TAU Performance Analysis

Nicholas Chaimov
ParaTools, Inc.

ALCF Simulation, Data and Learning Workshop
October 3rd, 2018
Overview

We will cover:

– Profiling and tracing via the TAU Performance System
– Hardware performance counters (PAPI)
– Performance analysis of C/C++, Fortran, Python
– Python+MPI analysis
PERFORMANCE CHARACTERIZATION
CONCEPTS AND TOOLS
Direct Performance Observation

– Execution actions exposed as events
  • In general, actions reflect some execution state
    ▪ presence at a code location or change in data
    ▪ occurrence in parallelism context (thread of execution)
  • Events encode actions for observation
– Observation is direct
  • Direct instrumentation of program code (probes)
  • Instrumentation invokes performance measurement
  • Event measurement = performance data + context
– Performance experiment
  • Actual events + performance measurements
Instrumentation

Code or compiler output is modified to explicitly trigger a measurement at the beginning and end of each function/region of interest.

- More detailed information
- Unequally distributed overhead (short-running functions -> larger % overhead)
- Need to process source code

Source instrumentation
Compiler instrumentation
Binary rewriting

call TAU_START('foo')
// code
call TAU_STOP('foo')
Indirect Performance Observation

- Program code instrumentation is not used
- Performance is observed indirectly
  - Execution is interrupted
    - can be triggered by different events
  - Execution state is queried (sampled)
    - different performance data measured
  - Event-based sampling (EBS)
- Performance attribution is inferred
  - Determined by execution context (state)
  - Observation resolution determined by interrupt period
  - Performance data associated with context for period
Sampling

- Periodically interrupt program and record program counter
- No modification to program needed
  - Just compile with debug symbols for address resolution
- Tradeoff between overhead and accuracy
  - Overhead evenly distributed
Measurement Approaches

Profiling

Shows how much time was spent in each routine

Tracing

Shows when events take place on a timeline
Types of Performance Profiles

– Flat profiles
  • Metric (e.g., time) spent in an event
  • Exclusive/inclusive, # of calls, child calls, …

– Callpath profiles
  • Time spent along a calling path (edges in callgraph)
  • “main=> f1 => f2 => MPI_Send”
  • Set the TAU_CALLPATH_DEPTH environment variable

– Phase profiles
  • Flat profiles under a phase (nested phases allowed)
  • Default “main” phase
  • Supports static or dynamic (e.g. per-iteration) phases
How much data do you want?

- Limited Profile
- Loop Profile
- Callpath Profile
- Flat Profile
- Phase Profile
- Trace

All levels support multiple metrics/counters

$O(\text{KB})$ → $O(\text{TB})$
Inclusive vs. Exclusive Measurements

– Exclusive measurements for region only
– Inclusive measurements includes child regions

```c
int foo()
{
    int a;
    a = a + 1;
    bar();
    a = a + 1;
    return a;
}
```
Direct Observation Events

– Interval events (begin/end events)
  • Measures exclusive & inclusive durations between events
  • Metrics monotonically increase
  • Example: Wall-clock timer

– Atomic events (trigger with data value)
  • Used to capture performance data state
  • Shows extent of variation of triggered values (min/max/mean)
  • Example: heap memory consumed at a particular point
Measure what matters

Profile

Identify Hotspots

File I/O

Yes

50x

No

Communication

10x

Yes

No

Memory

5x

Yes

No

Compute

2x

Yes

Refine the Profile

No

Focus Optimization

Buffers, data formats, in-memory filesystems

Collectives, blocking, non-blocking, topology, load balance

Bandwidth/latency, cache utilization

Vectors, branches, integer, floating point
A HIGH LEVEL OVERVIEW OF TAU’S CAPABILITIES
The TAU Performance System®

– Integrated toolkit for performance problem solving
  • Instrumentation, measurement, analysis, visualization
  • Portable profiling and tracing
  • Performance data management and data mining
– Direct and indirect measurement
– Free, open source, BSD license
– Available on all HPC platforms (and some non-HPC)
– http://tau.uoregon.edu/
Performance Tool Checklist

– Universal tool or integrated toolkit
– Unbiased, accurate measurements
  • File I/O: serial and parallel
  • Communication: inter- and intra-node
  • Memory: allocation and access
  • CPU: vectorization, cache utilization, etc.
– Minimal overhead
  • Provide multiple measurement methods
  • Focus on one performance aspect at a time

TAU Performance System®
TAU Workflow

**Instrumentation**
- **Source**
  - C, C++, Fortran, UPC, ...
  - Python, Java, ...
  - Robust parsers (PDT)
- **Library**
  - Interposition (PMPI, GASNET, …)
  - Wrapper generation
- **Linker**
  - Static, Dynamic
  - Preloading (LD_PRELOAD)
- **Executable**
  - Dynamic (Dyinst)
  - Binary (Dyinst, MAQAO, PEBIL)

**Measurement**
- **Events**
  - Static, Dynamic
  - Routine, Block, Loop
  - Threading, Communication
  - Heterogeneous
- **Profiling**
  - Flat, Callpath, Phase, Snapshot
  - Probe, Sampling, Compiler, Hybrid
- **Tracing**
  - TAU, Scalasca, ScoreP
  - Open Trace Format (OTF)
- **Metadata**
  - System
  - User defined

**Analysis**
- **Profiles**
  - ParaProf analyzer & visualizer
  - 3D profile data visualization
  - Communication matrix
  - Callstack analysis
  - Graph generation
  - PerfDMF
  - PerfExplorer profile data miner
- **Traces**
  - OTF, SLOG-2
  - Vampir
  - Jumpshot
- **Online**
  - Event unification
  - Statistics calculation
Instrument: Add Probes

- Sampling
  - Event-based sampling
- Source code instrumentation
  - PDT parsers, pre-processors
- Wrap external libraries
  - I/O, MPI, Memory, CUDA, OpenCL, pthread
- Rewrite the binary executable
  - Dyninst, MAQAO
Measure: Gather Data

- Direct measurement via probes
- Indirect measurement via sampling
- Throttling and runtime control
- Interface with external packages (PAPI)
Analyze: Synthesize Knowledge

- Data visualization
- Data mining
- Statistical analysis
- Import/export performance data
How Much Time per Code Region?

% paraprof (Click on label, e.g. “Mean” or “node 0”)
How Many Instructions per Code Region?

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
<th>Code Region</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAPI_FP_INS</td>
<td>8.8479E9</td>
<td>VPASSM [[fft.f] {501,7}-{836,9}]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SGBTF2 [[bandludcmp.f] {238,7}-{440,9}]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NOISE [[init.f] {451,7}-{746,26}]</td>
</tr>
<tr>
<td></td>
<td>4.1824E9</td>
<td>FILTERZ_FOURIER [[filtering.f] {39,7}-{115,36}]</td>
</tr>
<tr>
<td></td>
<td>4.0701E9</td>
<td>STBSV [[ludcmp.f] {4698,7}-{5043,9}]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DPDZ_FOURIER [[deriv.f] {5,7}-{75,33}]</td>
</tr>
<tr>
<td></td>
<td>7.8914E8</td>
<td>DPDZ_PHYS [[deriv.f] {81,7}-{152,30}]</td>
</tr>
<tr>
<td></td>
<td>6.3193E8</td>
<td>SGBTRS [[bandludcmp.f] {883,7}-{1069,9}]</td>
</tr>
<tr>
<td></td>
<td>6.0732E8</td>
<td>FFT99B [[fft.f] {1282,7}-{1360,9}]</td>
</tr>
<tr>
<td></td>
<td>5.4591E8</td>
<td>FILTERXY_FOURIER [[filtering.f] {119,7}-{172,37}]</td>
</tr>
<tr>
<td></td>
<td>5.1141E8</td>
<td>FFT99A [[fft.f] {1208,7}-{1281,9}]</td>
</tr>
<tr>
<td></td>
<td>4.2268E8</td>
<td>CONVEC::RHSFM [[interm.f] {1101,7}-{1988,26}]</td>
</tr>
<tr>
<td></td>
<td>4.0761E8</td>
<td>MATRIX_ELMSETUP [penalty.f] {409,7}-{670,36}]</td>
</tr>
<tr>
<td></td>
<td>3.2453E8</td>
<td>CONVEC [[interm.f] {7,7}-{2063,27}]</td>
</tr>
<tr>
<td></td>
<td>3.0524E8</td>
<td>DIVCOLC [[deriv.f] {207,7}-{273,29}]</td>
</tr>
<tr>
<td></td>
<td>1.6188E8</td>
<td>HORIZ [[horiz.f] {2,7}-{136,27}]</td>
</tr>
<tr>
<td></td>
<td>1.2698E8</td>
<td>CONVEC::ROTVISC [[interm.f] {1994,8}-{2038,29}]</td>
</tr>
<tr>
<td></td>
<td>7.4375E7</td>
<td>DPDZCOL [[deriv.f] {159,7}-{203,28}] [[THROTTLED]]</td>
</tr>
<tr>
<td></td>
<td>5.8376E7</td>
<td>UPDATE [[calcuvw.f] {603,7}-{726,27}]</td>
</tr>
<tr>
<td></td>
<td>4.6697E7</td>
<td>MAXCALC [[various.f] {549,7}-{842,28}]</td>
</tr>
<tr>
<td></td>
<td>4.0317E7</td>
<td>RELAX_RESET [[postp.f] {765,7}-{868,32}]</td>
</tr>
<tr>
<td></td>
<td>3.6434E7</td>
<td>PERTURBCALC [[postp.f] {693,7}-{759,32}]</td>
</tr>
<tr>
<td></td>
<td>2.6551E7</td>
<td>CALCUVWT [[calcuvw.f] {2,7}-{596,29}]</td>
</tr>
<tr>
<td></td>
<td>2.1258E7</td>
<td>PENALTY_MAIN [penalty.f] {237,7}-{286,33}]</td>
</tr>
</tbody>
</table>
How Many L1 or L2 Cache Misses?

% paraprof (Options → Select Metric... → Exclusive... → PAPI_L1_DCM)
How Much Memory Does the Code Use?

![Memory Utilization Table]

- **High-water mark**

  - `% paraprof` (Right-click label [e.g “node 0”] → Show Context Event Window)
How Much Memory Does the Code Use?

<table>
<thead>
<tr>
<th>Name</th>
<th>Total</th>
<th>NumSamples</th>
<th>MaxValue</th>
<th>MinValue</th>
<th>MeanValue</th>
<th>Std. Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TAU application</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>free size (bytes)</td>
<td>14,239,992.16</td>
<td>27,169.781</td>
<td>49,152</td>
<td>1</td>
<td>524.001</td>
<td>2,013.103</td>
</tr>
<tr>
<td>malloc size (bytes)</td>
<td>13,132,932</td>
<td>23,292</td>
<td>262,144</td>
<td>1</td>
<td>563.839</td>
<td>4,492.057</td>
</tr>
<tr>
<td><strong>MPI_Finalize()</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>free size (bytes)</td>
<td>1,298,918.679</td>
<td>1,495.125</td>
<td>461,766.25</td>
<td>4</td>
<td>868.769</td>
<td>16,928.073</td>
</tr>
<tr>
<td>malloc size (bytes)</td>
<td>48,150</td>
<td>20</td>
<td>36,032</td>
<td>11</td>
<td>2,407.5</td>
<td>7,911.992</td>
</tr>
<tr>
<td><strong>OurMain</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>free size (bytes)</td>
<td>3,465</td>
<td>9</td>
<td>769</td>
<td>32</td>
<td>385</td>
<td>260.2</td>
</tr>
<tr>
<td>malloc size (bytes)</td>
<td>4,314</td>
<td>12</td>
<td>769</td>
<td>32</td>
<td>359.5</td>
<td>240.981</td>
</tr>
<tr>
<td><strong>&lt;module&gt;</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>free size (bytes)</td>
<td>293,088</td>
<td>449</td>
<td>32,564</td>
<td>32</td>
<td>652.757</td>
<td>1,526.875</td>
</tr>
<tr>
<td>malloc size (bytes)</td>
<td>311,966</td>
<td>493</td>
<td>32,564</td>
<td>32</td>
<td>632.791</td>
<td>1,460.941</td>
</tr>
<tr>
<td><strong>staticCFF</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong><strong>init</strong></strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>&lt;module&gt;</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory Utilization (heap, in KB)</td>
<td>849,270.344</td>
<td>192,825.168</td>
<td>0.078</td>
<td>147,832.141</td>
<td>62,621.576</td>
<td></td>
</tr>
<tr>
<td>Message size for all-gather</td>
<td>4,096</td>
<td>1</td>
<td>4,096</td>
<td>4,096</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Message size for all-reduce</td>
<td>23,340</td>
<td>843</td>
<td>320</td>
<td>4</td>
<td>27.687</td>
<td>64.653</td>
</tr>
<tr>
<td>Message size for all-to-all</td>
<td>104</td>
<td>26</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Message size for broadcast</td>
<td>24,923</td>
<td>206</td>
<td>8,788</td>
<td>4</td>
<td>120.985</td>
<td>860.992</td>
</tr>
<tr>
<td>Message size for reduce</td>
<td>8,912</td>
<td>8</td>
<td>8,788</td>
<td>4</td>
<td>1,114</td>
<td>2,900.511</td>
</tr>
<tr>
<td>free size (bytes)</td>
<td>27,417,881</td>
<td>391,51</td>
<td>413,600</td>
<td>719</td>
<td>24,025</td>
<td>667</td>
</tr>
<tr>
<td>malloc size (bytes)</td>
<td>27,468,709</td>
<td>355,914</td>
<td>435,669</td>
<td>625</td>
<td>24,025</td>
<td>667</td>
</tr>
</tbody>
</table>

Total allocated/deallocated

%% paraprof (Right-click label [e.g “node 0”] → Show Context Event Window)
### Where is Memory Allocated / DEALLOCATED?

#### Allocation / DEALLOCATION EVENTS

<table>
<thead>
<tr>
<th>Name</th>
<th>Total</th>
<th>Num Samples</th>
<th>MaxValue</th>
<th>MinValue</th>
<th>MeanValue</th>
<th>Std. Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TAU application</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>free size (bytes)</td>
<td>1,298,918.679</td>
<td>1,495.125</td>
<td>461,766.25</td>
<td>4</td>
<td>868,769</td>
<td>16,928.073</td>
</tr>
<tr>
<td>malloc size (bytes)</td>
<td>48,150</td>
<td>20</td>
<td>36,032</td>
<td>11</td>
<td>2,407.5</td>
<td>7,911.992</td>
</tr>
<tr>
<td><strong>OurMain</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>free size (bytes)</td>
<td>3,465</td>
<td>9</td>
<td>769</td>
<td>32</td>
<td>385</td>
<td>260.2</td>
</tr>
<tr>
<td>malloc size (bytes)</td>
<td>4,314</td>
<td>12</td>
<td>769</td>
<td>32</td>
<td>359.5</td>
<td>240.981</td>
</tr>
<tr>
<td><strong>&lt;module&gt;</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>free size (bytes)</td>
<td>293,088</td>
<td>449</td>
<td>32,564</td>
<td>32</td>
<td>652,757</td>
<td>1,526.875</td>
</tr>
<tr>
<td>malloc size (bytes)</td>
<td>311,966</td>
<td>493</td>
<td>32,564</td>
<td>32</td>
<td>632,791</td>
<td>1,460.941</td>
</tr>
</tbody>
</table>

#### Memory Utilization (heap, in KB)

- 849,270.344 192,825.168 0.078 147,832.141 62,621.576
- 4,096 1 4,096 4,096 0
- 23,340 843 320 4 27,687 64,653
- 104 26 4 4 4 0
- 24,923 206 8,788 4 120,985 860.992
- 8,912 8 8,788 4 1,114 2,900.511
- 27,417,881,391.51 413,600,719 24,025,667 1 66,290,701 199,538,234
- 27,468,709,355.914 435,669,625 24,025,667 0 63,049,402 195,561,193

% paraprof (Right-click label [e.g “node 0”] → Show Context Event Window)
How Much Time is spent in Collectives?

- **Message sizes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Total</th>
<th>Num...</th>
<th>MaxValue</th>
<th>MinValue</th>
<th>MeanValue</th>
<th>Std. Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_Wait()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPI_Waitall()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Message size for all-gather</td>
<td>305,753,268</td>
<td>72</td>
<td>172,215,296</td>
<td>4</td>
<td>4,246,573,167</td>
<td>22,551,605,859</td>
</tr>
<tr>
<td>Message size for all-reduce</td>
<td>163,308</td>
<td>632</td>
<td>21,908</td>
<td>4</td>
<td>258,399</td>
<td>897.725</td>
</tr>
<tr>
<td>Message size for all-to-all</td>
<td>112</td>
<td>14</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>Message size for broadcast</td>
<td>692,208,045.5</td>
<td>3,346</td>
<td>18,117,620</td>
<td>0</td>
<td>206,876,284</td>
<td>1,284,673,036</td>
</tr>
<tr>
<td>Message size for gather</td>
<td>6,901,452,378</td>
<td>15,312</td>
<td>1,387,306,625</td>
<td>4</td>
<td>450,707,094</td>
<td>483,216,499</td>
</tr>
<tr>
<td>Message size for reduce</td>
<td>66,812</td>
<td>1,520</td>
<td>56</td>
<td>4</td>
<td>43.955</td>
<td>21.598</td>
</tr>
<tr>
<td>Message size for scatter</td>
<td>63,147,906</td>
<td>146</td>
<td>62,567,906</td>
<td>4</td>
<td>432.52</td>
<td>5,160.063</td>
</tr>
</tbody>
</table>

- **Time spent in collectives**
3D Profile Visualization

% paraprof (Windows → 3D Visualization)
3D Communication Visualization

% qsub –env TAU_COMM_MATRIX=1 ...
% paraprof (Windows → 3D Communication Matrix)
3D Topology Visualization

% paraprof (Windows → 3D Visualization → Topology Plot)
How Does Each Routine Scale?

% perfexplorer (Charts → Runtime Breakdown)
How Does Each Routine Scale?

% perfexplorer (Charts → Stacked Bar Chart)
When do Events Occur?

export TAU_TRACE=1
Intuitive Performance Engineering

USING TAU
Preinstalled Tools on Theta

– On Theta, versions of TAU and related tools are installed at /soft/perftools/tau
– Modules are available:
  – % module avail tau
    ----- /soft/environment/modules/modulefiles -------
    tau/2.25.2 tau/2.26  tau/2.26.1 tau/2.26.2 tau/2.26.3 tau/2.27
    tau/2.27.1 tau/2.27.2
  – % module load tau/2.27.2
Using TAU Directly

– An unusual thing about TAU installations
  • For most UNIX software, when running
    ▪ `./configure --foo; make install`
    ▪ `./configure --bar; make install`
  • the second install will overwrite the first install.
  • In TAU, the two configurations are installed **side by side**.
– To reduce overhead, many features are enabled at compile time rather than runtime.
  • Always use `--bfd=download --unwind=download` for sampling address resolution
  • Common `./configure` options: `-cc`, `-c++`, `-fortran` select compiler, `-mpi`, `-pthread`, `-openmp`, `-cuda`
– For each configuration of TAU, a Makefile is present in `$TAU/<arch>/lib/Makefile.tau-*`
  • `<arch>` is craycn1 on Theta
– TAU_MAKEFILE environment variable determines configuration used by compiler wrappers.
  • e.g., export TAU_MAKEFILE=`<path to TAU>/<arch>/lib/Makefile.tau-intel-papi-mpi-pdt`
– `-T` option determines configuration used by `tau_exec`
  • E.g., `tau_exec -T intel,papi,pdt`
  • `tau_exec` assumes `mpi`; specify `serial` if not
Typical Workflow

- Sample to identify hotspots (tau_exec -ebs)
- Selectively instrument hotspots (tau_f90.sh and friends, -optSelectFile)
- Gather hardware performance counter data (papi_avail, TAU_METRICS)
- Visualize performance data, derived metrics (paraprof)
- Visualize scaling data (perfexplorer)
Sampling with TAU

- Use `tau_exec -ebs`
  - Build without TAU as *dynamic executable*, *with* `–g` if you want line-level resolution.
    - Does not work with static executables, which are default with Cray compiler wrappers!
  - Run application through `tau_exec`, prepending launcher

**Makefile without TAU**

```bash
CXX = cc -dynamic
F90 = ftn -dynamic
CXXFLAGS = -g
LIBS =
OBJ = f1.o f2.o f3.o ... fn.o
app: $(OBJ)
   $(CXX) $(LD) $(OBJ) -o $@
   $(LIBS)
.cpp.o:
   $(CXX) $(CXXFLAGS) -c <$
```

tau_exec comes *after* launcher. Otherwise, we would sample aprun itself!

```bash
aprun -n 16 tau_exec -T mpi,pthread -ebs ./foo
```
Insert TAU API Calls Automatically

– Use TAU’s compiler wrappers
  • Replace CXX with tau_cxx.sh, etc.
  • Automatically instruments source code, links with TAU libraries.
– Use tau_cc.sh for C, tau_f90.sh for Fortran, etc.
– Run normally through launcher

Makefile without TAU

```makefile
CXX = cc
F90 = ftn
CXXFLAGS =
OBJJS = f1.o f2.o f3.o ... fn.o

app: $(OBJJS)
   $(CXX) $(LDFLAGS) $(OBJJS) -o $@
   $(LIBS)
CPP.O:
   $(CXX) $(CXXFLAGS) -c $<
```

Makefile with TAU

```makefile
CXX = tau_cxx.sh
F90 = tau_f90.sh
CXXFLAGS =
OBJJS = f1.o f2.o f3.o ... fn.o
LIBS = -lm

app: $(OBJJS)
   $(CXX) $(LDFLAGS) $(OBJJS) -o $@
   $(LIBS)
CPP.O:
   $(CXX) $(CXXFLAGS) -c $<
```
## Compile-Time Options

Optional parameters for the TAU_OPTIONS environment variable:

```sh
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`)
- `-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) …
<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 (for use with tau_exec –memory ./a.out)</td>
</tr>
<tr>
<td>TAU_TRACK_HEAP or TAU_TRACK_HEADROOM</td>
<td>0</td>
<td>Setting to 1 turns on tracking heap memory/headroom at routine entry &amp; exit using context events (e.g., Heap at Entry: main=&gt;foo=&gt;bar)</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_TRACK_IO_PARAMS</td>
<td>0</td>
<td>Setting to 1 with –optTrackIO or tau_exec –io captures arguments of I/O calls</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. Enabled by default to remove 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_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 xml format</td>
</tr>
<tr>
<td>TAU_METRICS</td>
<td>TIME</td>
<td>Setting to a comma separated list generates other metrics. (e.g., TIME:P_VIRTUAL_TIME:PAPI_FP_INS:PAPI_NATIVE_&lt;event&gt;:&lt;subevent&gt;)</td>
</tr>
</tbody>
</table>
Hardware Counters

Hardware performance counters available on most modern microprocessors can provide insight into:

1. Whole program timing
2. Cache behaviors
3. Branch behaviors
4. Memory and resource access patterns
5. Pipeline stalls
6. Floating point efficiency
7. Instructions per cycle

Hardware counter information can be obtained with:

1. Subroutine or basic block resolution
2. Process or thread attribution
What’s PAPI?

Open Source software from U. Tennessee, Knoxville
http://icl.cs.utk.edu/papi

Middleware to provide a consistent programming interface for the performance counter hardware found in most major micro-processors. Countable events are defined in two ways:
- Platform-neutral preset events
- Platform-dependent native events

Presets can be derived from multiple native events
All events are referenced by name and collected in EventSets
PAPI Utilities: papi_avail

$ utils/papi_avail
Available events and hardware information.

PAPI Version             : 4.0.0.0
Vendor string and code   : GenuineIntel (1)
Model string and code    : Intel Core i7 (21)
CPU Revision             : 5.000000
CPUID Info               : Family: 6  Model: 26  Stepping: 5
CPU Megahertz            : 2926.000000
CPU Clock Megahertz      : 2926
Hdw Threads per core     : 1
Cores per Socket         : 4
NUMA Nodes               : 2
CPU's per Node           : 4
Total CPU's              : 8
Number Hardware Counters : 7
Max Multiplex Counters   : 32

The following correspond to fields in the PAPI_event_info_t structure.

[MORE...]
PAPI Utilities: papi_avail

[CONTINUED…]

The following correspond to fields in the PAPI_event_info_t structure.

<table>
<thead>
<tr>
<th>Name</th>
<th>Code</th>
<th>Avail</th>
<th>Deriv</th>
<th>Description (Note)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAPI_L1_DCM</td>
<td>0x80000000</td>
<td>No</td>
<td>No</td>
<td>Level 1 data cache misses</td>
</tr>
<tr>
<td>PAPI_L1_ICM</td>
<td>0x80000001</td>
<td>Yes</td>
<td>No</td>
<td>Level 1 instruction cache misses</td>
</tr>
<tr>
<td>PAPI_L2_DCM</td>
<td>0x80000002</td>
<td>Yes</td>
<td>Yes</td>
<td>Level 2 data cache misses</td>
</tr>
<tr>
<td>PAPI_VEC_SP</td>
<td>0x80000069</td>
<td>Yes</td>
<td>No</td>
<td>Single precision vector/SIMD instruc</td>
</tr>
<tr>
<td>PAPI_VEC_DP</td>
<td>0x8000006a</td>
<td>Yes</td>
<td>No</td>
<td>Double precision vector/SIMD instruc</td>
</tr>
</tbody>
</table>

[...]

Of 107 possible events, 34 are available, of which 9 are derived.

avail.c                                  PASSED
PAPI Utilities: papi_avail

$ utils/papi_avail -e PAPI_FP_OPS
[...]

The following correspond to fields in the PAPI_event_info_t structure.

Event name: PAPI_FP_OPS
Event Code: 0x80000066
Number of Native Events: 2
Short Description: |FP operations|
Long Description: |Floating point operations|
Developer's Notes: ||
Derived Type: |DERIVED_ADD|
Postfix Processing String: ||
Native Code[0]: 0x4000801b |FP_COMP_OPS_EXE:SSE_SINGLE_PRECISION|
Number of Register Values: 2
Register[ 0]: 0x0000000f |Event Selector|
Register[ 1]: 0x00004010 |Event Code|
Native Event Description: |Floating point computational micro-ops, masks:SSE* FP single precision Uops|

Native Code[1]: 0x4000081b |FP_COMP_OPS_EXE:SSE_DOUBLE_PRECISION|
Number of Register Values: 2
Register[ 0]: 0x0000000f |Event Selector|
Register[ 1]: 0x00008010 |Event Code|
Native Event Description: |Floating point computational micro-ops, masks:SSE* FP double precision Uops|

-------------------------------------------------------------------------
### PAPI Utilities: `papi_native_avail`

UNIX> `utils/papi_native_avail`

Available native events and hardware information.

```
[...]
```

<table>
<thead>
<tr>
<th>Event Code</th>
<th>Symbol</th>
<th>Long Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x40000010</td>
<td>BR_INST_EXEC</td>
<td>Branch instructions executed</td>
</tr>
<tr>
<td>40000410</td>
<td>:ANY</td>
<td>Branch instructions executed</td>
</tr>
<tr>
<td>40000810</td>
<td>:COND</td>
<td>Conditional branch instructions executed</td>
</tr>
<tr>
<td>40001010</td>
<td>:DIRECT</td>
<td>Unconditional branches executed</td>
</tr>
<tr>
<td>40002010</td>
<td>:DIRECT_NEAR_CALL</td>
<td>Unconditional call branches executed</td>
</tr>
<tr>
<td>40004010</td>
<td>:INDIRECT_NEAR_CALL</td>
<td>Indirect call branches executed</td>
</tr>
<tr>
<td>40008010</td>
<td>:INDIRECT_NON_CALL</td>
<td>Indirect non call branches executed</td>
</tr>
<tr>
<td>40010010</td>
<td>:NEAR_CALLS</td>
<td>Call branches executed</td>
</tr>
<tr>
<td>40020010</td>
<td>:NON_CALLS</td>
<td>All non call branches executed</td>
</tr>
<tr>
<td>40040010</td>
<td>:RETURN_NEAR</td>
<td>Indirect return branches executed</td>
</tr>
<tr>
<td>40080010</td>
<td>:TAKEN</td>
<td>Taken branches executed</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Event Code</th>
<th>Symbol</th>
<th>Long Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x40000011</td>
<td>BR_INST_RETIRED</td>
<td>Retired branch instructions</td>
</tr>
<tr>
<td>40000411</td>
<td>:ALL_BRANCHES</td>
<td>Retired branch instructions (Precise Event)</td>
</tr>
<tr>
<td>40000811</td>
<td>:CONDITIONAL</td>
<td>Retired conditional branch instructions (Precise Event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[Event]</td>
</tr>
<tr>
<td>40001011</td>
<td>:NEAR_CALL</td>
<td>Retired near call instructions (Precise Event)</td>
</tr>
</tbody>
</table>
PAPI Utilities: papi_native_avail

unix> utils/papi_native_avail -e DATA_CACHE_REFILLS
Available native events and hardware information.

 The following correspond to fields in the PAPI_event_info_t structure.

<table>
<thead>
<tr>
<th>Event name</th>
<th></th>
<th>Event Code:</th>
<th>Number of Register Values:</th>
<th>Description:</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA_CACHE_REFILLS</td>
<td></td>
<td>0x4000000b</td>
<td>2</td>
<td>[Data Cache Refills from L2 or System]</td>
</tr>
</tbody>
</table>

| Register[ 0]:       | 0x0000000f       | Event Selector  |
| Register[ 1]:       | 0x00000042       | Event Code      |

Unit Masks:

<table>
<thead>
<tr>
<th>Mask Info:</th>
<th></th>
<th>Register[ 0]:</th>
<th>Event Selector</th>
</tr>
</thead>
<tbody>
<tr>
<td>:SYSTEM</td>
<td>Refill from System</td>
<td>0x0000000f</td>
<td></td>
</tr>
<tr>
<td>:L2_SHARED</td>
<td>Shared-state line from L2</td>
<td>0x000000142</td>
<td></td>
</tr>
<tr>
<td>:L2_EXCLUSIVE</td>
<td>Exclusive-state line from L2</td>
<td>0x000000242</td>
<td></td>
</tr>
</tbody>
</table>

| Register[ 0]:       | 0x0000000f       | Event Selector |
| Register[ 1]:       | 0x00000042       | Event Code     |
$ utils/papi_event_chooser PRESET PAPI_FP_OPS
Event Chooser: Available events which can be added with given events.

Name        Code    Deriv Description (Note)
PAPI_L1_DCM  0x80000000  No   Level 1 data cache misses
PAPI_L1_ICM  0x80000001  No   Level 1 instruction cache misses
PAPI_L2_ICM  0x80000003  No   Level 2 instruction cache misses
PAPI_L1_DCA  0x80000040  No   Level 1 data cache accesses
PAPI_L2_DCR  0x80000044  No   Level 2 data cache reads
PAPI_L2_DCW  0x80000047  No   Level 2 data cache writes
PAPI_L1_ICA  0x8000004c  No   Level 1 instruction cache accesses
PAPI_L2_ICA  0x8000004d  No   Level 2 instruction cache accesses
PAPI_L2_TCA  0x80000059  No   Level 2 total cache accesses
PAPI_L2_TCW  0x8000005f  No   Level 2 total cache writes
PAPI_FML_INS 0x80000061  No   Floating point multiply instructions
PAPI_FDV_INS 0x80000063  No   Floating point divide instructions

Total events reported: 34
event_chooser.c                          PASSED
PAPI Utilities: papi_event_chooser

```
$ utils/papi_event_chooser PRESET PAPI_FP_OPS PAPI_L1_DCM
Event Chooser: Available events which can be added with given events.
--------------------------------------------------------------------------
[...]
--------------------------------------------------------------------------

<table>
<thead>
<tr>
<th>Name</th>
<th>Code</th>
<th>Deriv</th>
<th>Description (Note)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAPI_TOT_INS</td>
<td>0x80000032</td>
<td>No</td>
<td>Instructions completed</td>
</tr>
<tr>
<td>PAPI_TOT_CYC</td>
<td>0x8000003b</td>
<td>No</td>
<td>Total cycles</td>
</tr>
</tbody>
</table>

--------------------------------------------------------------------------
Total events reported: 2
```

event_chooser.c                          PASSED
PAPI Utilities: papi_event_chooser

$ utils/papi_event_chooser NATIVE RESOURCE_STALLS:LD_ST X87_OPS_RETIRED INSTRUCTIONS_RETIRED

[...

UNHALTED_CORE_CYCLES 0x40000000
|count core clock cycles whenever the clock signal on the specific core is running (not halted). Alias to event CPU_CLK_UNHALTED:CORE_P|
|Register Value[0]: 0x20003 Event Selector|
|Register Value[1]: 0x3c Event Code|

UNHALTED_REFERENCE_CYCLES 0x40000002
|Unhalted reference cycles. Alias to event CPU_CLK_UNHALTED:REF|
|Register Value[0]: 0x40000 Event Selector|
|Register Value[1]: 0x13c Event Code|

CPU_CLK_UNHALTED 0x40000028
|Core cycles when core is not halted|
|Register Value[0]: 0x60000 Event Selector|
|Register Value[1]: 0x3c Event Code|

0x40001028 :CORE_P |Core cycles when core is not halted|
0x40008028 :NO_OTHER |Bus cycles when core is active and the other is halted|

Total events reported: 3
**TAU Workflow**

- Sampling to determine what to look at more closely
  - `tau_exec -ebs`
- Storage
  - `paraprof -pack file.ppk`
  - `taudb_loadtrial`
- Visualization
  - `paraprof`
- Selective Instrumentation
  - `tau_cc.sh, tau_cxx.sh, tau_f90.sh`
  - `export TAU_OPTIONS="-optSelectFile=path"`
  - (see [https://www.cs.uoregon.edu/research/tau/docs/newguide/bk01ch01s03.html](https://www.cs.uoregon.edu/research/tau/docs/newguide/bk01ch01s03.html) for syntax)
- Hardware performance counters
  - `papi_avail` to determine what’s available
  - `papi_event_chooser` to determine what’s compatible
  - `export TAU_METRICS=TIME:PAPI_L2_TCM:PAPI_L2_TCA`
- Derived metrics
  - `paraprof`
Python Performance Evaluation

HANDS-ON
Getting Started with TAU

– Series of exercises available at:

https://fs.paratools.com/TAU(SDL)_examples.tar.gz
Example 1: C Matmult (MPI + Pthreads) Source Instrumentation

- First, we will install a compatible configuration of PDT and TAU:

  ```
  wget http://tau.uoregon.edu/pdt.tar.gz
  tar xzf pdt.tar.gz
  cd pdtoolkit-3.25
  ./configure
  make install # installs into current directory
  
  cd ..
  wget http://tau.uoregon.edu/tau.tgz
  tar xzf tau.tgz
  cd tau-2.27.2p1
  ./configure -bfd=download -unwind=download -arch=craycnl -pdt=<path to PDT>/pdtoolkit-3.25 -pdt_c++=/usr/bin/g++ -mpi -pthread
  make install
  
  export PATH=<path to TAU>/tau-2.27.2p1/craycnl/bin:$PATH
  ```
Example 1: C Matmult (MPI + Pthreads) Source Instrumentation

$ cd workshop-python/01_matmult.c
$ make CC=tau_cc.sh

Run normally to generate profiles:
$ aprun -n 4 -N 4 ./matmult
$ ls profile.*  # Shows four files
$ paraprof --pack mm_c_flat.ppk

View the profiles:
pprof -a | less    #Command line
paraprof          #GUI (Java, X11)
Example 2: Fortran Matmult (MPI)

$ cd workshop-python/02_matmult.f90
$ make F90=tau_f90.sh

Run normally to generate profiles:
$ aprun -n 4 -N 4 ./matmult
$ ls profile.* # Shows four files
$ paraprof --pack mm_f90_flat.ppk

View the profiles:
pprof -a | less #Command line
paraprof #GUI (Java, X11)
Basic TAU Workflow

Choose your TAU_MAKEFILE:

– $ export TAU_MAKEFILE=$TAU/Makefile.tau-mpi-python-pdt

Use tau_f90.sh, tau_cxx.sh, etc. as compiler:

– $ ftn foo.f90
  
  changes to
  
  $ tau_f90.sh foo.f90

Edit Makefile or set compilers on command line:

$ make CC=tau_cc.sh

Execute application

Analyze performance data:

– pprof (for text based profile display)
– paraprof (for GUI)
Example 3: TAU with Pure Python

- Build a Python configuration of TAU:

  module load miniconda-3.6/conda-4.5.4
cd <TAU directory>
./configure -bfd=download -unwind=download -arch=craycnl -python
make install
FIXEDGRID

A simple chemical transport model in Python

\[
\frac{\partial c_x^t}{\partial t} = \sum_{k=1}^{d} \left[ \frac{\partial}{\partial x_k} \left( d_k(x, t) \frac{\partial c_x^t}{\partial x_k} - a_k(x, t) c_x^t \right) \right] + F
\]

**Advection**: Upwind-biased 2\textsuperscript{nd} order finite differences

**Diffusion**: 3\textsuperscript{rd} order finite differences

**Chemistry**: Rosenbrock time-stepping integrator
TAU with Pure Python

$ cd workshop-python/03_fixedgrid.py

Run with tau_python to generate profiles:
$ export TAU_CALLPATH=1 # Generate callpath profiles
$ aprun -n 1 -N 1 tau_python -T serial,intel,python fixedgrid.py
$ ls profile.* # shows profile.0.0.0
$ paraprof --pack fixedgrid_py_flat.ppk

View the profiles:
$ pprof -a | less # Command line
$ paraprof # GUI (Java, X11)
ParaProf Profile Visualizer

Left-click on a node name to see data for that node.
Right-click on a node name to see more options.
Exclusive Time in ParaProf
Inclusive Time in ParaProf
Statistics Table in ParaProf

<table>
<thead>
<tr>
<th>Name</th>
<th>Exclusive TIME</th>
<th>Inclusive TIME</th>
<th>Calls</th>
<th>Child C...</th>
</tr>
</thead>
<tbody>
<tr>
<td>.TAU application</td>
<td>2.051</td>
<td>54.92</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>exec</td>
<td>0.001</td>
<td>52.801</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>&lt;module&gt;</td>
<td>0.001</td>
<td>52.8</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>fixedgrid</td>
<td>0.004</td>
<td>35.636</td>
<td>1</td>
<td>79</td>
</tr>
<tr>
<td>discrete_rows</td>
<td>0.143</td>
<td>22.428</td>
<td>6</td>
<td>1,008</td>
</tr>
<tr>
<td>discrete</td>
<td>0.225</td>
<td>22.283</td>
<td>1,800</td>
<td>10,800</td>
</tr>
<tr>
<td>space_advec_diff</td>
<td>3.381</td>
<td>20.457</td>
<td>3,600</td>
<td>183,600</td>
</tr>
<tr>
<td>advect_diff</td>
<td>17.063</td>
<td>17.063</td>
<td>180,000</td>
<td>0</td>
</tr>
<tr>
<td>len</td>
<td>0.014</td>
<td>0.014</td>
<td>3,600</td>
<td>0</td>
</tr>
<tr>
<td>&lt;listcomp&gt;</td>
<td>0.028</td>
<td>0.113</td>
<td>3,600</td>
<td>3,600</td>
</tr>
<tr>
<td>copy</td>
<td>0.016</td>
<td>0.016</td>
<td>1,800</td>
<td>0</td>
</tr>
<tr>
<td>empty_like</td>
<td>0.003</td>
<td>0.003</td>
<td>108</td>
<td>0</td>
</tr>
<tr>
<td>discretize_cols</td>
<td>0.074</td>
<td>11.198</td>
<td>18</td>
<td>1,854</td>
</tr>
<tr>
<td>discretize</td>
<td>0.117</td>
<td>11.089</td>
<td>900</td>
<td>5,400</td>
</tr>
<tr>
<td>space_advec_diff</td>
<td>1.691</td>
<td>10.168</td>
<td>1,800</td>
<td>91,800</td>
</tr>
<tr>
<td>advect_diff</td>
<td>8.47</td>
<td>8.47</td>
<td>90,000</td>
<td>0</td>
</tr>
<tr>
<td>len</td>
<td>0.007</td>
<td>0.007</td>
<td>1,800</td>
<td>0</td>
</tr>
<tr>
<td>&lt;listcomp&gt;</td>
<td>0.0236</td>
<td>0.741</td>
<td>900</td>
<td>45,000</td>
</tr>
<tr>
<td>copy</td>
<td>0.014</td>
<td>0.054</td>
<td>1,800</td>
<td>1,800</td>
</tr>
<tr>
<td>empty_like</td>
<td>0.008</td>
<td>0.008</td>
<td>900</td>
<td>0</td>
</tr>
<tr>
<td>copy</td>
<td>0.009</td>
<td>0.031</td>
<td>900</td>
<td>900</td>
</tr>
<tr>
<td>empty</td>
<td>0.001</td>
<td>0.001</td>
<td>54</td>
<td>0</td>
</tr>
<tr>
<td>full</td>
<td>0.001</td>
<td>0.003</td>
<td>4</td>
<td>12</td>
</tr>
<tr>
<td>print</td>
<td>0.003</td>
<td>0.003</td>
<td>21</td>
<td>0</td>
</tr>
<tr>
<td>.find_and_load</td>
<td>0.002</td>
<td>19.163</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>find_module</td>
<td>0.005</td>
<td>0.045</td>
<td>1</td>
<td>21</td>
</tr>
<tr>
<td>compile</td>
<td>0.019</td>
<td>0.019</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>read</td>
<td>0.002</td>
<td>0.004</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>new_module</td>
<td>0.001</td>
<td>0.001</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Callgraph in ParaProf
Callgraph in ParaProf
Traces with Pure Python

To generate traces:
$ unset TAU_CALLPATH    #recommended
$ export TAU_TRACE=1
$ aprun -n 1 -N 1 tau_python -T serial,intel,python fixedgrid.py

Trace files must be post-processed:
$ tau_treemerge.pl
$ tau2slog2 tau.trc tau.edf -o \n     mm_py.slog2
$ jumpshot mm_py.slog2
Public Service Annoucement

Don’t forget to clean your environment!
(Some folks write scripts)

Show all TAU environment variables:
$ env | grep TAU

Unset the ones you don’t need anymore:
$ unset TAU_TRACE
$ unset TAU_CALLPATH
etc.
MPI in FIXEDGRID

MPI Rank 0
MPI Rank 1
MPI Rank 2
MPI Rank 3

NX NY
Example 4: TAU with Python + MPI

Build a Python+MPI configuration of TAU:

```
module load miniconda-3.6/conda-4.5.4

cd <TAU directory>

./configure -bfd=download -unwind=download -arch=craycnl -python -mpi

make install
```
**TAU with mpi4py**

$ cd 04_fixedgrid-mpi.py
$ aprun -n 4 -N 4 tau_python -T mpi,intel,python fixedgrid.py

View the profiles:

pprof -a | less #Command line
paraprof #GUI (Java, X11)
FIXEDGRID Profile

Left-click on a node name to see data for that node
Right-click on a node name to see more options
FIXEDGRID Profile

![Graph showing time metrics and functions]

- Metric: TIME
- Value: Exclusive
- Units: seconds

- Time breakdown:
  - **1.101** seconds for `advec_diff` ([fixedgrid.py][10])
  - **1.021** seconds for `MPI_Init_thread`
  - **0.289** seconds for `space_advec_diff` ([fixedgrid.py][39])
  - **0.232** seconds for `MPI_Sendrecv`
  - **0.075** seconds for `max`
  - **0.049** seconds for `discretize` ([fixedgrid.py][86])
  - **0.041** seconds for `map`
  - **0.031** seconds for `.TAU application`
  - **0.03** seconds for `<lambda>` ([fixedgrid.py][99])
  - **0.025** seconds for `discretize_cols` ([fixedgrid.py][123])
  - **0.016** seconds for `MPI_Finalize`
  - **0.016** seconds for `discretize_rows` ([fixedgrid.py][102])
  - **0.012** seconds for `add_newdoc` ([function_base.py][2945])
  - **0.009** seconds for `array`
FIXEDGRID Communication Matrix

$ export TAU_COMM_MATRIX=1
$ aprun -n 4 -N 4 tau_python -T mpi,intel,python fixedgrid.py

In Paraprof: Windows | Communication Matrix
FIXEDGRID Trace Shows Communication

$ jumpshot fixedgrid_mpi.slog2
PerfExplorer

$ cd 04_fixedgrid-mpi.py/analysis
$ taudb_configure  --create-default
$ taudb_loadtrial fixedgrid_np1.ppk
$ taudb_loadtrial fixedgrid_np2.ppk
$ taudb_loadtrial fixedgrid_np3.ppk
...
$ perfexplorer
Relative Speedup Chart

In PerfExplorer: Charts | Relative Speedup
Runtime Breakdown Chart

In PerfExplorer: Charts | Runtime Breakdown
Example 5: TAU + Python + mpi4py + C + OpenMP

- Build a Python+MPI+OpenMP configuration of TAU:

```bash
module load miniconda-3.6/conda-4.5.4

cd <TAU directory>

./configure -bfd=download -unwind=download -arch=craycnl -python
-mpi -ompt=download

make install
```
Example 5: TAU + Python + mpi4py + C + OpenMP

$ cd 05_fixedgrid-chem.c_py
$ export TAU_MAKEFILE=<path to Makefile from install step>
$ make CC=tau_cc.sh

Run with tau_exec and wrapper.py to generate profiles:
$ make clean
$ make CC=tau_cc.sh
$ aprun -n 4 -N 4 tau_python -T python,mpi,openmp,intel,ompt,tr6 -ompt fixedgrid.py
Download TAU from U. Oregon

http://tau.uoregon.edu

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

Free download, open source, BSD license
Questions or Problems?

support@paratools.com
Support Acknowledgments

US Department of Energy (DOE)
- Office of Science contracts
- SciDAC, LBL contracts
- LLNL-LANL-SNL ASC/NNSA contract
- Battelle, PNNL contract
- ANL, ORNL contract

Department of Defense (DoD)
- HPCMO

National Science Foundation (NSF)
- Glassbox, SI-2

Partners:
University of Oregon
ParaTools, Inc.
University of Tennessee, Knoxville
T.U. Dresden, GWT
Juelich Supercomputing Center