TAU Performance System®

Sameer Shende
University of Oregon and ParaTools, Inc.

MVAPICH Users Group Conference
OSU Translational Data Analytics Institute (TDAI), Pomerene Hall, Room #320

Monday, August 21, 2023, 9:30 – 10:30 am
http://mug.mvapich.cse.ohio-state.edu

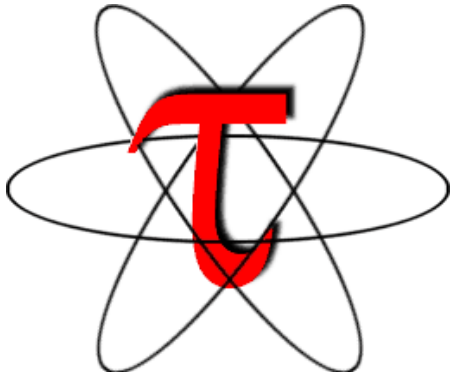
Slides:
http://tau.uoregon.edu/TAU_Tut_MUG23.pdf
Signup for AWS: https://e4s.io/tutorial

UNIVERSITY OF OREGON

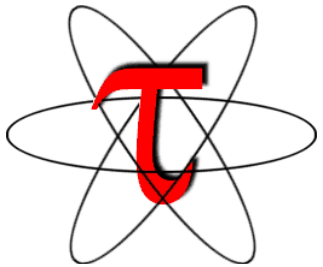
ParaTools

Acknowledgments

- **The MVAPICH2 team The Ohio State University**
  - http://mvapich.cse.ohio-state.edu
- **TAU team at the University of Oregon**
  - http://tau.uoregon.edu



TAU Performance System®



- **Tuning and Analysis Utilities (25+ year project)**
- **Comprehensive performance profiling and tracing**
  - Integrated, scalable, flexible, portable
  - Targets all parallel programming/execution paradigms
- **Integrated performance toolkit**
  - Instrumentation, measurement, analysis, visualization
  - Widely-ported performance profiling / tracing system
  - Performance data management and data mining
  - Open source (BSD-style license)
  - Uses performance and control variables to interface with MVAPICH2
- **Integrates with application frameworks**
- **http://tau.uoregon.edu**

TAU Performance System

Instrumentation

- Fortran, C++, C, UPC, Java, Python, Chapel, Spark
- Automatic instrumentation

Measurement and analysis support

- MPI, OpenSHMEM, ARMCI, PGAS, DMAPP, uGNI
- pthreads, OpenMP, OMPT interface, hybrid, other thread models
- GPU: CUDA, OpenCL, Level Zero, ROCm, OpenACC
- Parallel profiling and tracing
- Interfaces with OTF2 and Score-P

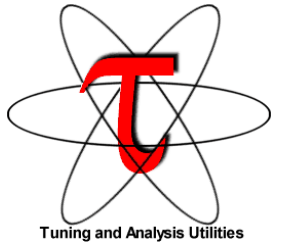
Analysis

- Parallel profile analysis (ParaProf), data mining (PerfExplorer)
- Performance database technology (TAUdb)
- 3D profile browser

Understanding Application Performance using TAU

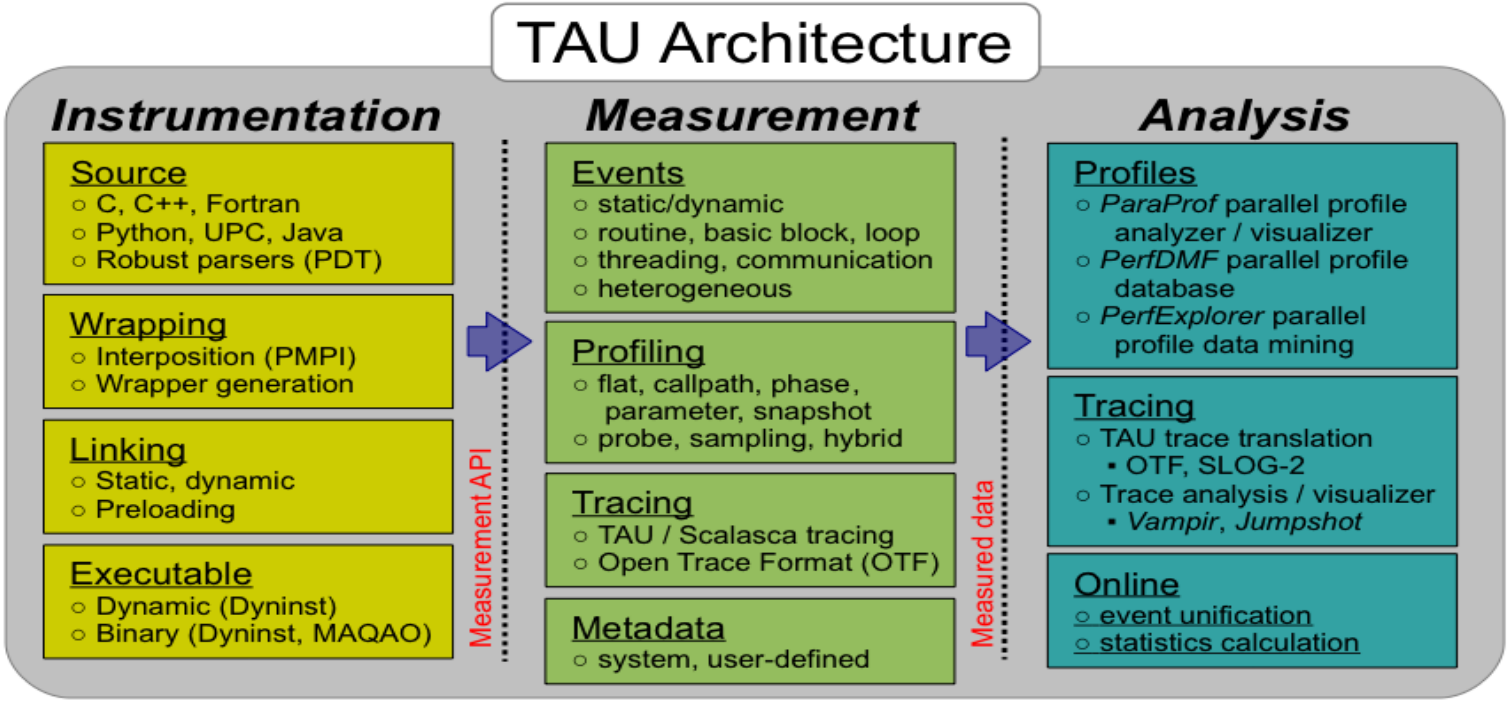
- **How much time** is spent in each application routine and outer *loops*? Within loops, what is the contribution of each *statement*? What is the time spent in OpenMP loops? In kernels on GPUs. How long did it take to transfer data between host and device (GPU)?
- **How many instructions** are executed in these code regions?
  Floating point, Level 1 and 2 *data cache misses*, hits, branches taken?
- **How much time did my application spend waiting at a barrier in MPI collective operations?**
- **What is the memory usage** of the code? When and where is memory allocated/de-allocated? Are there any memory leaks?
- **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?
- **How can I tune MPI for better performance?** What performance and control does MVAPICH2 export to observe and control its performance?

TAU Performance System®



Parallel performance framework and toolkit

- Supports all HPC platforms, compilers, runtime system
- Provides portable instrumentation, measurement, analysis



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 **mpirun tau_exec ./app** with options.
- Requires dynamic executable.

Simplifying the use of TAU!

Uninstrumented code:

- % module load mvapich2
- % make
- % mpirun -np 64 ./a.out

With TAU using event-based sampling (EBS):

- % mpirun –np 64 tau_exec –T mvapich2 –ebs ./a.out
- % paraprof (GUI)
- % pprof –a | more

NOTE:

- Requires dynamic executables (-dynamic link flag on Cray XC systems).
- Source code should be compiled with –g for access to symbol table.
- Replace srun with mpirun based on your appropriate launch command.

TAU Execution Command (tau_exec)

Uninstrumented execution

- % mpirun -np 256 ./a.out

Track GPU operations

- % mpirun -np 256 tau_exec –rocm ./a.out
- % mpirun -np 256 tau_exec –l0 ./a.out
- % mpirun -np 256 tau_exec –cupti ./a.out
- % mpirun -np 256 tau_exec –cupti -um ./a.out (for Unified Memory)
- % mpirun -np 256 tau_exec –opencl ./a.out
- % mpirun -np 256 tau_exec –openacc ./a.out

Track MPI performance

- % mpirun -np 256 tau_exec ./a.out

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

- % mpirun -np 256 tau_exec -io ./a.out

Track OpenMP and MPI execution (using OMPT for Intel v19)

- % export TAU_OMPT_SUPPORT_LEVEL=full;
  % mpirun -np 256 tau_exec –T ompt,v5,mpi -ompt ./a.out

Track memory operations

- % export TAU_TRACK_MEMORY_LEAKS=1
- % mpirun -np 256 tau_exec –memory_debug ./a.out (bounds check)

Use event based sampling (compile with –g)

- % mpirun -np 256 tau_exec –ebs ./a.out
- Also export TAU_METRICS=TIME,<PAPI_COUNTER> to use hardware perf. counters
- tau_exec -ebs_resolution=<file | function | line>

Configuration tags for tau_exec

```
% ls $TAU/Makefile*
/opt/tau/tau_latest/x86_64/lib/Makefile.tau-cupti-pdt
/opt/tau/tau_latest/x86_64/lib/Makefile.tau-intel-icpc-ompt-mpi-pdt-openmp
/opt/tau/tau_latest/x86_64/lib/Makefile.tau-mpc-mpc-mpi-pdt
/opt/tau/tau_latest/x86_64/lib/Makefile.tau-mpich-mpi-pdt
/opt/tau/tau_latest/x86_64/lib/Makefile.tau-mpi-cupti-pdt
/opt/tau/tau_latest/x86_64/lib/Makefile.tau-mpi-pdt
/opt/tau/tau_latest/x86_64/lib/Makefile.tau-mpi-pthread-adios2
/opt/tau/tau_latest/x86_64/lib/Makefile.tau-mpi-pthread-pdt
/opt/tau/tau_latest/x86_64/lib/Makefile.tau-mvapich2-mpi-pthread-pdt
/opt/tau/tau_latest/x86_64/lib/Makefile.tau-mvapich2-mpi-pthread-pdt-mpit
/opt/tau/tau_latest/x86_64/lib/Makefile.tau-openmpi-mpi-cupti-pdt
```

Each configuration of TAU creates a unique Stub Makefile (configuration file) that encodes paths to compilers and MPI. Choose an appropriate configuration/Makefile using tau_exec -T <tag>

```
% mpirun -np 4 tau_exec -T mvapich2,mpit -ebs  ./a.out
```

Configuration tags for tau_exec

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

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

% mpirun -np 256  tau_exec -T papi,mpi,pdt ./a.out
```
Preloads `$TAU/shared-papi-mpi-pdt/libTAU.so`
```
% mpirun -np 256  tau_exec -T papi ./a.out
```
Preloads `$TAU/shared-papi-mpi-pdt/libTAU.so` by matching.
```
% mpirun -np 256 tau_exec -T papi,mpi,pdt -s ./a.out
```
Does not execute the program. Just displays the library that it will preload if executed without the –s option.

NOTE: -mpi configuration is selected by default. Use –T serial for Sequential programs.

Setup: Installing TAU on Laptops

Prerequisites: Java in your path

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/M2)

  - http://tau.uoregon.edu/java_arm64.dmg
  - http://tau.uoregon.edu/tau_arm64.dmg

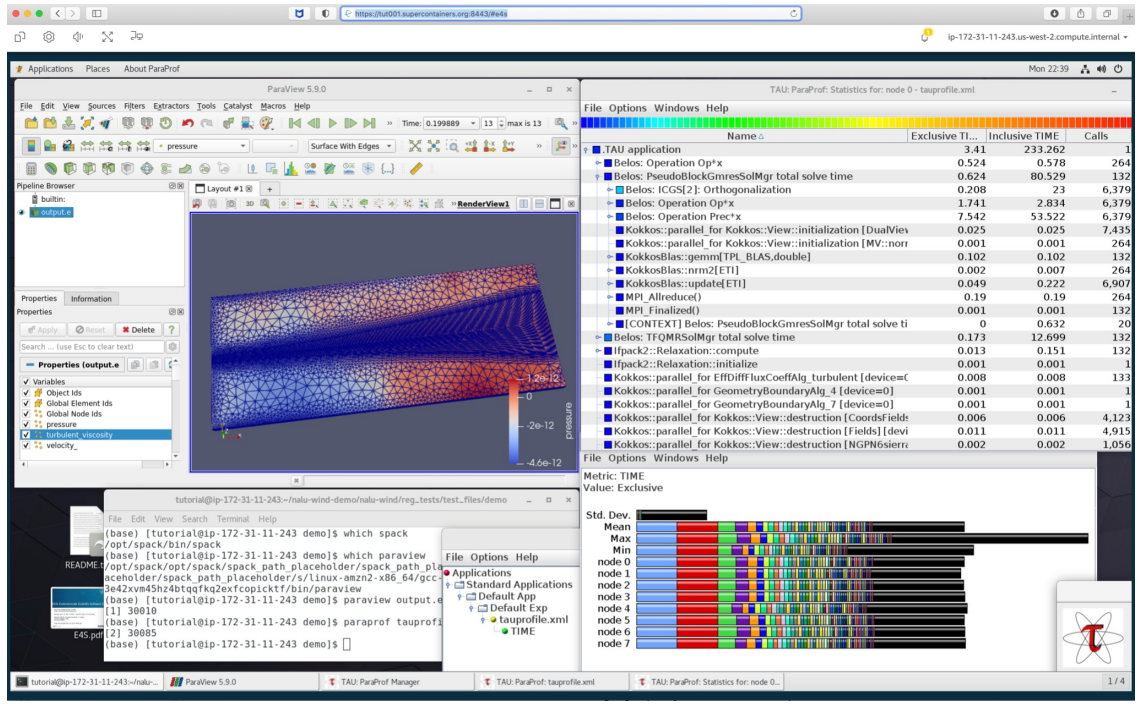
Linux (http://tau.uoregon.edu/tau.tgz)

- ./configure; make install; export PATH=<taudir>/x86_64/bin:$PATH; paraprof app.ppk &

Setup: AWS

Reserve an instance at https://e4s.io/tutorial
e.g., if you reserved tut011, then use:

https://tut011.supercontainers.org:8443/#e4s
login: tutorial
password: HPCLinux12!
Click on gnome-terminal icon on the desktop to start.
cat ~/README



Demo: Event-based Sampling, MPI-T, and ParaProf

cd ~/SRC/demo/CoMD/bin

./rmv2.sh

paraprof &

Another demo:

cd ~/SRC/demo

paraprof demo.ppk &

Demo: allreduce: Using different collective algorithms

```
cd ~/SRC/demo/allreduce
./run.sh
ls  *.ppk
% paraprof *.ppk &
```

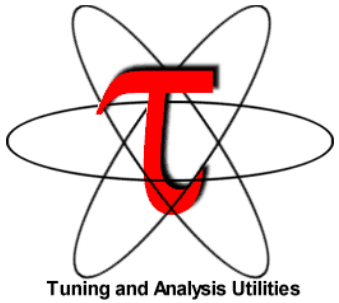
alltoall on AWS: Using different collective algorithms

```
cd ~/SRC/demo/alltoall
./run.sh
ls  *.ppk
% paraprof *.ppk &
```

3DStencil on AWS: Using TAU's MPI_T auto-tuning plugins

```
cd ~/SRC/demo/3Dstencil
./run.sh
ls  *.ppk
paraprof *.ppk &
```

Download TAU from U. Oregon



http://tau.uoregon.edu

http://taucommander.com

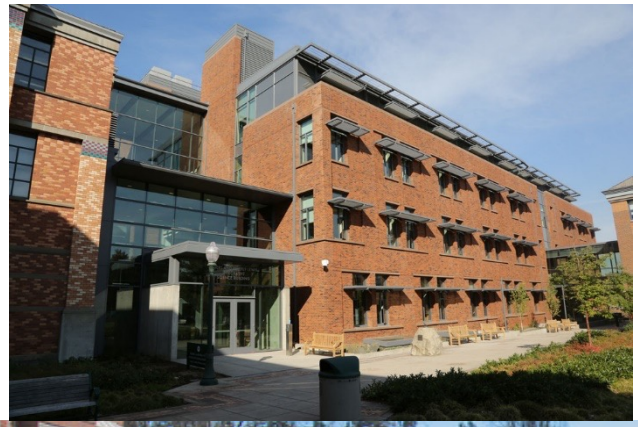
http://www.hpclinux.com [OVA for VirtualBox]

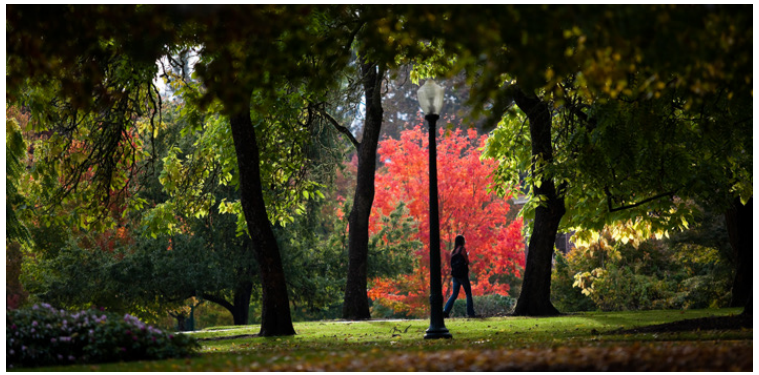
https://e4s.io [Extreme-scale Scientific Software Stack]

for more information

Free download, open source, BSD license

PRL, OACISS, University of Oregon, Eugene





www.uoregon.edu

Support Acknowledgments

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

CEA, France

Department of Defense (DoD)

- PETTT, HPCMP

National Science Foundation (NSF)

- SI2-SSI, Glassbox, CSSI

NASA

AMD, AWS, Broadcom, Google, IBM, Intel, NVIDIA, OCI

Partners:

- University of Oregon
- The Ohio State University
- ParaTools, Inc.
- University of Tennessee, Knoxville
- T.U. Dresden, GWT
- Jülich Supercomputing Center

Acknowledgment

“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.”

Reference

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 –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 (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 $(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

-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