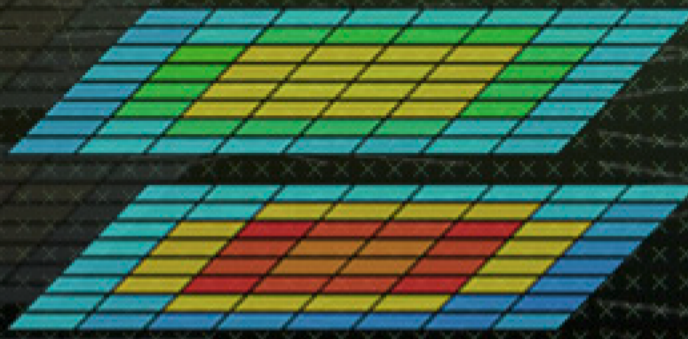
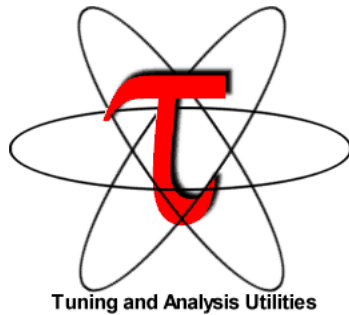




TAU Performance System®

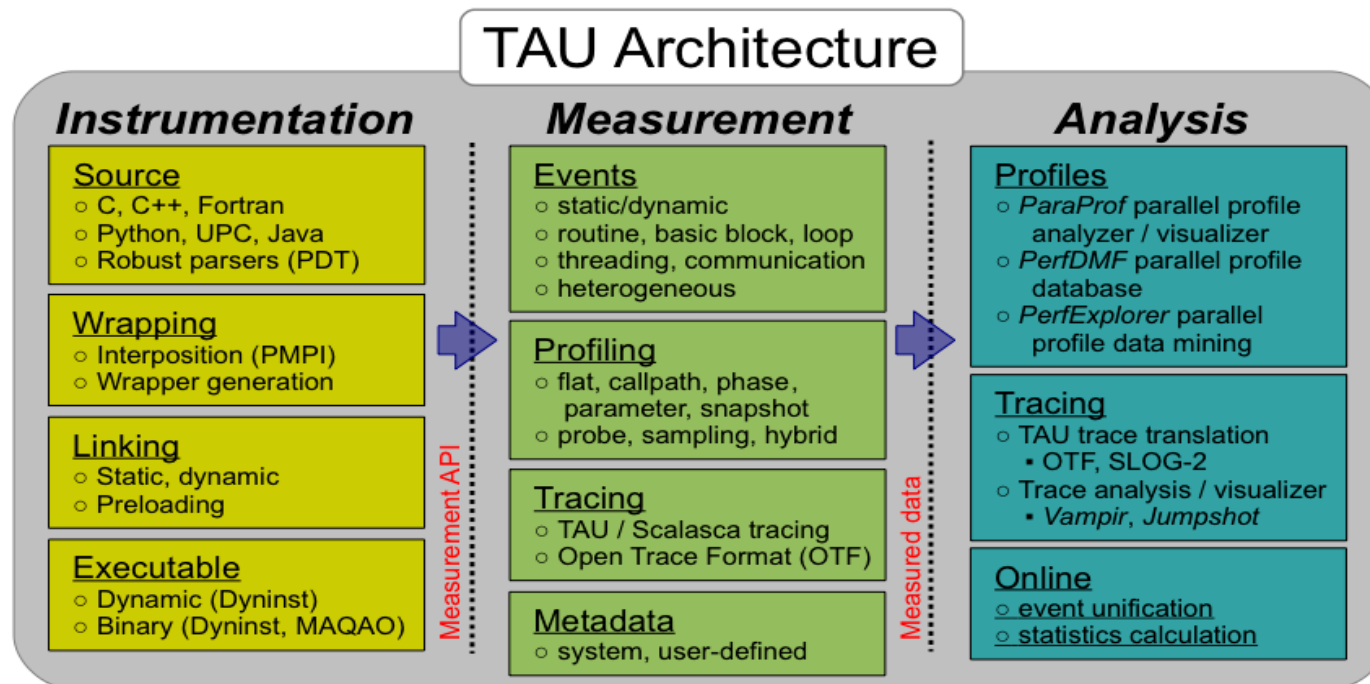


Sameer Shende
sameer@cs.uoregon.edu
 University of Oregon
<http://tau.uoregon.edu>



TAU Performance System®

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



TAU Performance System

- Instrumentation
 - Fortran, C++, C, UPC, Java, Python, Chapel
 - Automatic instrumentation
- Measurement and analysis support
 - MPI, OpenSHMEM, ARMCI, PGAS, DMAP
 - 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 profiles and trace generation using Score-P
- Analysis
 - Parallel profile analysis (ParaProf), data mining (PerfExplorer)
 - Performance database technology (TAUdb)
 - 3D profile browser

TAU Performance System

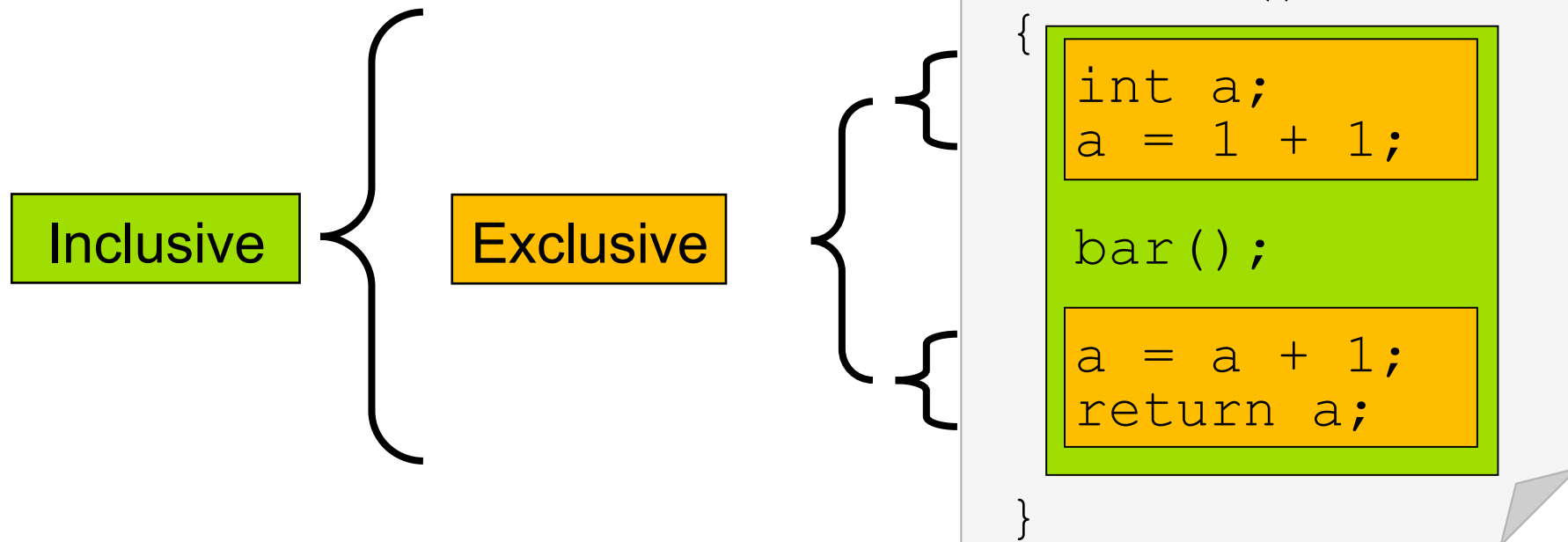
- TAU supports both sampling and direct instrumentation
- Memory debugging as well as I/O performance evaluation
- Profiling as well as tracing
- Interfaces with Score-P for more efficient measurements
- TAU's instrumentation covers:
 - Runtime library interposition (`tau_exec`)
 - Compiler-based instrumentation
 - 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 (OMPT, `tau_exec -T ompt`)
 - CUDA CUPTI, OpenCL (`tau_exec -T cupti -cupti`)

Application Performance Engineering using TAU

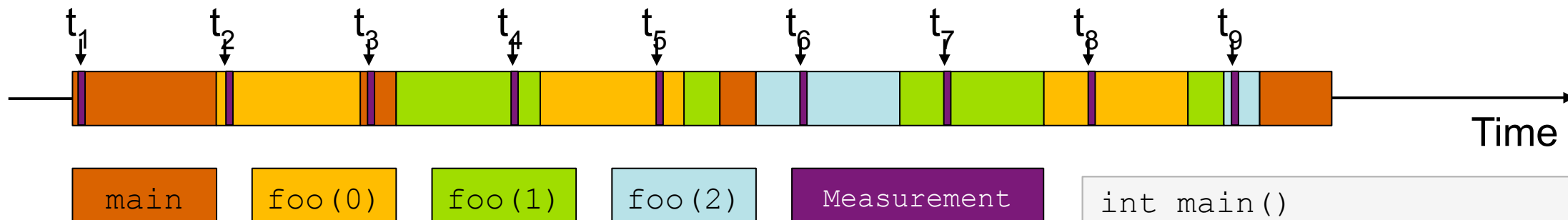
- How much time is spent in each application routine and outer *loops*? Within loops, what is the contribution of each *statement*? What is the time spent in OpenMP loops?
- How many instructions are executed in these code regions?
Floating point, Level 1 and 2 *data cache misses*, hits, branches taken?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

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

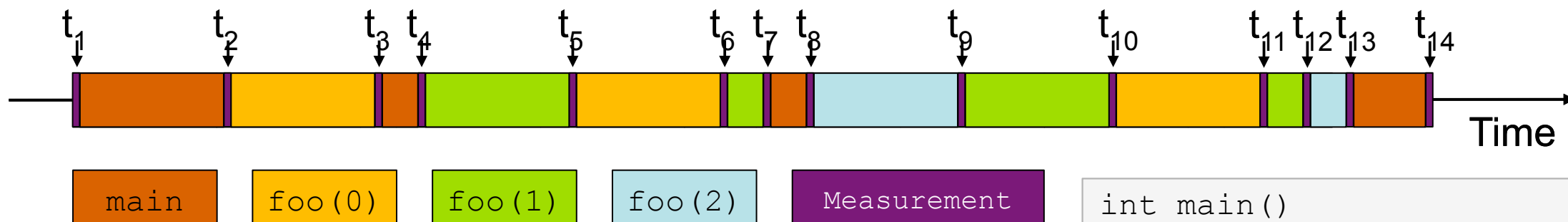
```
int main()
{
    int i;

    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 8 ./a.out`
 - `mpirun -np 8 tau_exec -T ompt,tr6,mpi -ompt ./a.out`

TAU hands-on exercises

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 (*romeo, CEA*)

- Setup preferred program environment compilers

```
% module load intel/2017.7 openmpi/20.4.1.1_icc_mt
% module load tau
# If you have previous performance data from Score-P, you may view it with TAU's paraprof
% paraprof profile.cubex &
```

For PerfExplorer:

```
% wget http://tau.uoregon.edu/data.tgz . ; tar xzf 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 = mpif77 # OpenMPI with Intel compiler
MPIF77 = mpif77
# Alternative variant to perform instrumentation
# MPIF77 = tau_f90.sh
# PREP is a generic preposition macro for instrumentation preparation
#MPIF77 = $(PREP) mpif77 -f77=ifort
#MPIF77 = scorep ...
...
```

Default (no instrumentation)

Uncomment MPIF77 and
comment Score-P's compiler
wrapper to generate
uninstrumented binary!

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=C 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 C
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.tau/bt-mz_C.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=**C**

Shortcut: `% make suite`

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 8 threads should be reasonable for 2 compute nodes of romeo
 - bt-mz_B.8 should run in less than 4 seconds
 - bt-mz_C.8 should run in around 15 seconds

NPB-MZ-MPI / BT with TAU (tau_exec -T ompt)

```
% cd bin
% cp ../jobscript/romeo/tau_ompt.slurm .
% sbatch./tau_ompt.slurm
% cat npb_btmz_ref.o<job_id>
NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP Benchmark
Number of zones:    16 x 16
Iterations:  200    dt:    0.000300
Number of active processes:      8
Total number of threads:        64 (  8.0 threads/process)

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.ppk
<Copy file over to desktop using scp>
% paraprof bt.ppk &
```

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

tau_exec launches un-instrumented binaries!

NPB-MZ-MPI / BT with TAU's Sampling (tau_exec -T ompt -ebs)

```
% cd bin
% cp ../jobscript/romeo/tau_ompt_ebs.slurm .
% sbatch ./tau_ompt_ebs.slurm
% cat npb_btmz_ref.o<job_id>
NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP Benchmark
Number of zones: 16 x 16
Iterations: 200 dt: 0.000300
Number of active processes: 8
Total number of threads: 64 ( 8.0 threads/process)

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 &
```

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

tau_exec launches un-instrumented binaries!

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

```
% cd bin
% cp ../jobscript/romeo/tau_otf.slurm .
% sbatch tau_otf.slurm
% cat npb_btmz_ref.o<job_id>
NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP Benchmark
Number of zones:    16 x 16
Iterations: 200    dt: 0.000300
Number of active processes:    8
Total number of threads:    64 ( 8.0 threads/process)

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 &
```

- Copy jobscript and launch as a hybrid MPI+OpenMP application
- TAU_TRACE=1
- TAU_TRACE_FORMAT=otf2

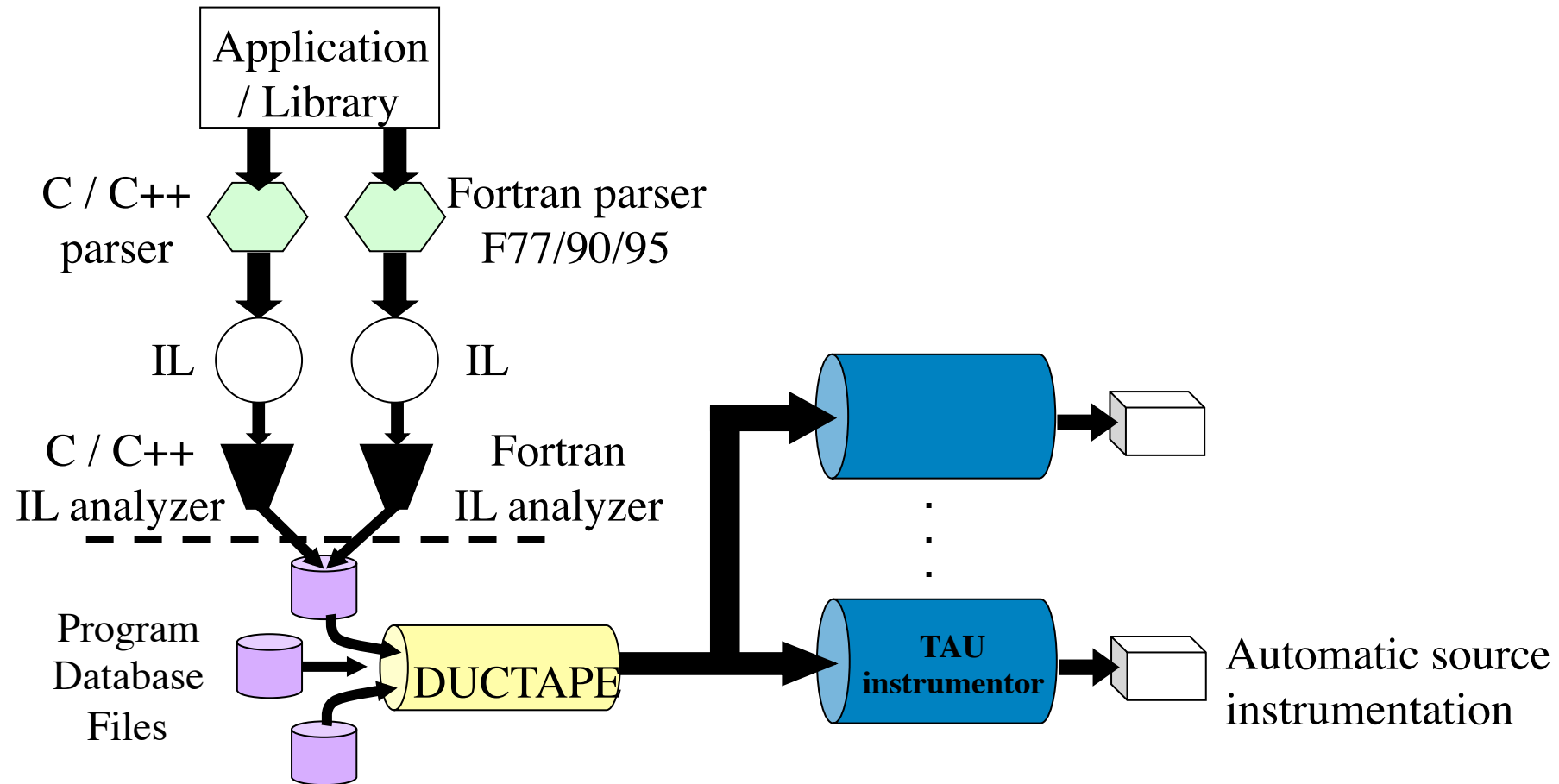
TAU generates native OTF2 traces for Vampir with OpenMP and MPI events for uninstrumented binaries!

TAU OMPT TR4 vs OMPT TR6 support

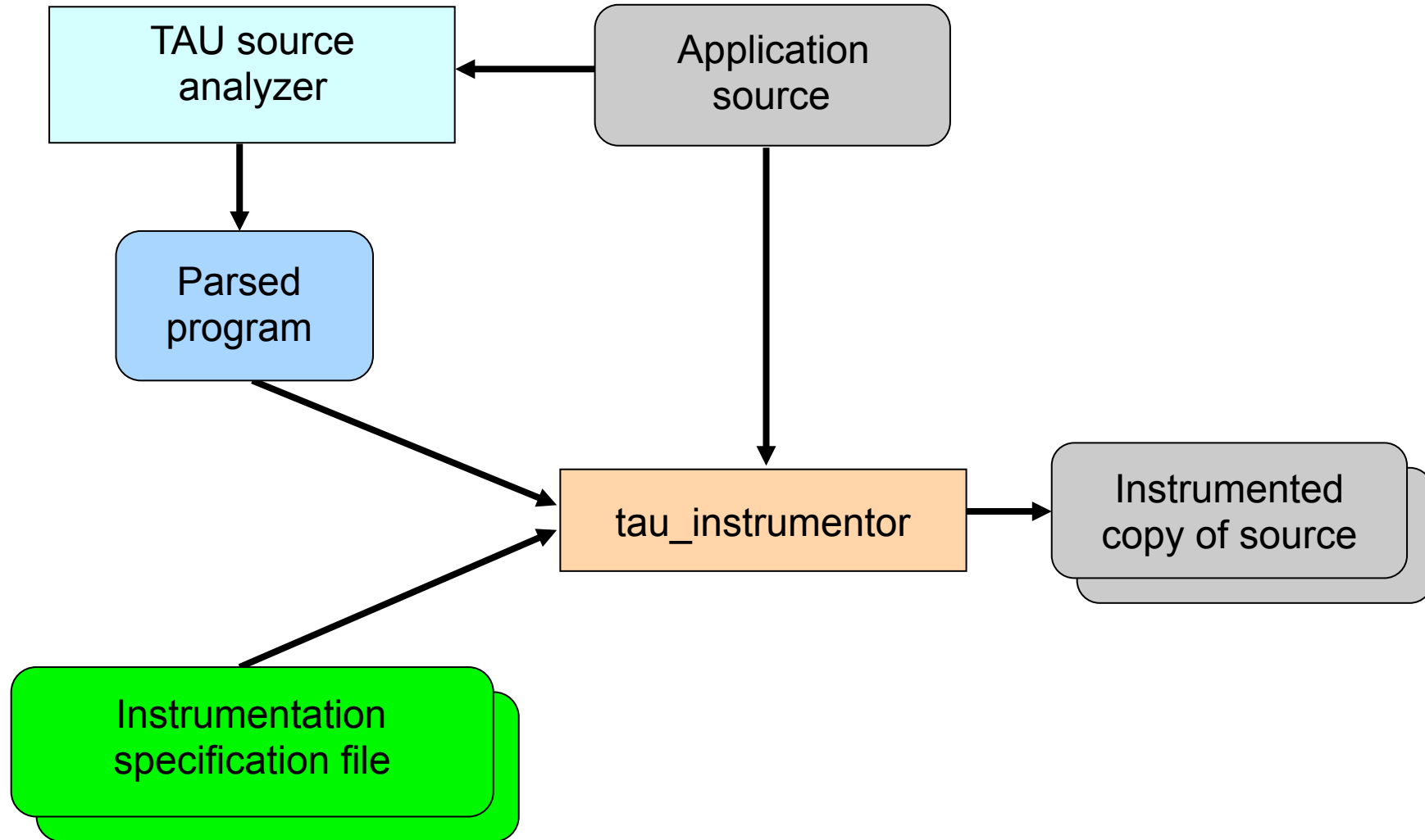
- Both standards (TR4 and TR6) allow us to *replace* an OpenMP library with one that will generate events.
- TR6 has a lower overhead, finer granularity of support
 - Generate OpenMP events on just thread 0, or threads 0.. N-1 (TAU_OMPT_SUPPORT_LEVEL=full)
 - export TAU_OMPT_SUPPORT_LEVEL=full
 - mpirun -np <N> tau_exec -T ompt, **tr6** -ompt ./a.out instead of
 - mpirun -np <N> tau_exec -T ompt -ompt ./a.out
- TR6: Can use eager address resolution or defer address resolution to postmortem
 - export TAU_OMPT_RESOLVE_ADDRESS_EAGERLY=1
 - cp /home/projects/tauscripts/tau_ompt_tr6.slurm . ; sbatch ./tau_ompt_tr6.slurm
- Can be used with Event Based Sampling (EBS)
 - cp /home/projects/tauscripts/tau_ompt_tr6_ebs.slurm . ; sbatch ./tau_ompt_tr6_ebs.slurm

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 -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:

```
% module load tau
```

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

```
% module load tau
% ls $TAU/Makefile.*
/apps/tau/2.27.2p1/x86_64/lib/Makefile.tau-icpc
/apps/tau/2.27.2p1/x86_64/lib/Makefile.tau-icpc-mpi-pdt
/apps/tau/2.27.2p1/x86_64/lib/Makefile.tau-icpc-papi-mpi-pdt-openmp-opari
/apps/tau/2.27.2p1/x86_64/lib/Makefile.tau-icpc-papi-mpi-pthread-pdt
/apps/tau/2.27.2p1/x86_64/lib/Makefile.tau-icpc-papi-ompt-mpi-pdt-openmp
/apps/tau/2.27.2p1/x86_64/lib/Makefile.tau-mpi-pdt
```

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

Makefile.tau-icpc-papi-ompt-mpi-pdt-openmp

- Supports MPI instrumentation & PDT for automatic source instrumentation

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

```
% tau_f90.sh matmult.f90 -o matmult
% mpirun -np 8 ./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
```

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 = mpif90 # OpenMPI with Intel compiler

# Alternative variant to perform instrumentation
MPIF77 = tau_f90.sh

# 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 with Source Instrumentation

```
% module load tau;
% export TAU_MAKEFILE=$TAU/Makefile.tau-icpc-papi-ompt-tr6-mpi-pdt-openmp
% make suite ; cd bin.tau; cp ../jobscript/reference.sbatch . ; \
sbatch reference.sbatch; paraprof
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 C
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.tau/bt-mz_C.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=**C**

Shortcut: `% make suite`

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
-optComplnst	Use compiler based instrumentation
-optNoComplnst	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 <i>-iowrapper</i>)
-optTrackGOMP	Enable tracking GNU OpenMP runtime layer (used without <i>-opari</i>)
-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 <i>tau_instrumentor</i>
-optTauWrapFile=" <file> "	Specify path to <i>link_options.tau</i> generated by <i>tau_gen_wrapper</i>
-optHeaderInst	Enable Instrumentation of headers
-optTrackUPCR	Track UPC runtime layer routines (used with tau_upc.sh)
-optLinking=""	Options passed to the linker. Typically <code>\$(TAU_MPI_FLIBS) \$(TAU_LIBS) \$(TAU_CXXLIBS)</code>
-optCompile=""	Options passed to the compiler. Typically <code>\$(TAU_MPI_INCLUDE) \$(TAU_INCLUDE) \$(TAU_DEFS)</code>
-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 gfparsers
-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= '-optComplnst -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

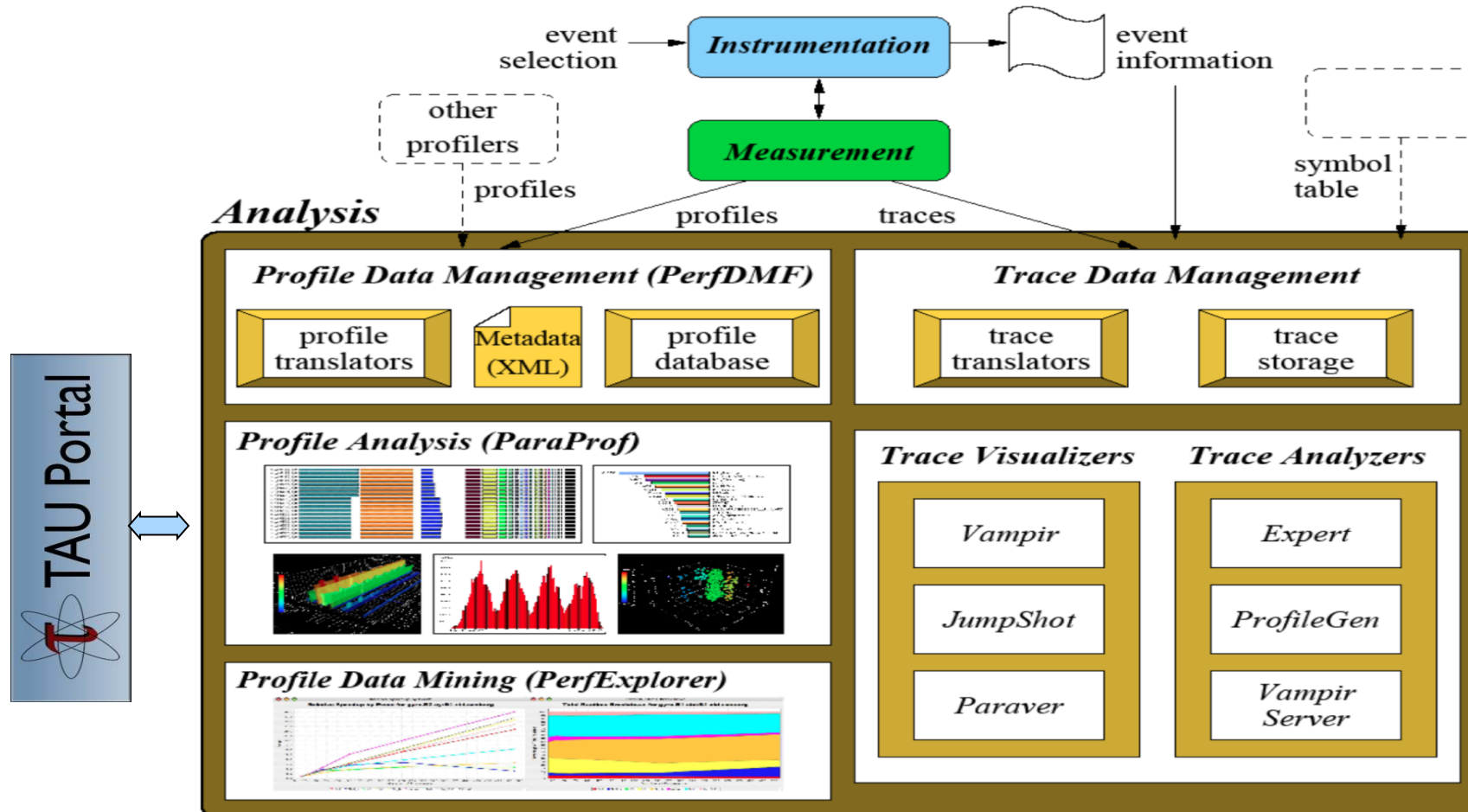
Environment Variable	Default	Description
TAU_TRACE	0	Setting to 1 turns on tracing
TAU_TRACE_FORMAT	default	Setting to “otf2” generates OTF2 traces natively.
TAU_CALLPATH	0	Setting to 1 turns on callpath profiling
TAU_CALLSITE	0	Setting to 1 generates callsite information in events. May be used with 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.)

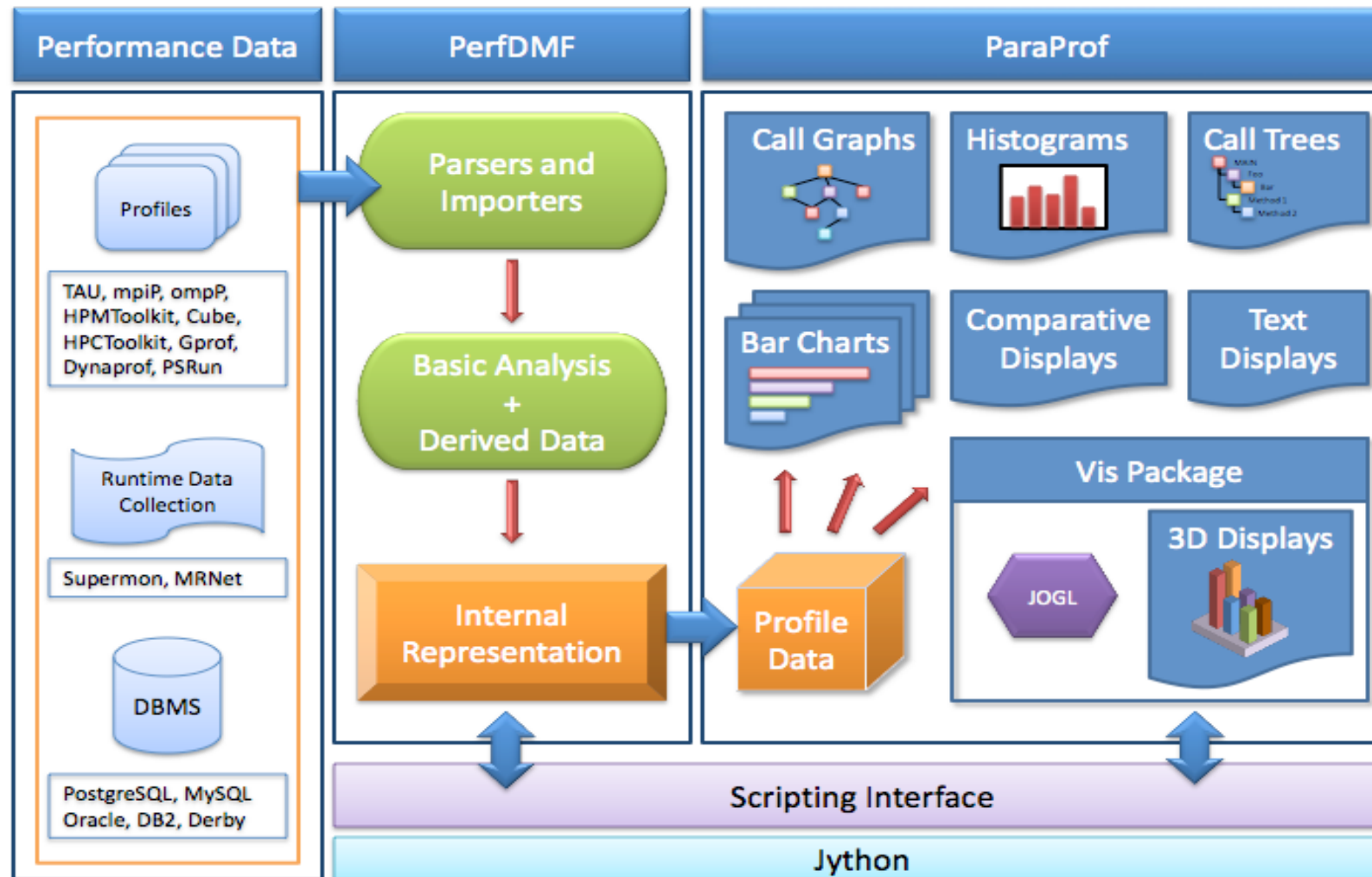
Environment Variable	Default	Description
TAU_TRACK_MEMORY_LEAKS	0	Tracks allocates that were not de-allocated (needs <code>-optMemDbg</code> or <code>tau_exec -memory</code>)
TAU_EBS_SOURCE	TIME	Allows using PAPI hardware counters for periodic interrupts for EBS (e.g., <code>TAU_EBS_SOURCE=PAPI_TOT_INS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

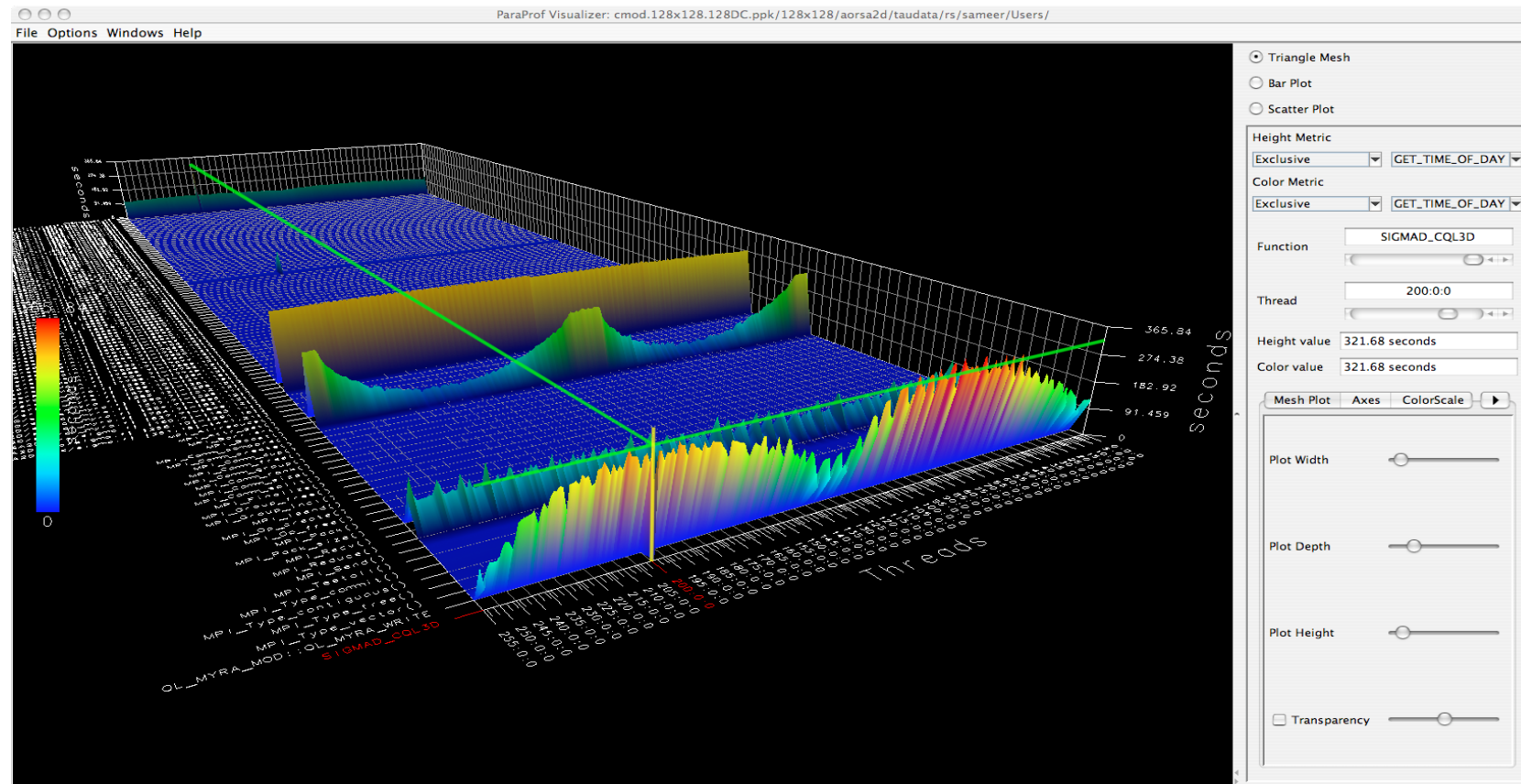
TAU Analysis



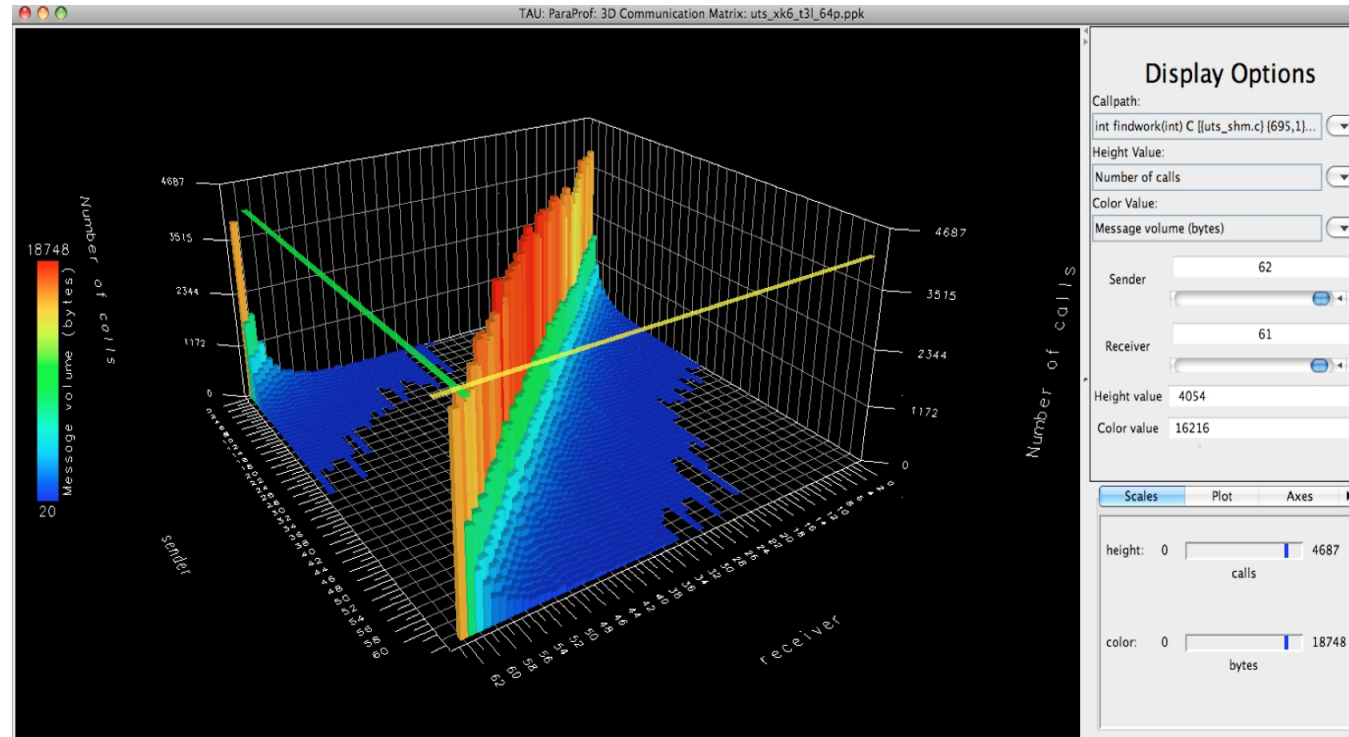
ParaProf Profile Analysis Framework



Parallel Profile Visualization: ParaProf



ParaProf 3D Communication Matrix



% export TAU_COMM_MATRIX=1

TAU Source Instrumentation

- Edit [config/make.def](#) to adjust build configuration
 - Uncomment specification of compiler/linker: [MPIF77 = tau_f90.sh](#)
- Make clean and build new tool-specific executable

```
% make clean  
% make bt-mz CLASS=C NPROCS=8  
Built executable ../bin.tau/bt-mz_C.8
```

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

```
% cd bin.tau  
% cp ../jobscript/romeo/run.msub .  
% sbatch reference.slurm
```

NPB-MZ-MPI / BT with TAU

```
% cd bin.tau
% cp ../jobscript/romeo/run.msub .
% sbatch reference.slurm
% cat npb_btmz_<job_id>.oe
NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP Benchmark
Number of zones:    16 x 16
Iterations:  200    dt:    0.000300
Number of active processes:      8
Total number of threads:        64 (  8.0 threads/process)

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

BT-MZ Benchmark Completed.
Time in seconds = 22.34
% paraprof &
% paraprof --pack bt.ppk
<Copy file over to desktop using scp>
% paraprof bt.ppk &
```

- 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

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
```

Metric

TAU: ParaProf Manager

Applications

- Standard Applications
 - Default App
 - Default Exp
 - bt_ompt.ppk
 - TIME

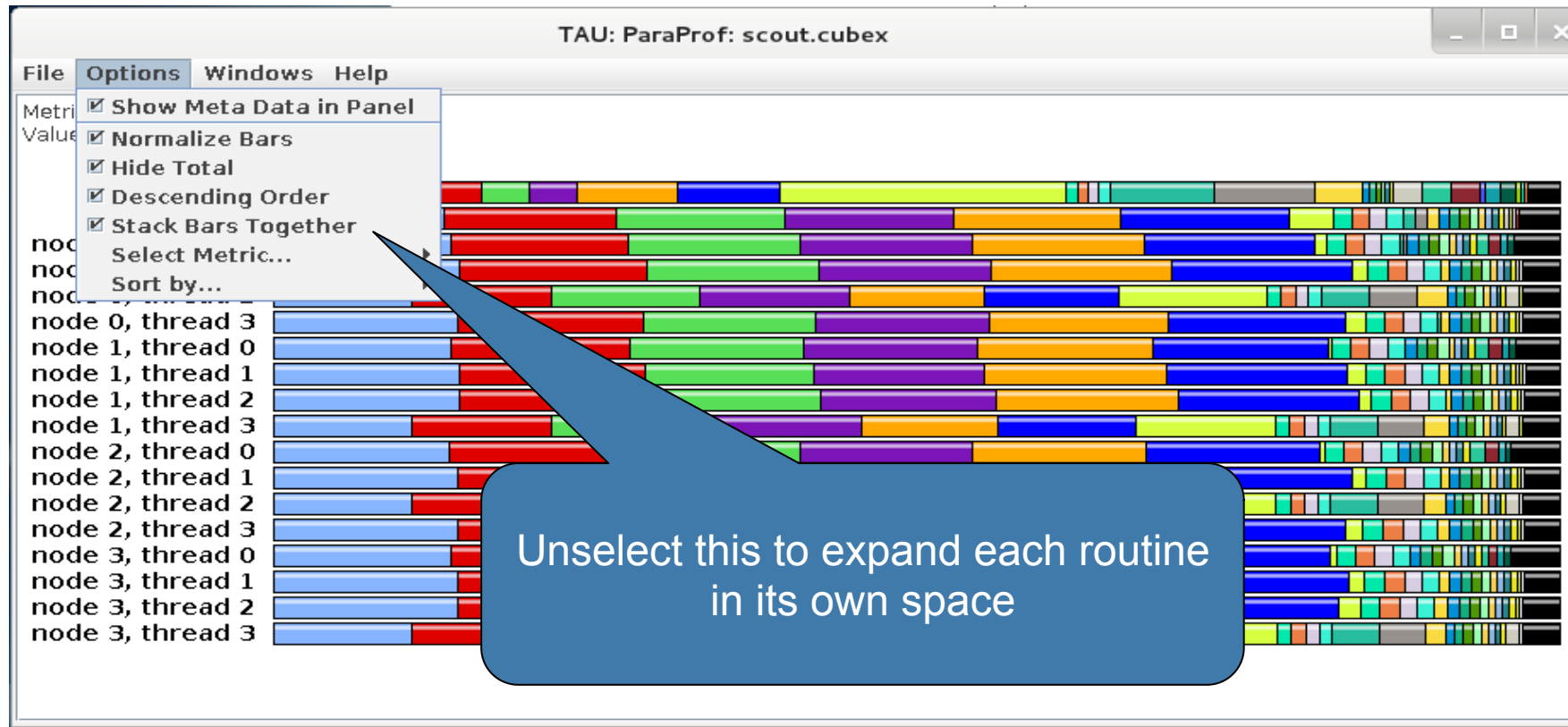
Default (jdbc:h2:/Users/sameer/.ParaProf/perfdmf/perfdmf;AUTO_SERVER=TRUE)

TrialField	Value
Name	bt_ompt.ppk
Application ID	0
Experiment ID	0
Trial ID	0
CPU Cores	8
CPU MHz	2600.000
CPU Type	Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz
CPU Vendor	GenuineIntel
CWD	/scratch/sameer/NPB3.3-MZ-MPI/bin
Cache Size	20480 KB
Command Line	./bt-mz_C.8
Executable	/scratch/sameer/NPB3.3-MZ-MPI/bin/bt-mz_C.8
File Type Index	0
File Type Name	ParaProf Packed Profile
Hostname	frog9
Local Time	2015-05-18T00:37:38+02:00
MPI Processor Name	frog9
Memory Size	65944056 kB
Node Name	frog9
OMP_CHUNK_SIZE	1
OMP_DYNAMIC	off
OMP_MAX_THREADS	4
OMP_NESTED	off
OMP_NUM_PROCS	4
OMP_SCHEDULE	UNKNOWN
OS Machine	x86_64
OS Name	Linux
OS Release	2.6.32-279.5.2.el6.Bull.33.x86_64
OS Version	#1 SMP Sat Nov 10 01:48:00 CET 2012

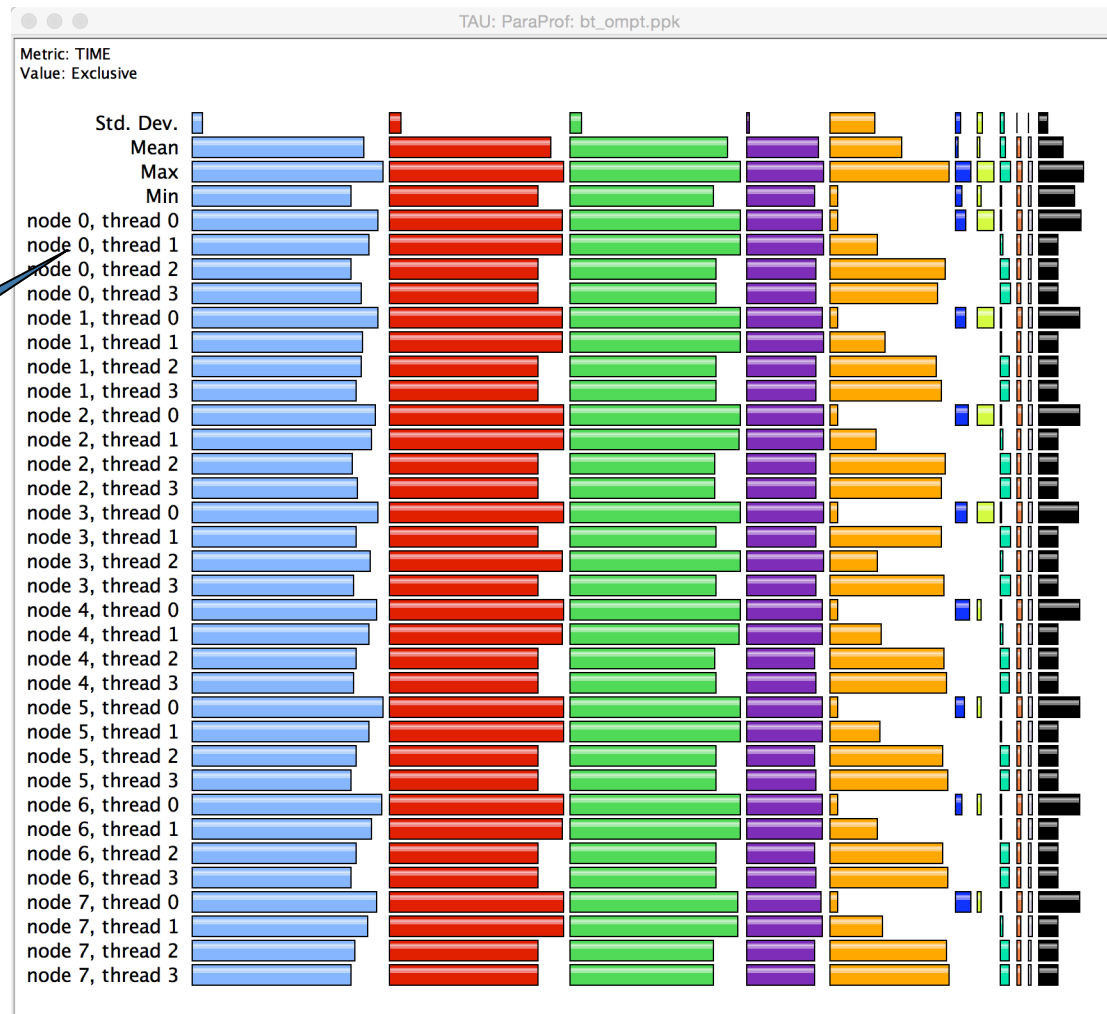
Paraprof main window



Paraprof main window

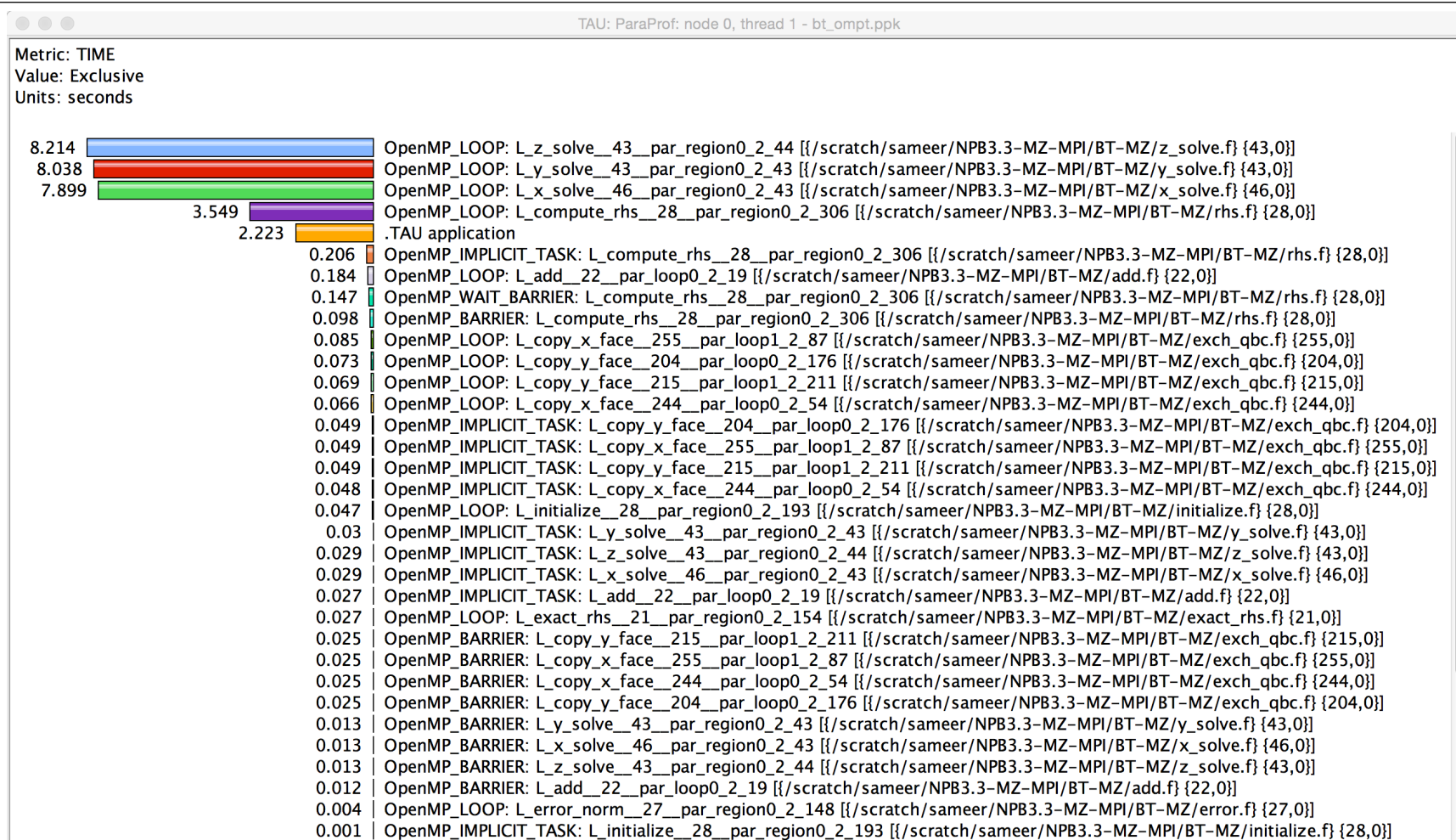


Paraprof main window



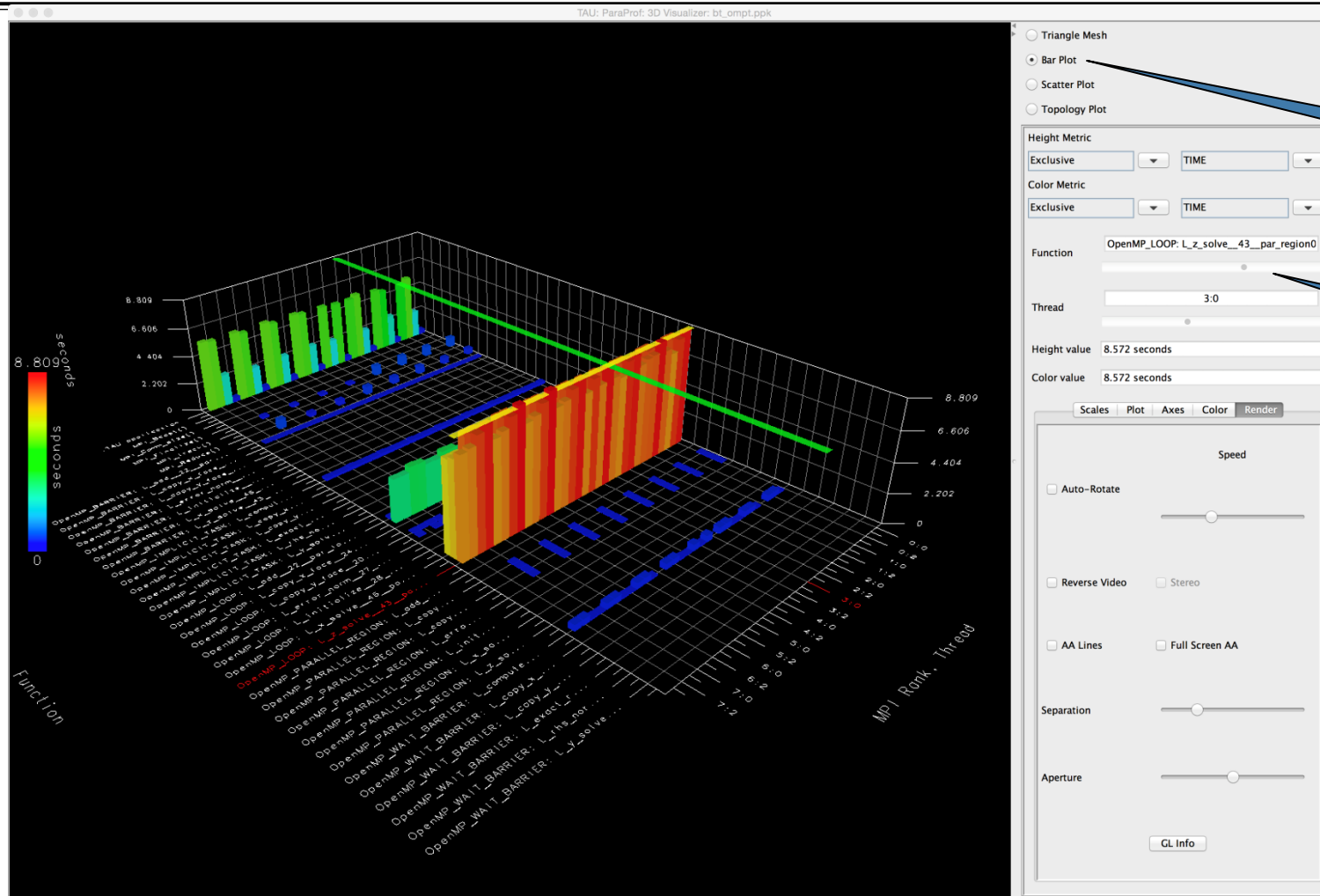
Each routine occupies its own space.
Can see the extent of imbalance
across all threads.

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

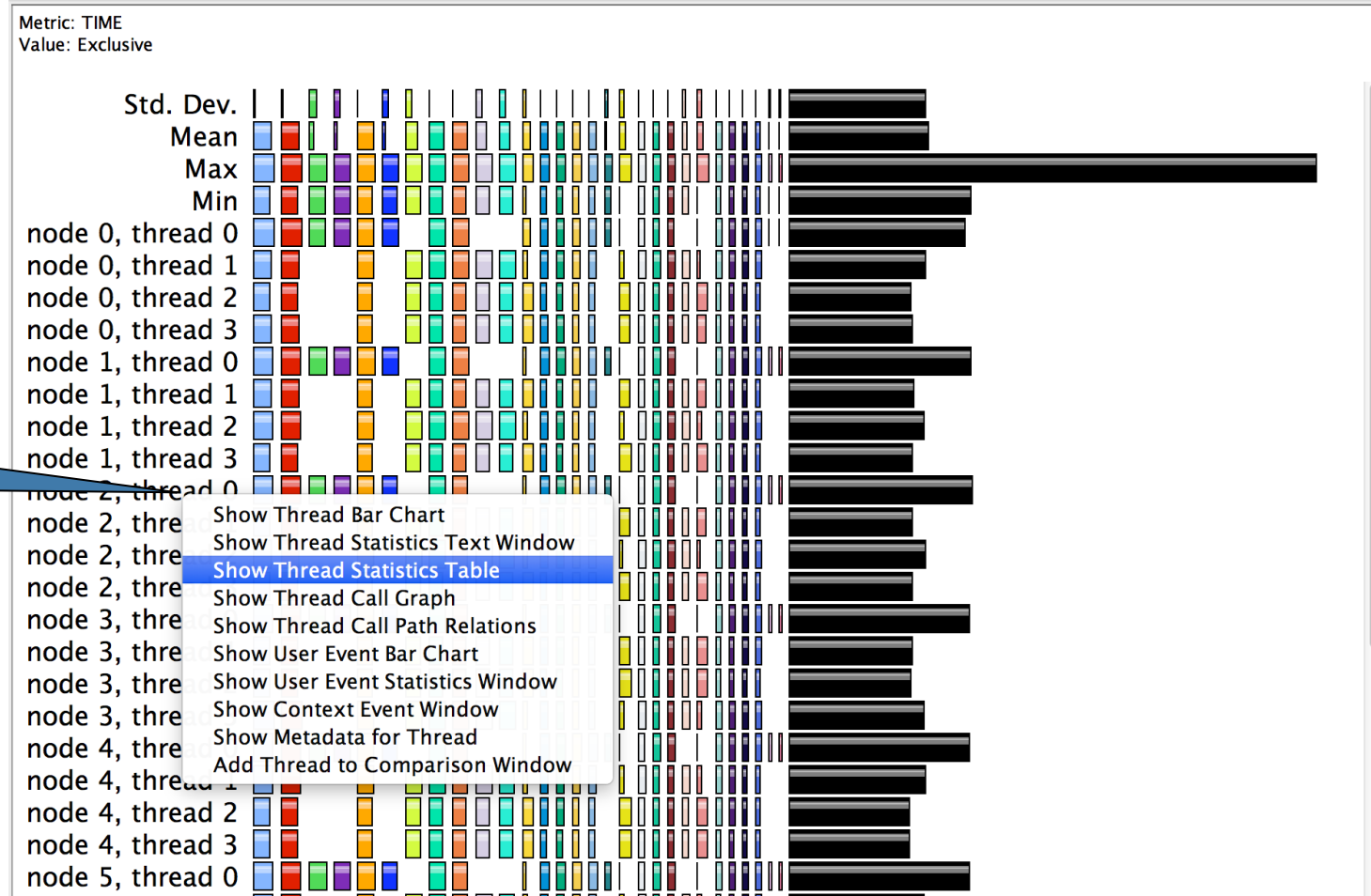


Click Bar Plot

Move Function
and Thread Sliders

Windows ->
3D visualization

Paraprof Thread Statistics Table with TAU_SAMPLING=1



Statement Level Profiling with TAU

```
File Help
TAU: ParaProf: Source Browser: /scratch/sameer/NPB3.3-MZ-MPI/BT-MZ/x_solve.f

353       call matmul_sub(lhs(1,1,aa,i),
354       v         lhs(1,1,cc,i-1),
355       v         lhs(1,1,bb,i))
356
357
358 C-----
359 c   multiply c(i,j,k) by b_inverse and copy back to c
360 c   multiply rhs(1,j,k) by b_inverse(1,j,k) and copy to rhs
361 C-----
362       call binvrhs( lhs(1,1,bb,i),
363       v         lhs(1,1,cc,i),
364       v         rhs(1,i,j,k) )
365
366       enddo
367
368 C-----
369 c   rhs(isize) = rhs(isize) - A*rhs(isize-1)
370 C-----
371       call matvec_sub(lhs(1,1,aa, isize),
372       v         rhs(1, isize-1,j,k), rhs(1, isize,j,k))
373
374 C-----
375 c   B(isize) = B(isize) - C(isize-1)*A(isize)
376 C-----
377       call matmul_sub(lhs(1,1,aa, isize),
378       v         lhs(1,1,cc, isize-1),
379       v         lhs(1,1,bb, isize))
380
381 C-----
382 c   multiply rhs() by b_inverse() and copy to rhs
383 C-----
384       call binvrhs( lhs(1,1,bb, isize),
385       v         rhs(1, isize,j,k) )
386
387 C-----
388 c   back solve: if last cell, then generate U(isize)=rhs(isize)
389 c   else assume U(isize) is loaded in un pack backsub_info
390 c   so just use it
391 c   after call u(istart) will be sent to next cell
392 C-----
393
394       do i=isize-1,0,-1
395       do m=1,BLOCK_SIZE
396       do n=1,BLOCK_SIZE
397       rhs(m,i,j,k) = rhs(m,i,j,k)
398       v         - lhs(m,n,cc,i)*rhs(n,i+1,j,k)
399       enddo
400       enddo
401       enddo
402
```

Source location where samples are taken. Compute intensive region.

Paraprof Thread Statistics Table

TAU: ParaProf: Statistics for: node 2, thread 0 - bt_ebs.ppk

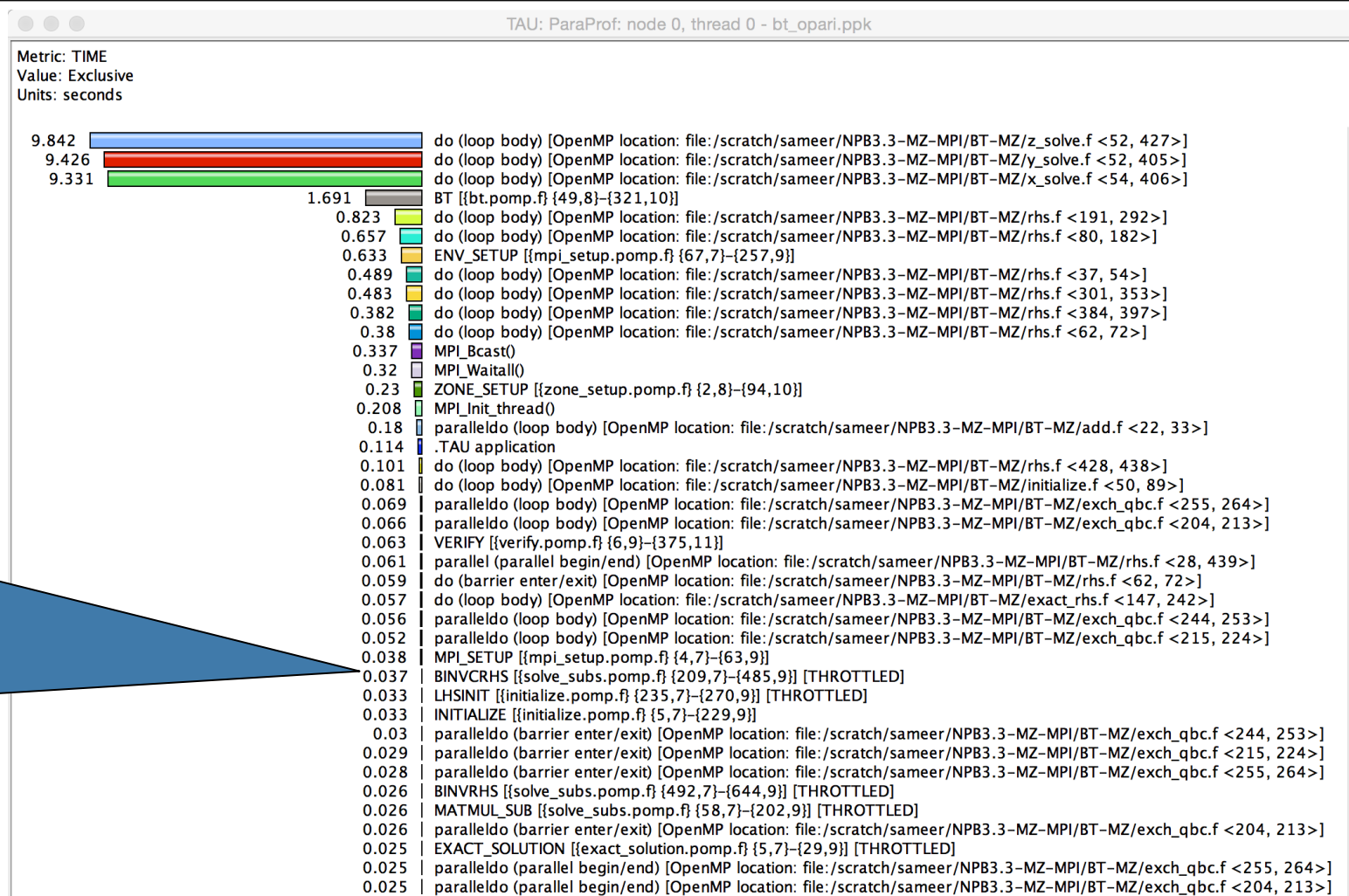
Name	Exclusive TIME	Inclusive TIME ▾	Calls	Child Calls
■ .TAU application	1.754	36.26	1	88,049
▼ ■ OpenMP_PARALLEL_REGION: L_z_solve_43_par_region0_2_44 [/{scratch/sameer/NPB3.3-MZ-MPI/BT-MZ/z_solve.f} {43,0}]	0.061	8.692	6,432	12,864
▼ ■ OpenMP_IMPLICIT_TASK: L_z_solve_43_par_region0_2_44 [/{scratch/sameer/NPB3.3-MZ-MPI/BT-MZ/z_solve.f} {43,0}]	0.04	8.568	6,432	6,432
▼ ■ OpenMP_LOOP: L_z_solve_43_par_region0_2_44 [/{scratch/sameer/NPB3.3-MZ-MPI/BT-MZ/z_solve.f} {43,0}]	8.528	8.528	6,432	0
▼ ■ [CONTEXT] OpenMP_LOOP: L_z_solve_43_par_region0_2_44 [/{scratch/sameer/NPB3.3-MZ-MPI/BT-MZ/z_solve.f} {43,0}]	0	9.23	847	0
▼ ■ [SUMMARY] L_z_solve_43_par_region0_2_44 [/{scratch/sameer/NPB3.3-MZ-MPI/BT-MZ/z_solve.f}]	3.67	3.67	340	0
▼ ■ [SAMPLE] L_z_solve_43_par_region0_2_44 [/{scratch/sameer/NPB3.3-MZ-MPI/BT-MZ/z_solve.f}]	3.67	3.67	340	0
■ [SAMPLE] L_z_solve_43_par_region0_2_44 [/{scratch/sameer/NPB3.3-MZ-MPI/BT-MZ/z_solve.f} {419}]	0.22	0.22	21	0
■ [SAMPLE] L_z_solve_43_par_region0_2_44 [/{scratch/sameer/NPB3.3-MZ-MPI/BT-MZ/z_solve.f} {58}]	0.17	0.17	16	0
■ [SAMPLE] L_z_solve_43_par_region0_2_44 [/{scratch/sameer/NPB3.3-MZ-MPI/BT-MZ/z_solve.f} {418}]	0.16	0.16	12	0
■ [SAMPLE] L_z_solve_43_par_region0_2_44 [/{scratch/sameer/NPB3.3-MZ-MPI/BT-MZ/z_solve.f} {123}]	0.11	0.11	11	0
■ [SAMPLE] L_z_solve_43_par_region0_2_44 [/{scratch/sameer/NPB3.3-MZ-MPI/BT-MZ/z_solve.f} {193}]	0.08	0.08	5	0
■ [SAMPLE] L_z_solve_43_par_region0_2_44 [/{scratch/sameer/NPB3.3-MZ-MPI/BT-MZ/z_solve.f} {126}]	0.07	0.07	7	0
■ [SAMPLE] L_z_solve_43_par_region0_2_44 [/{scratch/sameer/NPB3.3-MZ-MPI/BT-MZ/z_solve.f} {247}]	0.07	0.07	6	0
■ [SAMPLE] L_z_solve_43_par_region0_2_44 [/{scratch/sameer/NPB3.3-MZ-MPI/BT-MZ/z_solve.f} {158}]	0.06	0.06	5	0
■ [SAMPLE] L_z_solve_43_par_region0_2_44 [/{scratch/sameer/NPB3.3-MZ-MPI/BT-MZ/z_solve.f} {313}]	0.06	0.06	4	0
■ [SAMPLE] L_z_solve_43_par_region0_2_44 [/{scratch/sameer/NPB3.3-MZ-MPI/BT-MZ/z_solve.f} {230}]	0.06	0.06	4	0
■ [SAMPLE] L_z_solve_43_par_region0_2_44 [/{scratch/sameer/NPB3.3-MZ-MPI/BT-MZ/z_solve.f} {308}]	0.05	0.05	3	0
■ [SAMPLE] L_z_solve_43_par_region0_2_44 [/{scratch/sameer/NPB3.3-MZ-MPI/BT-MZ/z_solve.f} {191}]	0.05	0.05	3	0
■ [SAMPLE] L_z_solve_43_par_region0_2_44 [/{scratch/sameer/NPB3.3-MZ-MPI/BT-MZ/z_solve.f} {81}]	0.05	0.05	4	0
■ [SAMPLE] L_z_solve_43_par_region0_2_44 [/{scratch/sameer/NPB3.3-MZ-MPI/BT-MZ/z_solve.f} {301}]	0.05	0.05	5	0
■ [SAMPLE] L_z_solve_43_par_region0_2_44 [/{scratch/sameer/NPB3.3-MZ-MPI/BT-MZ/z_solve.f} {67}]	0.05	0.05	5	0
■ [SAMPLE] L_z_solve_43_par_region0_2_44 [/{scratch/sameer/NPB3.3-MZ-MPI/BT-MZ/z_solve.f} {175}]	0.04	0.04	4	0
■ [SAMPLE] L_z_solve_43_par_region0_2_44 [/{scratch/sameer/NPB3.3-MZ-MPI/BT-MZ/z_solve.f} {89}]	0.04	0.04	4	0
■ [SAMPLE] L_z_solve_43_par_region0_2_44 [/{scratch/sameer/NPB3.3-MZ-MPI/BT-MZ/z_solve.f} {55}]	0.04	0.04	4	0
■ [SAMPLE] L_z_solve_43_par_region0_2_44 [/{scratch/sameer/NPB3.3-MZ-MPI/BT-MZ/z_solve.f} {275}]	0.04	0.04	4	0
■ [SAMPLE] L_z_solve_43_par_region0_2_44 [/{scratch/sameer/NPB3.3-MZ-MPI/BT-MZ/z_solve.f} {129}]	0.04	0.04	4	0
■ [SAMPLE] L_z_solve_43_par_region0_2_44 [/{scratch/sameer/NPB3.3-MZ-MPI/BT-MZ/z_solve.f} {168}]	0.04	0.04	4	0
■ [SAMPLE] L_z_solve_43_par_region0_2_44 [/{scratch/sameer/NPB3.3-MZ-MPI/BT-MZ/z_solve.f} {238}]	0.04	0.04	4	0

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

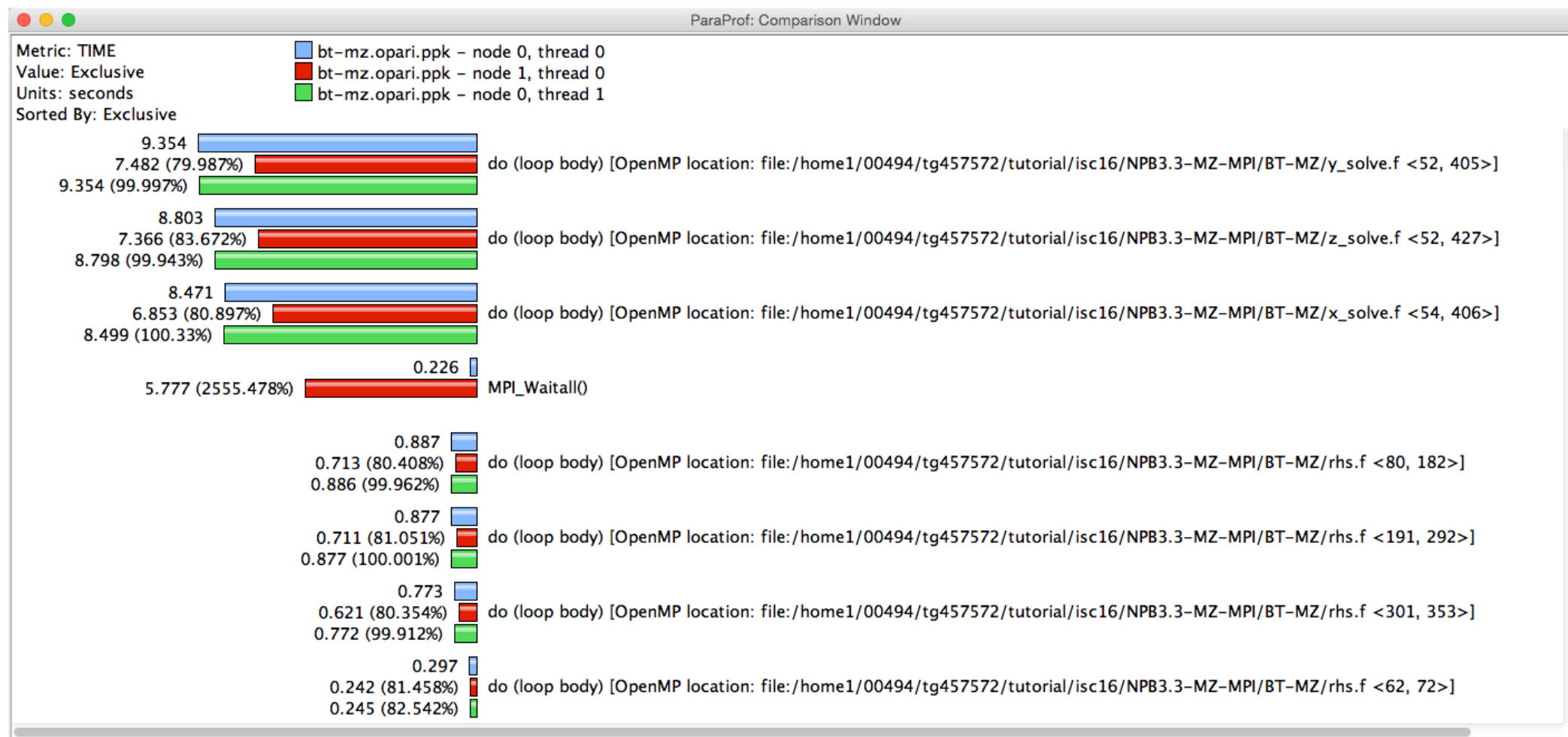
Show Source Code
Show Function Bar Chart
Show Function Histogram
Assign Function Color
Reset to Default Color

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

The screenshot shows the TAU: ParaProf Manager window. The left pane displays a tree view of applications, with 'scout.cubex' selected. The right pane shows a table of trial fields.

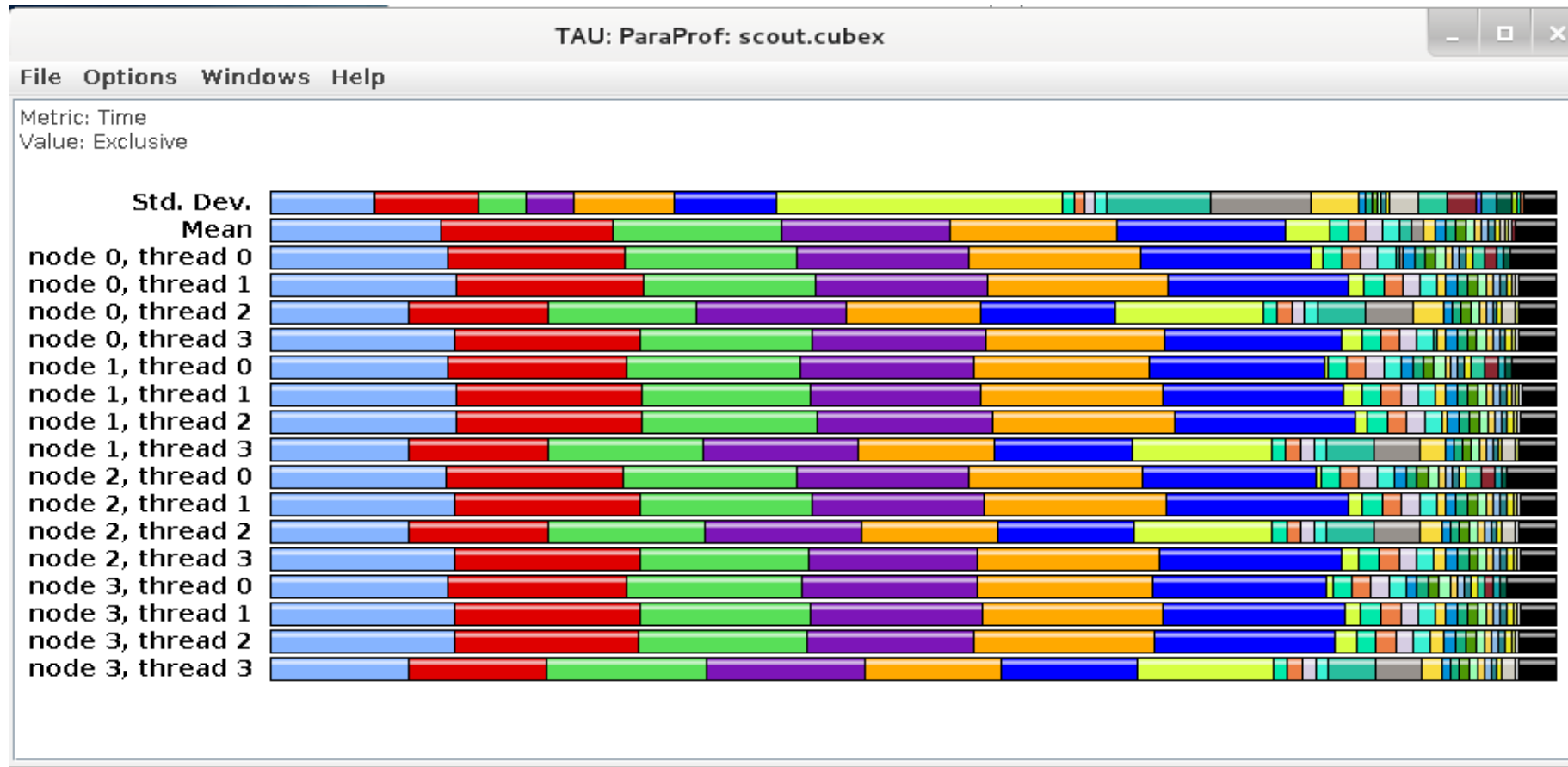
File Options Help

- Applications
 - Standard Applications
 - Default App
 - Default Exp
 - scout.cubex
 - Time
 - Wait at Barrier
 - Barrier Completion
 - Late Sender
 - Late Sender => Messages in Wrong Order
 - Late Sender => Messages in Wrong Order => Messages from different sources
 - Late Sender => Messages in Wrong Order => Messages from same source
 - Late Receiver
 - Early Reduce
 - Early Scan
 - Late Broadcast
 - Wait at N x N
 - N x N Completion
 - Management
 - Management => Fork
 - P2P send synchronizations
 - P2P send synchronizations => Late Receivers
 - P2P rcv synchronizations
 - P2P rcv synchronizations => Late Senders
 - P2P rcv synchronizations => Late Senders => Messages in Wrong Order
 - Collective synchronizations
 - P2P send communications
 - P2P send communications => Late Receivers
 - P2P rcv communications
 - P2P rcv communications => Late Senders
 - P2P rcv communications => Late Senders => Messages in Wrong Order
 - Collective exchange communications
 - Collective communications as source
 - Collective communications as destination
 - P2P bytes sent
 - P2P bytes received
 - Collective bytes outgoing
 - Collective bytes incoming
 - RMA bytes received
 - RMA bytes put

TrialField	Value
Name	scout.cubex
Application ID	0
Experiment ID	0
Trial ID	0
File Type Index	9
File Type Name	Cube

Metrics in the profile

ParaProf: Main Window



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	0.609	3,232	0
!\$omp do @rhs.f:80	0.583	0.583	3,232	0
MPI_Waitall	0.402	0.402	603	0
!\$omp implicit barrier	0.402	0.402	0	0
!\$omp do @rhs.f:301	0.36	0.36	0	0
!\$omp implicit barrier	0.026	0.026	0	0
!\$omp implicit barrier	0	0	0	0
!\$omp do @rhs.f:37	0.343	0.343	0	0
!\$omp do @rhs.f:62	0.225	0.225	0	0
!\$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: Callpath Thread Relations Window

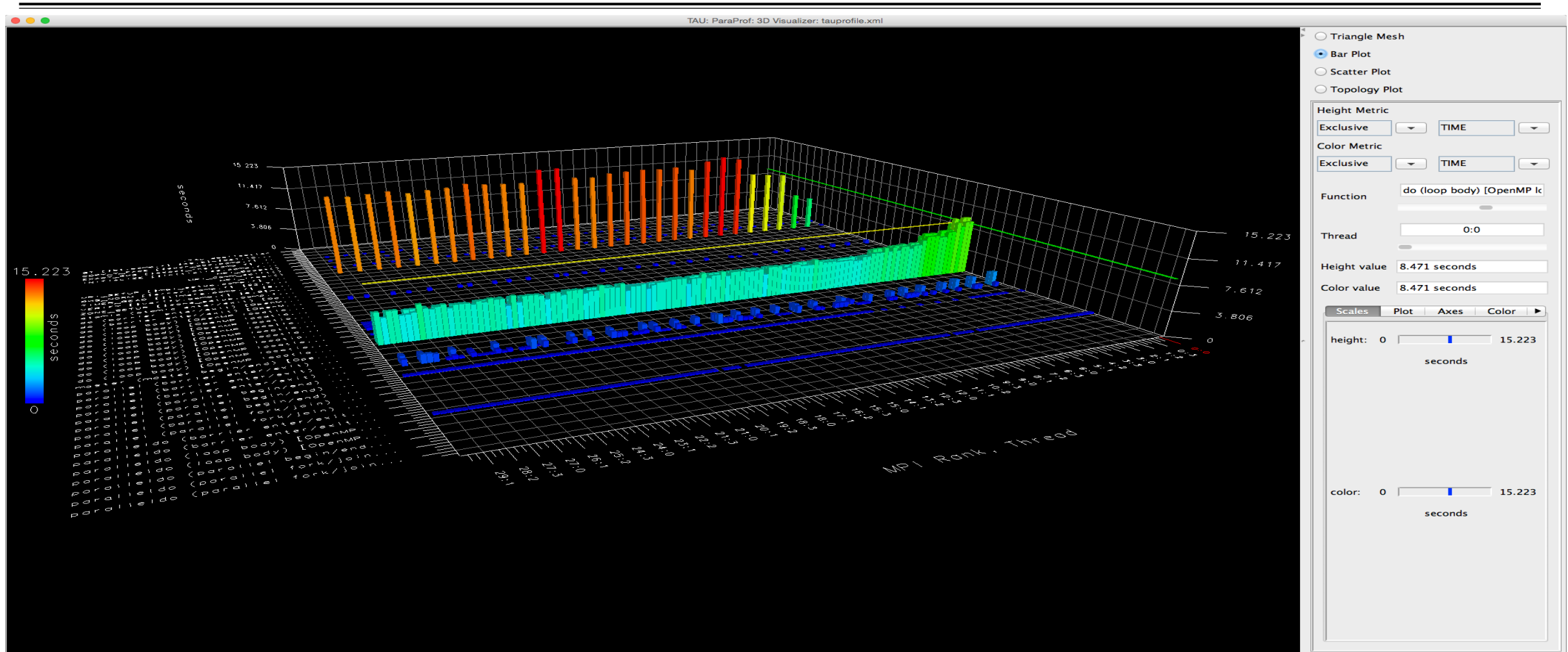
TAU: ParaProf: Call Path Data n,c,t, 0,0,0 – scout.cubex

File Options Windows Help

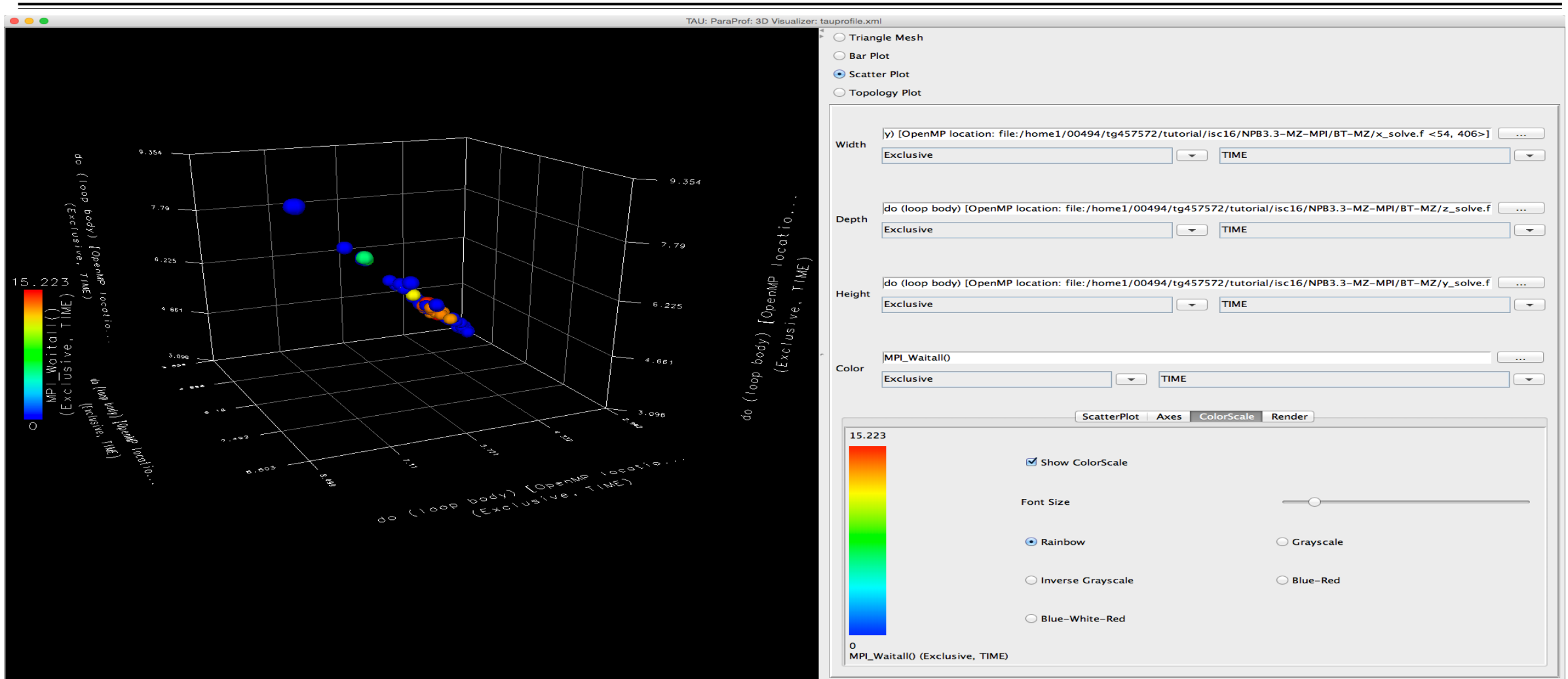
Metric Name: Time
Sorted By: Exclusive
Units: seconds

--v	0.04	0.04	32/32	!\$omp parallel @initialize.f:28
	0.04	0.04	32	!\$omp do @initialize.f:50
--v	0.03	2.536	3232/3232	compute_rhs_
	0.03	2.536	3232	!\$omp parallel @rhs.f:28
	9.8E-4	9.8E-4	3232/3232	!\$omp master @rhs.f:424
	0.225	0.228	3232/3232	!\$omp do @rhs.f:62
	0.002	0.002	3232/3232	!\$omp master @rhs.f:74
	0.002	0.002	3232/3232	!\$omp master @rhs.f:293
	0.199	0.199	3232/3232	!\$omp do @rhs.f:384
	0.002	0.002	3232/3232	!\$omp master @rhs.f:183
	0.343	0.343	3232/3232	!\$omp do @rhs.f:37
	0.016	0.016	3232/3232	!\$omp do @rhs.f:372
	0.014	0.027	3232/3232	!\$omp do @rhs.f:413
	0.609	0.609	3232/3232	!\$omp do @rhs.f:191
	0.36	0.386	3232/3232	!\$omp do @rhs.f:301
	0.583	0.583	3232/3232	!\$omp do @rhs.f:80
	0.019	0.019	3232/3232	!\$omp do @rhs.f:400
	0.006	0.006	3232/51680	!\$omp implicit barrier
	0.069	0.069	3232/3232	!\$omp do @rhs.f:428
	0.015	0.015	3232/3232	!\$omp do @rhs.f:359
--v	0.021	0.029	6432/6432	!\$omp parallel @exch_qbc.f:215
	0.021	0.029	6432	!\$omp parallel do @exch_qbc.f:215
	0.007	0.007	6432/51680	!\$omp implicit barrier
--v	0.02	0.033	6432/6432	!\$omp parallel @exch_qbc.f:255
	0.02	0.033	6432	!\$omp parallel do @exch_qbc.f:255
	0.013	0.013	6432/51680	!\$omp implicit barrier

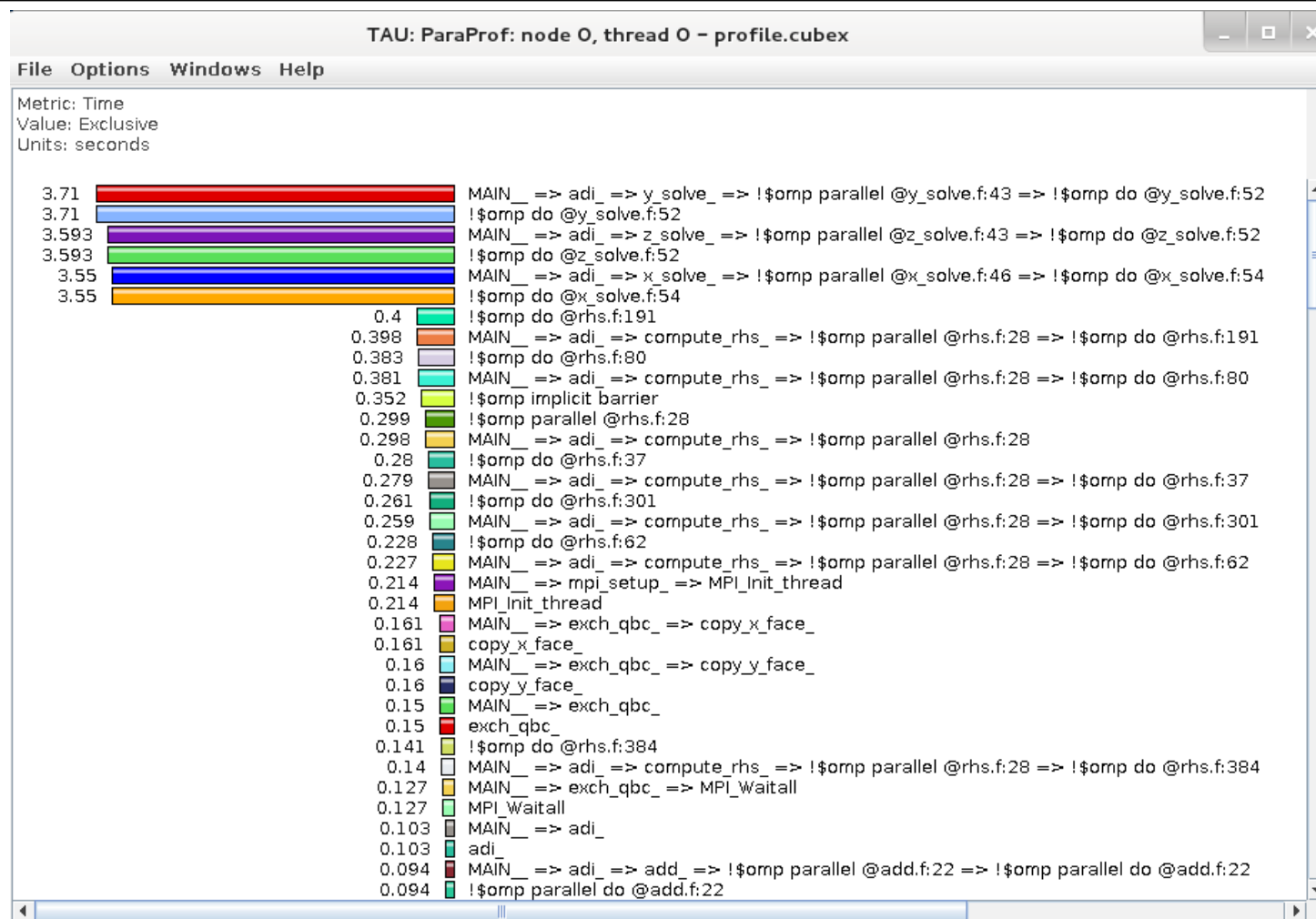
ParaProf: 3D Visualization Window Showing Entire Profile



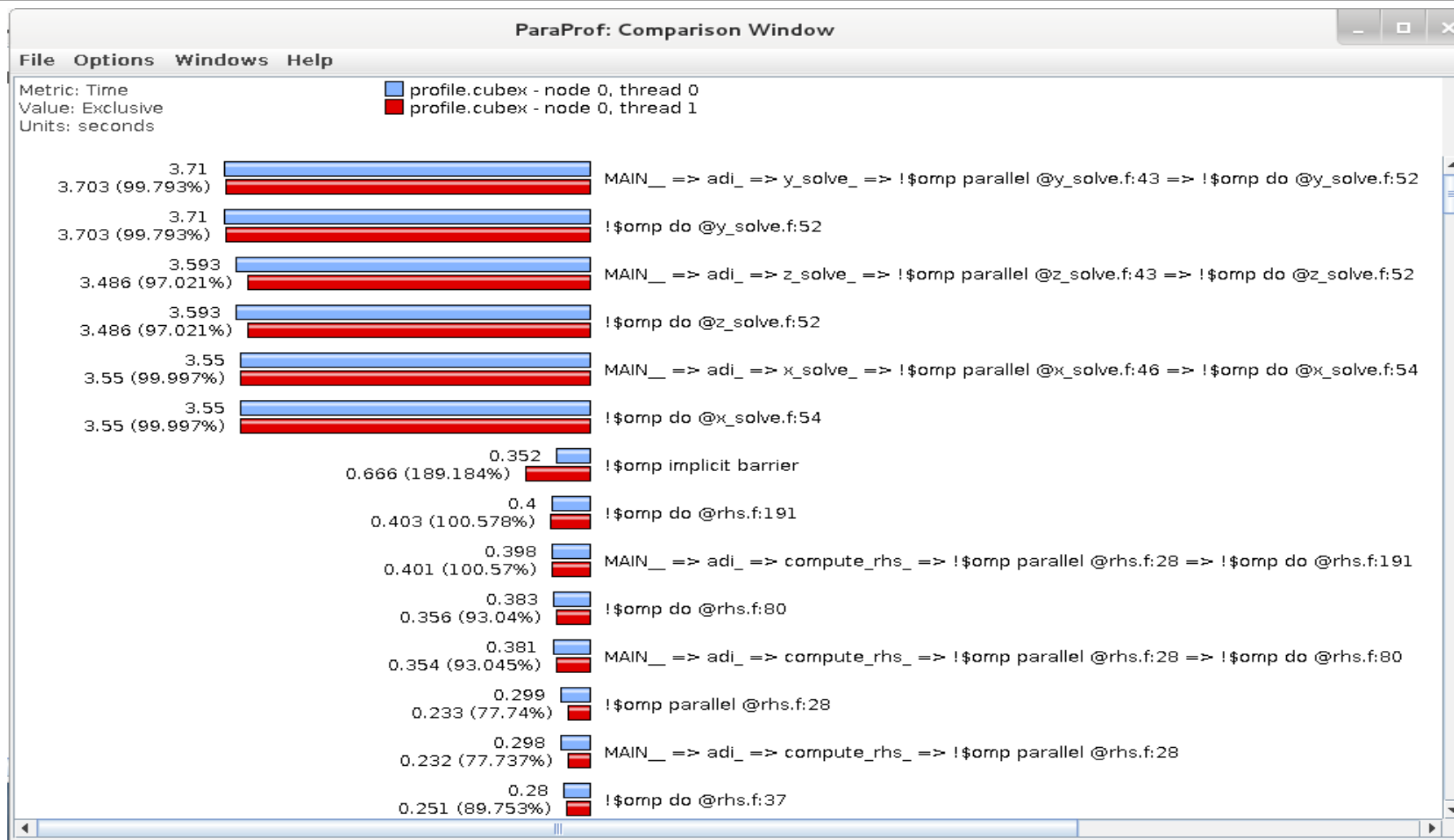
ParaProf: 3D Scatter Plot



ParaProf: Node View



ParaProf: Add thread to comparison window



ParaProf: Score-P Profile Files, Database

TAU: ParaProf Manager

File Options Help

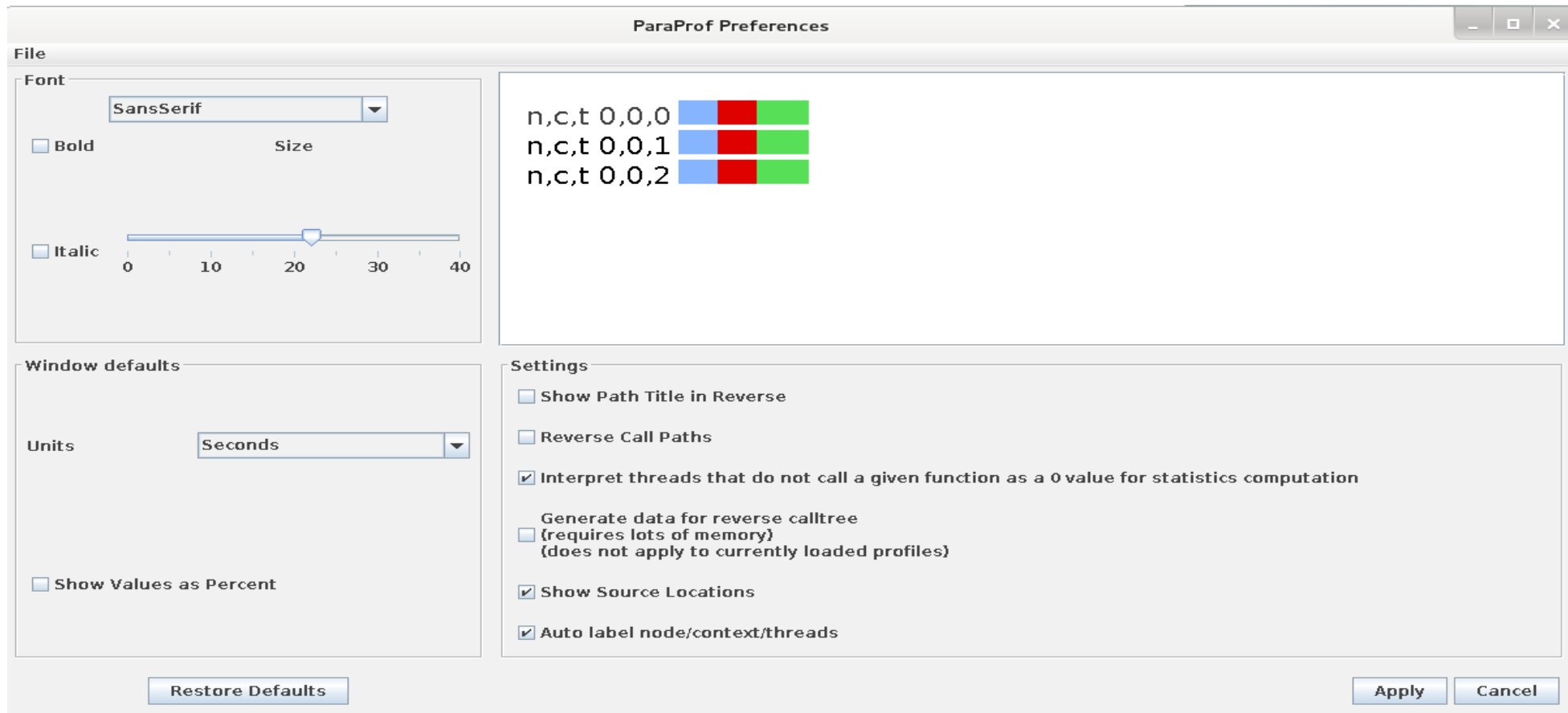
Applications

- Standard Applications
 - Default App
 - Default Exp
 - profile.cubex
 - Time
 - Minimum Inclusive Time
 - Maximum Inclusive Time
 - PAPI_TOT_CYC
 - PAPI_TOT_INS
 - PAPI_FP_INS
 - ru_utime
 - ru_stime
 - ru_maxrss
 - ru_ixrss
 - ru_idrss
 - ru_isrss
 - ru_minflt
 - ru_majflt
 - ru_nswap
 - ru_inblock
 - ru_oublock
 - ru_msgsnd
 - ru_msgrcv
 - ru_nsignals
 - ru_nvcsw
 - ru_nivcsw
 - bytes_sent
 - bytes_received
- Default (jdbc:h2:/home/livetau/.ParaProf/perfdmf;AUTO_SERVER=TRUE)
- perfexplorer_working (jdbc:h2:/home/livetau/.ParaProf/perfexplorer_wo

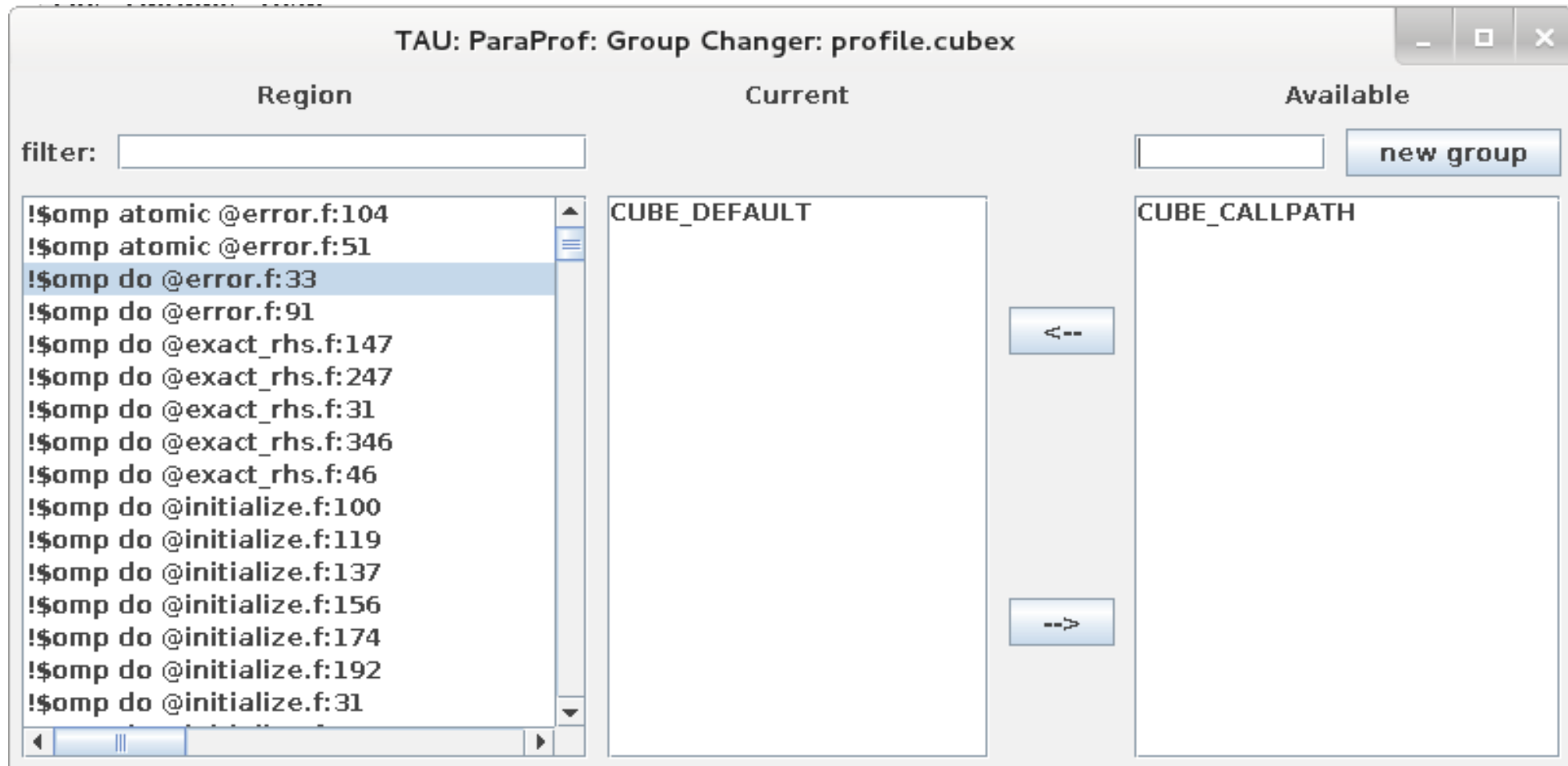
Add Application
Add Experiment
Add Trial

TrialField	Value
Name	profile.cubex
Application ID	0
Experiment ID	0
Trial ID	0
File Type Index	9
File Type Name	Cube

ParaProf: File Preferences Window



ParaProf: Group Changer Window



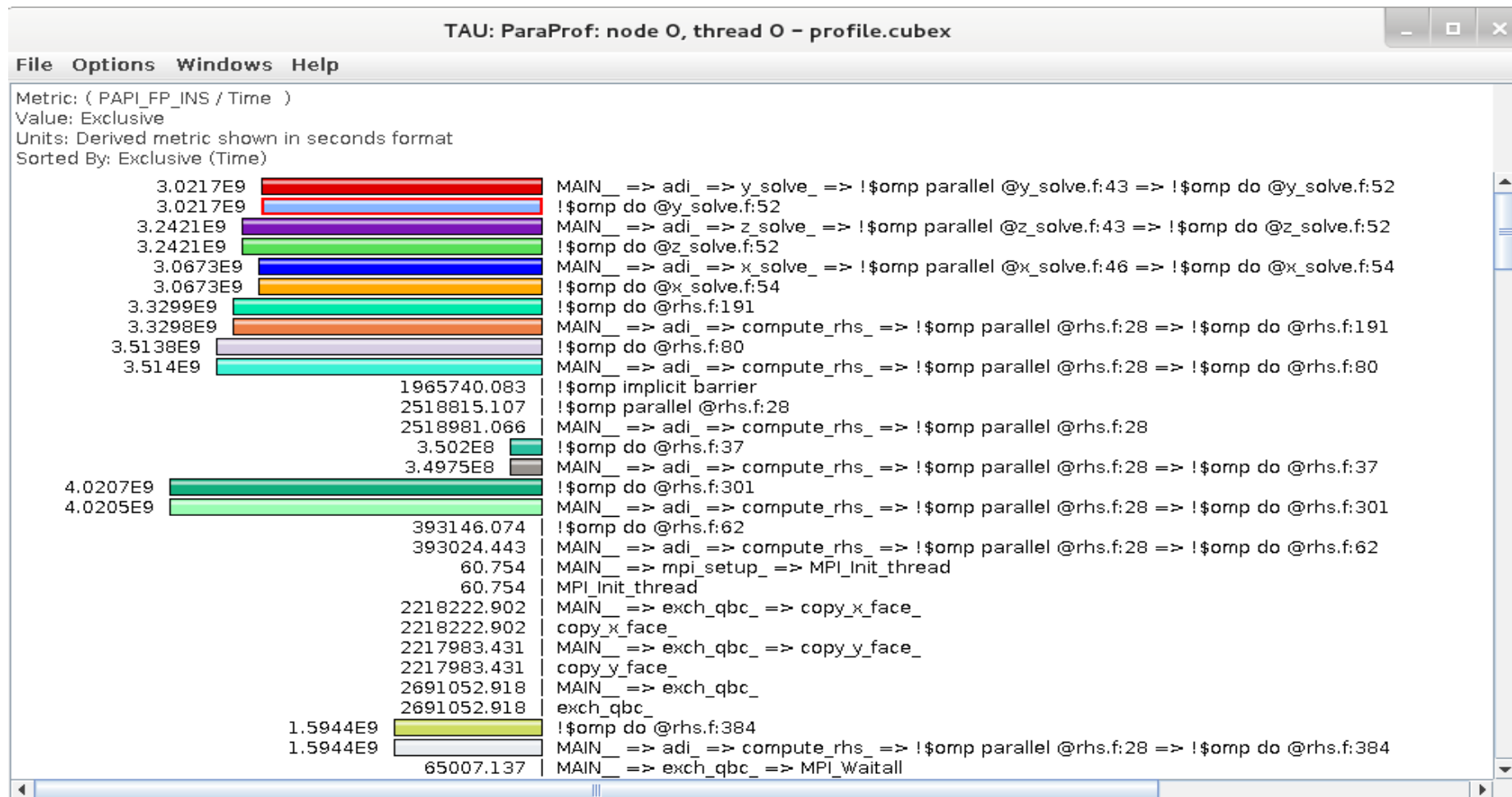
ParaProf: Derived Metric Panel in Manager Window

The screenshot displays the TAU: ParaProf Manager window. The interface includes a menu bar with 'File', 'Options', and 'Help'. On the left, a tree view shows the hierarchy: Applications > Standard Applications > Default App > Default Exp > profile.cubex. Under 'profile.cubex', a list of metrics is shown, with 'Time' selected. The right pane contains a table with two columns: 'MetricField' and 'Value'. The table lists several fields with their corresponding values.

MetricField	Value
Name	Time
Application ID	0
Experiment ID	0
Trial ID	0
Metric ID	0

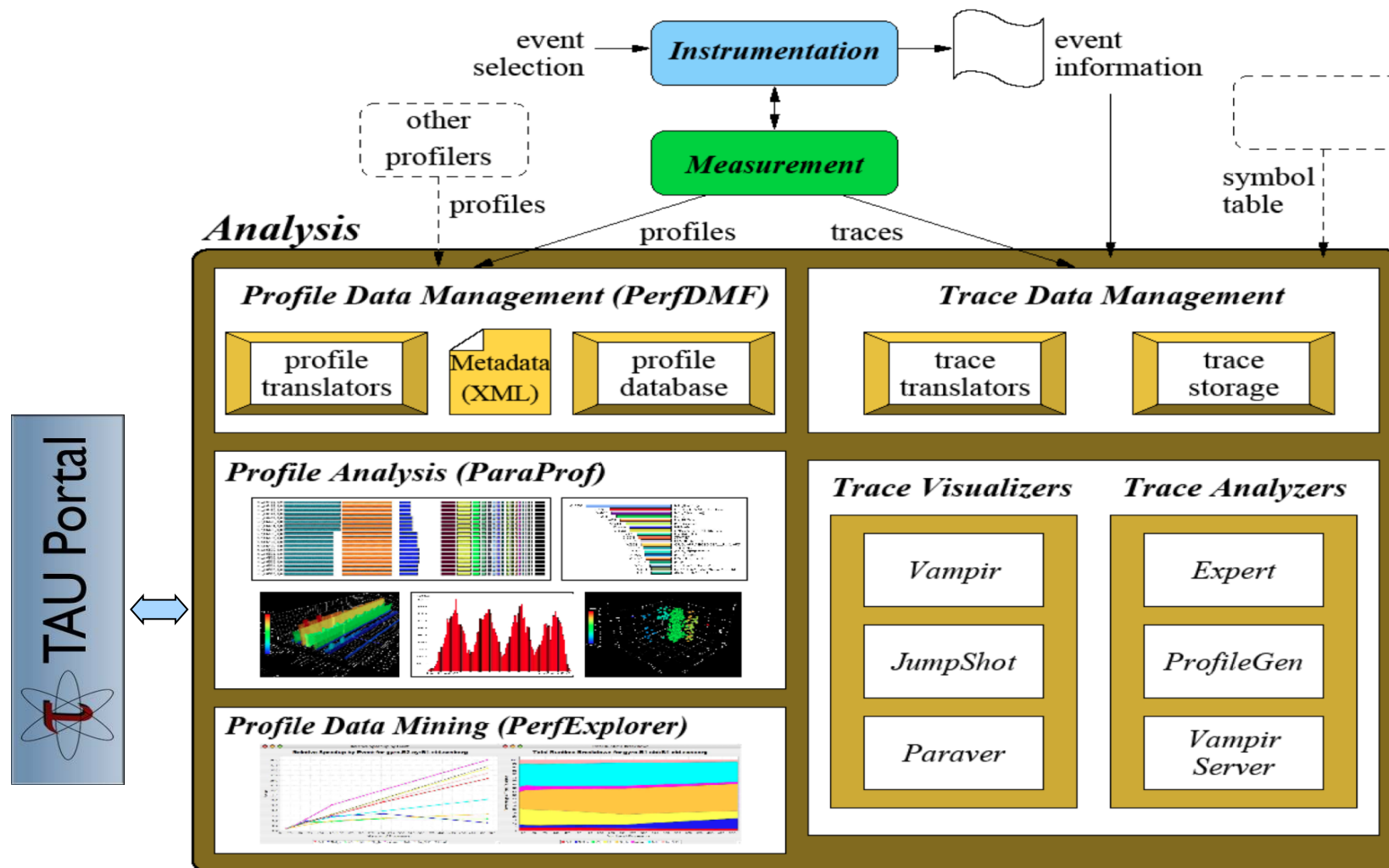
At the bottom, there is an 'Expression' field containing the text '"PAPI_FP_INS"/"Time"' and a 'Clear' button. Below the expression field is a row of buttons for mathematical operators: '+', '-', '*', '/', '=', '{', and '}', followed by an 'Apply' button.

Sorting Derived FLOPS metric by Exclusive Time

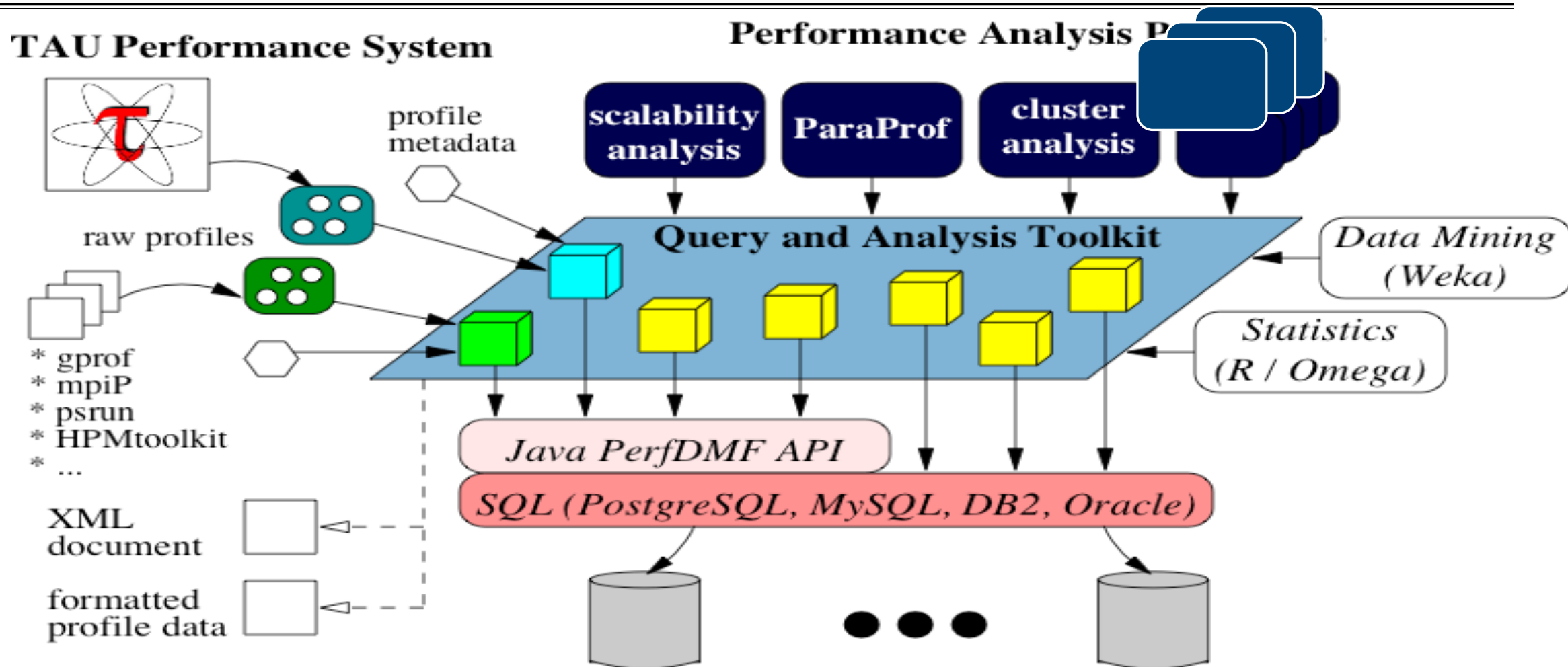


TAU's Analysis Tools: PerfExplorer

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 (*romeo, U. Reims*)

- Setup preferred program environment compilers

```
% module load tau
% cp $TAU/data.tgz . ; tar xzf data.tgz
% cd data
% cat README
  and follow the steps
% cd tau
% ./upload.sh
% perfexplorer
```

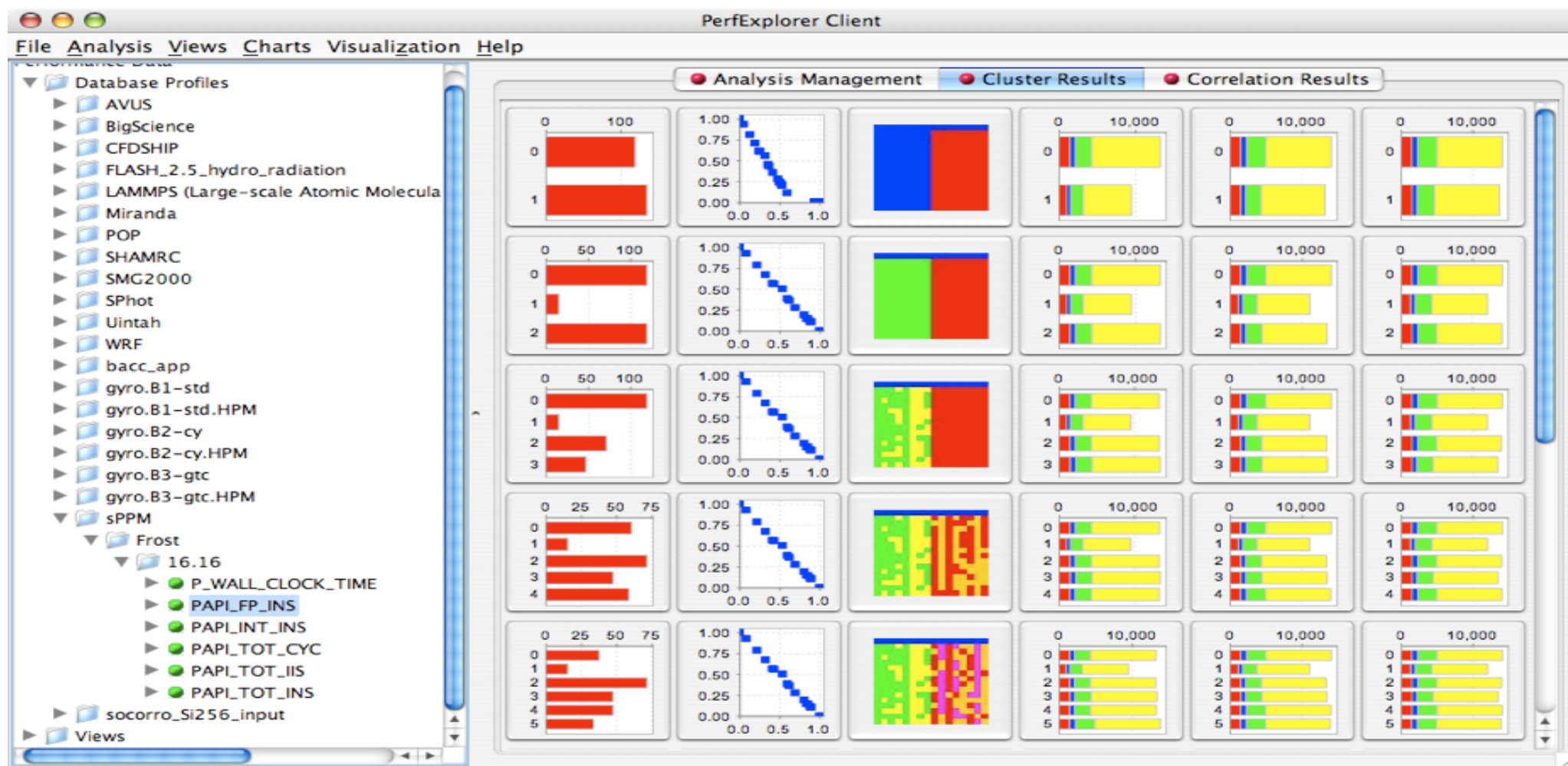
Performance Data Mining (PerfExplorer)

- 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

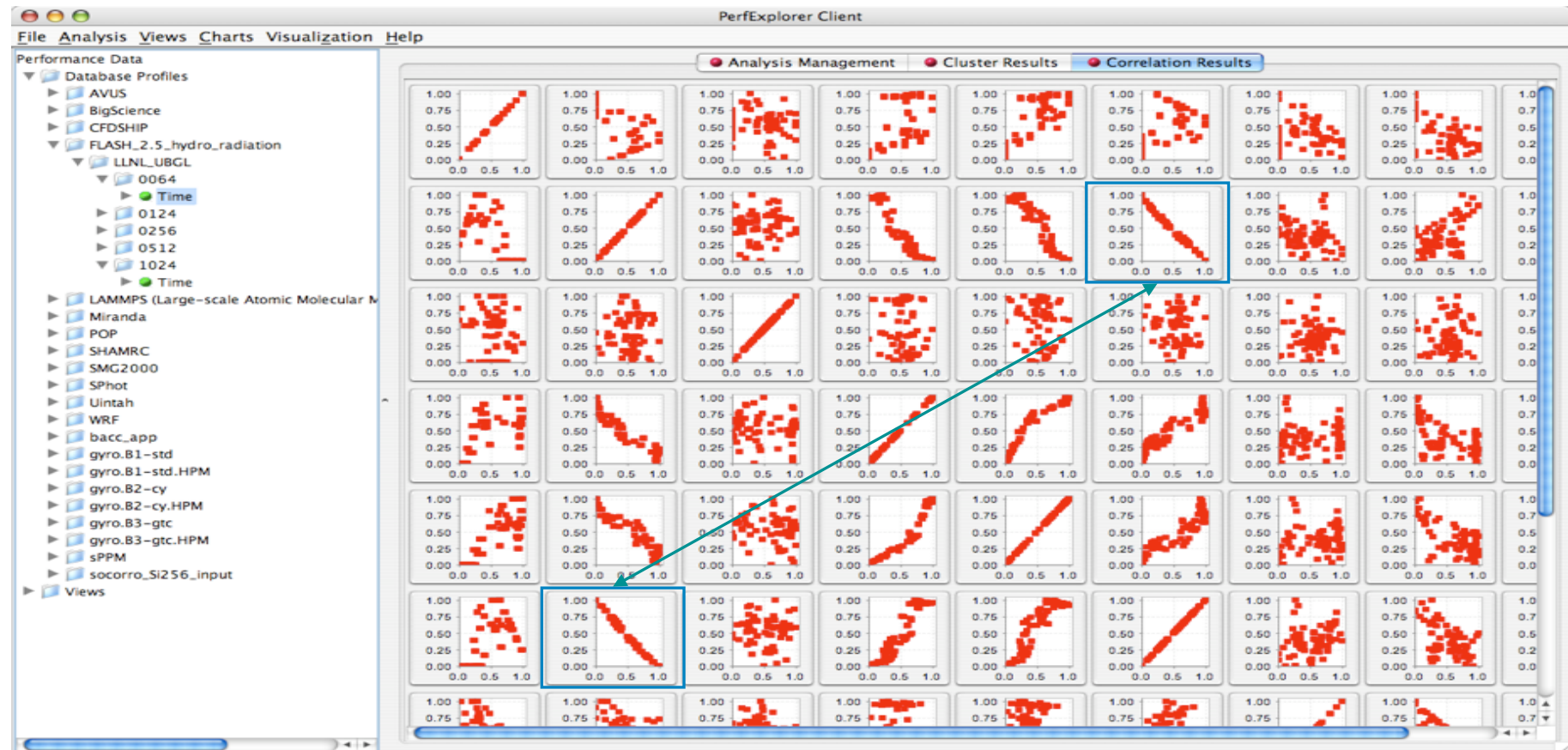
- Performance data represented as vectors - each dimension is the cumulative time for an event
- *k*-means: *k* random centers are selected and instances are grouped with the "closest" (Euclidean) center
- New centers are calculated and the process repeated until stabilization or max iterations
- Dimension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

PerfExplorer - Cluster Analysis (sPPM)



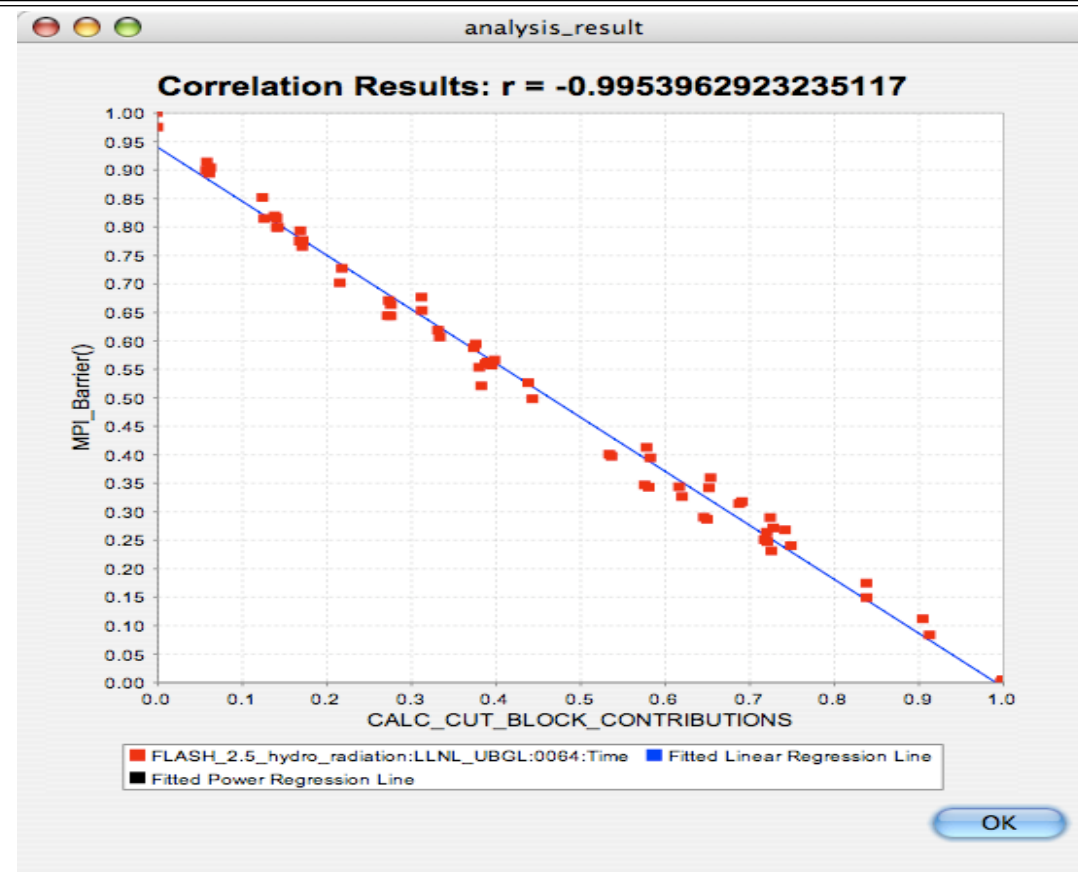
PerfExplorer - Correlation Analysis (Flash)

- 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

The screenshot shows the PerfExplorer Client interface. On the left is a file tree under 'gyro.B1-std'. The right pane is titled 'Analysis Management' and 'Performance Explorer', displaying a table of experiment metadata.

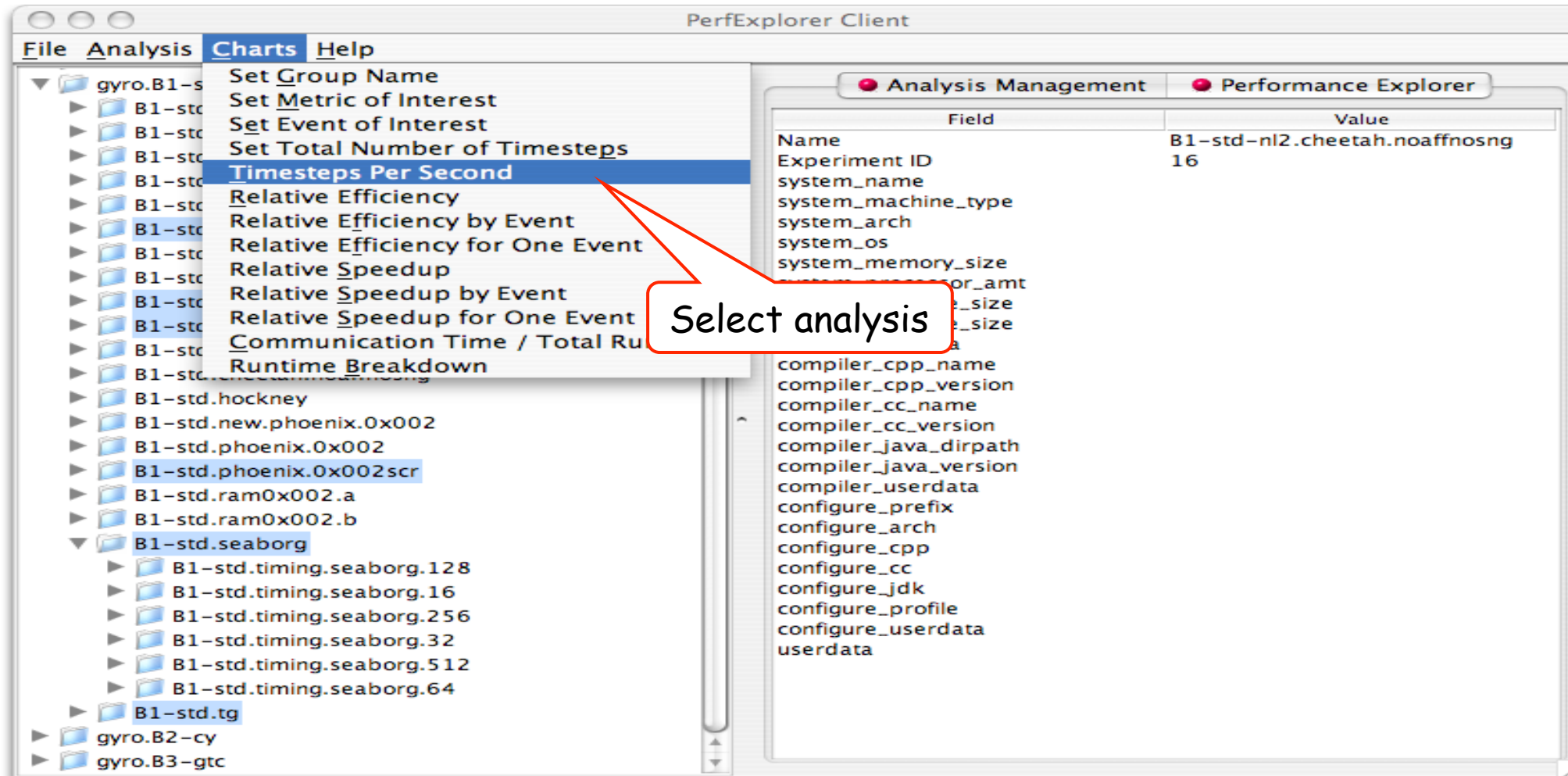
Select experiments and trials of interest

Experiment metadata

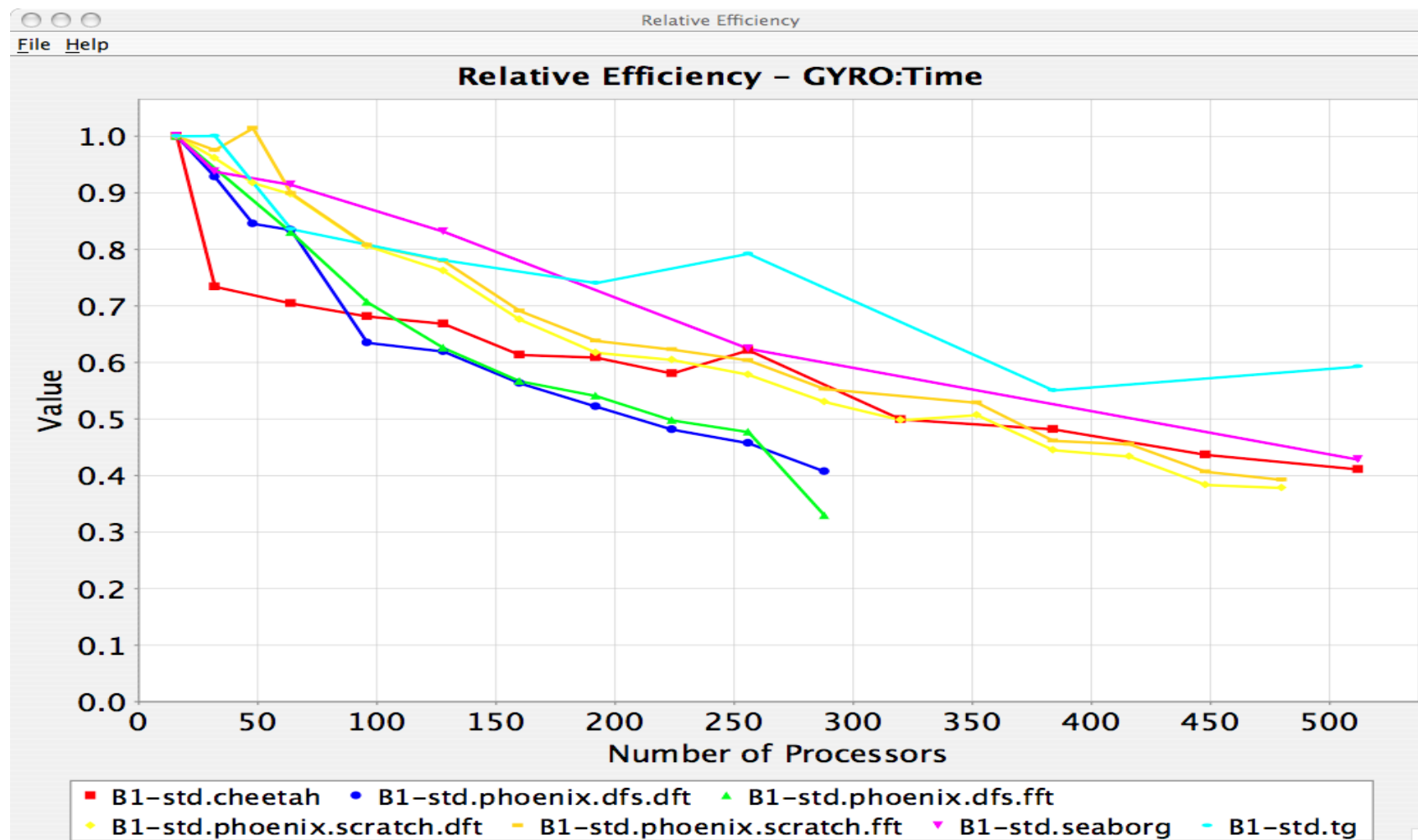
Data organized in application, experiment, trial structure (will allow arbitrary in future)

Field	Value
Name	B1-std-nl2.cheetah.noaffnosng
Experiment ID	16
system_name	
system_machine_type	
system_arch	
system_os	
system_memory_size	
system_processor_amt	
system_l1_cache_size	
system_l2_cache_size	
userdata	
cpp_name	
cpp_version	
cc_name	
cc_version	
compiler_java_dirpath	
compiler_java_version	
compiler_userdata	
configure_prefix	
configure_arch	

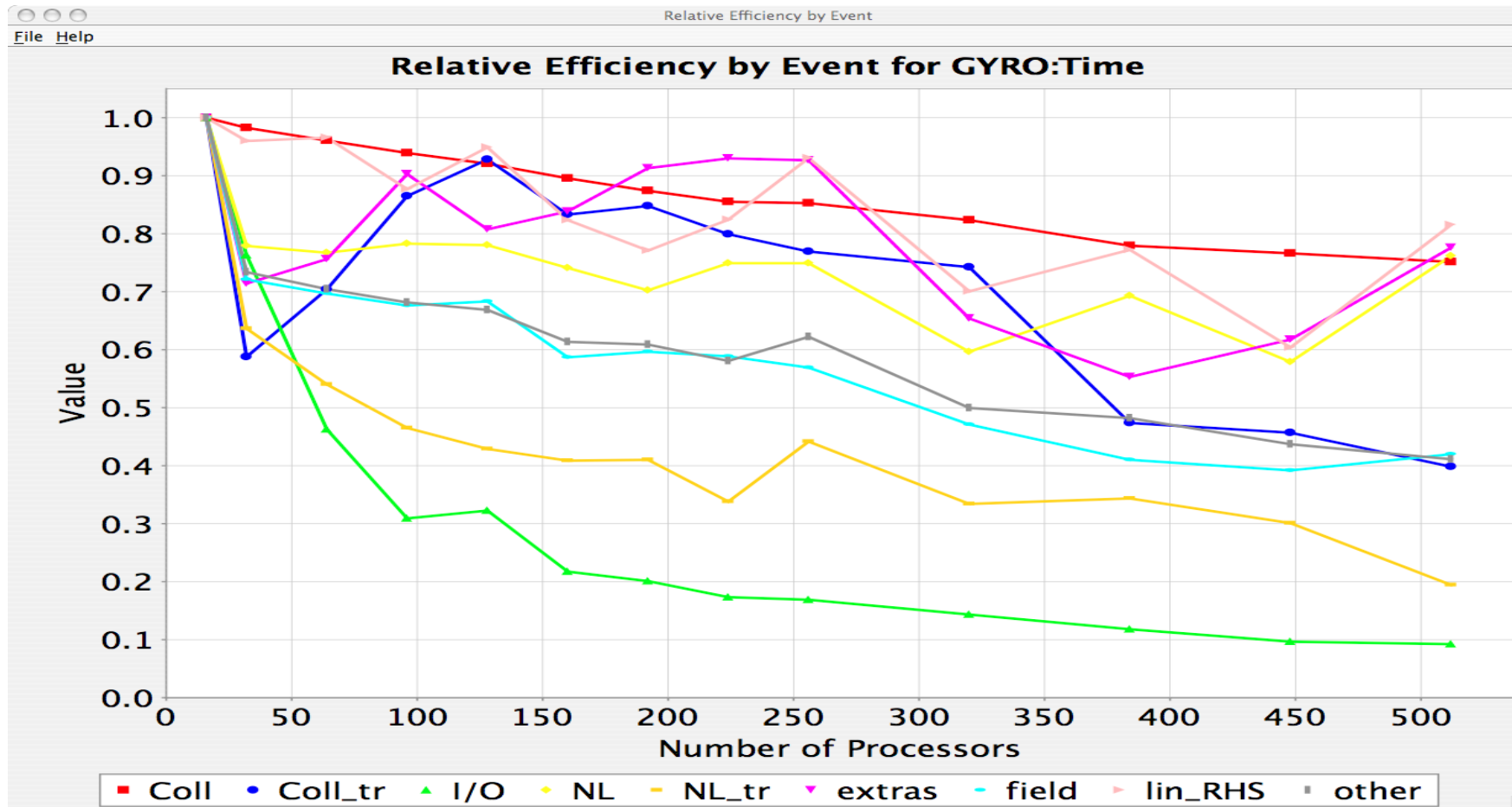
PerfExplorer - Interface



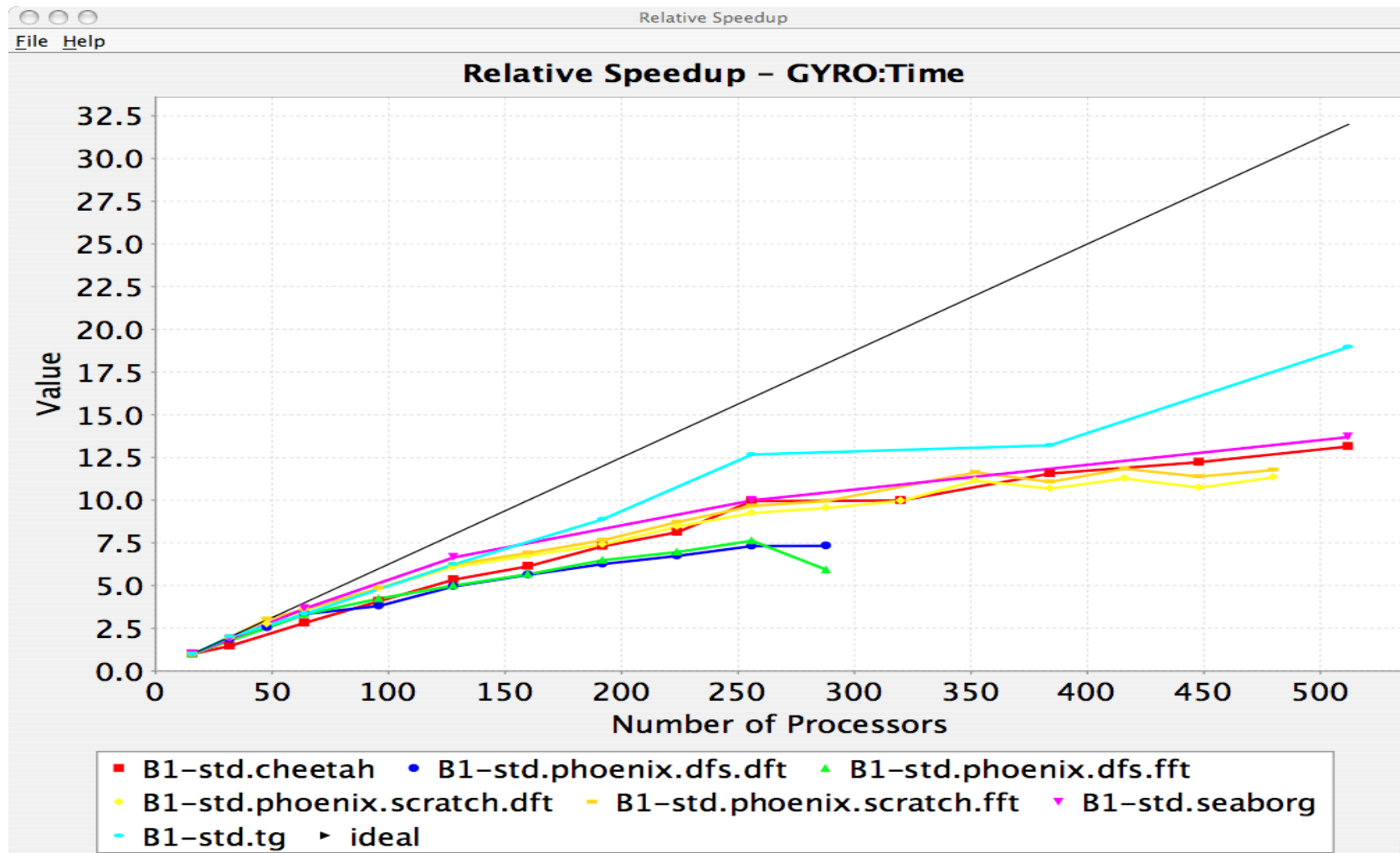
PerfExplorer - Relative Efficiency Plots



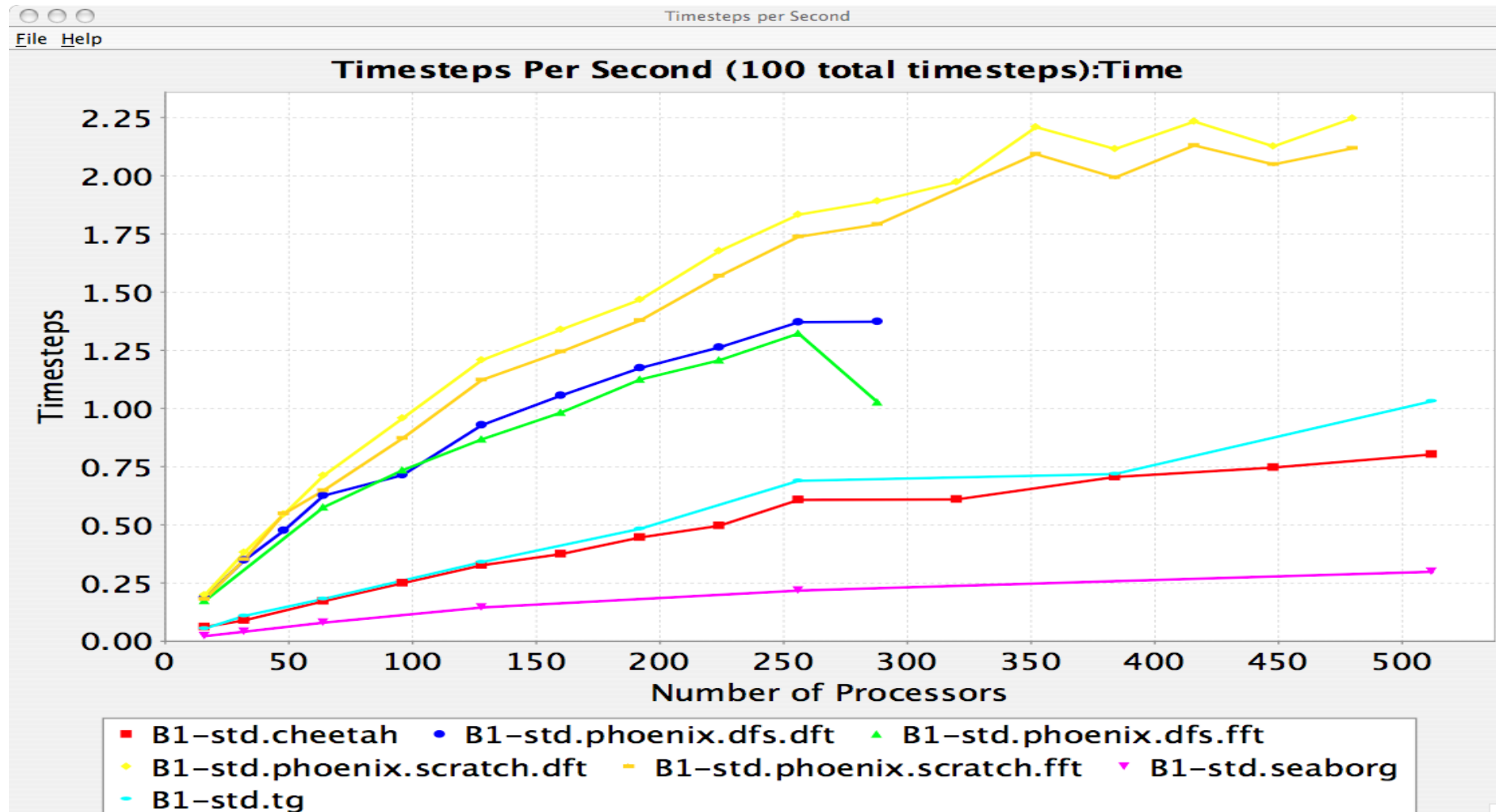
PerfExplorer - Relative Efficiency by Routine



PerfExplorer - Relative Speedup

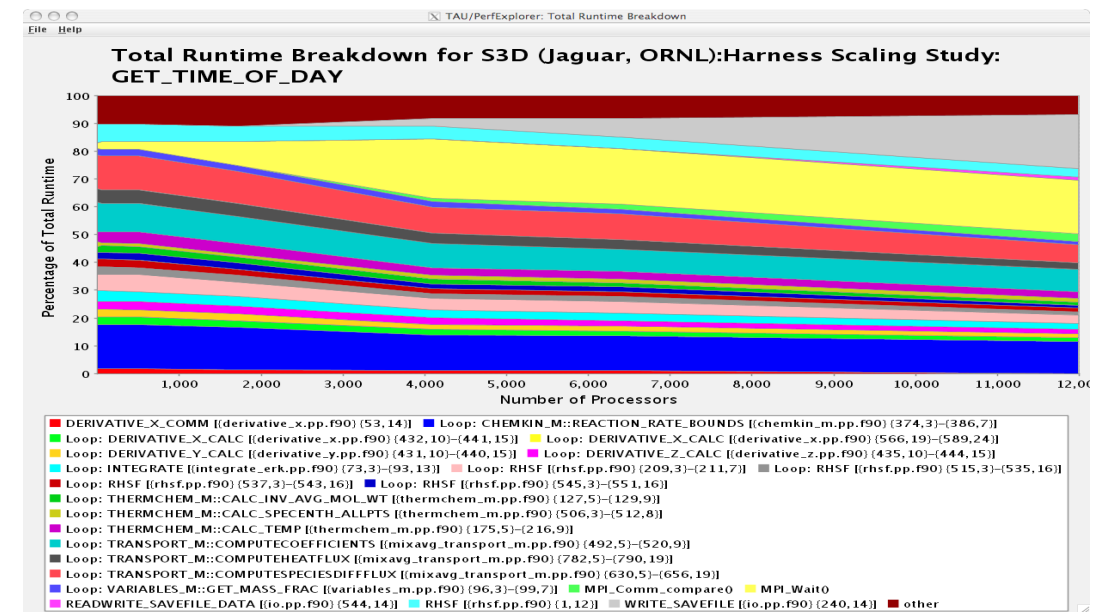
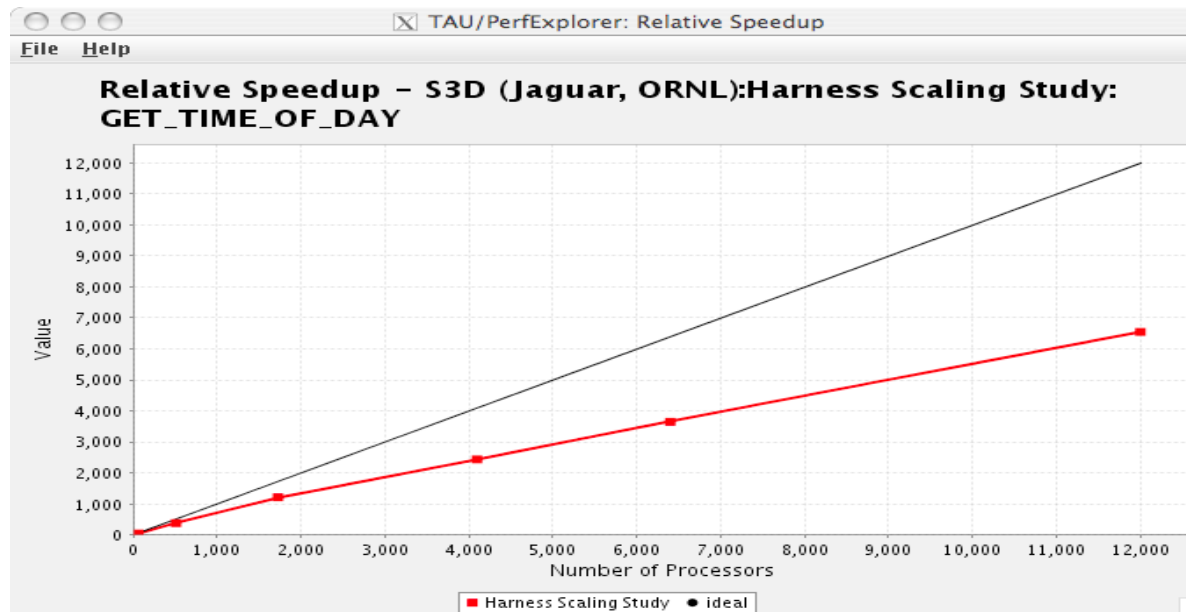


PerfExplorer - Timesteps Per Second

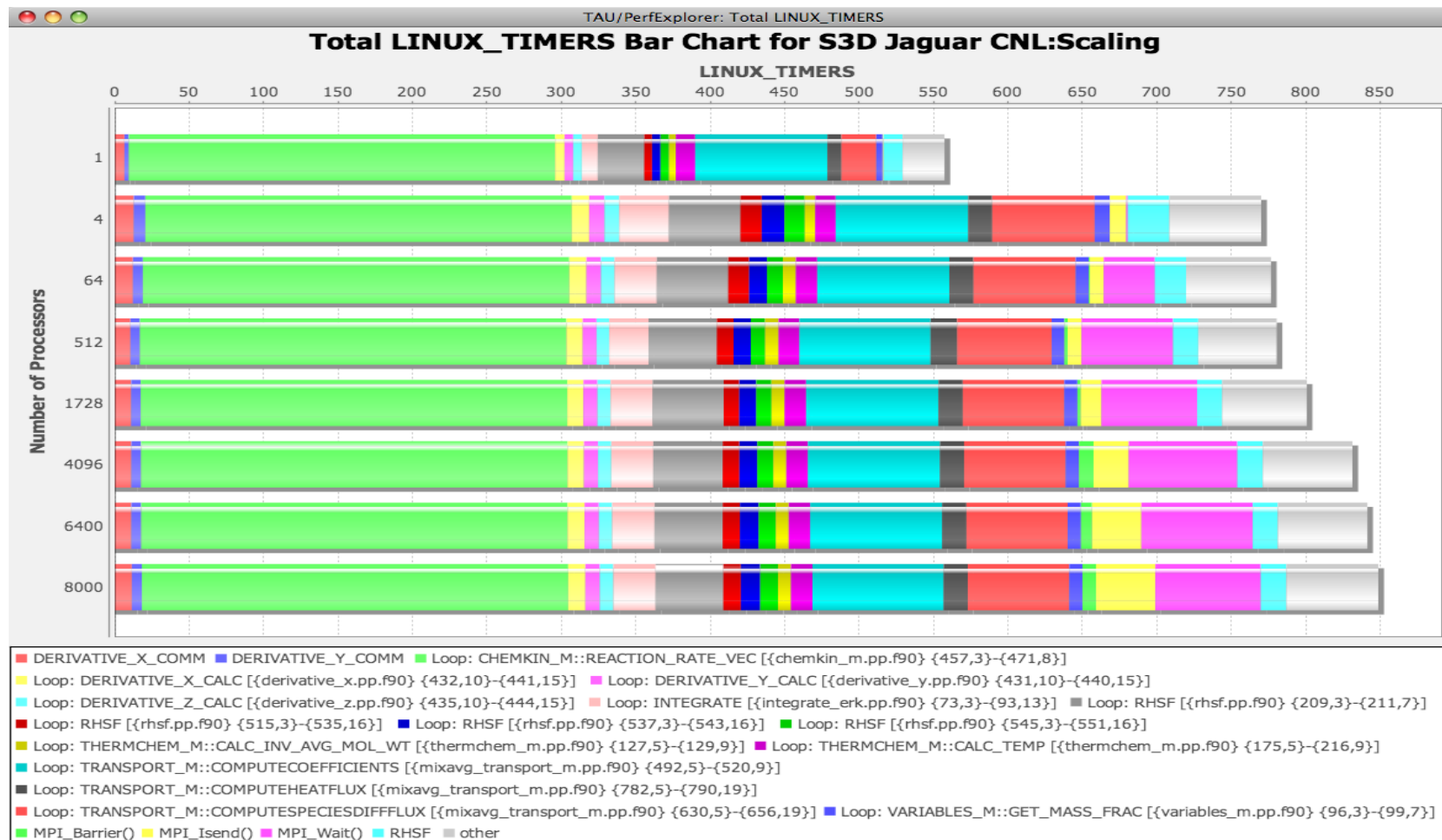


Evaluate Scalability

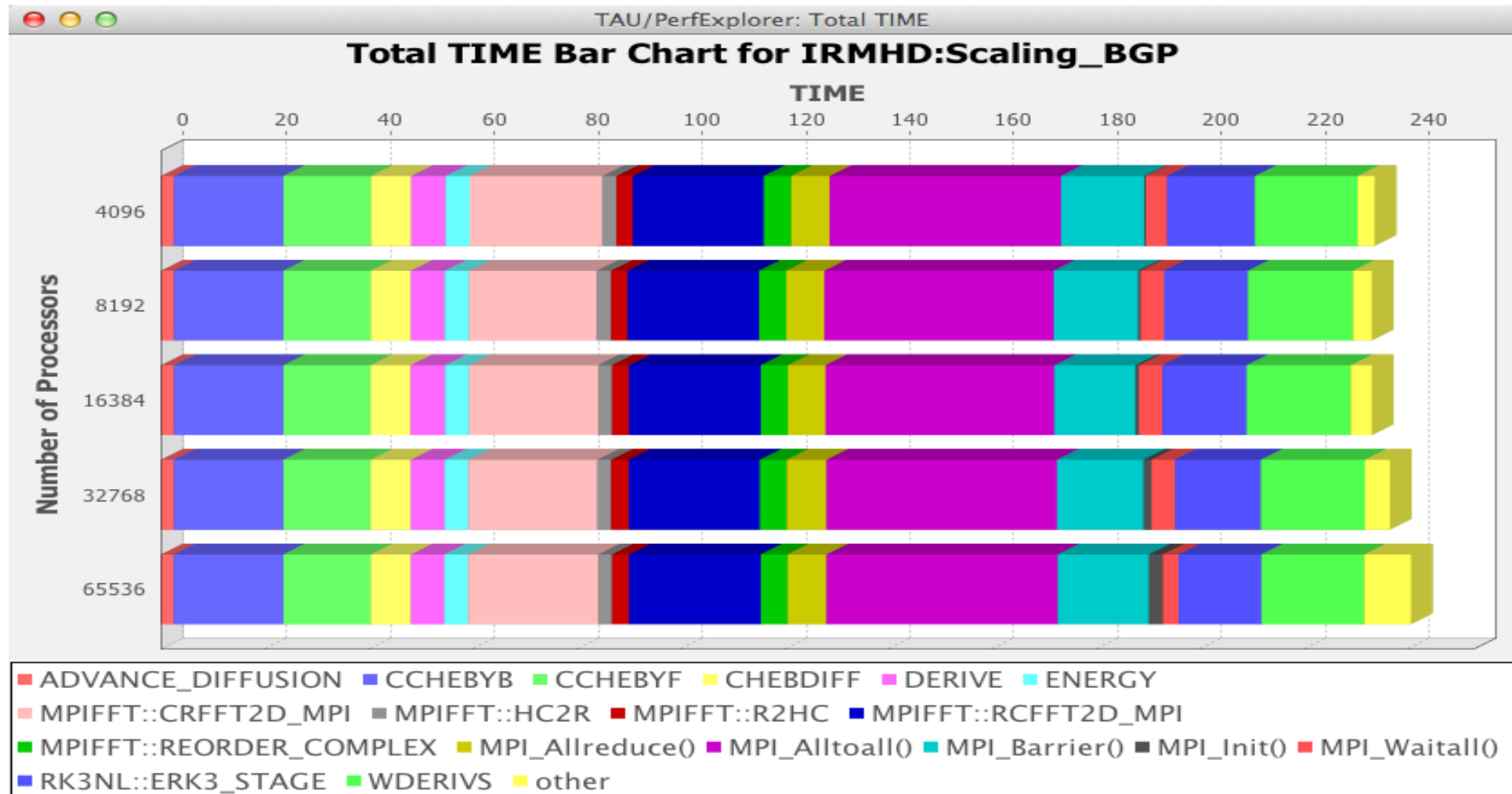
- Goal: How does my application scale? What bottlenecks occur at what core counts?
- Load profiles in taudb database and examine with PerfExplorer



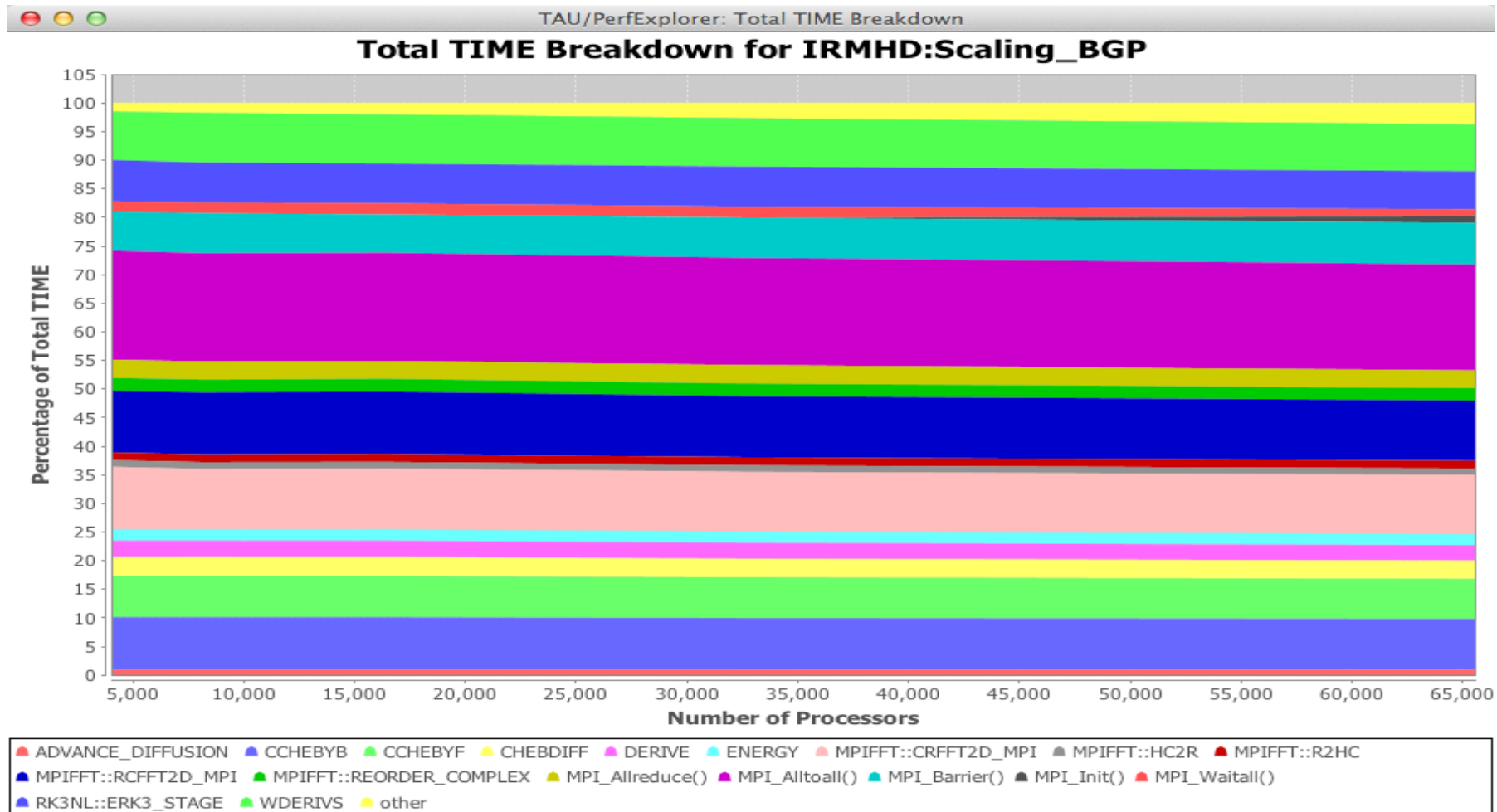
Evaluate Scalability



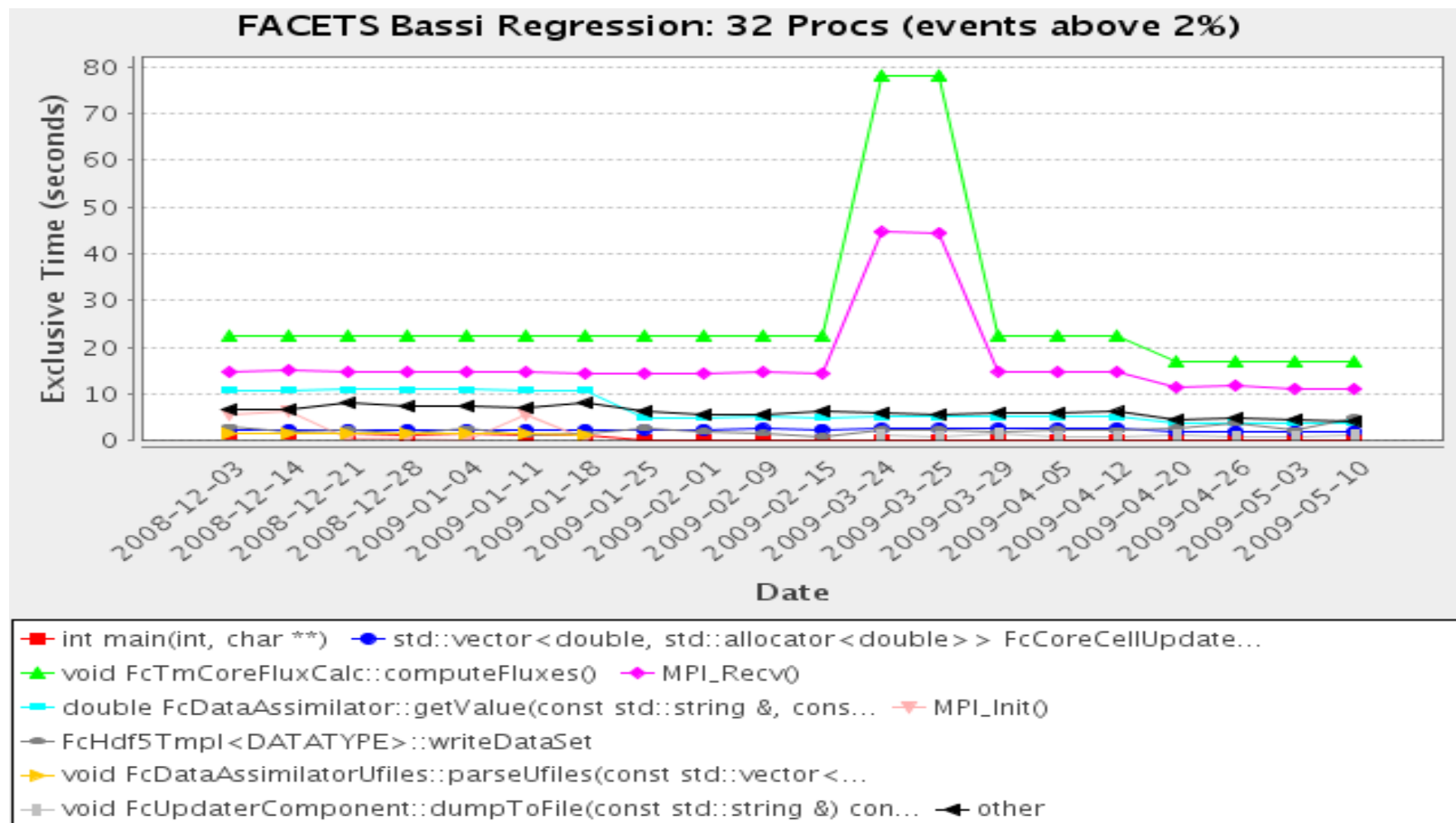
PerfExplorer



PerfExplorer



Performance Regression Testing



Performance Research Lab, University of Oregon, Eugene, USA

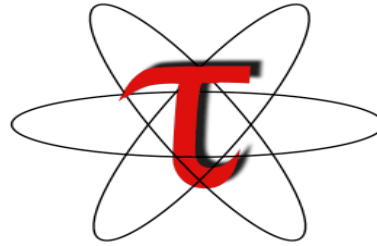


Support Acknowledgments

- US Department of Energy (DOE)
 - Office of Science contracts, 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
- 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