TAU Parallel Performance Profiling and Tracing Tool

Table of Contents

- TAU Installation information on Ranger and Kraken
- General guide for code instrumentation with TAU
  - Compilation, execution, and report generation
  - How to choose TAU Makefile Stub
- Guide for Navigating ParaProf and TAU Portal
- Reducing TAU Instrumentation Overhead
- Custom Profiling
  - Selective instrumentation
  - Callpath profiling
  - PAPI events profiling
  - Loop instrumentation
- Tracing
- References

What can expect from this documentation?

TAU is parallel profiling and tracing tool that offers advanced features for application characterization and is installed on TeraGrid large-scale systems.

This documentation provides TAU installation information on Ranger and Kraken; general and practical guide for using TAU.

It is intended to introduce the commonly used TAU features; provide recipes to successfully start using TAU; and provide quick references. TAU documentation and presentations need to be consulted for materials that are not covered in this page.

TAU Installation Information on Ranger and Kraken

On Ranger:

TAU is currently built with pgI7.2 using mvapich1.1_0.1, and intel10.1 using mvapich2.1_2 and mvapich1.1_0.1.

On Kraken:

TAU is currently built with PGI, PathScale, and GNU compilers, using mpich2.

To use TAU on Ranger or Kraken, first, load your intended User Environment and then load TAU with the module command. Instructions for loading the User Environment can be found in Ranger and Kraken’s User Guides.

Example on kraken:

% module avail PrgEnv (shows the available programming environments)

% module avail tau (shows the available versions of tau in your intended User Environment)

% module swap PrgEnv-pgi/2.2.41A PrgEnv-pathscale/2.2.41A
(swaps pgi with pathscale environment)

% module load tau/2.19 (loads tau-2.19 built with PathScale compiler)

**Example on Ranger:**

% module swap pgi intel (loads the default Intel User Environment)

% module avail tau (shows available versions)

% module load tau/2.18.3 (loads tau/2.18.3)

**Note** - Latest available tau version is always recommended to use. Recent versions support more features, and also environment variables setting methods at compile or run-time are backward compatible with the earlier versions of TAU.

General Guide for Code Instrumentation with TAU

Source instrumentation (automatic and manual), compiler directives, and binary rewriting (with Dyninst) are possible with TAU. **Only source auto instrumentation with PDT is explained here;** in this method, TAU inserts additional lines of code in the application using Program Database Toolkit (PDT). Instructions for source instrumentation are as follows:

**Step 1-- Source Automatic Instrumentation**

Idea in this step is to compile your application with TAU Compiler and Options.

- First, choose a suitable TAU Makefile stub from the list of available Makefiles (TAU stub Makefiles can be found in `<TAU-DIR>/craycnl/lib/Makefile* on kraken, and <TAU-DIR>/x86_64/lib/Makefile* installation directories on Ranger)

  See the section ‘**How to choose TAU Makefile stub**’ below for additional guide.

- setenv TAU_MAKEFILE <to_the_selected_tau_makefile_stub>
- setenv TAU_OPTIONS to one or more TAU options

  See [http://www.cs.uoregon.edu/research/tau/docs/newguide/re01.html](http://www.cs.uoregon.edu/research/tau/docs/newguide/re01.html)

  or ‘man tau_compiler.sh’ on the system for a list of TAU options.

Compile your code with one of the TAU scripts (tau_cc.sh, tau_f90.sh, and tau_cxx.sh for C, Fortran, or C++ code, respectively). TAU scripts use TAU_MAKEFILE to set the necessary environment variables for your source code instrumentation.
If ‘make’ is used in your build then you need to instruct the compiler and loader to use TAU script instead of regular compiler/loader commands. This can be as simple as using ‘% make CC=tau_cc.sh’ for a C application compilation, for example.

Example

% module load tau
% setenv TAU_MAKEFILE /sw/xt/tau/2.19/cnl2.2_pgi9.0.3/tau-2.19/craycnl/lib/Makefile.tau-papi-pdt-pgi
% setenv TAU_OPTIONS "-optVerbose -optKeepFiles"
% tau_cc.sh -o hello hello.c  (hello is an instrumented binary)

Step 2--Execution

- In this step, one just needs to submit the instrumented binary, generated in step 1, to the queue. Environment variables may be set to obtain a desired profiling. Information on environment variables is available from this link: http://www.cs.uoregon.edu/research/tau/docs/newguide/bk03apa.html#d0e1521

Note: on Ranger, one needs to load papi module for event counts (module load papi), as papi is a shared library and needs to be appended to LD_LIBRARY_PATH.

Successful execution in this step generates profile* files.

Step 3--Profiling Report

Profiling report can be generated with execution of % pprof for text; % paraprof or % perfexplorer for GUI data in the directory that profile* files reside. Pprof,paraprof, perfexplorer utilities are available in TAU’s bin directory. Bin directory can be added to the PATH (eg, set path=(<tau_path>/bin $path), or ‘module load tau’ sets the path to TAU bin directory.

-h provides options for pprof and paraprof as:
% Pprof –h, % paraprof –h, or %perfexplorer –h

Example of GUI data executing %paraprof:
To view the GUI profiling on Windows workstation:

Pack the profiles on remote machine where the profile files reside with

```shell
%paraprof --pack app.ppk
```

Transfer the app.ppk file to your workstation. Then clicking on app.ppk will start the GUI.

Viewing the visual report on the workstation is faster, but Java (from sun.com) and TAU should be installed first. Windows version of TAU is available from downloads:

[http://www.cs.uoregon.edu/research/tau/home.php](http://www.cs.uoregon.edu/research/tau/home.php)

**How to choose TAU Makefile stub:**

Selecting the right Makefile stub is a key step in TAU instrumentation as it sets the necessary environment variables for your source code instrumentation. First, determine the type of measurement that interests you. Then select the TAU Makefile stub that has your intended measurement words as combination of compiler, mpi and/or openMP, and others. Pay attention to the following keys and their meanings in your stub selection.

**pdt**: auto-instrumentor
Lack of pdt in a Makefile is intended for other methods such as compiler instrumentation.

**papi:** to obtain PAPI hardware events

**phase:** for phase profiling

**trace:** for trace generation

**MPI:** for MPI program profiling

**OpenMP:** for OpenMP program profiling

Lack of words MPI and OpenMP in the stub Makefile is meant for serial program.

**MPI+OpenMP:** for hybrid code instrumentation

**Opapi:** tool for automatic instrumentation of OpenMP directives

**pgi:** pgi compiler will be used for compilation

**Sun:** for Sun compiler

**Pathscale:** for Pathscale compiler

**Icpc:** for Intel compiler

**Example 1:**

`Makefile.tau-papi-mpi-pdt-openmp-opari-pgi`.

This TAU Makefile stub should be used for automatic instrumentation (words pdt and opari) in hybrid code (words mpi and openmp) for obtaining PAPI event counts with PGI compiler.

**Example 2:**

`Makefile.tau-pathscale-pdt`, for auto-instrumentation (pdt) of a serial code (no mpi or openmp) with Pathscale compiler.

---

**Guide for Navigating ParaProf and TAU Portal**

ParaProf provides many options to visualize and analyze the profiling data. Please see the link to “Navigating TAU Visual Displays using Paraprof and the TAU Portal,” from TeraGrid TAU page for an introductory guide.

---

**Reducing TAU Instrumentation Overhead**
Measuring performance by profiling or tracing can introduce overhead, thereby changing a program’s performance. The impact of measuring performance can be minimized by throttling.

**Throttling:**

By default, TAU throttles the tiny routines that are called more than 100,000 times, and each call takes less than 10 microsecond inclusive time. One can modify the TAU default throttling specification by setting the following environment variables at run-time:

```
Setenv TAU_THROTTLE 1  \ (TAU_THROTTLEL = 1 turns on and 0 turns off the throttling)
Setenv TAU_THROTTLE_NUMCALLS <n> \ (routines that are called more than n times in execution)
Setenv TAU_THROTTLE_PERCALL  <t> \ (routines’ execution time take less than t microsecond per call)
```

**Example:** Specification below causes the execution to not generate profiling data for routines that are called more than 1000 times and each call takes less than 50 microsecond.

Setenv TAU_THROTTLE 1  
Setenv TAU_THROTTLE_NUMCALLS 1000  
Setenv TAU_THROTTLE_PERCALL 50

---

**Custom Profiling**

1. **Selective Instrumentation**

User can selectively instrument routines of interest to reduce the instrumentation overhead and/or generate a custom profiling. Selected routines should be listed in a text file and used in compilation time. There are several ways to create the selective instrumentation text file.

   a) In ParaProf Manager window, choose **File menu, select Create Selective Instrumentation File**, **Specify the filtering criteria in selection window, Save the throttled routines as a file**, and use the throttled file in the compilation. **This method is visualized and demonstrated in the PowerPoint slides.** Link to the PowerPoint can be found in the TeraGrid TAU page (Navigating TAU Visual Display ....)

   b) Create a text file to exclude routines and/or files in the text file as shown in the example below:

```plaintext
BEGIN_EXCLUDE_LIST
int foo(int)
void Sequencer::threadRun(Sequencer *)
#this line is a comment and is ignored by the instrumentor
"#H5#"
END_EXCLUDE_LIST

BEGIN_FILE_EXCLUDE_LIST
foo.f90
bar_.c
END_FILE_EXCLUDE_LIST
```
# is wildcard for routines and * for file names. For leading # wildcard, place it in quotes as shown in the above example as # is used for comment character as well.

BEGIN_EXCLUDE_LIST / END_EXCLUDE_LIST and BEGIN_FILE_EXCLUDE_LIST/ END_FILE_EXCLUDE_LIST are directives for TAU instrumentor. For more information on directives see the link: http://www.cs.uoregon.edu/research/tau/docs/newguide/bk03ch01.html

Once the selective profiling text file (e.g., select.tau) in option a or b is generated, user needs to include it in the TAU_OPTIONS, before the compilation, as shown below:

```bash
setenv TAU_OPTIONS "-optVerbose -optKeepFiles -optPreProcess -optTauSelectFile=select.tau"
```

After the application compiled with the above flag, execution and profile report generation follows the general guide in step 2 and 3.

2. **Callpath profiling**
   - Instrument your code following the general guide.
   - Set the following variables at run-time and run your job.

```bash
setenv TAU_CALLPATH 1
setenv TAU_CALLPATHDEPTH 10 (default depth is 2)
```

Callpath profiling for matmul test code: shows where routines are called and how much time spent in each call, on rank 0.
3. **PAPI Events Profiling**

To obtain PAPI event counts with TAU

- setenv TAU_MAKEFILE stub to the one that has the word papi.
- Instrument your code following the general guide.
- At run-time, setenv TAU_METRICS to colon separated PAPI events
  
  **Example 1:**
  
  ```
  Setenv TAU_METRICS GET_TIME_OF_DAY:PAPI_FP_OPS:PAPI_TOT_CYC:PAPI_L1_DCM
  ```

  **Example 2:**
  
  ```
  Setenv TAU_METRICS LINUXTIMERS:PAPI_TOT_CYC:PAPI_TOT_INS:PAPI_FP_OPS:DISPATCH_STALLS
  ```

  The **first metric** in the colon separated list is time. Default timer is GET_TIME_OF_DAY, but on Linux and CrayCNL systems, TAU provides high resolution LINUXTIMERS metric. For more information on TAU timers see link: [http://www.cs.uoregon.edu/research/tau/docs/newguide/bk03ch01s04.html](http://www.cs.uoregon.edu/research/tau/docs/newguide/bk03ch01s04.html)

  **To find out** the available **PAPI events** on Ranger or Kraken, run `papi_avail` or `papi_native_avail` utilities in compute node (eg, aprun -n 12 /opt/xt-tools/papi/3.6.2.2/bin/papi_avail >&counters). Also, run `papi_event_chooser` to find out which events are compatible and can be included simultaneously in the colon separated list of TAU_METRICS (eg, , ibrun /opt/apps/papi/papi-3.6.0/bin/ papi_event_chooser PRESET PAPI_FP_OPS PAPI_L1_DCM). See TAU Users’ Guide, in hardware counters measurement section, for more information, [http://www.cs.uoregon.edu/research/tau/tau-usersguide.pdf](http://www.cs.uoregon.edu/research/tau/tau-usersguide.pdf)

---

**Call Graph** for matmul test code, shows the call tree and performance bottlenecks
4. Loop instrumentation

TAU automatically profiles only the outer loops. Profiling the inner loops need manual instrumentation and is not covered in this page; users need to consult the manual instrumentation section in TAU Users’ Guide. For outer loops profiling, create a selective text file and include it in TAU_OPTIONS at compile time, as shown below:

Setenv TAU_OPTIONS “-optVerbose -optKeepFiles -optTauSelectFile=select.tau”

% cat select.tau

BEGIN_INSTRUMENT_SECTION

loops file="matmul.f90" routine="MULTIPLY_MATRICES"

END_INSTRUMENT_SECTION

Loop profiling shows that ~96% of the elapsed time is spent in the loop located in line 31-36 of matmul test code

Tracing

Records time-stamped run-time events, such as enter/leave of functions for process/thread, MPI sender, receiver, length, tag, communicator. Since tracing preserves temporal and spatial relationship of events, trace files becomes very large. Selectively tracing block of the code (like selective instrumentation) significantly reduces the trace file sizes and is recommended.

For tracing, compile and instrument your code with TAU as explained in the general guide. Enable trace measurement at run time by ‘setenv TAU_TRACE 1’.

Execution of the instrumented binary with TAU trace enabled environment variable generates *.trc and *.edf files. Execute ‘% tau_treemerge.pl’ to merge the *.trc and *.edf files.
By default, TAU installation provides tau2slog2 utility to convert the trace files to Jumpshot readable format. Execute ‘% tau2slog2 tau.trc tau.edf -o tau.slog2 ’ to generate trace file in slog2 (Jumpshot readable). Then launch ‘% jumpshot tau.slog2’ to visualize and analyze the trace data with Jumpshot.

TAU supports other trace formats that are readable by SCALASCA and VAMPIR toolsets. To use other formats, TAU has to be configured with specific flags in installation. Those utilities are tau2elg for SCALASCA and tau2otf for VAMPIR which are not currently available on TAU versions installed on Ranger or Kraken.

References

http://www.cs.uoregon.edu/research/tau/tau-usersguide.pdf (Recent users’ guide)
http://www.cs.uoregon.edu/research/tau/docs/newguide/index.html (Old users’ guide)
http://www.psc.edu/general/software/packages/tau/TAU-quickref.pdf (TAU quick references)
http://www.cs.uoregon.edu/research/tau/docs/newguide/bk01ch06.html (Common profiling scenarios)
http://www.cs.uoregon.edu/research/tau/docs/newguide/bk03ch01s04.html (Papi hardware counters and timers)
http://www.cs.uoregon.edu/research/tau/docs/newguide/bk03ch02.html (Configuring TAU with tracing)
http://www.cs.uoregon.edu/research/tau/docs/newguide/bk03ch03.html (Memory profiling)
http://www.cs.uoregon.edu/research/tau/docs/newguide/bk02ch01.html#d0e891 (Installation guide)

Participants

Scientific Computing Performance Specialist, Mahin Mahmoodi at PSC has authored the performance tools page. For further assistance with using performance tools and/or commenting on this page please send email to majumdar@sdsc.edu, TeraGrid AUS Area Director.

Last updated: 6/17/10