How to use TAU for Performance Analysis

George S. Markomanolis

7 August 2019
Outline

• Introduction to TAU

• How to compile

• Explaining functionalities of TAU/ParaProf

• Presenting basic steps of PerfExplorer
TAU

• Tuning and Analysis Utilities, developed at University of Oregon
• Scalable and flexible performance analysis toolkit
• Automatic instrumentation through Program Database Toolkit (PDT) for routines, loops, I/O, memory, phases, etc.
• Installed version on Summit: v2.28.1
• Module: tau
• Web site: https://www.cs.uoregon.edu/research/tau/home.php
• Email: tau-bugs@cs.uoregon.edu
## Capability Matrix - TAU

<table>
<thead>
<tr>
<th>Capability</th>
<th>Profiling</th>
<th>Tracing</th>
<th>Notes/Limitations</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI, MPI-IO</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>OpenMP CPU</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>OpenMP GPU</td>
<td>Yes</td>
<td>Yes</td>
<td>Some restrictions apply regarding the CUPTI metrics</td>
</tr>
<tr>
<td>OpenACC</td>
<td>Yes</td>
<td>Yes</td>
<td>Some functionalities are not ready for production, no metrics available</td>
</tr>
<tr>
<td>CUDA</td>
<td>Yes</td>
<td>Yes</td>
<td>Some functionalities are not ready for production</td>
</tr>
<tr>
<td>POSIX I/O</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>POSIX threads</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Memory – app-level</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Memory – func-level</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Hotspot Detection</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Variance Detection</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Hardware Counters</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>
Compilation

- There are mainly three approaches to use an application with TAU
  - Use TAU Wrappers
    - For C: replace the compiler with tau_cc.sh
    - For C++: replace the compiler with tau_cxx.sh
    - For Fortran: replace the compiler with tau_f90.sh/tau_f77.sh
  - Dynamic instrumentation, for example:
    - jsrun -n 4 -r 4 -a 1 -c1 tau_exec -T mpi ./test
  - Rewrite the binary (support for x86_64):
    - tau_rewrite -T papi,pdf a.out -o a.inst
Compilation (cont.)

<table>
<thead>
<tr>
<th>Method</th>
<th>Requires recompiling</th>
<th>Requires PDT</th>
<th>Shows MPI events</th>
<th>Routine-level event</th>
<th>Low level events (loops, phases, etc...)</th>
<th>Throttling to reduce overhead</th>
<th>Ability to exclude file from instrumentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interposition</td>
<td></td>
<td></td>
<td>Yes</td>
<td></td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Compiler</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Source</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Interposition: `tau_exec`

Compiler:
`tau_cc.sh --tau_options=-optCompInst`
Set the `TAU_MAKEFILE`

Source:
`tau_cc.sh`
The `TAU_MAKEFILE` should include the PDT
tau_exec

Options:

-v Verbose mode
-s Show what will be done but don't actually do anything (dryrun)
-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)
-rocm Track ROCm events via rocprofiler
-ompt Track OpenMP events via OMPT interface
-ebs Enable event-based sampling
-ebs_period=<count> Sampling period (default 1000)
-ebs_source=<counter> Counter (default itimer)
-ebs_resolution=<file|function|line> Choose sampling granularity.
-um Enable Unified Memory events via CUPTI
-sass=<level> Track GPU events via CUDA with Source Code Locator activity (kernel level or source level)
-csv Outputs sass profile in CSV
-env Track GPU environment activity (power utilization, SM, memory frequency, temperature)
-T <CUPTI,DISABLE,GNU,GNU_MEM,MPI,OPENMP,PAPI,PDT,PGI,PGI_MEM,PROFILE,SERIAL> : Specify TAU tags
# TAU 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_LEAKS</td>
<td>0</td>
<td>Setting to 1 turns on leak detection</td>
</tr>
<tr>
<td>TAU.Track_HEAP</td>
<td>0</td>
<td>Setting to 1 turns on heap memory/headroom at routine entry &amp; exit</td>
</tr>
<tr>
<td>TAU_CALLPATH_DEPTH</td>
<td>2</td>
<td>Specifies depth of callpath</td>
</tr>
<tr>
<td>TAU.Track_IO_PARAMS</td>
<td>0</td>
<td>Setting to 1 with -optTrackIO</td>
</tr>
<tr>
<td>TAU_SAMPLING</td>
<td>1</td>
<td>Generates sample based profiles</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. Enabled by default to remove instrumentation in lightweight routines that are called frequently</td>
</tr>
<tr>
<td>TAU.THROTTLE_NUM_CALLS</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</td>
</tr>
<tr>
<td>TAU.COMPENSATE</td>
<td>0</td>
<td>Setting to 1 enables runtime compensation of instrumentation overhead</td>
</tr>
<tr>
<td>TAU_PROFILE_FORMAT</td>
<td>Profile</td>
<td>Setting to “merged” generates a single file, “snapshot” generates snapshot per thread</td>
</tr>
<tr>
<td>TAU_METRICS</td>
<td>TIME</td>
<td>Setting to a comma separated list (TIME:PAPI_TOT_INS:PAPI_FP_OPS)</td>
</tr>
</tbody>
</table>
# TAU Compile-Time Environment Variables

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-optVerbose</td>
<td>Turn on verbose debugging messages</td>
</tr>
<tr>
<td>-optComplInst</td>
<td>Use compiler based instrumentation</td>
</tr>
<tr>
<td>-optNoComplInst</td>
<td>Do not revert to compiler instrumentation if source instrumentation fails</td>
</tr>
<tr>
<td>-optTrackIO</td>
<td>Wrap POSIX I/O call and calculates vol/bw of I/O operations</td>
</tr>
<tr>
<td>-optKeepFiles</td>
<td>Does not remove .pdb and .inst.* files</td>
</tr>
<tr>
<td>-optPreProcess</td>
<td>Preprocess Fortran sources before instrumentation</td>
</tr>
<tr>
<td>-optTauSelectFile=&quot;&lt;file&gt;&quot;</td>
<td>Specify selective instrumentation file for tau_instrumentor</td>
</tr>
<tr>
<td>-optTauWrapFile=&quot;&lt;file&gt;&quot;</td>
<td>Specify path to link_options.tau generated by tau_gen_wrapper</td>
</tr>
<tr>
<td>-optHeaderInst</td>
<td>Enable instrumentation of headers</td>
</tr>
<tr>
<td>-optLinking=&quot;&quot;</td>
<td>Options passed to the linker</td>
</tr>
<tr>
<td>-optCompile=&quot;&quot;</td>
<td>Options passed to the compiler</td>
</tr>
<tr>
<td>-optPdtF95Opts=&quot;&quot;</td>
<td>Add options for Fortran parser in PDT</td>
</tr>
<tr>
<td>-optPdtF95Reset=&quot;&quot;</td>
<td>Reset options for Fortran parser in PDT</td>
</tr>
<tr>
<td>-optPdtCOpts=&quot;&quot;</td>
<td>Options for C parser in PDT</td>
</tr>
<tr>
<td>-optPdtCxxOpts=&quot;&quot;</td>
<td>Options for C++ parser in PDT</td>
</tr>
</tbody>
</table>

For using free format in .f files, use:

```bash
% export TAU_OPTIONS="-optPdtF95Opts=`-R free'"
```
How TAU works?

• Instrumentation:
  – Adds probes to perform measurements
  – Source code instrumentation
  – Wrapping external libraries (I/O, CUDA, OpenACC, OpenCL)
  – Rewriting the binary executable

• Measurement:
  – Profiling or Tracing
  – Direct instrumentation
  – Sampling
  – Throttling

• Analysis:
  – Visualization of profiles and traces
  – 3D visualization
  – Trace conversion tools
TAU Instrumentation/Measurement
Tau_exec

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

Options:

- v          Verbose mode
- v v        Very Verbose mode (enables TAU_VERBOSE=1)
- s          Show what will be done but don't actually do anything (dryrun)
- i o        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)
- rocm       Track ROCm events via roctoolkit
- ompt       Track OpenMP events via OMPT interface
- power      Track power events via PAPI's perf RAPL interface
- numa       Track remote DRAM, total DRAM events (needs papi with recent perf support for x86_64)
- ebs        Enable event-based sampling
- ebs_period= sampling period (default 1000)
- um         Enable Unified Memory events via CUPTI
- sass=level> Track GPU events via CUDA with Source Code Locator activity (kernel level or source level)
- csv        Outputs sass profile in CSV
MiniWeather MPI compilation

- module load pgi
- module load tau
- export TAU_MAKEFILE/sw/summit/tau/2.28.1_patch/ibm64linux/lib/Makefile.tau-pgi-papi-mpi-pdt-pgi
- Replace mpicxx with tau_cxx.sh in the Makefile
- export TAU_OPTIONS='-optLinking=-lpnetcdf -optVerbose'
- make mpi
MiniWeather MPI – Execution - Profiling

export TAU_METRICS=TIME:PAPI_TOT_INS:PAPI_TOT_CYC:PAPI_FP_OPS
#export TAU_CALLPATH=1
#export TAU_CALLPATH_DEPTH=10
export TAU_PROFILE=1
export TAU_TRACK_MESSAGE=1
export TAU_COMM_MATRIX=1

jsrun -n 64 -r 8 -a 1 -c 1 ./miniWeather_mpi
Or if compiled with mpicxx
jsrun -n 64 -r 8 -a 1 -c 1 tau_exec ./miniWeather_mpi
MiniWeather MPI - Execution

- When the execution finished, there is one folder for each TAU_METRICS declaration with the format MULTI_

- If there is no TAU_METRICS declared, then by default is used the metric TIME and the profiling files are not in a folder, in this case you need to pack them and execute paraprof:

  ```plaintext
  summit> paraprof –pack name.ppk
  summit> paraprof name.ppk
  ```

- To visualize the results execute paraprof (check also pprof for text mode)
MiniWeather MPI - Paraprof

- The default metric is TIME
- Each color is a different call
- Each horizontal line is a process or Std.Dev./mean/max/min
Exploring Paraprof

- Options -> Uncheck Stack Bars Together
- It is easier to check the load imbalance
- We will call this window as the main one
Exploring Paraprof

- Click on any color, values per process, name of routine with callpath (if activated), units in seconds, value exclusive, max, min, mean, std, values.
Exploring Paraprof

- Scroll down
Exploring Paraprof

- Click on any label on the left (node 0, mean, etc.). You can see immediately which calls take more time.
Paraprof – Thread Statistics Text Window

• Right click on any label of the main window, select “Show Thread Statistics Text Window”
Paraprof – Thread Statistics Table

- Right click on any label of the main window, select “Show Thread Statistics Table”

<table>
<thead>
<tr>
<th>Name</th>
<th>Exclusive TIME</th>
<th>Inclusive TIME</th>
<th>Calls</th>
<th>Child Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAU application</td>
<td>0</td>
<td>34.572</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MPI_Reduce()</td>
<td>5.295</td>
<td>5.295</td>
<td>790</td>
<td>0</td>
</tr>
<tr>
<td>MPI_Barrier()</td>
<td>0.001</td>
<td>0.001</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>MPI_Bcast()</td>
<td>0.158</td>
<td>0.158</td>
<td>299</td>
<td>0</td>
</tr>
<tr>
<td>MPI_Comm_dup()</td>
<td>0.089</td>
<td>0.089</td>
<td>150</td>
<td>0</td>
</tr>
<tr>
<td>MPI_Comm_free()</td>
<td>0.036</td>
<td>0.036</td>
<td>150</td>
<td>0</td>
</tr>
<tr>
<td>MPI_Comm_get_attr()</td>
<td>0</td>
<td>0</td>
<td>299</td>
<td>0</td>
</tr>
<tr>
<td>MPI_Comm_rank()</td>
<td>0.001</td>
<td>0.001</td>
<td>1,504</td>
<td>0</td>
</tr>
<tr>
<td>MPI_Comm_size()</td>
<td>0.333</td>
<td>0.333</td>
<td>300</td>
<td>0</td>
</tr>
<tr>
<td>MPI_File_close()</td>
<td>0.001</td>
<td>0.001</td>
<td>150</td>
<td>0</td>
</tr>
<tr>
<td>MPI_File_open()</td>
<td>1.034</td>
<td>1.034</td>
<td>300</td>
<td>299</td>
</tr>
<tr>
<td>MPI_File_set_view()</td>
<td>0.1</td>
<td>0.1</td>
<td>750</td>
<td>0</td>
</tr>
<tr>
<td>MPI_File_write_at()</td>
<td>3.814</td>
<td>3.814</td>
<td>451</td>
<td>601</td>
</tr>
<tr>
<td>MPI_File_write_at_all()</td>
<td>4.881</td>
<td>4.881</td>
<td>600</td>
<td>1,200</td>
</tr>
<tr>
<td>MPI_Finalize()</td>
<td>0.055</td>
<td>0.055</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MPI_get_address()</td>
<td>0.008</td>
<td>0.008</td>
<td>25,032</td>
<td>0</td>
</tr>
<tr>
<td>MPI_get_count()</td>
<td>0</td>
<td>0</td>
<td>149</td>
<td>0</td>
</tr>
<tr>
<td>MPI_info_free()</td>
<td>0.001</td>
<td>0.001</td>
<td>150</td>
<td>0</td>
</tr>
<tr>
<td>MPI_init()</td>
<td>1.448</td>
<td>1.448</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>MPI_init()</td>
<td>0.025</td>
<td>0.025</td>
<td>54,000</td>
<td>0</td>
</tr>
<tr>
<td>MPI_init()</td>
<td>0.918</td>
<td>0.918</td>
<td>54,000</td>
<td>0</td>
</tr>
<tr>
<td>MPI_status_set_elements()</td>
<td>0.001</td>
<td>0.001</td>
<td>1,995</td>
<td>0</td>
</tr>
<tr>
<td>MPI_Type_dup()</td>
<td>0.001</td>
<td>0.001</td>
<td>1,000</td>
<td>0</td>
</tr>
<tr>
<td>MPI_Type_free()</td>
<td>0.001</td>
<td>0.001</td>
<td>600</td>
<td>0</td>
</tr>
<tr>
<td>MPI_Type_free()</td>
<td>0.002</td>
<td>0.002</td>
<td>600</td>
<td>0</td>
</tr>
<tr>
<td>MPI_Type_free()</td>
<td>0.001</td>
<td>0.001</td>
<td>1,000</td>
<td>0</td>
</tr>
<tr>
<td>MPI_Type_free()</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Paraprof – User Bar Chart

- Right click on any label of the main window, select “Show User Bar Chart”
Paraprof – User Event

• Options -> Select Value Type -> Max. Value
Paraprof – User Event Statistics Window

- Right click on any label of the main window, select “Show User Event Statistics Window”
Paraprof – Context Event Window

- Right click on any label of the main window, select “Show Context Event Window” (with callpath)
• Right click on node 0 and select “Add Thread to Comparison Window”, similar for node 12. You could use any number of processes that you prefer.
Derived Metrics

Options -> Show Derived Metric Panel, select the metrics and then operator and then click Apply. Then uncheck the Show Derived Metric
Paraprof - IPC

- Click on the new metric, PAPI_TOT_INS/PAPI_TOT_CYC
Paraprof – Mean IPC

• Click on the label mean
Paraprof – IPC for thread 0

- From the main window with the PAPI_TOT_INS/PAPI_TOT_CYC metric, right click on node 0 and select Show Thread Statistics Table
Paraprof

- From the main window select Options -> Select Metric… -> Exclusive -> PAPI_FP_OPS
Paraprof – 3D Visualization

- Menu Windows -> 3D Visualization (3D demands OpenGL)
- Exclusive Time and Exclusive Floating operations
Paraprof – 3D Visualization

• Menu Windows -> 3D Visualization (3D demands OpenGL)
• Specific routine and thread
Paraprof – 3D Visualization

• Menu Windows -> 3D Visualization (3D demands OpenGL)
• Exclusive time and total instructions
Paraprof – 3D Visualization

Menu Windows -> 3D Visualization (3D demands OpenGL)
Paraprof – 3D Visualization

- Menu Windows -> 3D Visualization (3D demands OpenGL)
- Exclusive time and instructions per cycle
Paraprof – 3D Visualization

- Menu Windows -> 3D Visualization (3D demands OpenGL)
- Bar Plot
Paraprof – 3D Visualization

- Menu Windows -> 3D Visualization (3D demands OpenGL)
- Scatter Plot
Paraprof – 3D Visualization

• Menu Windows -> 3D Visualization (3D demands OpenGL)
• Topology Plot
Paraprof – 3D Communication Matrix

- Menu Windows -> 3D Visualization (3D demands OpenGL)
- Max message size vs Number of calls
Paraprof

Menu Windows -> Communication Matrix
Which loops require the most time?

• File select.tau:

BEGIN_INSTRUMENT_SECTION
loops routine="#"
END_INSTRUMENT_SECTION

• Declare TAU options:

export TAU_OPTIONS="-optTauSelectFile=select.tau -optLinking=-lpnetcdf -optVerbose"

• Do not forget to unset TAU_OPTIONS when not required
• Execute as before
Paraprof - Loops
Paraprof - Loops

Select Options -> Select Metric... -> Exclusive... -> PAPI_TOT_INS
Paraprof - Loops

Select Options -> Select Metric... -> Exclusive... ->

PAPI_TOT_INS/PAPI_TOT_CYC
Paraprof - Loops

Select Options -> Select Metric... -> Exclusive... -> PAPI_FP_OPS
Paraprof

From the main window select a node

Click on node 0
Paraprof – Function Histogram

From the main window select a node
Callpath

export TAU_METRICS=TIME:PAPI_TOT_INS:PAPI_TOT_CYC:PAPI_FP_OPS
export TAU_CALLPATH=1
export TAU_CALLPATH_DEPTH=10
export TAU_PROFILE=1
export TAU_TRACK_MESSAGE=1
export TAU_COMM_MATRIX=1

jsrun -n 64 -r 8 -a 1 -c 1 ./miniWeather_mpi
Paraprof - Callpath

From the main Window right click on any label (node 0, mean etc.) and select “Show Thread Call Graph”
Paraprof - Callpath

From the main Window right click on any label (node 0, mean etc.) and select “Show Thread Statistics Table”