TAU PERFORMANCE ANALYSIS

SAMEER SHENDE
Director, Performance Research Lab, University of Oregon
ParaTools, Inc.
sameer@cs.uoregon.edu

5/3/2017 10am – 10:45am. ALCF, Building 240 conference room, ANL
TAU PERFORMANCE SYSTEM®

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

**TAU Architecture**

**Instrumentation**
- **Source**
  - C, C++, Fortran
  - Python, UPC, Java
  - Robust parsers (PDT)

- **Wrapping**
  - Interposition (PMPI)
  - Wrapper generation

- **Linking**
  - Static, dynamic
  - Preloading

- **Executable**
  - Dynamic (Dyninst)
  - Binary (Dyninst, MAQAO)

**Measurement**
- **Events**
  - static/dynamic
  - routine, basic block, loop
  - threading, communication
  - heterogeneous

- **Profiling**
  - flat, callpath, phase, parameter, snapshot
  - probe, sampling, hybrid

- **Tracing**
  - TAU / Scalasca tracing
  - Open Trace Format (OTF)

- **Metadata**
  - system, user-defined

**Analysis**
- **Profiles**
  - ParaProf parallel profile analyzer / visualizer
  - PerfDMF parallel profile database
  - PerfExplorer parallel profile data mining

- **Tracing**
  - TAU trace translation
    - OTF, SLOG-2
  - Trace analysis / visualizer
    - Vampir, Jumpshot

- **Online**
  - event unification
  - statistics calculation
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

- **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).
SIMPLIFYING TAU’S USAGE (TAU_EXEC)

- Uninstrumented execution
  - % mpirun -np 64 ./a.out

- Track MPI performance
  - % mpirun -np 64 tau_exec ./a.out

- Use event based sampling (compile with -g)
  - % mpirun -np 64 tau_exec -ebs ./a.out
  - Also -ebs_source=<PAPI_COUNTER> -ebs_period=<overflow_count>

- Track POSIX I/O and MPI performance (MPI enabled by default)
  - % mpirun -np 64 tau_exec -T mpi,pdt,papi -io ./a.out

- Track OpenMP runtime routines
  - % mpirun -np 64 tau_exec -T ompt,pdt,mpi -ompt ./a.out

- Track memory operations
  - % export TAU_TRACK_MEMORY_LEAKS=1
  - % mpirun -np 64 tau_exec -memory_debug ./a.out (bounds check)

- Load wrapper interposition library
  - % mpirun -np 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
HANDS-ON
NPB 3.3 MZ

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

```bash
% mkdir /lus/theta-fs0/projects/Comp_Perf_Workshop/$USER
% cd !$; tar zxf /soft/perftools/tau/workshop.tgz
% module load tau
% cd MZ-NPB3.3-MPI; cat README
% make clean
% make suite
% cd bin
In a second window:
% qsub -I -n 1 -A Comp_Perf_Workshop -t 50 -q cache-quad
% cd bin; module load tau
% export OMP_NUM_THREADS=4
% aprun -n 16 ./bt-mz.B.16
% aprun -n 16 tau_exec -T ompt,mpi,pdt -ompt -ebs ./bt-mz.B.16
% paraprof --pack ex1.ppk
In the first window:
% paraprof ex1.ppk &
```
The NAS Parallel Benchmark suite (MPI+OpenMP version)
   - Available from:
     [http://www.nas.nasa.gov/Software/NPB](http://www.nas.nasa.gov/Software/NPB)
   - 3 benchmarks in Fortran77
   - Configurable for various sizes & classes

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
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
#                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
...
BUILDING AN NPB-MZ-MPI BENCHMARK

To make a NAS multi-zone benchmark type

```bash
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

[...]
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 can enable event based sampling while launching the executable using env

TAU_SAMPLING=1 or tau_exec -ebs
EVENT BASED SAMPLING WITH TAU

- Launch paraprof

```bash
% cd MZ-NPB3.3-MPI; cat README
% make clean;
% make suite
% cd bin
% qsub -I -n 1 -A Comp_Perf_Workshop -t 50 -q cache-quad
% export OMP_NUM_THREADS=4
% aprun -n 16 tau_exec -T ompt -ebs ./bt-mz.B.16
% On head node:
% module load tau
% paraprof
```

- Right Click on Node 0 and choose Show Thread Statistics Table
Click on Columns: to sort by incl time
Open binvcrhs
Click on Sample
<table>
<thead>
<tr>
<th>Name</th>
<th>Excl. Time</th>
<th>Incl. Time</th>
<th>Calls</th>
<th>Child Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>[SUMMARY]</td>
<td>(timefmt)</td>
<td>(timefmt)</td>
<td>9,457</td>
<td>6,398</td>
</tr>
<tr>
<td>[SAMPLE]</td>
<td>(timefmt)</td>
<td>(timefmt)</td>
<td>0.04</td>
<td>0.04</td>
</tr>
<tr>
<td>[SAMPLE]</td>
<td>(timefmt)</td>
<td>(timefmt)</td>
<td>0.11</td>
<td>0.11</td>
</tr>
<tr>
<td>[SAMPLE]</td>
<td>(timefmt)</td>
<td>(timefmt)</td>
<td>0.08</td>
<td>0.08</td>
</tr>
<tr>
<td>[SAMPLE]</td>
<td>(timefmt)</td>
<td>(timefmt)</td>
<td>0.06</td>
<td>0.06</td>
</tr>
<tr>
<td>[SAMPLE]</td>
<td>(timefmt)</td>
<td>(timefmt)</td>
<td>0.04</td>
<td>0.04</td>
</tr>
<tr>
<td>[SAMPLE]</td>
<td>(timefmt)</td>
<td>(timefmt)</td>
<td>0.04</td>
<td>0.04</td>
</tr>
<tr>
<td>[SAMPLE]</td>
<td>(timefmt)</td>
<td>(timefmt)</td>
<td>0.04</td>
<td>0.04</td>
</tr>
<tr>
<td>[SAMPLE]</td>
<td>(timefmt)</td>
<td>(timefmt)</td>
<td>0.04</td>
<td>0.04</td>
</tr>
<tr>
<td>[SAMPLE]</td>
<td>(timefmt)</td>
<td>(timefmt)</td>
<td>0.04</td>
<td>0.04</td>
</tr>
<tr>
<td>[SAMPLE]</td>
<td>(timefmt)</td>
<td>(timefmt)</td>
<td>0.04</td>
<td>0.04</td>
</tr>
<tr>
<td>[SAMPLE]</td>
<td>(timefmt)</td>
<td>(timefmt)</td>
<td>0.03</td>
<td>0.03</td>
</tr>
<tr>
<td>[SAMPLE]</td>
<td>(timefmt)</td>
<td>(timefmt)</td>
<td>0.03</td>
<td>0.03</td>
</tr>
<tr>
<td>[SAMPLE]</td>
<td>(timefmt)</td>
<td>(timefmt)</td>
<td>0.03</td>
<td>0.03</td>
</tr>
<tr>
<td>[SAMPLE]</td>
<td>(timefmt)</td>
<td>(timefmt)</td>
<td>0.03</td>
<td>0.03</td>
</tr>
<tr>
<td>[SAMPLE]</td>
<td>(timefmt)</td>
<td>(timefmt)</td>
<td>0.03</td>
<td>0.03</td>
</tr>
<tr>
<td>[SAMPLE]</td>
<td>(timefmt)</td>
<td>(timefmt)</td>
<td>0.03</td>
<td>0.03</td>
</tr>
<tr>
<td>[SAMPLE]</td>
<td>(timefmt)</td>
<td>(timefmt)</td>
<td>0.03</td>
<td>0.03</td>
</tr>
<tr>
<td>[SAMPLE]</td>
<td>(timefmt)</td>
<td>(timefmt)</td>
<td>0.03</td>
<td>0.03</td>
</tr>
<tr>
<td>[SAMPLE]</td>
<td>(timefmt)</td>
<td>(timefmt)</td>
<td>0.03</td>
<td>0.03</td>
</tr>
<tr>
<td>[SAMPLE]</td>
<td>(timefmt)</td>
<td>(timefmt)</td>
<td>0.03</td>
<td>0.03</td>
</tr>
<tr>
<td>[SAMPLE]</td>
<td>(timefmt)</td>
<td>(timefmt)</td>
<td>0.03</td>
<td>0.03</td>
</tr>
<tr>
<td>[SAMPLE]</td>
<td>(timefmt)</td>
<td>(timefmt)</td>
<td>0.03</td>
<td>0.03</td>
</tr>
</tbody>
</table>
TAU SOURCE INSTRUMENTATION

- Choose TAU configuration
  ```bash
  module load tau;
  export TAU_MAKEFILE=$TAU/Makefile.tau-intel-papi-ompt-mpi-pdt-openmp
  ```
- 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
CREATE A SELECTIVE INSTRUMENTATION FILE, RE-INSTRUMENT, RE-RUN
PARAPROF WITH OPTIMIZED INSTRUMENTATION
3D VISUALIZATION WITH PARAPROF
PARAPROF: NODE 0

- Optimized instrumentation!
SOURCE INSTRUMENTATION
TAU’S STATIC ANALYSIS SYSTEM: PROGRAM DATABASE TOOLKIT (PDT)

Application / Library

C / C++ parser

Fortran parser F77/90/95

IL analyzer

C / C++ IL analyzer

Fortran IL analyzer

Program Database Files

DUCTAPE

TAU instrumentor

Automatic source instrumentation
PDT: AUTOMATIC SOURCE INSTRUMENTATION

- TAU source analyzer
- Parsed program

- Application source

- tau_instrumentor

- Instrumentation specification file

- Instrumented copy of source
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:
  % module load UNITE tau

  % 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:
  - Download Java
  - http://tau.uoregon.edu/java.dmg
  - Install java.dmg
  - 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.*


For an MPI+OpenMP+F90 application with Intel MPI, you may choose
Makefile.tau-intel-papi-ompt-mpi-pdt-openmp
  - Supports MPI instrumentation & PDT for automatic source instrumentation

% export TAU_MAKEFILE=$TAU/Makefile.tau-intel-papi-ompt-mpi-pdt-openmp
% tau_f90.sh app.f90 -o app; aprun -n 256 ./app; paraprof
Optional parameters for the TAU_OPTIONS environment variable:

% tau_compiler.sh

- **optVerbose**
  - Turn on verbose debugging messages

- **optComplInst**
  - Use compiler based instrumentation

- **optNoComplInst**
  - 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/gfpars) …
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= ‘-optCompInst -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/select.tau -optVerbose’
% cat select.tau

BEGIN_EXCLUDE_LIST
BINVCRHS
MATMUL_SUB
MATVEC_SUB
EXACT_SOLUTION
BINVRHS
LHS#INIT
TIMER_
END_EXCLUDE_LIST

NOTE: paraprof can create this file from an earlier execution for you.
File → Create Selective Instrumentation File → save
## RUNTIME ENVIRONMENT VARIABLES

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAU_TRACE</td>
<td>0</td>
<td>Setting to 1 turns on tracing</td>
</tr>
<tr>
<td>TAU_CALLPATH</td>
<td>0</td>
<td>Setting to 1 turns on callpath profiling</td>
</tr>
<tr>
<td>TAU_TRACK_MEMORY_FOOTPRINT</td>
<td>0</td>
<td>Setting to 1 turns on tracking memory usage by sampling periodically the resident set size and high water mark of memory usage</td>
</tr>
<tr>
<td>TAU_TRACK_POWER</td>
<td>0</td>
<td>Tracks power usage by sampling periodically.</td>
</tr>
<tr>
<td>TAU_CALLPATH_DEPTH</td>
<td>2</td>
<td>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)</td>
</tr>
<tr>
<td>TAU_SAMPLING</td>
<td>1</td>
<td>Setting to 1 enables event-based sampling.</td>
</tr>
<tr>
<td>TAU_TRACK_SIGNALS</td>
<td>0</td>
<td>Setting to 1 generate debugging callstack info when a program crashes</td>
</tr>
<tr>
<td>TAU_COMM_MATRIX</td>
<td>0</td>
<td>Setting to 1 generates communication matrix display using context events</td>
</tr>
<tr>
<td>TAU_THROTTLE</td>
<td>1</td>
<td>Setting to 0 turns off throttling. Throttles instrumentation in lightweight routines that are called frequently</td>
</tr>
<tr>
<td>TAU_THROTTLE_NUMCALLS</td>
<td>100000</td>
<td>Specifies the number of calls before testing for throttling</td>
</tr>
<tr>
<td>TAU_THROTTLE_PERCALL</td>
<td>10</td>
<td>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</td>
</tr>
<tr>
<td>TAU_CALLSITE</td>
<td>0</td>
<td>Setting to 1 enables callsite profiling that shows where an instrumented function was called. Also compatible with tracing.</td>
</tr>
<tr>
<td>TAU_PROFILE_FORMAT</td>
<td>Profile</td>
<td>Setting to “merged” generates a single file. “snapshot” generates xml format</td>
</tr>
<tr>
<td>TAU_METRICS</td>
<td>TIME</td>
<td>Setting to a comma separated list generates other metrics. (e.g., ENERGY,TIME,P_VIRTUAL_TIME,PAPI_FP_INS,PAPI_NATIVE_&lt;event&gt;:&lt;subevent&gt;)</td>
</tr>
</tbody>
</table>
### RUNTIME ENVIRONMENT VARIABLES (CONTD.)

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

http://www.hpclinux.com [OVA file]
http://tau.uoregon.edu
for more information

Free download, open source, BSD license
PRL, UNIVERSITY OF OREGON, EUGENE

www.uoregon.edu
SUPPORT ACKNOWLEDGMENTS

- US Department of Energy (DOE)
  - ANL
  - Office of Science contracts
  - 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
- Partners:
  - University of Oregon
  - The Ohio State University
  - ParaTools, Inc.
  - University of Tennessee, Knoxville
  - T.U. Dresden, GWT
  - Jülich Supercomputing Center
THANK YOU! QUESTIONS?