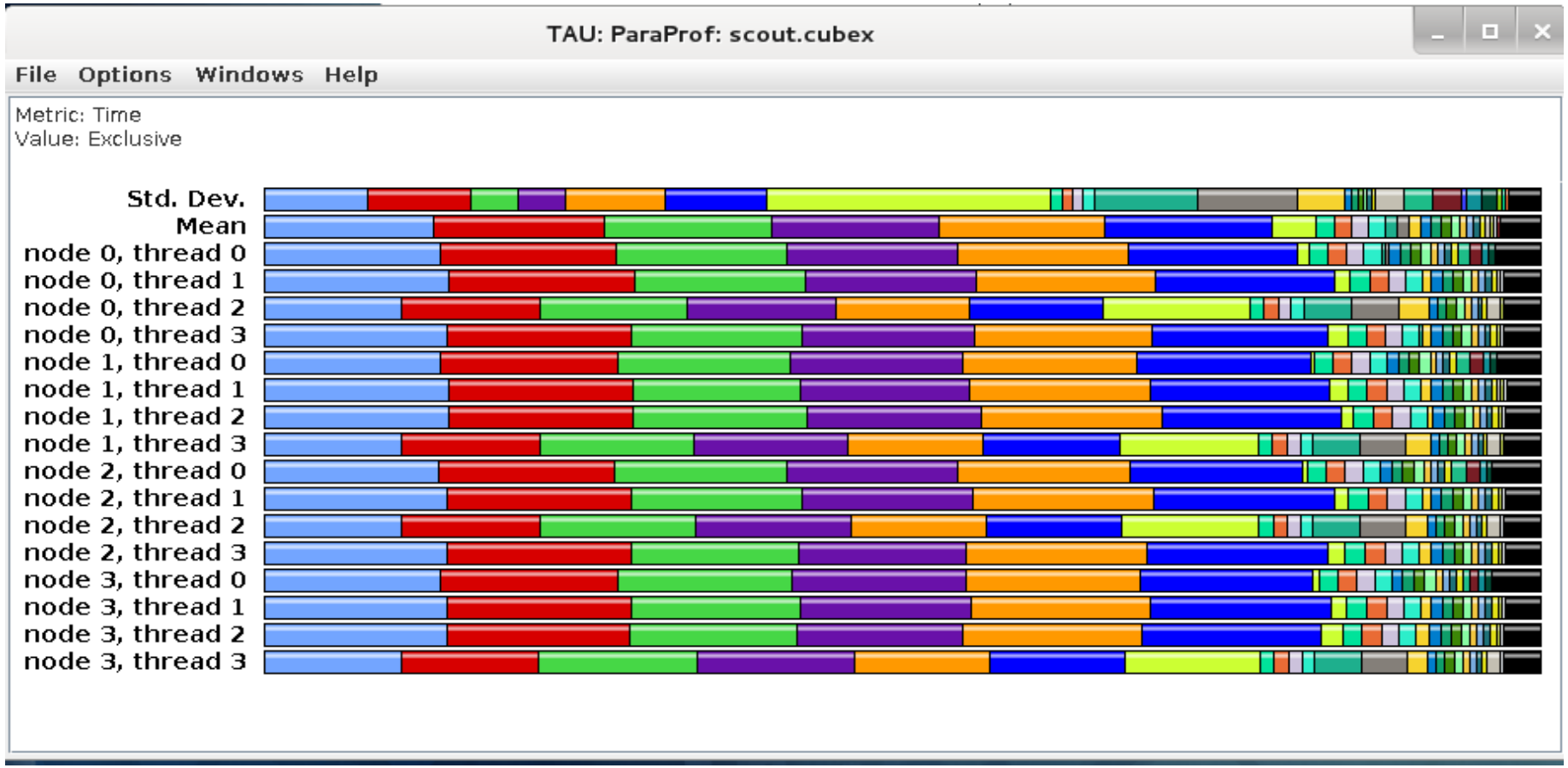
Energy Profiling in TAU



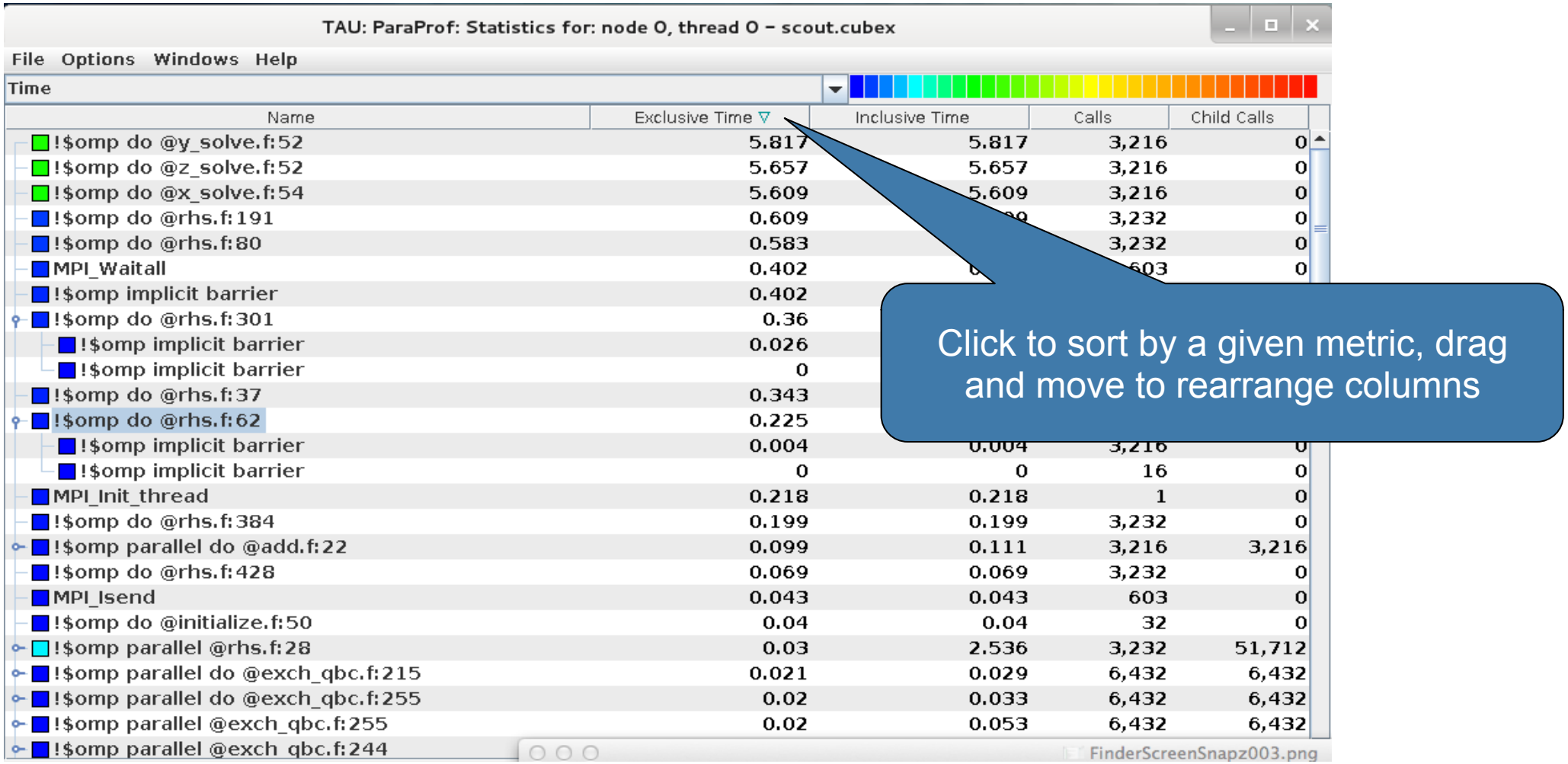
ParaProf Manager Window: scout.cubex



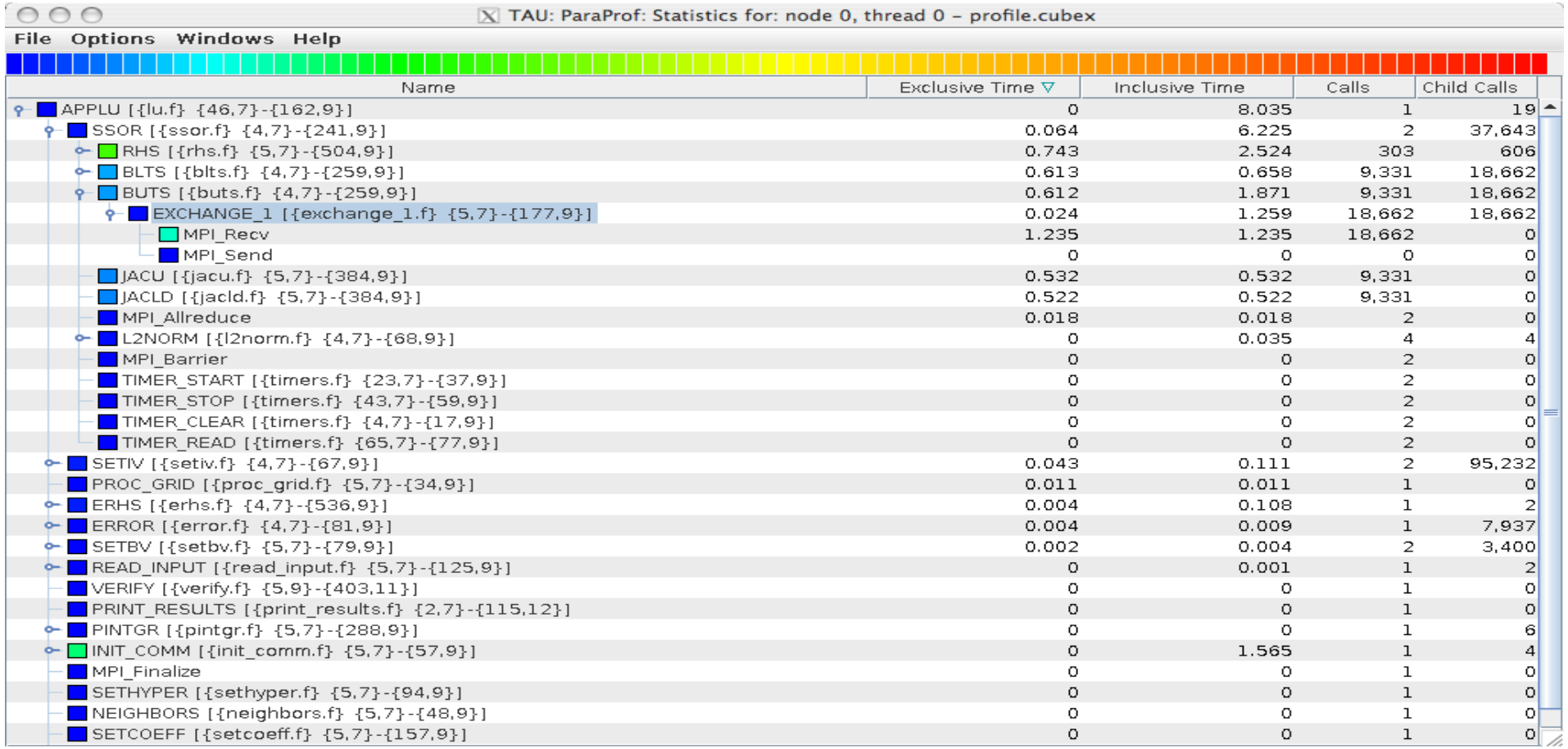
ParaProf: Main Window



ParaProf: Thread Statistics Table

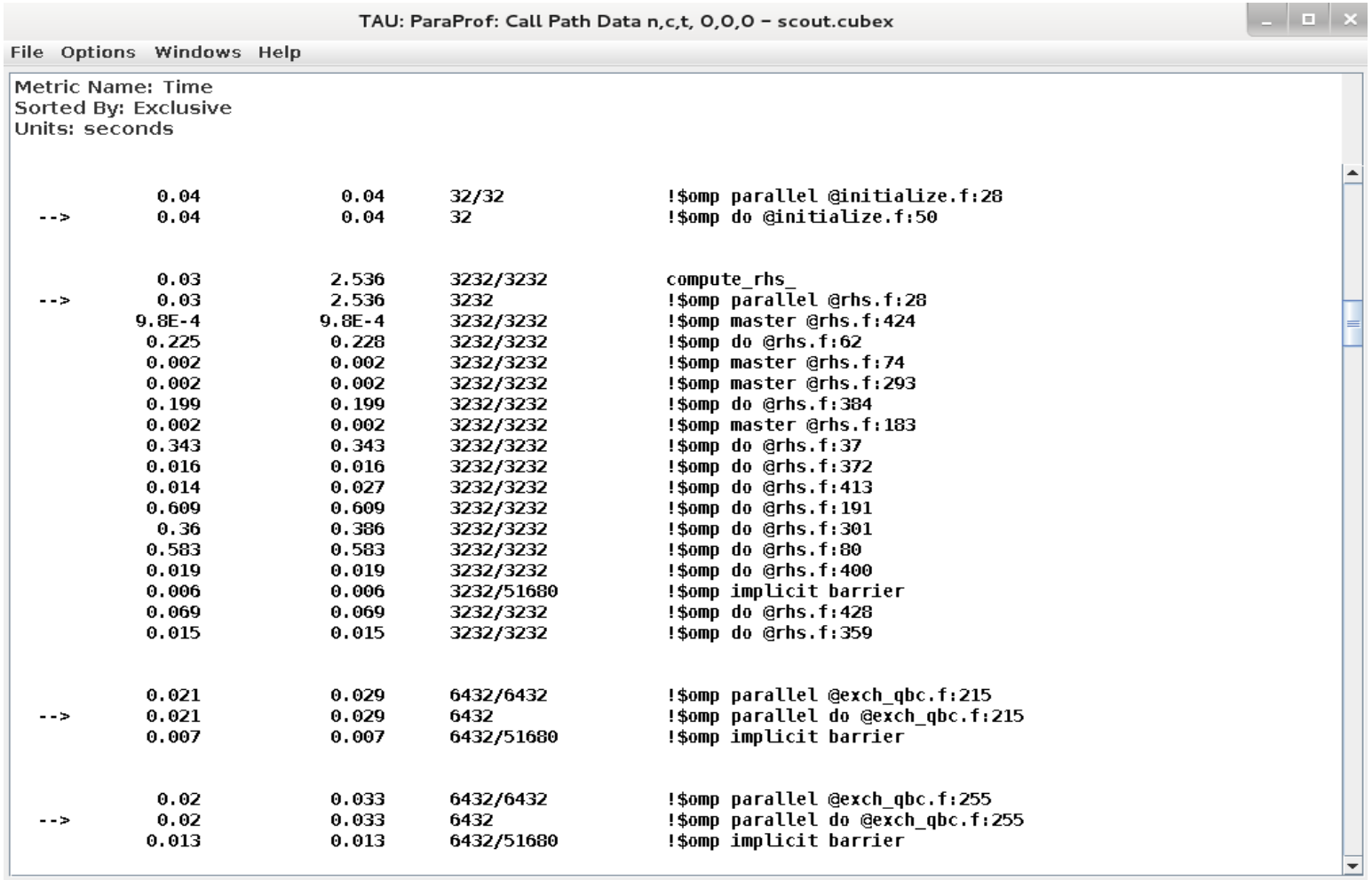


Example: Score-P with TAU (NPB LU)



Name	Exclusive Time ▽	Inclusive Time	Calls	Child Calls
APPLU [{lu.f} {46,7}-{162,9}]	0	8.035	1	19
SSOR [{ssor.f} {4,7}-{241,9}]	0.064	6.225	2	37,643
RHS [{rhs.f} {5,7}-{504,9}]	0.743	2.524	303	606
BLTS [{blts.f} {4,7}-{259,9}]	0.613	0.658	9,331	18,662
BUTS [{buts.f} {4,7}-{259,9}]	0.612	1.871	9,331	18,662
EXCHANGE_1 [{exchange_1.f} {5,7}-{177,9}]	0.024	1.259	18,662	18,662
MPI_Recv	1.235	1.235	18,662	0
MPI_Send	0	0	0	0
JACU [{jacu.f} {5,7}-{384,9}]	0.532	0.532	9,331	0
JACLD [{jacld.f} {5,7}-{384,9}]	0.522	0.522	9,331	0
MPI_Allreduce	0.018	0.018	2	0
L2NORM [{l2norm.f} {4,7}-{68,9}]	0	0.035	4	4
MPI_Barrier	0	0	2	0
TIMER_START [{timers.f} {23,7}-{37,9}]	0	0	2	0
TIMER_STOP [{timers.f} {43,7}-{59,9}]	0	0	2	0
TIMER_CLEAR [{timers.f} {4,7}-{17,9}]	0	0	2	0
TIMER_READ [{timers.f} {65,7}-{77,9}]	0	0	2	0
SETIV [{setiv.f} {4,7}-{67,9}]	0.043	0.111	2	95,232
PROC_GRID [{proc_grid.f} {5,7}-{34,9}]	0.011	0.011	1	0
ERHS [{erhs.f} {4,7}-{536,9}]	0.004	0.108	1	2
ERROR [{error.f} {4,7}-{81,9}]	0.004	0.009	1	7,937
SETBV [{setbv.f} {5,7}-{79,9}]	0.002	0.004	2	3,400
READ_INPUT [{read_input.f} {5,7}-{125,9}]	0	0.001	1	2
VERIFY [{verify.f} {5,9}-{403,11}]	0	0	1	0
PRINT_RESULTS [{print_results.f} {2,7}-{115,12}]	0	0	1	0
PINTGR [{pintgr.f} {5,7}-{288,9}]	0	0	1	6
INIT_COMM [{init_comm.f} {5,7}-{57,9}]	0	1.565	1	4
MPI_Finalize	0	0	1	0
SETHYPER [{sethyper.f} {5,7}-{94,9}]	0	0	1	0
NEIGHBORS [{neighbors.f} {5,7}-{48,9}]	0	0	1	0
SETCOEFF [{setcoeff.f} {5,7}-{157,9}]	0	0	1	0

ParaProf: Callpath Thread Relations Window



TAU demo objectives

- Familiarise with usage of TAU tools
  - complementary tools' capabilities & interoperability
- Prepare to apply tools productively to *your* applications(s)
- Exercise is based on a small portable benchmark code
  - unlikely to have significant optimisation opportunities

- Optional (recommended) exercise extensions
  - analyse performance of alternative configurations
  - investigate effectiveness of system-specific compiler/MPI optimisations and/or placement/binding/affinity capabilities
  - investigate scalability and analyse scalability limiters
  - compare performance on different HPC platforms
  - …

Local Installation (Fujitsu HPC Prime FX10 *Pi, U. Kobe*)

- Setup preferred program environment compilers
  - Default set Intel Compilers with Intel MPI
  - Generate profile files using Score-P

```
% source /home/S11505/shared/tools/setup.sh
% paraprof profile.cubex &
```

For PerfExplorer:

```
% wget http://tau.uoregon.edu/data.tgz; tar zxf data.tgz; cd data
% cat README
And follow the steps
```

NPB-MZ-MPI Suite

- The NAS Parallel Benchmark suite (MPI+OpenMP version)
  - Available from:

    http://www.nas.nasa.gov/Software/NPB
  - 3 benchmarks in Fortran77
  - Configurable for various sizes & classes
- Move into the NPB3.3-MZ-MPI root directory

```
% ls
bin/     common/  jobscript/  Makefile  README.install   SP-MZ/
BT-MZ/   config/  LU-MZ/      README    README.tutorial  sys/
```

- Subdirectories contain source code for each benchmark
  - plus additional configuration and common code
- The provided distribution has already been configured for the tutorial, such that it's ready to “make” one or more of the benchmarks and install them into a (tool-specific) “bin” subdirectory

NPB-MZ-MPI / BT: config/make.def

```
#                SITE- AND/OR PLATFORM-SPECIFIC DEFINITIONS.
#
#---------------------------------------------------------------------

#---------------------------------------------------------------------
# Configured for generic MPI with GCC compiler
#---------------------------------------------------------------------
#OPENMP  = -fopenmp      # GCC compiler
OPENMP = -openmp         # Intel compiler

...
#---------------------------------------------------------------------
# The Fortran compiler used for MPI programs
#---------------------------------------------------------------------
# MPIF77 = mpifrtpx # Fujitsu compiler

# Alternative variant to perform instrumentation
MPIF77 = tau_f90.sh -tau_makefile=<path>/Makefile.tau-[options]

# PREP is a generic preposition macro for instrumentation preparation
#MPIF77 = $(PREP) mpif77 -f77=ifort
#MPIF77 = scorep …
...
```

Default (no instrumentation)

Uncomment TAU's compiler wrapper to do source instrumentation with TAU
Comment out Score-P wrapper

Building an NPB-MZ-MPI Benchmark

```
% make
  ============================================
  =      NAS PARALLEL BENCHMARKS 3.3         =
  =      MPI+OpenMP Multi-Zone Versions      =
  =      F77                                 =
  ============================================

  To make a NAS multi-zone benchmark type

        make <benchmark-name> CLASS=<class> NPROCS=<nprocs>

  where <benchmark-name> is "bt-mz", "lu-mz", or "sp-mz"
        <class>          is "S", "W", "A" through "F"
        <nprocs>         is number of processes

  [...]

 ***********************************************************
 * Custom build configuration is specified in config/make.def  *
 * Suggested tutorial exercise configuration for HPC systems:  *
 *       make bt-mz CLASS=B NPROCS=8                           *
 ***********************************************************
```

- Type "make" for instructions

Building an NPB-MZ-MPI Benchmark

```
% make suite
make[1]: Entering directory `BT-MZ'
make[2]: Entering directory `sys'
cc  -o setparams setparams.c -lm
make[2]: Leaving directory `sys'
../sys/setparams bt-mz 8 B
make[2]: Entering directory `../BT-MZ'
tau_f90.sh -c  -O3 -g -openmp       bt.f
[…]
tau_f90.sh -c  -O3 -g -openmp       mpi_setup.f
cd ../common;  mpiifort -c  -O3 -g -openmp        print_results.f
cd ../common;  mpiifort -c  -O3 -g -openmp        timers.f
tau_f90.sh -O3 -g -openmp   -o ../bin.tau/bt-mz_C.8 bt.o
 initialize.o exact_solution.o exact_rhs.o set_constants.o adi.o
 rhs.o zone_setup.o x_solve.o y_solve.o  exch_qbc.o solve_subs.o
 z_solve.o add.o error.o verify.o mpi_setup.o ../common/print_results.o
 ../common/timers.o
make[2]: Leaving directory `BT-MZ'
Built executable ../bin/bt-mz_B.8
make[1]: Leaving directory `BT-MZ'
```

- Specify the benchmark configuration
  - benchmark name: **bt-mz**, lu-mz, sp-mz
  - the number of MPI processes: NPROCS=**8**
  - the benchmark class (S, W, A, B, C, D, E): CLASS=**B**

Shortcut: `% make suite`

NPB-MZ-MPI / BT without TAU

```
% make bt-mz CLASS=B NPROCS=4
% cd bin
% cp ../jobscript/fx10/run.sh .
% pjsub run.sh
% cat mzmpibt.o<job_id>
 NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP Benchmark
 Number of zones:   8 x 8
 Iterations:  200   dt:   0.000300
 Number of active processes:     8
 Total number of threads:       32(  4.0 threads/process)

 Time step    1
 Time step   20
  [...]
 Time step  180
 Time step  200
 Verification Successful

 BT-MZ Benchmark Completed.
 Time in seconds = 16.03
%
```

- Copy jobscript and launch as a hybrid MPI+OpenMP application

Hint: save the benchmark output (or note the run time) to be able to refer to it later

NPB-MZ-MPI / BT (Block Tridiagonal Solver)

- What does it do?
  - Solves a discretized version of the unsteady, compressible Navier-Stokes equations in three spatial dimensions
  - Performs 200 time-steps on a regular 3-dimensional grid
- Implemented in 20 or so Fortran77 source modules

- Uses MPI & OpenMP in combination
  - 8 processes each with 4 threads should be reasonable

  - bt-mz_B.8 should take around 16 seconds

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:
    mpiexec -n 2 tau_exec -T icpc,ompt,mpi  -ompt ./a.out
    mpiexec -n 2 tau_exec -io ./a.out
Example - event-based sampling with samples taken every 1,000,000 FP instructions
    mpiexec -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)
    mpiexec -n 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!

Using tau_exec -ebs

- Load TAU tools in your path

```
% source /home/S11505/shared/tools/setup.sh
```

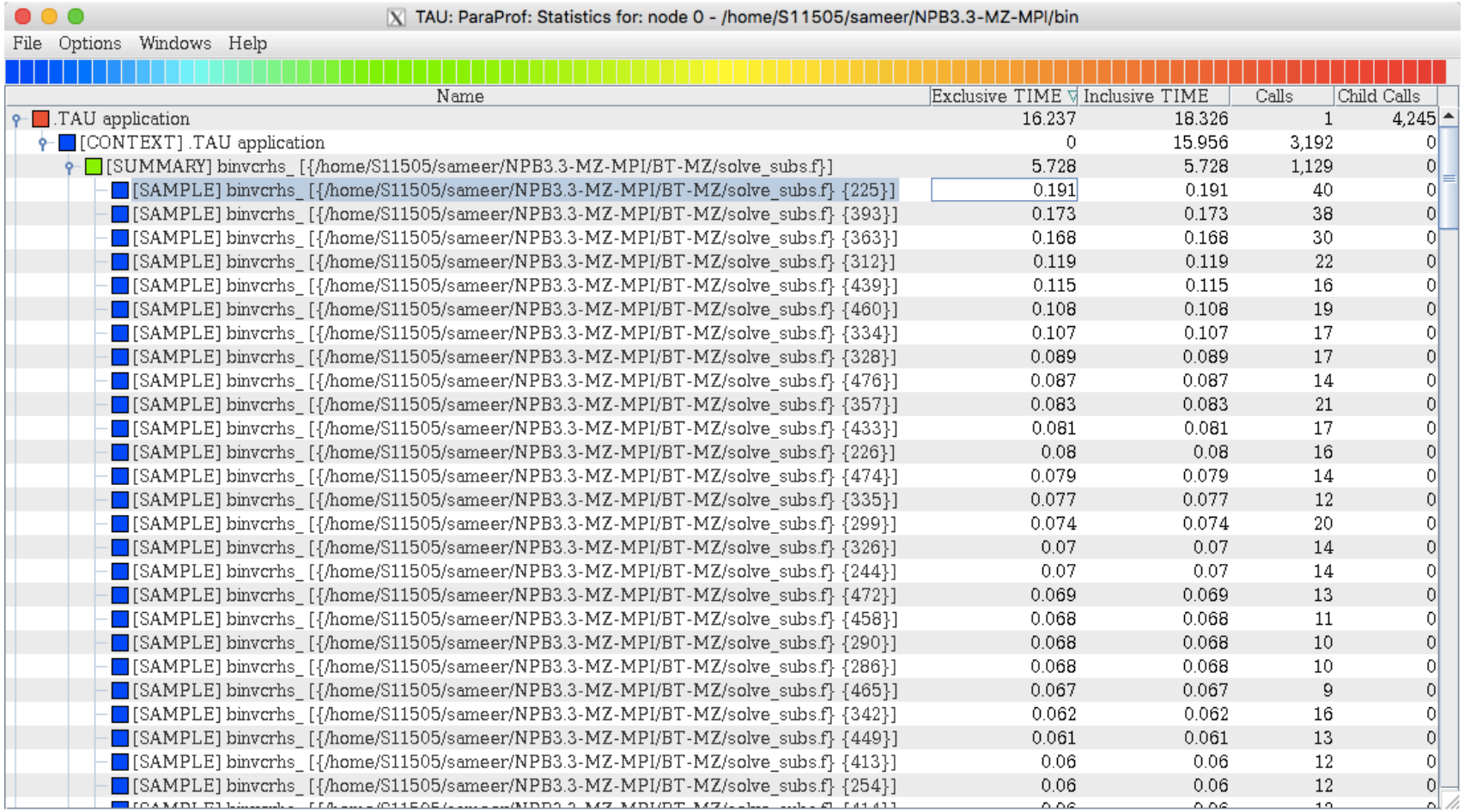
- Make clean and build new tool-specific executable; compile application with -g

```
% make clean
% make suite
Built executable ../bin/bt-mz_B.8
```

- Change to the directory containing the new executable before running it with the desired tool configuration

```
% cp ../jobscript/fx10/run.sh .
EDIT run.sh and add tau_exec -ebs:
mpirun -np $NPROCS $EXE                changes to
mpirun -np $NPROCS tau_exec -ebs $EXE
% pjsub run.sh
% paraprof &
```

ParaProf: Thread Statistics Table (right click on node 0)



Name	Exclusive TIME	Inclusive TIME	Calls	Child Calls
.TAU application	16.237	18.326	1	4,245
[CONTEXT] .TAU application	0	15.956	3,192	0
[SUMMARY] binvcrhs_ [{/home/S11505/sameer/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f}]	5.728	5.728	1,129	0
[SAMPLE] binvcrhs_ [{/home/S11505/sameer/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {225}]	0.191	0.191	40	0
[SAMPLE] binvcrhs_ [{/home/S11505/sameer/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {393}]	0.173	0.173	38	0
[SAMPLE] binvcrhs_ [{/home/S11505/sameer/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {363}]	0.168	0.168	30	0
[SAMPLE] binvcrhs_ [{/home/S11505/sameer/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {312}]	0.119	0.119	22	0
[SAMPLE] binvcrhs_ [{/home/S11505/sameer/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {439}]	0.115	0.115	16	0
[SAMPLE] binvcrhs_ [{/home/S11505/sameer/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {460}]	0.108	0.108	19	0
[SAMPLE] binvcrhs_ [{/home/S11505/sameer/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {334}]	0.107	0.107	17	0
[SAMPLE] binvcrhs_ [{/home/S11505/sameer/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {328}]	0.089	0.089	17	0
[SAMPLE] binvcrhs_ [{/home/S11505/sameer/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {476}]	0.087	0.087	14	0
[SAMPLE] binvcrhs_ [{/home/S11505/sameer/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {357}]	0.083	0.083	21	0
[SAMPLE] binvcrhs_ [{/home/S11505/sameer/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {433}]	0.081	0.081	17	0
[SAMPLE] binvcrhs_ [{/home/S11505/sameer/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {226}]	0.08	0.08	16	0
[SAMPLE] binvcrhs_ [{/home/S11505/sameer/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {474}]	0.079	0.079	14	0
[SAMPLE] binvcrhs_ [{/home/S11505/sameer/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {335}]	0.077	0.077	12	0
[SAMPLE] binvcrhs_ [{/home/S11505/sameer/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {299}]	0.074	0.074	20	0
[SAMPLE] binvcrhs_ [{/home/S11505/sameer/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {326}]	0.07	0.07	14	0
[SAMPLE] binvcrhs_ [{/home/S11505/sameer/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {244}]	0.07	0.07	14	0
[SAMPLE] binvcrhs_ [{/home/S11505/sameer/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {472}]	0.069	0.069	13	0
[SAMPLE] binvcrhs_ [{/home/S11505/sameer/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {458}]	0.068	0.068	11	0
[SAMPLE] binvcrhs_ [{/home/S11505/sameer/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {290}]	0.068	0.068	10	0
[SAMPLE] binvcrhs_ [{/home/S11505/sameer/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {286}]	0.068	0.068	10	0
[SAMPLE] binvcrhs_ [{/home/S11505/sameer/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {465}]	0.067	0.067	9	0
[SAMPLE] binvcrhs_ [{/home/S11505/sameer/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {342}]	0.062	0.062	16	0
[SAMPLE] binvcrhs_ [{/home/S11505/sameer/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {449}]	0.061	0.061	13	0
[SAMPLE] binvcrhs_ [{/home/S11505/sameer/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {413}]	0.06	0.06	12	0
[SAMPLE] binvcrhs_ [{/home/S11505/sameer/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {254}]	0.06	0.06	12	0

```
% paraprof
```

Right click on node 0 -> Show Thread Statistics Table. Click on column to sort, open keys on left

TAU Source Instrumentation

- See TAU commands

```
% cat /home/S11505/shared/tools/UO/README.TAU
```

- Make clean and build new tool-specific executable

```
% make clean
% export TAU_MAKEFILE=$TAU/Makefile.tau-mpi-pdt-openmp-opari-scorep-fujitsu
% make MPIF77=tau_f77.sh suite
Built executable ../bin.tau/bt-mz_B.8
```

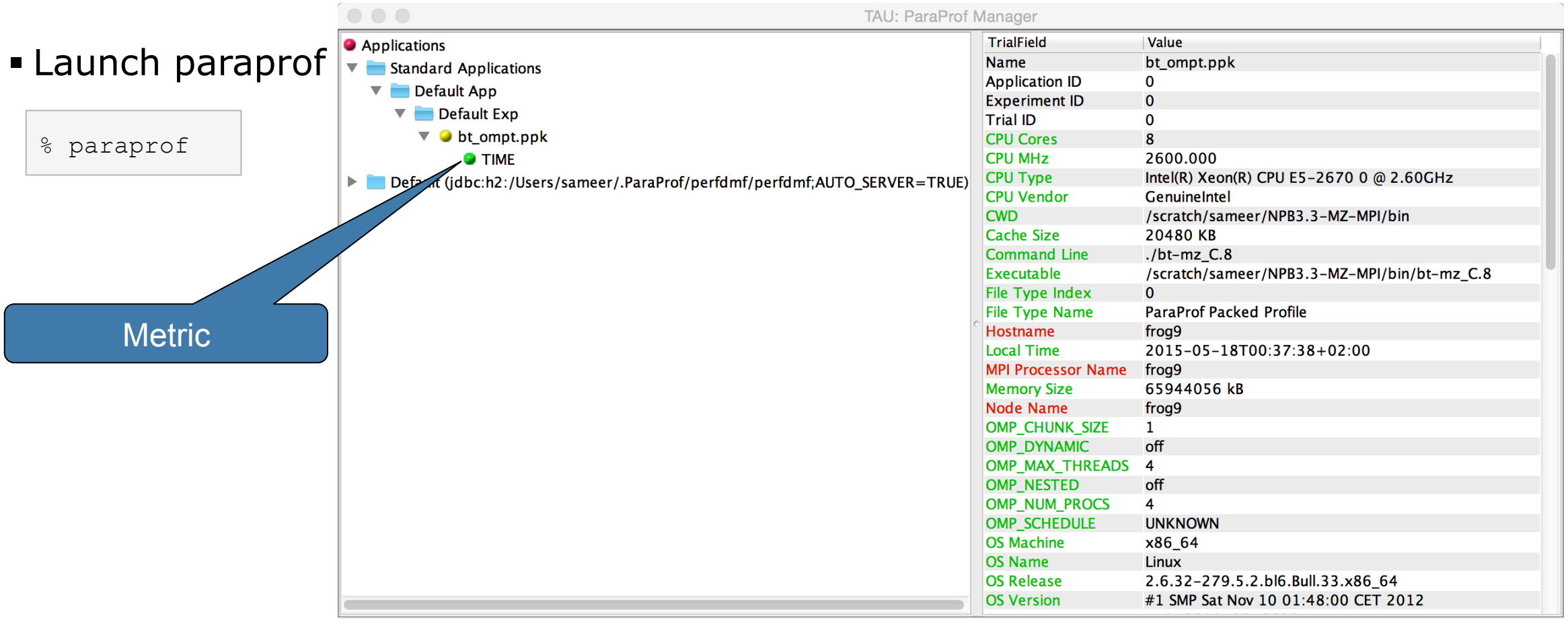
- Change to the directory containing the new executable before running it with the desired tool configuration

```
% cd bin.tau
% cp ../jobscript/fx10/run.sh .
% pjsub run.sh
% paraprof &
```

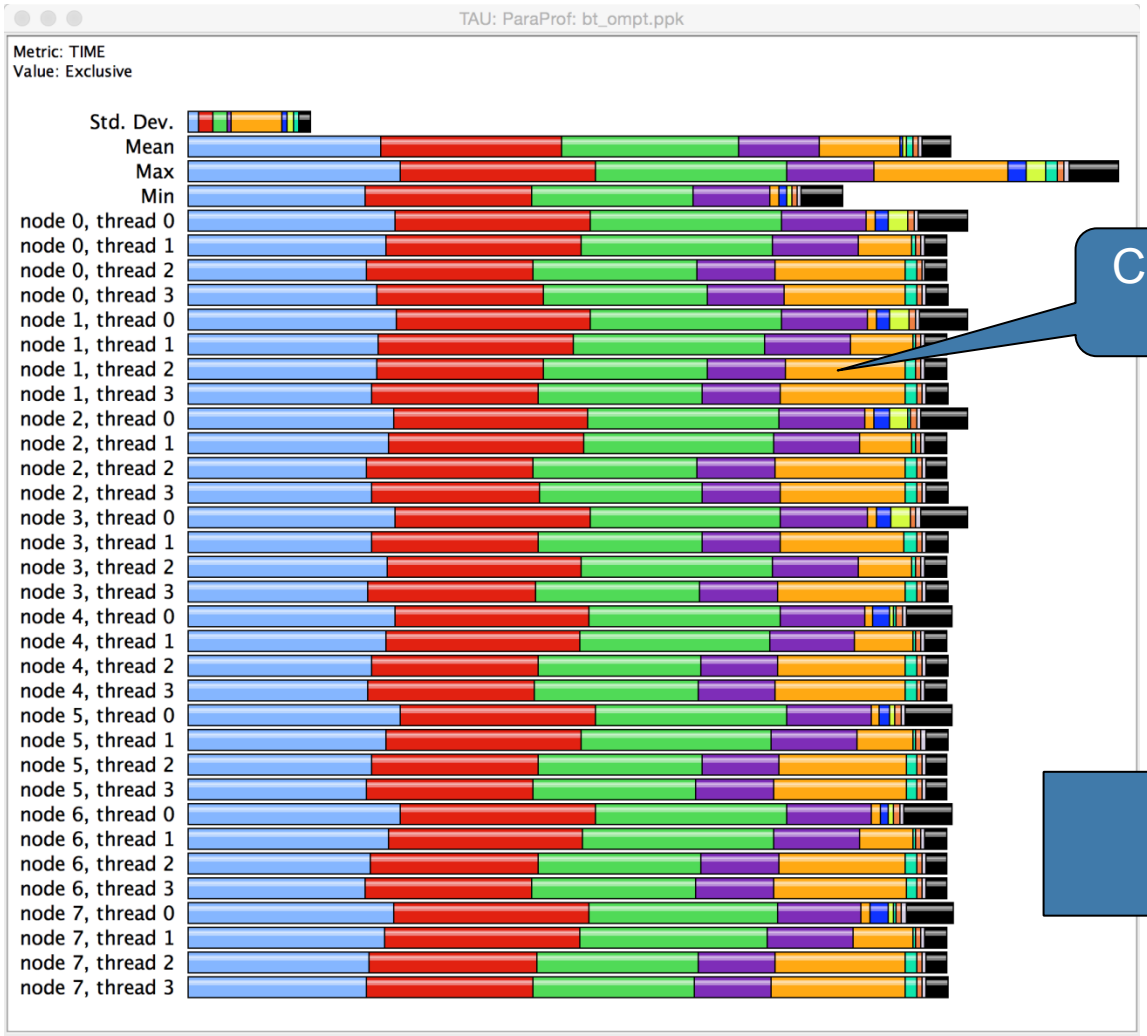
TAU Analysis Tools: paraprof

- Launch paraprof

```
% paraprof
```



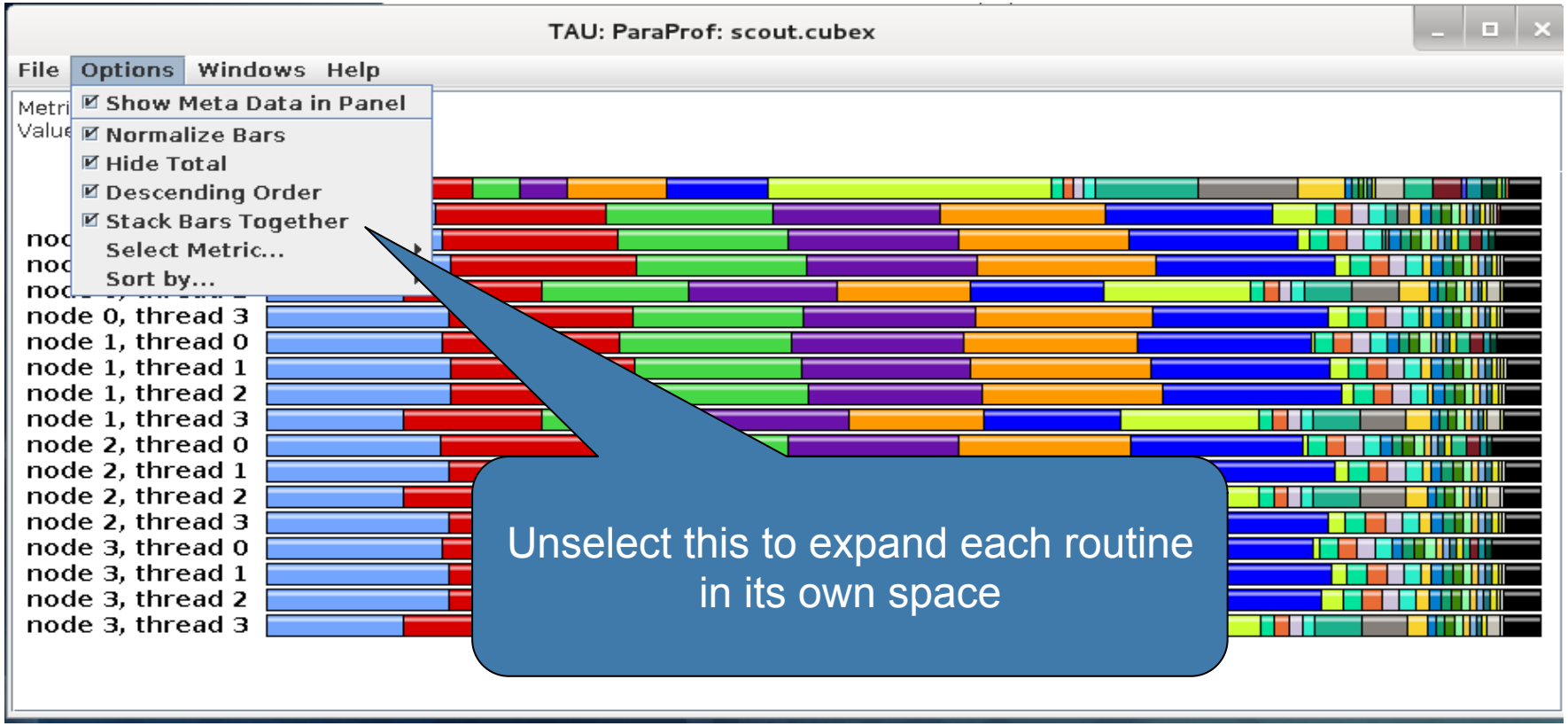
Paraprof main window



Colors represent code regions

Options -> uncheck Stack Bars Together

Paraprof main window



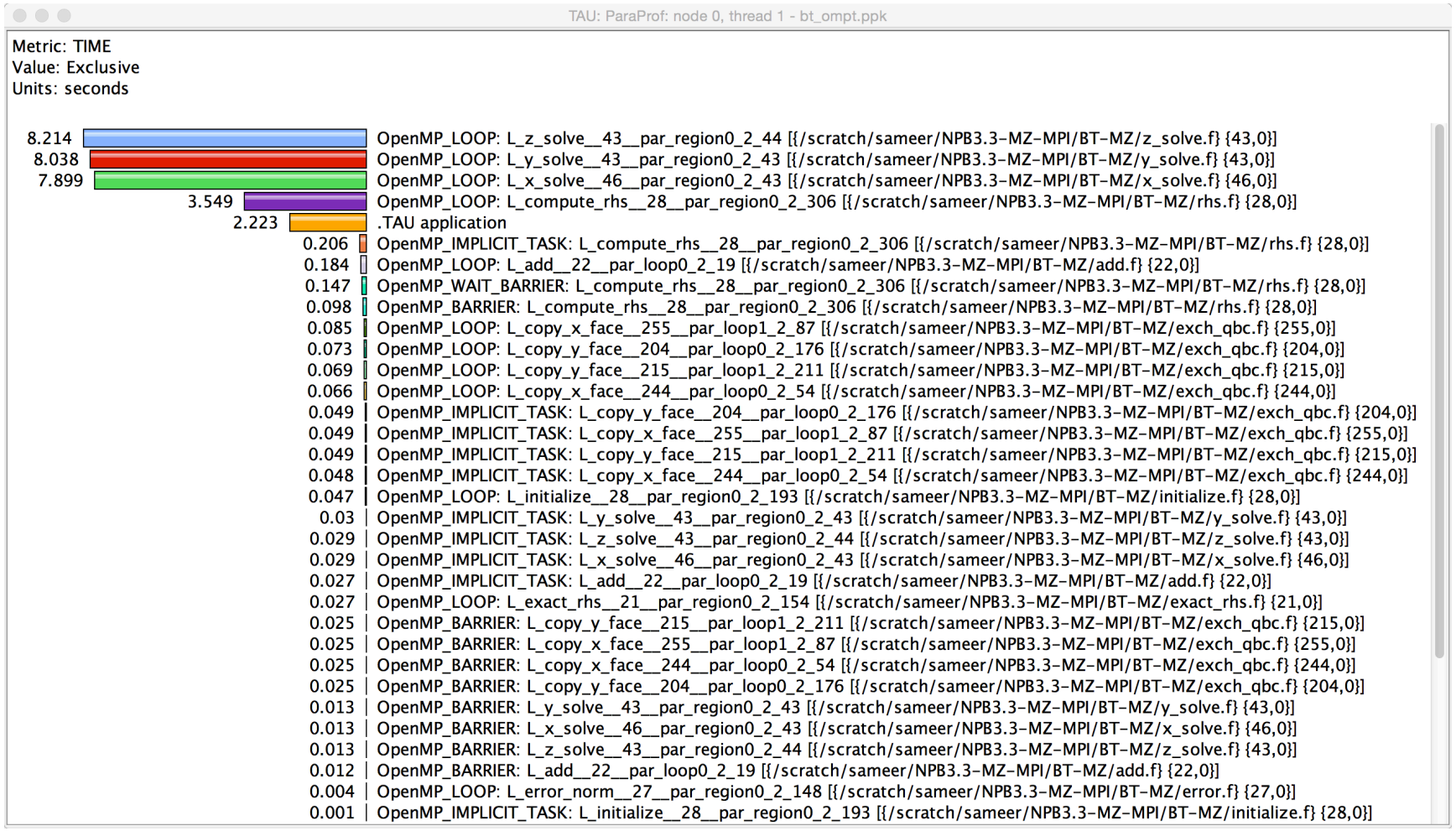
Paraprof main window

Left/right click here

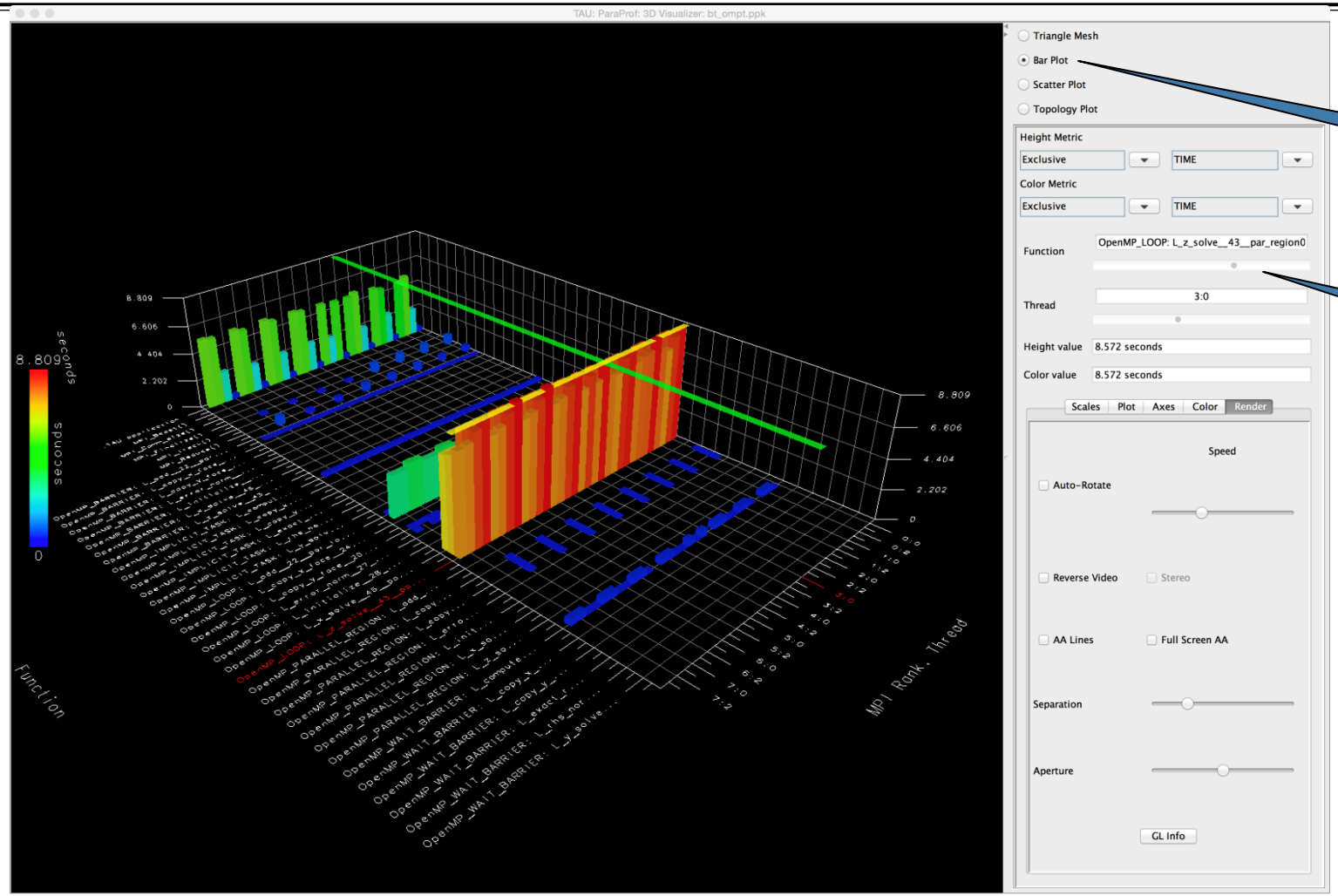
Each routine occupies its own space. Can see the extent of imbalance across all threads.

Paraprof node window (function barchart window)

Exclusive time spent in each code region (OpenMP loop) is shown here for MPI rank 0 thread 1



Paraprof 3D visualization window



Click Bar Plot

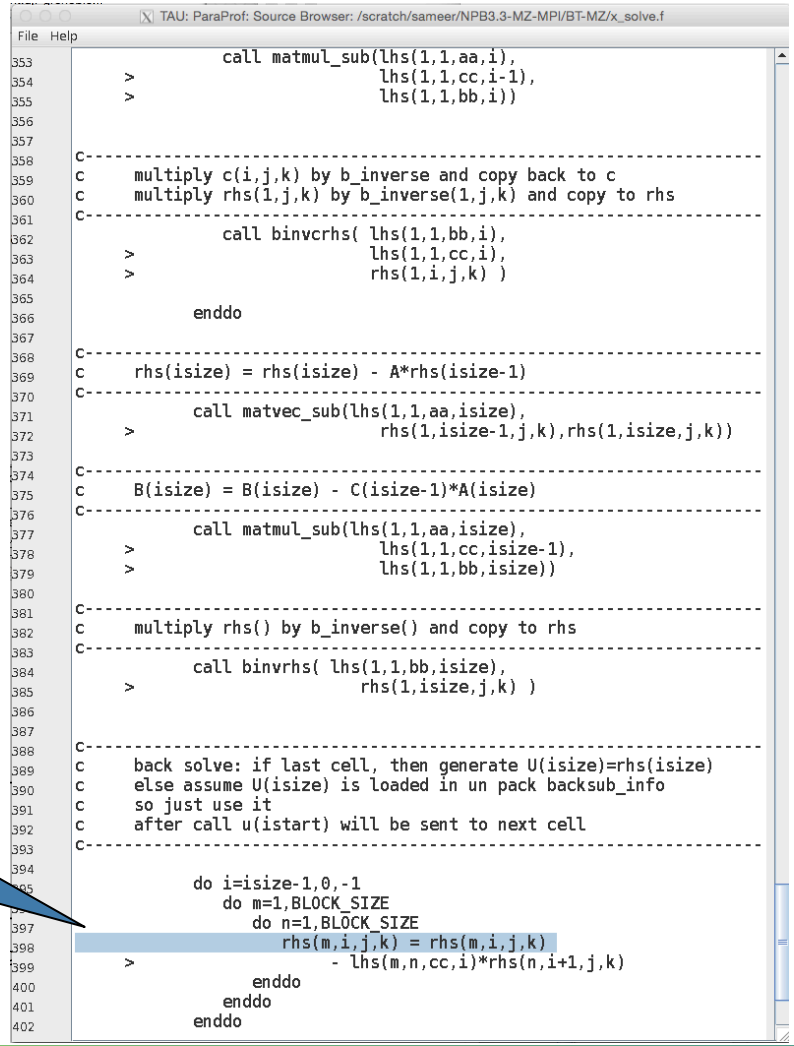
Move Function and Thread Sliders

Windows -> 3D visualization

Paraprof Thread Statistics Table with TAU_SAMPLING=1

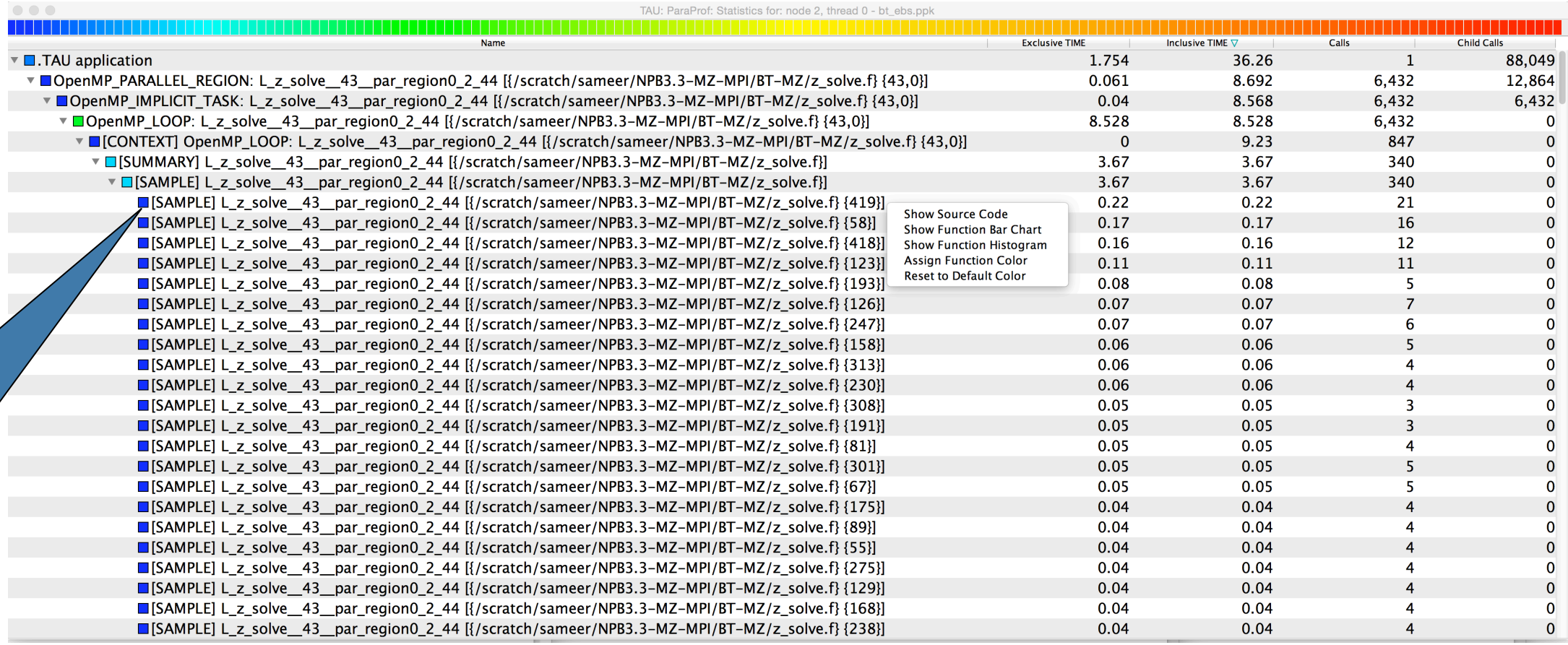
Right click here

Statement Level Profiling with TAU



Source location where samples are taken. Compute intensive region.

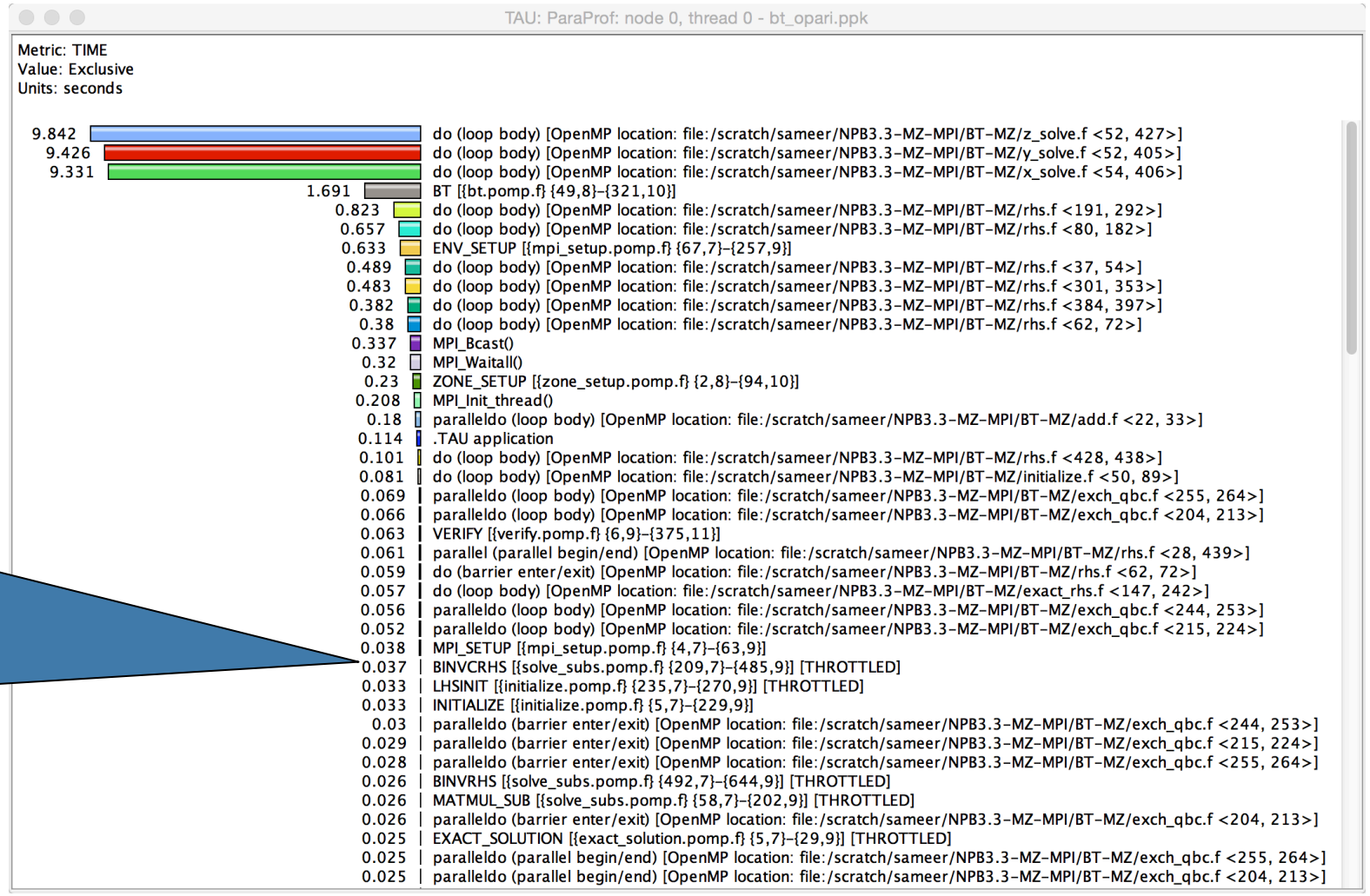
Paraprof Thread Statistics Table



Right click here and choose “Show Source Code” for a sample

Instrumenting Source Code with PDT and Opari

Frequently executing lightweight routines are automatically throttled at runtime. Reduces runtime dilation.



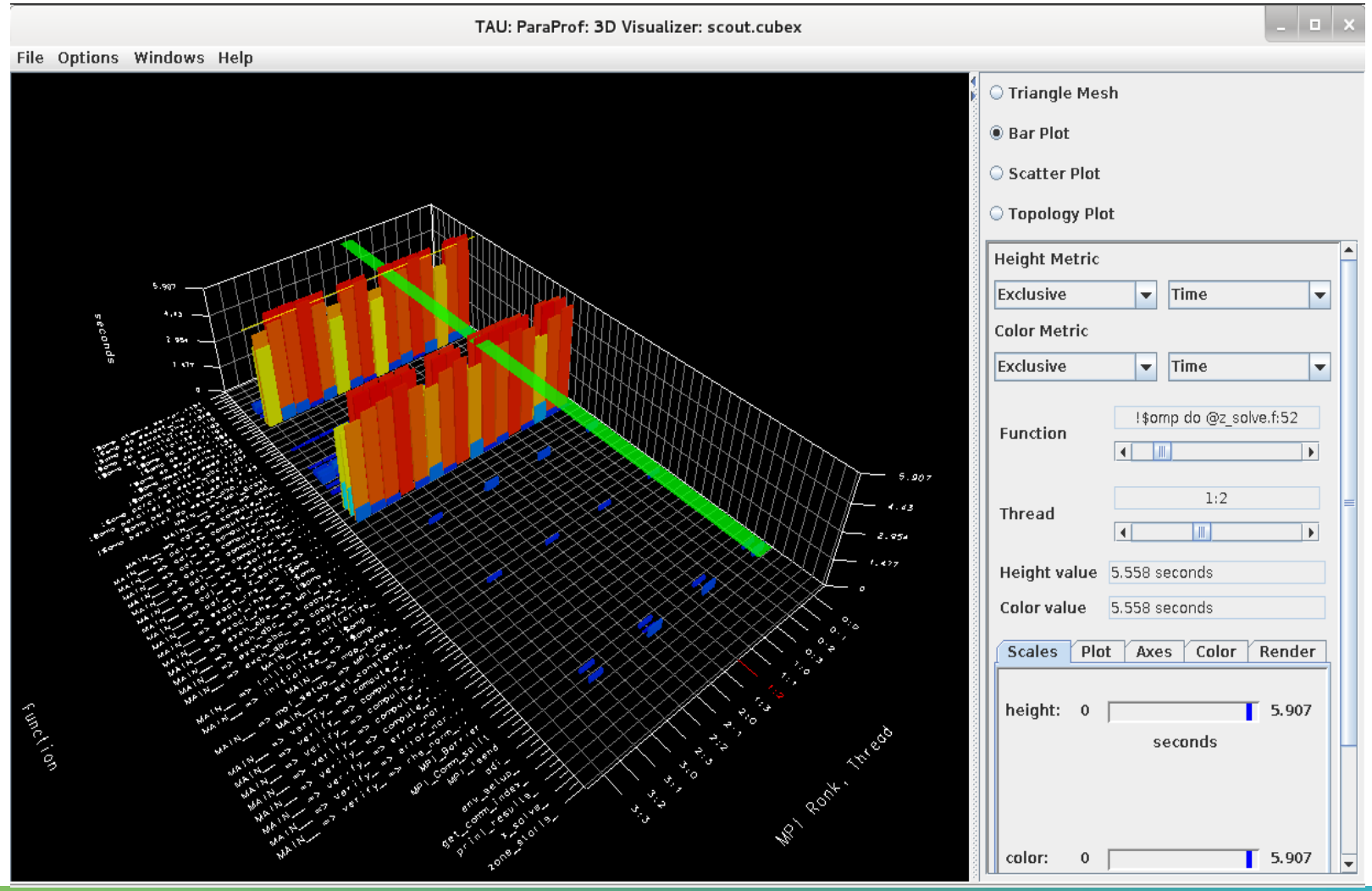
Marker Events in TAU Show Sudden Spikes

TAU: ParaProf: Context Events for: node 2 – rapl_marker_16p.ppk

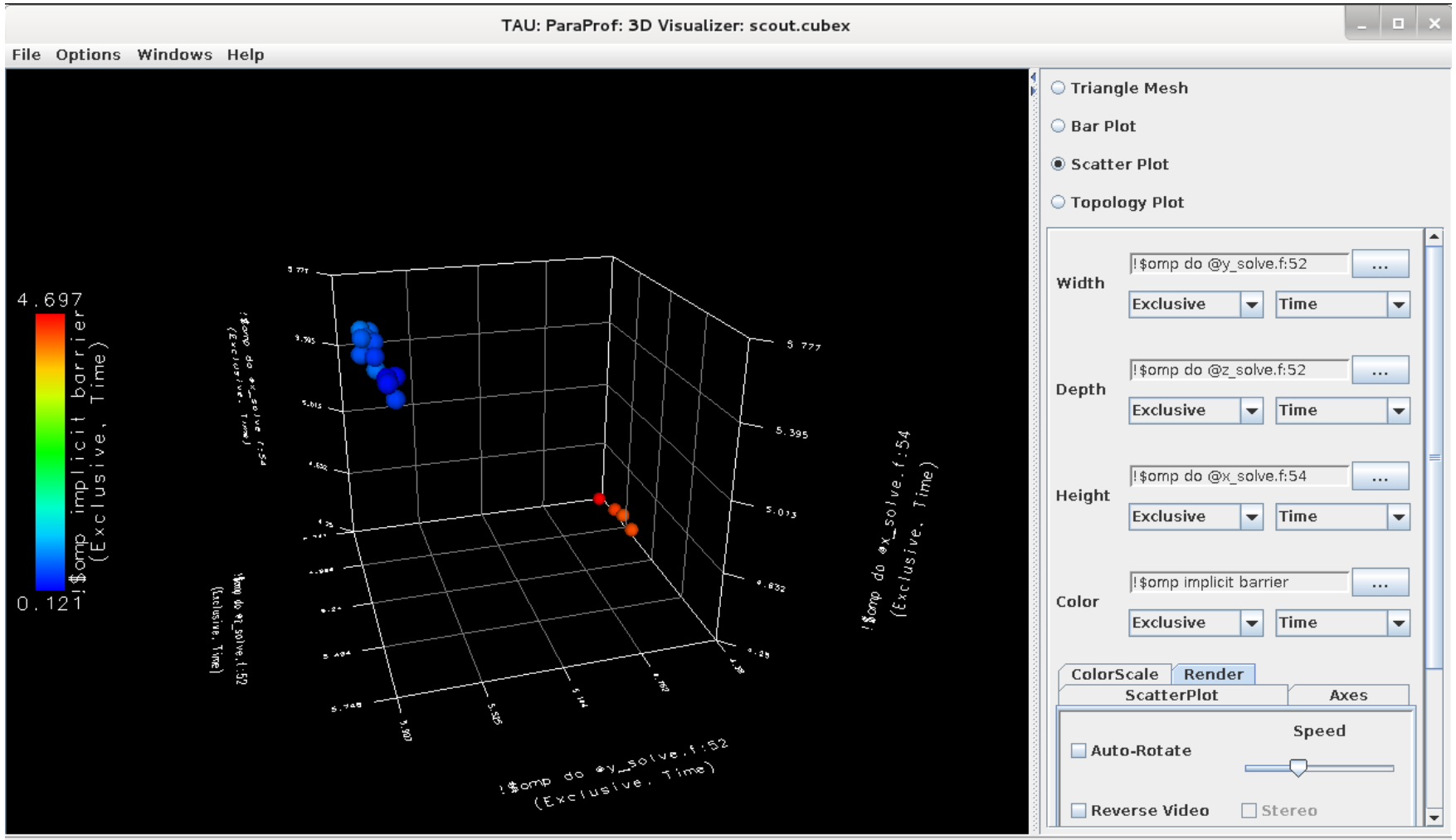
Name	MaxValue	MinValue	NumSamples	MeanValue	Std. Dev.
int main(int, char **) C [{matmult.c} {165,1}–{237,1}]					
double do_work(void) C [{matmult.c} {126,1}–{157,1}]					
void compute(double **, double **, double **, int, int, int) C [{matmult.c} {84,1}–{103,1}]					
[GROUP=MAX_MARKER] rapl:::DRAM_ENERGY:PACKAGE0 (Power in Watts)	17.585	17.469	5	17.521	0.037
[GROUP=MAX_MARKER] rapl:::DRAM_ENERGY:PACKAGE1 (Power in Watts)	15.261	15.218	4	15.237	0.016
[GROUP=MAX_MARKER] rapl:::PACKAGE_ENERGY:PACKAGE0 (Power in Watts)	118.903	114.923	22	116.98	1.201
[GROUP=MAX_MARKER] rapl:::PACKAGE_ENERGY:PACKAGE1 (Power in Watts)	113.466	110.207	22	111.778	0.996
[GROUP=MAX_MARKER] rapl:::PP0_ENERGY:PACKAGE0 (Power in Watts)	100.138	96.266	24	98.206	1.13
[GROUP=MAX_MARKER] rapl:::PP0_ENERGY:PACKAGE1 (Power in Watts)	95.846	92.758	24	94.319	0.937
[GROUP=MIN_MARKER] rapl:::DRAM_ENERGY:PACKAGE0 (Power in Watts)	17.397	17.303	4	17.358	0.035
[GROUP=MIN_MARKER] rapl:::DRAM_ENERGY:PACKAGE1 (Power in Watts)	15.048	15.042	2	15.045	0.003
int mysleep(int) C [{matmult.c} {46,1}–{49,1}]					
[GROUP=MIN_MARKER] rapl:::DRAM_ENERGY:PACKAGE0 (Power in Watts)	15.84	15.84	1	15.84	0
[GROUP=MIN_MARKER] rapl:::DRAM_ENERGY:PACKAGE1 (Power in Watts)	14.275	14.275	1	14.275	0
[GROUP=MIN_MARKER] rapl:::PACKAGE_ENERGY:PACKAGE1 (Power in Watts)	96.853	96.853	1	96.853	0
[GROUP=MIN_MARKER] rapl:::PACKAGE_ENERGY:PACKAGE0 (Power in Watts)	93.125	93.125	1	93.125	0
[GROUP=MIN_MARKER] rapl:::PP0_ENERGY:PACKAGE0 (Power in Watts)	75.096	75.096	1	75.096	0
[GROUP=MIN_MARKER] rapl:::PP0_ENERGY:PACKAGE1 (Power in Watts)	79.646	79.646	1	79.646	0
void compute_interchange(double **, double **, double **, int, int, int) C [{matmult.c} {105,1}–{124,1}]					
[GROUP=MAX_MARKER] rapl:::DRAM_ENERGY:PACKAGE0 (Power in Watts)	26.064	25.711	2	25.887	0.176
[GROUP=MAX_MARKER] rapl:::DRAM_ENERGY:PACKAGE1 (Power in Watts)	24.373	23.965	4	24.232	0.159
[GROUP=MAX_MARKER] rapl:::PACKAGE_ENERGY:PACKAGE0 (Power in Watts)	126.872	125.182	6	125.732	0.557
[GROUP=MAX_MARKER] rapl:::PACKAGE_ENERGY:PACKAGE1 (Power in Watts)	124.377	116.689	5	122.428	2.885
[GROUP=MAX_MARKER] rapl:::PP0_ENERGY:PACKAGE0 (Power in Watts)	103.981	102.21	6	102.769	0.584
[GROUP=MAX_MARKER] rapl:::PP0_ENERGY:PACKAGE1 (Power in Watts)	102.615	101.693	4	102.115	0.33
rapl:::DRAM_ENERGY:PACKAGE0 (Power in Watts)	26.064	15.84	36	19.053	3.39
rapl:::DRAM_ENERGY:PACKAGE1 (Power in Watts)	24.373	14.275	36	16.435	3.155
rapl:::PACKAGE_ENERGY:PACKAGE0 (Power in Watts)	126.872	93.125	36	117.729	5.403
rapl:::PACKAGE_ENERGY:PACKAGE1 (Power in Watts)	124.377	96.853	36	112.961	4.776
rapl:::PP0_ENERGY:PACKAGE0 (Power in Watts)	103.981	75.096	36	98.208	4.466
rapl:::PP0_ENERGY:PACKAGE1 (Power in Watts)	102.615	79.646	36	94.872	3.662

```
% export TAU_EVENT_THRESHOLD 0.5
```

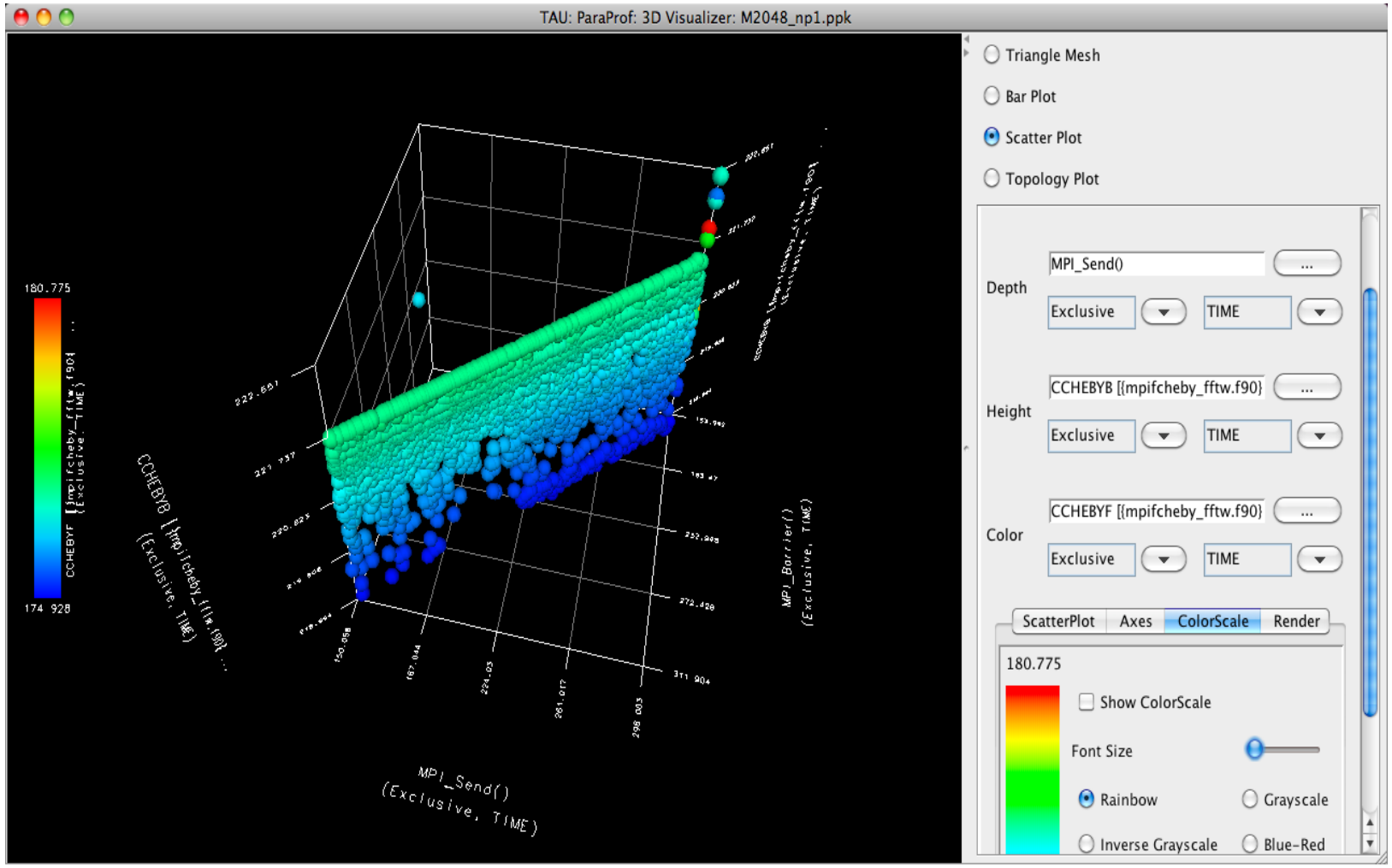
ParaProf: 3D Visualization Window



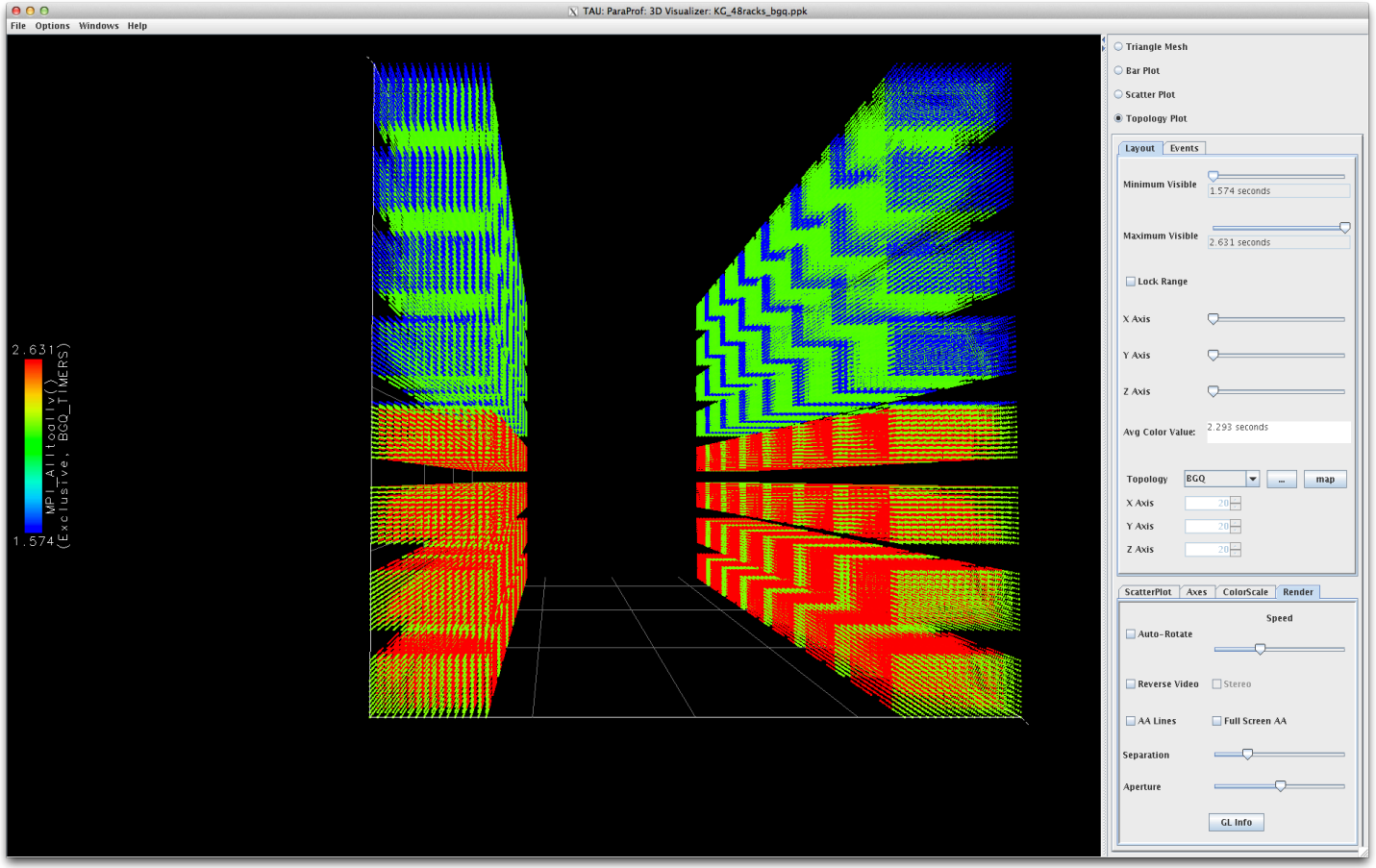
ParaProf: 3D Scatter Plot



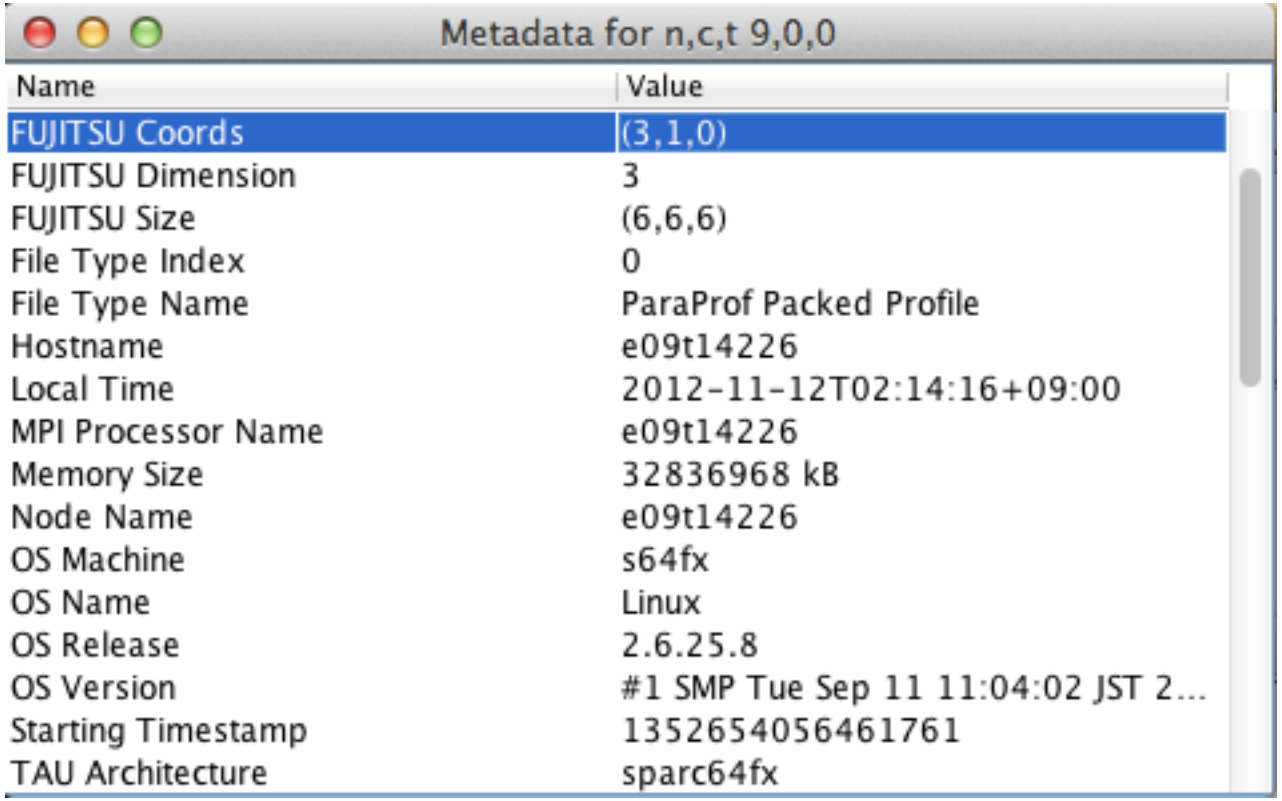
ParaProf: Scatter Plot



ParaProf Topology View: Torus

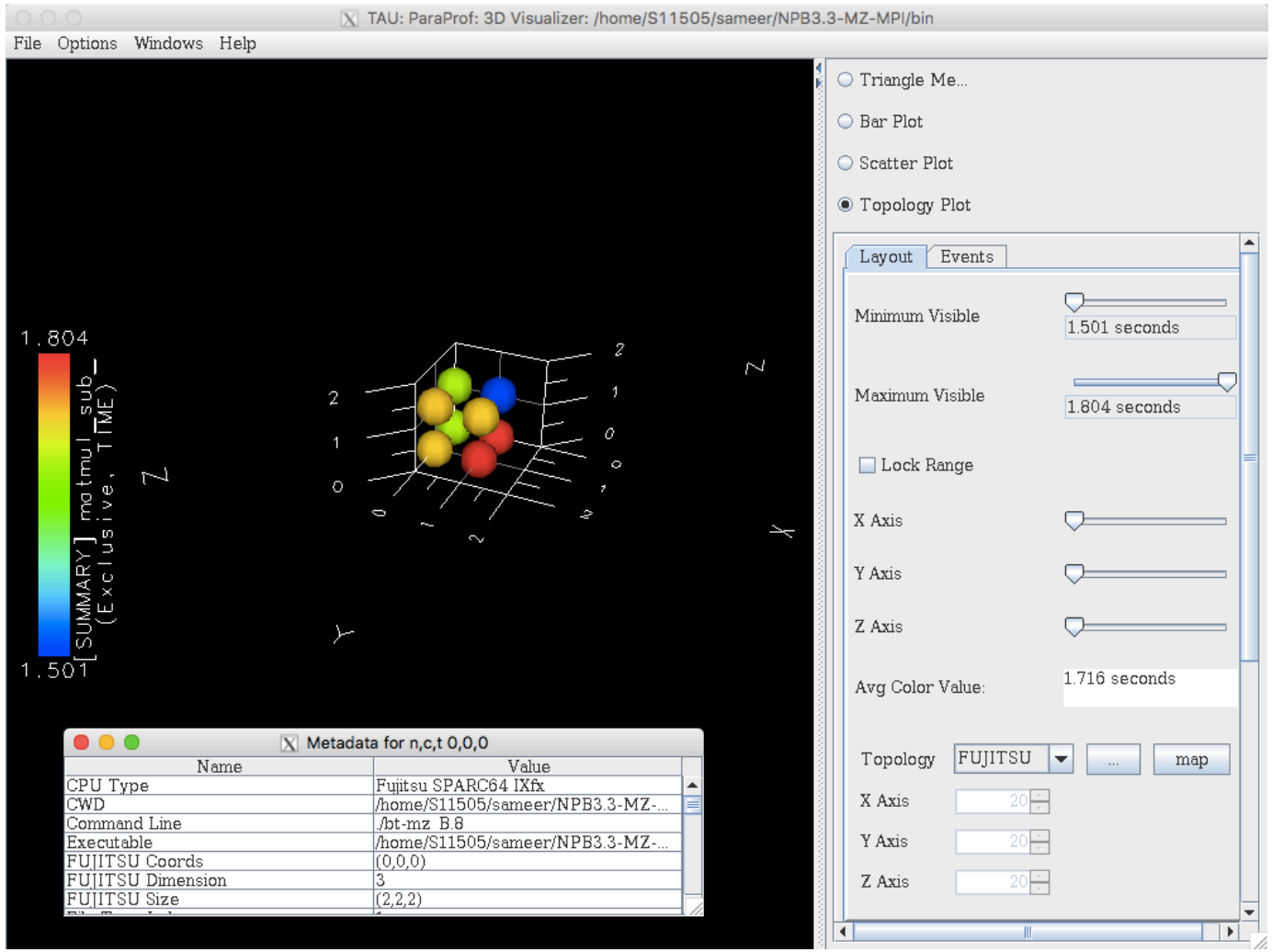


ParaProf MetaData: FX10




```
% pjsub –interact –L node="6x6x6"
% mpiexec –n 216 ./a.out
```

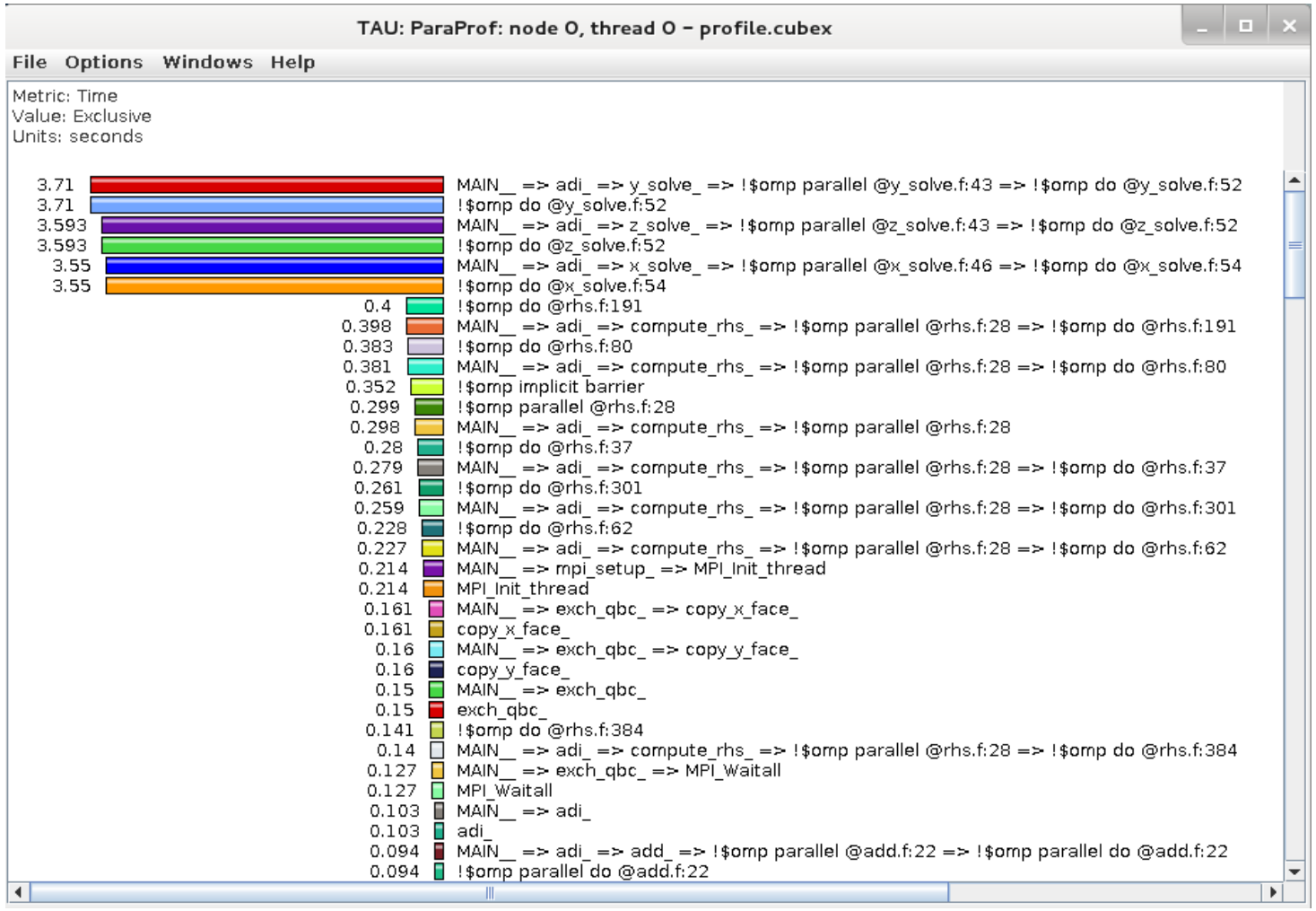
ParaProf 3D Topology Display for Fujitsu FX10



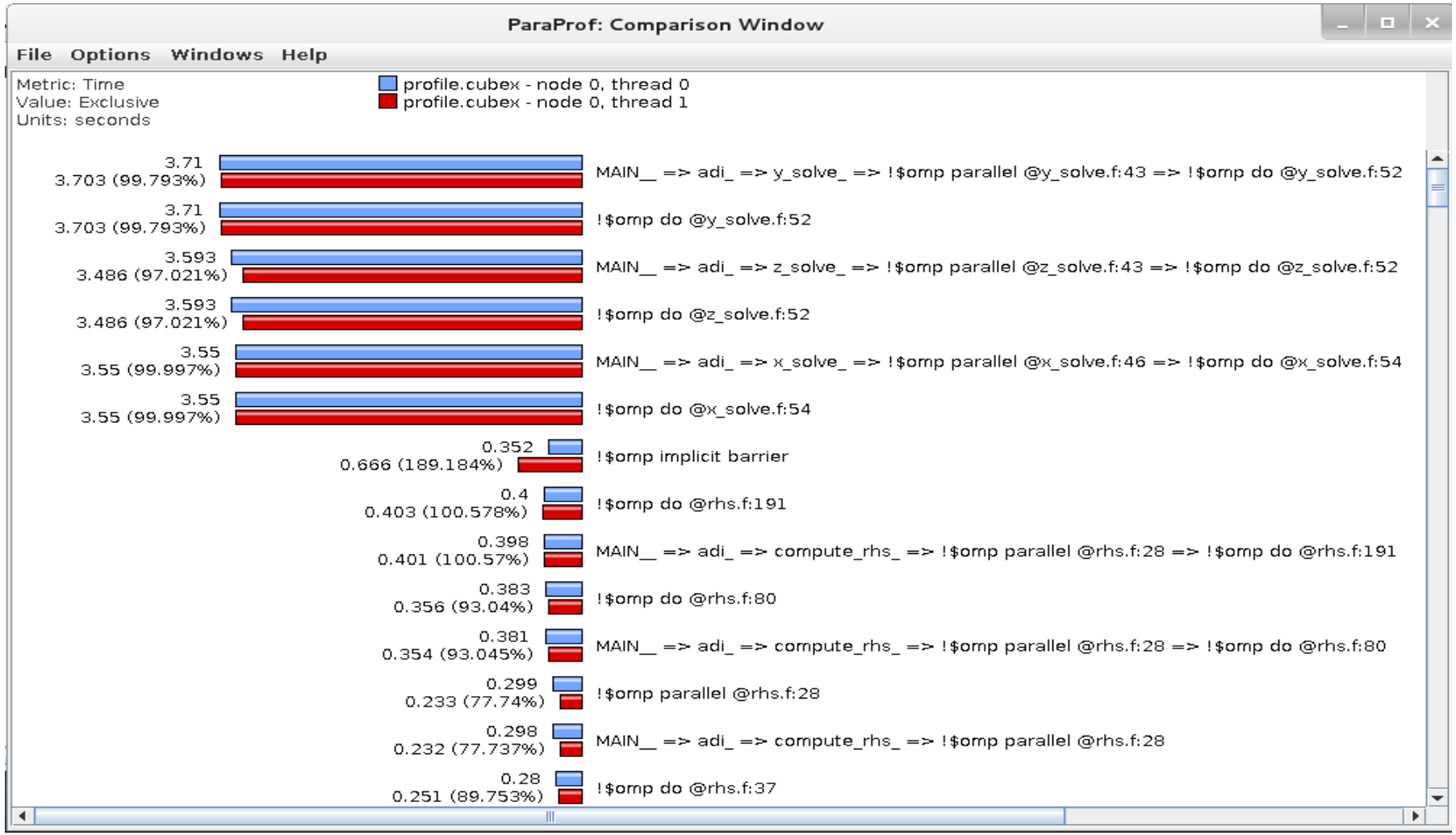
#PJSUB –L node="2x2x2"

mpiexec –n 8 tau_exec –ebs $EXE

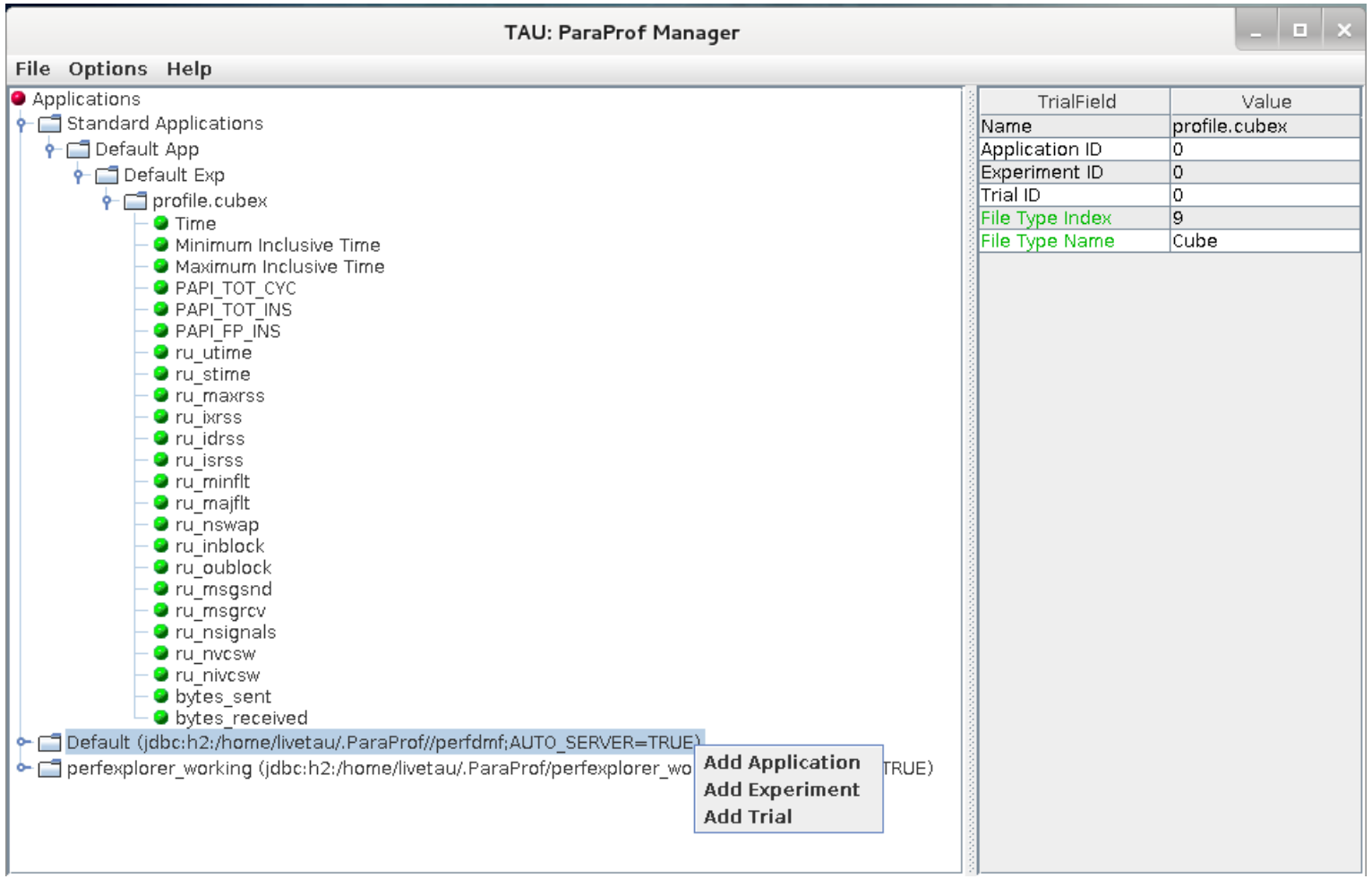
ParaProf: Node View



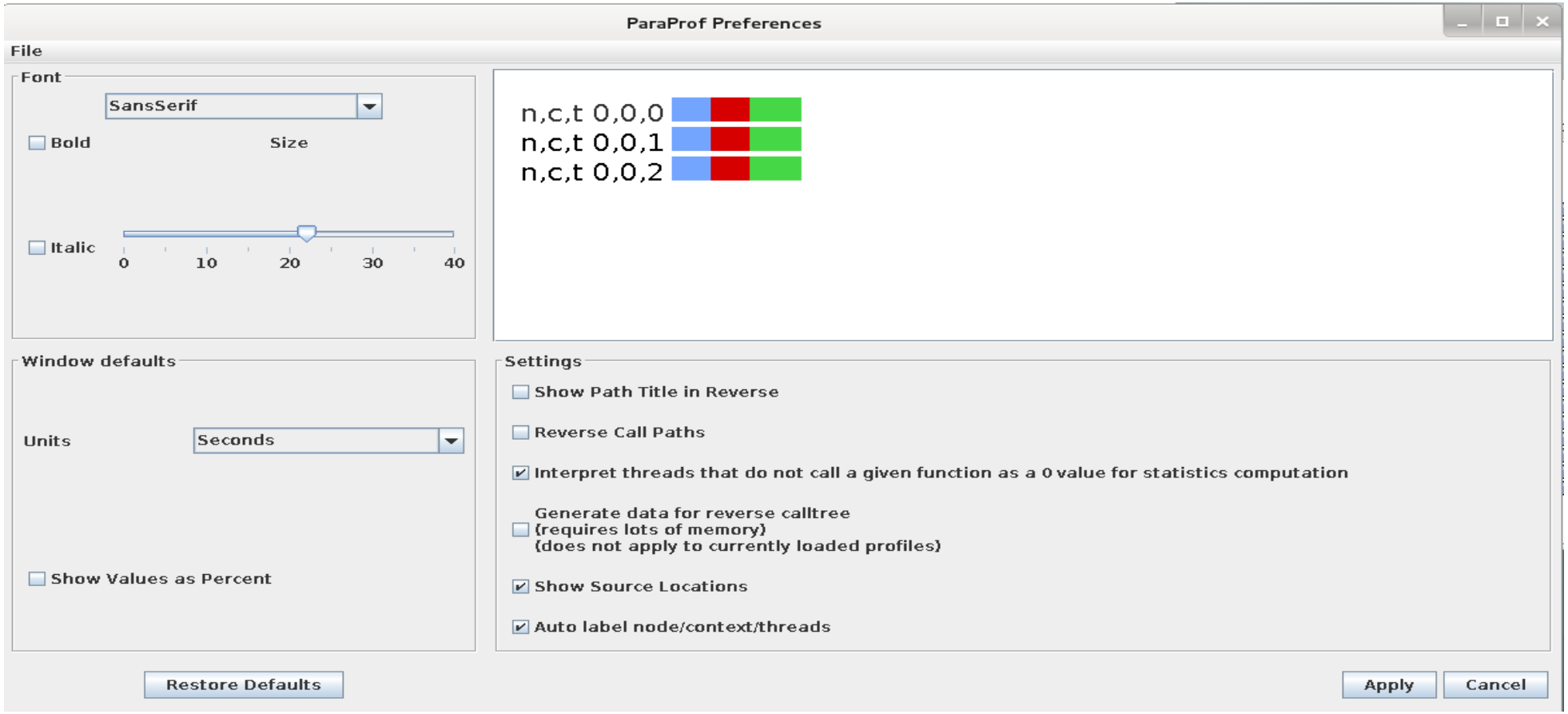
ParaProf: Add Thread to Comparison Window



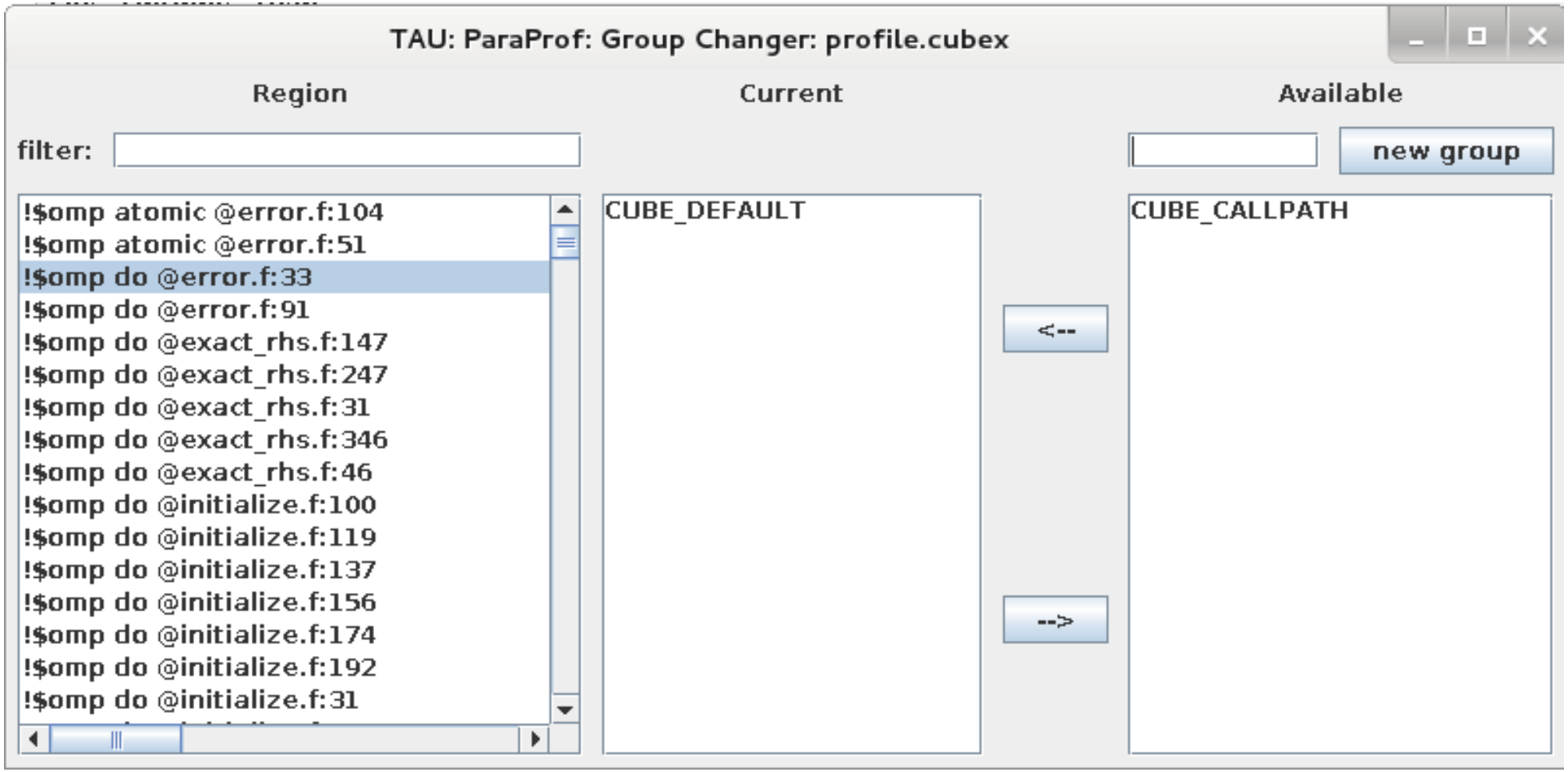
ParaProf: Score-P Profile Files, Database



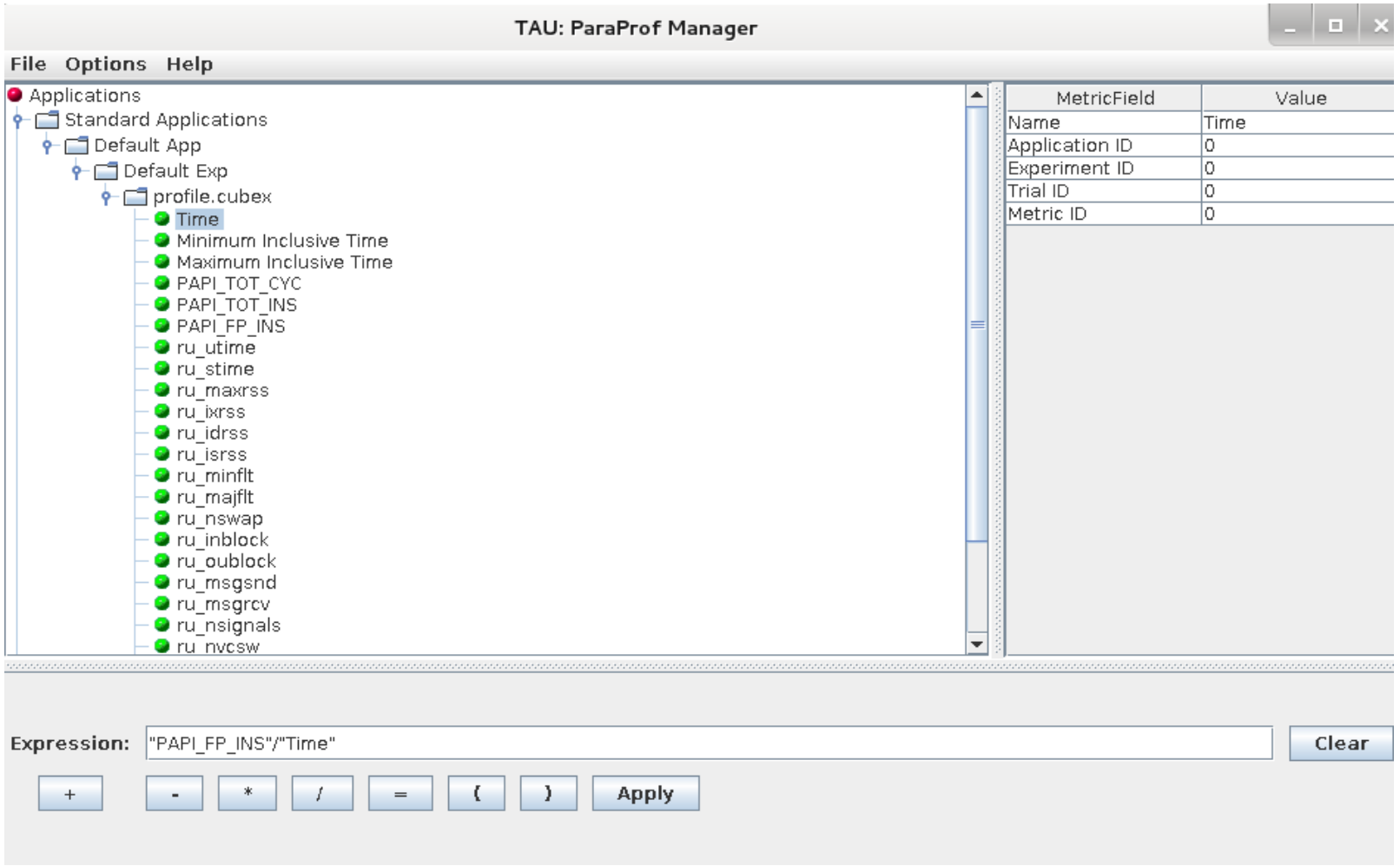
ParaProf: File Preferences Window



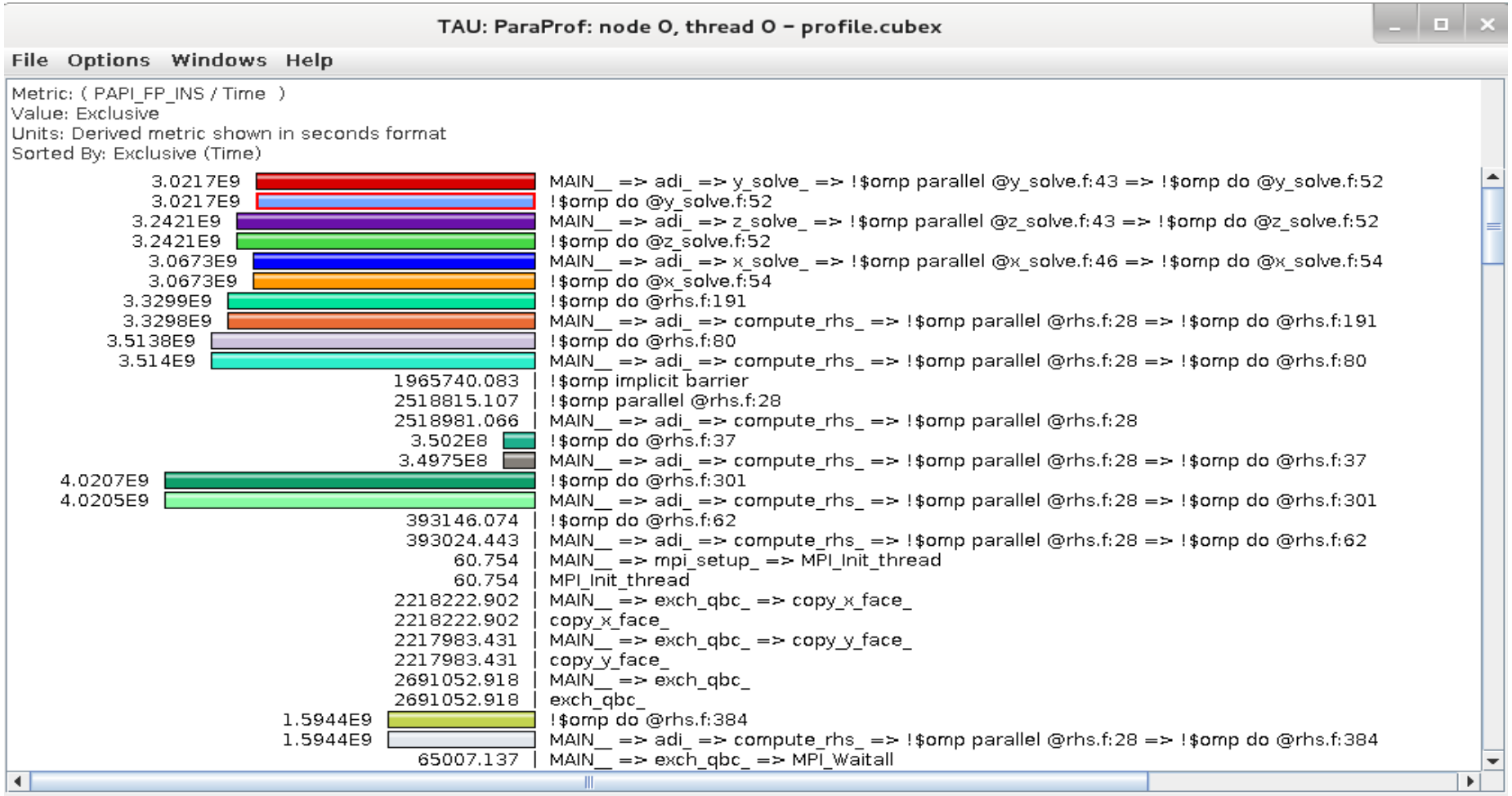
ParaProf: Group Changer Window



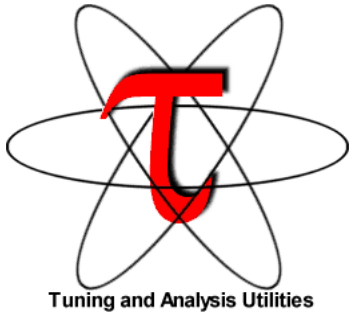
ParaProf: Derived Metric Panel in Manager Window



Sorting Derived FLOPS Metric by Exclusive Time

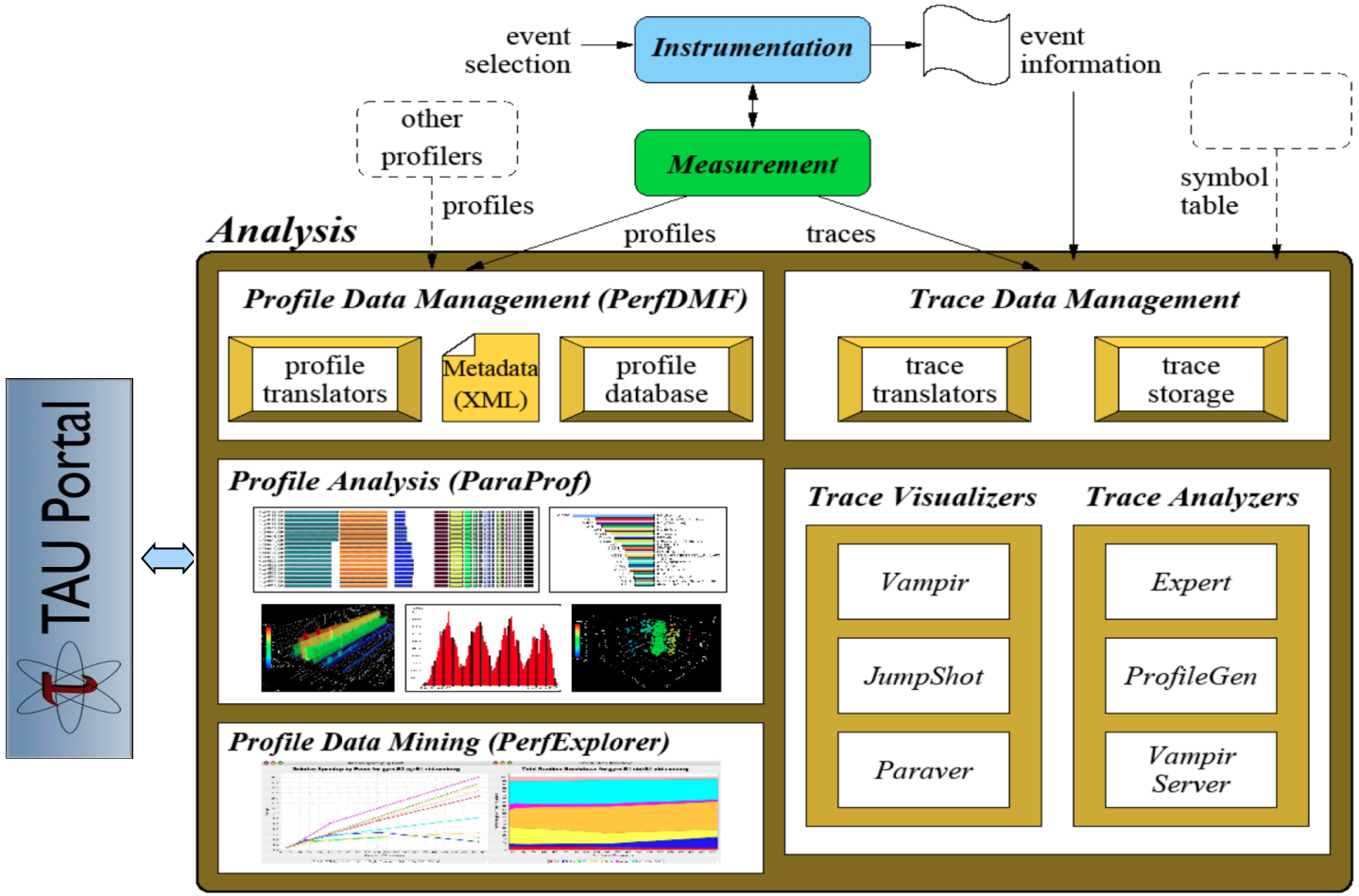


Performance Data Management with TAU PerfExplorer

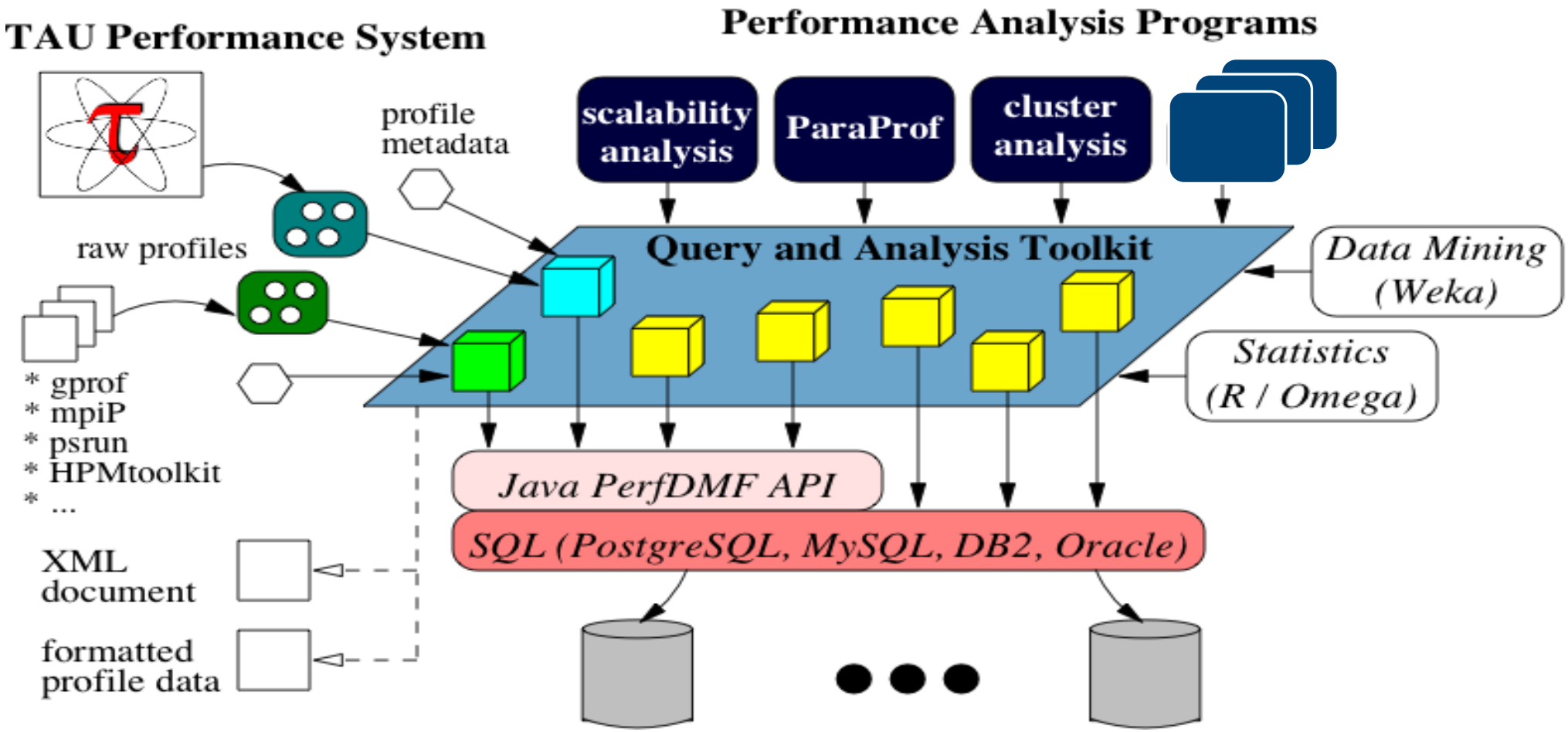


Sameer Shende
sameer@cs.uoregon.edu
Performance Research Laboratory
University of Oregon
http://tau.uoregon.edu

TAU Analysis



TAUdb: Performance Data Management Framework



Using TAUdb

- Configure TAUdb (Done by each user)

  % taudb_configure --create-default
  - Choose derby, PostgreSQL, MySQL, Oracle or DB2
  - Hostname
  - Username
  - Password
  - Say yes to downloading required drivers (we are not allowed to distribute these)
  - Stores parameters in your ~/.ParaProf/taudb.cfg file
- Configure PerfExplorer (Done by each user)

  % perfexplorer_configure
- Execute PerfExplorer

  % perfexplorer

Using PerfExplorer

```
% wget http://tau.uoregon.edu/data.tgz  (Contains CUBE profiles from Score-P)
% taudb_configure --create-default
(Chooses derby, blank user/passwd, yes to save passwd, defaults)
% perfexplorer_configure
(Yes to load schema, defaults)
% paraprof
(load each trial: DB -> Add Trial -> Type (Paraprof Packed Profile) -> OK)
OR use taudb_loadtrial -a "app" -x "experiment" -n "name" file.ppk
Then,
% tar zxf $TAU/data.tgz; cd data/tau;
% taudb_loadtrial -a BT_MZ -x "Class_B" bt-mz_B.*.ppk
% perfexplorer
(Select experiment, Menu: Charts -> Speedup)
```

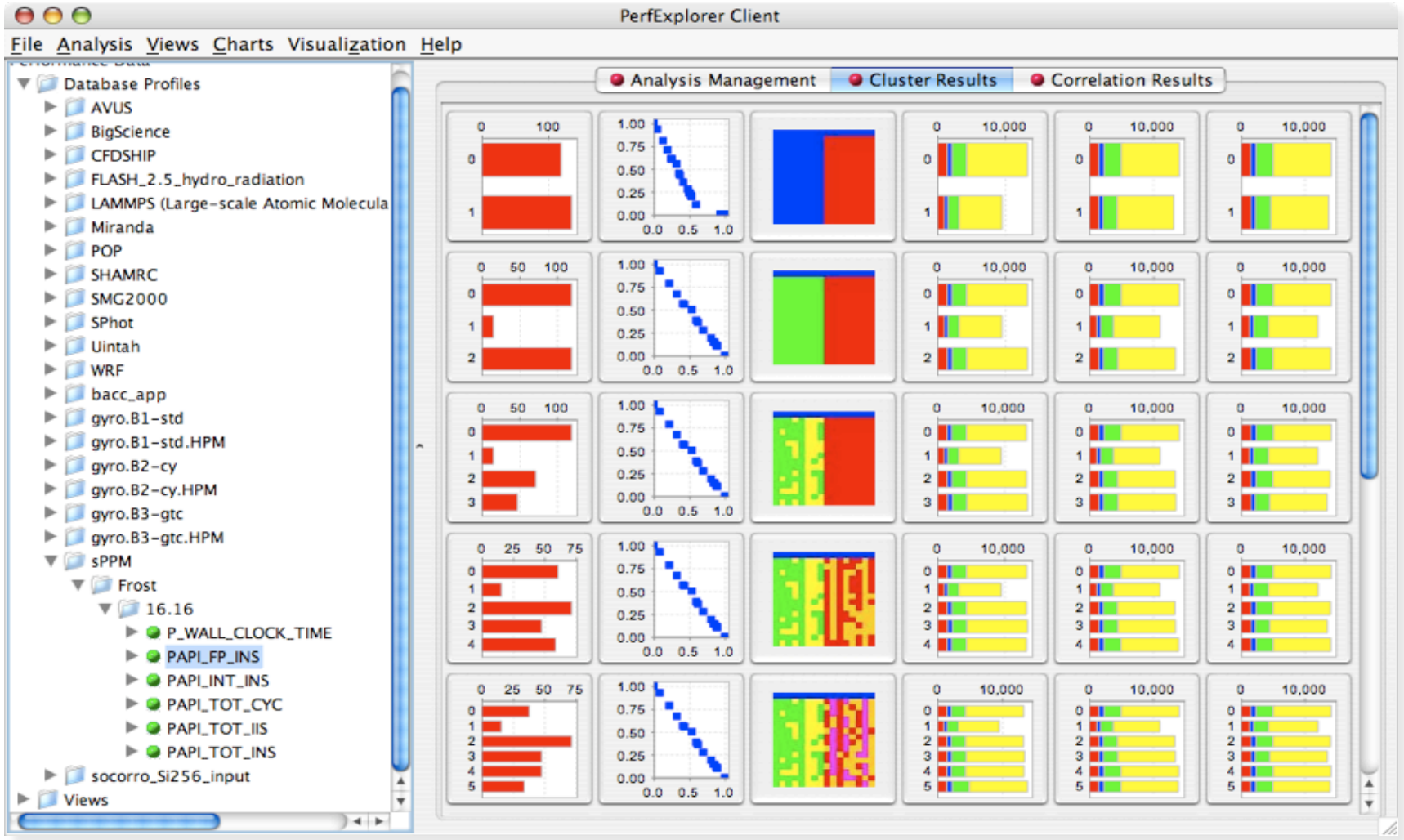
Performance Data Mining (PerfExplorer)

- Performance knowledge discovery framework
  - Data mining analysis applied to parallel performance data
    - comparative, clustering, correlation, dimension reduction, …
  - Use the existing TAU infrastructure
    - TAU performance profiles, taudb
  - Client-server based system architecture
- Technology integration
  - Java API and toolkit for portability
  - taudb
  - R-project/Omegahat, Octave/Matlab statistical analysis
  - WEKA data mining package
  - JFreeChart for visualization, vector output (EPS, SVG)

PerfExplorer: Using Cluster Analysis

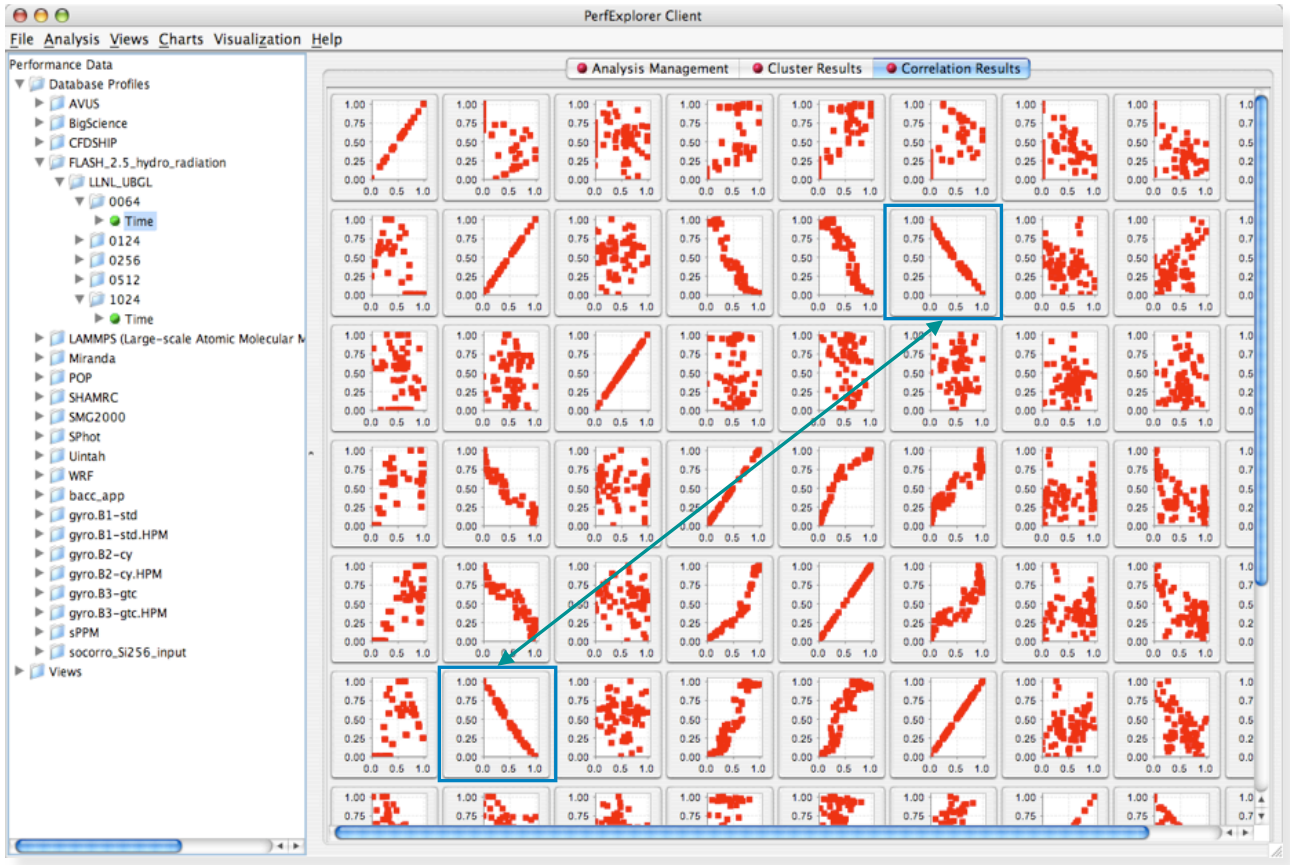
- Performance data represented as vectors - each dimension is the cumulative time for an event
- *k*-means: *k* random centers are selected and instances are grouped with the "closest" (Euclidean) center
- New centers are calculated and the process repeated until stabilization or max iterations
- Dimension reduction necessary for meaningful results
- Virtual topology, summaries constructed

PerfExplorer - Cluster Analysis (sPPM)



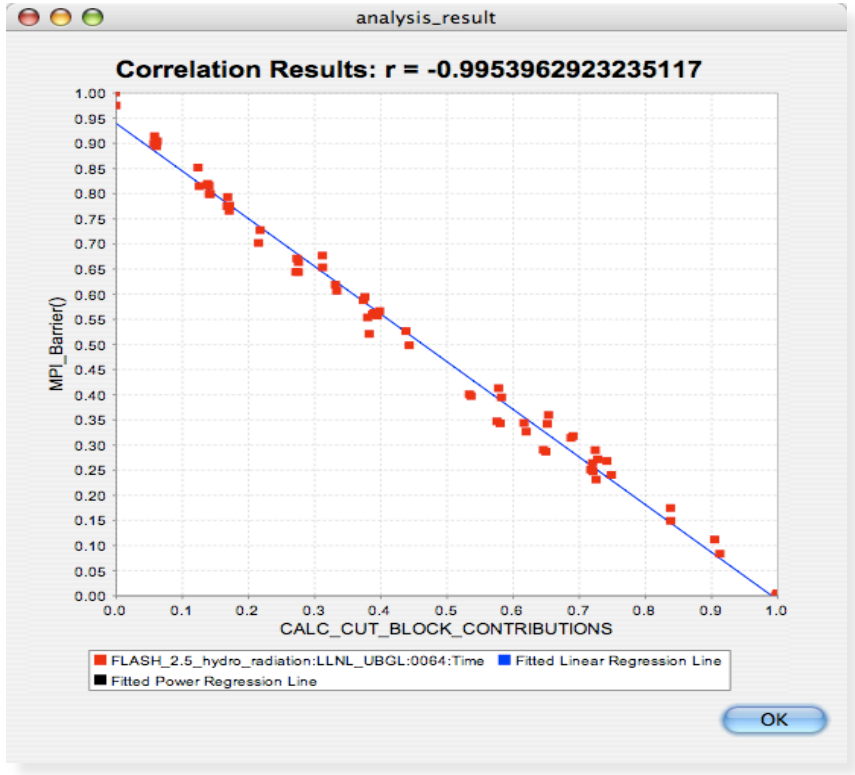
PerfExplorer - Correlation Analysis (Flash)

- Describes strength and direction of a linear relationship between two variables (events) in the data



PerfExplorer - Correlation Analysis (Flash)

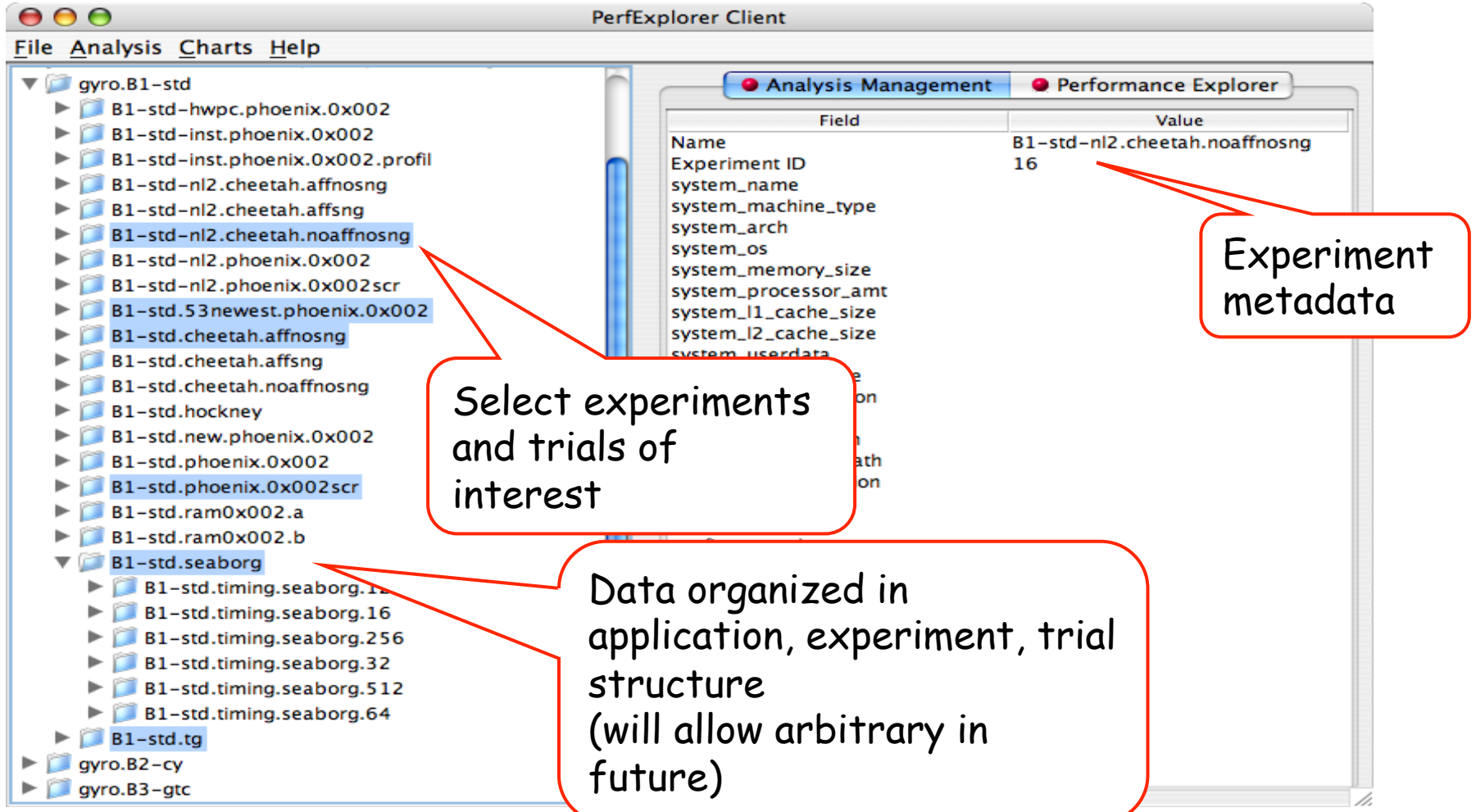
- -0.995 indicates strong, negative relationship
- As CALC_CUT_BLOCK_CONTRIBUTIONS() increases in execution time, MPI_Barrier() decreases



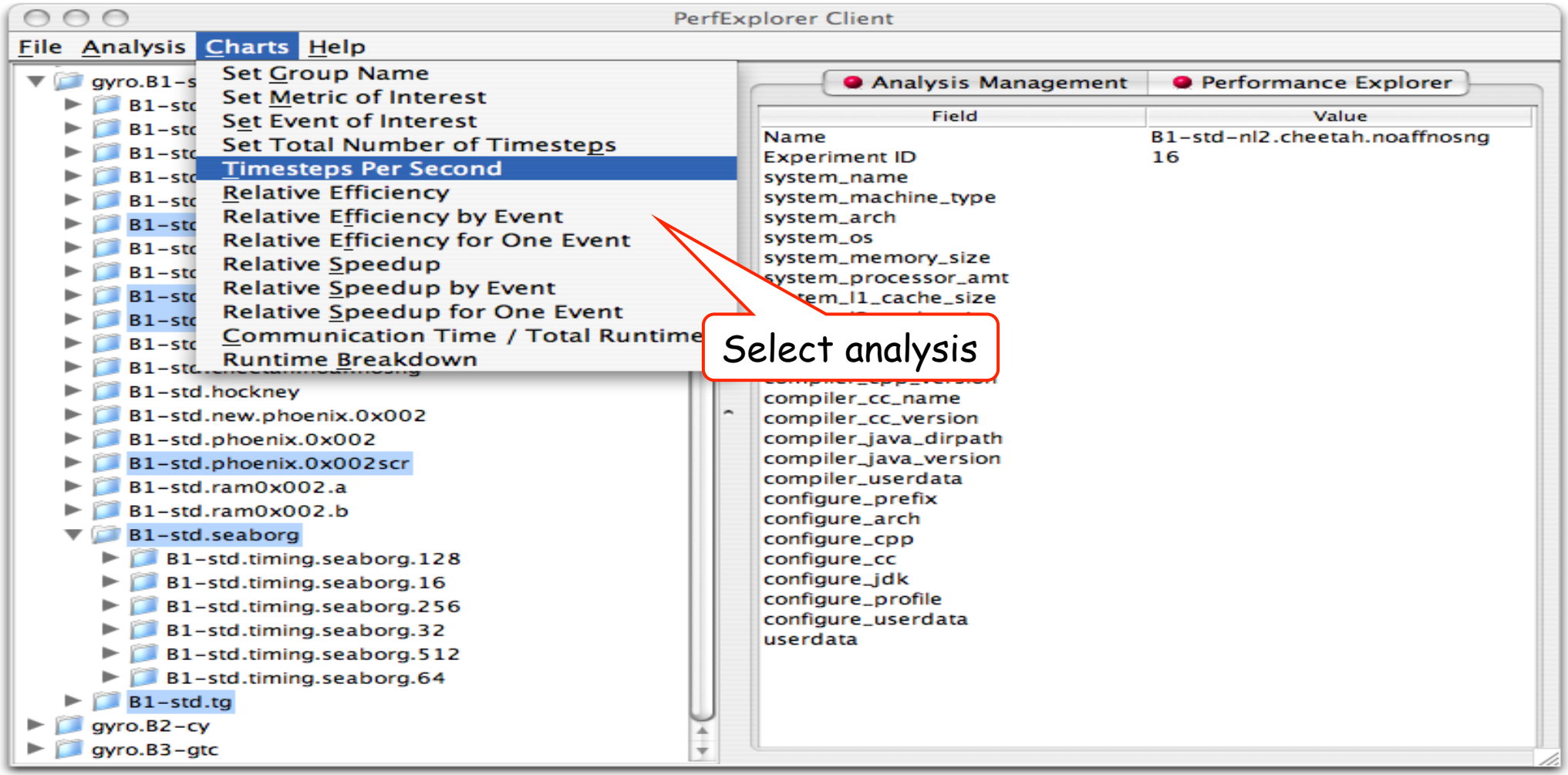
PerfExplorer - Comparative Analysis

- Relative speedup, efficiency
  - total runtime, by event, one event, by phase
- Breakdown of total runtime
- Group fraction of total runtime
- Correlating events to total runtime
- Timesteps per second

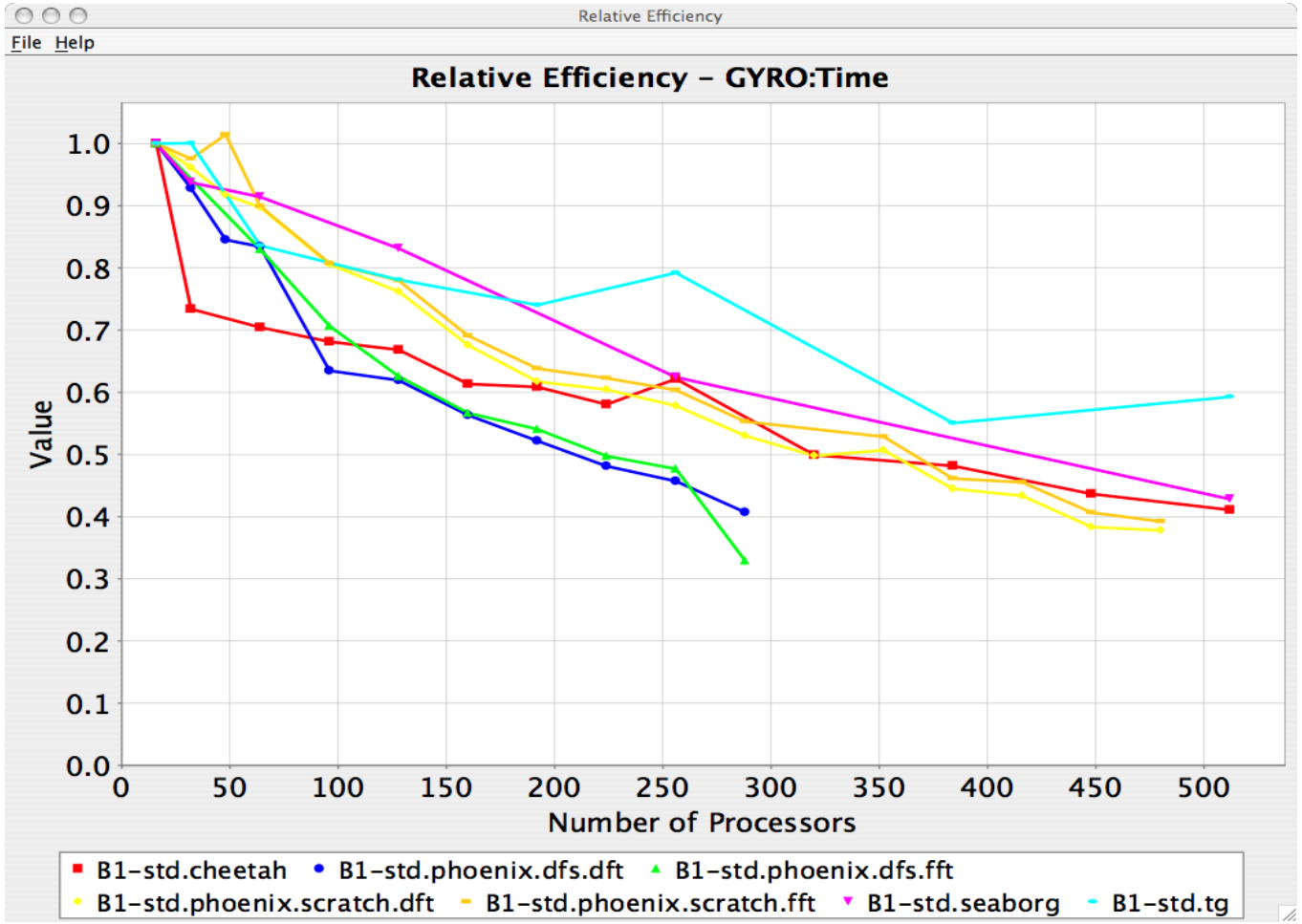
PerfExplorer - Interface



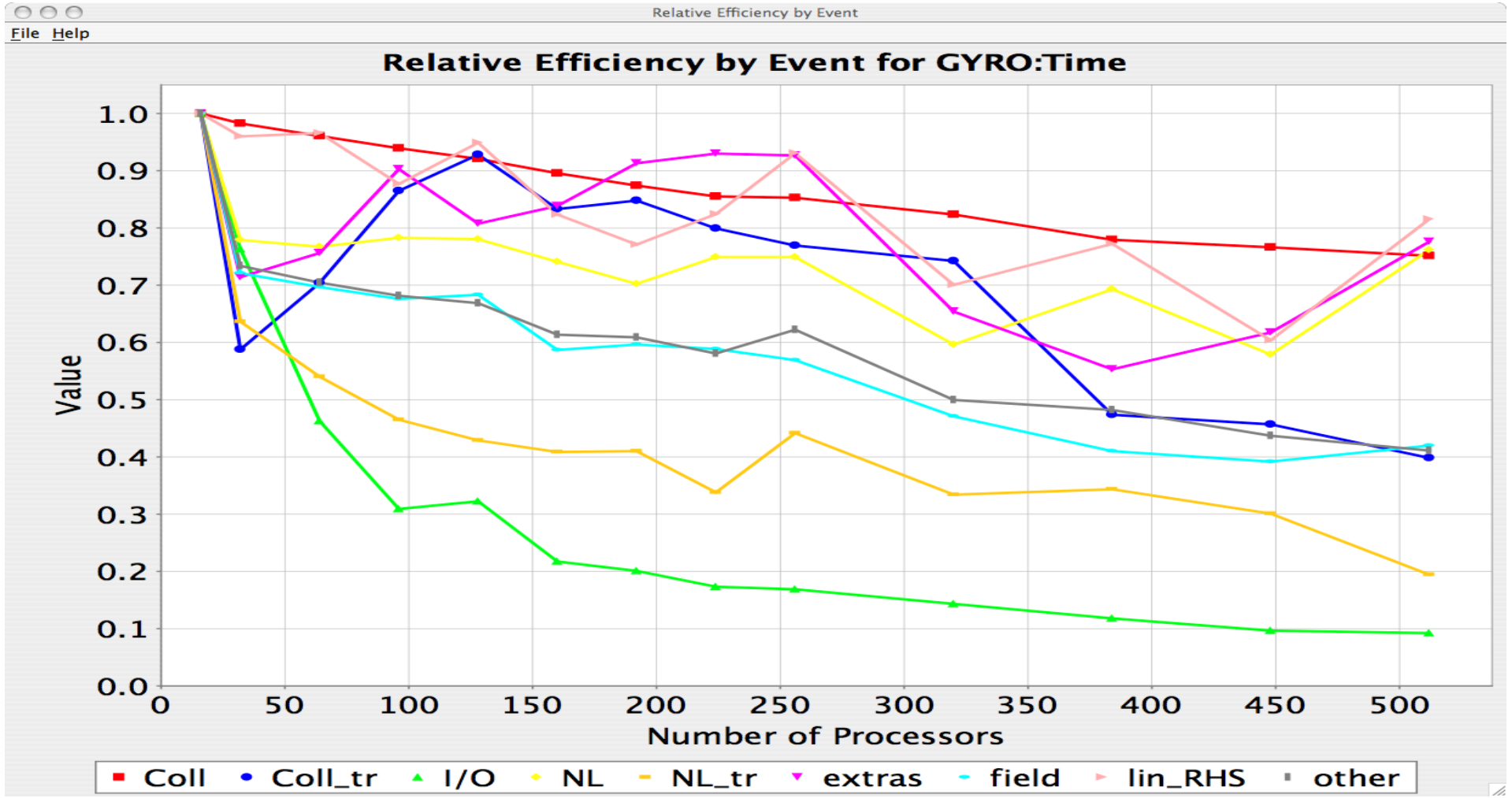
PerfExplorer - Interface



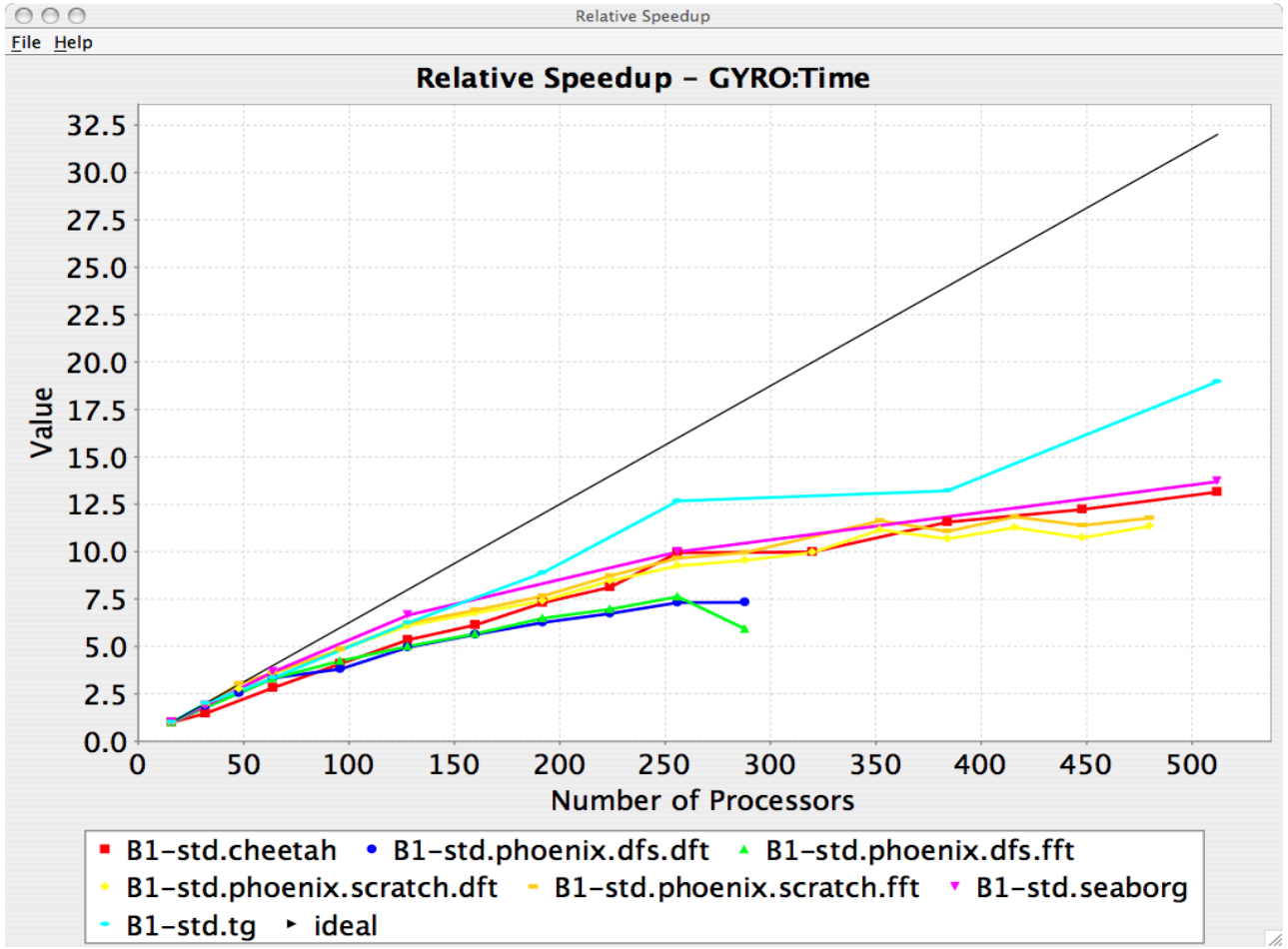
PerfExplorer - Relative Efficiency Plots



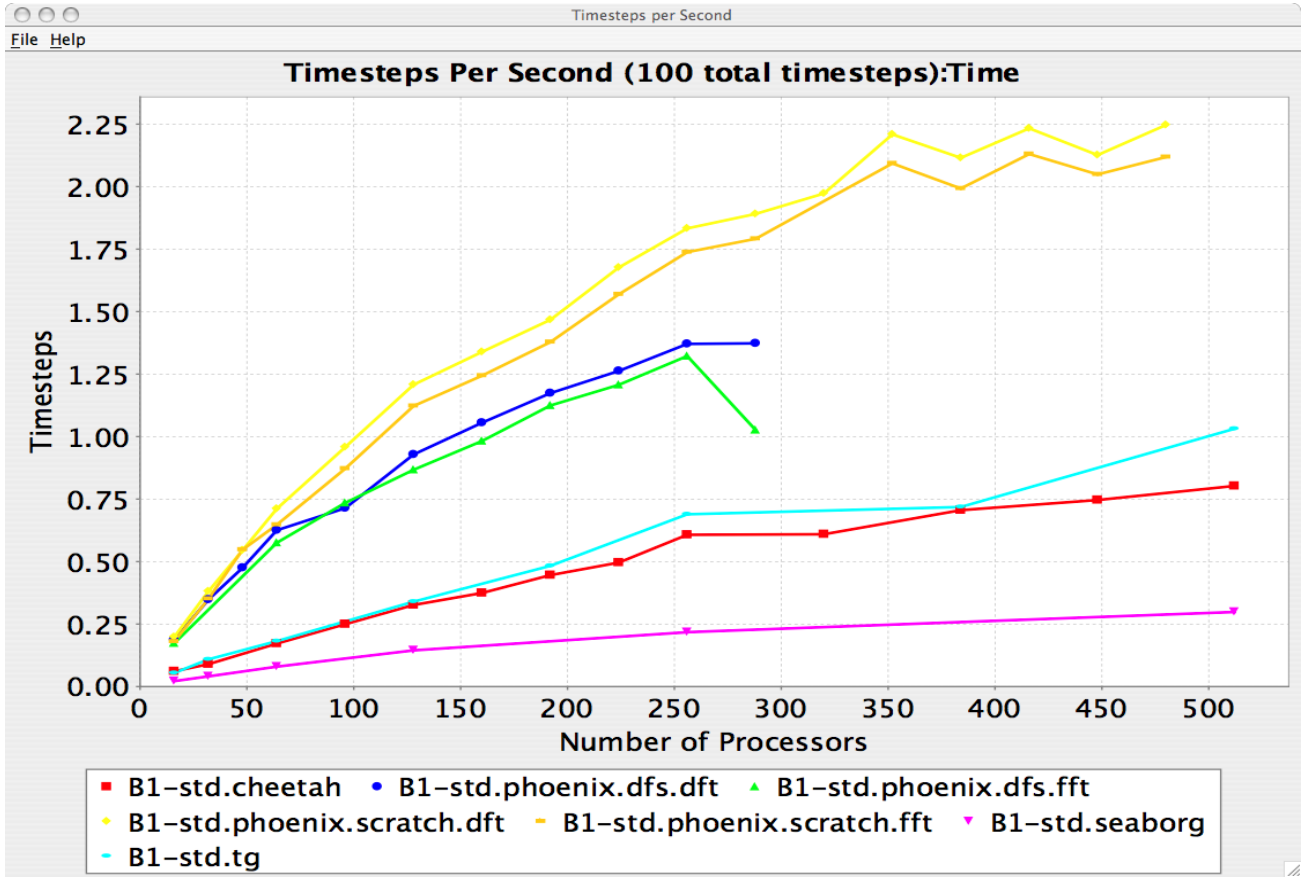
PerfExplorer - Relative Efficiency by Routine



PerfExplorer - Relative Speedup

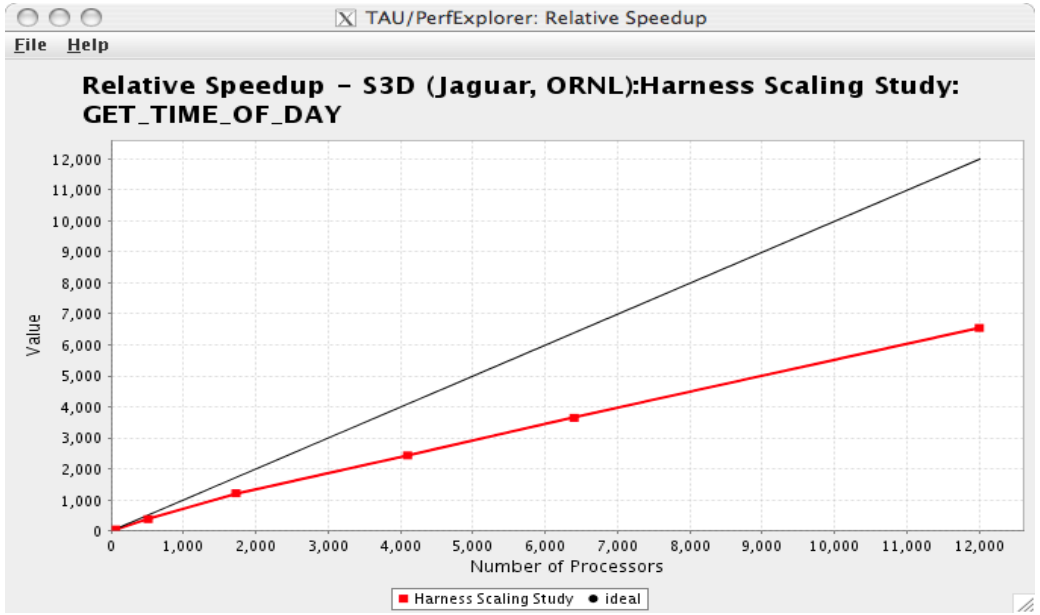


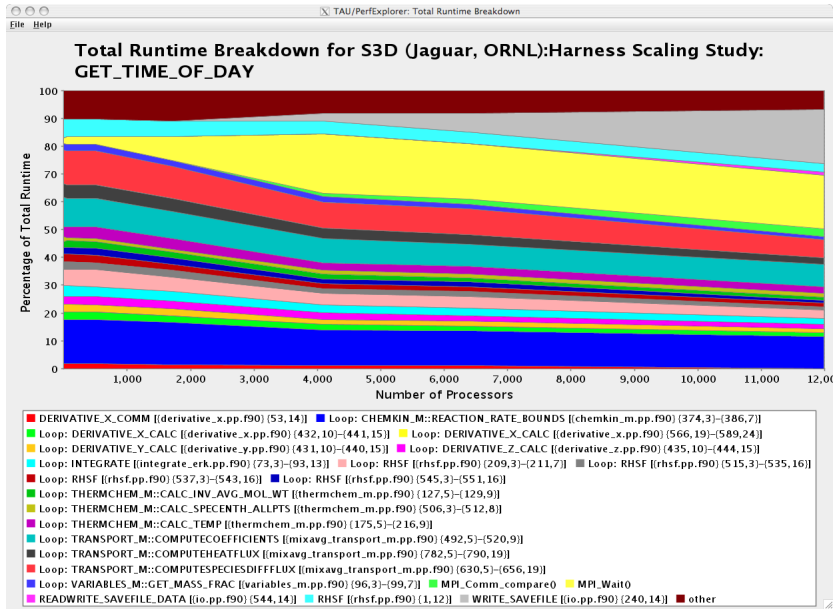
PerfExplorer - Timesteps Per Second



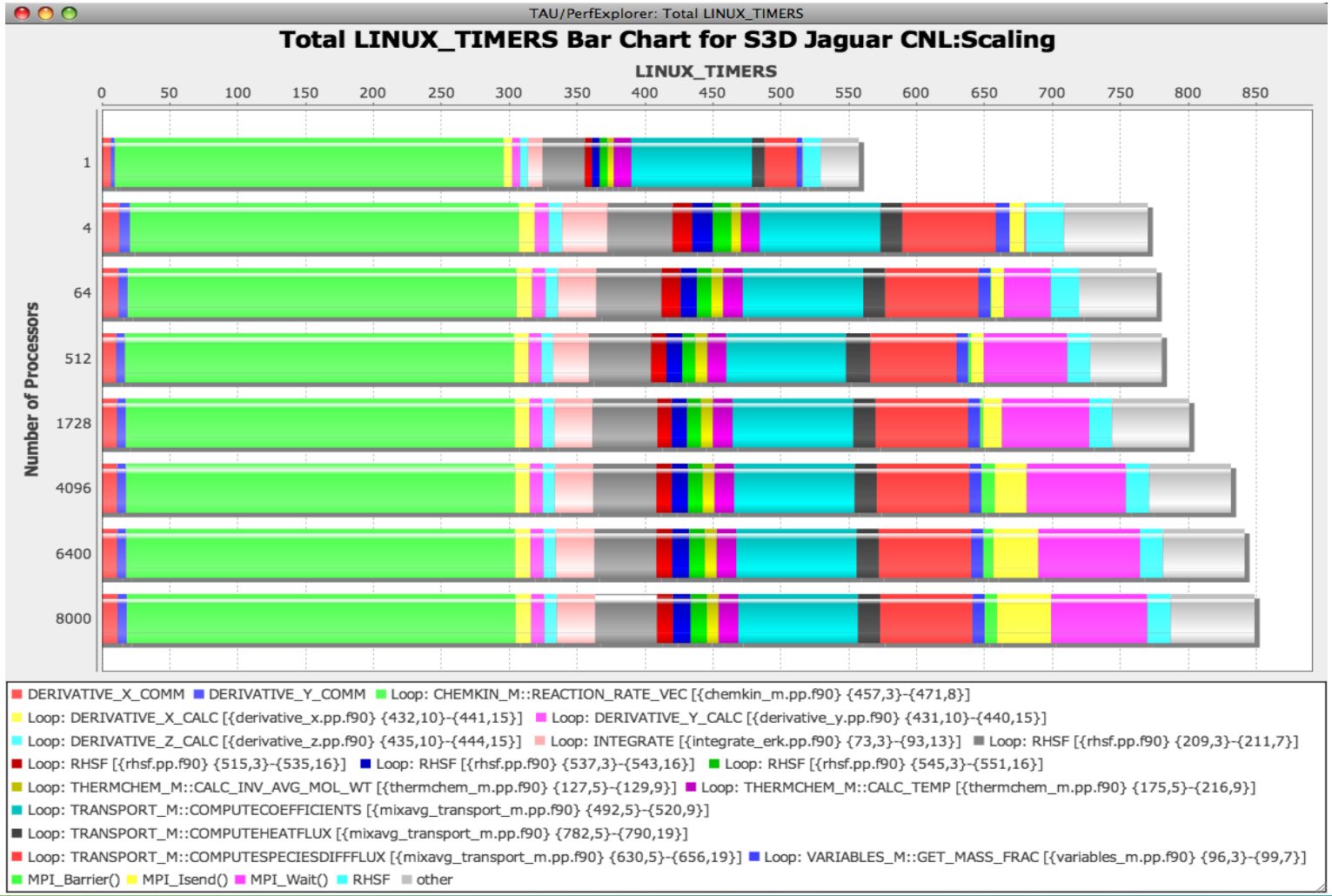
Evaluate Scalability

- Goal: How does my application scale? What bottlenecks occur at what core counts?
- Load profiles in taudb database and examine with PerfExplorer

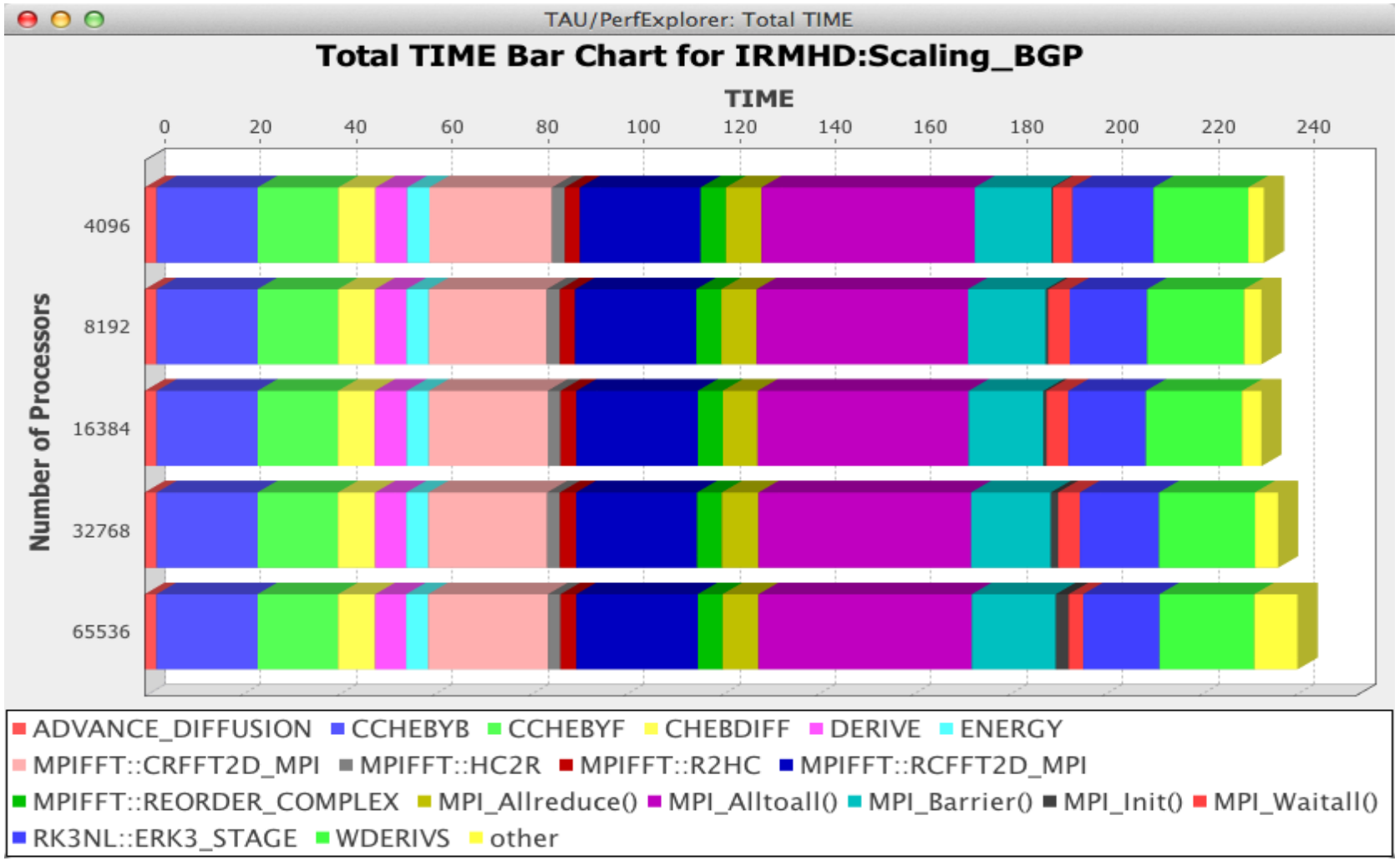




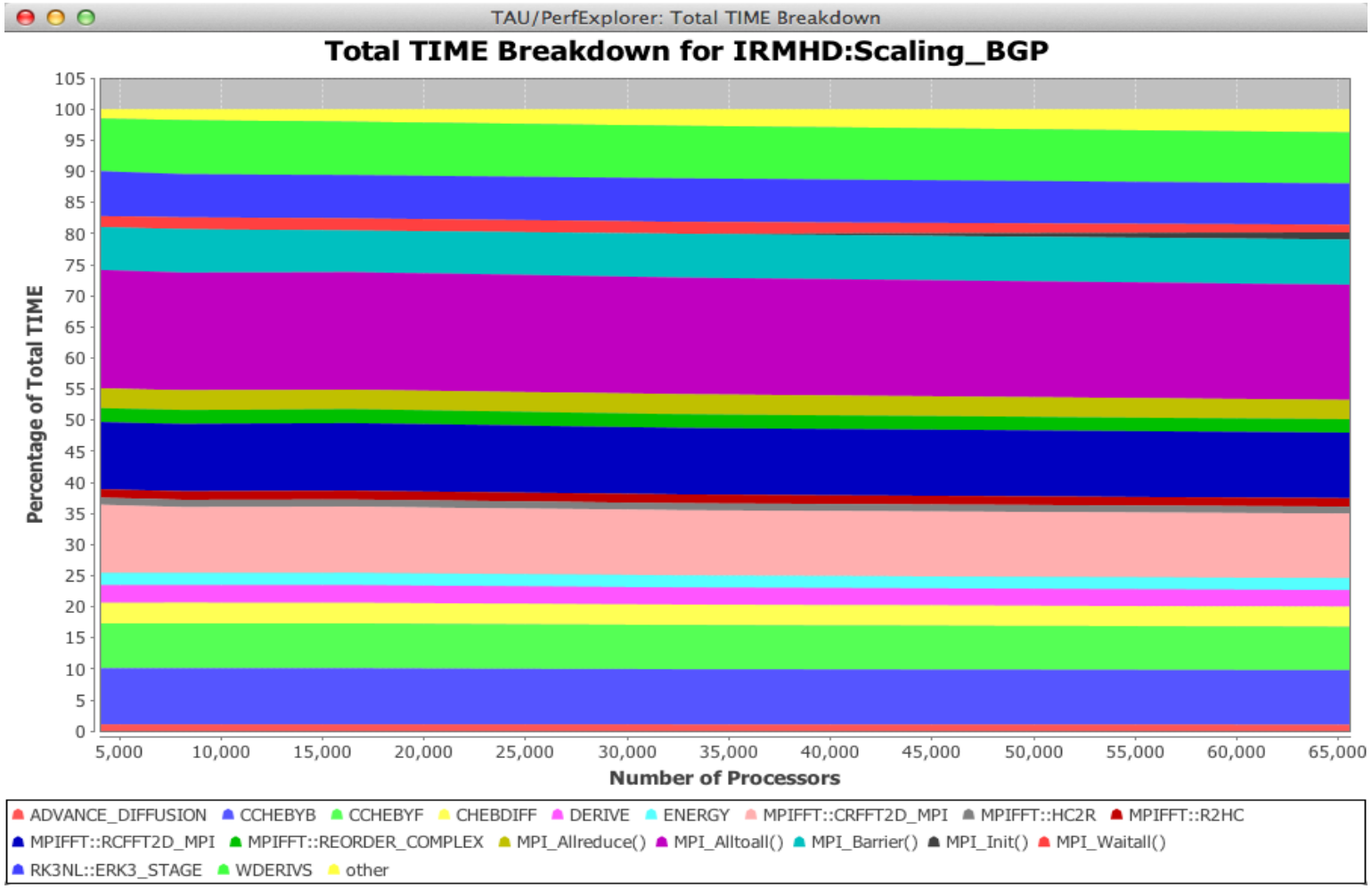
Evaluate Scalability



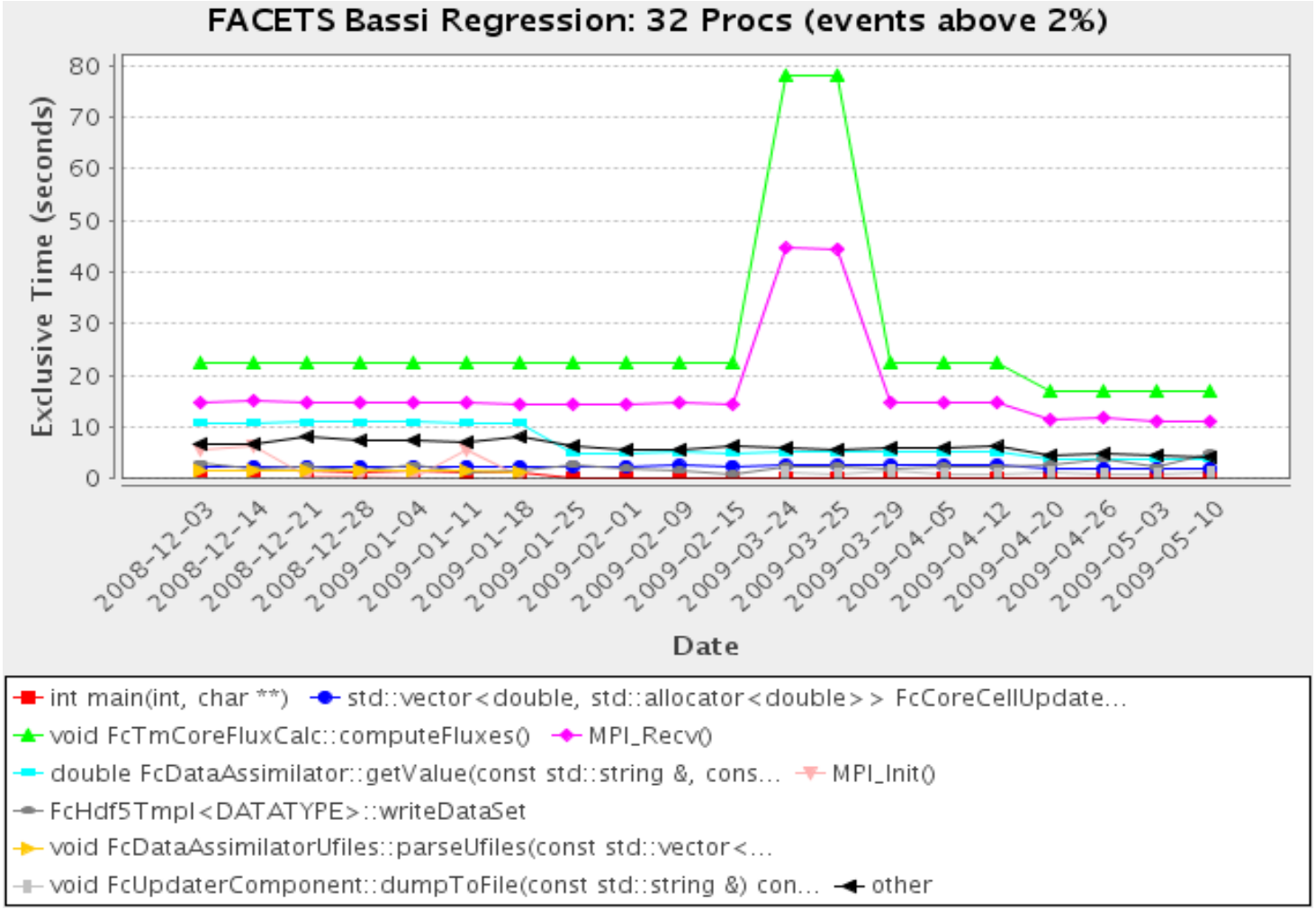
PerfExplorer



PerfExplorer



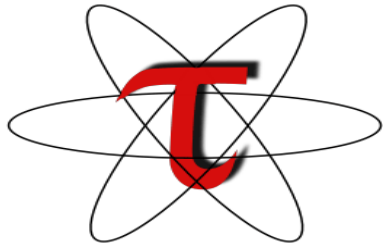
Performance Regression Testing



Support Acknowledgments

- U.S. Department of Energy (DOE)
  - Office of Science
  - PNNL, LBL, ORNL
  - ASC/NNSA, Tri-labs (LLNL,LANL, SNL)
- U.S. Department of Defense (DoD)
  - HPC Modernization Office (HPCMO)
- NSF Software Development for Cyberinfrastructure (SDCI)
- Juelich Supercomputing Center, NIC
- Argonne National Laboratory
- T.U. Dresden
- ParaTools, Inc.

Download TAU from U. Oregon



http://tau.uoregon.edu

http://www.hpclinux.com [LiveDVD, OVA]

Free download, open source, BSD license