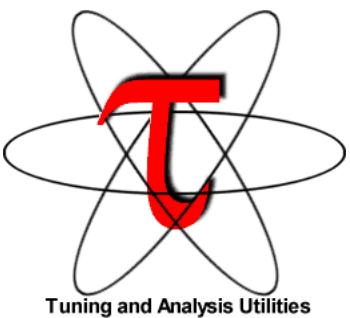


## TAU Performance System®

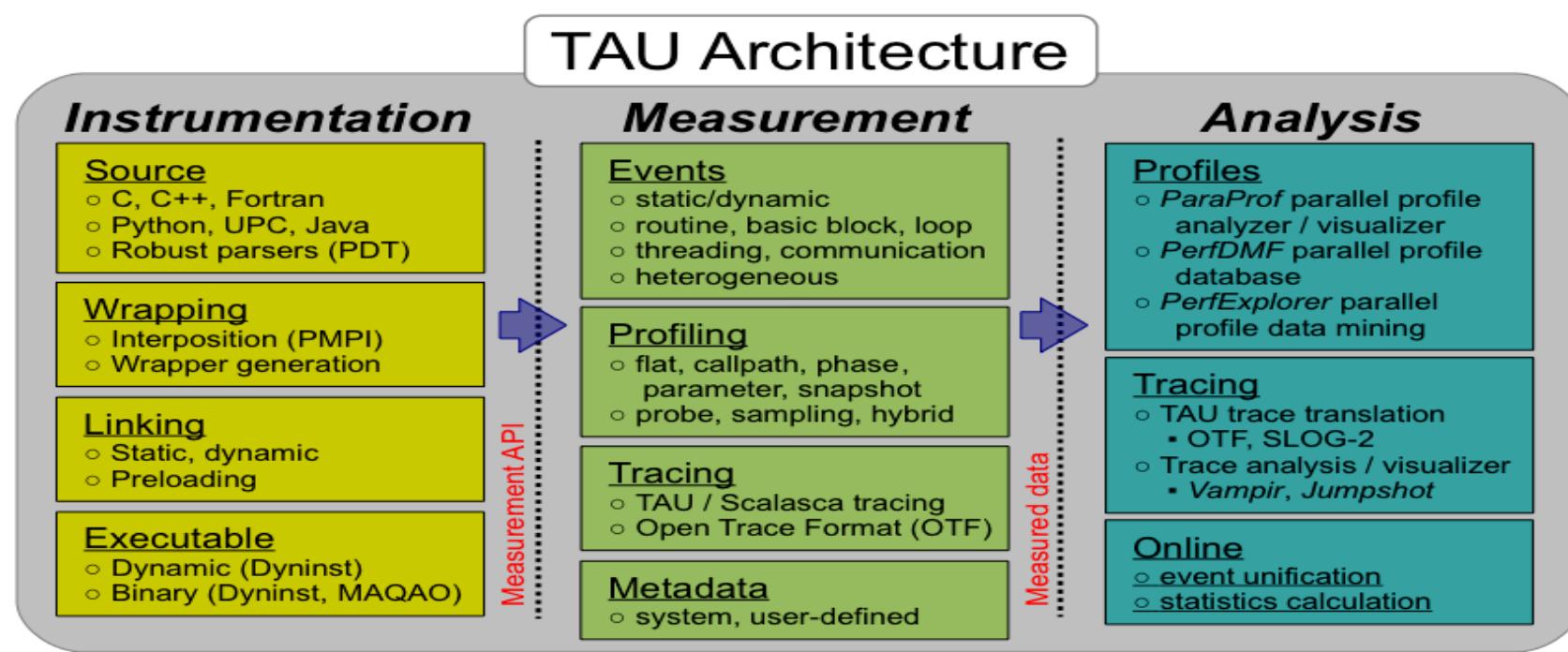
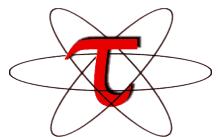


Sameer Shende  
[sameer@cs.uoregon.edu](mailto:sameer@cs.uoregon.edu)  
University of Oregon  
<http://tau.uoregon.edu>



# 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, Spark
  - Automatic instrumentation
- Measurement and analysis support
  - MPI, OpenSHMEM, ARMCI, PGAS, DMAPP
  - pthreads, OpenMP, OMPT interface, hybrid, other thread models
  - GPU, CUDA, OpenCL, OpenACC
  - Parallel profiling and tracing
  - Use of Score-P for native OTF2 and CUBEX generation
  - Efficient callpath 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
  - PDT based Source level instrumentation: routine & loop
  - Event based sampling (`TAU_SAMPLING=1` or `tau_exec -ebs`)
  - Callstack unwinding with sampling (`TAU_EBS_UNWIND=1`)
  - OpenMP Tools Interface (OMPT, `tau_exec -T ompt`)
  - CUDA CUPTI, OpenCL (`tau_exec -T cupti -cupti`)

# Application Performance Engineering using TAU

---

- How much time is spent in each application routine and outer *loops*? Within loops, what is the contribution of each *statement*? What is the time spent in OpenMP loops?
- How many instructions are executed in these code regions?  
Floating point, Level 1 and 2 *data cache misses*, hits, branches taken? What is the extent of vectorization for loops on Intel MIC?
- What is the memory usage of the code? When and where is memory allocated/de-allocated? Are there any memory leaks? What is the memory footprint of the application? What is the memory high water mark?
- How much energy does the application use in Joules? What is the peak power usage?
- What are the I/O characteristics of the code? What is the peak read and write *bandwidth* of individual calls, total volume?
- What is the contribution of each *phase* of the program? What is the time wasted/spent waiting for collectives, and I/O operations in Initialization, Computation, I/O phases?
- How does the application *scale*? What is the efficiency, runtime breakdown of performance across different core counts?

# Using TAU

---

- 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 UNITE tau
```

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

Use tau\_f90.sh, tau\_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)
```

# Installing and Configuring TAU

---

## ■ Installing PDT:

- wget [http://tau.uoregon.edu/pdt\\_lite.tgz](http://tau.uoregon.edu/pdt_lite.tgz)
- ./configure –prefix=<dir>; make ; make install

## ■ Installing TAU:

- wget <http://tau.uoregon.edu/tau.tgz>
- ./configure –scorep=download –arch=x86\_64 -bfd=download -pdt=<dir> -papi=<dir> ...
- For MIC (KNC):
- ./configure -scorep=download –arch=mic\_linux –pdt=<dir> -pdt\_c++=g++ -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

# Different Makefiles for TAU Compiler and Runtime Options

```
% module load UNITE tau  
% ls $TAU/Makefile.*  
  
Makefile.tau-icpc-papi-mpi-pdt  
Makefile.tau-icpc-papi-mpi-pdt-openmp-opari  
Makefile.tau-icpc-papi-mpi-pdt-openmp-opari-scorep
```

```
Makefile.tau-icpc-papi-mpi-pdt-scorep  
Makefile.tau-icpc-papi-mpi-pthread-pdt  
Makefile.tau-icpc-papi-ompt-mpi-pdt-openmp
```

- 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
```

On Claix, this is already setup for you in the UNITE tau module

```
% tau_f90.sh matmult.f90 -o matmult  
% mpirun -np 256 ./matmult  
% paraprof
```

# Compile-Time Options

---

Optional parameters for the TAU\_OPTIONS environment variable:

% tau_compiler.sh	
-optVerbose	Turn on verbose debugging messages
-optComInst	Use compiler based instrumentation
-optNoComInst	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 <code>-iowrapper</code> )
-optTrackGOMP	Enable tracking GNU OpenMP runtime layer (used without <code>-opari</code> )
-optMemDbg	Enable runtime bounds checking (see <code>TAU_MEMDBG_*</code> env vars)
-optKeepFiles	Does not remove intermediate .pdb and .inst.* files
-optPreProcess	Preprocess sources (OpenMP, Fortran) before instrumentation
-optTauSelectFile=" <code>&lt;file&gt;</code> "	Specify selective instrumentation file for <code>tau_instrumentor</code>
-optTauWrapFile=" <code>&lt;file&gt;</code> "	Specify path to <code>link_options.tau</code> generated by <code>tau_gen_wrapper</code>
-optHeaderInst	Enable Instrumentation of headers
-optTrackUPCR	Track UPC runtime layer routines (used with <code>tau_upc.sh</code> )
-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 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=‘-optComInst -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

# Selective Instrumentation File With Program Database Toolkit (PDT)

---

To use an instrumentation specification file for source instrumentation:

```
% export TAU_OPTIONS=' -optTauSelectFile=/path/to/pdt.inst -optVerbose ' ,  
% cat config/pdt.inst
```

```
BEGIN_EXCLUDE_LIST  
    BINVCRHS  
    MATMUL_SUB  
    MATVEC_SUB  
    EXACT_SOLUTION  
    BINVRHS  
    LHS#INIT  
    TIMER_  
END_EXCLUDE_LIST
```

# Runtime Environment Variables

Environment Variable	Default	Description
TAU_TRACE	0	Setting to 1 turns on tracing
TAU_CALLPATH	0	Setting to 1 turns on callpath profiling
TAU_TRACK_MEMORY_FOOTPRINT	0	Setting to 1 turns on tracking memory usage by sampling periodically the resident set size and high water mark of memory usage
TAU_TRACK_POWER	0	Tracks power usage by sampling periodically.
TAU_CALLPATH_DEPTH	2	Specifies depth of callpath. Setting to 0 generates no callpath or routine information, setting to 1 generates flat profile and context events have just parent information (e.g., Heap Entry: foo)
TAU_SAMPLING	1	Setting to 1 enables event-based sampling.
TAU_TRACK_SIGNALS	0	Setting to 1 generate debugging callstack info when a program crashes
TAU_COMM_MATRIX	0	Setting to 1 generates communication matrix display using context events
TAU_THROTTLE	1	Setting to 0 turns off throttling. Throttles instrumentation in lightweight routines that are called frequently
TAU_THROTTLE_NUMCALLS	100000	Specifies the number of calls before testing for throttling
TAU_THROTTLE_PERCALL	10	Specifies value in microseconds. Throttle a routine if it is called over 100000 times and takes less than 10 usec of inclusive time per call
TAU_CALLSITE	0	Setting to 1 enables callsite profiling that shows where an instrumented function was called. Also compatible with tracing.
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_ALIGNMENT	Sizeof(int)	Byte alignment for memory allocations
TAU_EVENT_THRESHOLD	0.5	Define a threshold value (e.g., .25 is 25%) to trigger marker events for min/max

# Simplifying TAU's usage (`tau_exec`)

---

- Uninstrumented execution
  - % mpirun -np 8 ./a.out

- Track MPI performance
  - % mpirun -np 8 `tau_exec` ./a.out

- Track POSIX I/O and MPI performance (MPI enabled by default)

- % mpirun -np 8 `tau_exec -T mpi,pdt -io` ./a.out

- Track OpenMP runtime routines
  - % mpirun -np 8 `tau_exec -T ompt,pdt,mpi -ompt` ./a.out

- Track memory operations
  - % export TAU\_TRACK\_MEMORY\_LEAKS=1
  - % mpirun -np 8 `tau_exec -memory_debug` ./a.out (bounds check)

- Use event based sampling (compile with -g)
  - % mpirun -np 8 `tau_exec -ebs` ./a.out
  - Also -ebs\_source=<PAPI\_COUNTER> -ebs\_period=<overflow\_count>

- Load wrapper interposition library
  - % mpirun -np 8 `tau_exec -loadlib=<path/libwrapper.so>` ./a.out

- Track GPGPU operations
  - % mpirun -np 8 `tau_exec -cuhti` ./a.out
  - % mpirun -np 8 `tau_exec -opencl` ./a.out
  - % mpirun -np 8 `tau_exec -openacc` ./a.out

# Binary Rewriting Instrumentation

---

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

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

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

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

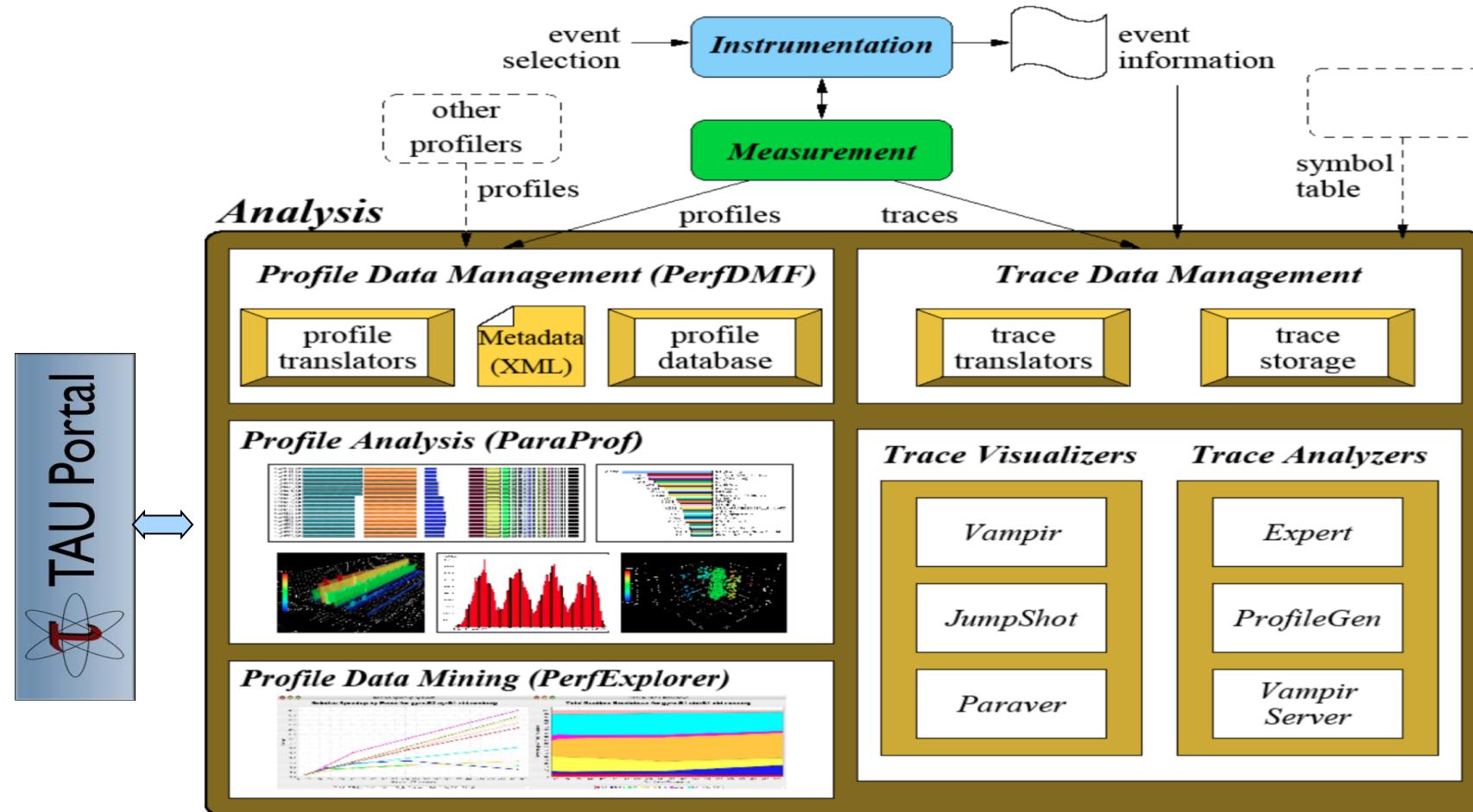
- Pebil [UC San Diego]:

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

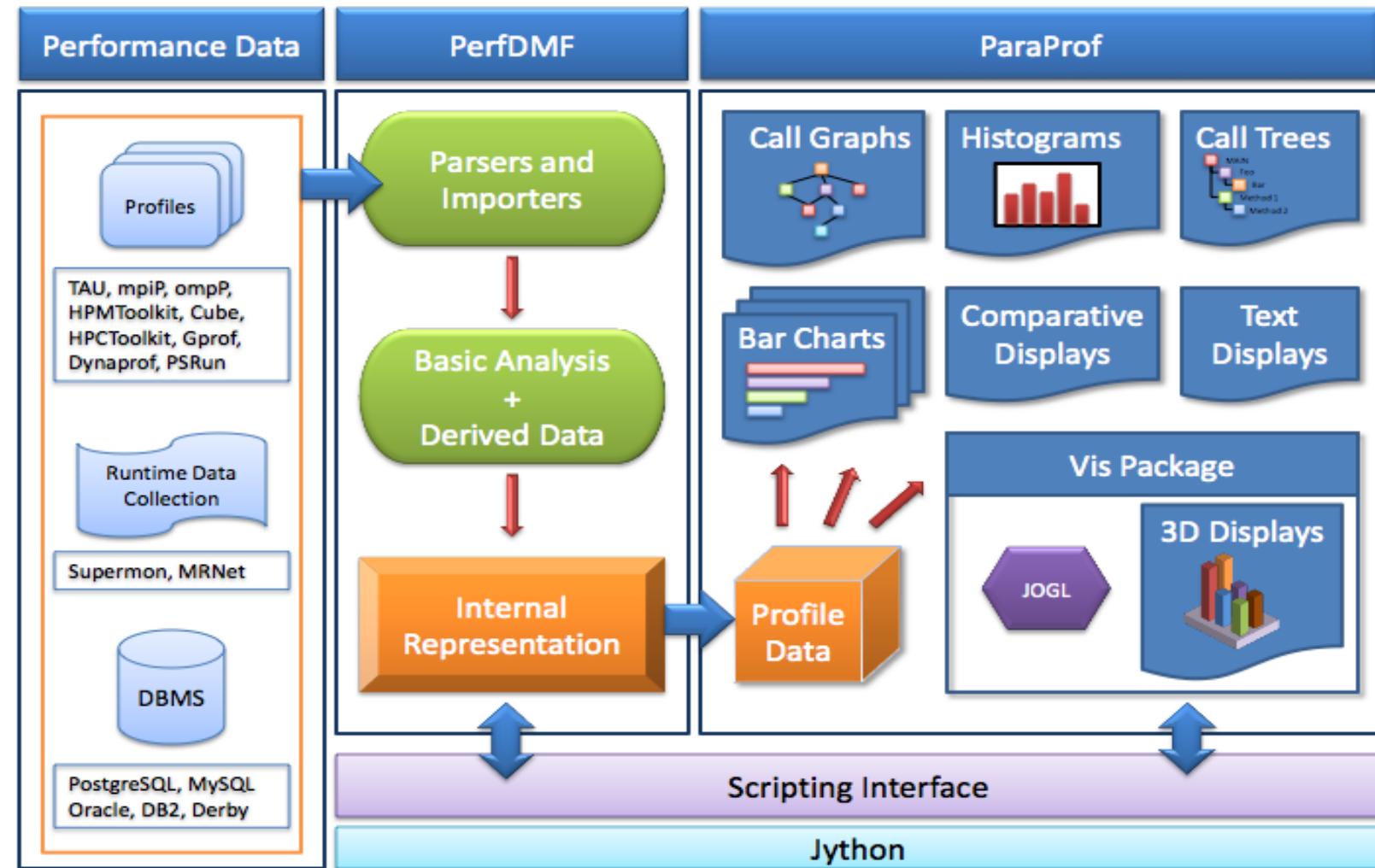
- Execute the application to get measurement data:

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

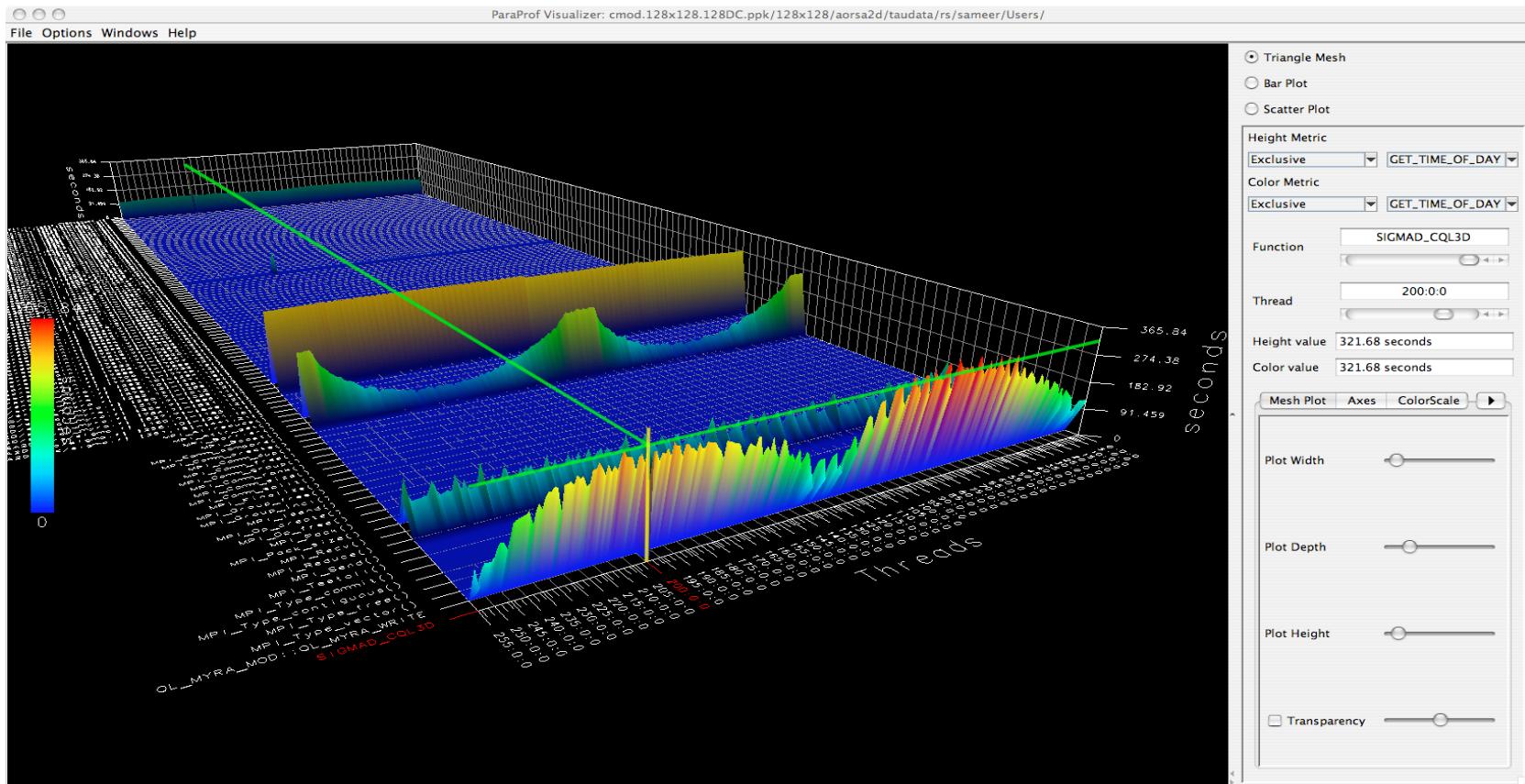
# TAU Analysis



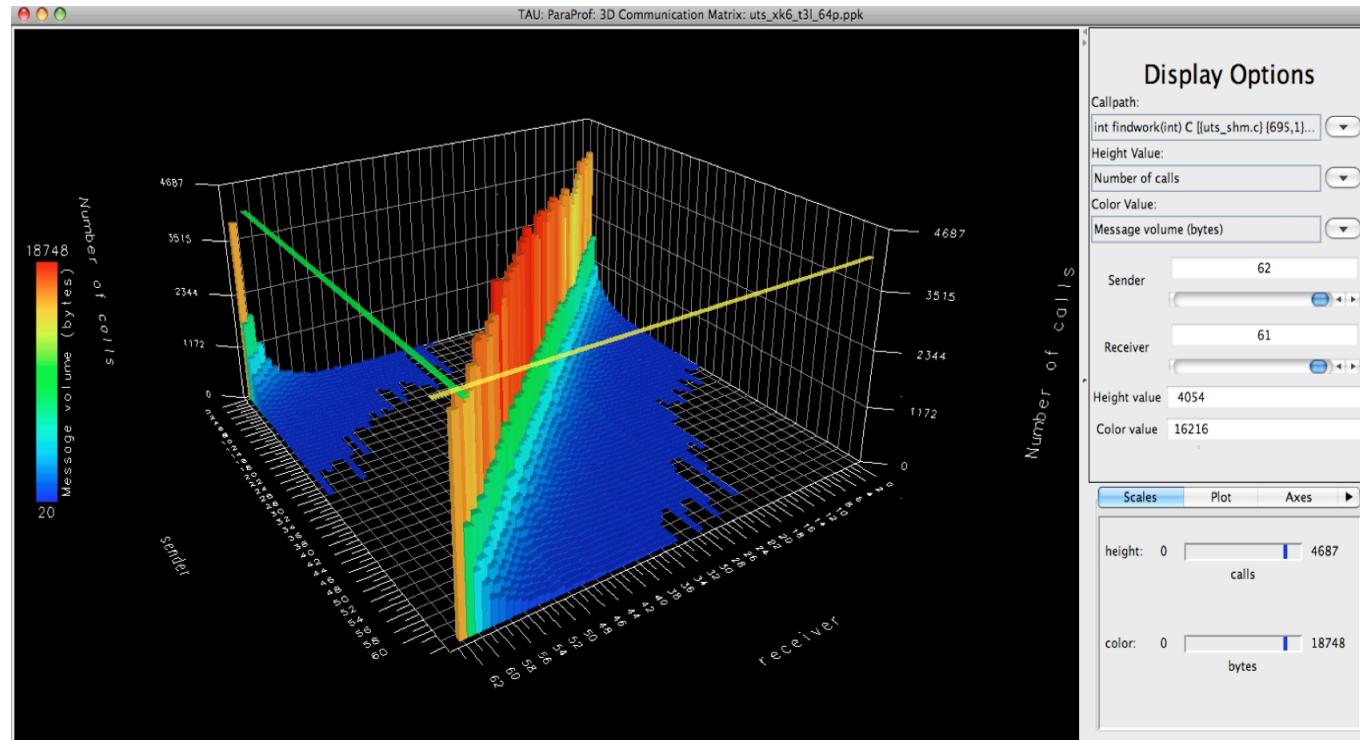
# ParaProf Profile Analysis Framework



# Parallel Profile Visualization: ParaProf



# ParaProf 3D Communication Matrix



% export TAU\_COMM\_MATRIX=1

# 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 (*Claix, RWTH*)

---

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

```
% module load UNITE tau
% paraprof profile.cubex &
```

For PerfExplorer:

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

## NPB-MZ-MPI Suite

---

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

<http://www.nas.nasa.gov/Software/NPB>

- 3 benchmarks in Fortran77
- Configurable for various sizes & classes
- Move into the NPB3.3-MZ-MPI root directory

```
% ls
bin/    common/   jobsctpt/  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 = mpiifort # Intel compiler  
  
# Alternative variant to perform instrumentation  
MPIF77 = tau_f77.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

```
% 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_f77.sh -c -O3 -g -openmp          bt.f
[...]
tau_f77.sh -c -O3 -g -openmp          mpi_setup.f
cd ..;/common; mpiifort -c -O3 -g -qopenmp      print_results.f
cd ..;/common; mpiifort -c -O3 -g -qopenmp      timers.f
tau_f77.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 6 threads should be reasonable
  - bt-mz\_C.8 should take around 45 seconds

# TAU Source Instrumentation

---

- Edit `config/make.def` to adjust build configuration
  - Uncomment specification of compiler/linker: `MPIF77 = tau_f77.sh` or use `make MPIF77=tau_f77.sh`
- Make clean and build new tool-specific executable

```
% make clean  
% make suite MPIF77=tau_f77.sh  
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/claix/tau.lsf .  
% bsub < tau.lsf
```

# NPB-MZ-MPI / BT with TAU

```
% cd bin  
% cp ..../jobscript/claiX/tau.lsf .  
% bsub < tau.lsf  
% cat mzmpibt.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: 48 ( 6.0 threads/process)  
  
Time step 1  
Time step 20  
[...]  
Time step 180  
Time step 200  
Verification Successful  
  
BT-MZ Benchmark Completed.  
Time in seconds = 45.88  
% 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 env **TAU\_SAMPLING=1** or tau\_exec **-ebs**

# NPB-MZ-MPI / BT with TAU

```
% cd bin  
% cp ..../jobscript/claix/tau_exec.lsf .  
% bsub < tau_exec.lsf  
% cat mzmpibt.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: 48 ( 6.0 threads/process)  
  
Time step 1  
Time step 20  
[...]  
Time step 180  
Time step 200  
Verification Successful  
  
BT-MZ Benchmark Completed.  
Time in seconds = 45.88  
% 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 Analysis Tools: paraprof

- Launch paraprof

```
% paraprof
```

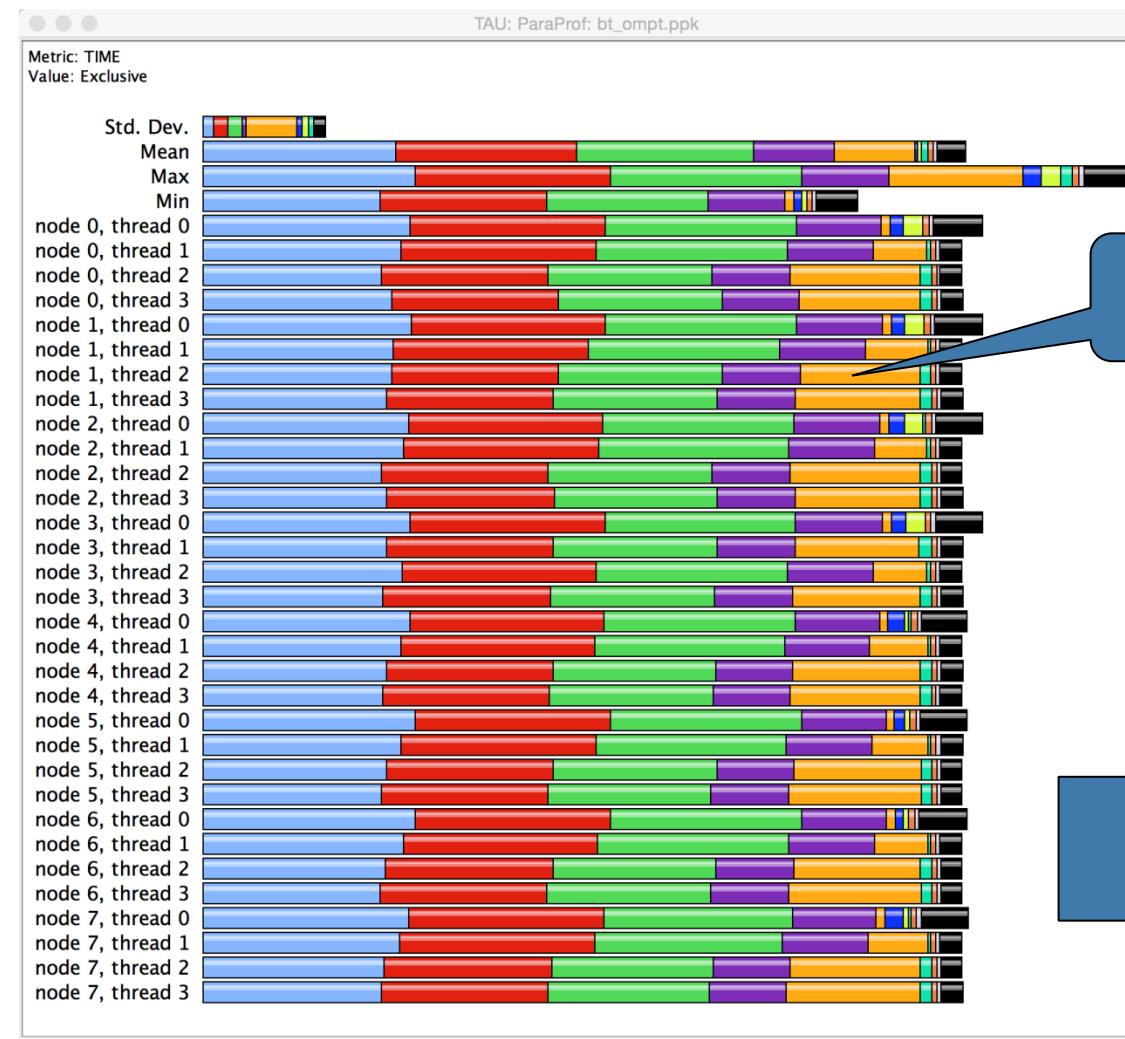
Metric

TAU: ParaProf Manager

The screenshot shows the TAU ParaProf Manager window. On the left, there is a tree view under the 'Applications' section. It includes 'Standard Applications', 'Default App' (which has 'Default Exp' and 'bt\_ompt.ppk' as children), and a connection to 'Default (jdbc:h2:/Users/sameer/.ParaProf/perfdmf/perfdmf;AUTO\_SERVER=TRUE)'. A blue arrow points from the 'Metric' button in the bottom-left towards the 'bt\_ompt.ppk' entry in the tree. On the right, there is a large table titled 'TAU: ParaProf Manager' with two columns: 'TrialField' and 'Value'. The table lists various system and application parameters.

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.b16.Bull.33.x86_64
OS Version	#1 SMP Sat Nov 10 01:48:00 CET 2012

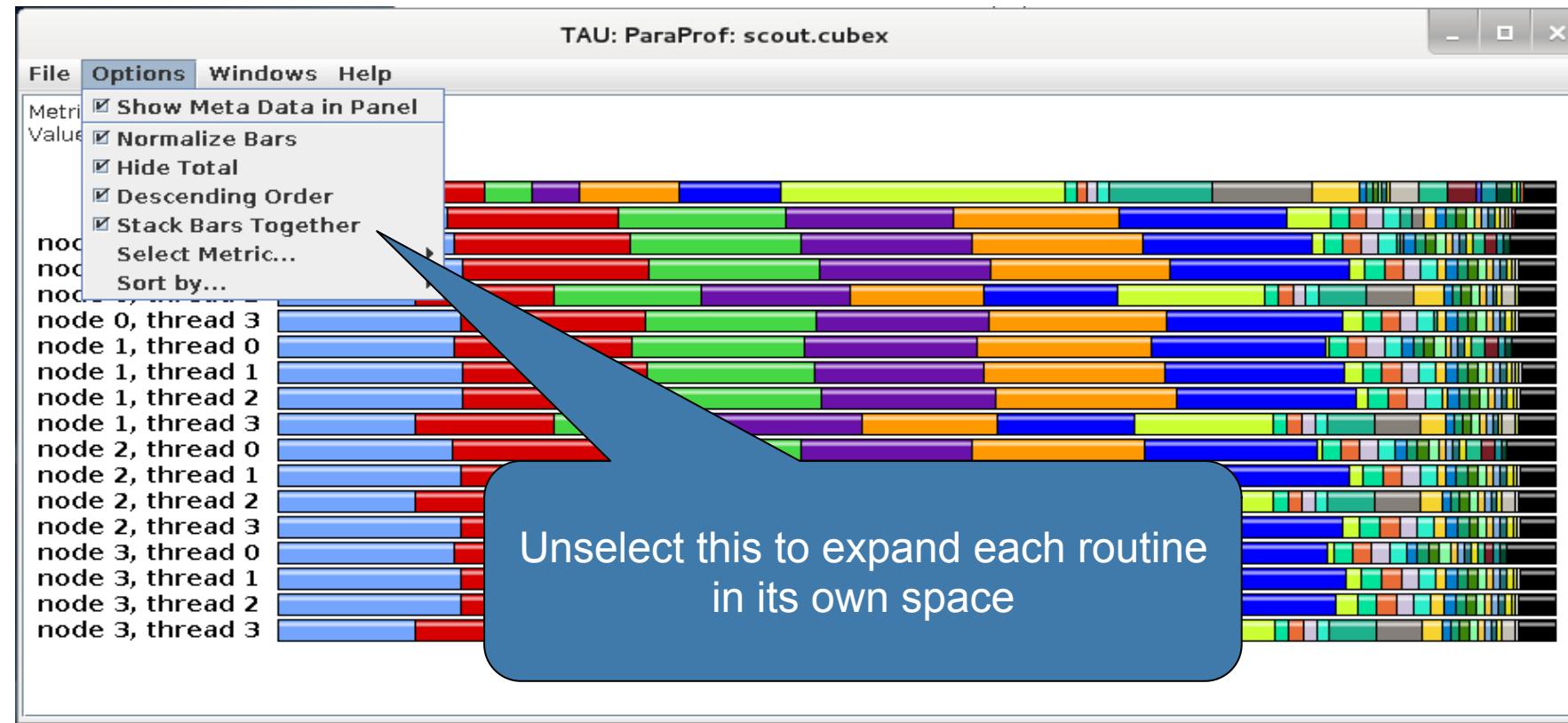
# Paraprof main window



Colors represent code regions

Options -> uncheck Stack Bars  
Together

# Paraprof main window



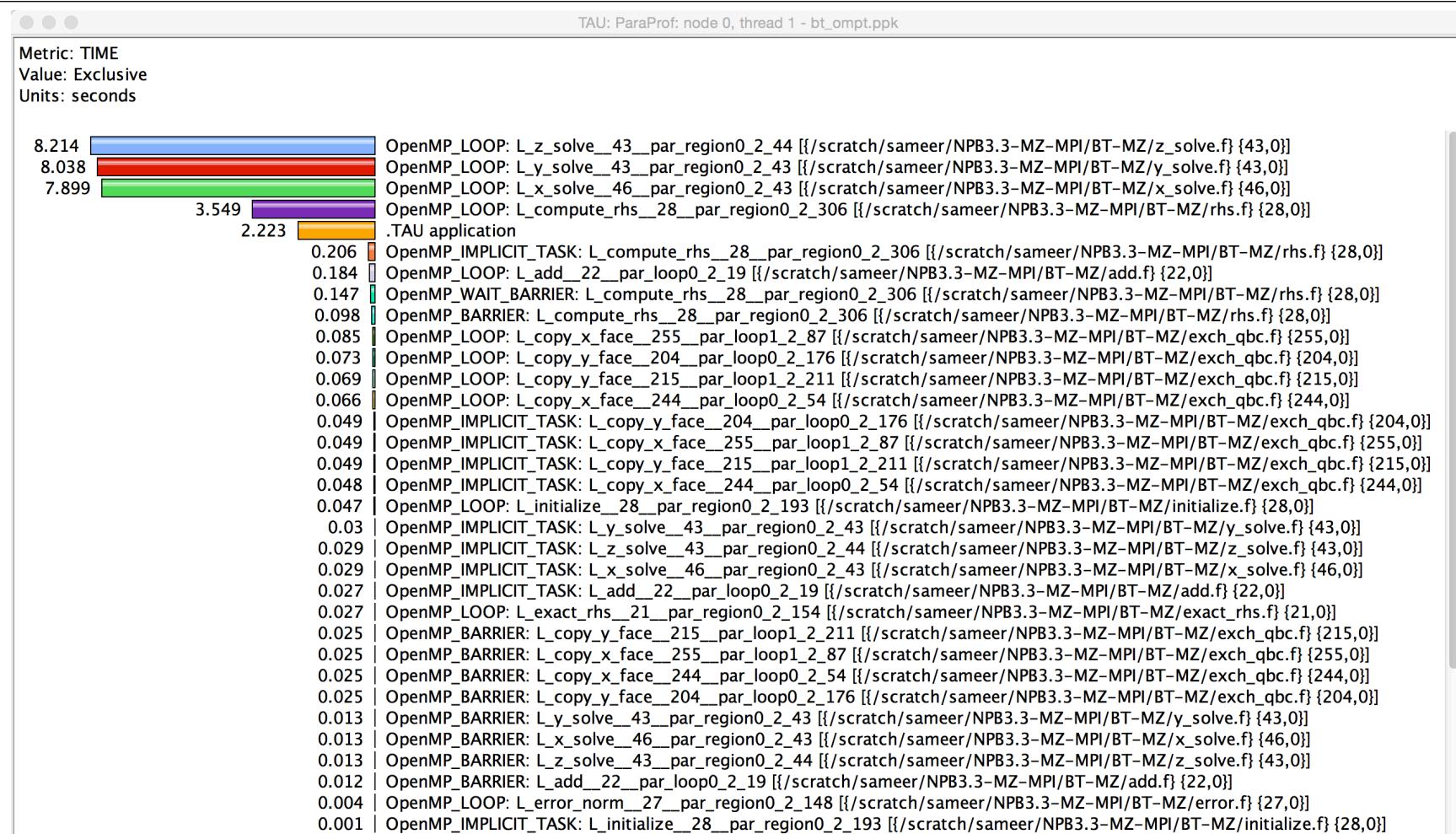
# Paraprof main window



Left/right  
click here

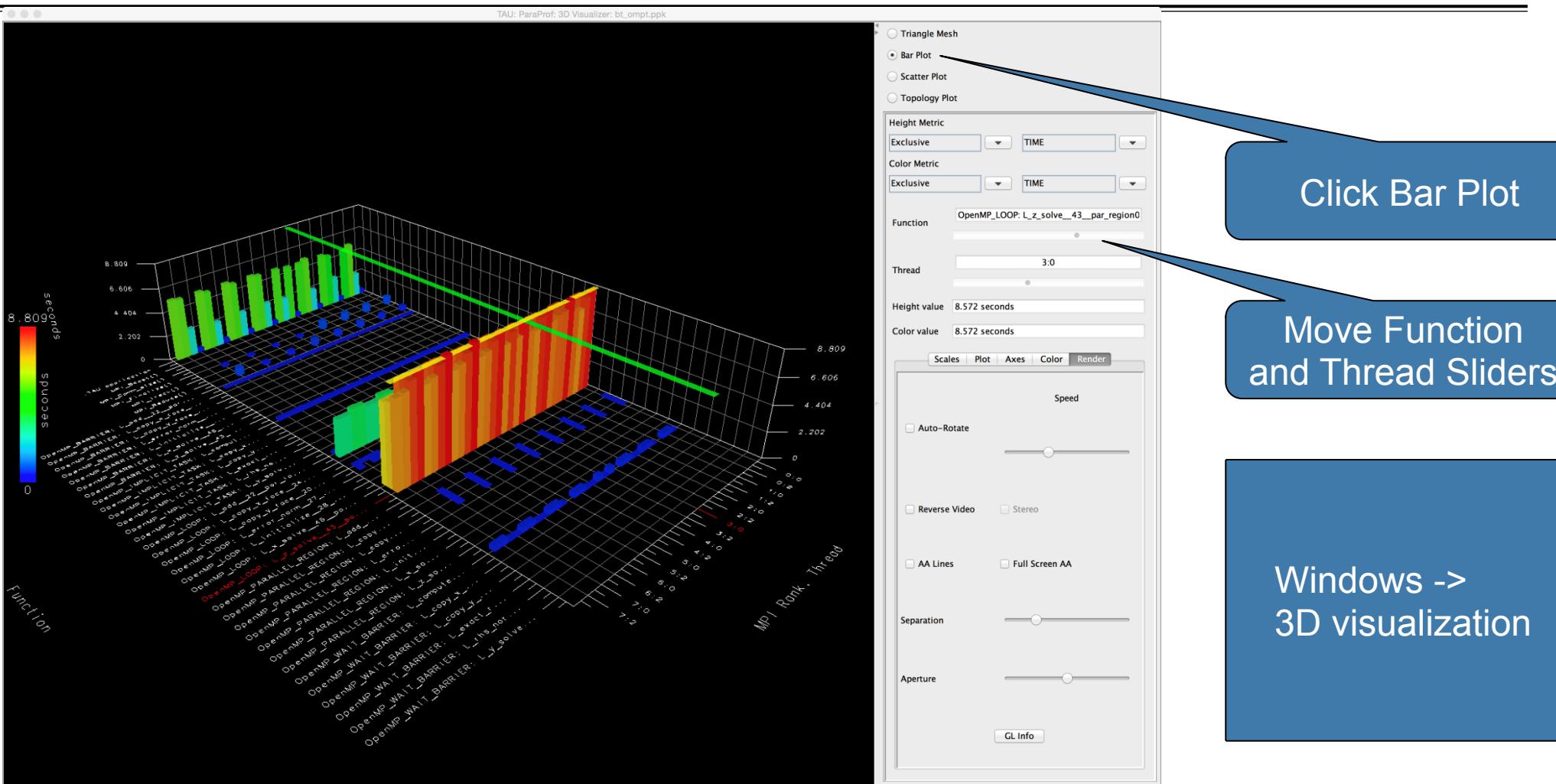
Each routine occupies its own space.  
Can see the extent of imbalance  
across all threads.

# Paraprof node window (function barchart window)

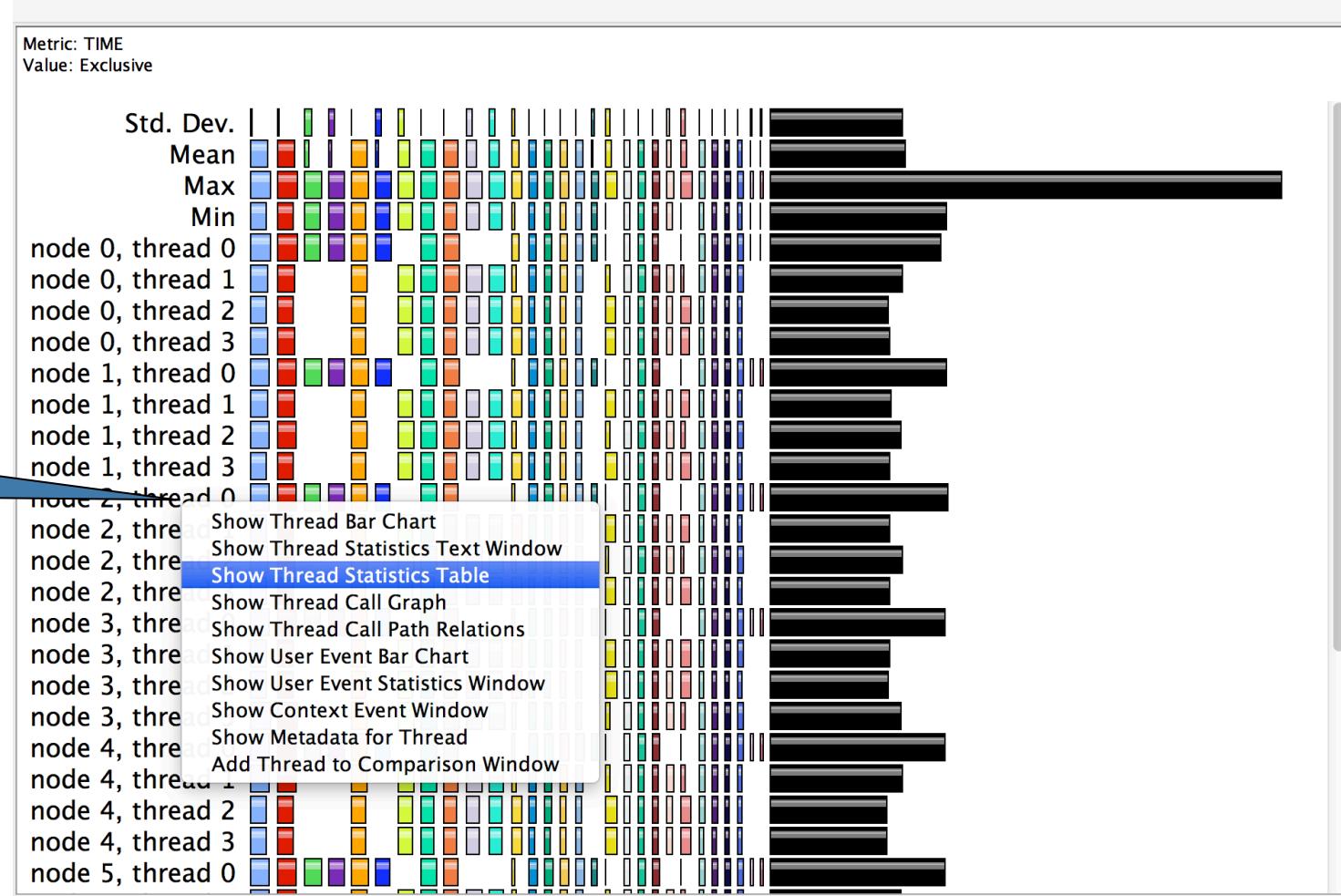


Exclusive time spent in each code region (OpenMP loop) is shown here for MPI rank 0 thread 1

# Paraprof 3D visualization window



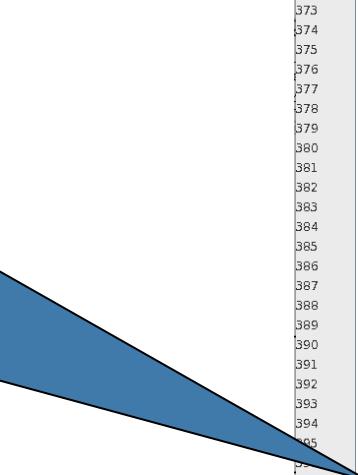
# Paraprof Thread Statistics Table with TAU\_SAMPLING=1



Right click here

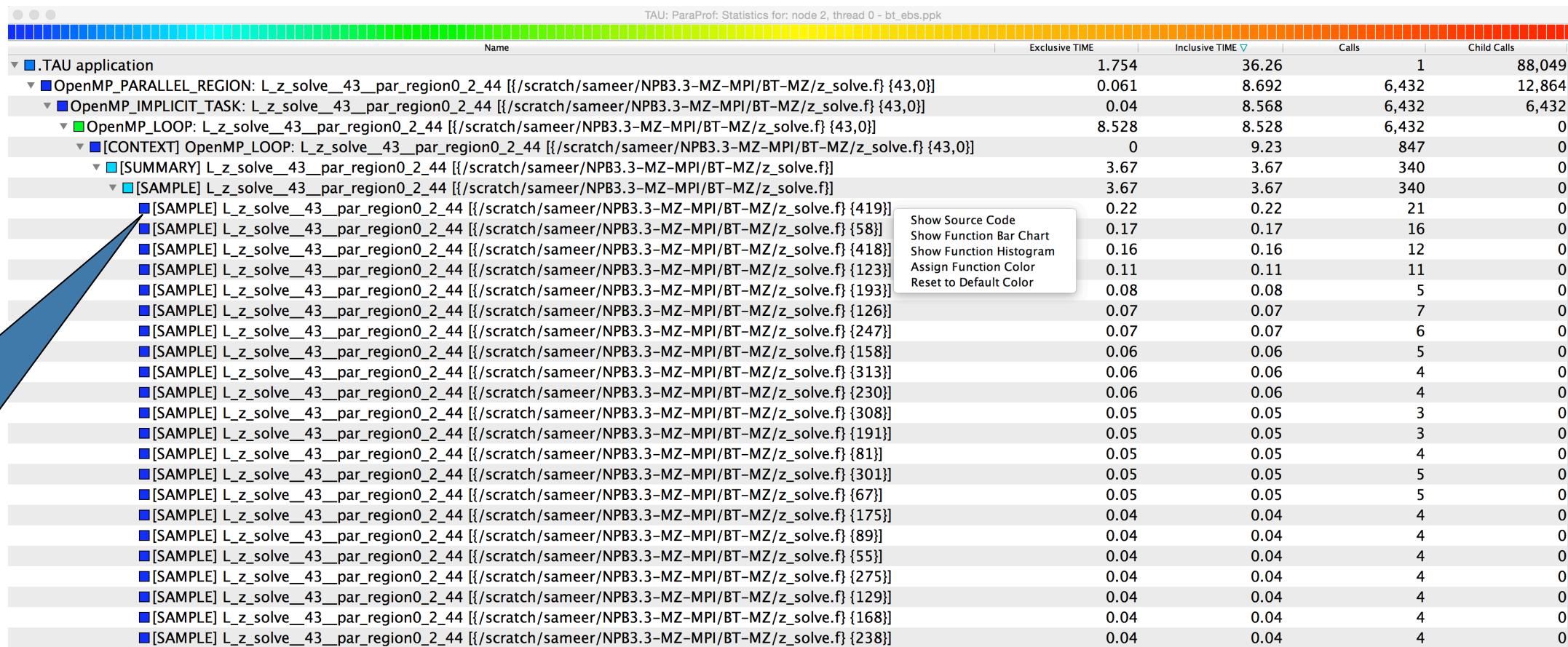
# Statement Level Profiling with TAU

Source location where samples are taken.  
Compute intensive region.



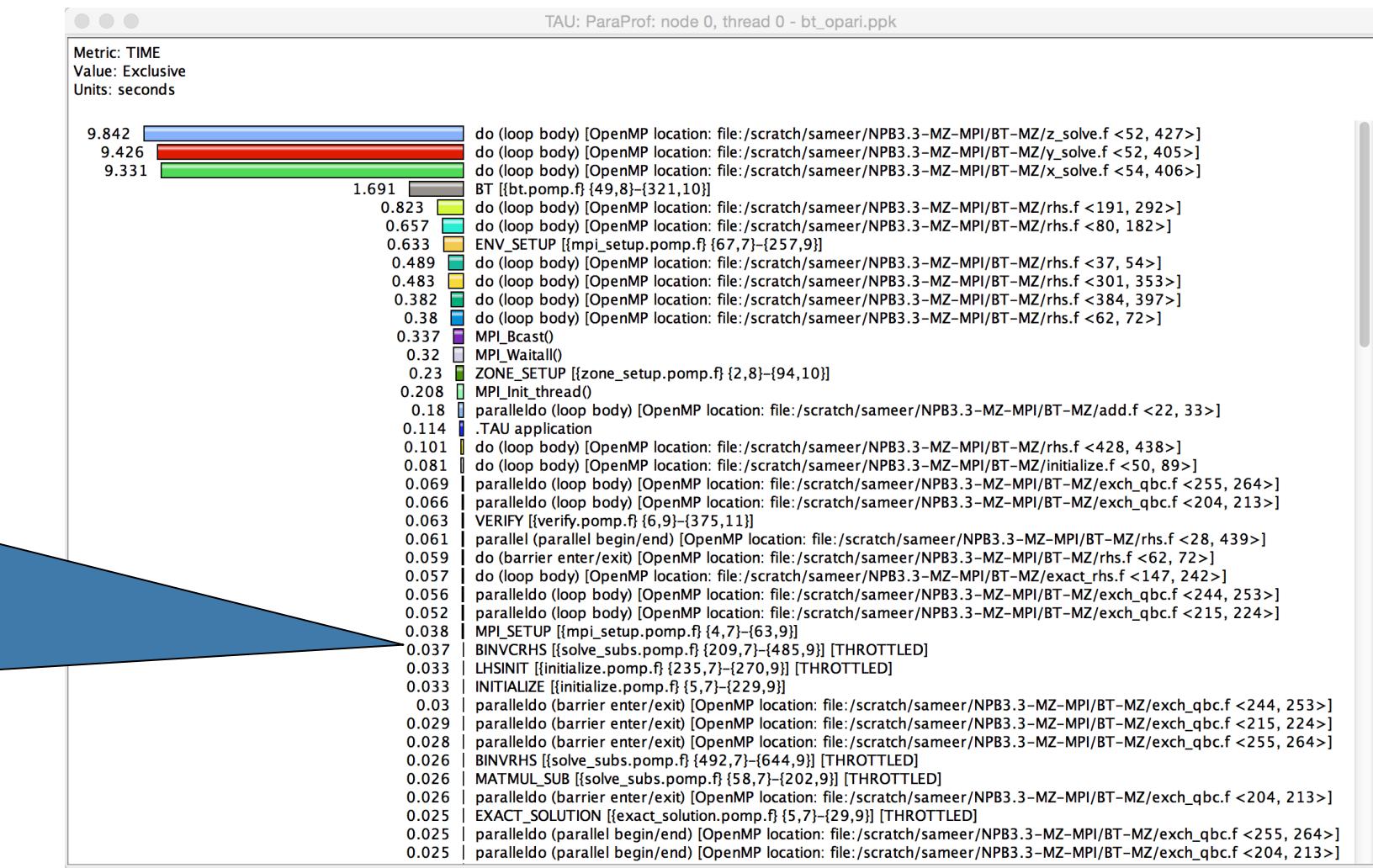
```
File Help
353     call matmul_sub(lhs(1,1,aa,i),
354     >           lhs(1,1,cc,i-1),
355     >           lhs(1,1,bb,i))
356
357
358 C-----c
359 c   multiply c(i,j,k) by b_inverse and copy back to c
360 c   multiply rhs(1,j,k) by b_inverse(1,j,k) and copy to rhs
361 C-----c
362     call binvcrhs( lhs(1,1,bb,i),
363     >           lhs(1,1,cc,i),
364     >           rhs(1,i,j,k) )
365
366     enddo
367
368 C-----c
369 c   rhs(isize) = rhs(isize) - A*rhs(isize-1)
370 C-----c
371     call matvec_sub(lhs(1,1,aa,isize),
372     >           rhs(1,isize-1,j,k),rhs(1,isize,j,k))
373
374 C-----c
375 c   B(isize) = B(isize) - C(isize-1)*A(isize)
376 C-----c
377     call matmul_sub(lhs(1,1,aa,isize),
378     >           lhs(1,1,cc,isize-1),
379     >           lhs(1,1,bb,isize))
380
381 C-----c
382 c   multiply rhs() by b_inverse() and copy to rhs
383 C-----c
384     call binvrhs( lhs(1,1,bb,isize),
385     >           rhs(1,isize,j,k) )
386
387 C-----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-----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           >           - lhs(m,n,cc,i)*rhs(n,i+1,j,k)
399         enddo
400       enddo
401     enddo
402
```

# Paraprof Thread Statistics Table

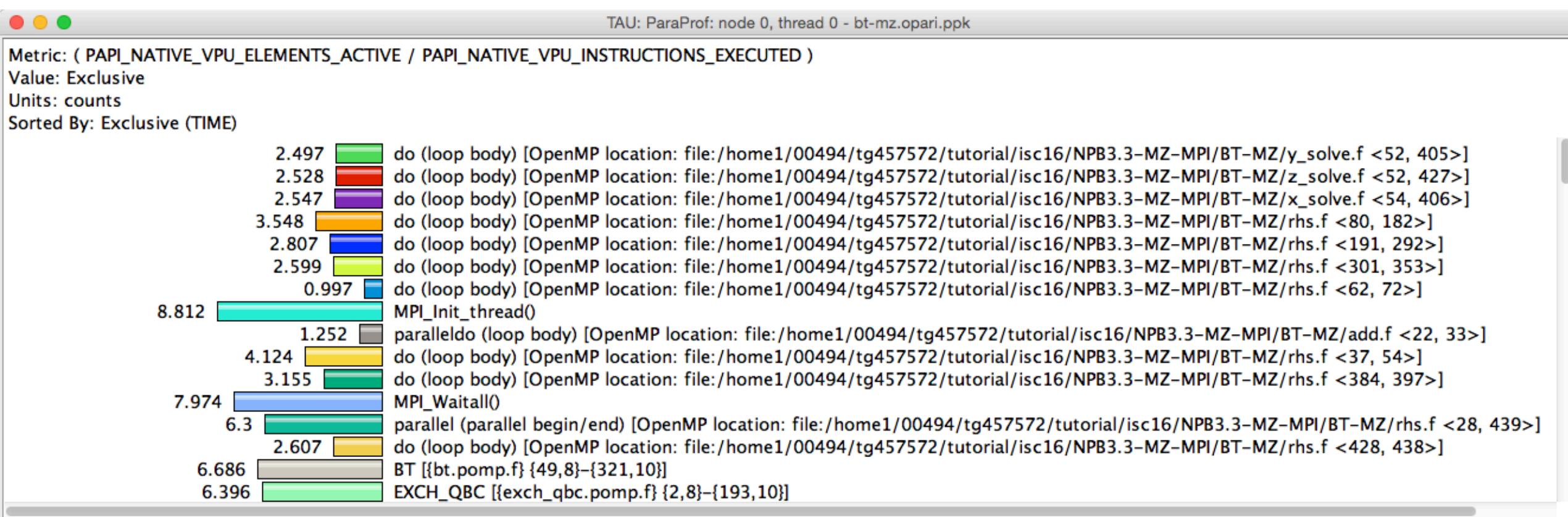


# Instrumenting Source Code with PDT and Opari

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

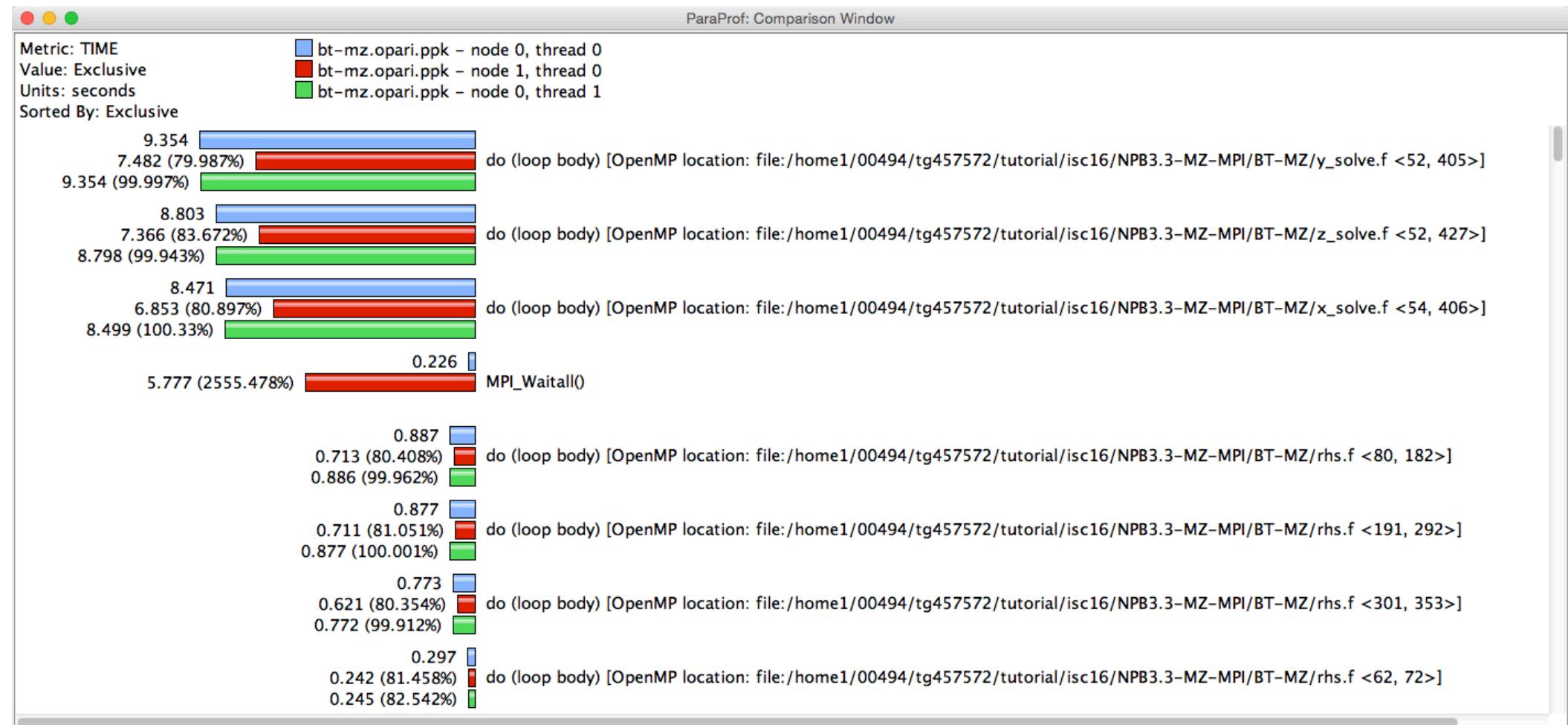


# ParaProf Derived Metric Window: Intel MIC Vectorization Intensity



```
% export TAU_MAKEFILE=$TAU/Makefile.tau-icpc-papi-mpi-pdt-openmp-opari
% export TAU_METRICS=TIME,PAPI_NATIVE_VPU_ELEMENTS_ACTIVE,PAPI_NATIVE_VPU_INSTRUCTIONS_EXECUTED
% export TAU_PROFILE_FORMAT="merged"
% scp claix:tauprofile.xml . ; paraprof tauprofile.xml [Options -> Show Derived Metric Panel ]
```

# ParaProf Comparison Window



# ParaProf Manager Widow: scout.cubex

TAU: ParaProf Manager

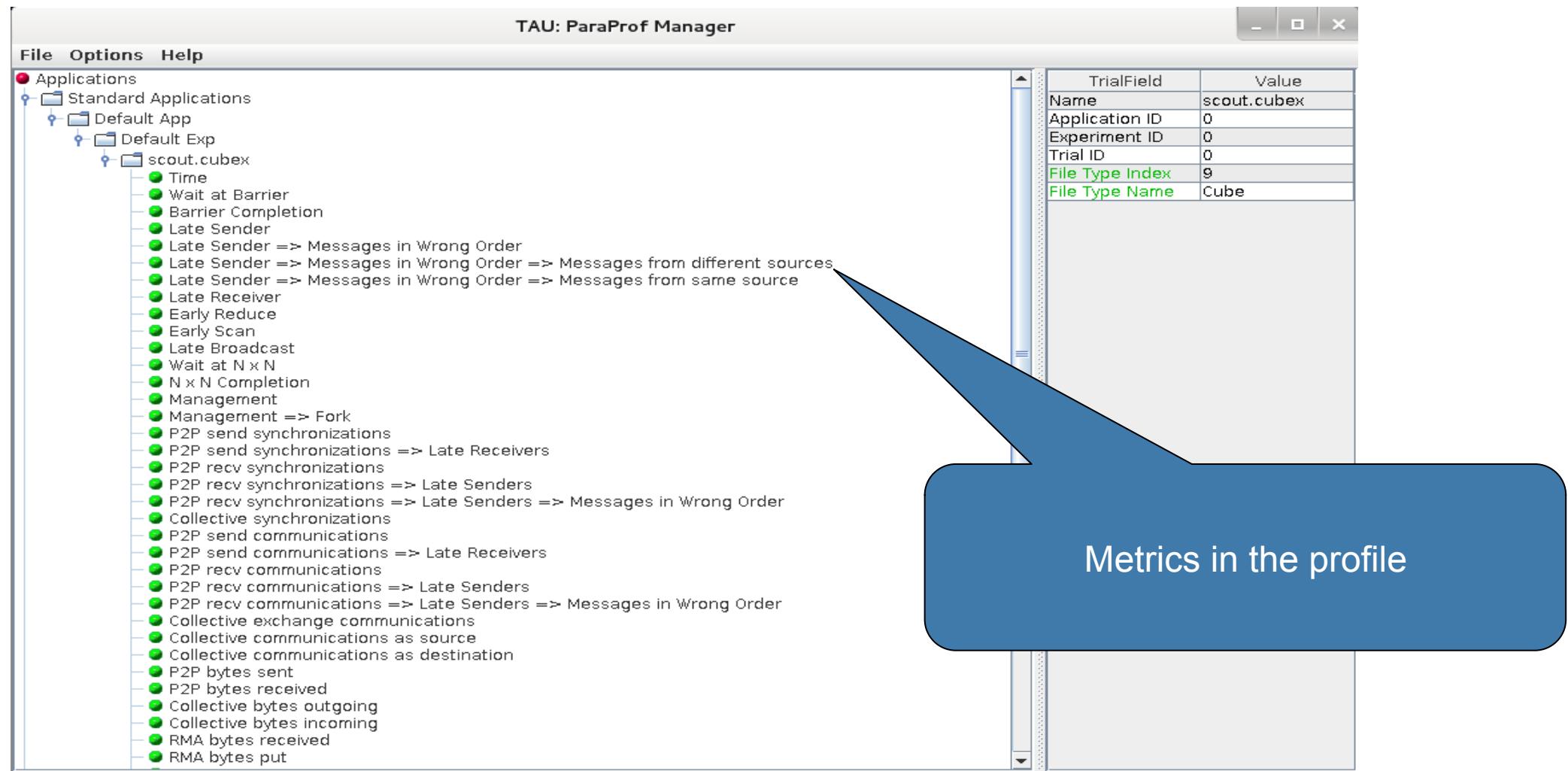
File Options Help

Applications

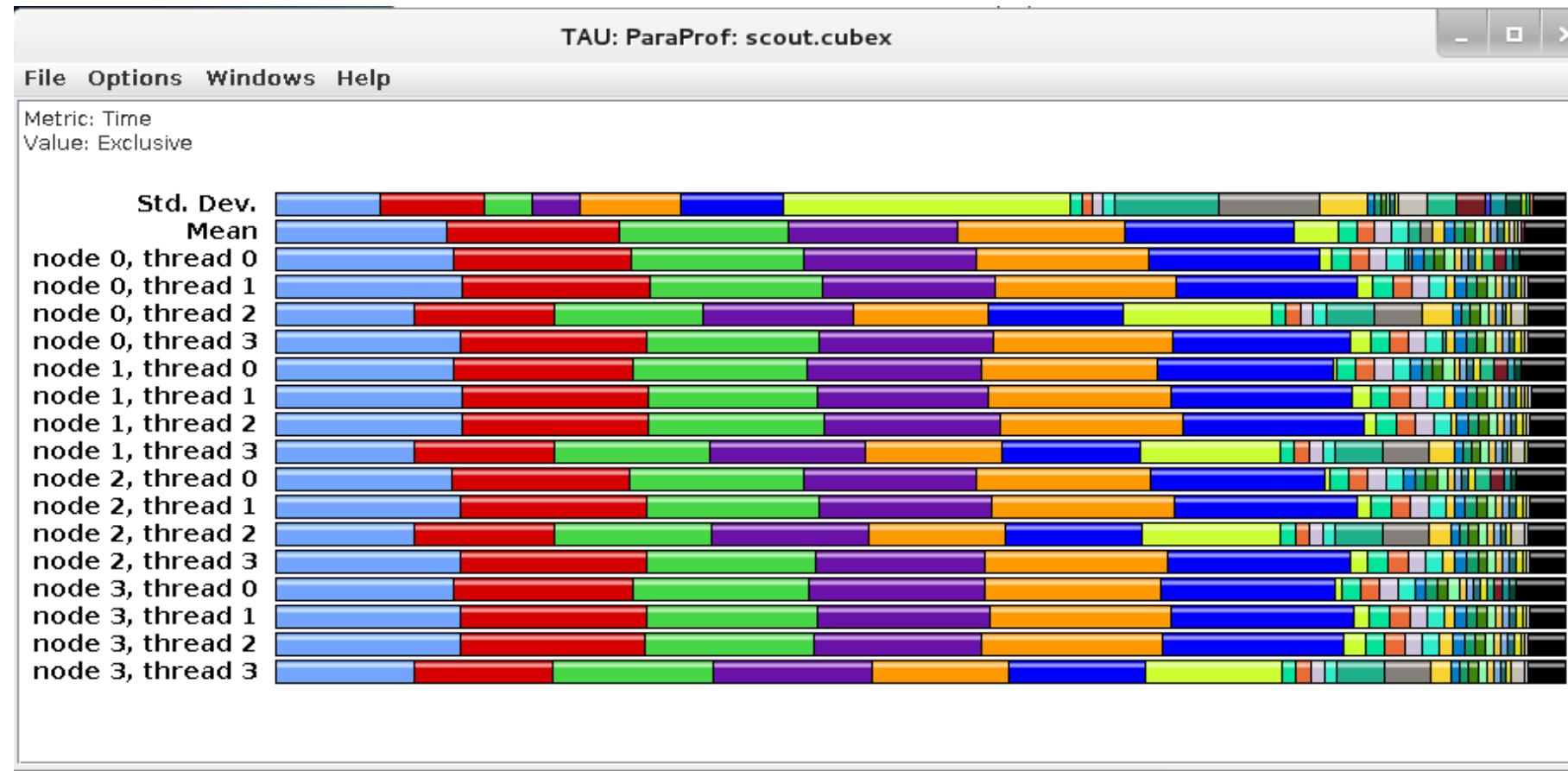
- Standard Applications
- Default App
- Default Exp
- scout.cubex
  - Time
  - Wait at Barrier
  - Barrier Completion
  - Late Sender
  - Late Sender => Messages in Wrong Order
  - Late Sender => Messages in Wrong Order => Messages from different sources
  - Late Sender => Messages in Wrong Order => 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 recv synchronizations
  - P2P recv synchronizations => Late Senders
  - P2P recv synchronizations => Late Senders => Messages in Wrong Order
  - Collective synchronizations
  - P2P send communications
  - P2P send communications => Late Receivers
  - P2P recv communications
  - P2P recv communications => Late Senders
  - P2P recv 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

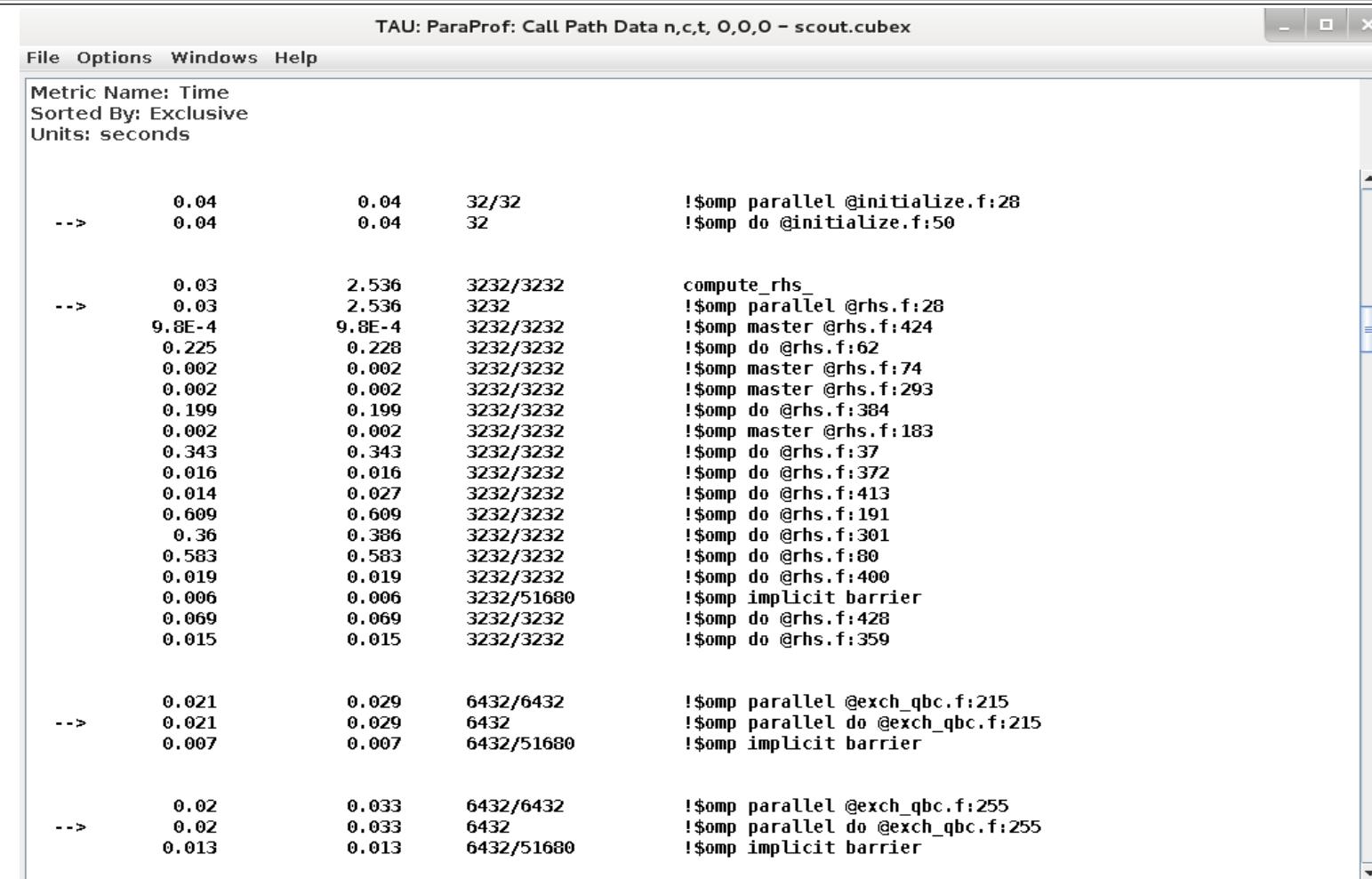
Time

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	0	0	0	0

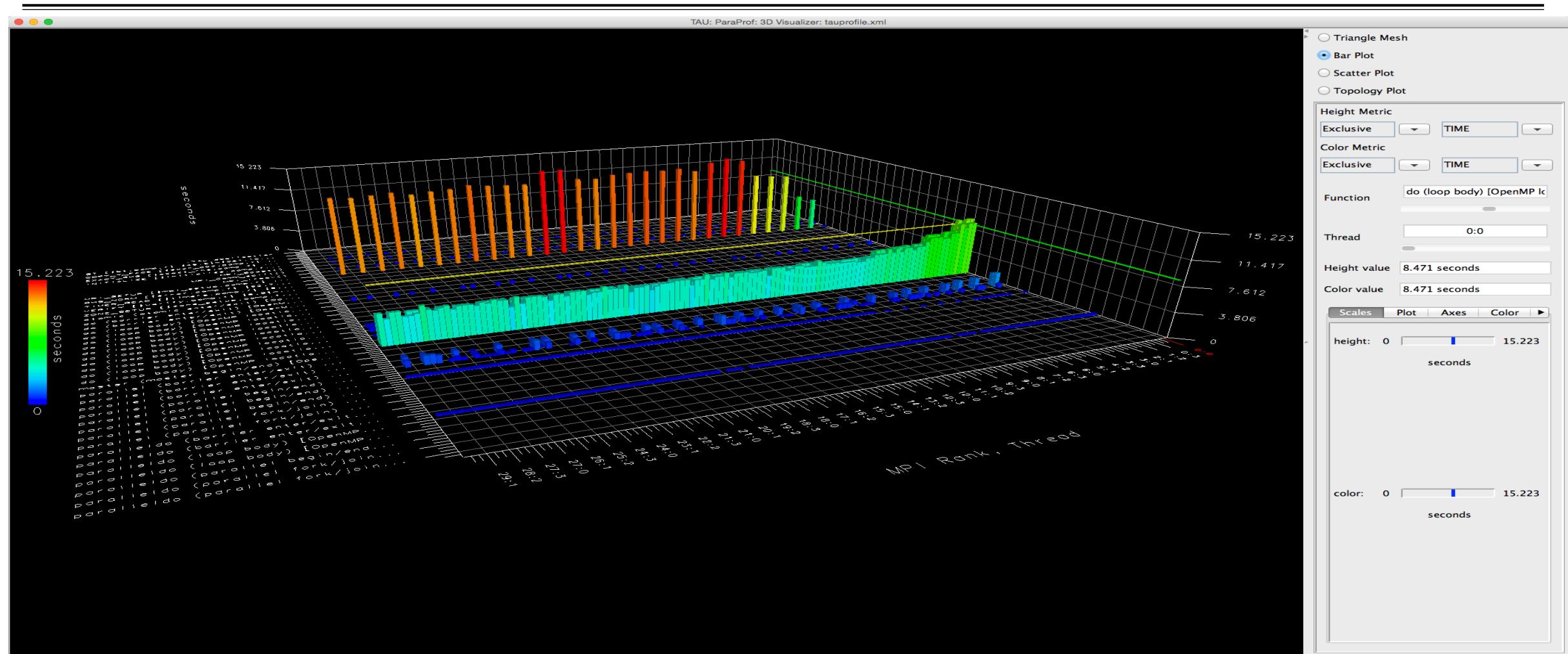
Click to sort by a given metric, drag and move to rearrange columns

FinderScreenSnapz003.png

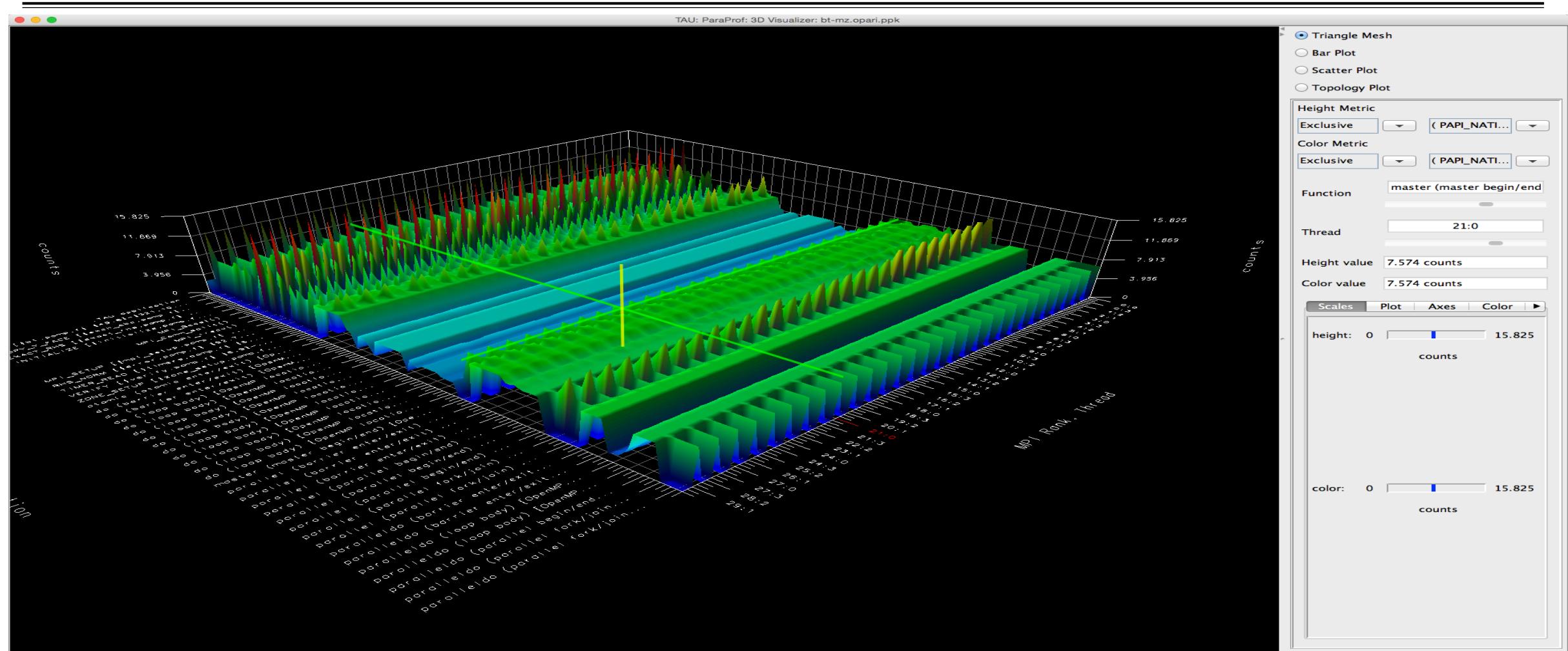
# ParaProf: Callpath Thread Relations Window



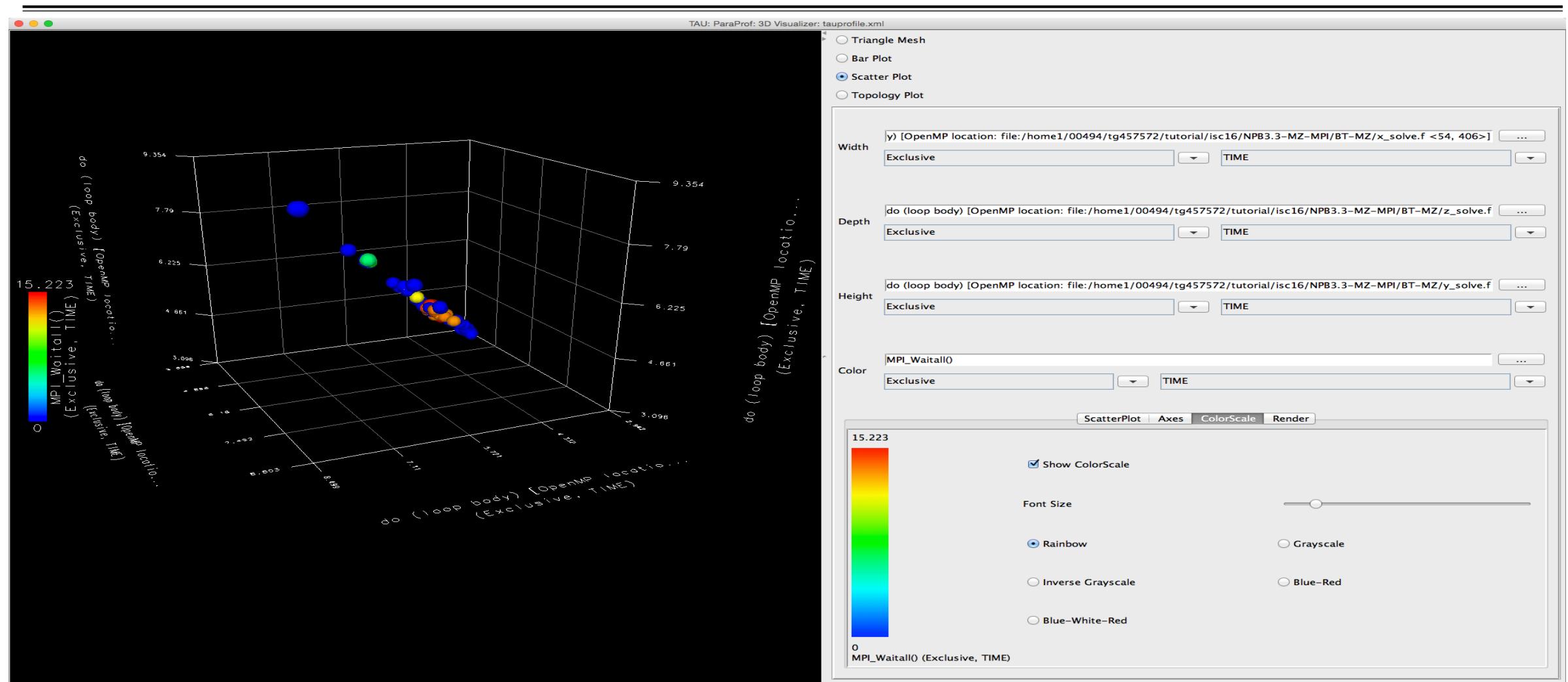
# ParaProf: 3D Visualization Window Showing Entire Profile



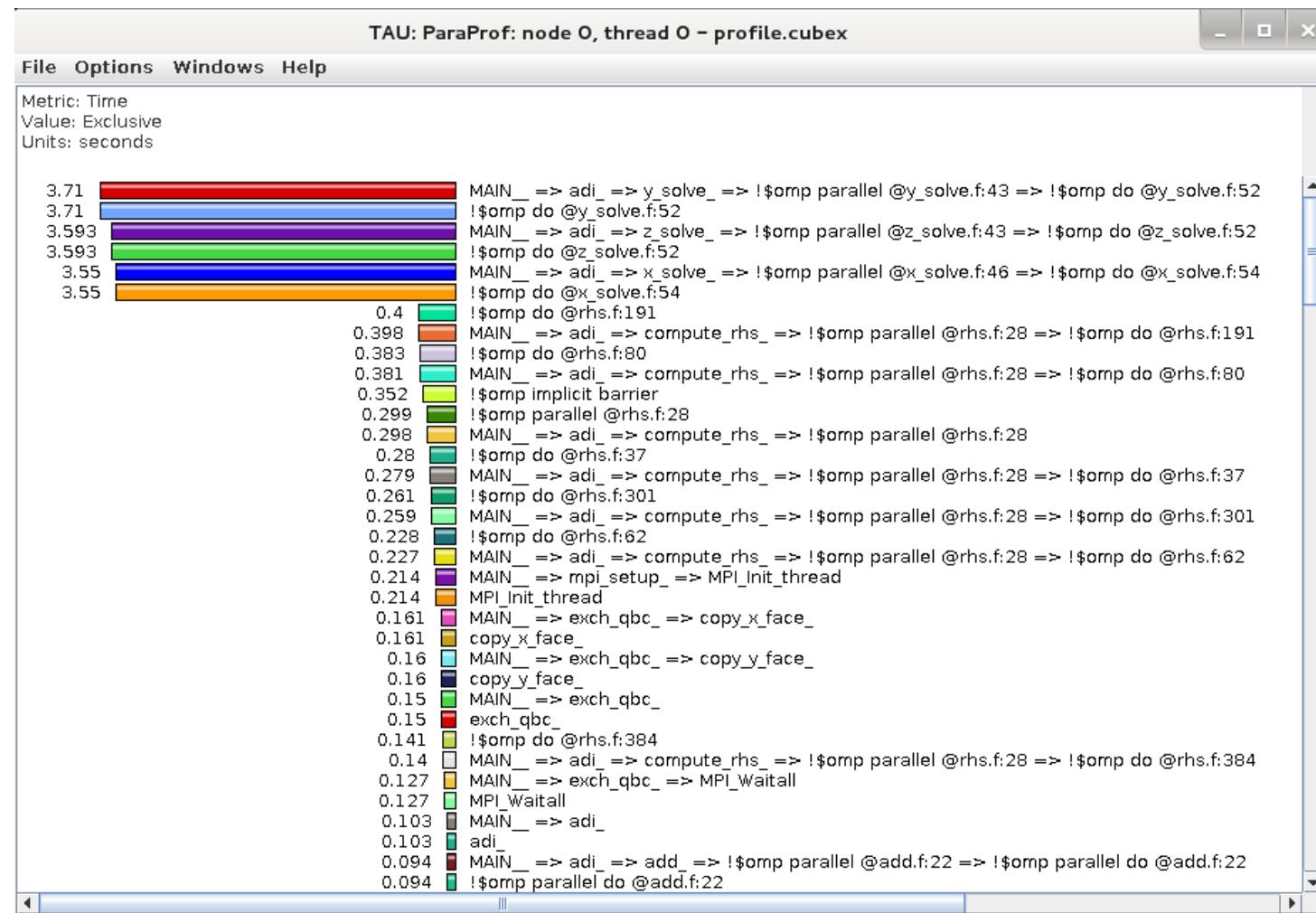
# ParaProf: 3D Visualization Window MIC Vectorization Intensity



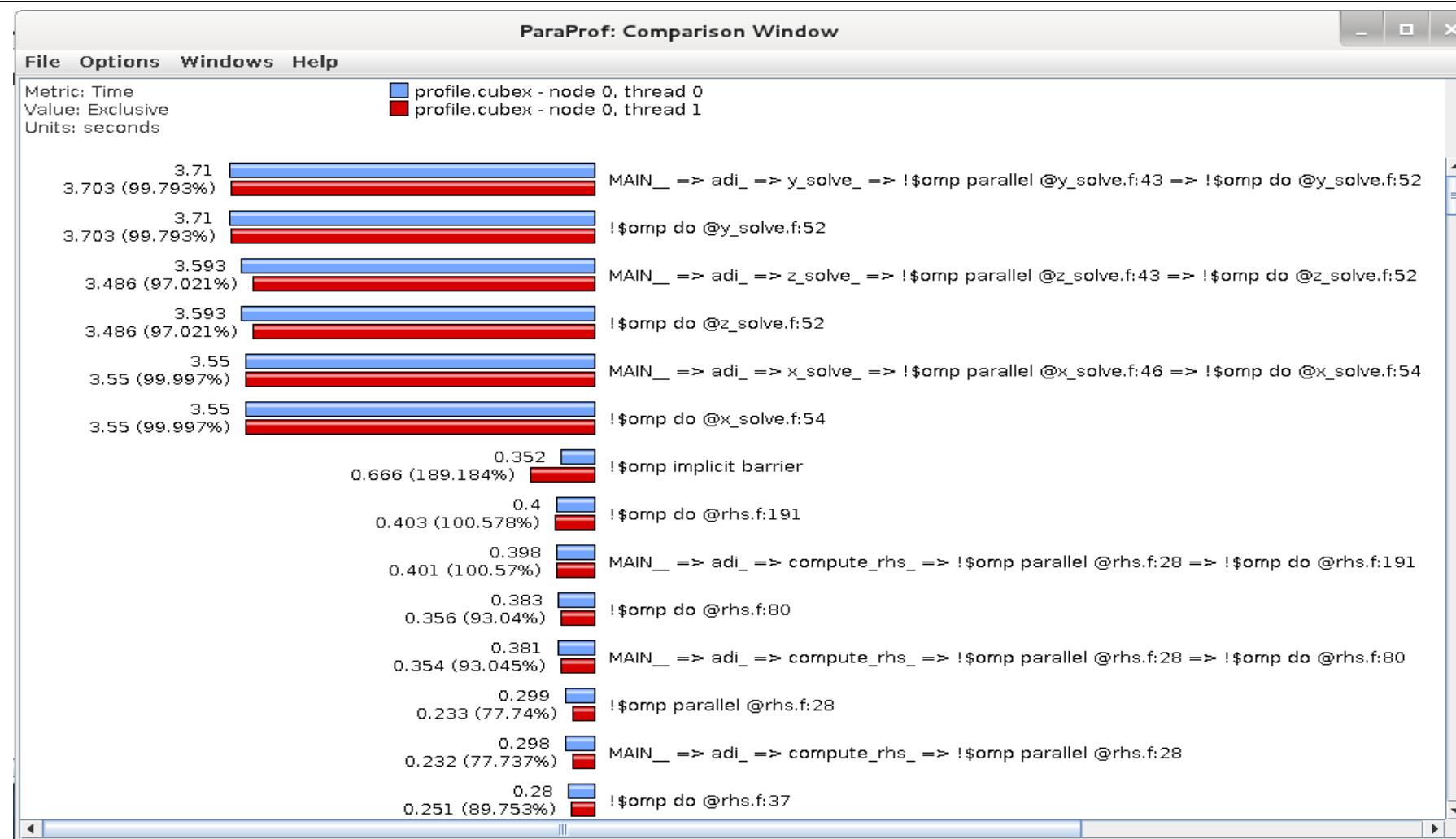
# ParaProf: 3D Scatter Plot



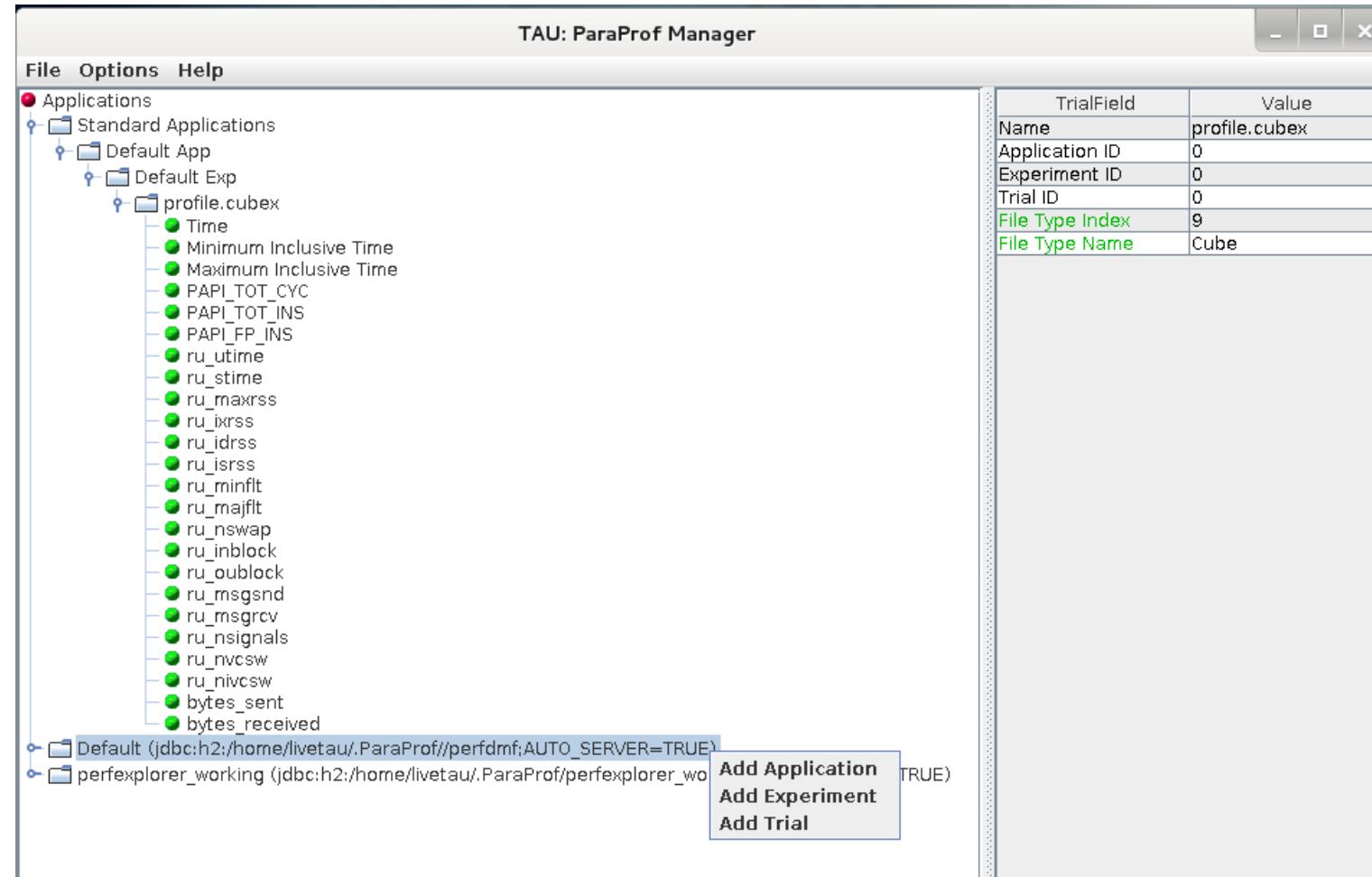
# ParaProf: Node View



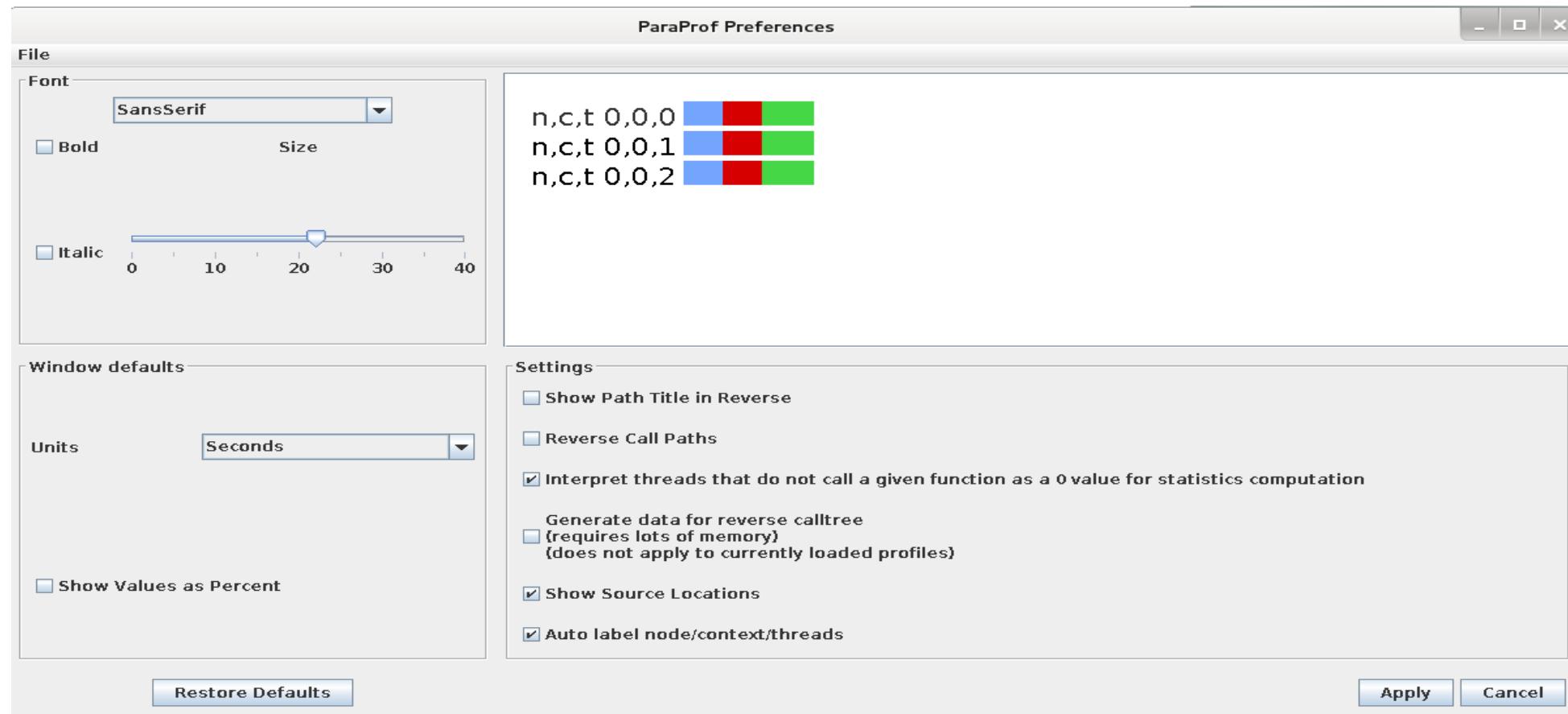
# ParaProf: Add Thread to Comparison Window



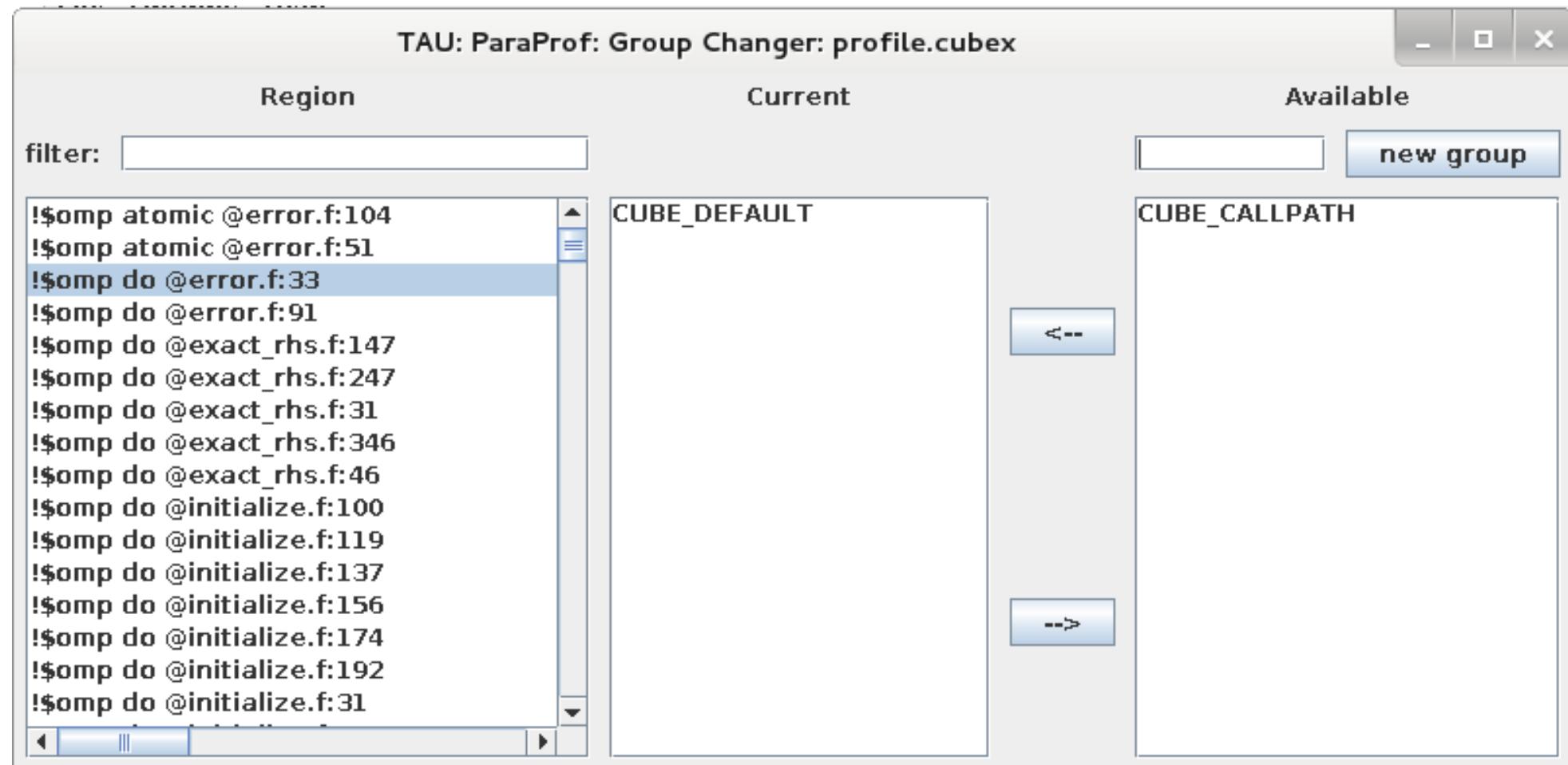
# ParaProf: Score-P Profile Files, Database



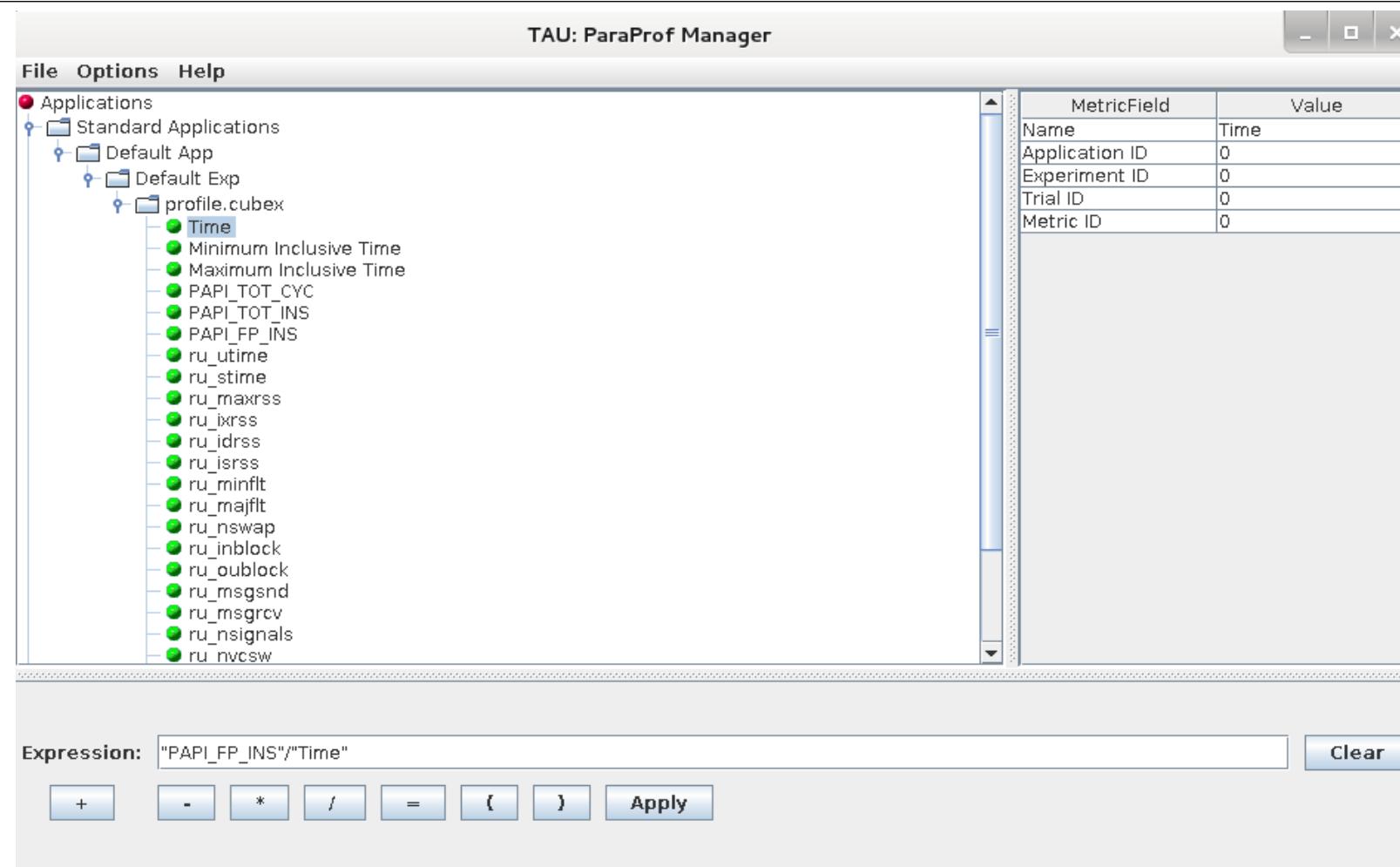
# ParaProf: File Preferences Window



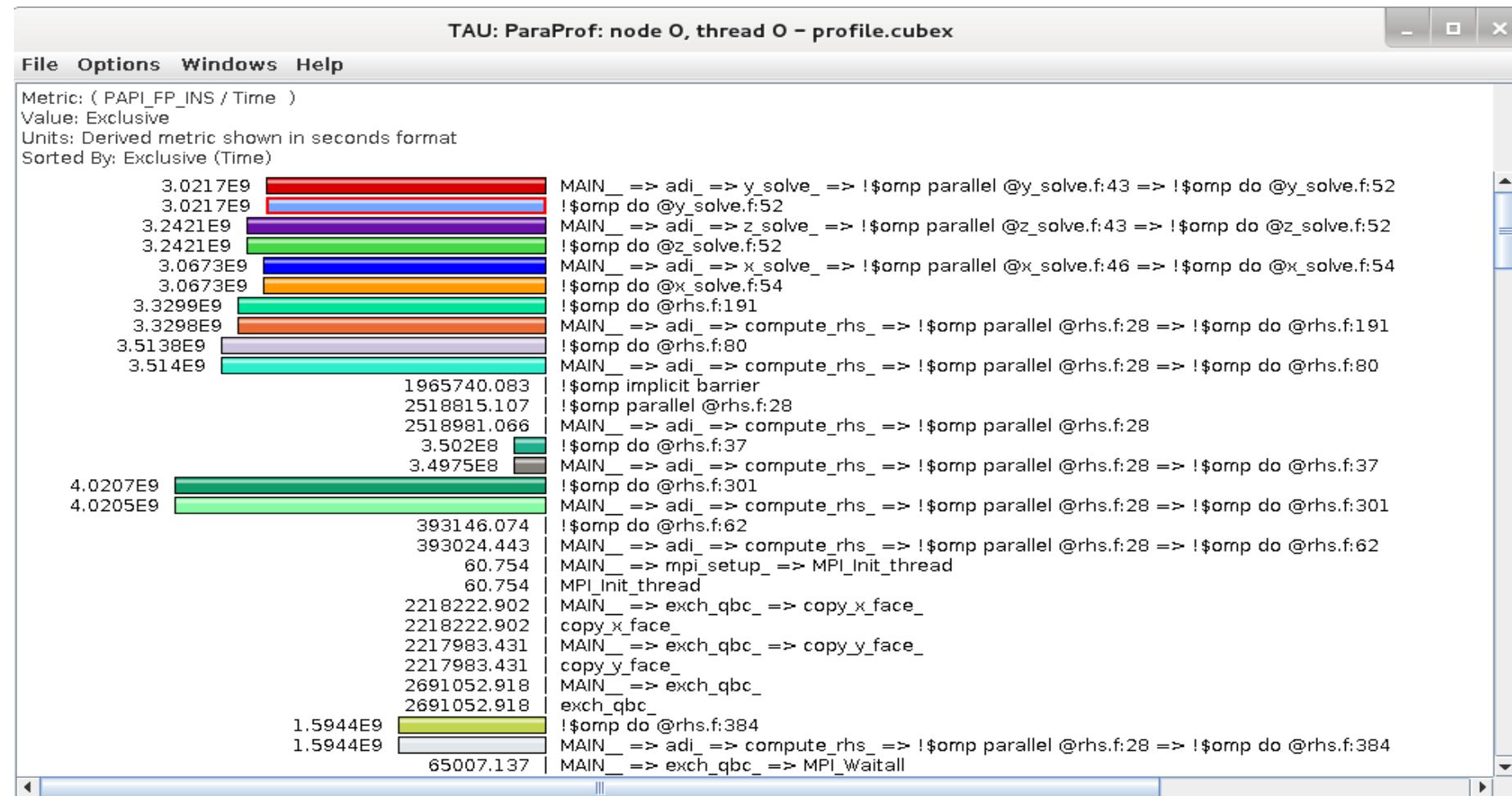
# ParaProf: Group Changer Window



# ParaProf: Derived Metric Panel in Manager Window

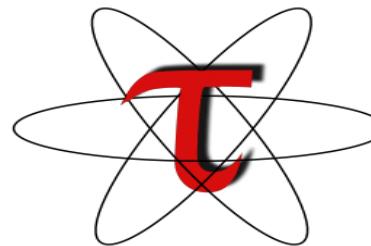


# Sorting Derived FLOPS metric by Exclusive Time



## Download TAU from U. Oregon

---



<http://tau.uoregon.edu>

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

**Free download, open source, BSD license**