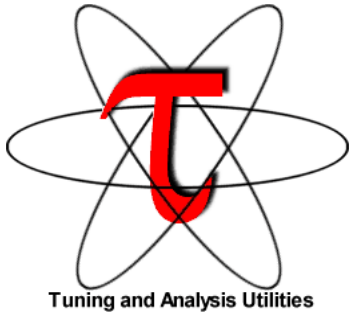
TAU Performance System®



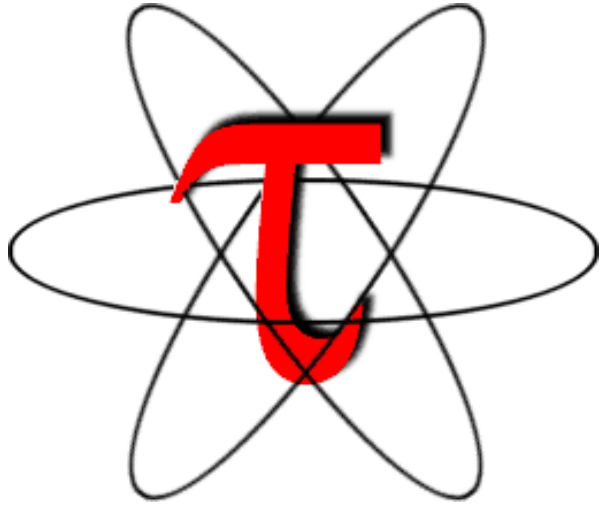
Sameer Shende
sameer@cs.uoregon.edu
University of Oregon
http://tau.uoregon.edu

TAU: The story so far… Excellent VI-HPS tools that you have seen!

- mpiP
  - Callsite instrumentation with preloading of MPI wrapper interposition library; Unwinding of callstack.
- MAQAO
  - Binary rewriting as well as event based sampling.
- Score-P
  - Compiler-based instrumentation, reduction of instrumentation overhead by selective instrumentation (filtering).
  - Efficient callpath profiling using a tree based data structure; generating CUBEX file format.
  - Native OTF2 trace file generation capability.
- Scalasca
  - Parallel scalable analysis of OTF2 traces to generate CUBEX profile files using Scout.
  - Profiles contain performance properties (bottlenecks)! Cube may be used to visualize profiles.
- Vampir
  - OTF2: hierarchical trace format – no need to merge and convert large trace files!
  - Scalable visualization of OTF2 traces. Analysis can begin immediately after program execution!
- Extrae/Paraver
  - Preloading MPI libraries: no need to recompile/relink! Intuitive trace analysis and visualization.

What if we had a simple tool that could operate on unmodified binaries? A tool that…

- Provides callsite instrumentation, preloading measurement library
- Supports profiling as well as tracing, works on all HPC platforms including GPUs
- Rewrites binaries, supports event-based sampling, memory and I/O instrumentation
- Supports compiler-based instrumentation, source-based instrumentation, filtering
- Interfaces with Score-P and other performance counter libraries like LIKWID and PAPI
- Supports callpath profiling, callsite profiling, works with Vampir, MAQAO, and Paraver!

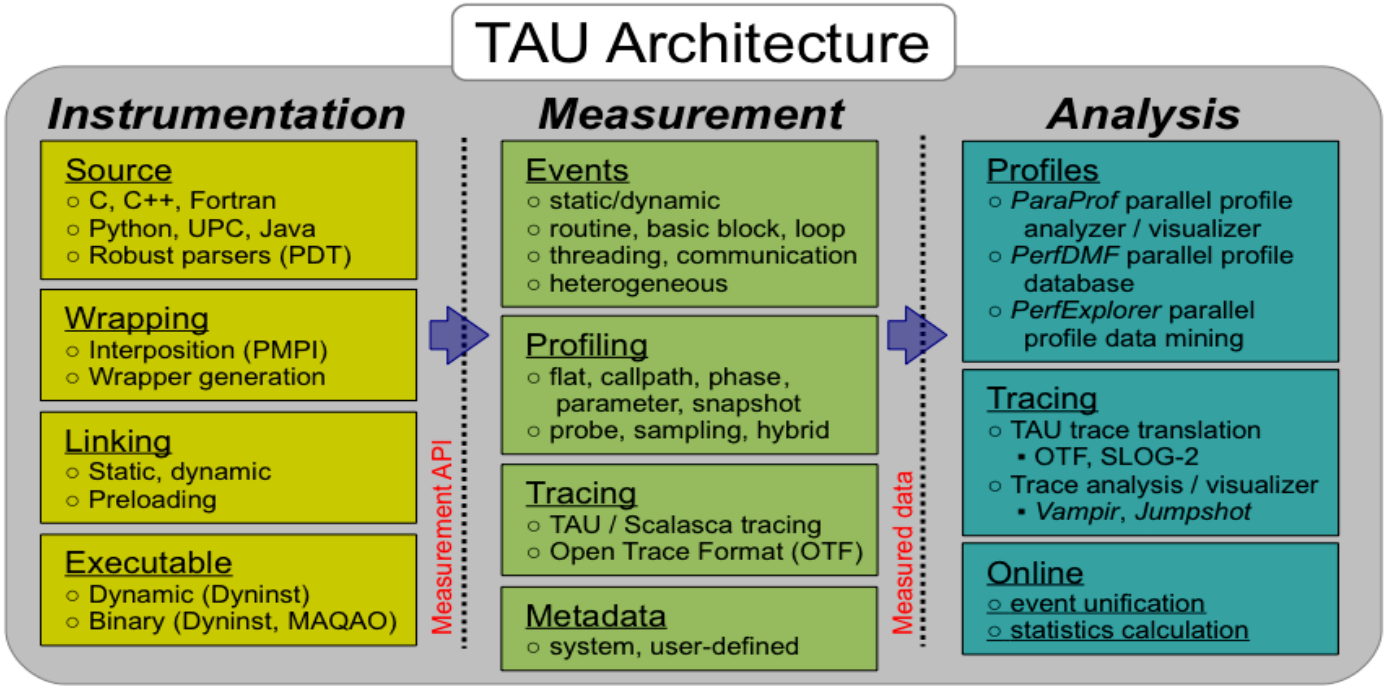


TAU Performance System

- Instrumentation
  - Fortran, C++, C, UPC, Java, Python, Chapel, CoArray Fortran
  - Automatic instrumentation
- Measurement and analysis support
  - MPI, OpenSHMEM, ARMCI, PGAS, DMAPP, HPX
  - pthreads, OpenMP, OMPT TR6 interface (substitute OpenMP library), hybrid, other thread models
  - GPU, CUDA, OpenCL, OpenACC
  - Parallel profiling and tracing
  - Native OTF2 generation with support for callsite instrumentation
  - Efficient callpath proflles and trace generation using Score-P
- Analysis
  - Parallel profile analysis (ParaProf), data mining (PerfExplorer)
  - Performance database technology (TAUdb)
  - 3D profile browser that supports TAU and CUBEX profile formats!

TAU Performance System®

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



TAU Performance System

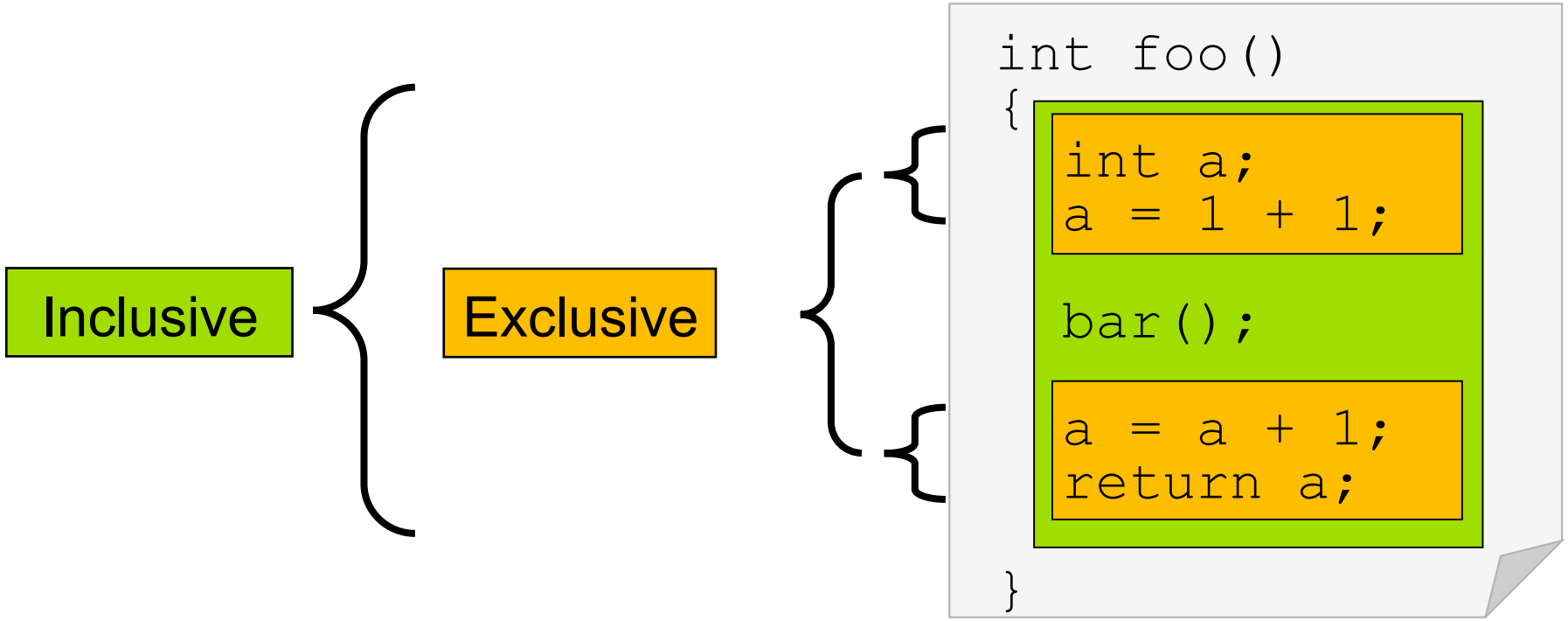
- TAU supports both sampling and direct instrumentation
- Memory debugging as well as I/O performance evaluation
- Profiling as well as tracing
- Interfaces with Score-P for more efficient measurements
- TAU's instrumentation covers:
  - Runtime library interposition (tau_exec)
  - Compiler-based instrumentation
  - Supports both PAPI and LIKWID [FAU] for hardware performance counter integration
  - 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 (OMPT, tau_exec –T ompt,tr6). Substitutes an alternate OpenMP lib. Not compiler based
  - CUDA CUPTI, OpenCL (tau_exec -T cupti -cupti)

Application Performance Engineering using TAU

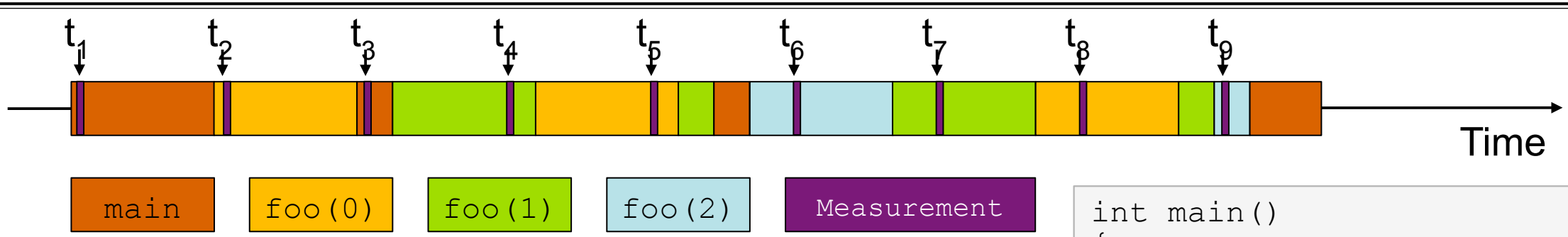
- How much time is spent in each application routine and outer *loops*? Within loops, what is the contribution of each *statement*? What is the time spent in specific OpenMP loops and statements?
- How many instructions are executed in these code regions?
  Floating point, Level 1 and 2 *data cache misses*, hits, branches taken? What is the extent of vectorization for loops?
- 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?
- 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



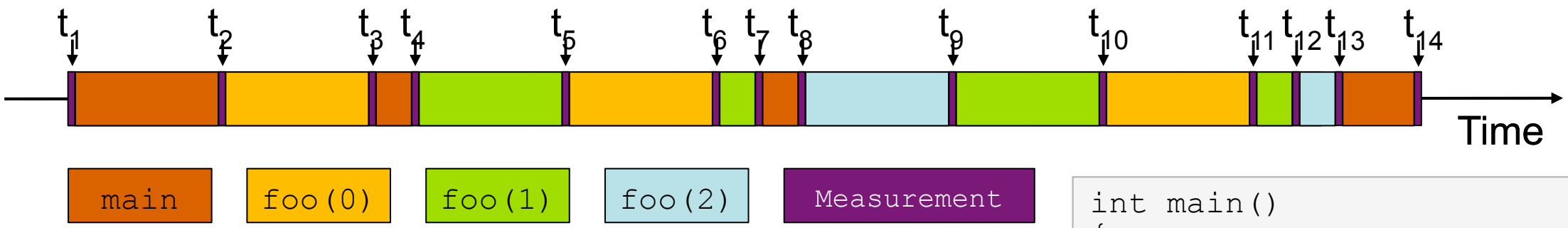
Sampling



- 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

```
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



- 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;
  Start("main");
  for (i=0; i < 3; i++)
    foo(i);
  Stop ("main");
  return 0;
}

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

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
  - mpirun -np 64 ./a.out
  - mpirun -np 64 tau_exec −T ompt ./a.out

TAU demo!

TAU tutorial exercise 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 (*CoolMUC3, LRZ*)

- Setup preferred program environment compilers

```
% source /home/hpc/a2c06/lu23vox/load_tau.sh
% tar xf ~lu23vox/workshop.tgz; cd workshop; cat README; cd NPB3.1;make
# If you have previous performance data from Score-P, you may view it with TAU's paraprof
% paraprof profile.cubex &
```

For PerfExplorer:

```
% cp ~lu23vox/data.tgz . ; tar zxf data.tgz; cd data
% cat README
  and follow the steps
```

NPB-MPI / BT with TAU's Sampling (tau_exec –ebs)

```
% cd bin
% mpirun –np 64 tau_exec –ebs ./bt.B.64

NAS Parallel Benchmarks (NPB3.1-MPI) …

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

 BT-MZ Benchmark Completed.
 Time in seconds = 17.34
% paraprof &
% paraprof --pack bt_ebs.ppk
<Copy file over to desktop using scp>
% paraprof bt_ebs &
```

tau_exec launches un-instrumented binaries!

NPB-MZ-MPI / BT with TAU's native OTF2 traces

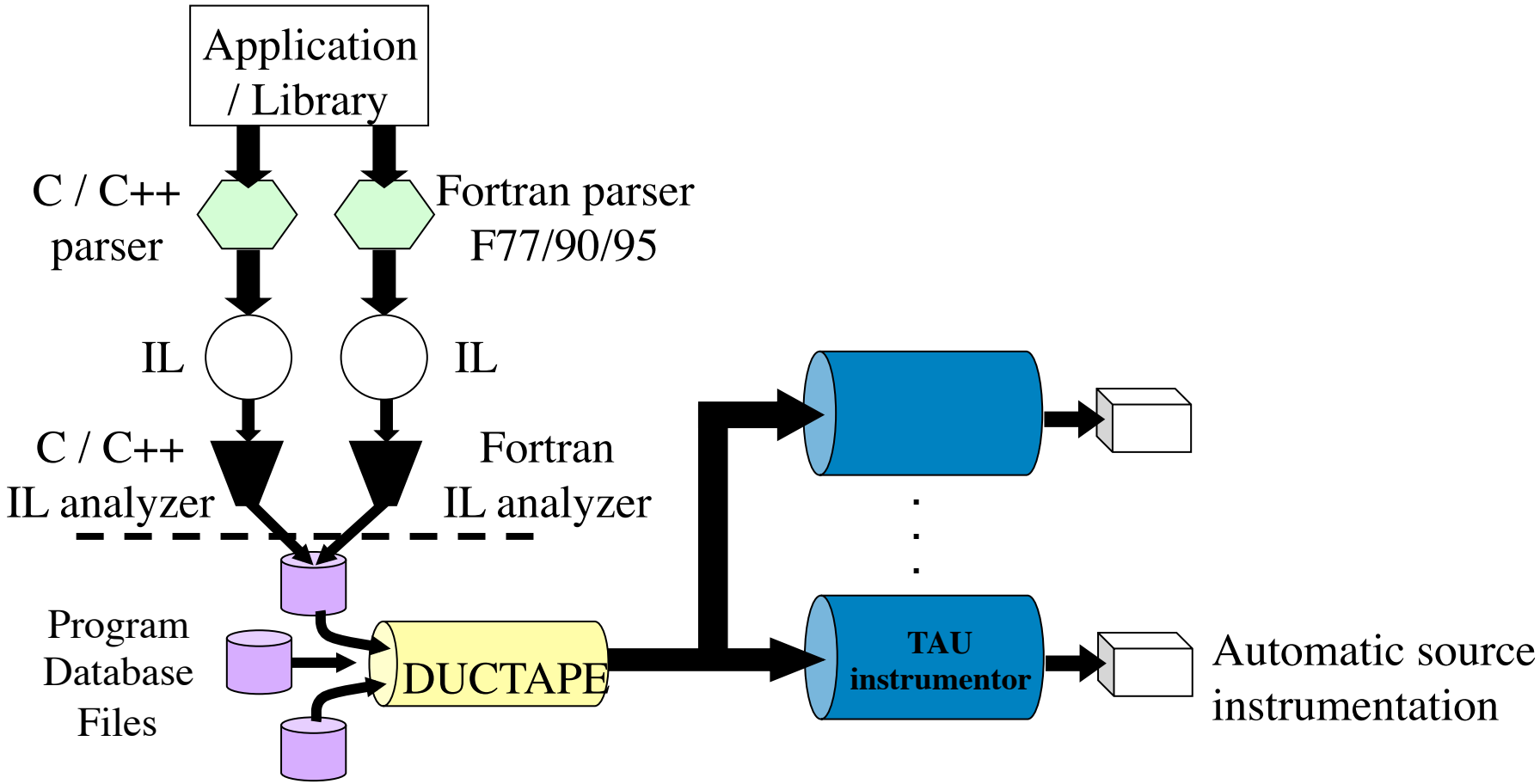
```
% cd bin
% export TAU_TRACE=1
% export TAU_TRACE_FORMAT=otf2
% export TAU_CALLSITE=1
% mpirun -np 64 tau_exec ./bt.B.64
 Time step    1
 Time step   20
  [...]
 Time step  180
 Time step  200
 Verification Successful

 BT-MZ Benchmark Completed.
 Time in seconds = 17.34
% ls traces.otf2
% module load vampir
% vampir traces.otf2 &
```

- Launch application
- TAU_TRACE=1
- TAU_CALLSITE=1
- TAU_TRACE_FORMAT=otf2

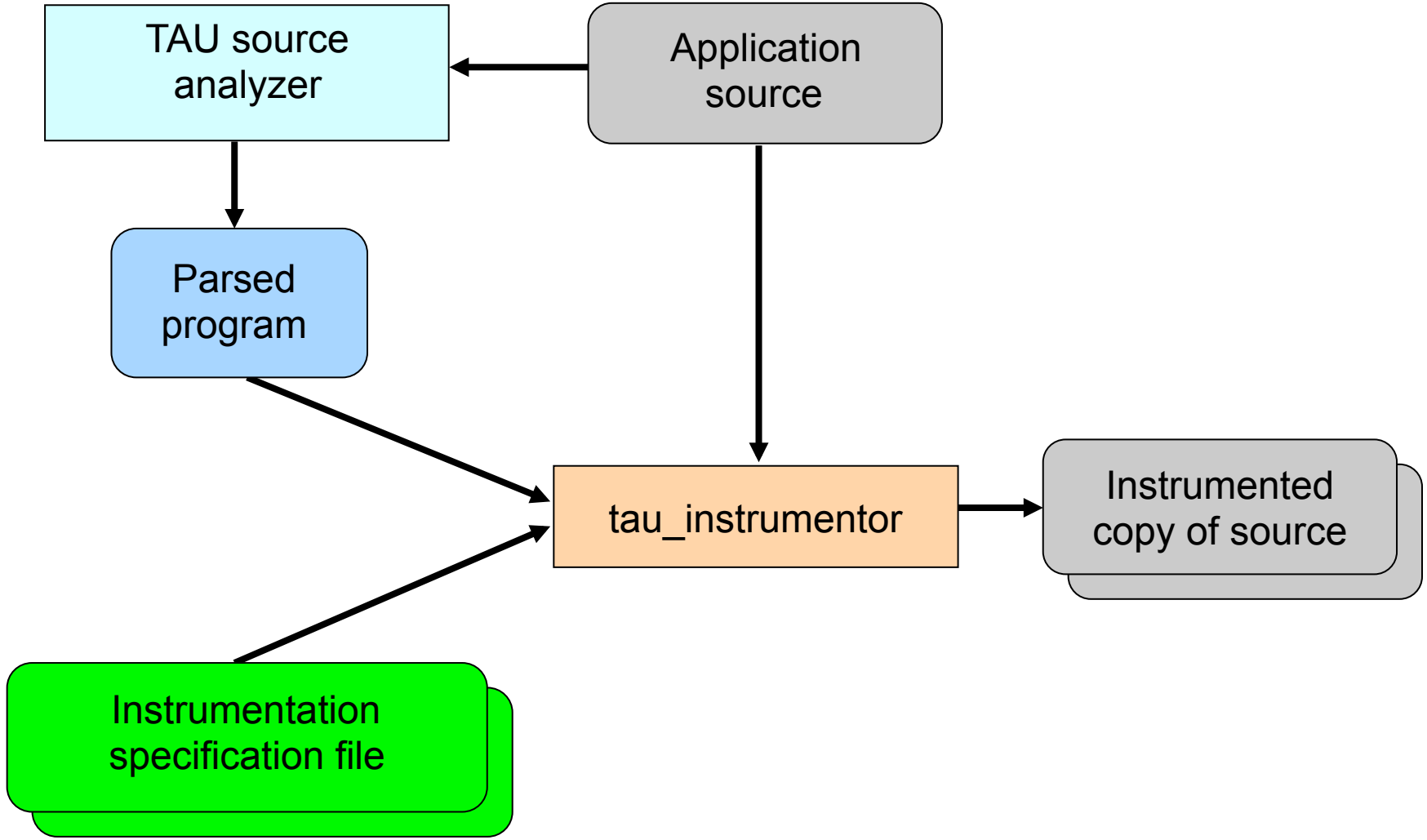
TAU generates native OTF2 traces for Vampir with callsite info and MPI events for uninstrumented binaries!

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_lite.tgz
  - ./configure; make ; make install
- Installing TAU:
  - wget http://tau.uoregon.edu/tau.tgz
  - ./configure –ompt=download-tr6 –arch=x86_64 -mpi -bfd=download -pdt=<dir> -papi=<dir> ...
  - make install
- Using TAU:
  - 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:

```
% source ~lu23vox/load_tau.sh
% ls $TAU/Makefile*

% export TAU_MAKEFILE=$TAU/Makefile.tau-icpc-papi-ompt-tr6-mpi-pdt-openmp
% 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:

```
% 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)
```

Different Makefiles for TAU Compiler and Runtime Options

```
% source ~lu23vox/load_tau.sh
% ls $TAU/Makefile.*
/home/hpc/a2c06/lu23vox/pkgs/tau-2.27.1/x86_64/lib/Makefile.tau-icpc-papi-mpi-pdt
/home/hpc/a2c06/lu23vox/pkgs/tau-2.27.1/x86_64/lib/Makefile.tau-icpc-papi-mpi-pdt-scorep
/home/hpc/a2c06/lu23vox/pkgs/tau-2.27.1/x86_64/lib/Makefile.tau-icpc-papi-ompt-tr6-mpi-pdt-openmp
/home/hpc/a2c06/lu23vox/pkgs/tau-2.27.1/x86_64/lib/Makefile.tau-icpc-papi-pdt
/home/hpc/a2c06/lu23vox/pkgs/tau-2.27.1/x86_64/lib/Makefile.tau-papi-pdt
/home/hpc/a2c06/lu23vox/pkgs/tau-2.27.1/x86_64/lib/Makefile.tau-pdt-likwid
```

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

Makefile.tau-icpc-papi-mpi-pdt

- Supports MPI instrumentation & PDT for automatic source instrumentation

```
% export TAU_MAKEFILE=$TAU/Makefile.tau-icpc-papi-mpi-pdt


% tau_f90.sh matmult.f90 -o matmult
% mpirun -np 64  ./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:
```
% mpirun -np 256 ./a.out     to
% mpirun -np 256  tau_exec -T <comma_separated_options> ./a.out

% mpirun -np 256  tau_exec -T papi,mpi,pdt ./a.out
```
Preloads `$TAU/shared-papi-mpi-pdt/libTAU.so`
```
% mpirun -np 256  tau_exec -T papi ./a.out
```
Preloads `$TAU/shared-papi-mpi-pdt/libTAU.so` by matching.
```
% mpirun -np 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
  - % mpirun -np 16 ./a.out
- Track MPI performance
  - % mpirun -np 16 tau_exec ./a.out
- Track POSIX I/O, OpenMP, and MPI performance (MPI enabled by default)
  - % mpirun -np 16 tau_exec –T mpi,pdt,ompt –io ./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>
- Load wrapper interposition library
  - % mpirun –np 16 tau_exec –loadlib=<path/libwrapper.so> ./a.out
- Track GPGPU operations
  - % mpirun –np 16 tau_exec –cupti ./a.out
  - % mpirun –np 16 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:

```
% mpirun -np 256 ./a.inst
```

Compile-Time Options for TAU's compiler scripts (e.g., tau_f90.sh)

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
-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

See tau_compiler.sh for a full list of TAU_OPTIONS.

…

Compiling Fortran Codes with TAU

- If your Fortran code uses free format in .f files (fixed is default for .f), you may use:
  % export TAU_OPTIONS= '-optPdtF95Opts="-R free" -optVerbose '

- To use the compiler based instrumentation instead of PDT (source-based):
  % export TAU_OPTIONS= '-optCompInst -optVerbose'

- If your Fortran code uses C preprocessor directives (#include, #ifdef, #endif):
  % export TAU_OPTIONS= '-optPreProcess -optVerbose -optDetectMemoryLeaks'

- To use an instrumentation specification file:
  % export TAU_OPTIONS= '-optTauSelectFile=select.tau -optVerbose -optPreProcess'
  % cat select.tau
  BEGIN_INSTRUMENT_SECTION
  loops routine="#"
  # this statement instruments all outer loops in all routines. # is wildcard as well as comment in first column.
  END_INSTRUMENT_SECTION

TAU's Selective Instrumentation (Filter) File Format

```
% export TAU_OPTIONS='-optTauSelectFile=/path/to/select.tau …'
% cat select.tau
BEGIN_INCLUDE_LIST
int main#
int dgemm#
END_INCLUDE_LIST
BEGIN_FILE_INCLUDE_LIST
Main.c
Blas/*.f77
END_FILE_INCLUDE_LIST
# replace INCLUDE with EXCLUDE list for excluding routines/files

BEGIN_INSTRUMENT_SECTION
loops routine="foo"
loops routine="int main#"
END_INSTRUMENT_SECTION
% export TAU_SELECT_FILE=select.tau    (to use at runtime)
```

Runtime Environment Variables

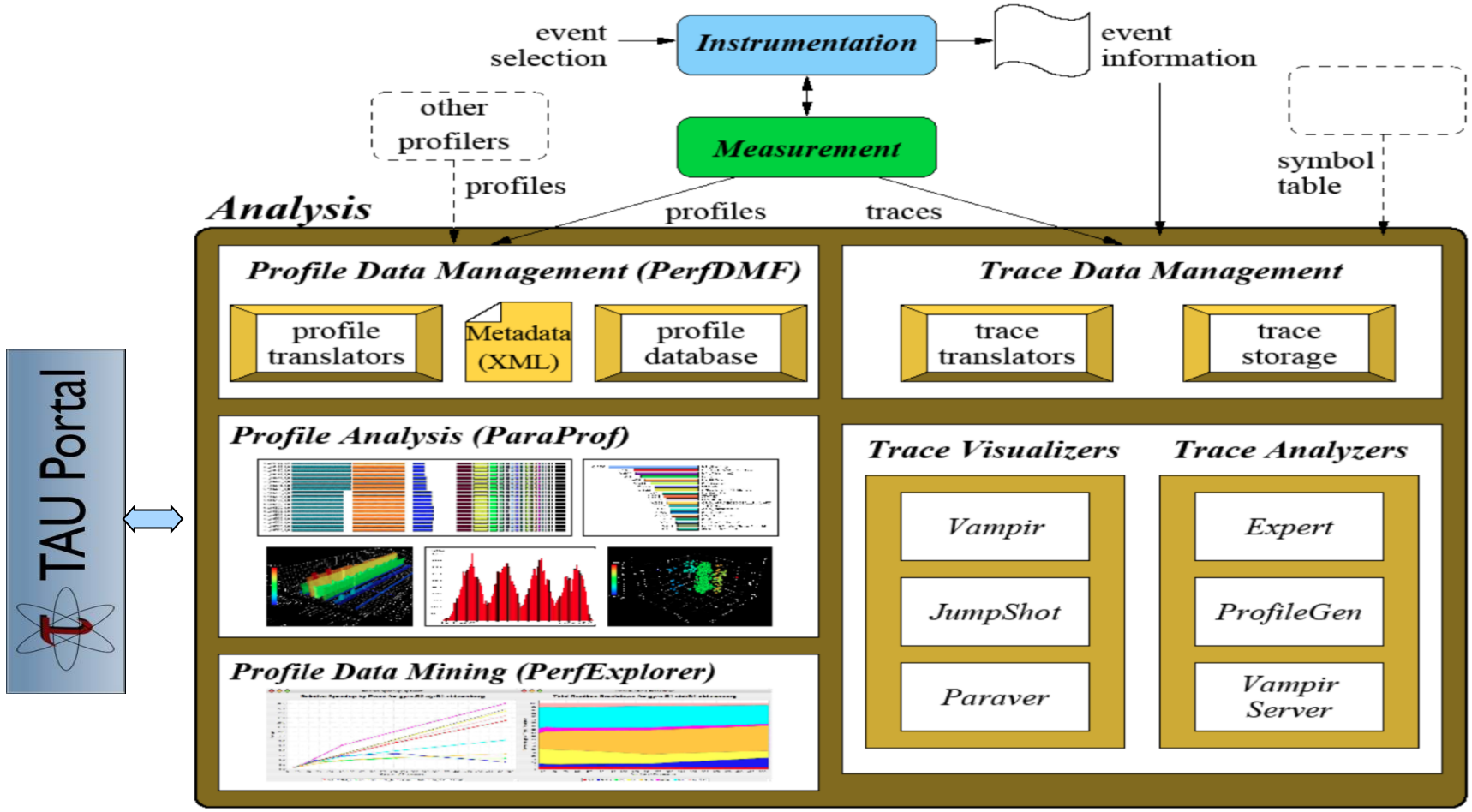
Environment Variable	Default	Description
TAU_TRACE	0	Setting to 1 turns on tracing
TAU_TRACE_FORMAT	default	Setting to “otf2” generates OTF2 traces natively.
TAU_CALLPATH	0	Setting to 1 turns on callpath profiling
TAU_CALLSITE	0	Setting to 1 generates callsite information in events. May be used with tracing.
TAU_TRACK_MEMORY_FOOTPRINT	0	Setting to 1 turns on tracking memory usage by tracking the resident set size and high water mark of memory usage
TAU_TRACK_LOAD	0	Setting to 1 tracks system load 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. A routine is throttled if it takes less than 10 microseconds per call (and called > 10000 times).
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.)

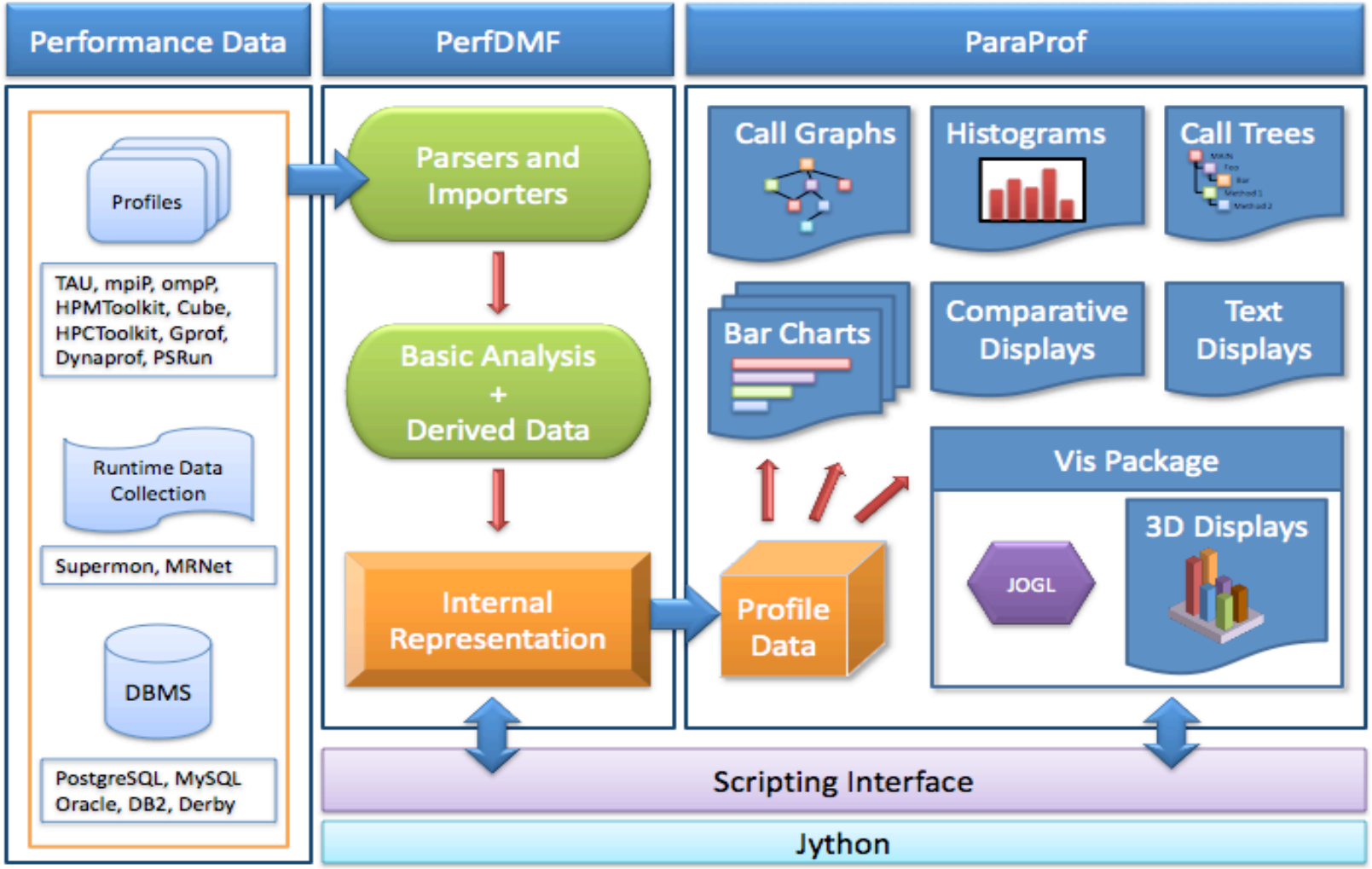
Environment Variable	Default	Description
TAU_TRACK_MEMORY_LEAKS	0	Tracks allocates that were not de-allocated (needs –optMemDbg or tau_exec –memory)
TAU_EBS_SOURCE	TIME	Allows using PAPI hardware counters for periodic interrupts for EBS (e.g., TAU_EBS_SOURCE=PAPI_TOT_INS when TAU_SAMPLING=1)
TAU_EBS_PERIOD	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

TAU's Analysis Tools: ParaProf

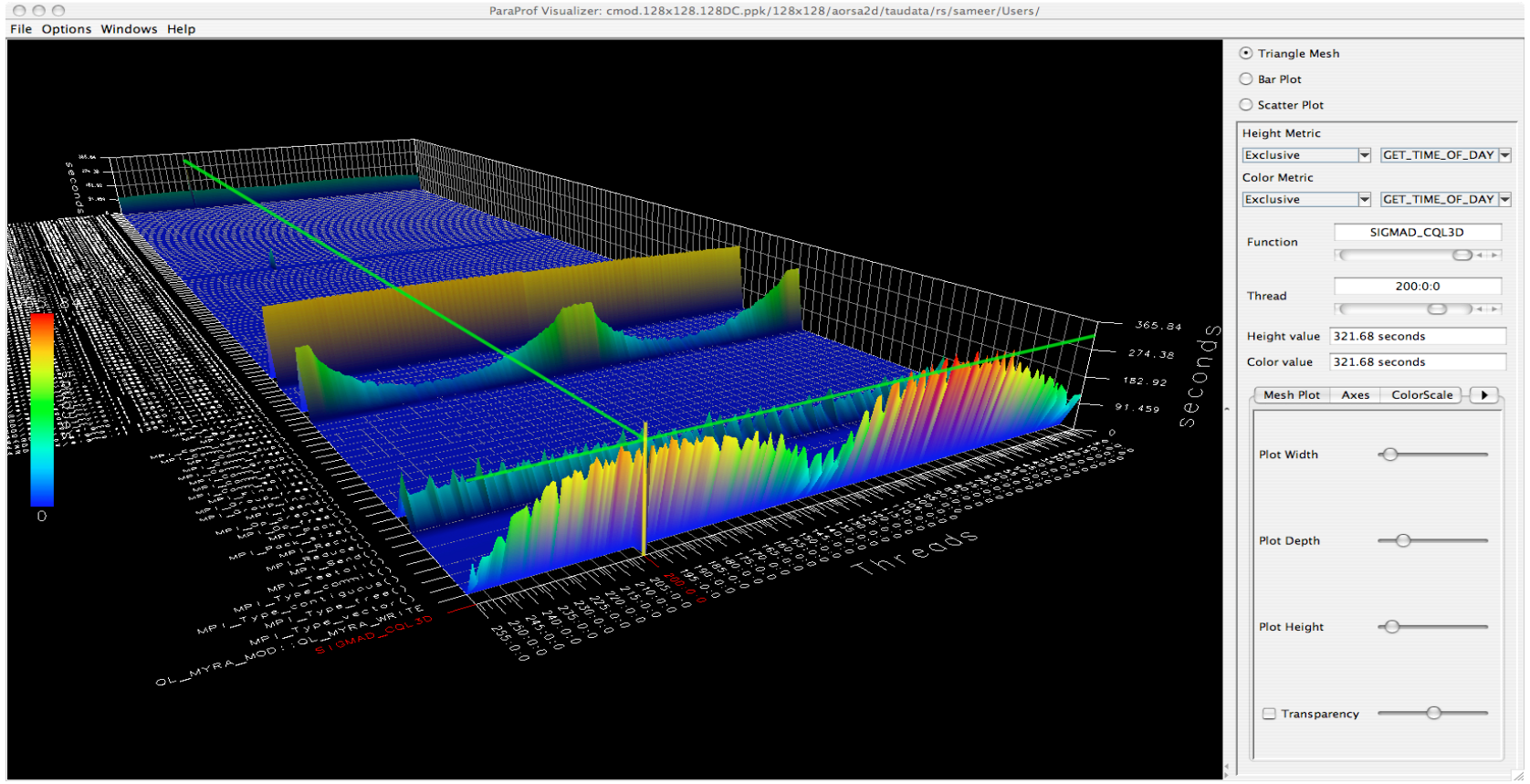
TAU Analysis



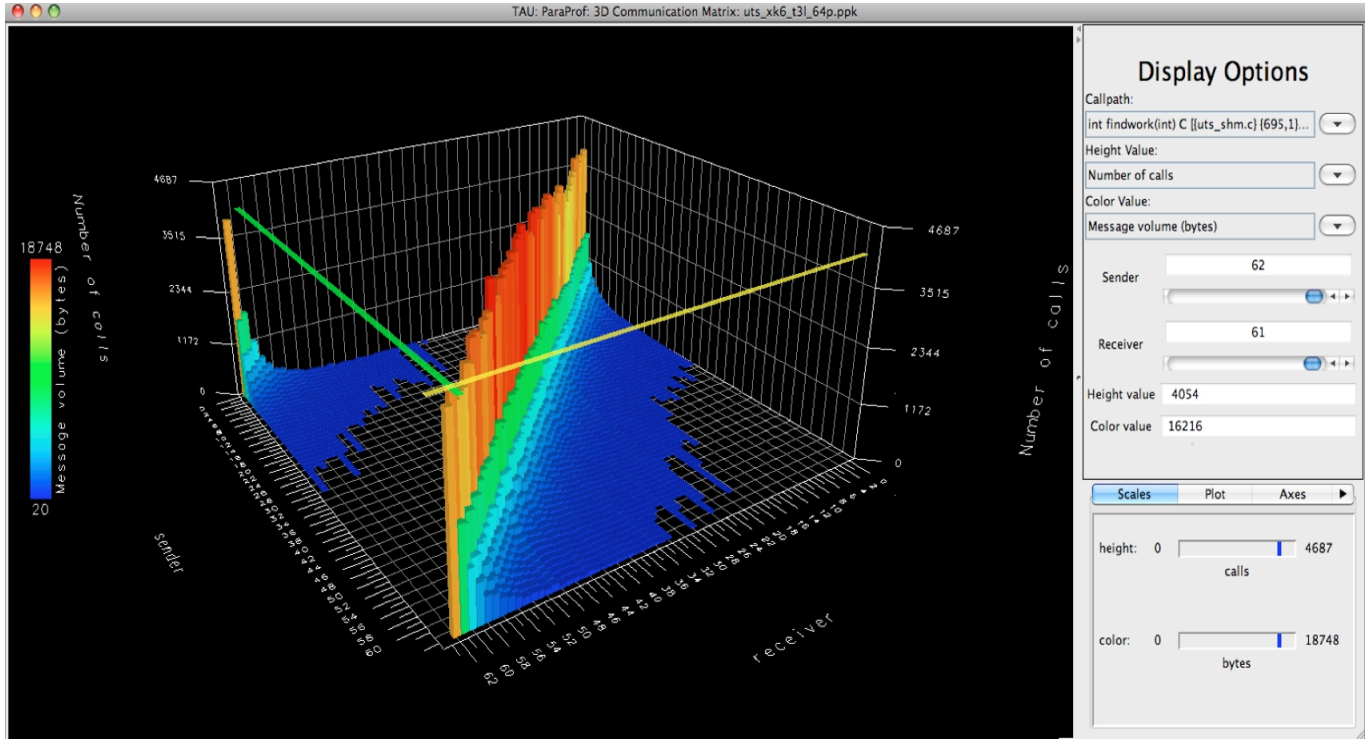
ParaProf Profile Analysis Framework



Parallel Profile Visualization: ParaProf



ParaProf 3D Communication Matrix



% export TAU_COMM_MATRIX=1

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)

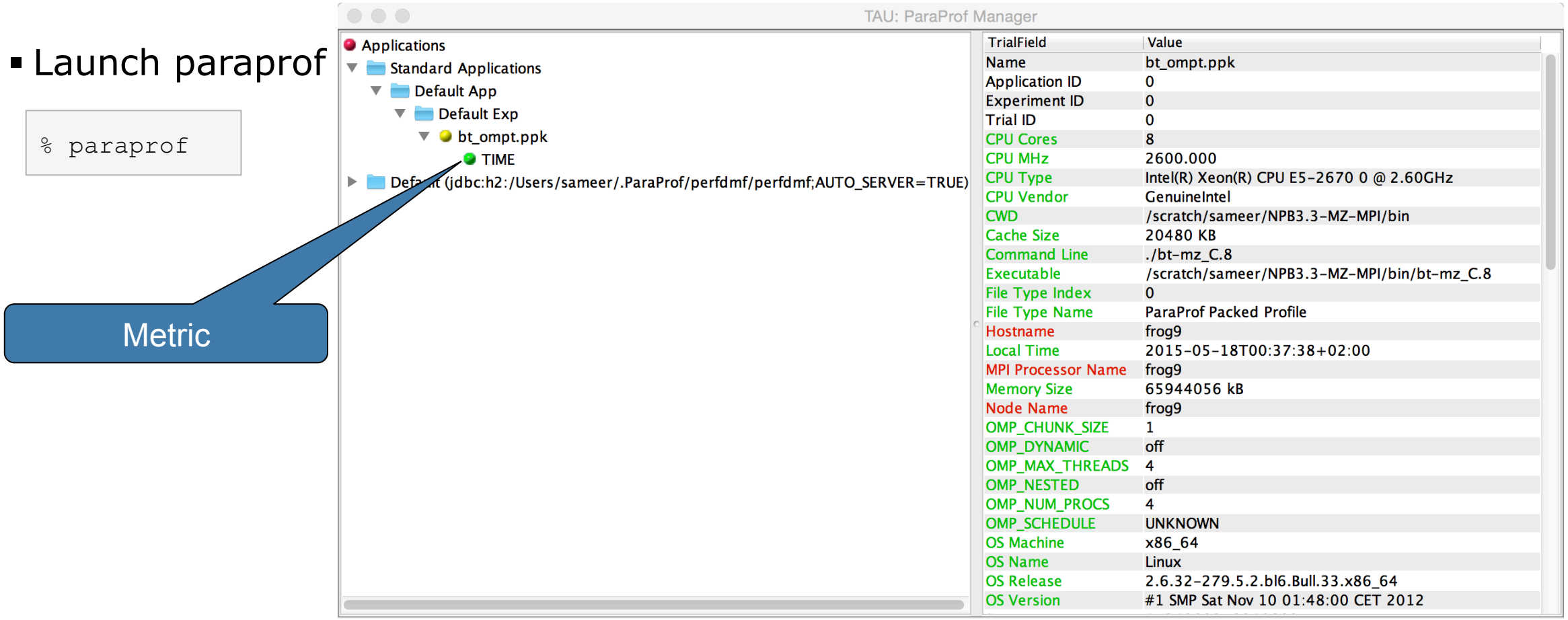
```
Example:
    mpirun -np 2 tau_exec -T icpc,ompt,mpi  -ompt ./a.out
    mpirun -np 2 tau_exec -io ./a.out
Example - event-based sampling with samples taken every 1,000,000 FP instructions
    mpirun -np 8 tau_exec -ebs -ebs_period=1000000 -ebs_source=PAPI_FP_INS ./ring
Examples - GPU:
    tau_exec -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 Analysis Tools: paraprof

- Launch paraprof

```
% paraprof
```

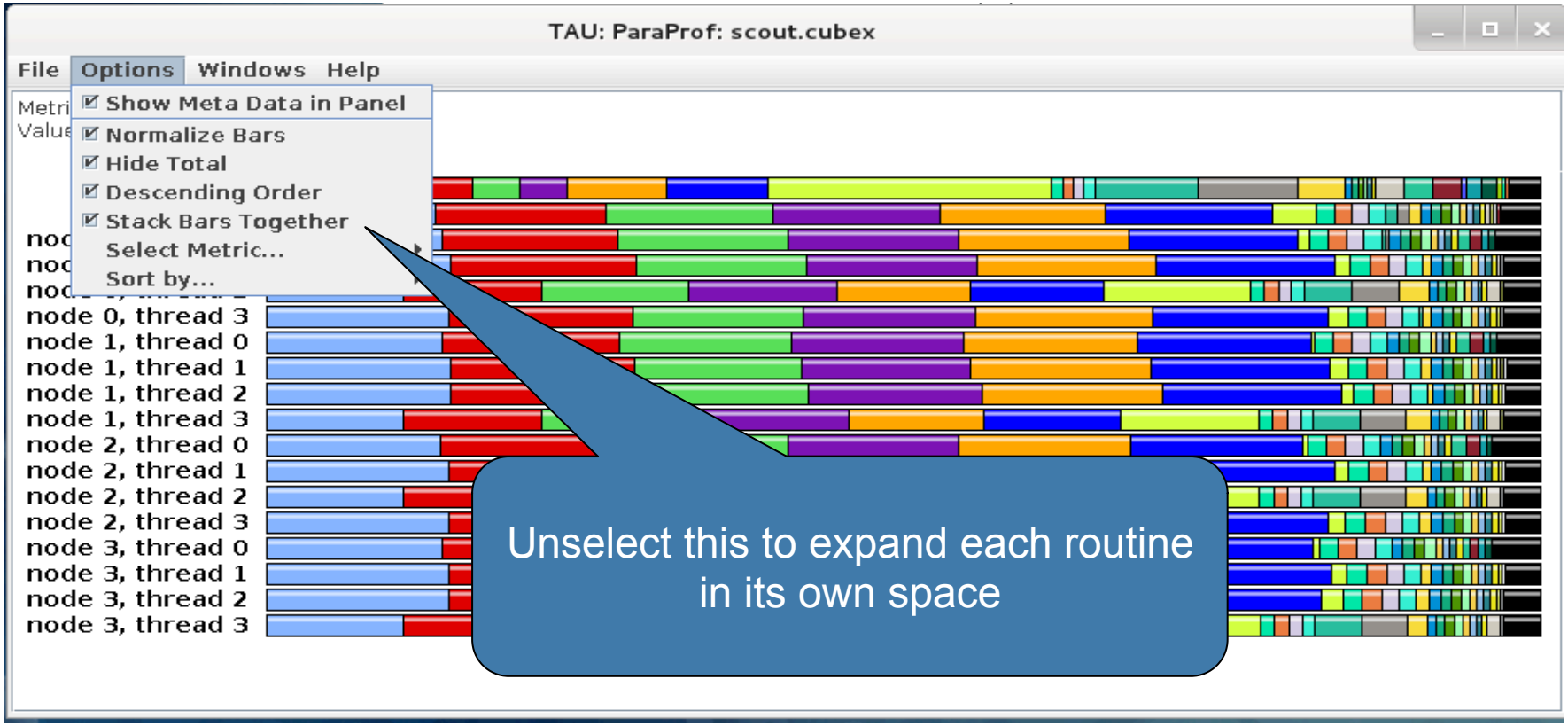


Paraprof main window

Colors represent code regions

Options -> uncheck Stack Bars Together

Paraprof main window



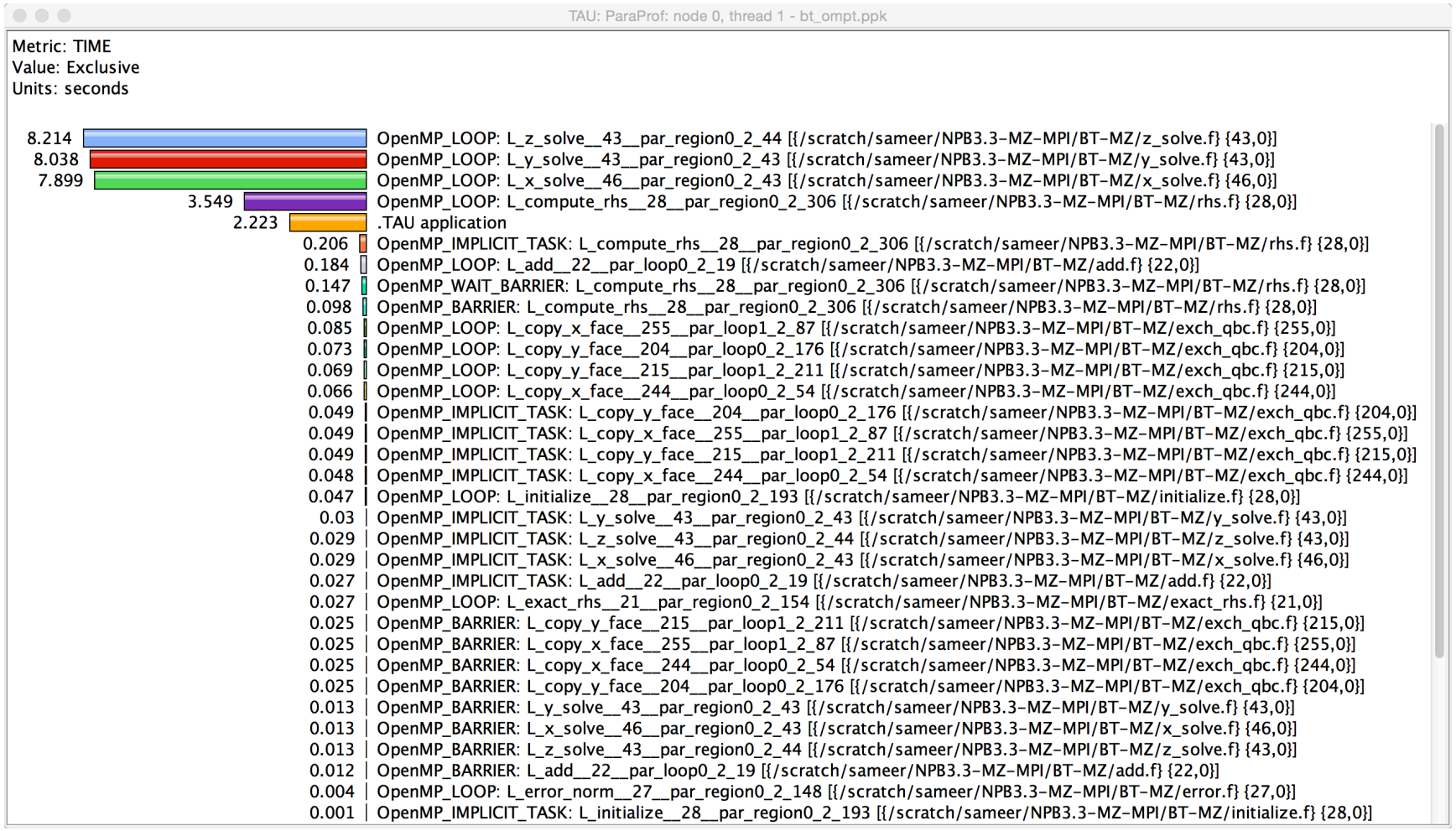
Paraprof main window

Left/right click here

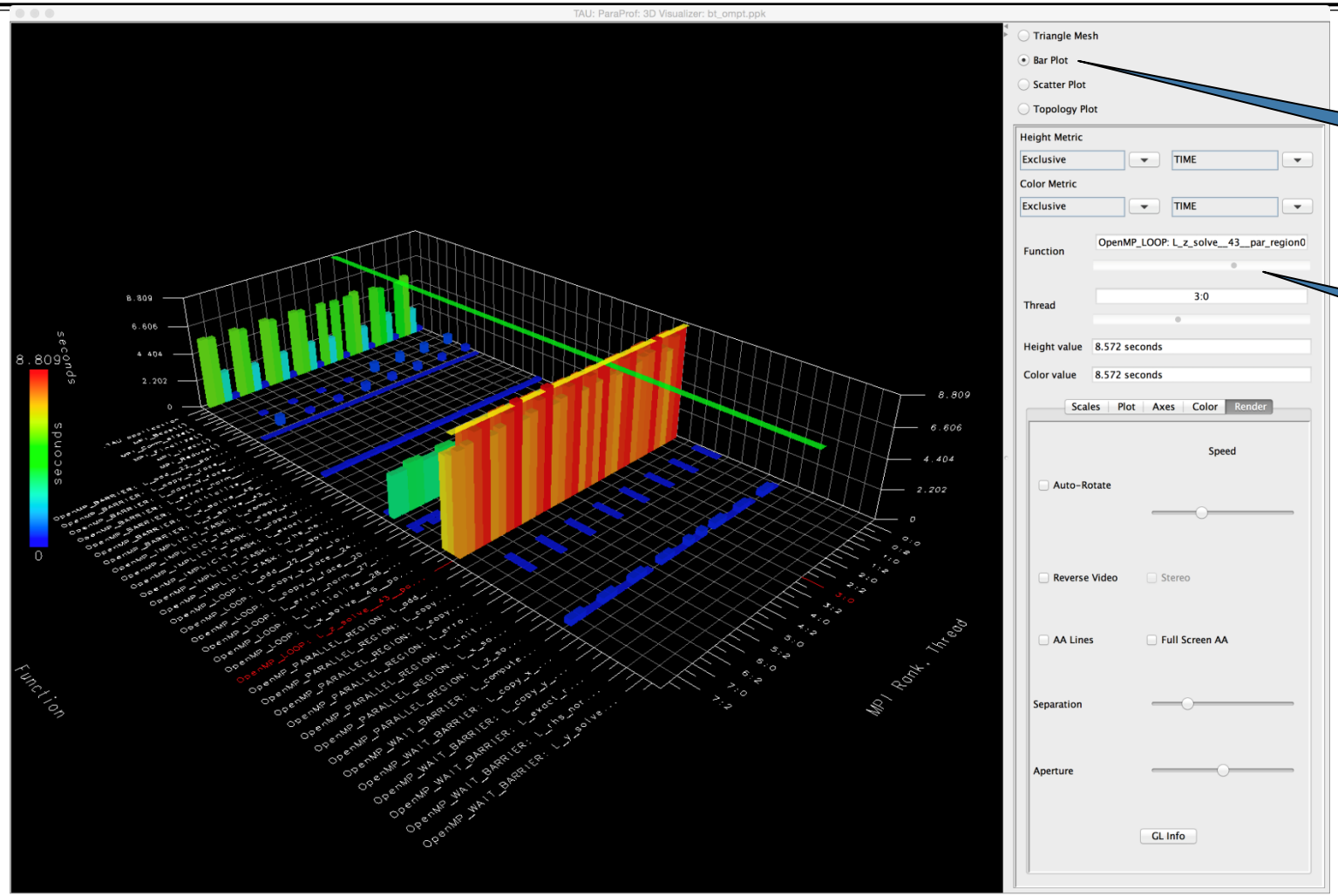
Each routine occupies its own space. Can see the extent of imbalance across all threads.

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

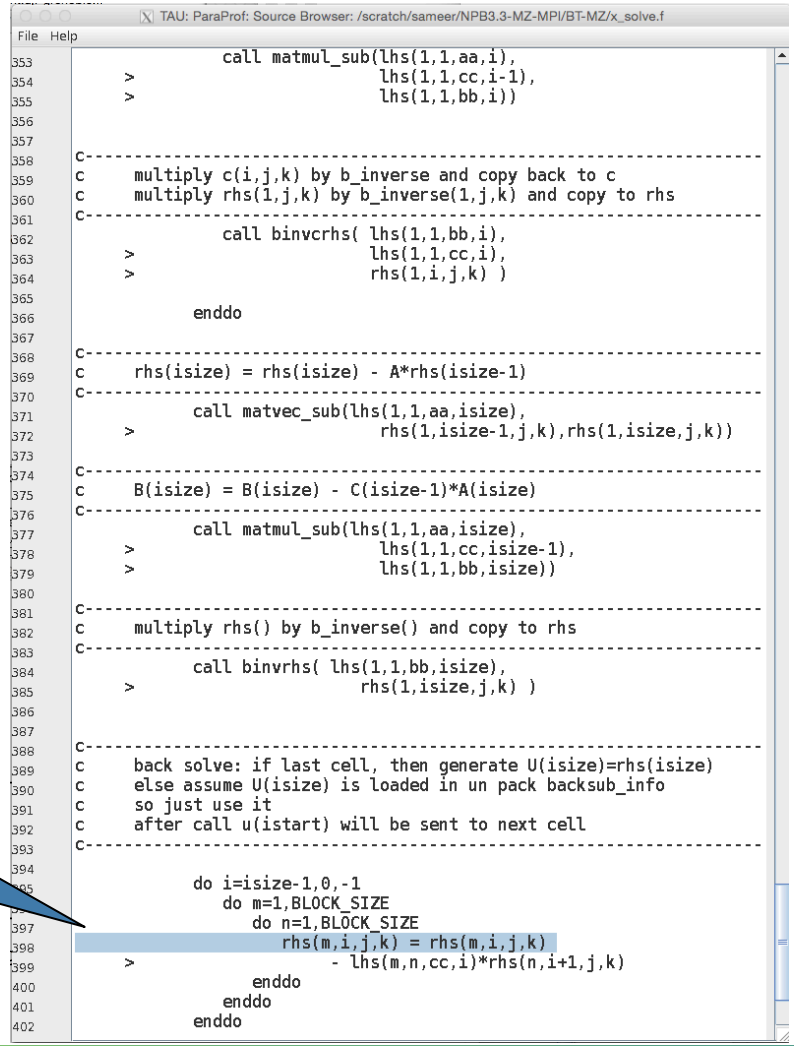
Windows -> 3D visualization

Paraprof Thread Statistics Table with TAU_SAMPLING=1

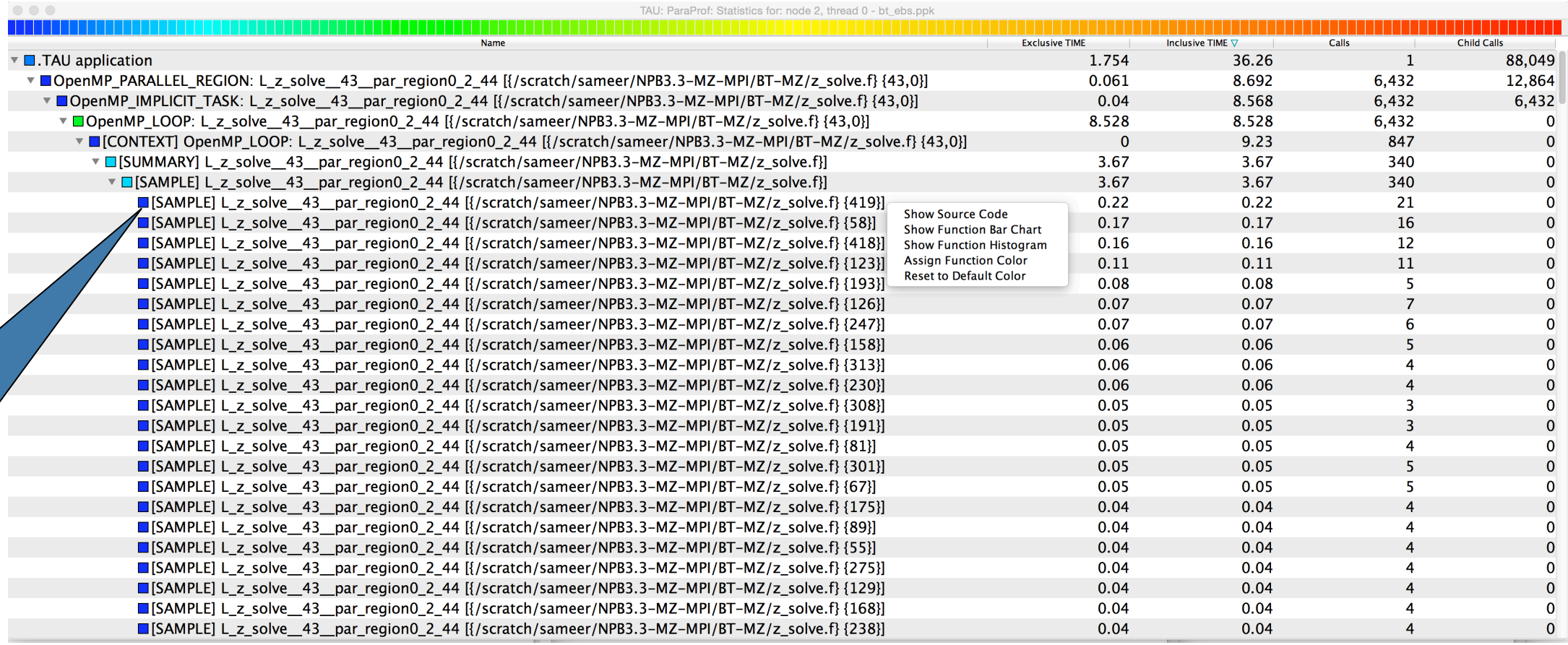
Right click here

Statement Level Profiling with TAU

Source location where samples are taken. Compute intensive region.



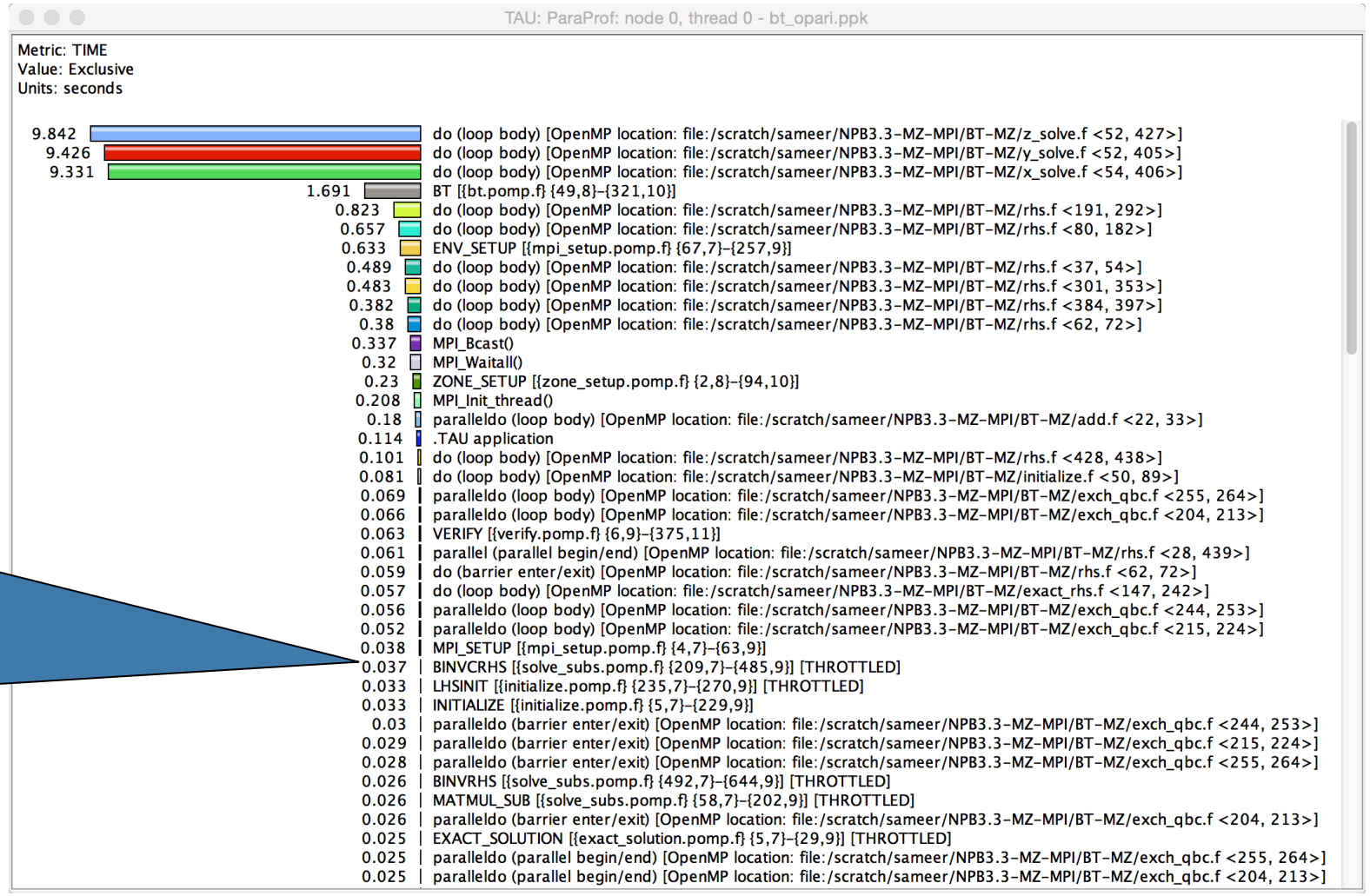
Paraprof Thread Statistics Table



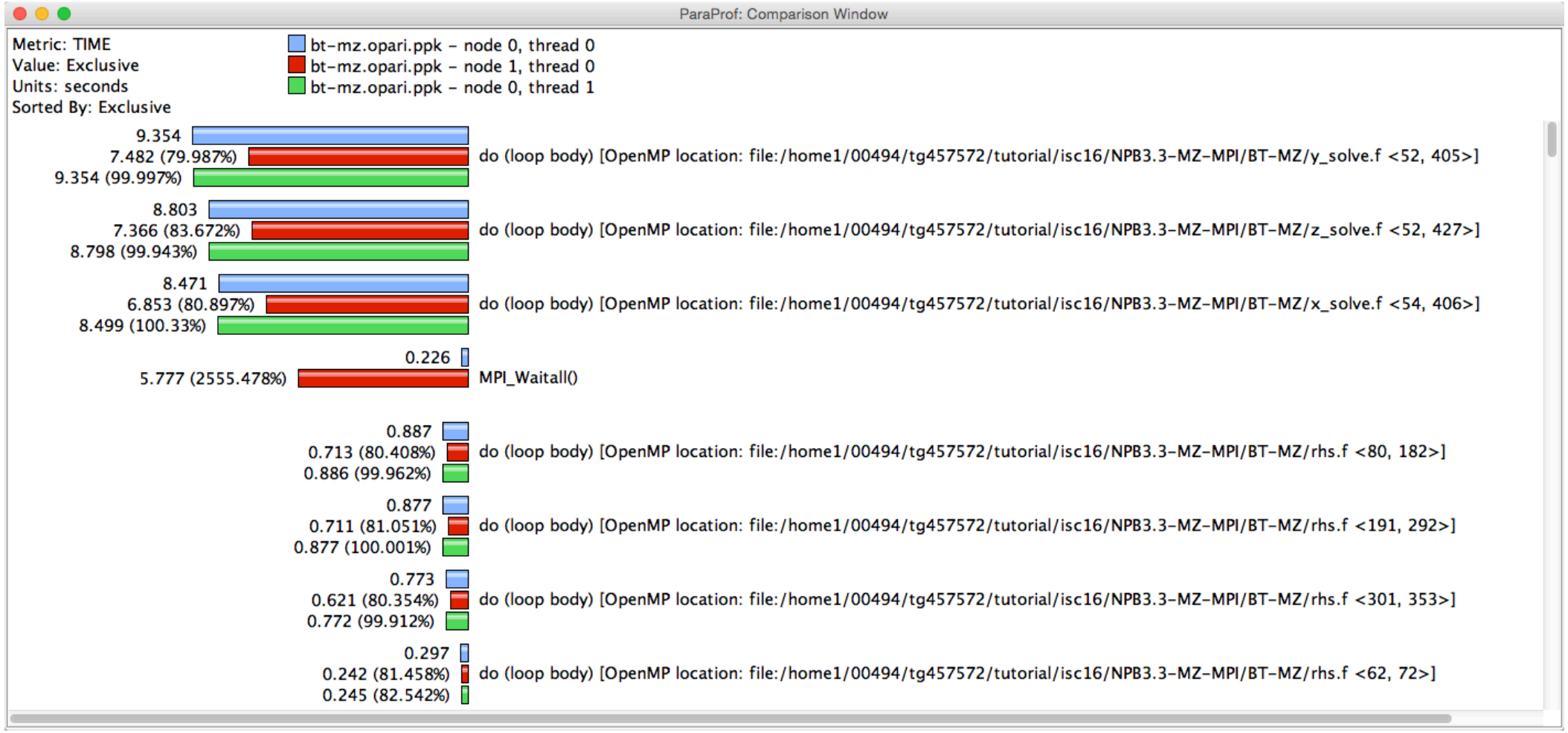
Right click here and choose “Show Source Code” for a sample

Instrumenting Source Code with PDT and Opari

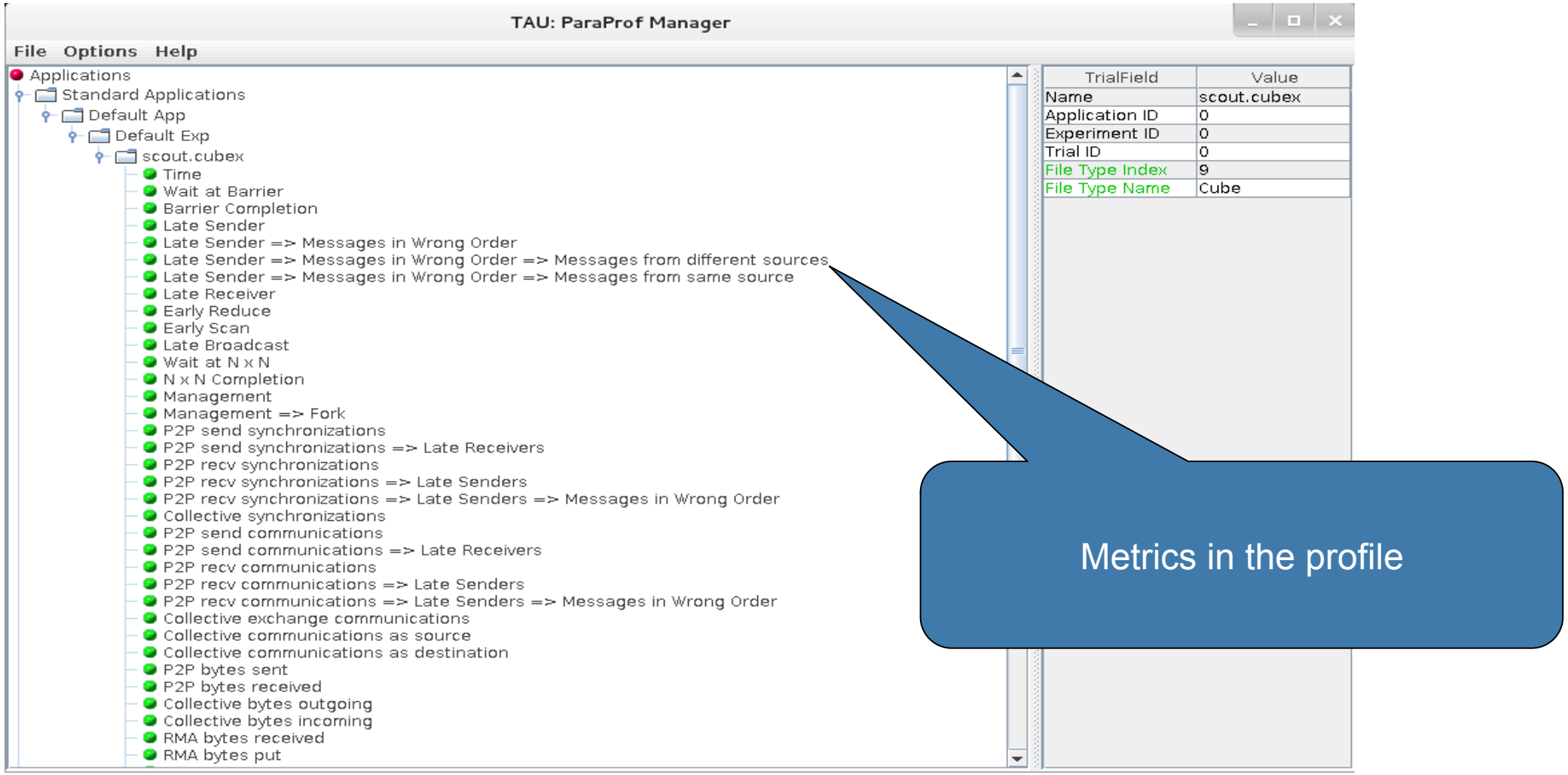
Frequently executing lightweight routines are automatically throttled at runtime. Reduces runtime dilation.



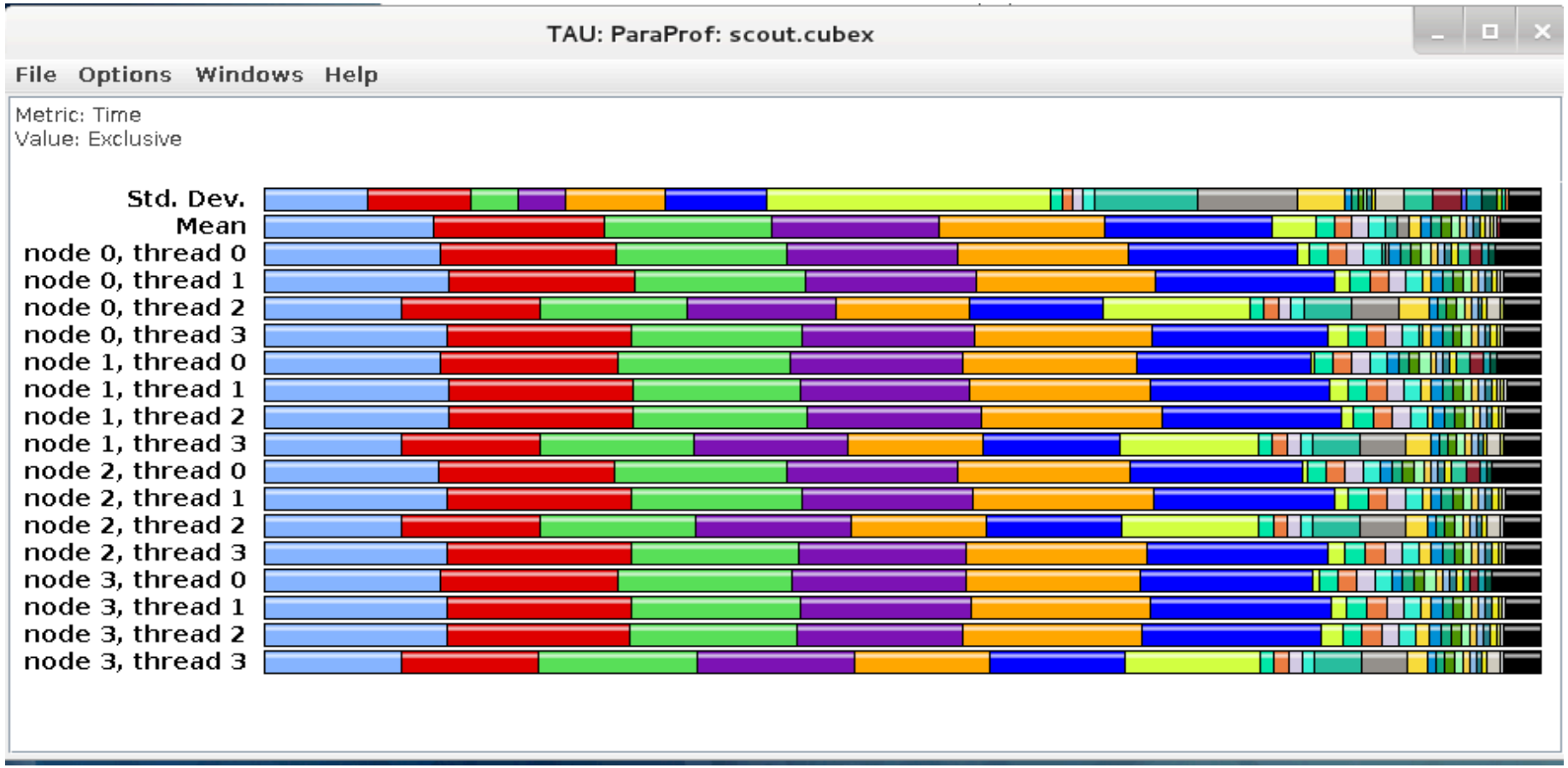
ParaProf Comparison Window



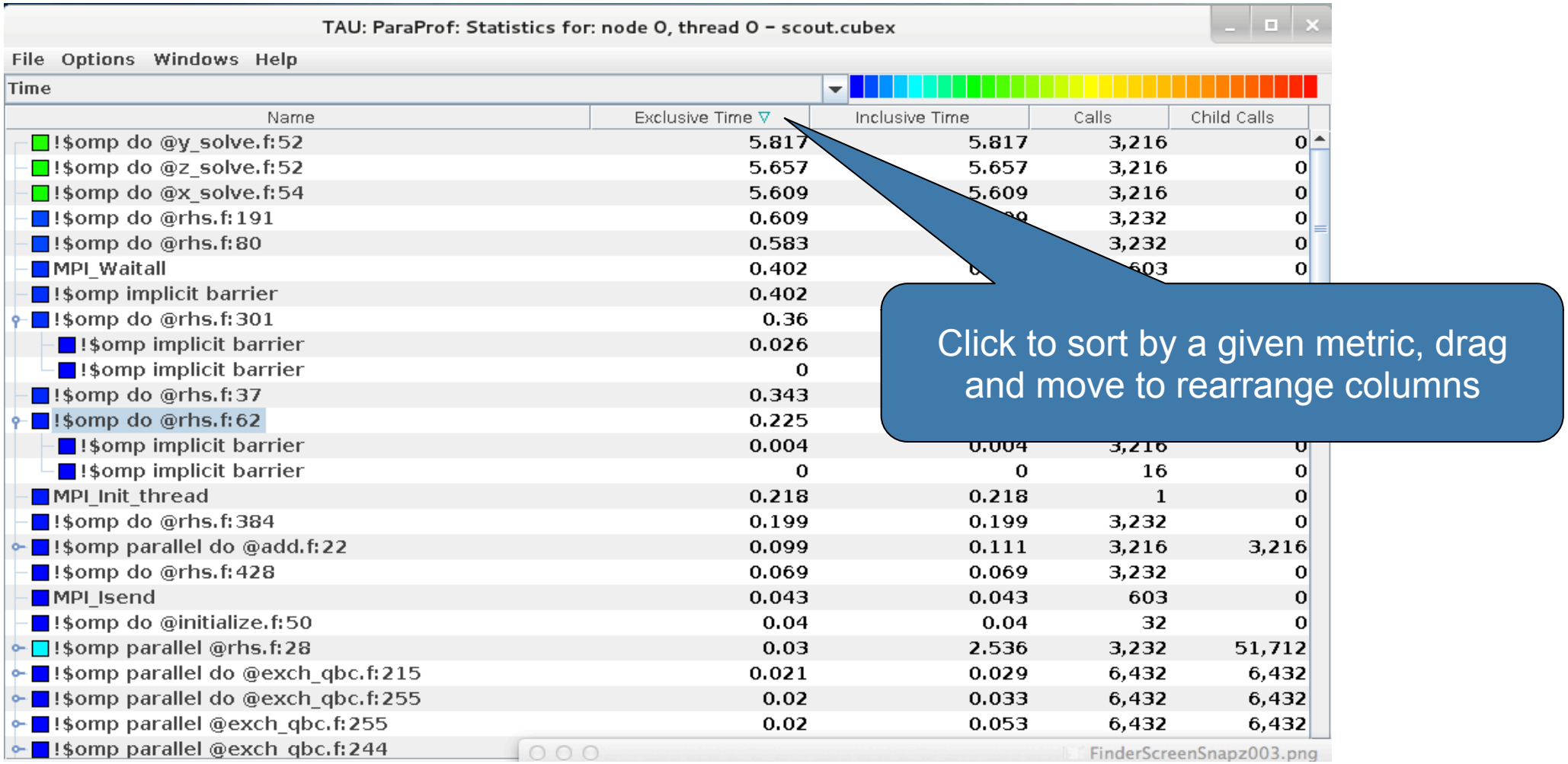
ParaProf Manager Widow: scout.cubex



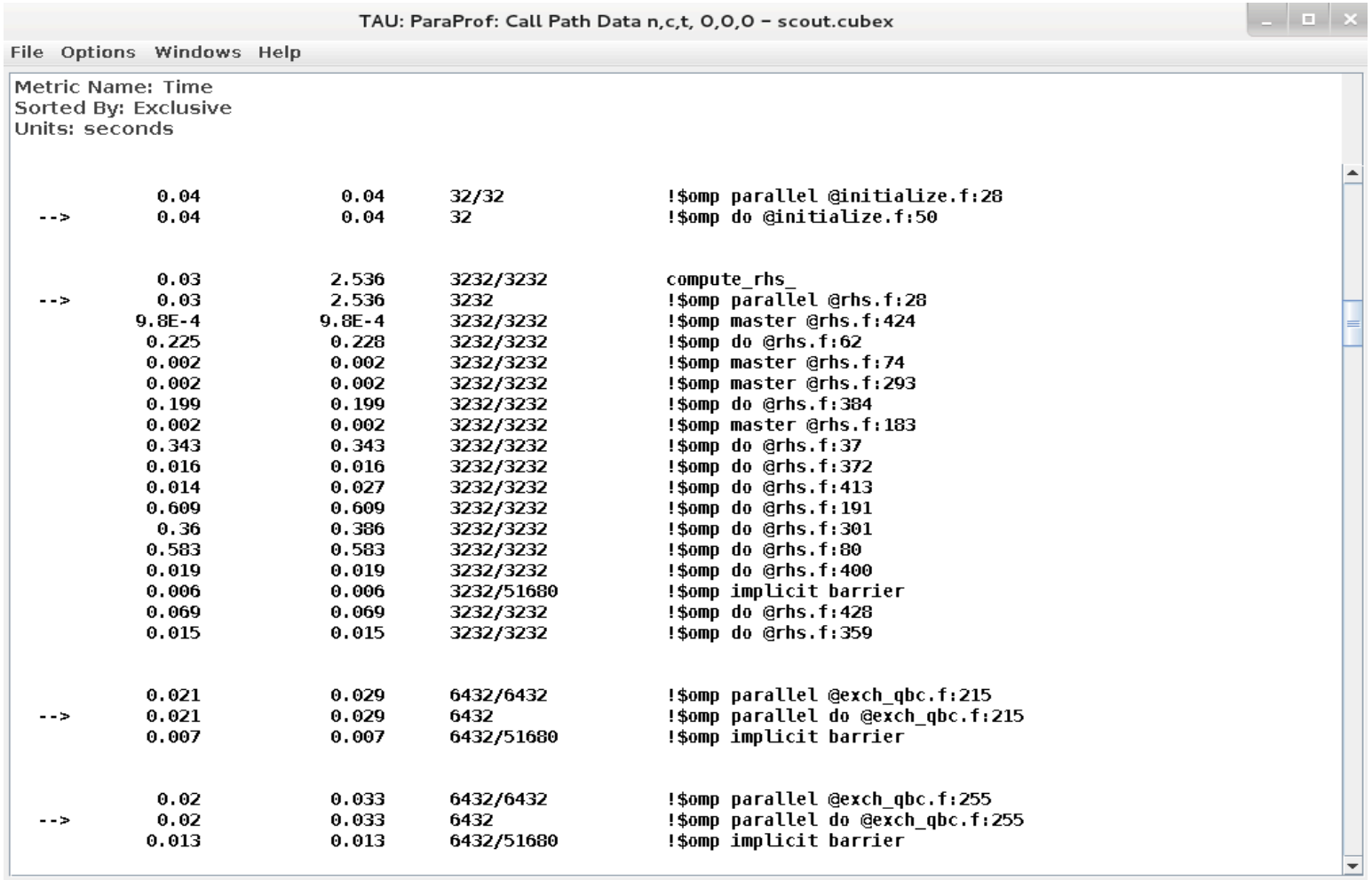
ParaProf: Main Window



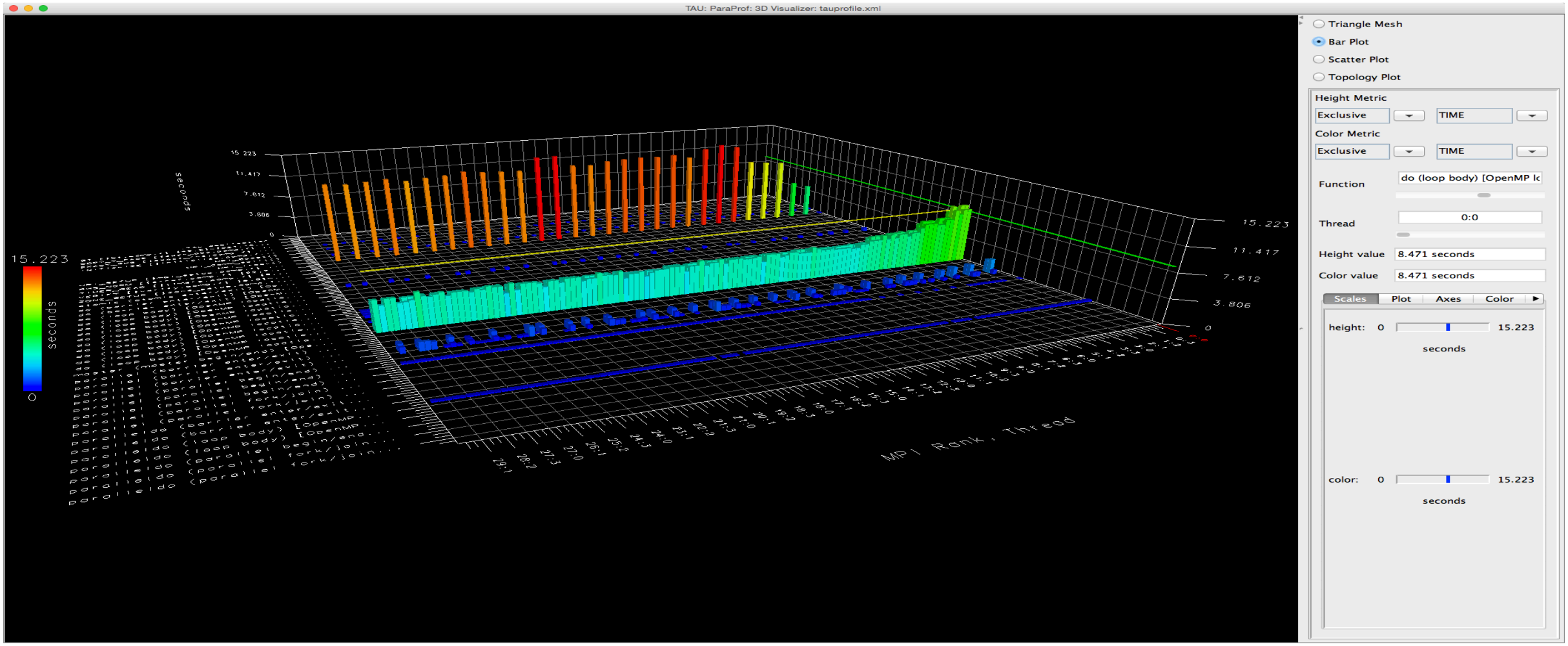
ParaProf: Thread Statistics Table



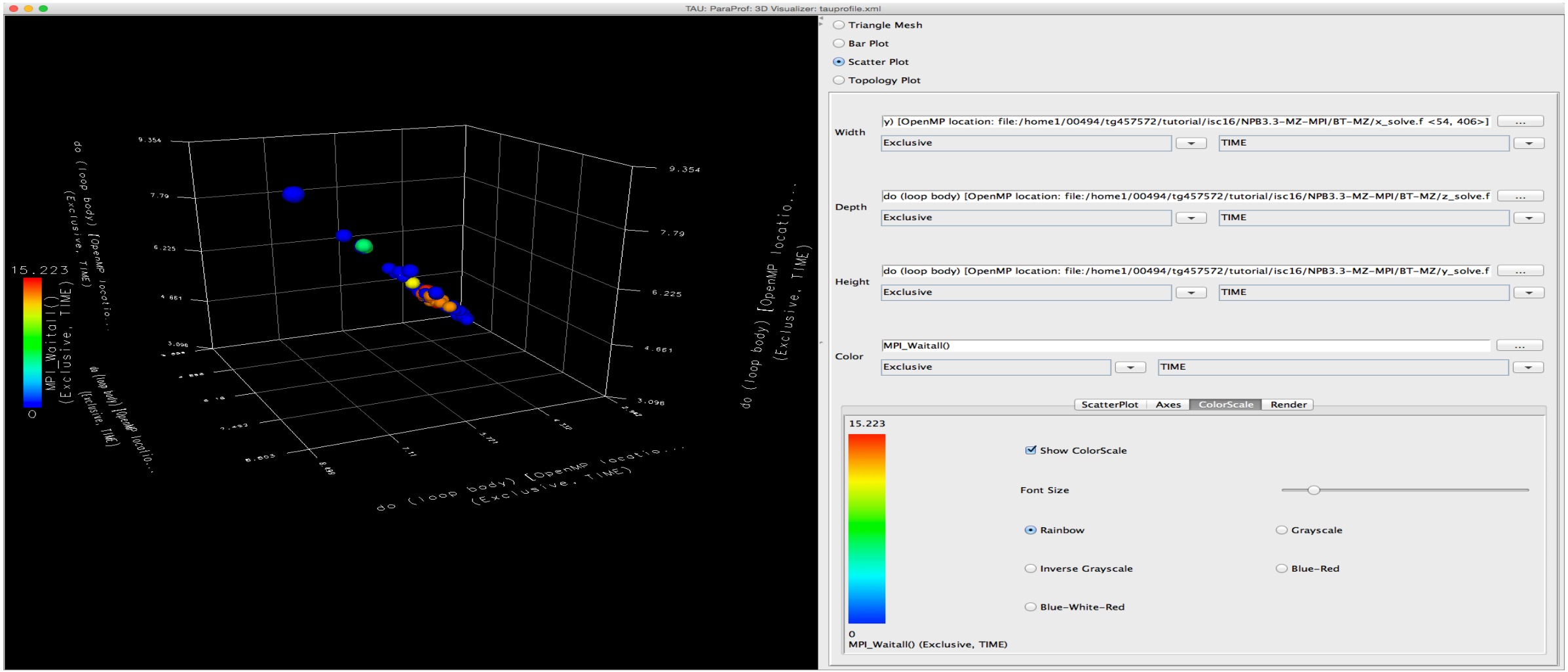
ParaProf: Callpath Thread Relations Window



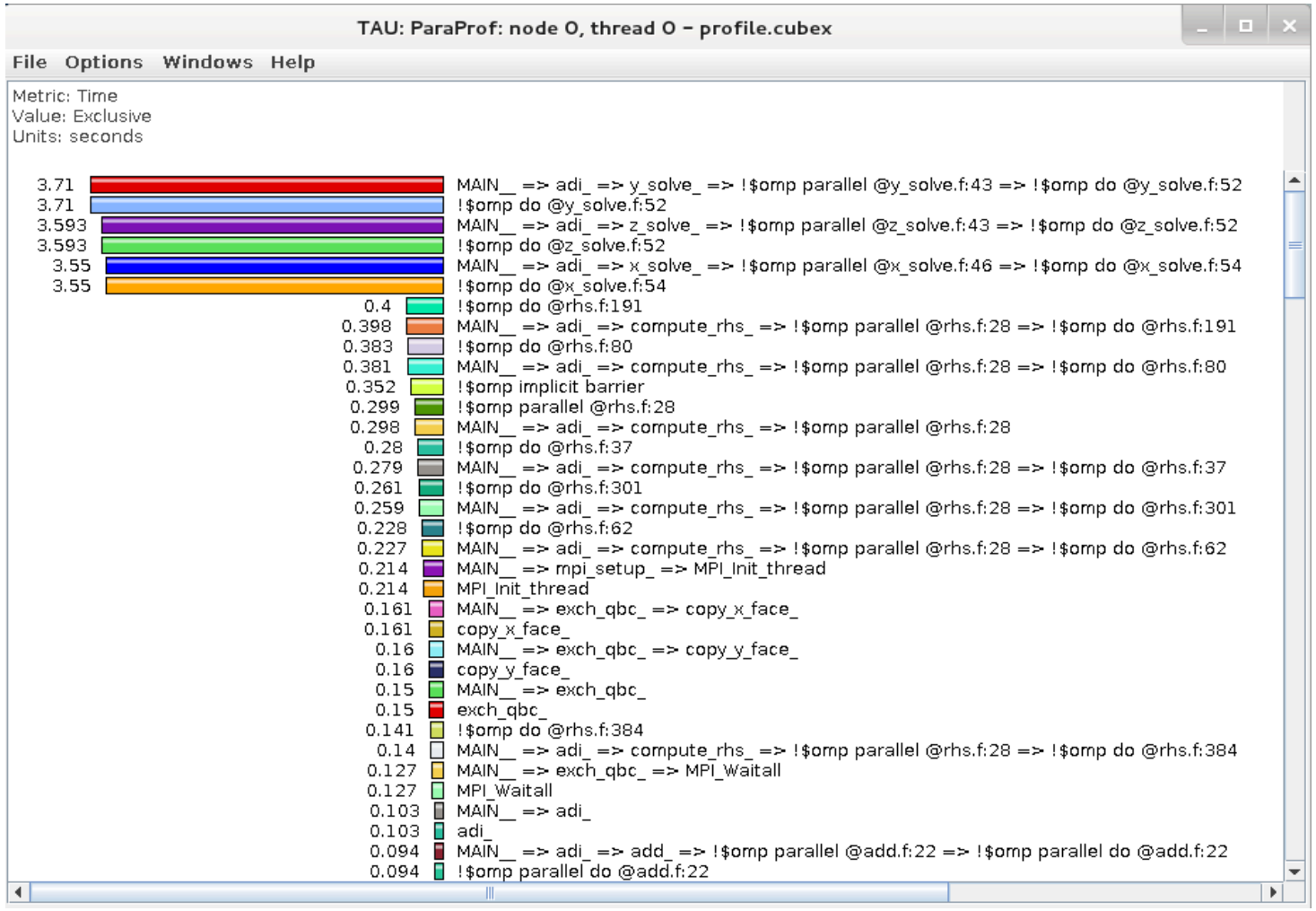
ParaProf: 3D Visualization Window Showing Entire Profile



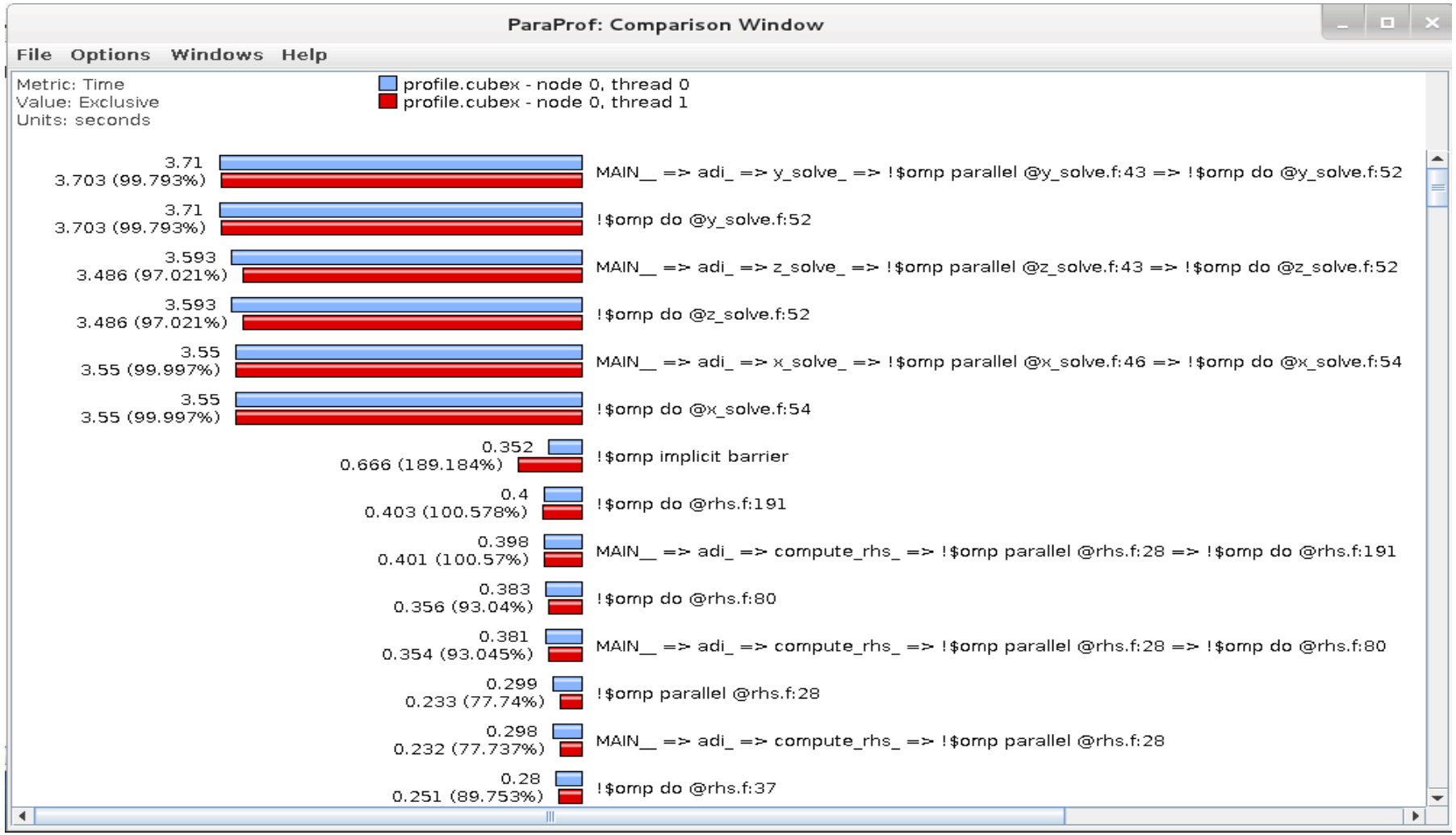
ParaProf: 3D Scatter Plot



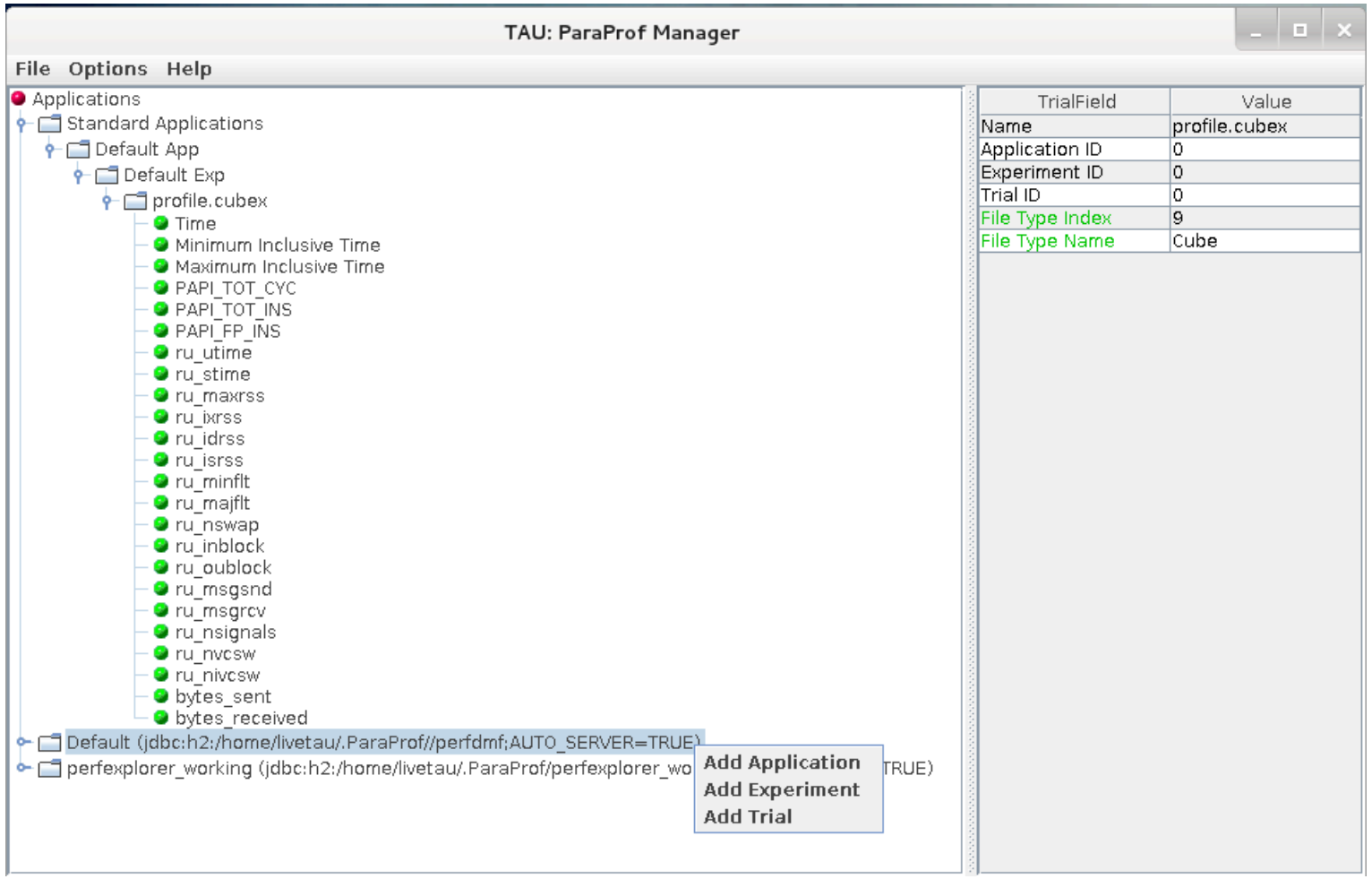
ParaProf: Node View



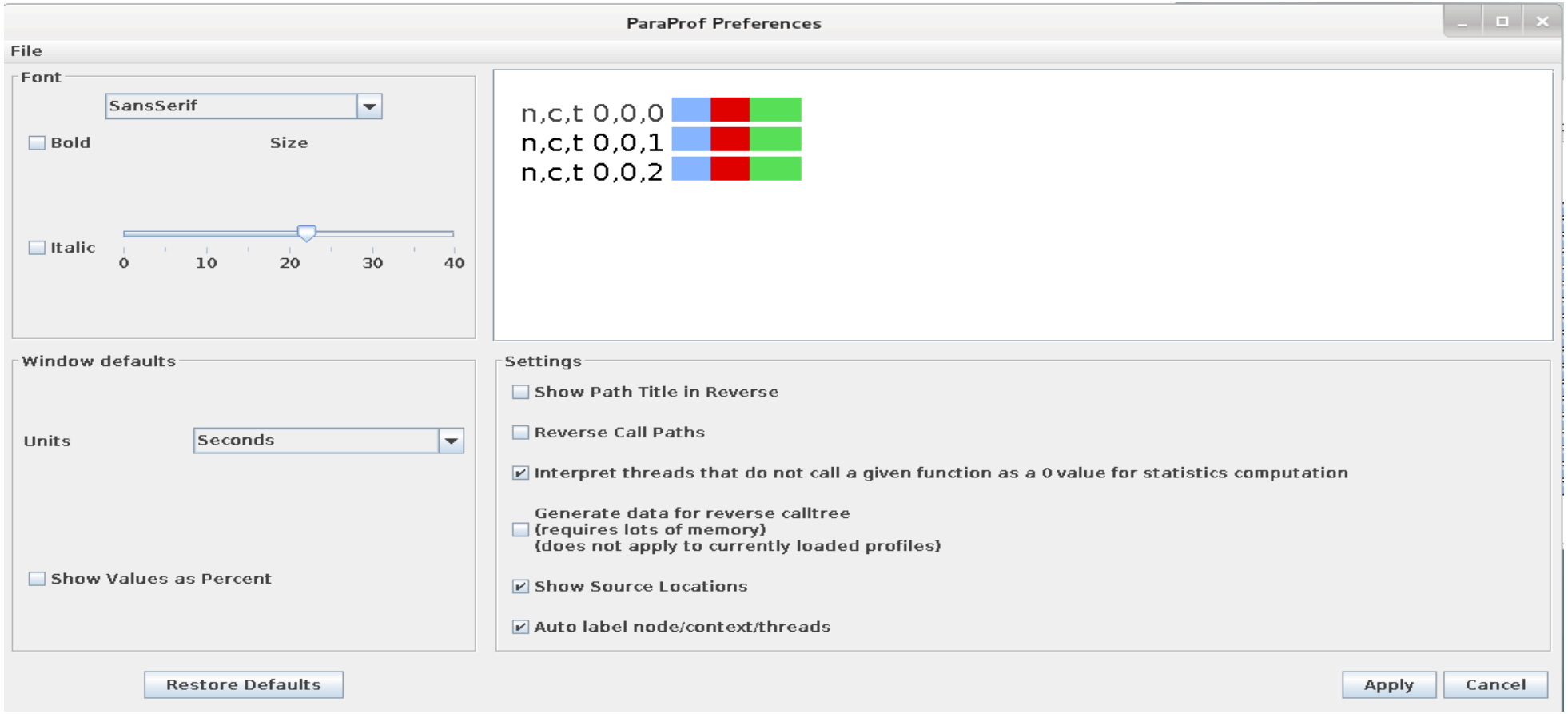
ParaProf: Add thread to comparison window



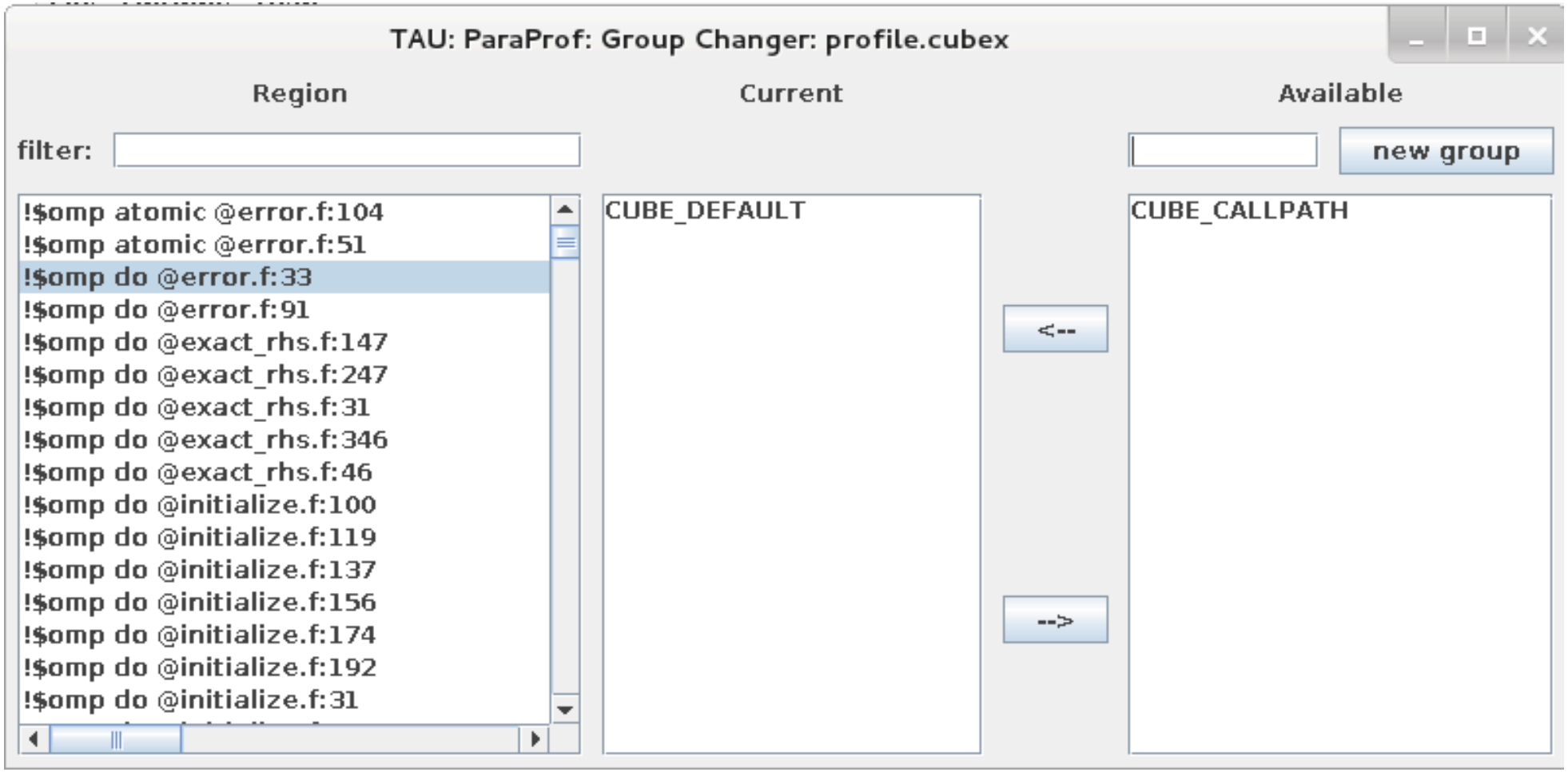
ParaProf: Score-P Profile Files, Database



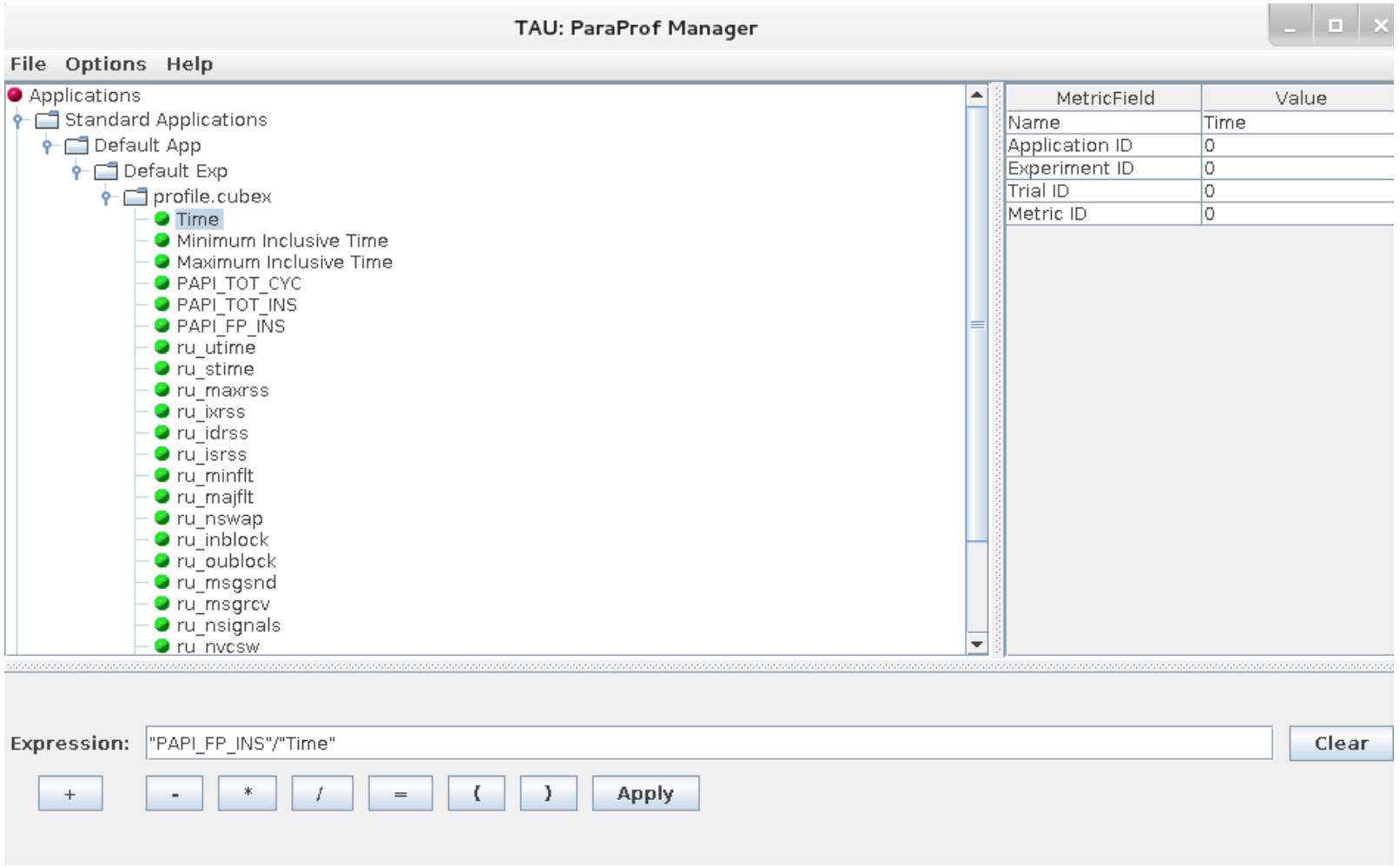
ParaProf: File Preferences Window



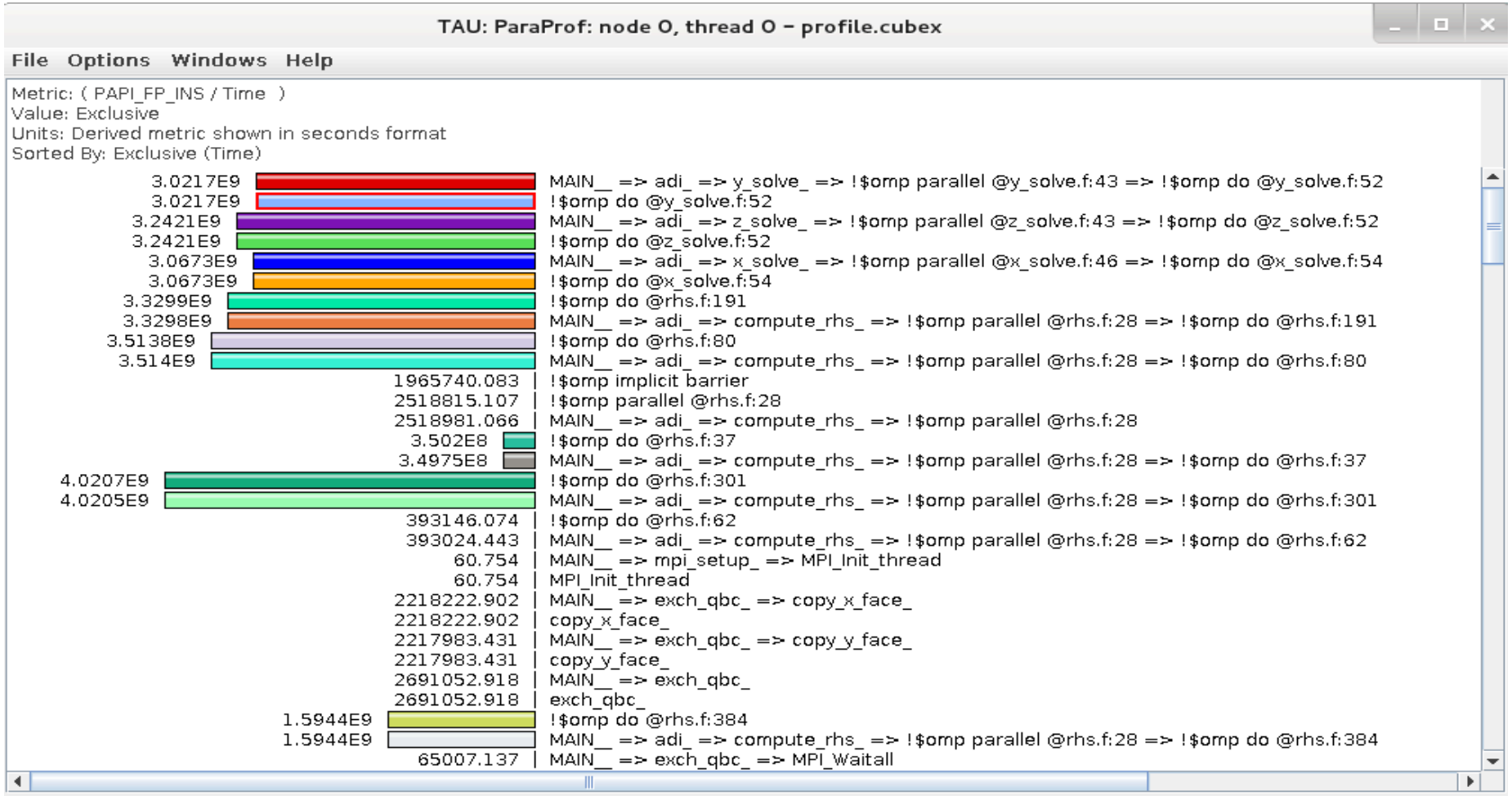
ParaProf: Group Changer Window



ParaProf: Derived Metric Panel in Manager Window

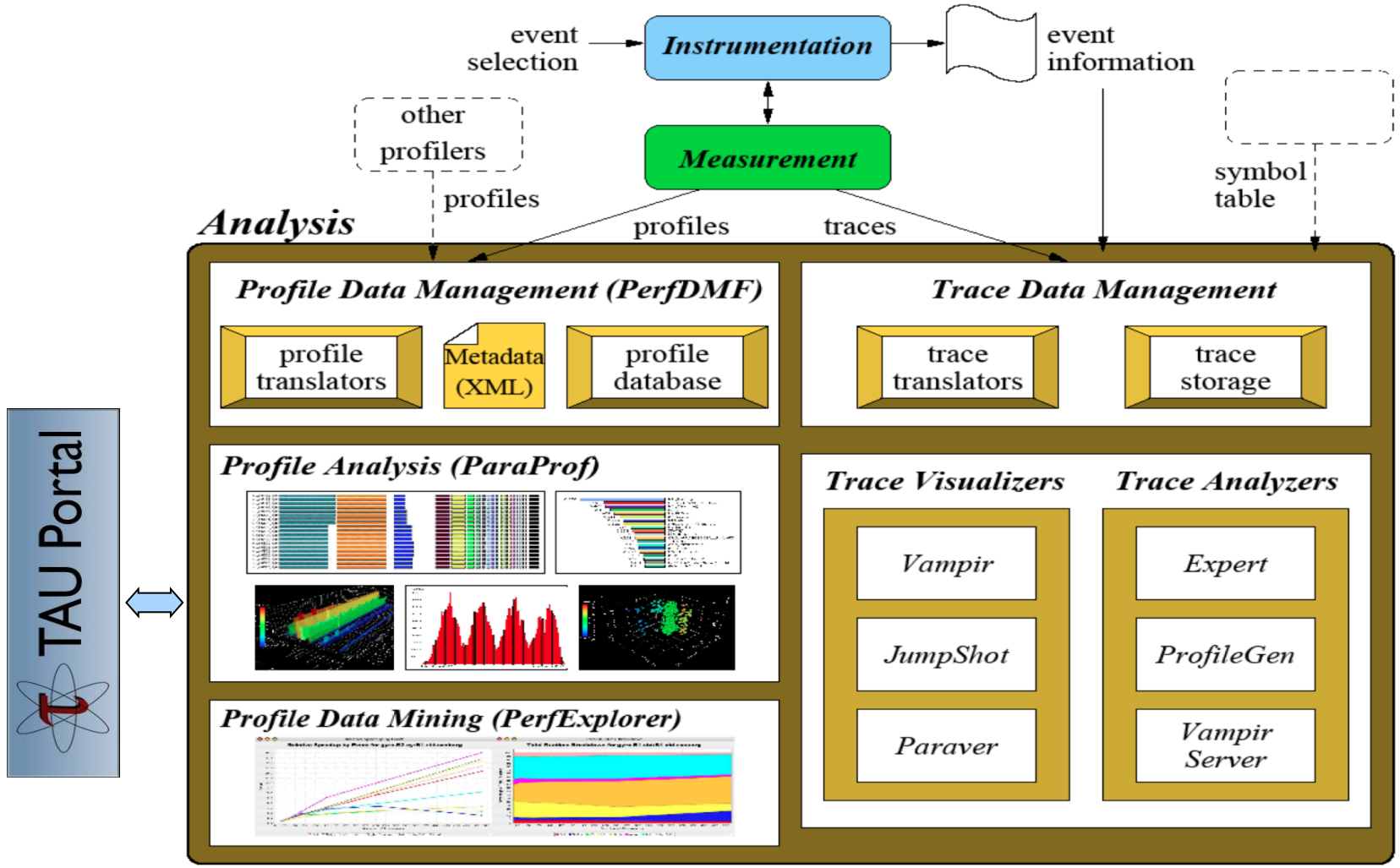


Sorting Derived FLOPS metric by Exclusive Time

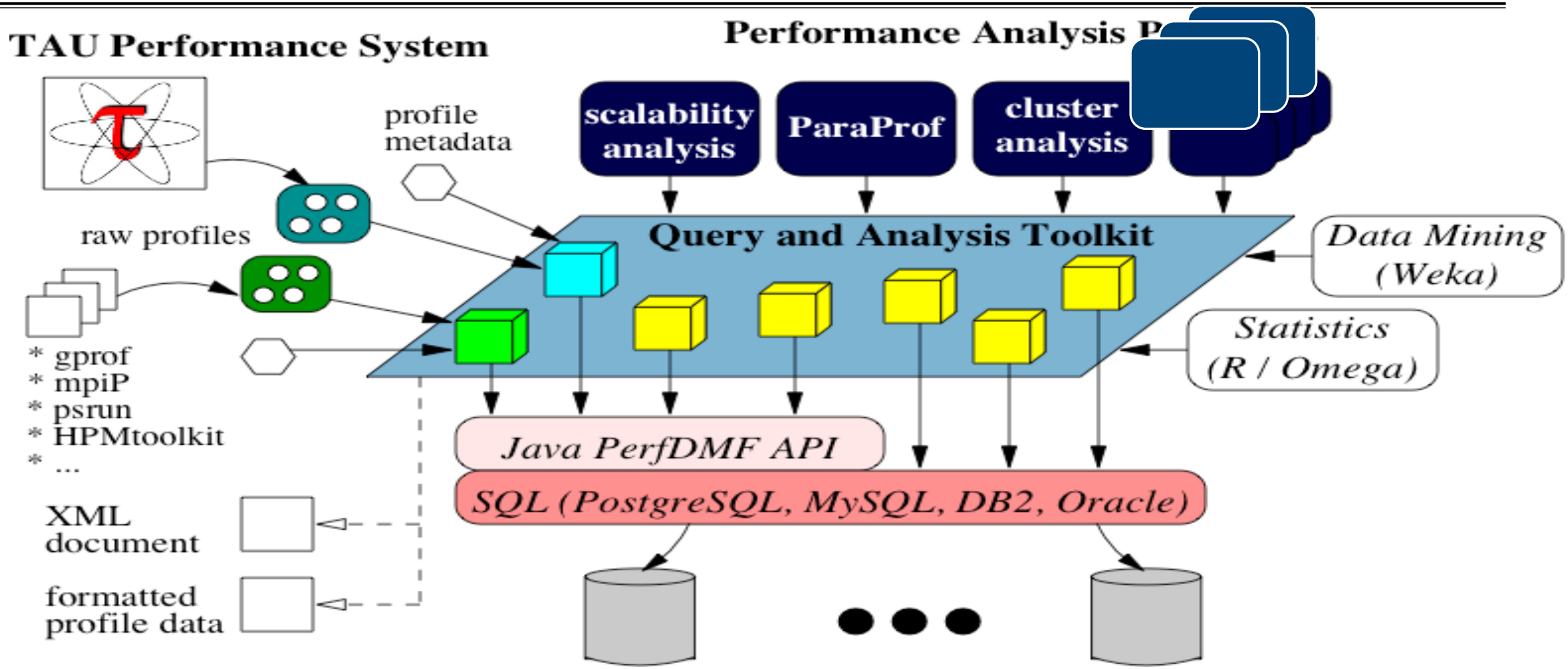


TAU's Analysis Tools: PerfExplorer

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

Local Installation (*Inti, CEA*)

- Setup preferred program environment compilers

```
% cp ~lu23vox/data.tgz . ; tar zxf data.tgz
% cd data
% cat README
  and follow the steps
% cd tau
% ./upload.sh
% perfexplorer
```

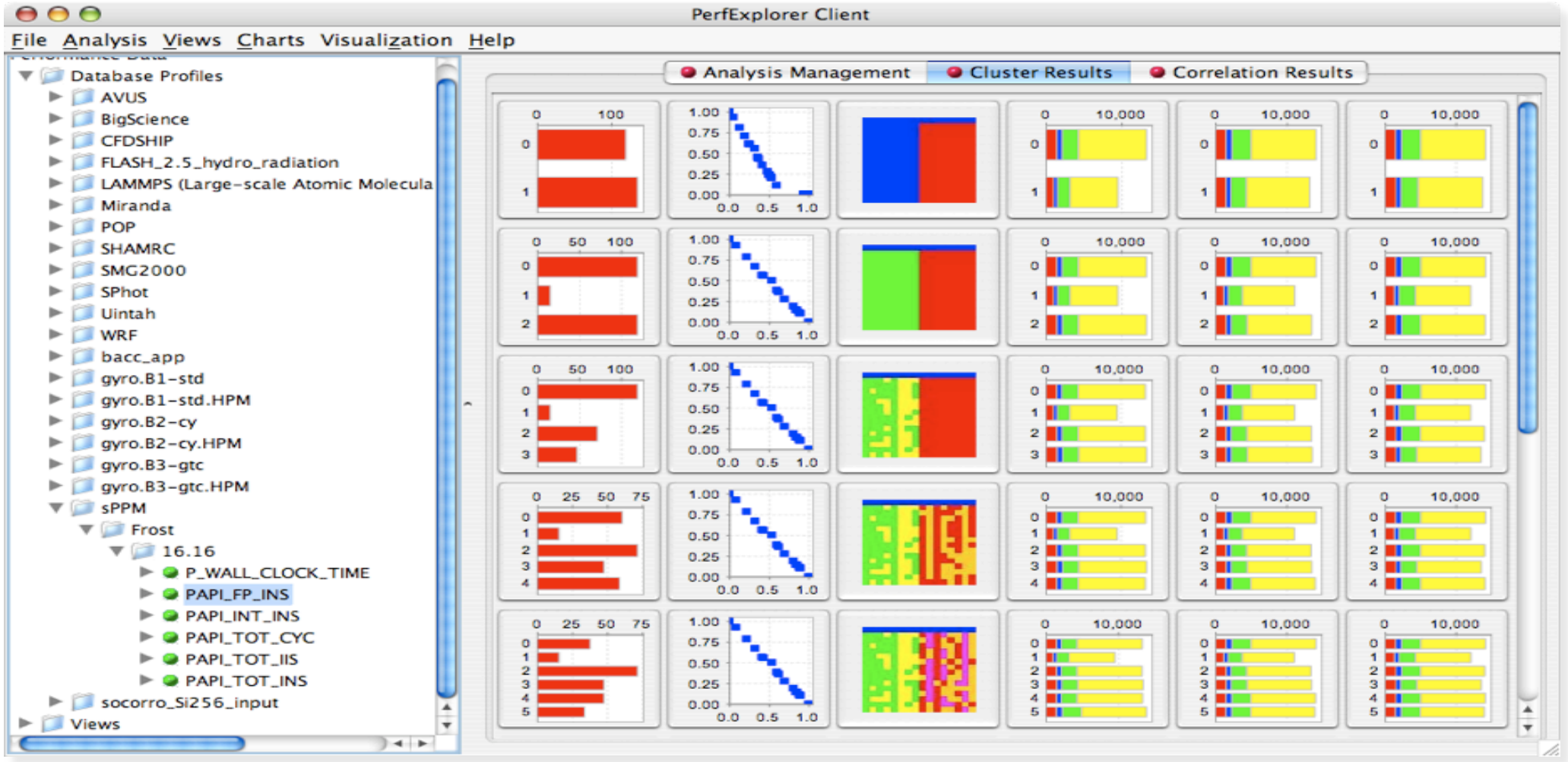
Performance Data Mining (PerfExplorer)

- 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

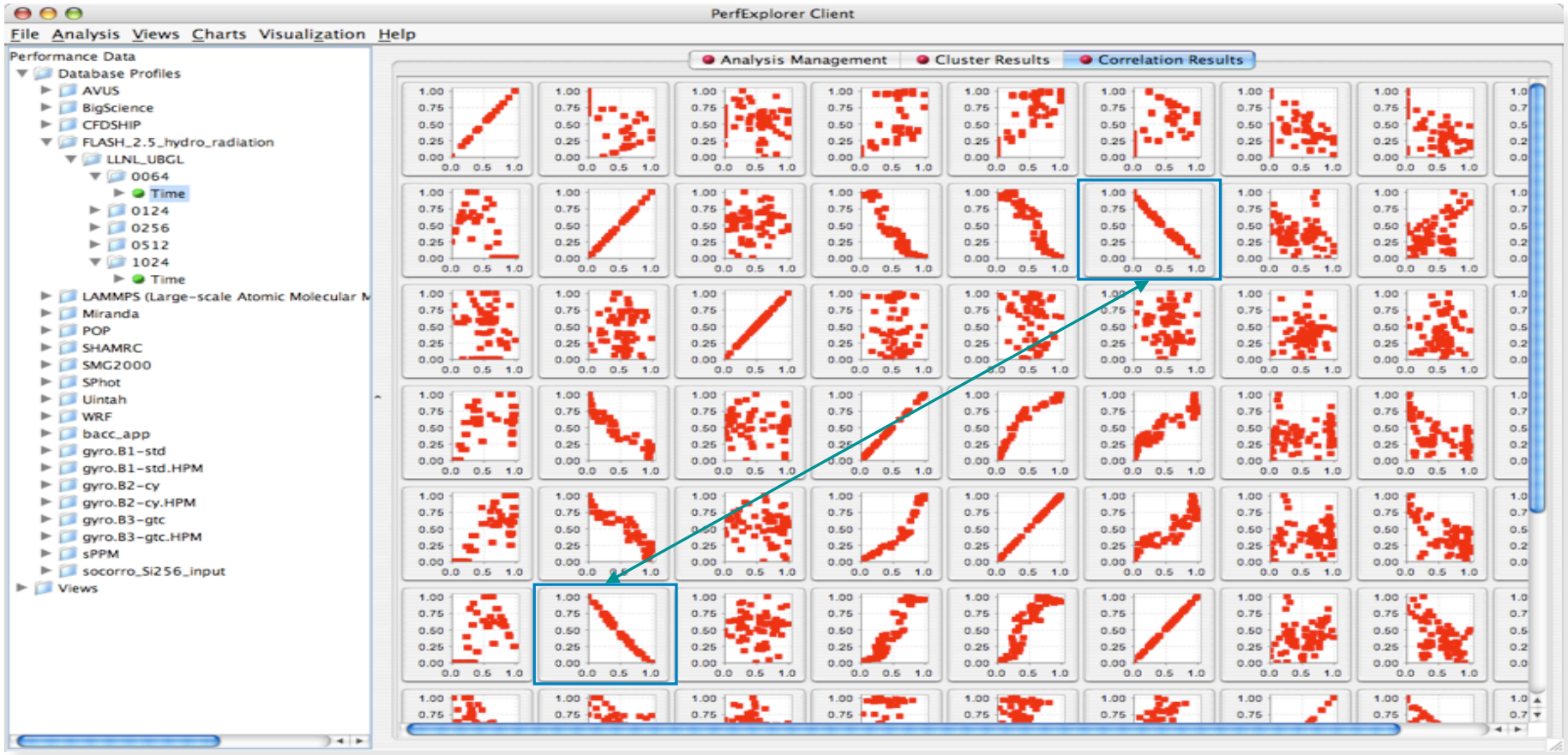
- Performance data represented as vectors - each dimension is the cumulative time for an event
- *k*-means: *k* random centers are selected and instances are grouped with the "closest" (Euclidean) center
- New centers are calculated and the process repeated until stabilization or max iterations
- Dimension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

PerfExplorer - Cluster Analysis (sPPM)



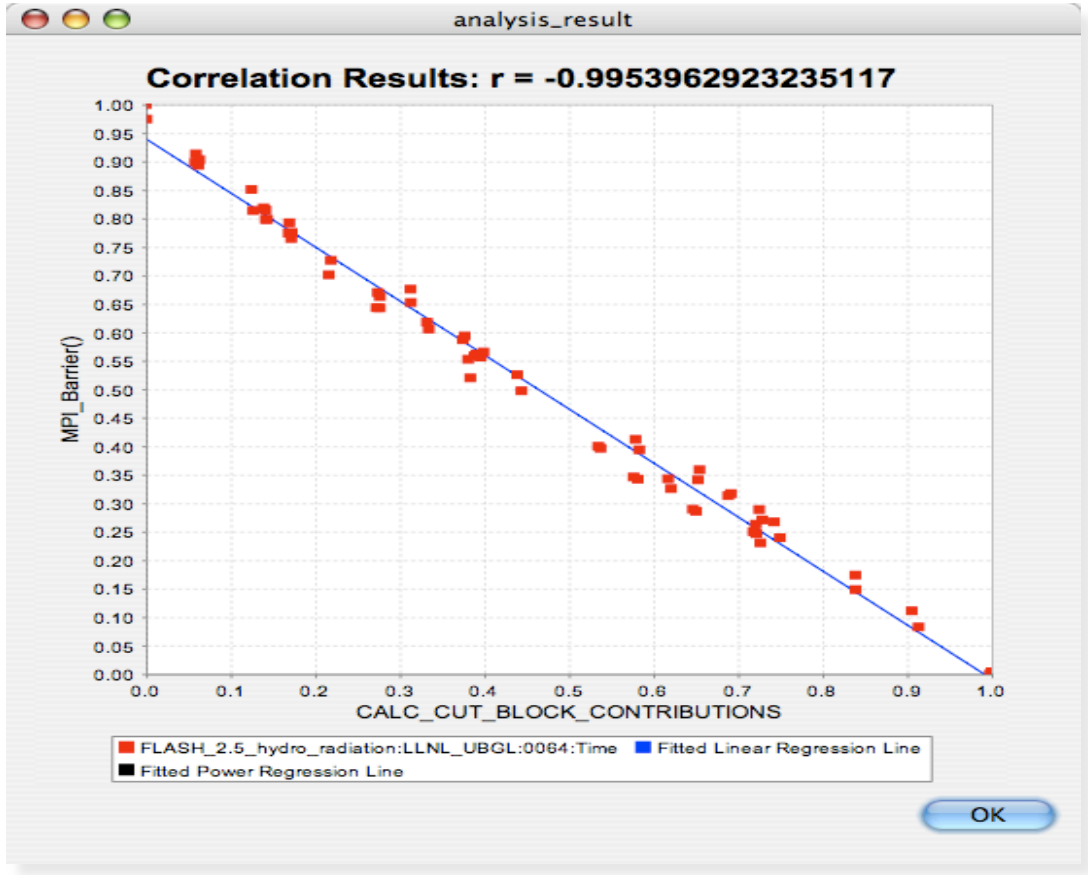
PerfExplorer - Correlation Analysis (Flash)

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



PerfExplorer - Correlation Analysis (Flash)

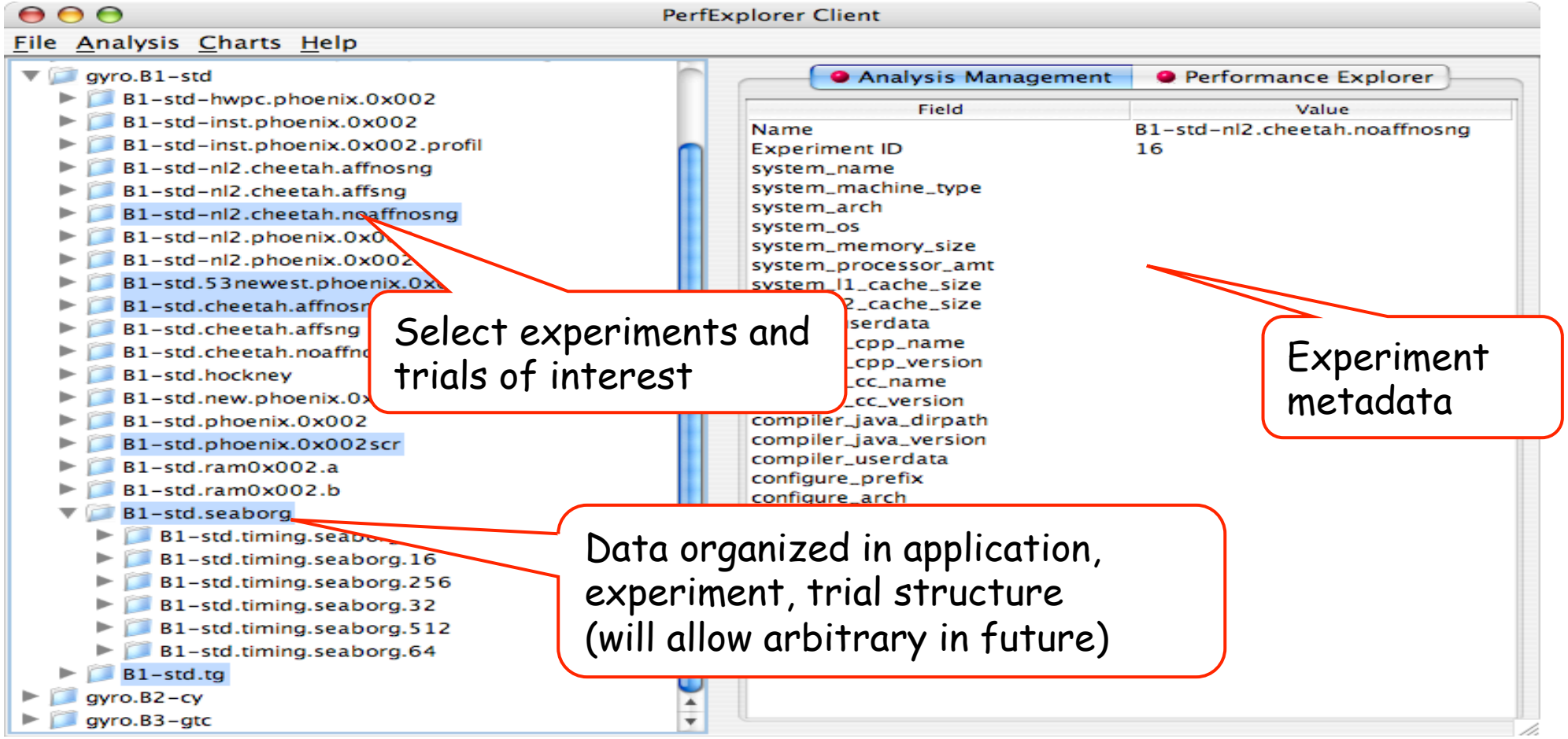
- -0.995 indicates strong, negative relationship
- As CALC_CUT_BLOCK_CONTRIBUTIONS() increases in execution time, MPI_Barrier() decreases



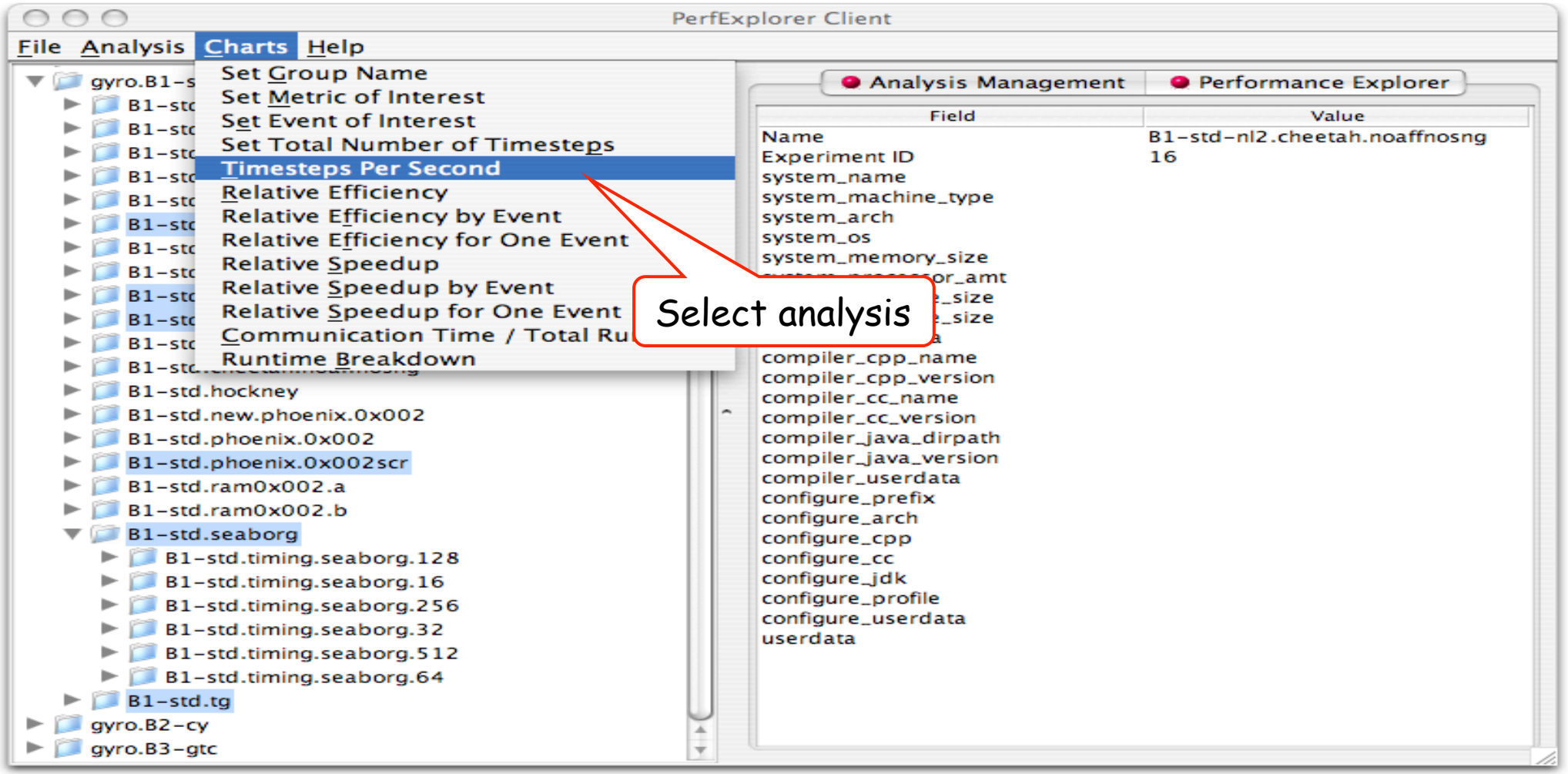
PerfExplorer - Comparative Analysis

- 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

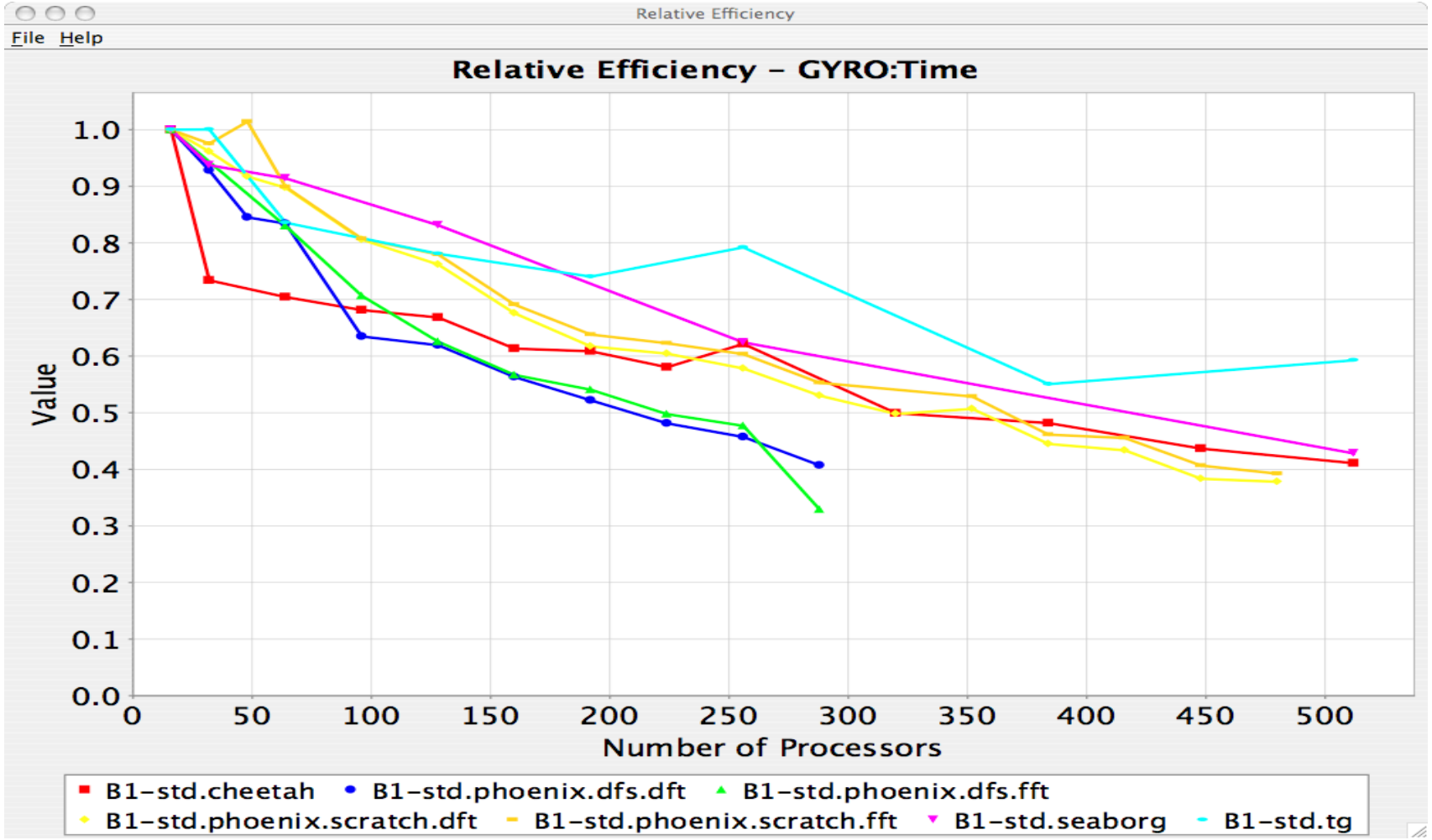
PerfExplorer - Interface



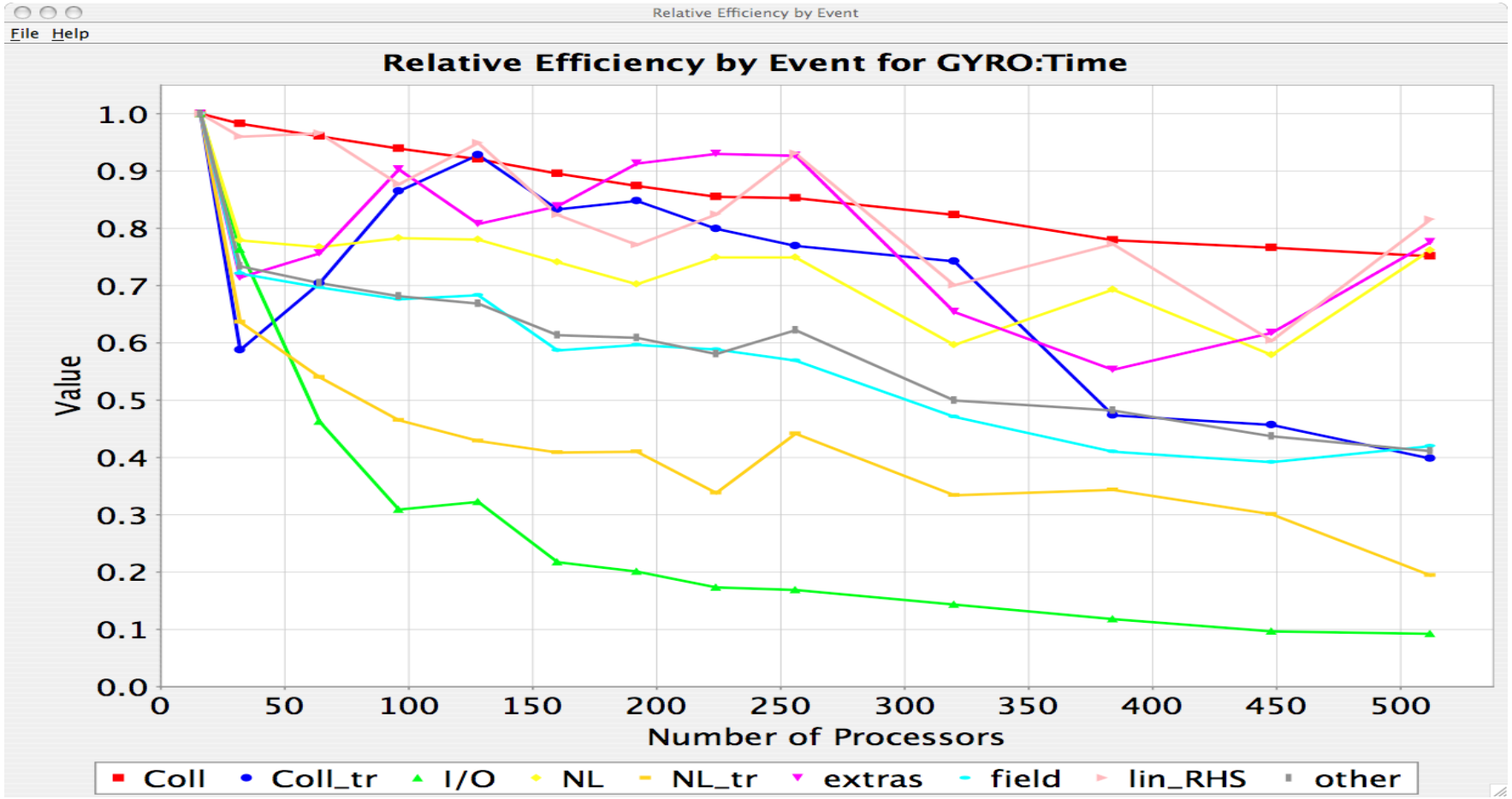
PerfExplorer - Interface



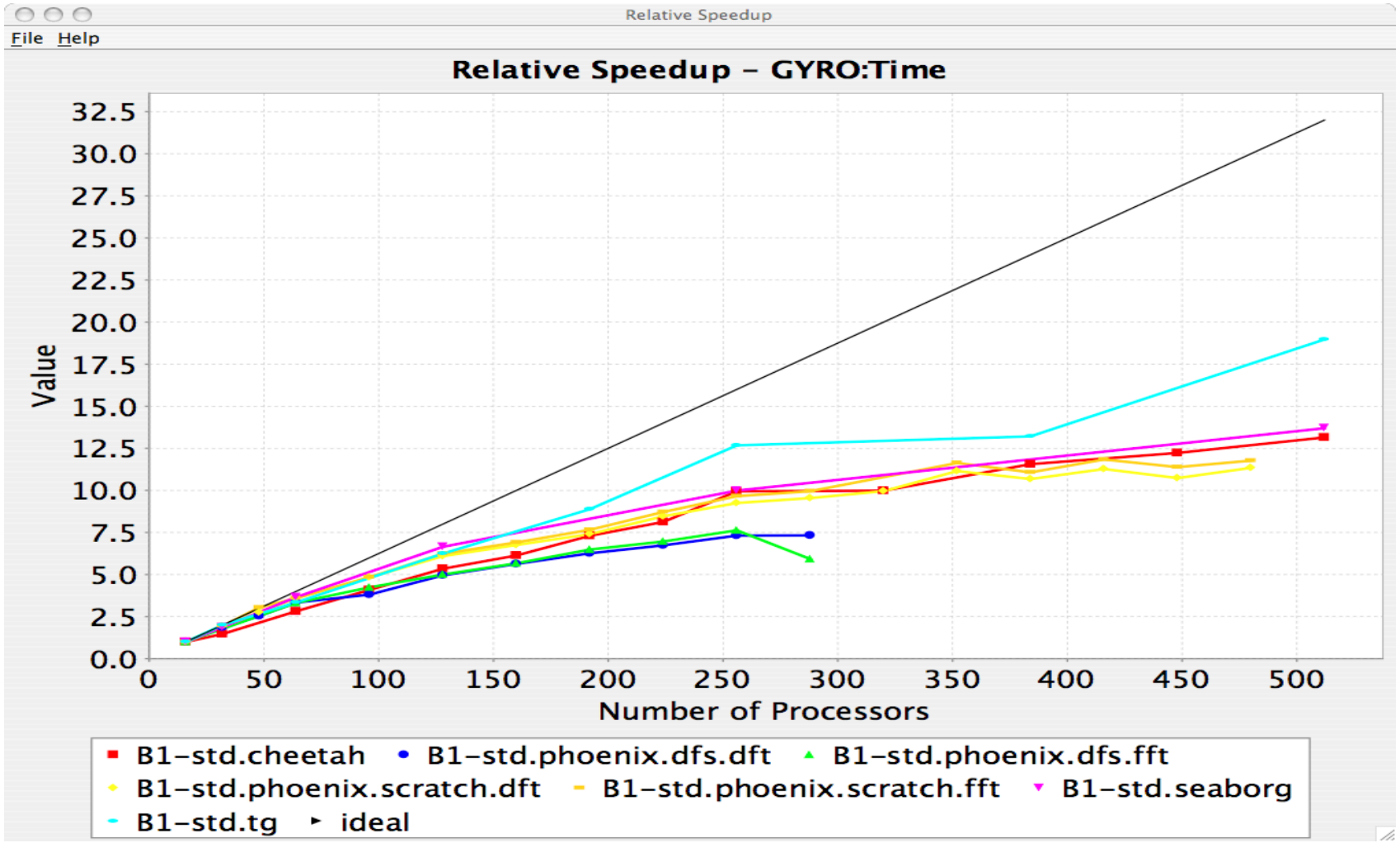
PerfExplorer - Relative Efficiency Plots



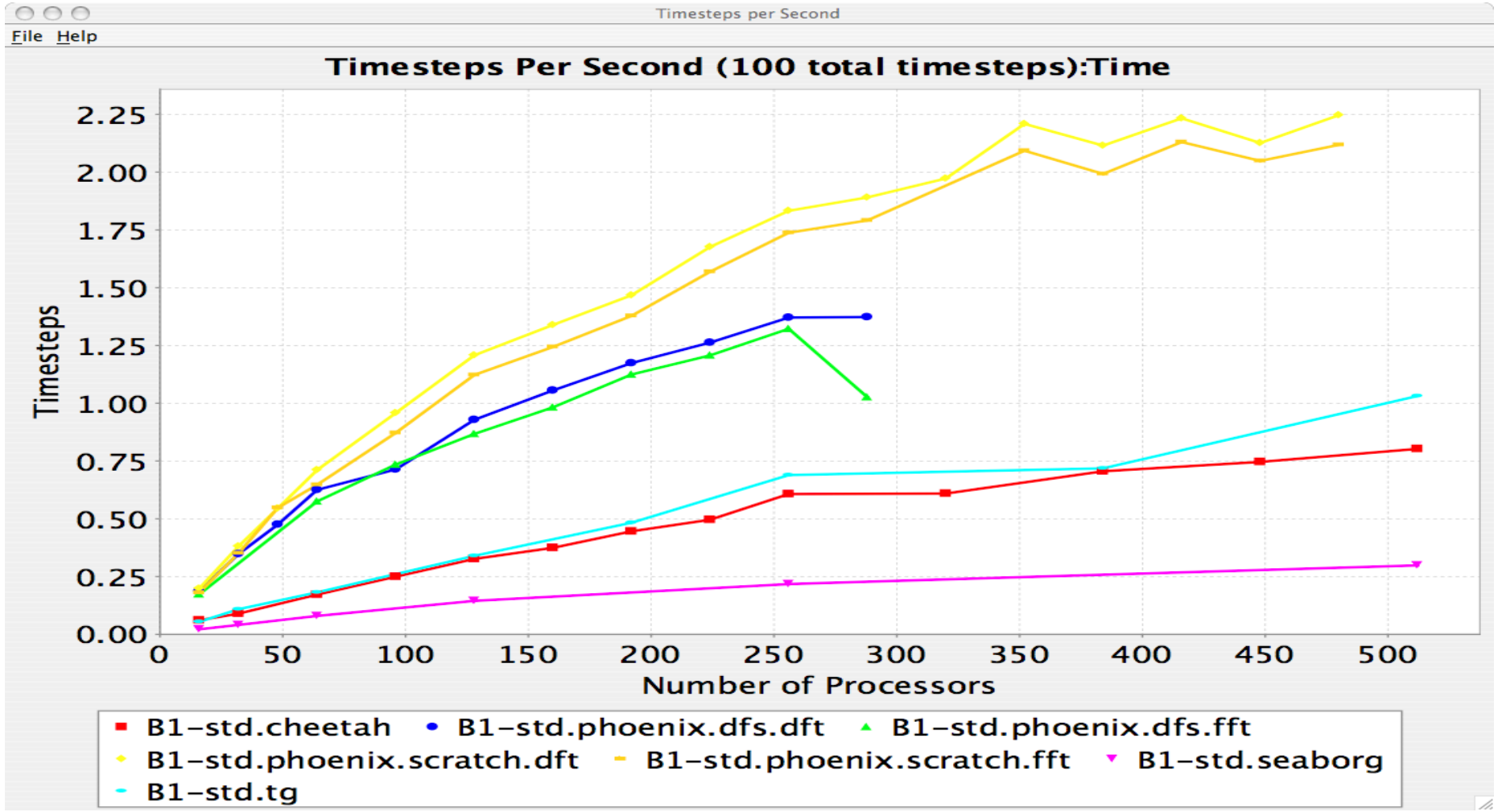
PerfExplorer - Relative Efficiency by Routine



PerfExplorer - Relative Speedup

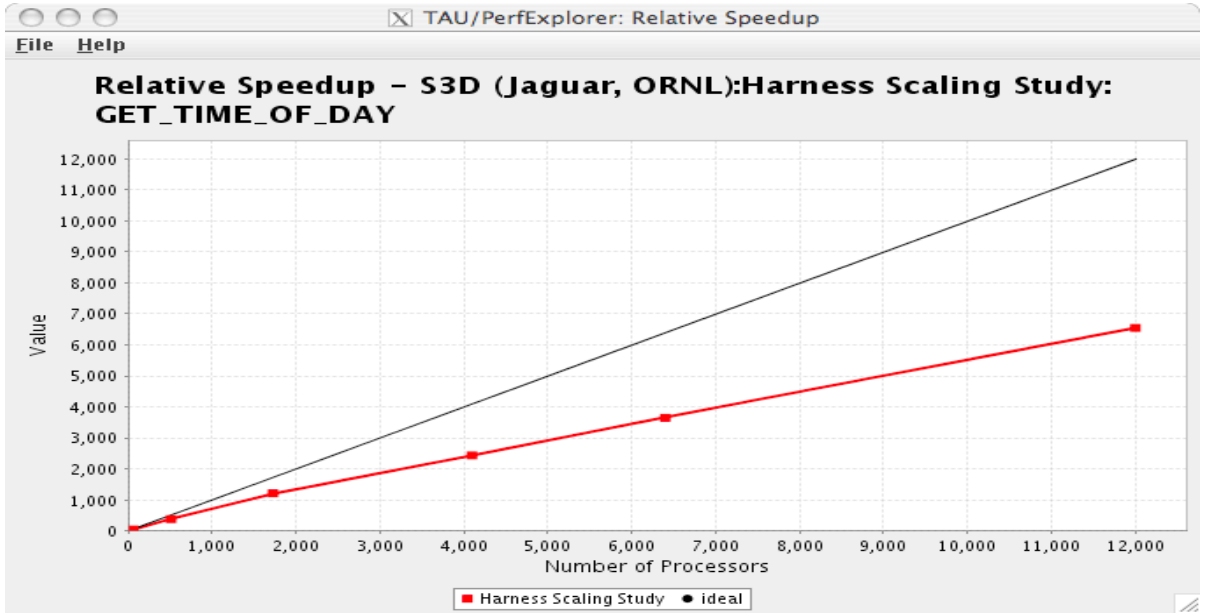


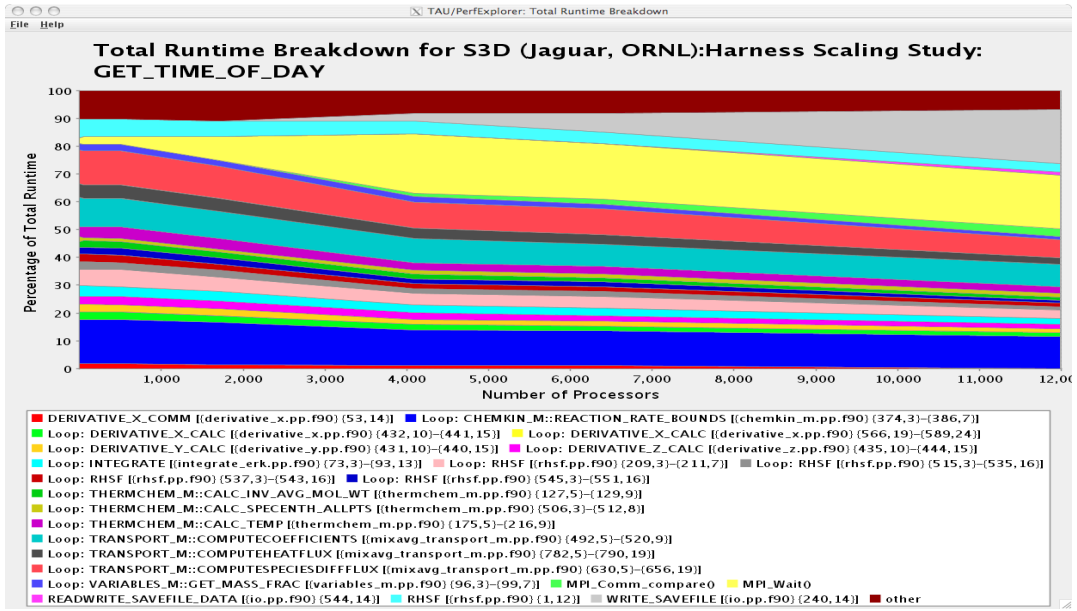
PerfExplorer - Timesteps Per Second



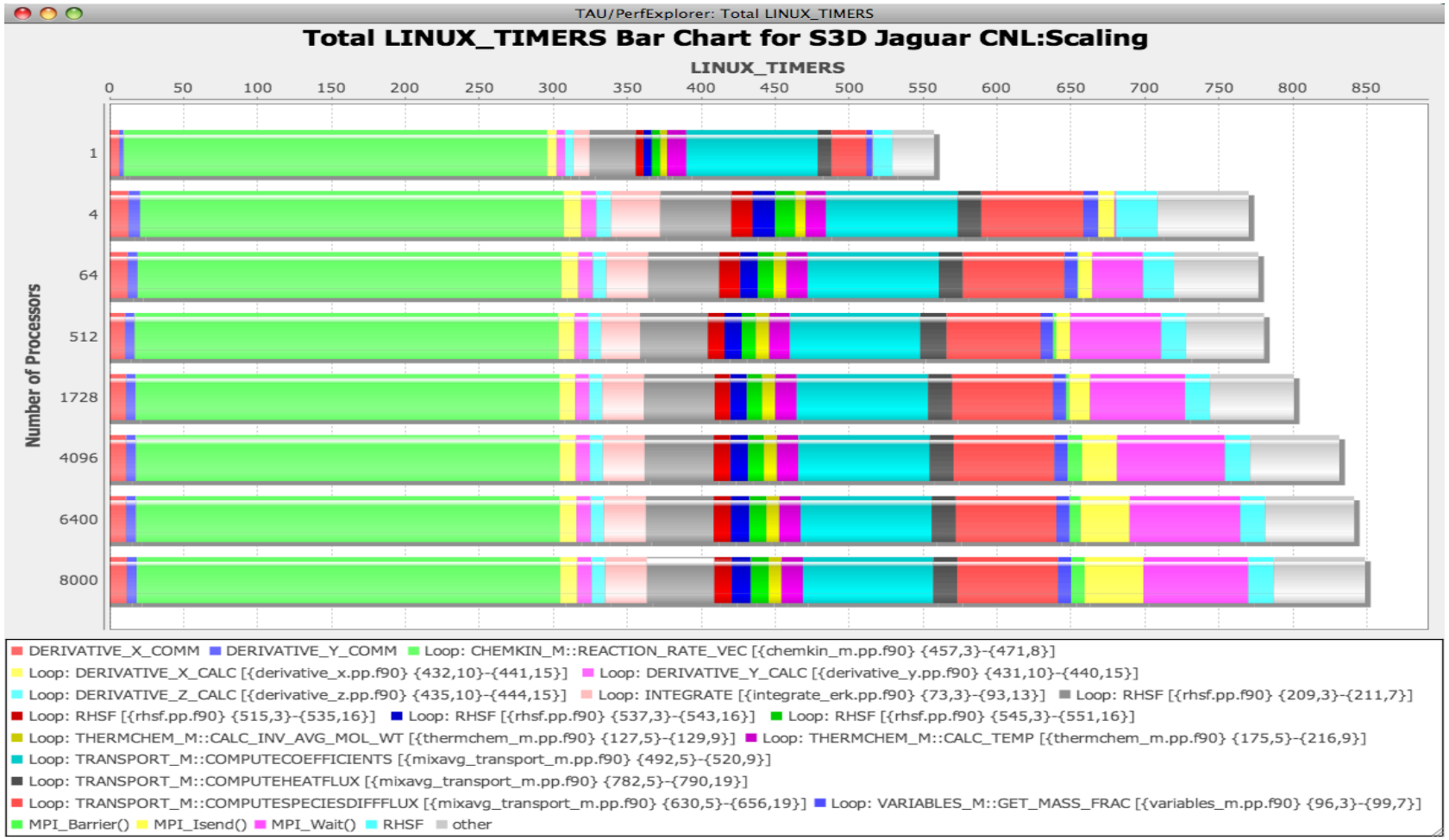
Evaluate Scalability

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

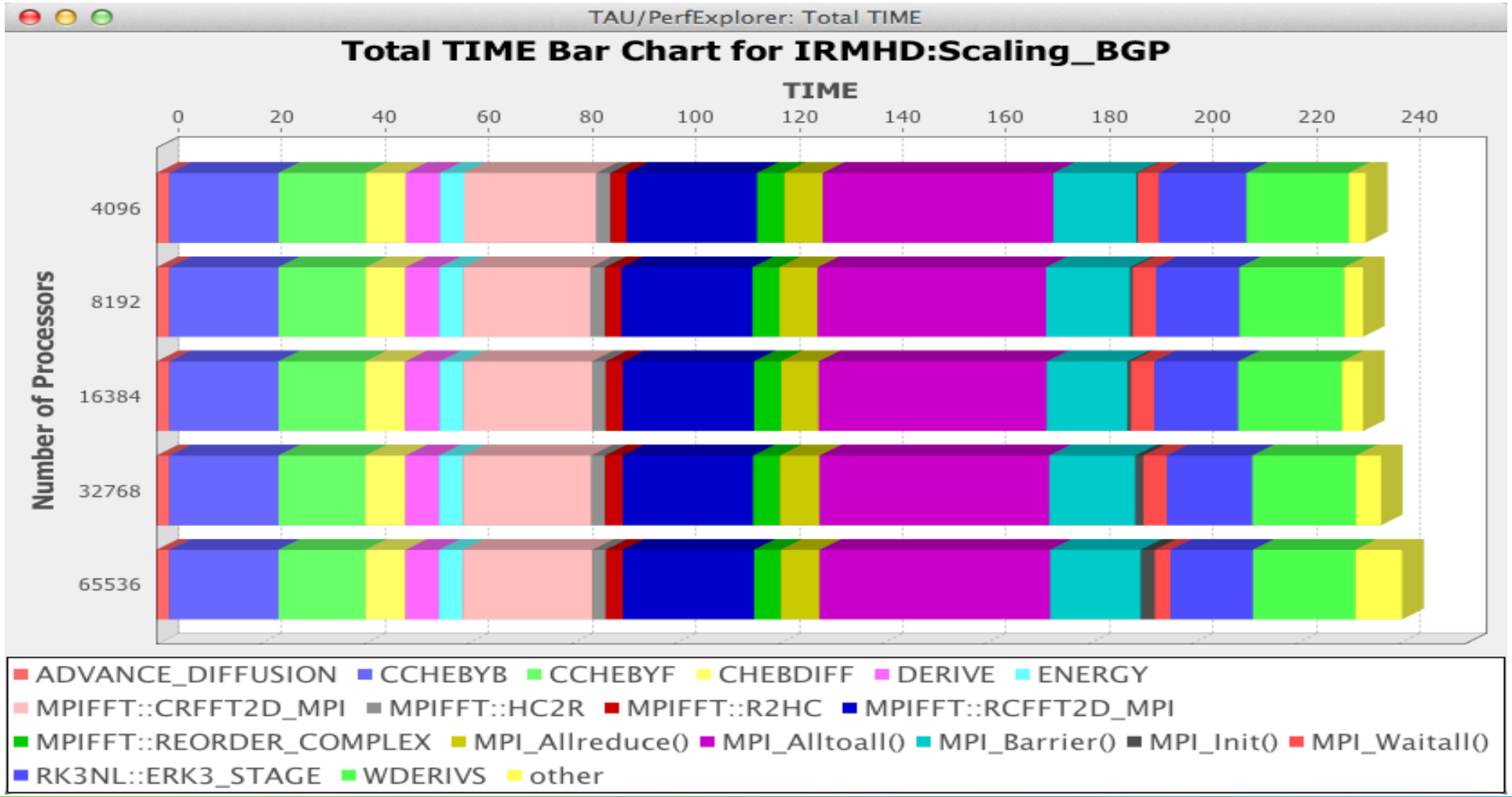




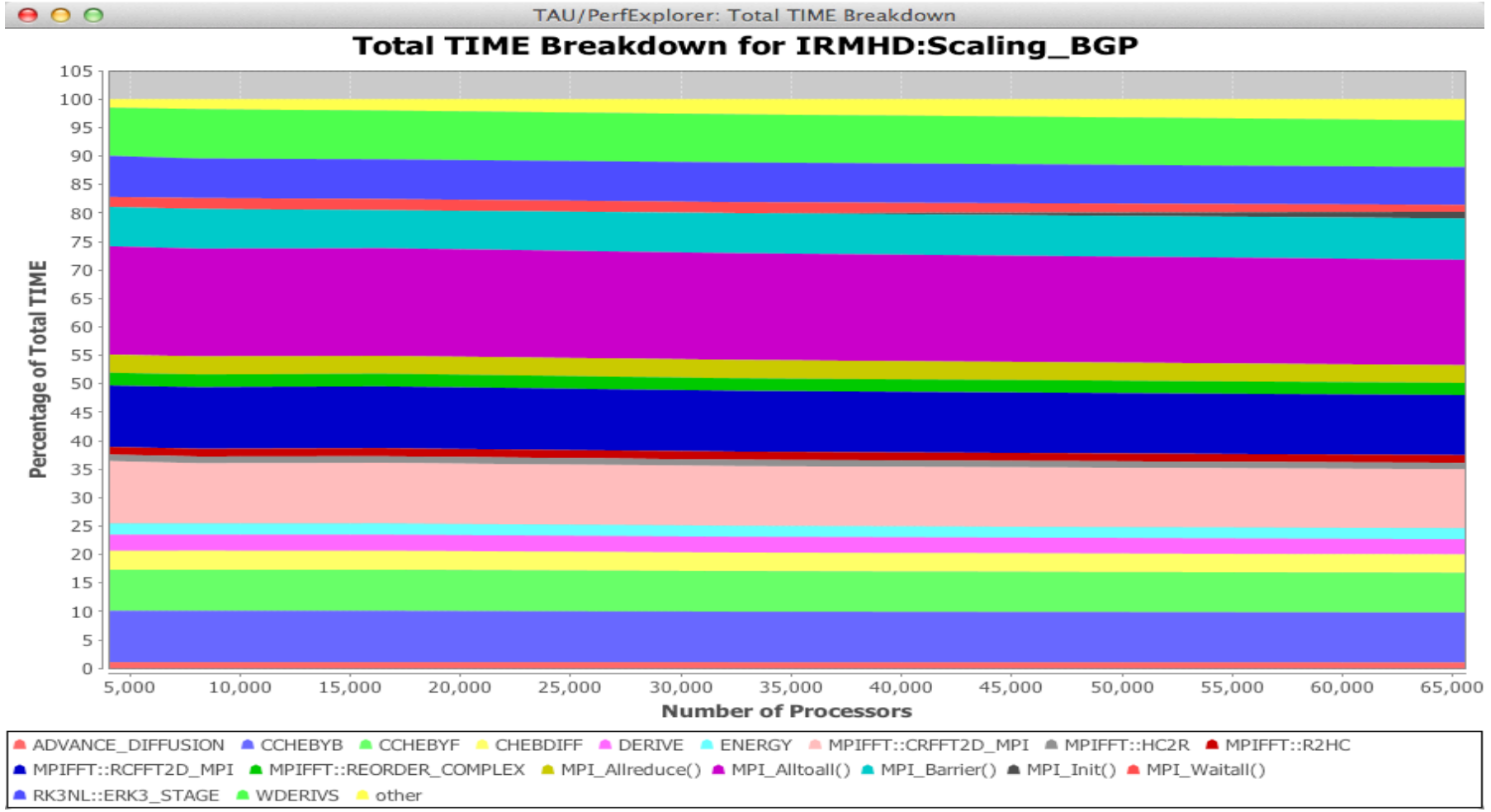
Evaluate Scalability



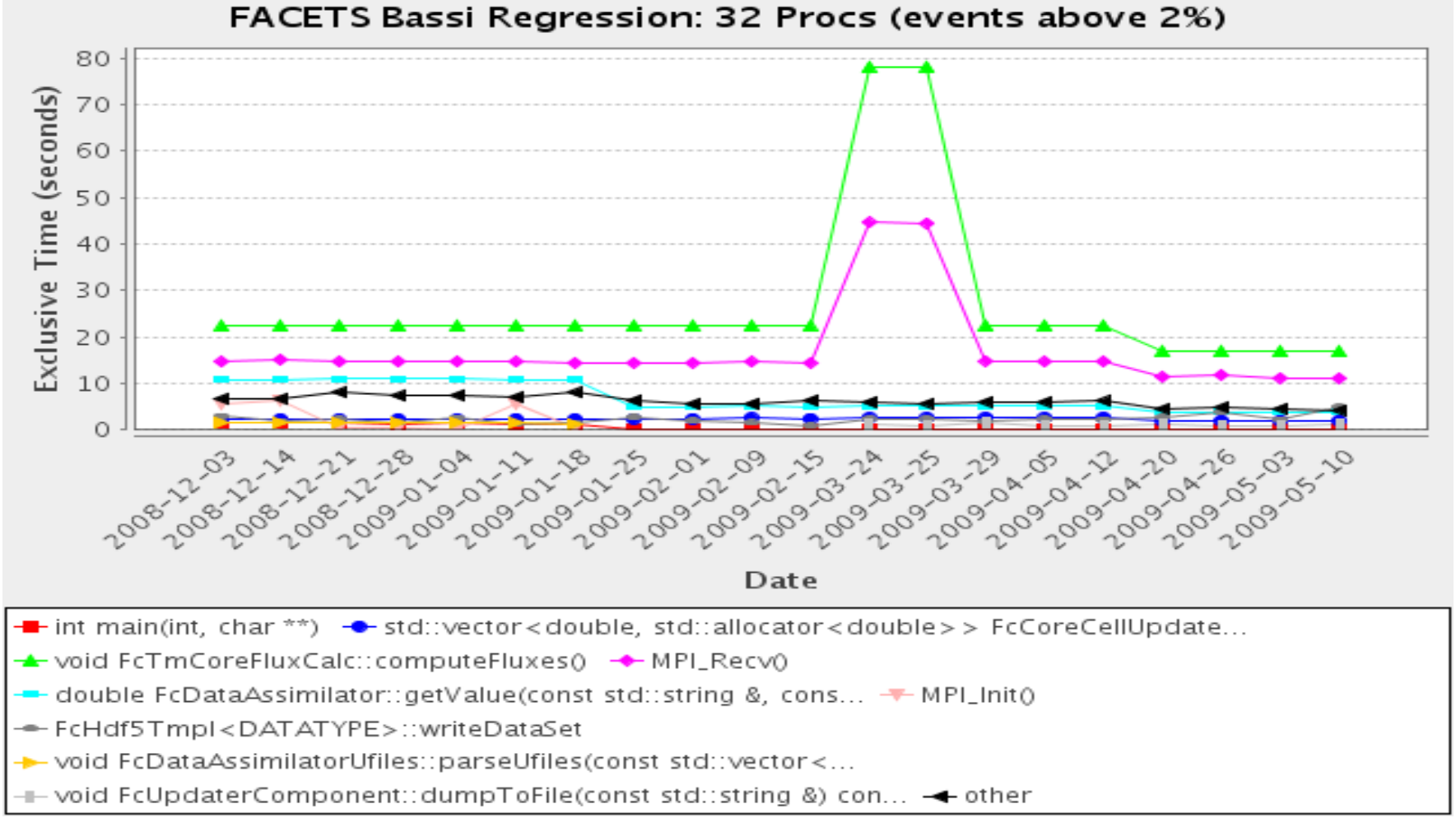
PerfExplorer



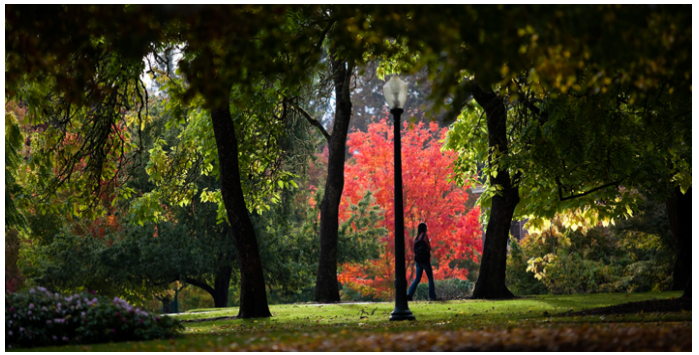
PerfExplorer



Performance Regression Testing



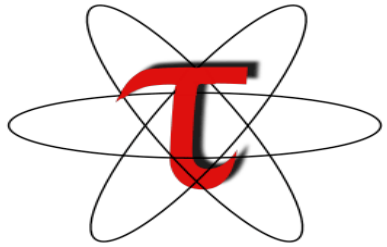
Performance Research Lab, University of Oregon, Eugene, USA



Support Acknowledgments

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

Download TAU from U. Oregon



http://tau.uoregon.edu

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

Free download, open source, BSD license