
# 2020 Virtual ALCF Computational Performance Workshop

## TAU Performance System

Sameer Shende
Director, Performance Research Laboratory, University of Oregon

May 6, 2020 11am – 12:00pm

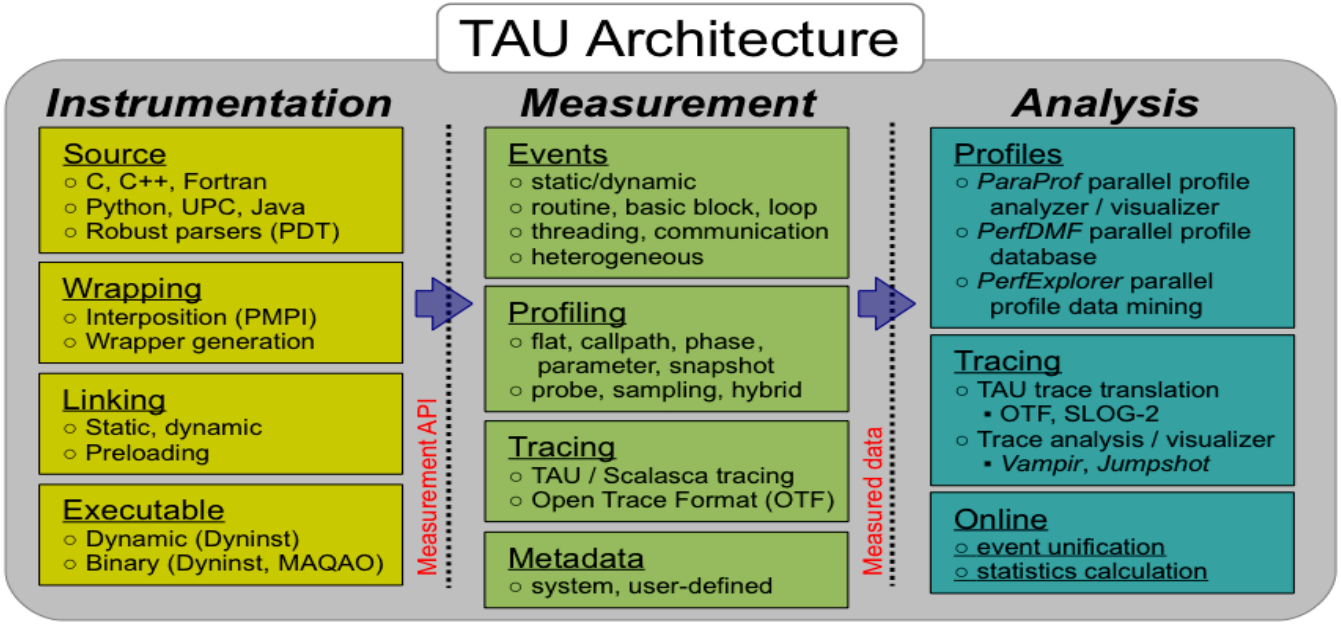
Slides are also available at:

http://tau.uoregon.edu/tau_alcf20.pdf




# 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, ROCm, OpenACC
  - Parallel profiling and tracing
- Analysis
  - Parallel profile analysis (ParaProf), data mining (PerfExplorer)
  - Performance database technology (TAUdb)
  - 3D profile browser

# 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?

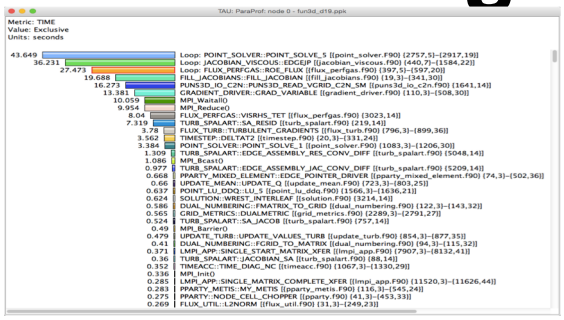
# Instrumentation

## Add hooks in the code to perform measurements

- **Source instrumentation using a preprocessor**
  - Add timer start/stop calls in a copy of the source code.
  - Use Program Database Toolkit (PDT) for parsing source code.
  - Requires recompiling the code using TAU shell scripts (tau_cc.sh, tau_f90.sh)
  - Selective instrumentation (filter file) can reduce runtime overhead and narrow instrumentation focus.
- **Compiler-based instrumentation**
  - Use system compiler to add a special flag to insert hooks at routine entry/exit.
  - Requires recompiling using TAU compiler scripts (tau_cc.sh, tau_f90.sh…)
- **Runtime preloading of TAU's Dynamic Shared Object (DSO)**
  - No need to recompile code! Use **aprun tau_exec ./app** with options.
  - Requires dynamic executable (link using **–dynamic** on Theta).

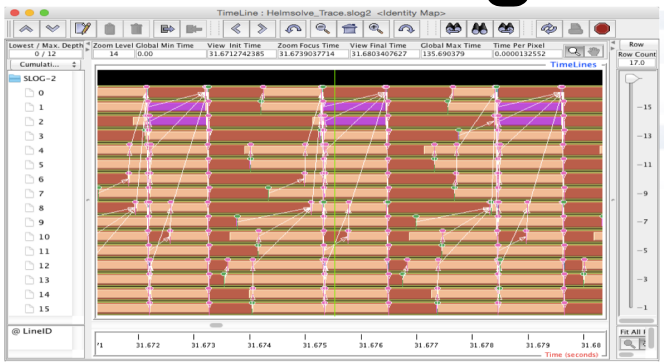
# Profiling and Tracing

## Profiling



- **Profiling** shows you **how much** (total) time was spent in each routine

## Tracing



- Tracing shows you when the events take place on a timeline

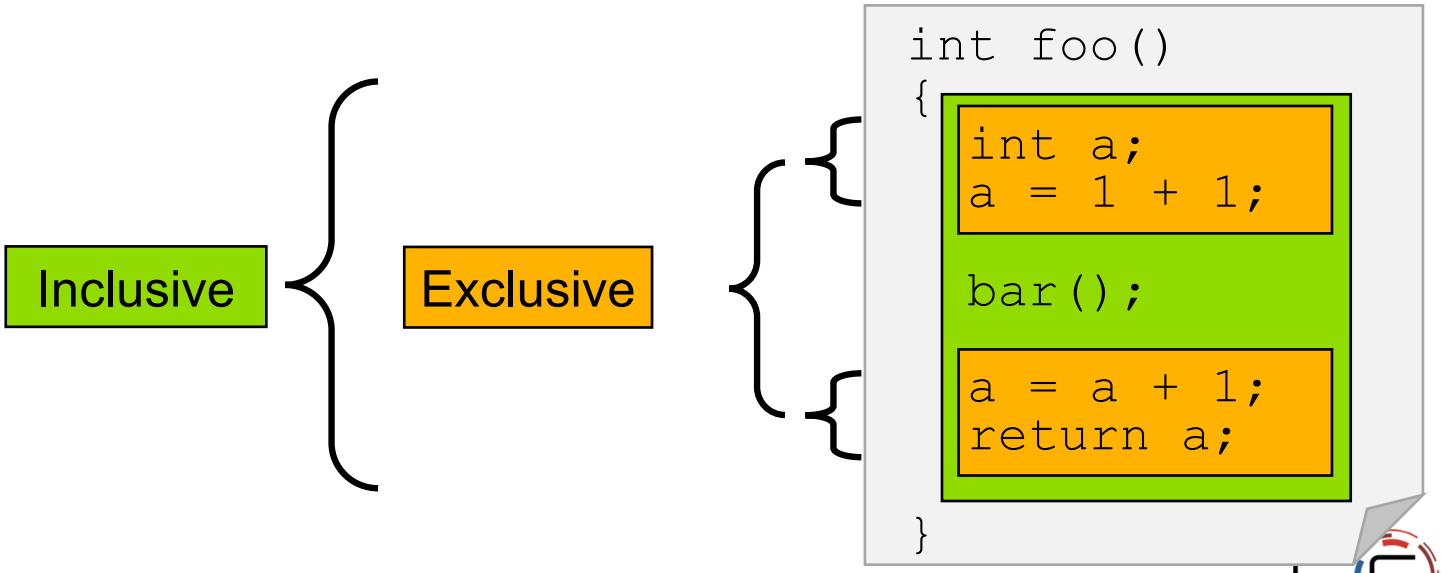
- Profiling and tracing

  **Profiling** shows you **how much** (total) time was spent in each routine

  **Tracing** shows you **when** the events take place on a timeline

# Inclusive vs. Exclusive values

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



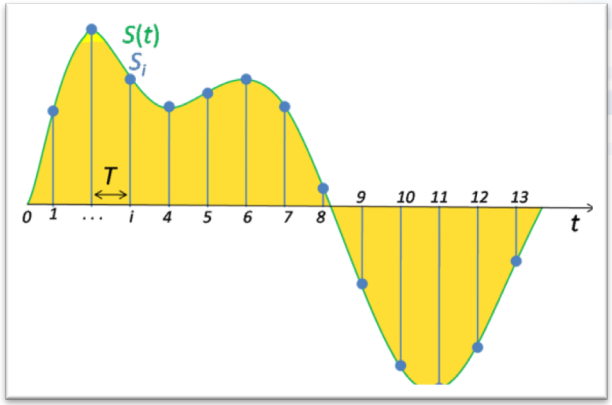
# Performance Data Measurement

## Direct via Probes

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

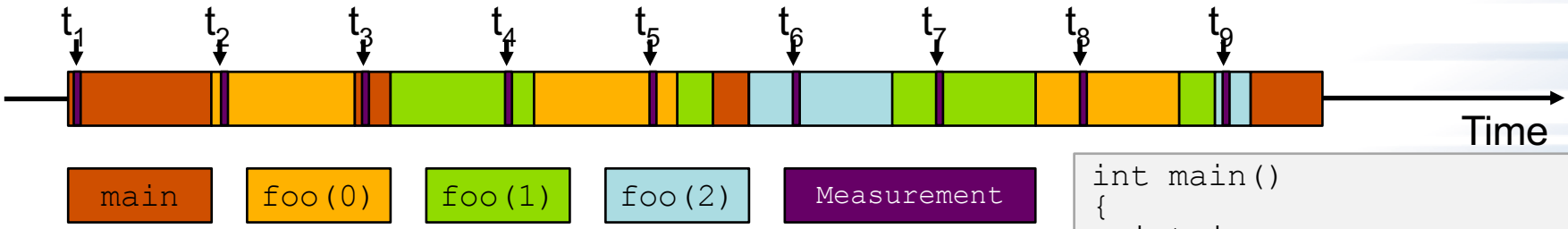
- Exact measurement
- Fine-grain control
- Calls inserted into code

## Indirect via Sampling




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

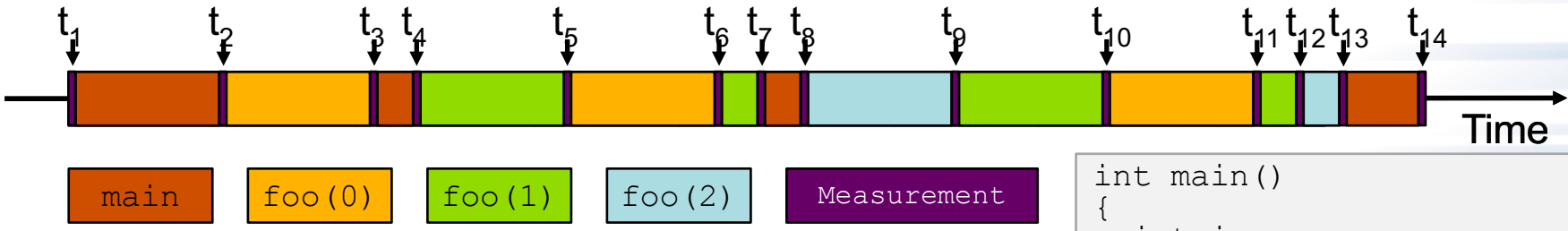
# Sampling




- 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




```
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");
}
```

- 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

# 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)

# TAU's support for OpenMP Tools Interface

- OpenMP Tools Interface (OMPT)
  - Replacement LLVM/Intel OpenMP library
    - TR4 – Intel 18
    - TR6
  - OMPT support without preloading any wrapper interposition library
    - TR7 – Intel 19.0.1
    - V5 - Intel 19.0.3+


EXASCALE COMPUTING PROJECT

# Demo!




U.S. DEPARTMENT OF
ENERGY
Office of
Science


National Nuclear Security Administration

# Simplifying TAU's usage (tau_exec)

- Uninstrumented execution
  - % aprun -n 64 ./a.out
- Track MPI performance
  - % aprun -n 64 tau_exec ./a.out
- Use event based sampling (compile with –g)
  - % aprun -n 64 tau_exec –ebs ./a.out
  - Also –ebs_source=<PAPI_COUNTER> -ebs_period=<overflow_count> -ebs_resolution=[file|function|line] (line is default)
- Track POSIX I/O and MPI performance (MPI enabled by default)
  - % aprun -n 64 tau_exec –T mpi,pdt,papi –io ./a.out
- Track OpenMP runtime routines
  - % aprun –n 64 tau_exec –T ompt,v5,pdt,mpi –ompt ./a.out
- Track memory operations
  - % export TAU_TRACK_MEMORY_LEAKS=1
  - % aprun -n 64 tau_exec –memory_debug ./a.out (bounds check)
- Load wrapper interposition library
  - % aprun –n 64 tau_exec –loadlib=<path/libwrapper.so> ./a.out

# RUNTIME PRELOADING

- Injects TAU DSO in the executing application
- Requires dynamic executables
- We must compile with –dynamic –g
- Use tau_exec while launching the application

# Copy the workshop tarball

- Setup preferred program environment compilers
  - Default set Intel Compilers with Intel MPI. You must compile with **–dynamic -g**

```
% module load tau;
% tar zxf /soft/perftools/tau/workshop.tgz
% cd workshop/MZ-NPB3.3-MPI; cat README
% make clean
% make suite
% cd bin
In a second window:
% qsub -I -n 1 -A <Project_ID> -q training -t 50
% cd bin; module load tau
% export OMP_NUM_THREADS=4
% aprun -n 16 ./bt-mz.B.16
% export TAU_OMPT_SUPPORT_LEVEL=full; export TAU_OMPT_RESOLVE_ADDRESS_EAGERLY=1
% aprun -n 16 tau_exec -T ompt,v5,mpi,pdt -ompt ./bt-mz.B.16
% paraprof --pack ex1.ppk
In the first window:
% paraprof ex1.ppk &
```

# 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
- 

```
% 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 (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
  - 16 processes each with 4 threads should be reasonable

  - bt-mz.B.16 should take around 1 minute

# 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 = -qopenmp -extend-source        # Intel compiler

...
#---------------------------------------------------------------------
# The Fortran compiler used for MPI programs
#---------------------------------------------------------------------

F77 = ftn # Intel compiler

# Alternative variant to perform instrumentation

...
```

Default (no instrumentation)

# 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
- make suite

# TAU Source Instrumentation

- Edit config/make.def to adjust build configuration
  - Uncomment specification of compiler/linker: F77 = tau_f77.sh or use make F77=tau_f77.sh
- Make clean and build new tool-specific executable
- Change to the directory containing the new executable before running it with the desired tool configuration

# tau_exec

```
$ tau_exec

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

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

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

- Tau_exec preloads the TAU wrapper libraries and performs measurements.

No need to recompile the application!

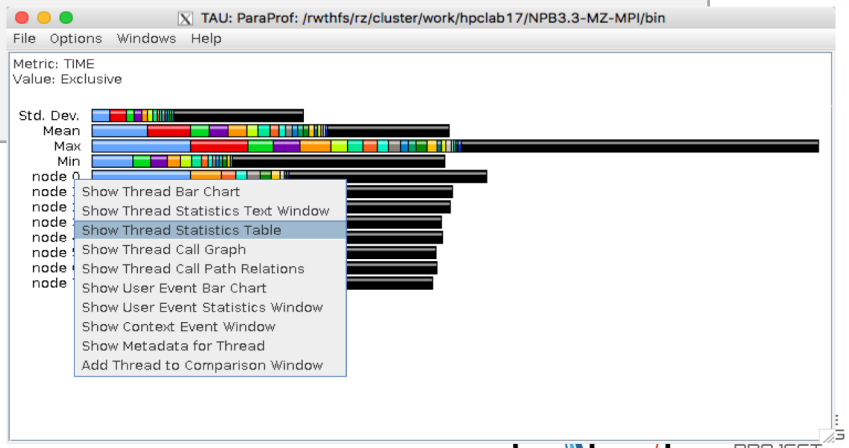
# tau_exec Example (continued)

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

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

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

- tau_exec can enable event based sampling while launching the executable using env TAU_SAMPLING=1 or tau_exec -ebs

# Event Based Sampling with TAU

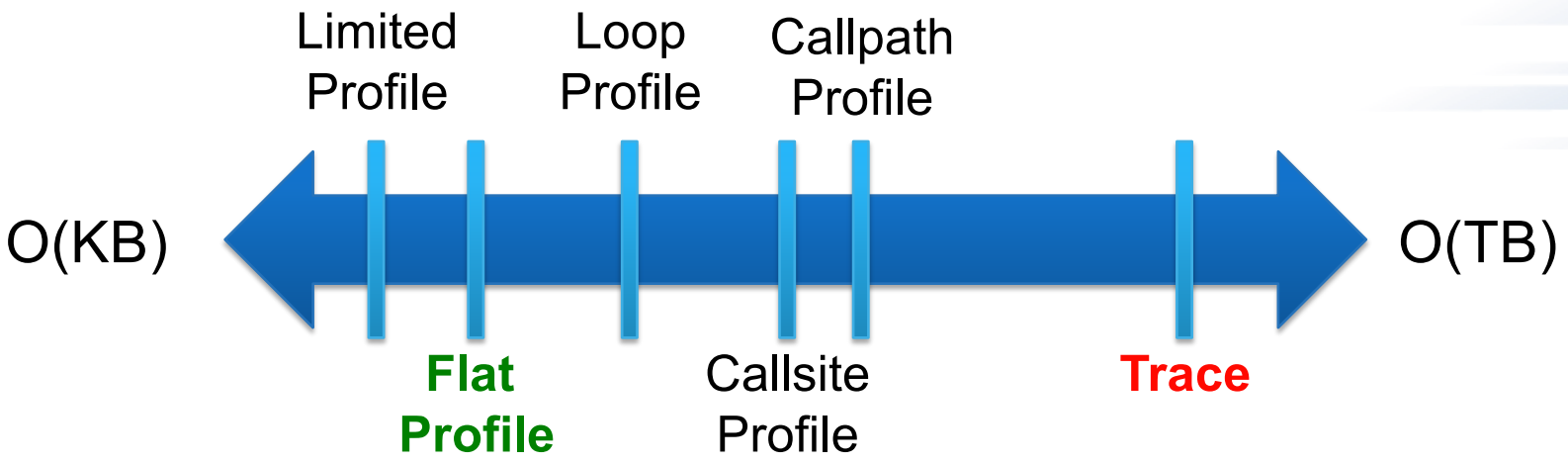
```
% cd MZ-NPB3.3-MPI; cat README
% make clean;
% make suite
% cd bin
% qsub -I -n 1 -A <Project_ID> -q training -t 50
% module unload darshan; module load intel tau
% export OMP_NUM_THREADS=4
% export TAU_OMPT_SUPPORT_LEVEL=full; export TAU_OMPT_RESOLVE_ADDRESS_EAGERLY=1
% aprun -n 16 tau_exec -T ompt,tr6 -ebs ./bt-mz.B.16
% On head node:
% module load tau
% paraprof
```




- Right Click on Node 0 and choose

Show Thread Statistics Table

## How much data do you want?



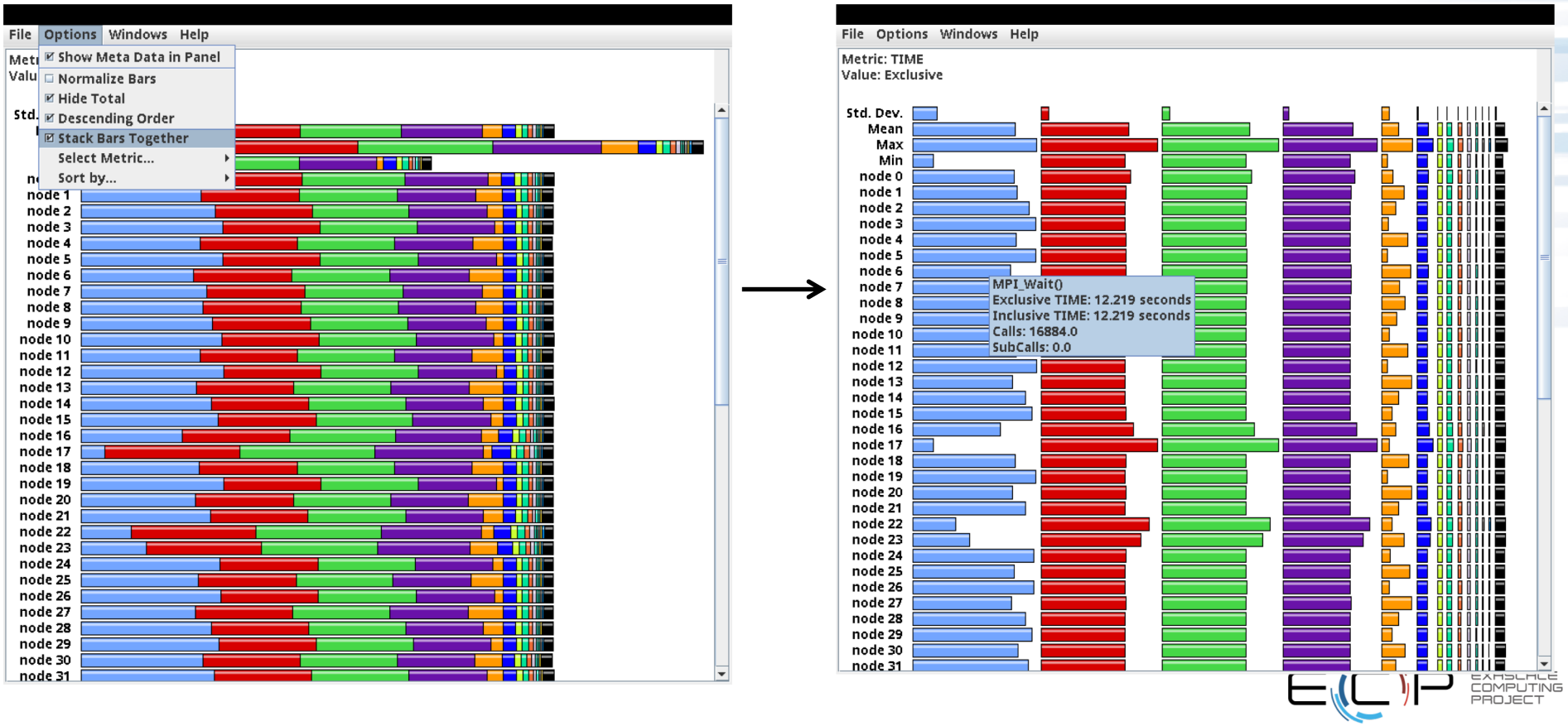
# Types of Performance Profiles

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

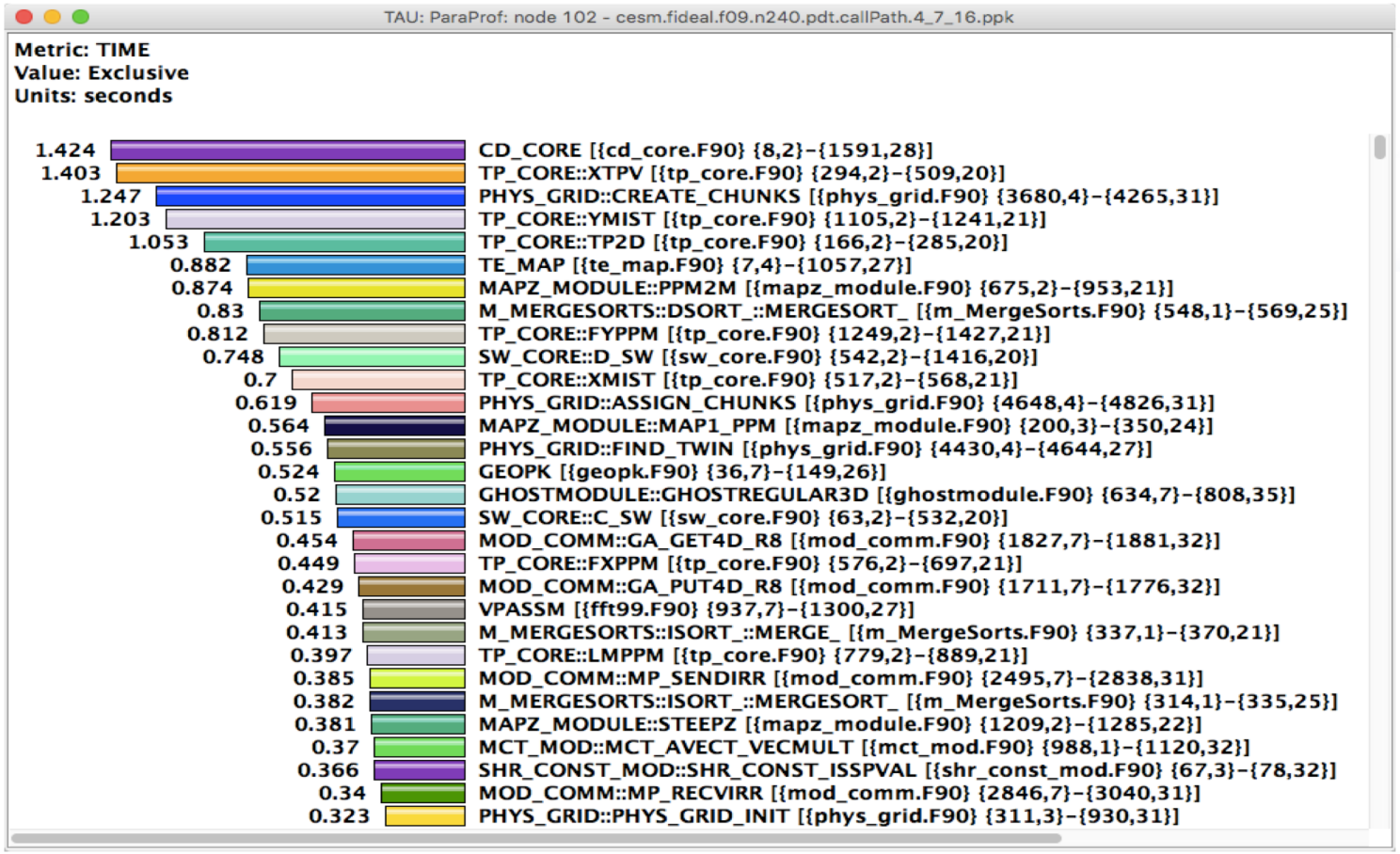
# ParaProf Profile Browser

```
% paraprof
```

# ParaProf Profile Browser



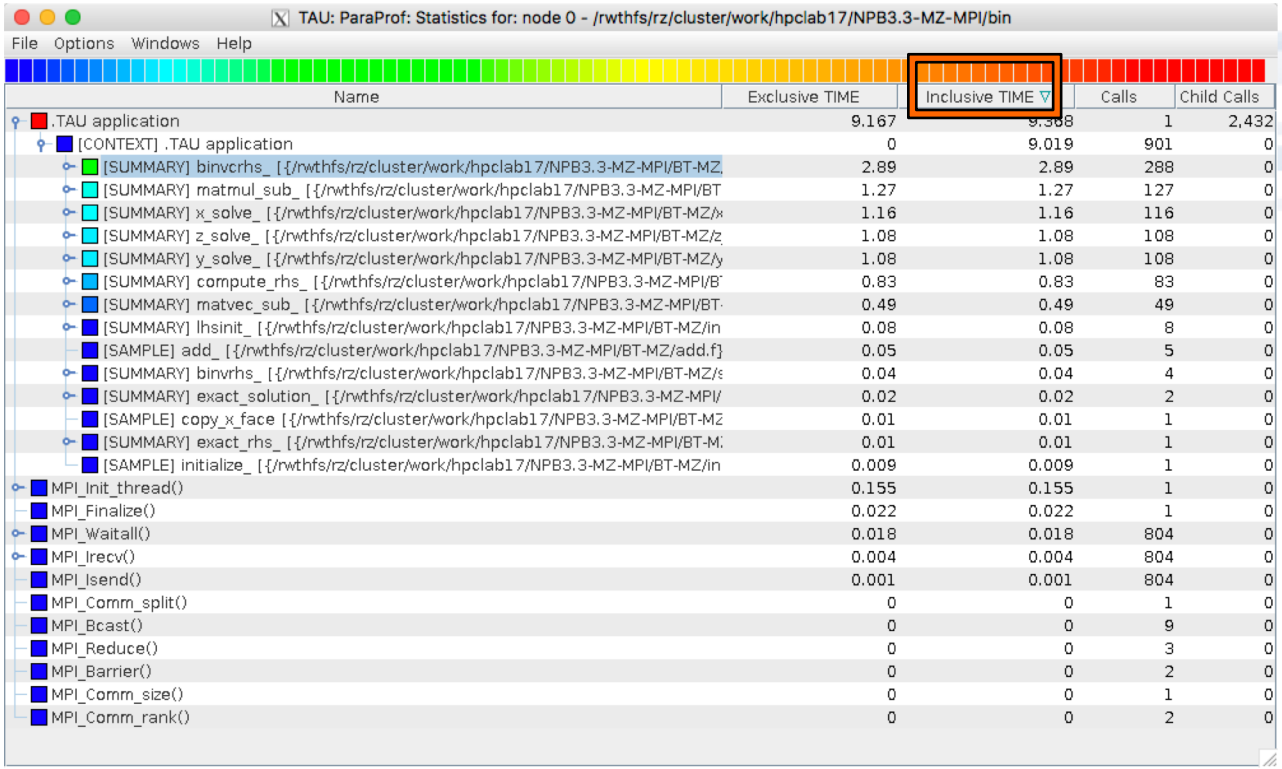
# TAU – Flat Profile



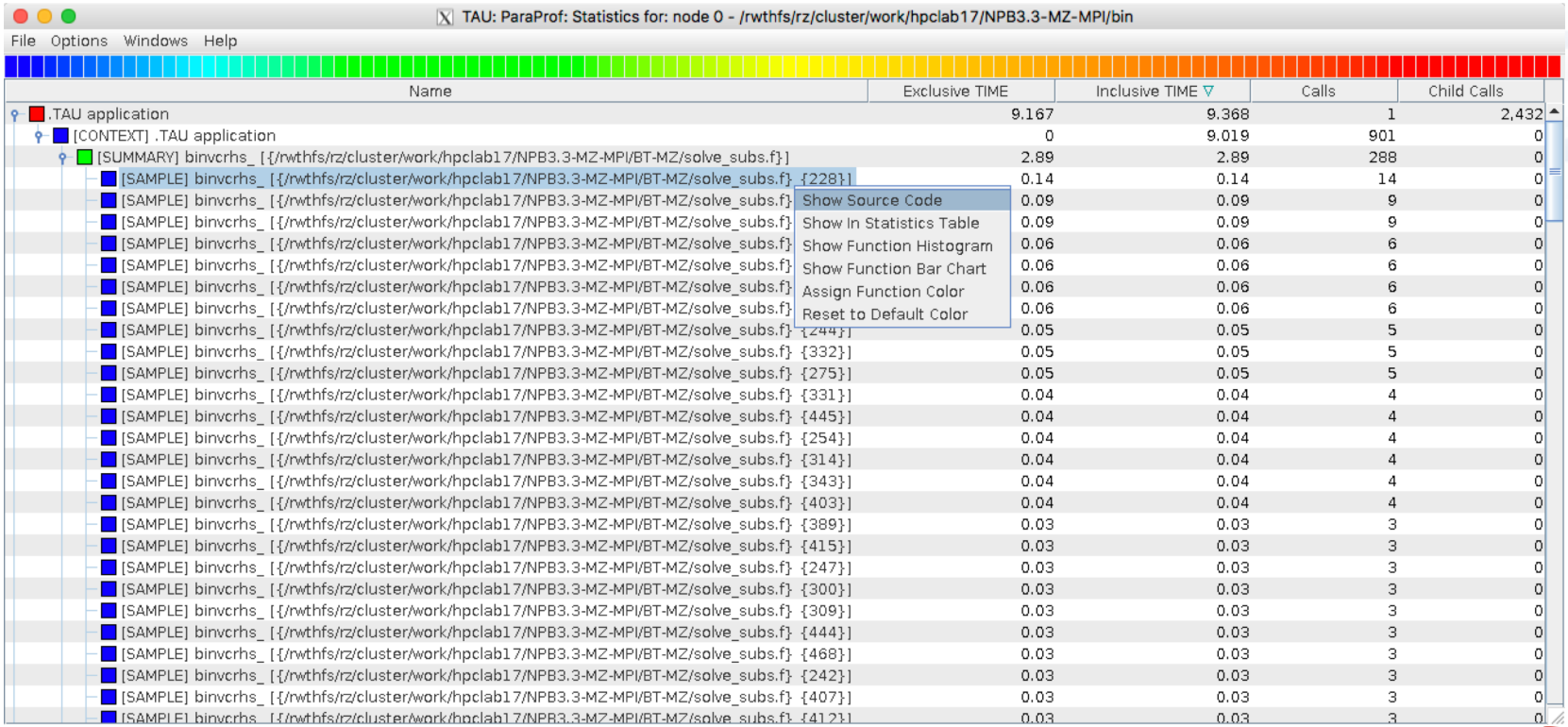
# ParaProf

- Click on Columns:
- to sort by incl time

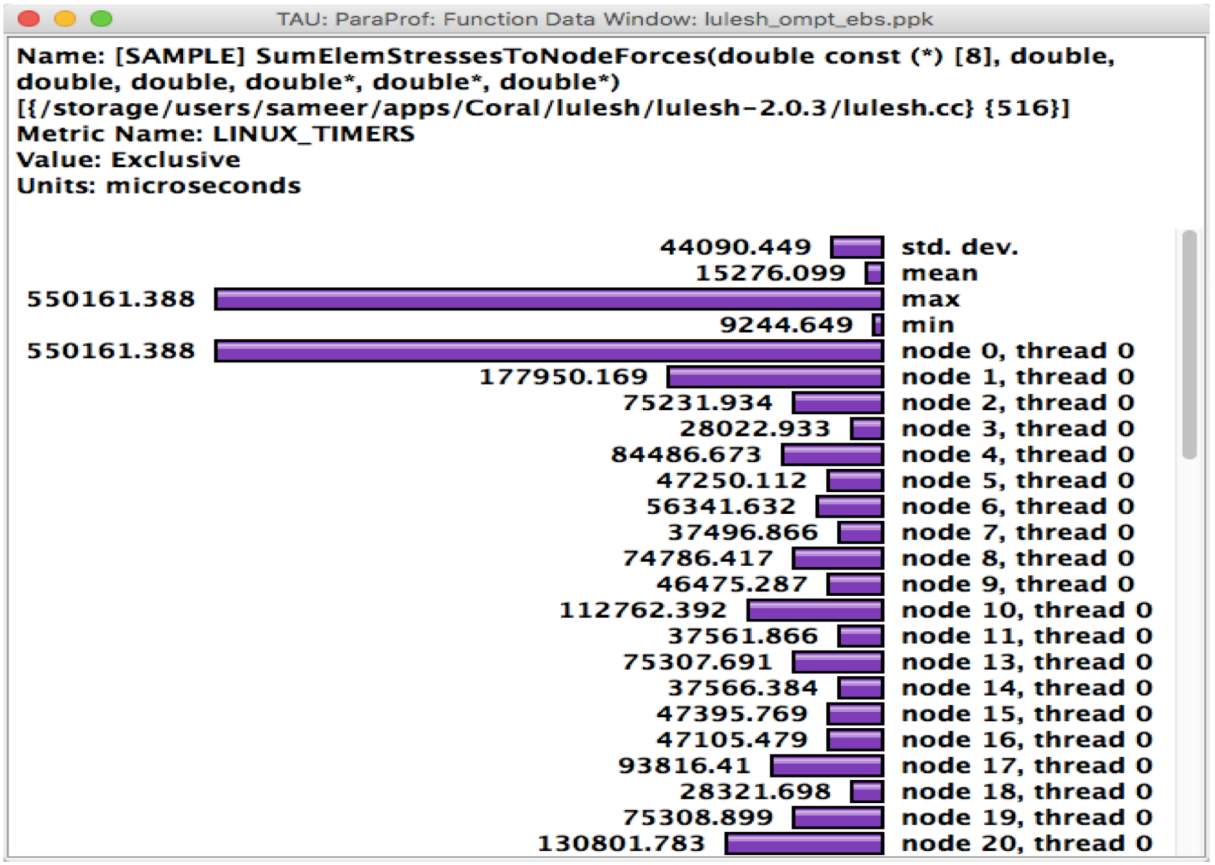
- Open binvcrhs
- Click on Sample



# ParaProf



# TAU – Event Based Sampling (EBS)




% export TAU_SAMPLING=1

# Callstack Sampling in TAU

TAU: ParaProf: Statistics for: n,c,t 2,0,0 - gamess_unw_call_ebs.ppk

| Name | Inclusive TIME ▽ | Calls |
|---|---|---|
| .TAU application | 79.592 | 1 |
| MPI_Recv() | 75.607 | 6,870 |
| [CONTEXT] MPI_Recv() | 74.848 | 1,497 |
| [UNWIND] /gpfs/mira-home/sameer/gamess-theta-tau/object/unport.f.410 [@] MAIN__ [{/gpfs/mira-home/sameer/gamess-theta-t | 26.196 | 524 |
| [UNWIND] /gpfs/mira-home/yuri/dist/Github/gamess-theta-tau/ddi/src/ddi_fortran.c.67 [@] beging_ [{/gpfs/mira-home/sameer/g | 21.7 | 434 |
| [UNWIND] /gpfs/mira-home/sameer/gamess-theta-tau/object/gamess.f.538 [@] main [{/gpfs/mira-home/sameer/gamess-theta-ta | 11.85 | 237 |
| [UNWIND] /gpfs/mira-home/yuri/dist/Github/gamess-theta-tau/ddi/src/ddi_init.c.113 [@] ddi_init_ [{/gpfs/mira-home/yuri/dist/Gi | 8.701 | 174 |
| [UNWIND] /gpfs/mira-home/yuri/dist/Github/gamess-theta-tau/ddi/src/ddi_server.c.99 [@] DDI_Init [{/gpfs/mira-home/yuri/dist/C | 5.75 | 115 |
| [UNWIND] /lib64/libc-2.22.so.0 [@] _start [{/home/abuild/rpmbuild/BUILD/glibc-2.22/csu/../sysdeps/x86_64/start.S} {118}] | 0.2 | 4 |
| [SAMPLE] GNII_DlaProgress [{/opt/cray/ugni/6.0.14-6.0.4.0_14.1__ge7db4a2.ari/lib64/libugni.so.0.6.0} {0}] | 0.2 | 4 |
| [UNWIND] [/opt/cray/ugni/6.0.14-6.0.4.0_14.1__ge7db4a2.ari/lib64/libugni.so.0.6.0.0] [@] UNRESOLVED UNKNOWN | 0.15 | 3 |
| [SAMPLE] GNI_CqGetEvent [{/opt/cray/ugni/6.0.14-6.0.4.0_14.1__ge7db4a2.ari/lib64/libugni.so.0.6.0} {0}] | 0.051 | 1 |
| [UNWIND] /opt/cray/pe/mpt/7.6.3/gni/mpich-intel/16.0/lib/libmpich_intel.so.3.0.1.0 [@] MPIDI_CH3I_Progress [{/opt/cray/pe/mpt/7 | 0.05 | 1 |
| MPI_Finalize() | 3.601 | 1 |
| MPI_Send() | 0.122 | 6,866 |
| MPI_Init_thread() | 0.112 | 1 |
| [CONTEXT] .TAU application | 0.05 | 1 |
| MPI_Bcast() | 0.014 | 6 |
| MPI_Allgather() | 0.004 | 3 |
| MPI_Barrier() | 0.003 | 7 |
| MPI_Comm_create() | 0.002 | 4 |
| MPI_Gather() | 0.002 | 1 |
| MPI_Comm_split() | 0.002 | 1 |
| MPI_Group_intersection() | 0.001 | 1 |
| MPI_Comm_group() | 0.001 | 1 |
| MPI_Group_incl() | 0 | 3 |
| MPI_Comm_rank() | 0 | 6 |
| MPI_Comm_size() | 0 | 2 |

% export TAU_SAMPLING=1; export TAU_EBS_UNWIND=1

# UNWINDING CALLSTACKS

TAU: ParaProf: Statistics for: n,c,t 2,0,0 - gamess_unw_call_ebs.ppk

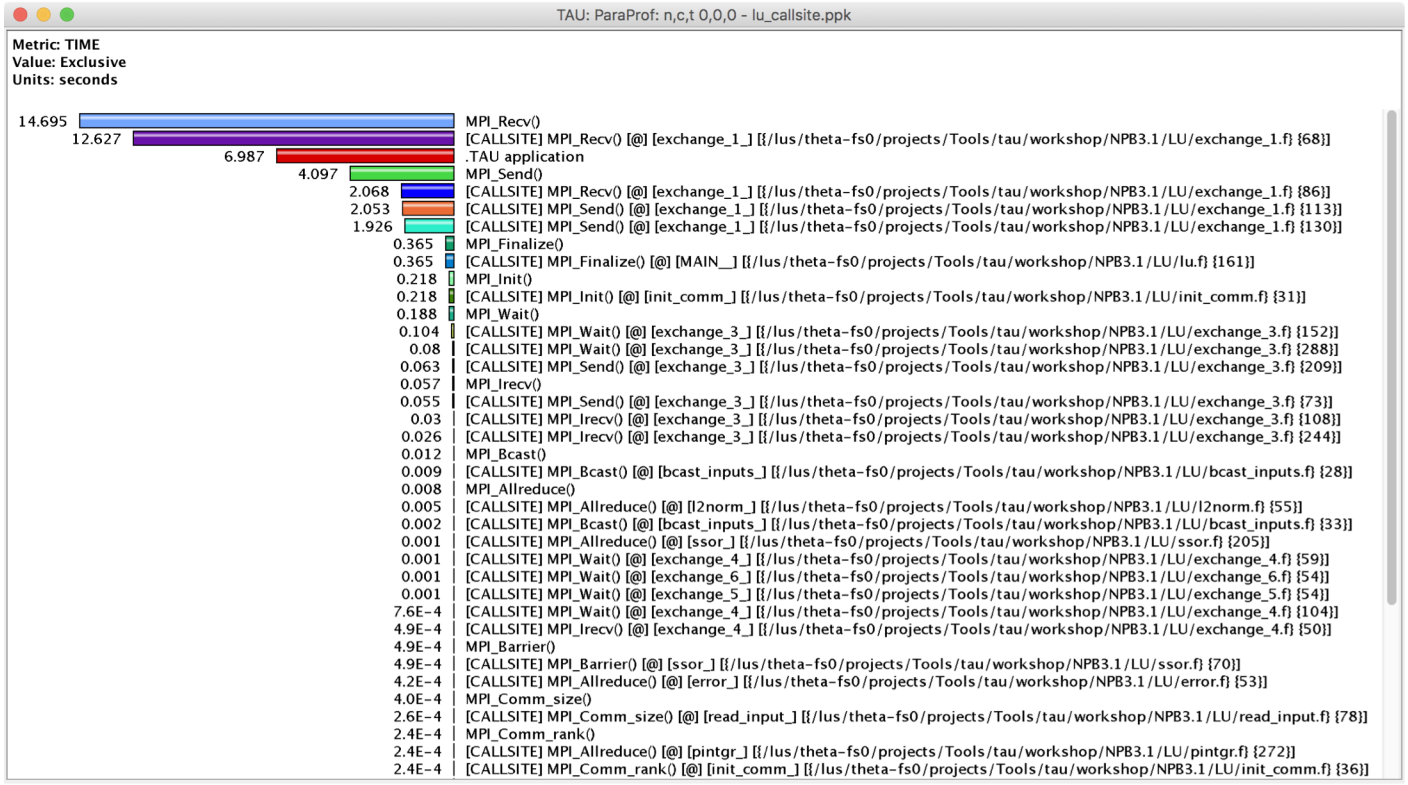
| Name | Inclusive TIME ▽ | Calls |
|---|---|---|
| .TAU application | 79.592 | 1 |
| MPI_Recv() | 75.607 | 6,870 |
| [CONTEXT] MPI_Recv() | 74.848 | 1,497 |
| [UNWIND] /gpfs/mira-home/sameer/gamess-theta-tau/object/unport.f.410 [@] MAIN__ [{/gpfs/mira-home/sameer/gamess-theta- | 26.196 | 524 |
| [UNWIND] /gpfs/mira-home/yuri/dist/Github/gamess-theta-tau/ddi/src/ddi_fortran.c.67 [@] beging_ [{/gpfs/mira-home/sameer/g | 21.7 | 434 |
| [UNWIND] /gpfs/mira-home/yuri/dist/Github/gamess-theta-tau/ddi/src/ddi_init.c.113 [@] ddi_init_ [{/gpfs/mira-home/yuri/dist | 21.7 | 434 |
| [UNWIND] /gpfs/mira-home/yuri/dist/Github/gamess-theta-tau/ddi/src/ddi_server.c.99 [@] DDI_Init [{/gpfs/mira-home/yuri/ | 21.7 | 434 |
| [UNWIND] /gpfs/mira-home/yuri/dist/Github/gamess-theta-tau/ddi/src/ddi_recv.c.65 [@] DDI_Server [{/gpfs/mira-home/y | 21.7 | 434 |
| [UNWIND] /lus/theta-fs0/software/perftools/tau/tau-2.26.3/src/Profile/TauMpi.c.2371 [@] DDI_Recv_request [{/gpfs/mira | 21.7 | 434 |
| [UNWIND] /opt/cray/pe/mpt/7.6.3/gni/mpich-intel/16.0/lib/libmpich_intel.so.3.0.1.0 [@] MPI_Recv [{/lus/theta-fs0/soft | 21.7 | 434 |
| [UNWIND] /opt/cray/pe/mpt/7.6.3/gni/mpich-intel/16.0/lib/libmpich_intel.so.3.0.1.0 [@] PMPI_Recv [{/opt/cray/pe/n | 21.7 | 434 |
| [UNWIND] /opt/cray/pe/mpt/7.6.3/gni/mpich-intel/16.0/lib/libmpich_intel.so.3.0.1.0 [@] MPIDI_CH3I_Progress [{/ | 21.45 | 429 |
| [UNWIND] /opt/cray/ugni/6.0.14-6.0.4.0_14.1__ge7db4a2.ari/lib64/libugni.so.0.6.0.0 [@] MPID_nem_gni_poll [{/ | 15.95 | 319 |
| [SAMPLE] GNI_SmsgGetNextWTag [{/opt/cray/ugni/6.0.14-6.0.4.0_14.1__ge7db4a2.ari/lib64/libugni.so.0.6.0} | 10.349 | 207 |
| [SAMPLE] GNI_CqGetEvent [{/opt/cray/ugni/6.0.14-6.0.4.0_14.1__ge7db4a2.ari/lib64/libugni.so.0.6.0} {0}] | 5.6 | 112 |
| [UNWIND] gni_poll.c.0 [@] MPID_nem_gni_poll [{/opt/cray/pe/mpt/7.6.3/gni/mpich-intel/16.0/lib/libmpich_inte | 5.25 | 105 |
| [UNWIND] /opt/cray/pe/mpt/7.6.3/gni/mpich-intel/16.0/lib/libmpich_intel.so.3.0.1.0 [@] MPID_nem_gni_poll [{/ | 0.25 | 5 |
| [UNWIND] UNRESOLVED [@] MPIDI_CH3I_Progress [{/opt/cray/pe/mpt/7.6.3/gni/mpich-intel/16.0/lib/libmpich_int | 0.25 | 5 |
| [UNWIND] /gpfs/mira-home/sameer/gamess-theta-tau/object/gamess.f.538 [@] main [{/gpfs/mira-home/sameer/gamess-theta-ta | 11.85 | 237 |
| [UNWIND] /gpfs/mira-home/yuri/dist/Github/gamess-theta-tau/ddi/src/ddi_init.c.113 [@] ddi_init_ [{/gpfs/mira-home/yuri/dist/G | 8.701 | 174 |
| [UNWIND] /gpfs/mira-home/yuri/dist/Github/gamess-theta-tau/ddi/src/ddi_server.c.99 [@] DDI_Init [{/gpfs/mira-home/yuri/dist/( | 5.75 | 115 |
| [UNWIND] /lib64/libc-2.22.so.0 [@] _start [{/home/abuild/rpmbuild/BUILD/glibc-2.22/csu/../sysdeps/x86_64/start.S} {118}] | 0.2 | 4 |
| [SAMPLE] GNII_DlaProgress [{/opt/cray/ugni/6.0.14-6.0.4.0_14.1__ge7db4a2.ari/lib64/libugni.so.0.6.0} {0}] | 0.2 | 4 |
| [UNWIND] [/opt/cray/ugni/6.0.14-6.0.4.0_14.1__ge7db4a2.ari/lib64/libugni.so.0.6.0.0] [@] UNRESOLVED UNKNOWN | 0.15 | 3 |
| [SAMPLE] GNI_CqGetEvent [{/opt/cray/ugni/6.0.14-6.0.4.0_14.1__ge7db4a2.ari/lib64/libugni.so.0.6.0} {0}] | 0.051 | 1 |
| [UNWIND] /opt/cray/pe/mpt/7.6.3/gni/mpich-intel/16.0/lib/libmpich_intel.so.3.0.1.0 [@] MPIDI_CH3I_Progress [{/opt/cray/pe/mpt/ | 0.05 | 1 |
| MPI_Finalize() | 3.601 | 1 |
| MPI_Send() | 0.122 | 6,866 |
| MPI_Init_thread() | 0.112 | 1 |
| [CONTEXT] .TAU application | 0.05 | 1 |

% export TAU_SAMPLING=1; export TAU_EBS_UNWIND=1

# UNWINDING CALLSTACKS

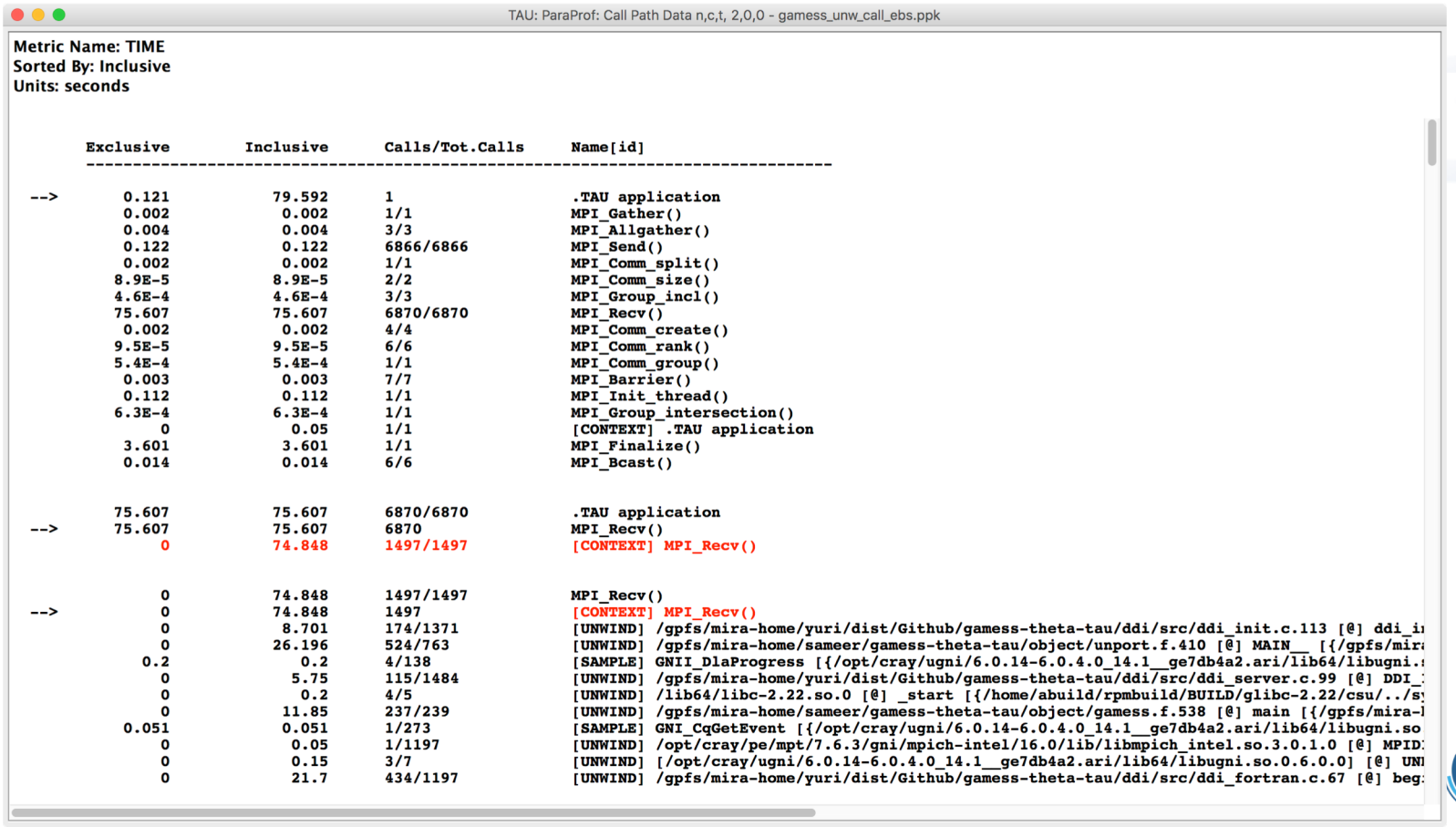

| Name | Inclusive TIME | Calls |
|---|---|---|
| .TAU application | 79.592 | 1 |
| MPI_Recv() | 75.607 | 6,870 |
| [CONTEXT] MPI_Recv() | 74.848 | 1,497 |
| [UNWIND] /gpfs/mira-home/sameer/gamess-theta-tau/object/unport.f.410 [@] MAIN__ [{/gpfs/mira-home/sameer/gamess-theta- | 26.196 | 524 |
| [UNWIND] /gpfs/mira-home/yuri/dist/Github/gamess-theta-tau/ddi/src/ddi_fortran.c.67 [@] beging_ [{/gpfs/mira-home/sameer/g | 21.7 | 434 |
| [UNWIND] /gpfs/mira-home/sameer/gamess-theta-tau/object/gamess.f.538 [@] main [{/gpfs/mira-home/sameer/gamess-theta-ta | 11.85 | 237 |
| [UNWIND] /gpfs/mira-home/sameer/gamess-theta-tau/object/unport.f.410 [@] MAIN__ [{/gpfs/mira-home/sameer/gamess-thet | 11.85 | 237 |
| [UNWIND] /gpfs/mira-home/yuri/dist/Github/gamess-theta-tau/ddi/src/ddi_fortran.c.67 [@] beging_ [{/gpfs/mira-home/sam | 11.85 | 237 |
| [UNWIND] /gpfs/mira-home/yuri/dist/Github/gamess-theta-tau/ddi/src/ddi_init.c.113 [@] ddi_init_ [{/gpfs/mira-home/yur | 11.85 | 237 |
| [UNWIND] /gpfs/mira-home/yuri/dist/Github/gamess-theta-tau/ddi/src/ddi_server.c.99 [@] DDI_Init [{/gpfs/mira-home/ | 11.85 | 237 |
| [UNWIND] /gpfs/mira-home/yuri/dist/Github/gamess-theta-tau/ddi/src/ddi_recv.c.65 [@] DDI_Server [{/gpfs/mira-ho | 11.85 | 237 |
| [UNWIND] /lus/theta-fs0/software/perftools/tau/tau-2.26.3/src/Profile/TauMpi.c.2371 [@] DDI_Recv_request [{/gpfs | 11.85 | 237 |
| [UNWIND] /opt/cray/pe/mpt/7.6.3/gni/mpich-intel/16.0/lib/libmpich_intel.so.3.0.1.0 [@] MPI_Recv [{/lus/theta-fs( | 11.85 | 237 |
| [UNWIND] /opt/cray/pe/mpt/7.6.3/gni/mpich-intel/16.0/lib/libmpich_intel.so.3.0.1.0 [@] PMPI_Recv [{/opt/cray, | 11.7 | 234 |
| [SAMPLE] MPIDI_CH3I_Progress [{/opt/cray/pe/mpt/7.6.3/gni/mpich-intel/16.0/lib/libmpich_intel.so.3.0.1} {0 | 11.3 | 226 |
| [SAMPLE] MPIDU_Sched_are_pending [{/opt/cray/pe/mpt/7.6.3/gni/mpich-intel/16.0/lib/libmpich_intel.so.3.0 | 0.2 | 4 |
| [SAMPLE] MPID_nem_gni_poll [{/opt/cray/pe/mpt/7.6.3/gni/mpich-intel/16.0/lib/libmpich_intel.so.3.0.1} {0}] | 0.15 | 3 |
| [SAMPLE] MPID_nem_network_poll [{/opt/cray/pe/mpt/7.6.3/gni/mpich-intel/16.0/lib/libmpich_intel.so.3.0.1 | 0.05 | 1 |
| [UNWIND] ch3_progress.c.0 [@] PMPI_Recv [{/opt/cray/pe/mpt/7.6.3/gni/mpich-intel/16.0/lib/libmpich_intel.so. | 0.15 | 3 |
| [UNWIND] /gpfs/mira-home/yuri/dist/Github/gamess-theta-tau/ddi/src/ddi_init.c.113 [@] ddi_init_ [{/gpfs/mira-home/yuri/dist/G | 8.701 | 174 |
| [UNWIND] /gpfs/mira-home/yuri/dist/Github/gamess-theta-tau/ddi/src/ddi_server.c.99 [@] DDI_Init [{/gpfs/mira-home/yuri/dist/( | 5.75 | 115 |
| [UNWIND] /lib64/libc-2.22.so.0 [@] _start [{/home/abuild/rpmbuild/BUILD/glibc-2.22/csu/../sysdeps/x86_64/start.S} {118}] | 0.2 | 4 |
| [SAMPLE] GNII_DlaProgress [{/opt/cray/ugni/6.0.14-6.0.4.0_14.1__ge7db4a2.ari/lib64/libugni.so.0.6.0} {0}] | 0.2 | 4 |
| [UNWIND] [/opt/cray/ugni/6.0.14-6.0.4.0_14.1__ge7db4a2.ari/lib64/libugni.so.0.6.0.0] [@] UNRESOLVED UNKNOWN | 0.15 | 3 |
| [SAMPLE] GNI_CqGetEvent [{/opt/cray/ugni/6.0.14-6.0.4.0_14.1__ge7db4a2.ari/lib64/libugni.so.0.6.0} {0}] | 0.051 | 1 |
| [UNWIND] /opt/cray/pe/mpt/7.6.3/gni/mpich-intel/16.0/lib/libmpich_intel.so.3.0.1.0 [@] MPIDI_CH3I_Progress [{/opt/cray/pe/mpt/ | 0.05 | 1 |
| MPI_Finalize() | 3.601 | 1 |
| MPI_Send() | 0.122 | 6,866 |
| MPI_Init_thread() | 0.112 | 1 |
| [CONTEXT] .TAU application | 0.05 | 1 |

# Callsite Profiling and Tracing




% export TAU_CALLSITE=1

# CALLPATH THREAD RELATIONS WINDOW



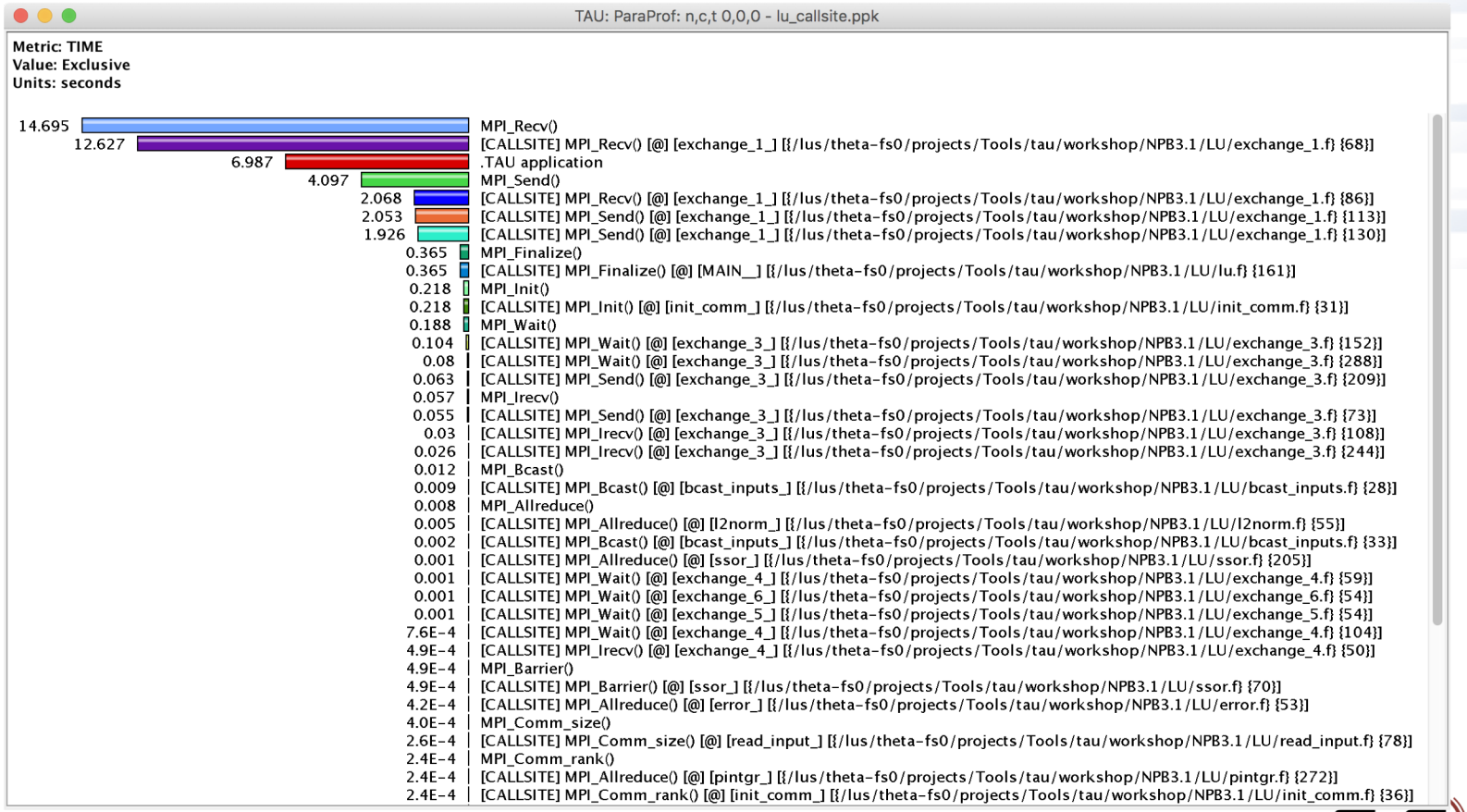
# CALLPATH THREAD RELATIONS WINDOW

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

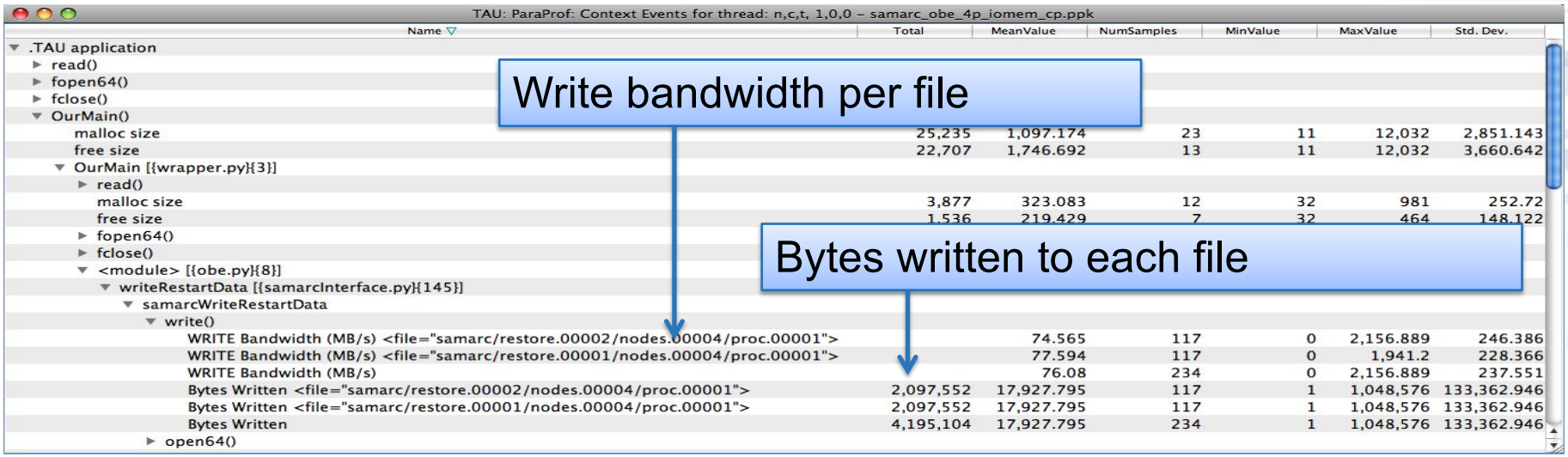
Metric Name: TIME
Sorted By: Exclusive
Units: seconds

| | Exclusive | Inclusive | Calls/Tot.Calls | Name[id] |
|---|---|---|---|---|
| | 75.607 | 75.607 | 6870/6870 | .TAU application |
| --> | 75.607 | 75.607 | 6870 | MPI_Recv() |
| | 0 | 74.848 | 1497/1497 | [CONTEXT] MPI_Recv() |
| | 0.15 | 0.15 | 3/444 | [UNWIND] /opt/cray/pe/mpt/7.6.3/gni/mpich-intel/16.0/lib/libmpich_intel.so.3.0.1.0 [@] PMPI_Recv |
| | 22.046 | 22.046 | 441/444 | [UNWIND] /opt/cray/pe/mpt/7.6.3/gni/mpich-intel/16.0/lib/libmpich_intel.so.3.0.1.0 [@] MPIDI_CH3 |
| --> | 22.196 | 22.196 | 444 | [SAMPLE] MPID_nem_gni_poll [{/opt/cray/pe/mpt/7.6.3/gni/mpich-intel/16.0/lib/libmpich_intel.so.3 |
| | 5.6 | 5.6 | 112/273 | [UNWIND] /opt/cray/ugni/6.0.14-6.0.4.0_14.1__ge7db4a2.ari/lib64/libugni.so.0.6.0.0 [@] MPID_nem_ |
| | 0.051 | 0.051 | 1/273 | [CONTEXT] MPI_Recv() |
| | 7.651 | 7.651 | 153/273 | [UNWIND] /opt/cray/ugni/6.0.14-6.0.4.0_14.1__ge7db4a2.ari/lib64/libugni.so.0.6.0.0 [@] MPID_nem_ |
| | 0.35 | 0.35 | 7/273 | [UNWIND] [/opt/cray/ugni/6.0.14-6.0.4.0_14.1__ge7db4a2.ari/lib64/libugni.so.0.6.0.0] [@] UNRESOLV |
| --> | 13.652 | 13.652 | 273 | [SAMPLE] GNI_CqGetEvent [{/opt/cray/ugni/6.0.14-6.0.4.0_14.1__ge7db4a2.ari/lib64/libugni.so.0.6. |
| | 11.3 | 11.3 | 226/226 | [UNWIND] /opt/cray/pe/mpt/7.6.3/gni/mpich-intel/16.0/lib/libmpich_intel.so.3.0.1.0 [@] PMPI_Recv |
| --> | 11.3 | 11.3 | 226 | [SAMPLE] MPIDI_CH3I_Progress [{/opt/cray/pe/mpt/7.6.3/gni/mpich-intel/16.0/lib/libmpich_intel.so |
| | 10.349 | 10.349 | 207/207 | [UNWIND] /opt/cray/ugni/6.0.14-6.0.4.0_14.1__ge7db4a2.ari/lib64/libugni.so.0.6.0.0 [@] MPID_nem_ |
| --> | 10.349 | 10.349 | 207 | [SAMPLE] GNI_SmsgGetNextWTag [{/opt/cray/ugni/6.0.14-6.0.4.0_14.1__ge7db4a2.ari/lib64/libugni.so |
| | 0.2 | 0.2 | 4/138 | [CONTEXT] MPI_Recv() |
| | 6.701 | 6.701 | 134/138 | [UNWIND] /opt/cray/ugni/6.0.14-6.0.4.0_14.1__ge7db4a2.ari/lib64/libugni.so.0.6.0.0 [@] GNI_CqGetI |
| --> | 6.901 | 6.901 | 138 | [SAMPLE] GNII_DlaProgress [{/opt/cray/ugni/6.0.14-6.0.4.0_14.1__ge7db4a2.ari/lib64/libugni.so.0. |
| | 5.25 | 5.25 | 105/109 | [UNWIND] gni_poll.c.0 [@] MPID_nem_gni_poll [{/opt/cray/pe/mpt/7.6.3/gni/mpich-intel/16.0/lib/lil |
| | 0.2 | 0.2 | 4/109 | [UNWIND] gni_poll.c.0 [@] MPIDI_CH3I_Progress [{/opt/cray/pe/mpt/7.6.3/gni/mpich-intel/16.0/lib/ |
| --> | 5.45 | 5.45 | 109 | [SAMPLE] MPID_nem_gni_check_localCQ [{gni_poll.c} {0}] |
| | 3.601 | 3.601 | 1/1 | .TAU application |
| --> | 3.601 | 3.601 | 1 | MPI_Finalize() |

# Callsite Profiling and Tracing (TAU_CALLSITE=1)

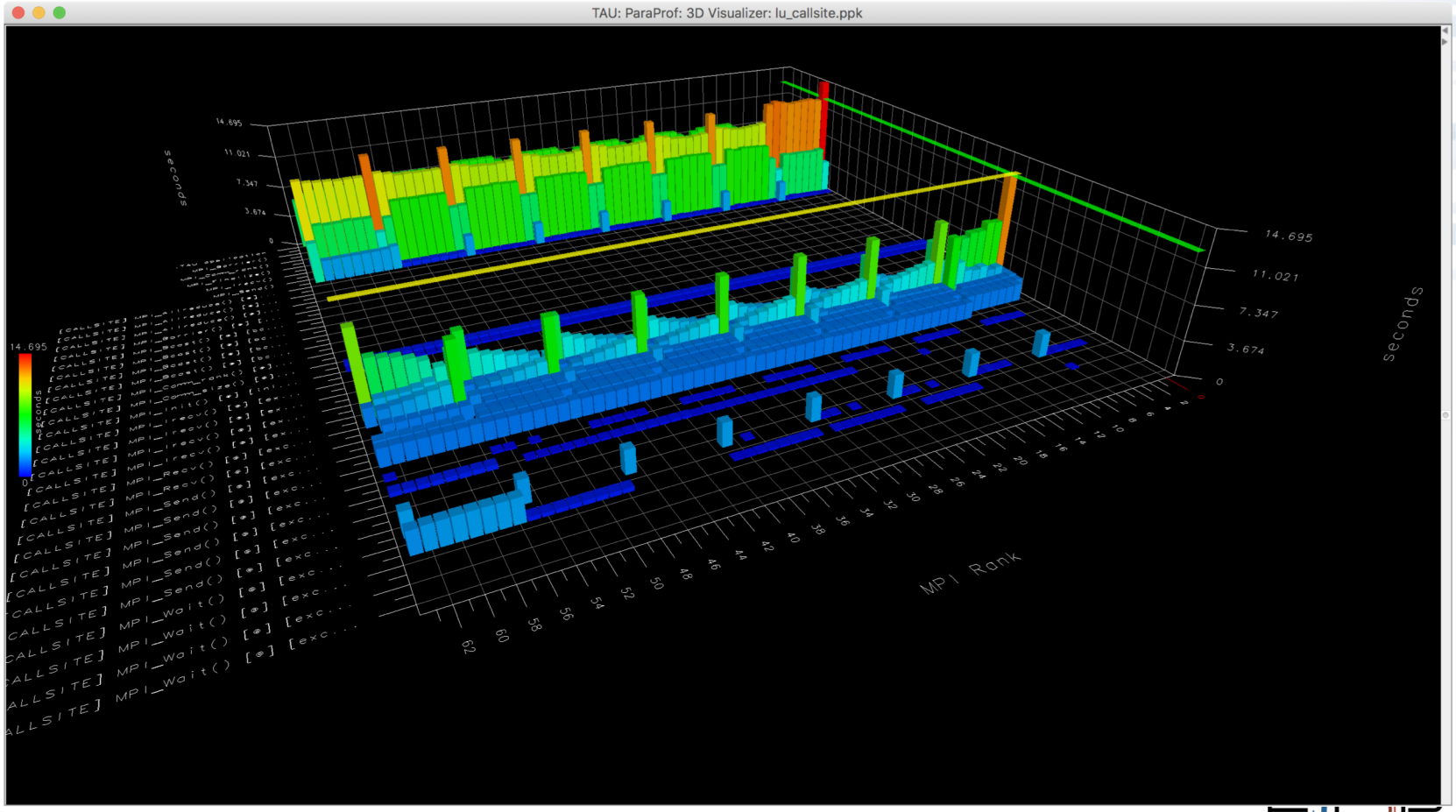


# TAU – Context Events

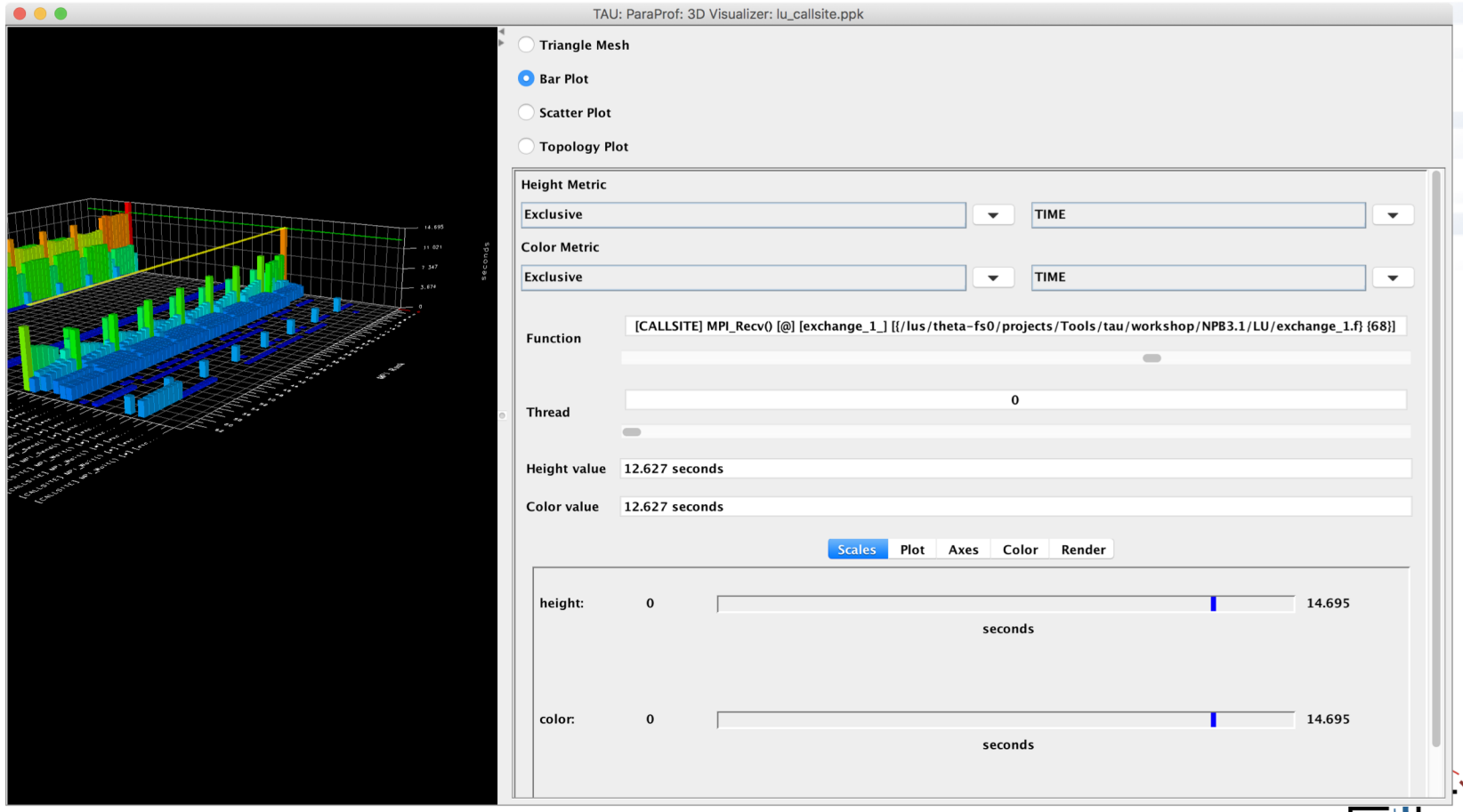



| Name | Total | MeanValue | NumSamples | MinValue | MaxValue | Std. Dev. |
|---|---|---|---|---|---|---|
| .TAU application | | | | | | |
| read() | | | | | | |
| fopen64() | | | | | | |
| fclose() | | | | | | |
| OurMain() | | | | | | |
| malloc size | 25,235 | 1,097.174 | 23 | 11 | 12,032 | 2,851.143 |
| free size | 22,707 | 1,746.692 | 13 | 11 | 12,032 | 3,660.642 |
| OurMain [{wrapper.py}{3}] | | | | | | |
| read() | | | | | | |
| malloc size | 3,877 | 323.083 | 12 | 32 | 981 | 252.72 |
| free size | 1,536 | 219.429 | 7 | 32 | 464 | 148.122 |
| fopen64() | | | | | | |
| fclose() | | | | | | |
| <module> [{obe.py}{8}] | | | | | | |
| writeRestartData [{samarcInterface.py}{145}] | | | | | | |
| samarcWriteRestartData | | | | | | |
| write() | | | | | | |
| WRITE Bandwidth (MB/s) <file="samarc/restore.00002/nodes.00004/proc.00001"> | | 74.565 | 117 | 0 | 2,156.889 | 246.386 |
| WRITE Bandwidth (MB/s) <file="samarc/restore.00001/nodes.00004/proc.00001"> | | 77.594 | 117 | 0 | 1,941.2 | 228.366 |
| WRITE Bandwidth (MB/s) | | 76.08 | 234 | 0 | 2,156.889 | 237.551 |
| Bytes Written <file="samarc/restore.00002/nodes.00004/proc.00001"> | 2,097,552 | 17,927.795 | 117 | 1 | 1,048,576 | 133,362.946 |
| Bytes Written <file="samarc/restore.00001/nodes.00004/proc.00001"> | 2,097,552 | 17,927.795 | 117 | 1 | 1,048,576 | 133,362.946 |
| Bytes Written | 4,195,104 | 17,927.795 | 234 | 1 | 1,048,576 | 133,362.946 |
| open64() | | | | | | |

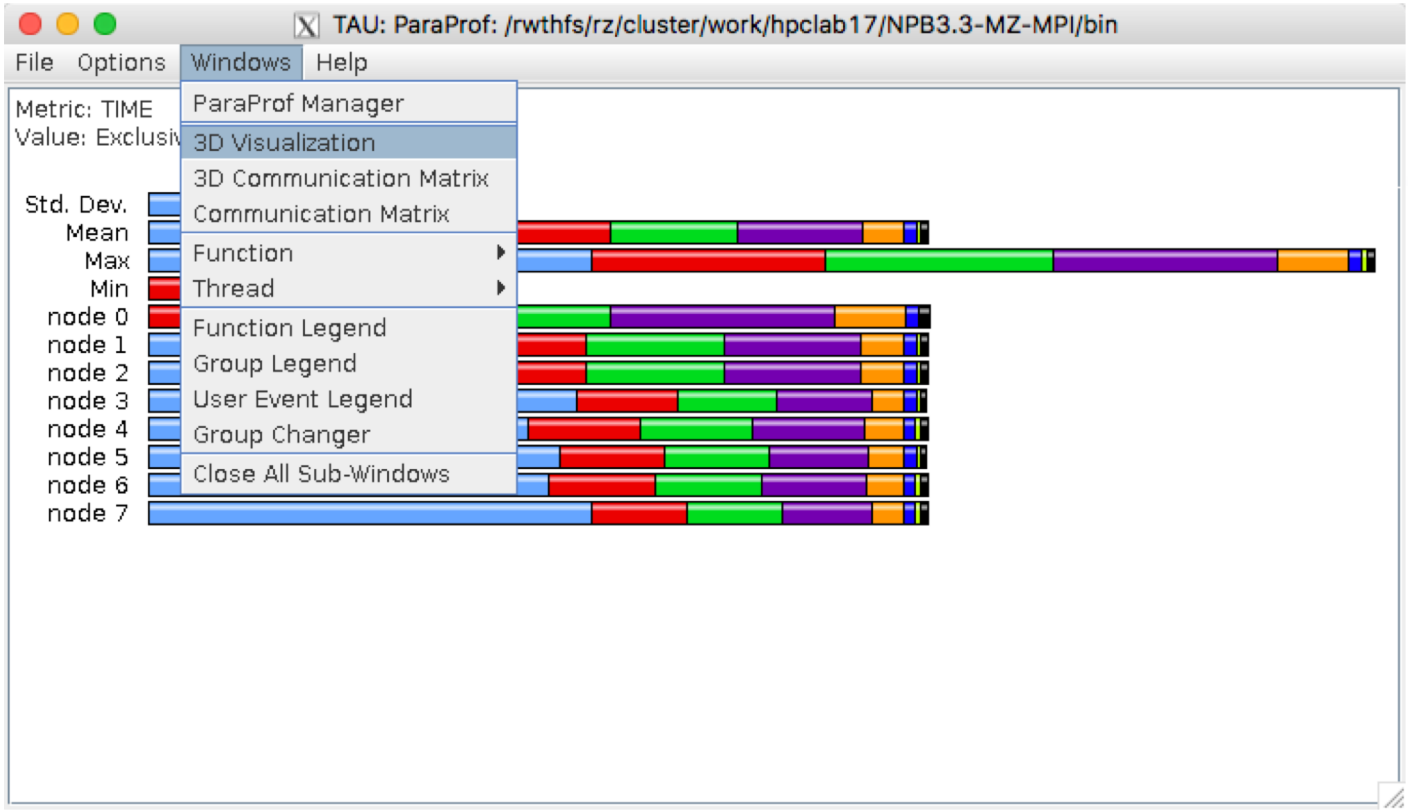
# Callsite Profiling and Tracing



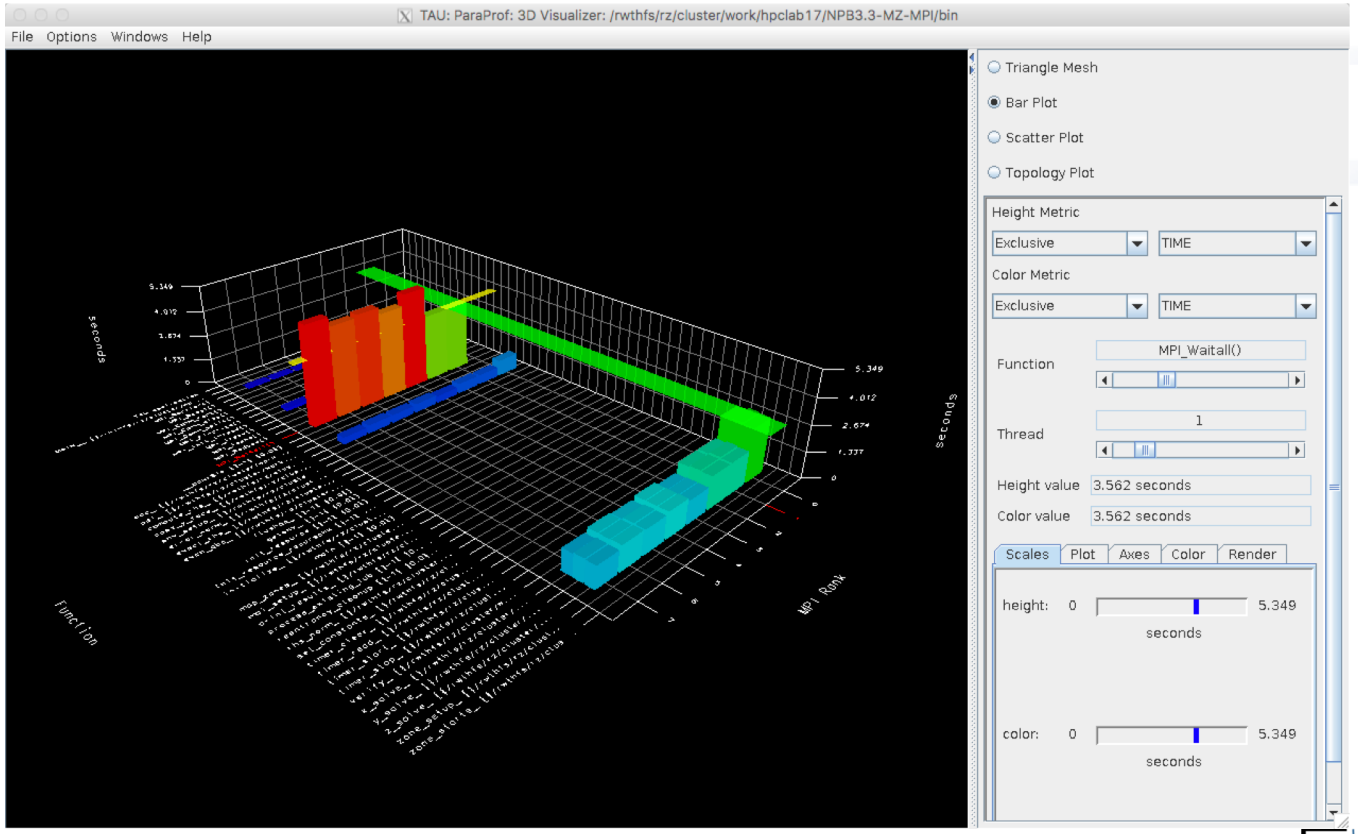
# Callsite Profiling and Tracing



# ParaProf with Optimized Instrumentation



# 3D Visualization with ParaProf

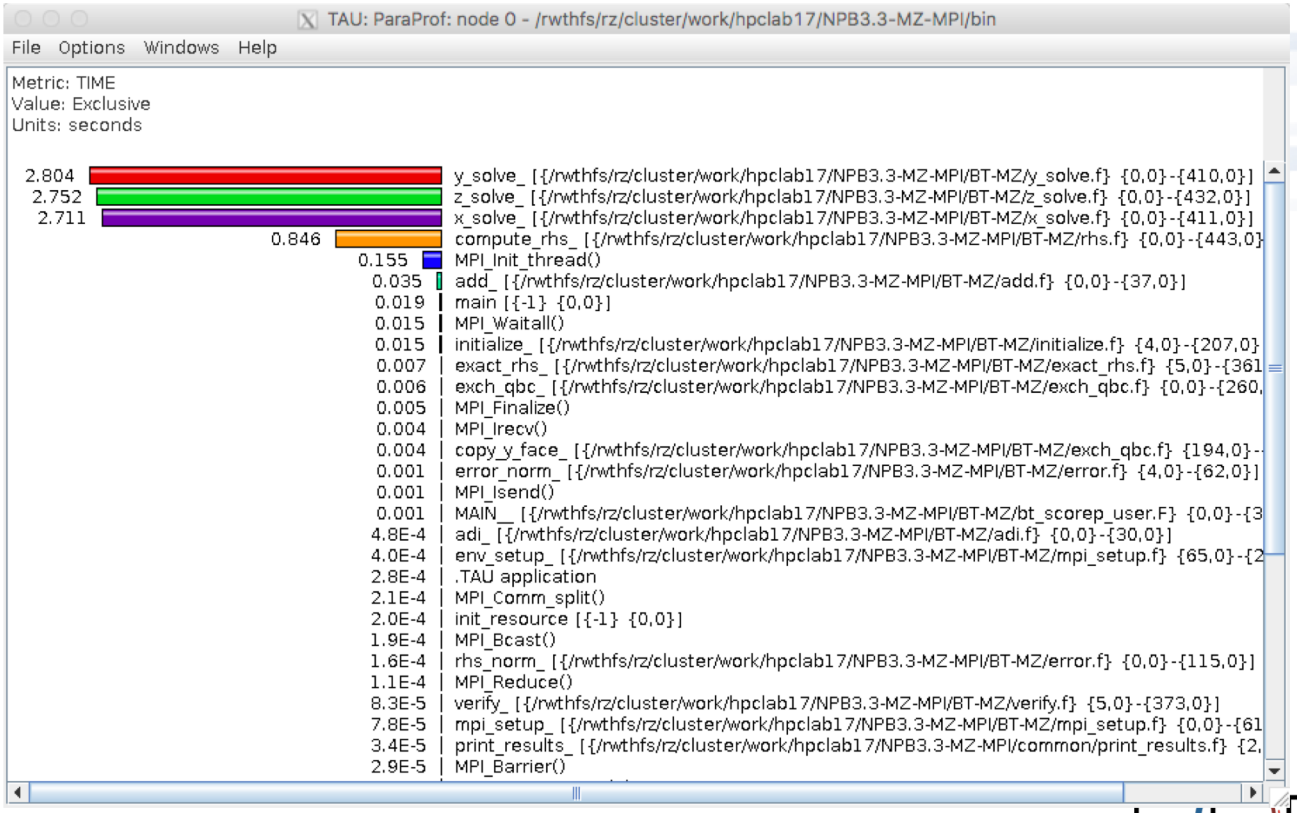





EXASCALE COMPUTING PROJECT

# ParaProf: Node 0

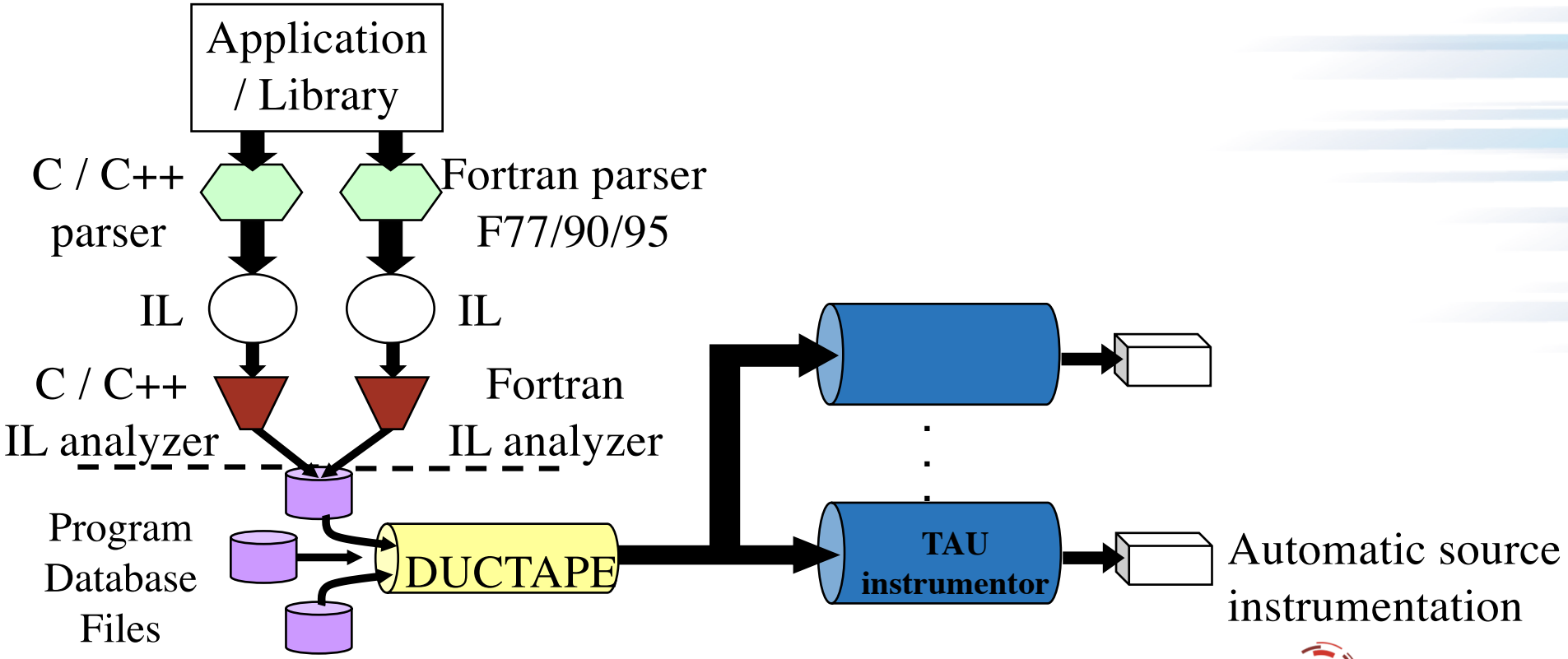
- Optimized
- instrumentation!



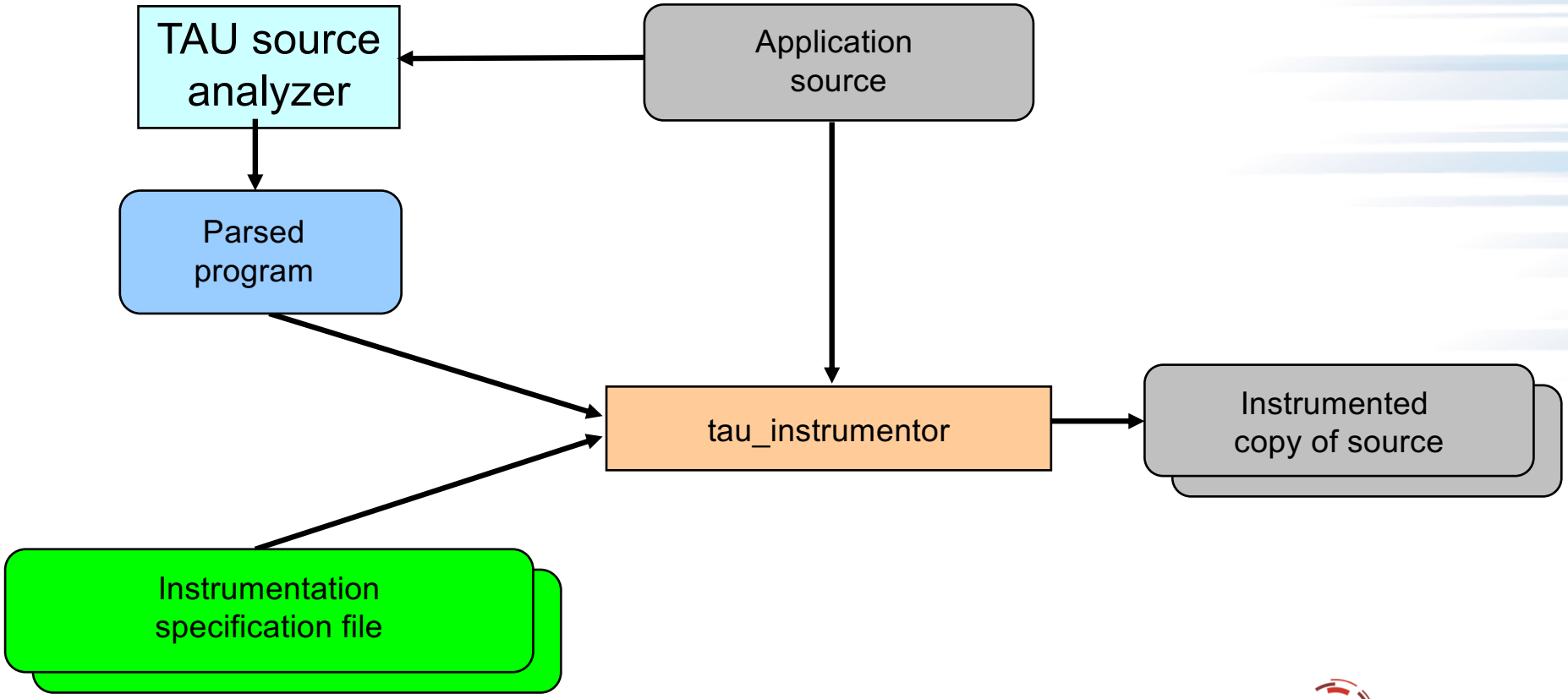

# Source Instrumentation




# TAU's Static Analysis System: Program Database Toolkit (PDT)



# PDT: automatic source instrumentation



# Using SOURCE Instrumentation in TAU

- TAU supports several compilers, measurement, and thread options

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

  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:

  **% export TAU_MAKEFILE=$TAU/Makefile.tau-intel-papi-mpi-pdt**

  **% 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:

  **% ftn 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 TAU

- Installing PDT:
  - wget http://tau.uoregon.edu/pdt_lite.tgz
  - ./configure –prefix=<dir>; make ; make install
- Installing TAU on Theta:
  - wget http://tau.uoregon.edu/tau.tgz
  - ./configure –arch=craycnl -mpi –pdt=<dir> -bfd=download –unwind=download –iowrapper;
  - make install
  - For x86_64 clusters running Linux
  - ./configure -c++=mpicxx –cc=mpicc –fortran=mpif90 –pdt=<dir> -bfd=download –unwind=download
  - 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

# INSTALLING TAU on Laptops

- Installing TAU under Mac OS X:
  - wget http://tau.uoregon.edu/tau.dmg
  - Install tau.dmg
- Installing TAU under Windows
  - http://tau.uoregon.edu/tau.exe
- Installing TAU under Linux
  - http://tau.uoregon.edu/tau.tgz
  - ./configure; make install
  - export PATH=<taudir>/x86_64/bin:$PATH

# Different Makefiles for TAU Compiler

```
%  module load tau
% ls $TAU/Makefile.*
/soft/perftools/tau/tau-2.29/craycnl/lib/Makefile.tau-intel-papi-mpi-pdt
/soft/perftools/tau/tau-2.29/craycnl/lib/Makefile.tau-intel-papi-mpi-pthread-pdt
/soft/perftools/tau/tau-2.29/craycnl/lib/Makefile.tau-intel-papi-ompt-v5-mpi-pdt-openmp
/soft/perftools/tau/tau-2.29/craycnl/lib/Makefile.tau-intel-papi-pthread-pdt
```

For an MPI+OpenMP+F90 application with Intel compilers and Cray MPI, you may choose **Makefile.tau-intel-papi-mpi-pdt**

Supports MPI instrumentation & PDT for automatic source instrumentation

```
% export TAU_MAKEFILE=$TAU/Makefile.tau-intel-papi-mpi-pdt
% tau_f90.sh app.f90 -o app; aprun –n 256 ./app; paraprof
```

# Configuration tags for tau_exec

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

% ./configure -pdt=<dir> -mpi; make install
```
creates
```
Makefile.tau-mpi-pdt
shared-mpi-pdt/libTAU.so
```
To explicitly choose preloading of shared-<options>/libTAU.so change:
```
% aprun -n  256 ./a.out     to
% aprun -n  256  tau_exec -T <comma_separated_options> ./a.out

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

# Compile-Time Options

- Optional parameters for the TAU_OPTIONS environment variable:
  % tau_compiler.sh --help

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

# Compile-Time Options (contd.)

- Optional parameters for the TAU_OPTIONS environment variable:
  % tau_compiler.sh

| | |
|---|---|
| -optMICOffload | Links code for Intel MIC offloading, requires both host and MIC TAU libraries |
| -optShared | Use TAU's shared library (libTAU.so) instead of static library (default) |
| · -optPdtCxxOpts="" | Options for C++ parser in PDT (cxxparse). |
| · -optPdtF90Parser="" | Specify a different Fortran parser |
| · -optPdtCleanscapeParser | Specify the Cleanscape Fortran parser instead of GNU gfparser |
| -optTau="" | Specify options to the tau_instrumentor |
| -optTrackDMAPP | Enable instrumentation of low-level DMAPP API calls on Cray |
| -optTrackPthread | Enable instrumentation of pthread calls |

See tau_compiler.sh for a full list of TAU_OPTIONS.

…

# Selective Instrumentation File Format

- To use an instrumentation specification file for source instrumentation:

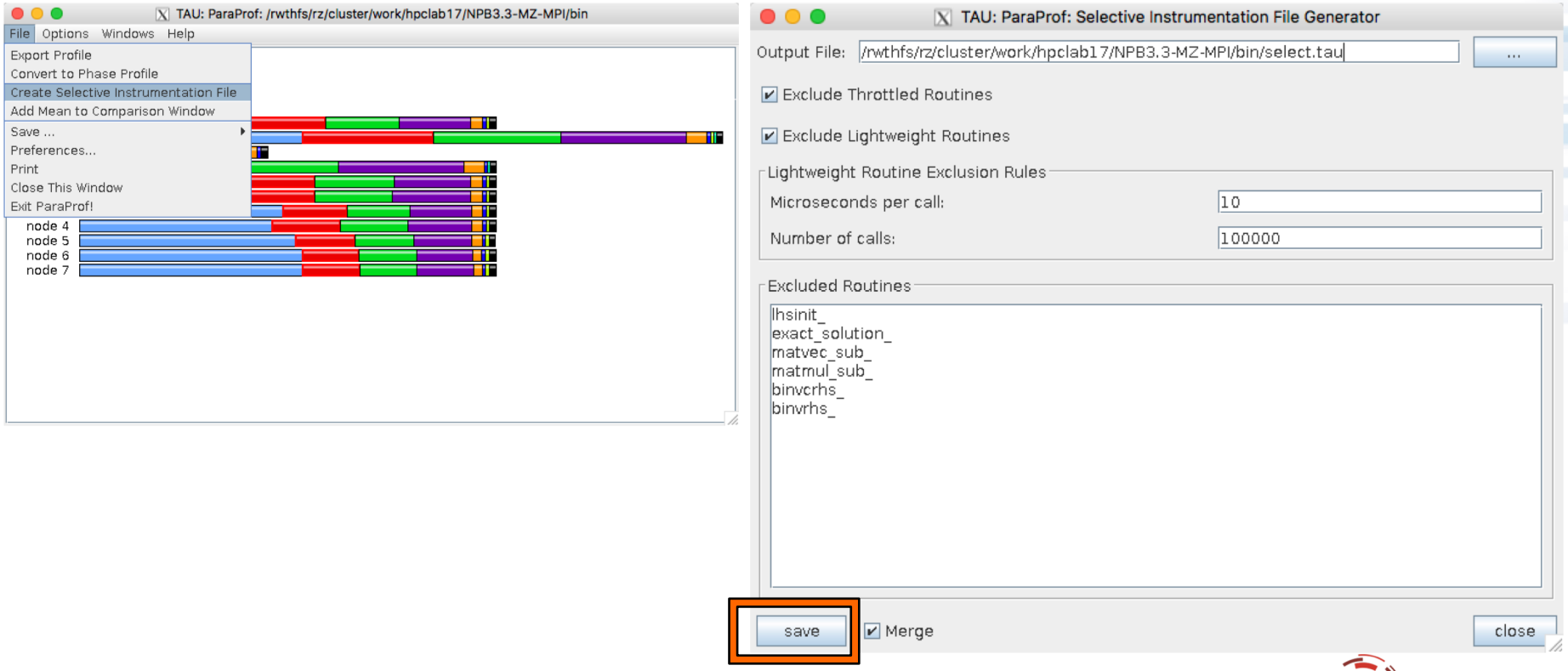
  % export TAU_OPTIONS= '-optTauSelectFile=/path/to/select.tau –optVerbose '

  % cat select.tau

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

```
NOTE: paraprof can create this file from an earlier execugtion for you.
File -> Create Selective Instrumentation File -> save
Selective instrumentation at runtime:
% export TAU_SELECT_FILE=select.tau
```

# Create a Selective Instrumentation File, Re-instrument, Re-run



## Installing and Configuring TAU

- Installing PDT:
  - wget tau.uoregon.edu/pdt_lite.tgz
  - ./configure –prefix=<dir>; make ; make install
- Installing TAU:
  - wget tau.uoregon.edu/tau.tgz; tar zxf tau.tgz; cd tau-2.<ver>
  - wget http://tau.uoregon.edu/ext.tgz ; tar xf ext.tgz
  - ./configure -bfd=download -pdt=<dir> -papi=<dir> -mpi –pthread –c++=mpicxx –cc=mpicc –fortran=mpif90 –dwarf=download –unwind=download –otf=download –arch=craycnl –iowrapper –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

# Compile-Time Options

•Optional parameters for the TAU_OPTIONS environment variable:
% tau_compiler.sh

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

# Compile-Time Options (contd.)

•Optional parameters for the TAU_OPTIONS environment variable:
% tau_compiler.sh

| | |
|---|---|
| -optCompile="" | Options passed to the compiler. Typically $(TAU_MPI_INCLUDE) $(TAU_INCLUDE) $(TAU_DEFS) |
| -optPdtF95Opts="" | Add options for Fortran parser in PDT (f95parse/gfparse) … |
| -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.

…

# TAU's 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>) |

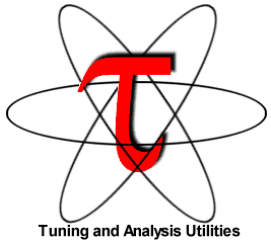
# Runtime Environment Variables

| Environment Variable | Default | Description |
|---|---|---|
| TAU_TRACE | 0 | Setting to 1 turns on tracing |
| TAU_TRACE_FORMAT | Default | Setting to “otf2” turns on TAU’s native OTF2 trace generation (configure with –otf=download) |
| TAU_EBS_UNWIND | 0 | Setting to 1 turns on unwinding the callstack during sampling (use with tau_exec –ebs or TAU_SAMPLING=1) |
| TAU_EBS_RESOLUTION | line | Setting to “function” or “file” changes the sampling resolution to function or file level respectively. |
| TAU_TRACK_LOAD | 0 | Setting to 1 tracks system load on the node |
| TAU_SELECT_FILE | Default | Setting to a file name, enables selective instrumentation based on exclude/include lists specified in the file. |
| TAU_OMPT_SUPPORT_LEVEL | basic | Setting to “full” improves resolution of OMPT TR6 regions on threads 1.. N-1. Also, “lowoverhead” option is available. |
| TAU_OMPT_RESOLVE_ADDRESS_EAGERLY | 1 | Setting to 1 is necessary for event based sampling to resolve addresses with OMPT. Setting to 0 allows the user to do offline address translation. |

# Runtime Environment Variables

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

# Download TAU from U. Oregon




http://tau.uoregon.edu

http://taucommander.com

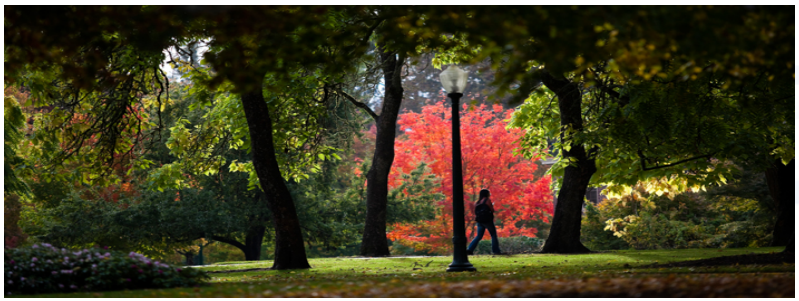
http://www.hpclinux.com [OVA for VirtualBox]

https://e4s.io [ Extreme-Scale Scientific Software Stack, Containers for HPC]

for more information

Free download, open source, BSD license

# PRL, University of Oregon, Eugene






www.uoregon.edu

# Support Acknowledgements

- US Department of Energy (DOE)
  - ANL
  - Office of Science contracts, ECP
  - SciDAC, LBL contracts
  - LLNL-LANL-SNL ASC/NNSA contract
  - Battelle, PNNL and ORNL contract
- Department of Defense (DoD)
  - PETTT, HPCMP
- National Science Foundation (NSF)
  - SI2-SSI, Glassbox
- NASA
- CEA, France
- Partners:
  - –University of Oregon
  - –The Ohio State University
  - –ParaTools, Inc.
  - –University of Tennessee, Knoxville
  - –T.U. Dresden, GWT
  - –Jülich Supercomputing Center


# Acknowledgement

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