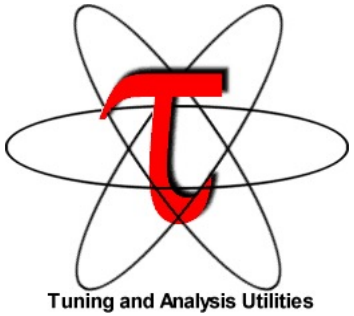
TAU Performance System®



Sameer Shende
sameer@cs.uoregon.edu
University of Oregon

http://tau.uoregon.edu/TAU_TW38_handson.pdf

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
  - …

Installing TAU on your laptop for paraprof (GUI)

- Microsoft Windows
  - Install Java from Oracle.com
  - http://tau.uoregon.edu/tau.exe
  - Install, click on a ppk file to launch paraprof
- macOS (x86_64)
  - Install Java 11.0.3:
    - Download and install http://tau.uoregon.edu/java.dmg
    - If you have multiple Java installations, add to your ~/.zshrc (or ~/.bashrc as appropriate):
    - export PATH=/Library/Java/JavaVirtualMachines/jdk-11.0.3.jdk/Contents/Home/bin:$PATH
    - java -version
  - Download and install TAU (copy to /Applications from dmg):
    - http://tau.uoregon.edu/tau.dmg
    - export PATH=/Applications/TAU/tau/apple/bin:$PATH
    - paraprof app.ppk &
- macOS (arm64, M1)
  - http://tau.uoregon.edu/java_arm64.dmg
  - http://tau.uoregon.edu/tau_arm64.dmg
- Linux (http://tau.uoregon.edu/tau.tgz)
  - ./configure; make install; export PATH=<taudir>/x86_64/bin:$PATH; paraprof app.ppk &

Using TAU on IvyMUC at LRZ

- Setup preferred program environment compilers using Intel compilers with OpenMP

```
% ssh -Y lxlogin10.lrz.de -l <USER>
See https://doku.lrz.de/display/PUBLIC/Linux+Cluster
Add to your ~/.bashrc:
# User specific aliases and functions
module use /lrz/sys/courses/vihps/modulefiles/
module load tau
module load salloc_conf/ivymuc

% tar zxf /lrz/sys/courses/vihps/material/NPB3.3-MZ-MPI.tar.gz
% cd NPB3.3-MZ-MPI; make clean; make suite; cd bin; cp ../jobscript/ivymuc/* .
% sbatch --reservation=hhps1s21_workshop ./tau.1.sbatch
(and next tau.2.sbatch, and 3)
% scp bt_mz.1.ppk <laptop>:  ; paraprof bt_mz.1.ppk
% pprof -a -m | more
```

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 is ready to “make” one or more of the benchmarks
  - but config/make.def may first need to be adjusted to specify appropriate compiler flags

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          # Intel compiler

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

# Alternative variants to perform instrumentation
...

#MPIF77 = scorep --user  mpiifort
...
```

Uncomment COMPILER flags according to current environment

Default (no instrumentation)

Hint: uncomment a compiler wrapper to do 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 Archer:       *
 *       make bt-mz CLASS=C NPROCS=16                           *
 ***********************************************************
```

- Type "make" for instructions

Building an NPB-MZ-MPI Benchmark

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

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

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 5 threads should be reasonable for 2 compute node of IvyMUC
  - bt-mz_C.16 should run in 50 seconds with the Intel + IntelMPI toolchain

NPB-MZ-MPI / BT Reference Execution

```
% cd bin
% cp ../jobscript/ivymuc/reference.sbatch .
% less reference.sbatch
% sbatch <--reservation=hhps1s21_workshop> reference.sbatch

% cat npb_btmz.o<job_id>
 NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP Benchmark
 Number of zones:  16 x  16
 Iterations:  200    dt:    0.000100
 Number of active processes:     16
 Use the default load factors with threads
 Total number of threads:    40 (  5.0 threads/process)

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

 BT-MZ Benchmark Completed.
 Time in seconds = 5.85
```

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

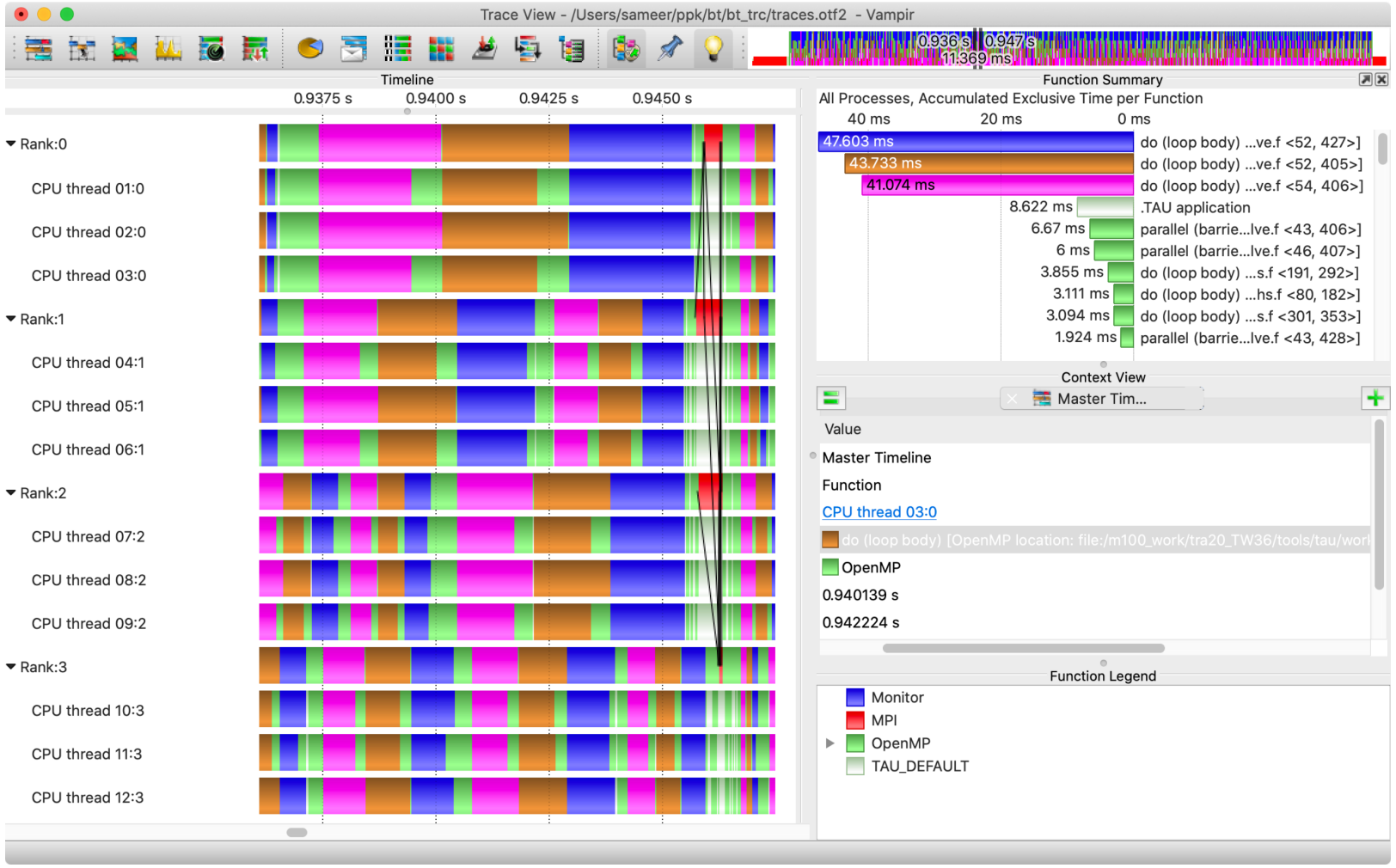
Note: the thread binding settings (`OMP_PROC_BIND`, `OMP_PLACES`) in the batch script are specific to the Intel toolchain!

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

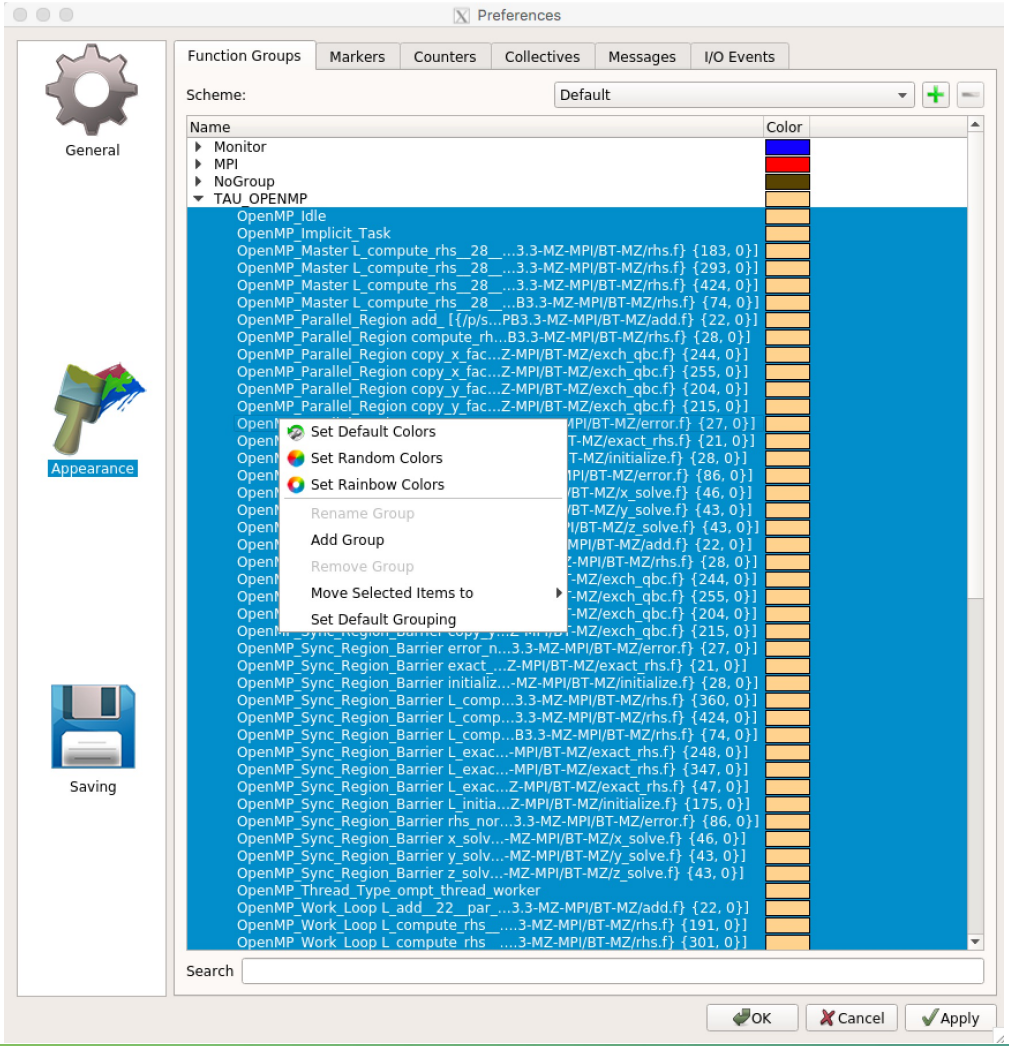
NPB-MZ-MPI / BT Execution with TAU

```
% cp ../jobscript/ivymuc/tau*.sbatch .
% sbatch --reservation=hhps1s21_workshop ./tau.1.sbatch
% paraprof --pack bt1.ppk  (if it doesn't exist)
<SCP file bt1.ppk to your laptop>
% paraprof bt1.ppk &
…
% sbatch --reservation=hhps1s21_workshop ./tau.2.sbatch
% module load vampir  (or take traces* to a machine with Vampir)
% vampir traces.otf2 &
```

TAU's OTF2 BT.C trace in Vampir [TU Dresden]

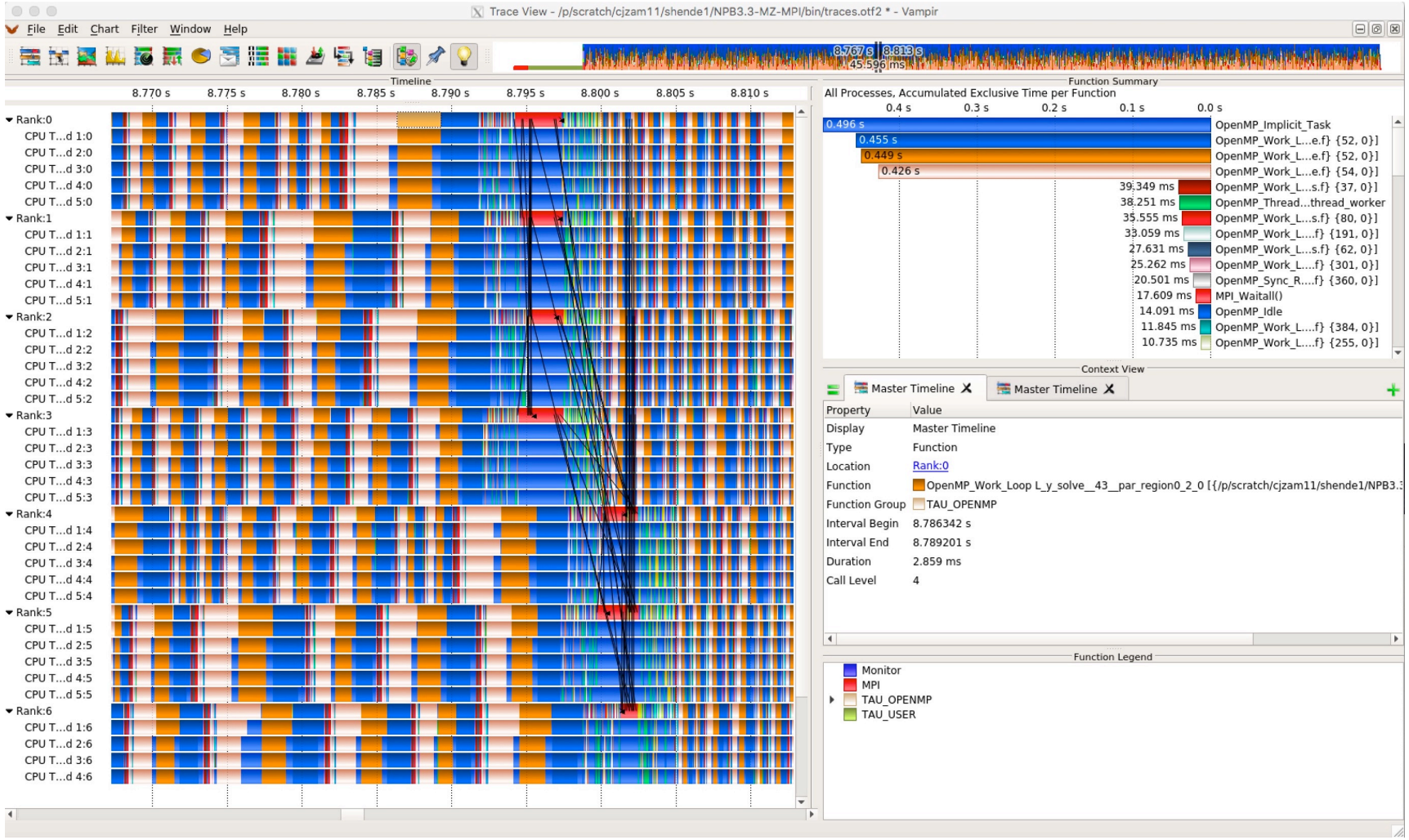


Change Colors for Events in Vampir [TU Dresden]



- File -> Preferences -> Appearance
- Multi-select functions under TAU_OPENMP
- Select “Set Random Colors”
- Click Apply, OK

Vampir [TU Dresden] Provides a Timeline Display



TAU generates OTF2 traces

% cat run.pbs
…
export TAU_TRACE=1
export TAU_TRACE_FORMAT=otf2

CoMD Example

```
MPI:
% cp -r /dss/dsshome1/08/hpckurs03/workshop.tgz.; tar xf workshop.tgz
% cd workshop/CoMD/src-mpi
% make
% cd ../bin
% sbatch --reservation= tau.1.sbatch
% paraprof --pack comd.1.ppk ; <SCP comd1.ppk laptop:>
% pprof -a -m | more
% paraprof comd.1.ppk &  (on your laptop)

MPI+OpenMP:
% cd ../src-openmp; make clean; make; cd ../bin
% sbatch -reservation=hhps1s21_workshop ./tau_ompt.1.sbatch;
% paraprof --pack comd_ompt.1.ppk;
<SCP file comd_ompt.1.ppk to your laptop>
% paraprof comd_ompt.1.ppk &
…
```

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>)

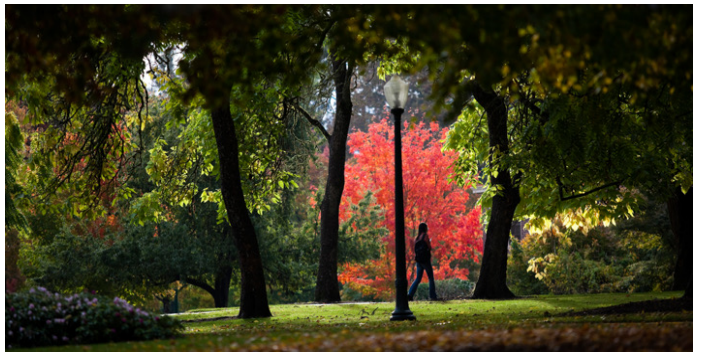
Runtime Environment Variables

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	0	Setting to 1 is necessary for event based sampling to resolve addresses with OMPT TR6 (-ompt=download-tr6)

Runtime Environment Variables

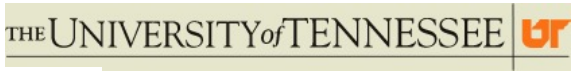
Environment Variable	Default	Description
TAU_TRACK_MEMORY_LEAKS	0	Tracks allocates that were not de-allocated (needs –optMemDbg or tau_exec –memory)
TAU_EBS_SOURCE	TIME	Allows using PAPI hardware counters for periodic interrupts for EBS (e.g., TAU_EBS_SOURCE=PAPI_TOT_INS when TAU_SAMPLING=1)
TAU_EBS_PERIOD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

Performance Research Lab, University of Oregon, Eugene, USA



Support Acknowledgments

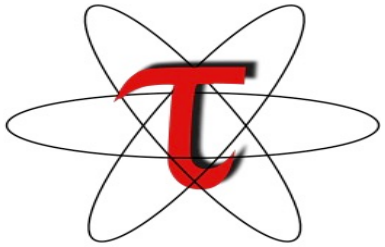
- US Department of Energy (DOE)
  - Office of Science contracts, ECP
  - SciDAC, LBL contracts
  - LLNL-LANL-SNL ASC/NNSA contract
  - Battelle, PNNL contract
  - ANL, ORNL contract
- Department of Defense (DoD)
  - PETTT, HPCMP
- National Science Foundation (NSF)
  - Glassbox, SI-2
- NASA
- CEA, France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

Download TAU from U. Oregon



http://tau.uoregon.edu

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

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

Free download, open source, BSD license