Intuitive Performance Engineering at the Exascale with TAU and TAU Commander

John C. Linford
ParaTools, Inc.

Argonne Extreme Scale Computing Training Program
11 August 2014, Pheasant Run “Resort”
Overview

• Motivation

• TAU Overview

• TAU Commander

• Case Studies
Intuitive Performance Engineering

MOTIVATION
Why don’t we just use this?

About 8 flops (0.1 flops/Watt)
Because this is better than an abacus

10 petaflops (2.17 gigaflops/Watt)
A lot better...

1.25 quadrillion abacuses
The Metrics We Care About

Performance

Efficiency

Productivity
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/
How do we Improve Productivity?

XKCD, Randall Munroe
How do we Improve Productivity?

TAU Commander
An intuitive interface to the TAU Performance System
Intuitive Performance Engineering

THE TAU PERFORMANCE SYSTEM
The TAU Performance System®

- Tuning and Analysis Utilities (20+ year project)
- Comprehensive performance profiling and tracing
  - Integrated, scalable, flexible, portable
  - Targets all parallel programming/execution paradigms
- Integrated performance toolkit
  - Instrumentation, measurement, analysis, visualization
  - Widely-ported performance profiling/tracing system
  - Performance data management and data mining
  - Open source (BSD-style license)
- Integrates with application frameworks
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?

All levels support multiple metrics/counters
Direct via Probes

- Exact measurement
- Fine-grain control
- Calls inserted into code

Indirect via Sampling

- No code modification
- Minimal effort
- Relies on debug symbols (-g option)

```c
// code
call TAU_START('potential')
call TAU_STOP('potential')
```
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.

Makefile without TAU

CXX = mpicxx
F90 = mpif90
CXXFLAGS =
LIBS = -lm
OBJJS = f1.o f2.o f3.o ... fn.o

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

Makefile with TAU

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

app: $(OBJJS)
    $(CXX) $(LDFLAGS) $(OBJJS) -o $@
    $(LIBS)
.cpp.o:
    $(CXX) $(CXXFLAGS) -c $<
TAU Supports All HPC Platforms

- C/C++
- Fortran
- pthreads
- Intel
- GNU
- MinGW

- CUDA
- UPC
- OpenACC
- Intel MIC
- LLVM
- PGI
- OpenACC
- BlueGene
- Fujitsu
- Android

- Python
- MPI
- Java
- OpenMP
- Cray
- Sun
- AIX
- ARM
- MPC
- OS X
TAU Architecture and 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

- **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
• Each configuration of TAU corresponds to a unique stub makefile
  (**TAU_MAKEFILE**) in the TAU installation directory

```bash
% ls /soft/perftools/tau/tau_latest/bgq/lib/Makefile.*
Makefile.tau-bgqtimers-mpi-pdt-openmp-opari
Makefile.tau-bgqtimers-mpi-pthread-pdt
Makefile.tau-bgqtimers-papi-mpi-pdt
Makefile.tau-bgqtimers-papi-mpi-pdt-openmp-opari
Makefile.tau-bgqtimers-papi-mpi-pdt-scorep
Makefile.tau-papi-mpi-pdt-openmp-opari-scorep
Makefile.tau-papi-mpi-pdt-scorep
```
1. Choose an appropriate TAU_MAKEFILE:

   % soft add +tau-latest
   % export TAU_MAKEFILE=/soft/perftools/tau/tau_latest/bgq/lib/Makefile.tau-bgqttimers-mpi-pdt
   % export TAU_OPTIONS='--optVerbose ...'
   # (see tau_compiler.sh -help for more options)

2. Use tau_f90.sh, tau_cxx.sh, etc. as Fortran, C++, etc. compiler:

   % mpiXlf90_r foo.f90
   changes to
   % tau_f90.sh foo.f90

3. Execute application:

   % qsub --A <queue> --q R.bc --n 256 --t 10 ./a.out

   Note: If TAU_MAKEFILE has “papi” in its name, set TAU_METRICS:
   % qsub --env TAU_METRICS=BGQ_TIMERS:PAPI_L2_DCM...

4. Analyze performance data:

   pprof (for text based profile display)
   paraprof (for GUI)
% ssh mira.alcf.anl.gov
% tar xzvf /soft/perftools/tau/workshop.tgz
% cd workshop
% less README

For an MPI+F90 application, you may want to start with:
% soft add +tau-latest
% export TAU_MAKEFILE=
  $TAU/Makefile.tau-bgqtimers-papi-mpi-pdt
% make CC=tau_cc.sh CXX=tau_cxx.sh F90=tau_f90.sh
% qsub -q R.bc -n 2 --mode c16 -t 10 -A ... ./a.out
% paraprof
How Much Time per Code Region?

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

% paraprof (Options → Select Metric... → Exclusive... → PAPI_FP_INS)
How Many L1 or L2 Cache Misses?

```bash
% paraprof (Options ➔ Select Metric... ➔ Exclusive... ➔ PAPI_L1_DCM)
```
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>.TAU application</td>
<td>14,236,992.16</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</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>MPI_Finalize()</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</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>OurMain()</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</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>&lt;module&gt;</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</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>staticCFD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>init</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;module&gt;</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>4</td>
<td>0</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>27,417,881,391.51</td>
<td>413,600,719</td>
<td>24,025,667</td>
<td>1</td>
<td>66,290.701</td>
<td>199,538.234</td>
</tr>
<tr>
<td>malloc size (bytes)</td>
<td>27,468,709,355.914</td>
<td>435,669,625</td>
<td>24,025,667</td>
<td>0</td>
<td>63,049.402</td>
<td>195,561.193</td>
</tr>
</tbody>
</table>

% 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>.TAU application</td>
<td>14,236,992.16</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</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>MPI_Finalize()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OurMain()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>free size</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</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>&lt;module&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>free size</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</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>staticCFD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>init</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;module&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Memory Utilization (heap, in KB)
- Total: 849,270.344
- Mean: 192,825.168
- Std. Dev.: 0.078
- 90th Percentile: 147,832.141
- 99th Percentile: 62,621.576

Message size for all-gather
- Total: 4,096
- Mean: 4,096
- Std. Dev.: 0

Message size for all-reduce
- Total: 23,340
- Mean: 843
- Std. Dev.: 320

Message size for all-to-all
- Total: 104
- Mean: 26
- Std. Dev.: 4

Message size for broadcast
- Total: 24,923
- Mean: 206
- Std. Dev.: 8,788

Message size for reduce
- Total: 8,912
- Mean: 8
- Std. Dev.: 8,788

Free size (bytes)
- Total: 27,417,881,391.51
- Mean: 413,600.719
- Std. Dev.: 24,025,667

Malloc size (bytes)
- Total: 27,468,709,355.914
- Mean: 435,669.625
- Std. Dev.: 24,025,667

% paraprof (Right-click label [e.g. “node 0”] → Show Context Event Window)
### Allocation / Deallocation Events

<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>free size (bytes)</td>
<td>14,236,992.16</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>OurMain()</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>&lt;module&gt;</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>staticCFD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>init</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;module&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

- Total: 849,270.344
- NumSamples: 192,825.168
- MaxValue: 0.078
- MinValue: 147,832.141
- MeanValue: 62,621.576

---

% paraprof (Right-click label [e.g. “node 0”] → Show Context Event Window)
What are the I/O Characteristics?

Write bandwidth per file

Bytes written to each file

% paraprof (Right-click label [e.g “node 0”] → Show Context Event Window)
What are the I/O Characteristics?

<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>Incl</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Initialize</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LoadBodyEuler</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LoadMesh</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPI-IO Bytes Written</td>
<td>4,328,712</td>
<td>144</td>
<td>893,152</td>
<td>0</td>
<td>30,060.5</td>
<td>128,042.696</td>
</tr>
<tr>
<td>MPI-IO Write Bandwidth (MB/s)</td>
<td>144</td>
<td>196.86</td>
<td>0</td>
<td>3.421</td>
<td>16.87</td>
<td></td>
</tr>
<tr>
<td>MPI_Allgatherv()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPI_Bcast()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPI_Comm_create()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPI_File_close()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPI_File_open()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPI_File_write_all()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPI_File_write_at()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPI_Finalize()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPI_Gather()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPI_Gatherv()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Peak MPI-IO Write Bandwidth

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

<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>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>

---

**Message sizes**

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

% paraprof (Windows ➔ 3D 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)
Which Events Correlate with Runtime?

Correlation Results: All Trials: TIME

% perfexplorer (Charts ➔ Correlate Events with Total Runtime)
When do Events Occur?
To generate a trace and visualize it in Jumpshot:

% qsub --env TAU_TRACE=1 ...
% tau_treemerge.pl
% tau2slog2 tau.trc tau.edf --o app.slog2
% jumpshot app.slog2
What Caused My Application to Crash?

% qsub –env TAU_TRACK_SIGNALS=1 ...
% paraprof
What Caused My Application to Crash?

Right-click to see source code

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BACKTRACE 1</td>
<td>[SAMINT:::timestamp(double, cloude)]</td>
</tr>
<tr>
<td>BACKTRACE 2</td>
<td>[samarcstep(double, double)]</td>
</tr>
<tr>
<td>BACKTRACE 3</td>
<td>[_wrap_samarcstep]</td>
</tr>
<tr>
<td>BACKTRACE 4</td>
<td>[call_function]</td>
</tr>
<tr>
<td>BACKTRACE 5</td>
<td>[fast_function]</td>
</tr>
<tr>
<td>BACKTRACE 6</td>
<td>[PyEval_EvalCodeEx]</td>
</tr>
<tr>
<td>BACKTRACE 7</td>
<td>[PyEval_EvalCode]</td>
</tr>
<tr>
<td>BACKTRACE 8</td>
<td>[PyImport_ExecCodeModuleEx]</td>
</tr>
<tr>
<td>BACKTRACE 9</td>
<td>[load_source_module]</td>
</tr>
<tr>
<td>BACKTRACE 10</td>
<td>[load_next]</td>
</tr>
<tr>
<td>BACKTRACE 11</td>
<td>[load_module_level]</td>
</tr>
<tr>
<td>BACKTRACE 12</td>
<td>[supports]</td>
</tr>
<tr>
<td>BACKTRACE 13</td>
<td>[call_builtin]</td>
</tr>
<tr>
<td>BACKTRACE 14</td>
<td>[PyOriect_Call]</td>
</tr>
<tr>
<td>BACKTRACE 15</td>
<td>[PyEval_CallObjectWithKeywords]</td>
</tr>
<tr>
<td>BACKTRACE 16</td>
<td>[PyEval_FrameEx]</td>
</tr>
<tr>
<td>BACKTRACE 17</td>
<td>[fast_function]</td>
</tr>
<tr>
<td>BACKTRACE 18</td>
<td>[PyEval_EvalCodeEx]</td>
</tr>
<tr>
<td>BACKTRACE 19</td>
<td>[PyEval_EvalCode]</td>
</tr>
<tr>
<td>BACKTRACE 20</td>
<td>[run_mod]</td>
</tr>
<tr>
<td>BACKTRACE 21</td>
<td>[exec_statement]</td>
</tr>
<tr>
<td>BACKTRACE 22</td>
<td>[PyEval_EvalCodeEx]</td>
</tr>
<tr>
<td>BACKTRACE 23</td>
<td>[fast_function]</td>
</tr>
<tr>
<td>BACKTRACE 24</td>
<td>[run_mod]</td>
</tr>
<tr>
<td>BACKTRACE 25</td>
<td>[PyRun_SimpleFileExFlags]</td>
</tr>
<tr>
<td>BACKTRACE 26</td>
<td>[Py_Main]</td>
</tr>
<tr>
<td>BACKTRACE 27</td>
<td>[Py_MPI_Main_with_communicator]</td>
</tr>
<tr>
<td>BACKTRACE 28</td>
<td>[main]</td>
</tr>
<tr>
<td>BACKTRACE 29</td>
<td>[start]</td>
</tr>
</tbody>
</table>

ATPSEC'14, Copyright © ParaTools, Inc.
What Caused My Application to Crash?

Error shown in ParaProf Source Browser

```c
void SAMINT::timestep(const double time, const double dt) {
    cout << "SAMINT::timestep()" << endl;
    timestep_(time, dt);
    int x = 4 / (4-4); // Error: Division by zero
    cout << "x = " << x << endl;
}
```

```c
void SAMINT::writePlotData(const double time, const int step) {
    cout << "SAMINT::writePlotData()" << endl;
}"
```
Memory debugging

MPI/Pthread/Python/C++/Fortran

Runtime Overhead

- Tracking
- Debugging
- Full Tracking
- Full Debugging
- Valgrind

TAU with various options

Note: Requires working mprotect() so BGQ not supported
Intuitive Performance Engineering

TAU COMMANDER
TAU: Powerful and Complex

How do we navigate?
TAU Commander

• Two-year $1M project, DOE SBIR Phase II
• Simplify TAU usage
  – Develop a common framework for TAU user interface
  – Develop an intuitive, high-productivity user interface
• Phase I results:
  – Study of 124 workflows involving eleven test cases on six computing systems
  – TAU Commander reduces number of steps in TAU workflow by approximately 50%
  – Reduces the number of commands a user must know from approximately eight to exactly one
The TAU Commander Approach

• Say where you’re going, not how to get there
• **TAU Projects** give context to the user’s actions
  – Defines desired metrics and measurement approach
  – Defines operating environment
  – Establishes a baseline for error checking

43° 74’ 35” N
69° 39’ 15” W

VS.
TAU Usage (SBIR Phase I)

Setup

Instrument

Troubleshoot

Measure

Need to Parse Code?

Download PDT

Configure PDT

Compile/Install PDT

Yes

No

Configure TAU

Compile TAU

Compilation Error?

Yes

No

Setup Environment

Instrument Source Code?

Yes

No

Use TAU to Compile

Incorrect TAU Config

Compiler Env Vars Correct?

Yes

No

Compilation Error?

Yes

No

Setup Environment

Required tau_exec Tags Available?

Yes

No

tau_exec Required?

Yes

No

Launch with tau_exec

Launch Application

Runime Env Vars Correct?

Yes

No

Desired Metrics Measured?

Yes

No

Analyze

Yes

No

Yes
TAU Commander Usage (SBIR Phase I)

1. Add TAU to PATH
2. Create or Select Project
3. Use TAU to compile
4. Launch Application
5. Desired Metrics Measured?
   - Yes
   - No
6. Analyze

Setup

Instrument

Measure
TAU Commander User Interface

Usage:
tau [options] <command> [<args>...]
tau -h | --help
tau --version

Options:
--verbose Same as --log=DEBUG
--quiet Same as --log=CRITICAL
--log=<level> Output level, one of CRITICAL, ERROR, WARNING, INFO, or DEBUG.

Commands:
build Instrument programs during compilation and/or linking.
help Get help with a command.
make Build your application with 'make' and the TAU compilers.
pack Package profile files into a PPK file.
project Create and manage TAU projects.
run Gather measurements from an application.
show Display application profile or trace data.

Shortcuts:
<compiler> A compiler command, e.g. gcc, mpif90, upcc, nvcc, etc.
An alias for 'tau build <compiler>'
<executable> A program executable, e.g. ./a.out
An alias for 'tau execute <executable>'
<profile> View profile data (*.ppk, *.xml, profile.*, etc.) in ParaProf
An alias for 'tau show <profile>'
<trace> View trace data (*.otf *.slog2, etc.) in Jumpshot
An alias for 'tau show <trace>'

See 'tau <command> --help' for more information on a specific command.

jlinfo@mavericks-pro ~ $
## TAU Commander Example Usage

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
<th>Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Add TAU to path</td>
<td>export PATH=$HOME/workspace/taucmd:$PATH</td>
</tr>
<tr>
<td>2</td>
<td>Create project</td>
<td>cd matmult&lt;br&gt;tau project create matmult --openmp --mpi --callpath=100 --memory</td>
</tr>
<tr>
<td>3</td>
<td>Instrument with TAU</td>
<td>tau mpif90 *.f90 -o matmult</td>
</tr>
<tr>
<td>4</td>
<td>Execute instrumented application</td>
<td>tau qsub -A &lt;queue&gt; -q R.bc -n 256 -t 10 .matmult</td>
</tr>
<tr>
<td>5</td>
<td>Visualize data</td>
<td>tau show</td>
</tr>
</tbody>
</table>

Note: the ‘tau’ command is in beta. Its usage may change.
Intuitive Performance Engineering

CASE STUDIES
IRMHD on Intrepid and Mira

• INCITE magnetohydrodynamics simulation to understand solar winds and coronal heating
  – First direct numerical simulations of Alfvén wave (AW) turbulence in extended solar atmosphere accounting for inhomogeneities
  – Team
    • University of New Hampshire (Jean Perez and Benjamin Chandran)
    • ALCF (Tim Williams)
    • University of Oregon (Sameer Shende)

• IRMHD (Inhomogeneous Reduced Magnetohydrodynamics)
  – Fortran 90 and MPI
  – Excellent weak and strong scaling properties
  – Tested and benchmarked on Intrepid and Mira

• HPC Source article and ALCF news
IRMHD Communication Analysis

- Source-based (direct) instrumentation
- MPI instrumentation and volume measurement
- IRMHD exhibited significant synchronous communication bottlenecks
- On 2,408 cores of BG/P:
  - `MPI_Send` and `MPI_Bcast` take significant time
  - Opportunities for communication/computation overlap
  - Identified possible targets for computation improvements
• On 2,408 cores, overall execution time reduced from 528.18 core hours to 70.8 core hours (>7x improvement)

• Non-blocking communication substrate

• More efficient implementation of underlying FFT
• At 32K cores, load imbalance visible in topology
  – Negative impact on MPI_Alltoall performance
IRMHD Optimization on MIRA (BG/Q)

- Remove unnecessary MPI_BARRIER calls
• Oversubscribe nodes: 32k ranks vs. 16k per node
• Overall time improvement: 71.23% of original
NWChem

- A leading chemistry modeling code
- Relies on global arrays (GA)
  - Unified view of physically distributed arrays
  - One-sided random access
  - Use Aggregate Remote Memory Copy Interface (ARMCI)
• What is the performance of representative workloads on different platforms?
• Understand data server / compute node interplay as a function of scaling
• Understand core allocation tradeoffs
  – All to computation vs. some to communication

NWChem Instrumentation

• Source-based (direct) instrumentation
• ARMCI interposition library with TAU hooks (TAU PARMCI)
• Wrapped external libraries, e.g. BLAS
• Test Systems:
  – Fusion: Pacific Northwest National Lab
  – Intrepid: Argonne National Lab
• Mira and hopper will scale greater but will likely show similar effects
NWChem with Pinning on Fusion

- Test on 8 cores (no separate data server thread)
- With no pinning, ARMCI communication overhead increases dramatically and no scaling is observed
- Pinning communication buffers shows dramatic effects
- Communication overhead increases, but not dramatically
NWChem with Pinning on Intrepid

- Tests with interrupt or communication helper thread (CHT)
  - One core dedicated to CHT
- ARMCI calls are barely noticeable
- DAXPY calculation shows up more
- CHT performs better in both SMP and DUAL modes
Download TAU from U. Oregon

http://tau.uoregon.edu

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

Free download, open source, BSD license
Acknowledgements

- Department of Energy
  - Office of Science
  - Argonne National Laboratory
  - Oak Ridge National Laboratory
  - NNSA/ASC Trilabs (SNL, LLNL, LANL)
- HPCMP DoD PETTT Program
- National Science Foundation
  - Glassbox, SI-2
- University of Tennessee
- University of New Hampshire
  - Jean Perez, Benjamin Chandran
- University of Oregon
  - Allen D. Malony, Sameer Shende
  - Kevin Huck, Wyatt Spear
- TU Dresden
  - Holger Brunst, Andreas Knupfer
  - Wolfgang Nagel
- Research Centre Jülich
  - Bernd Mohr
  - Felix Wolf
Online References

• PAPI:
  – PAPI documentation is available from the PAPI website:
    http://icl.cs.utk.edu/papi/

• TAU:
  – TAU Users Guide and papers available from the TAU website:
    http://tau.uoregon.edu/

• VAMPIR:
  – VAMPIR website:
    http://www.vampir.eu/

• Scalasca:
  – Scalasca documentation page:
    http://www.scalasca.org/

• Eclipse PTP:
  – Documentation available from the Eclipse PTP website:
    http://www.eclipse.org/ptp/
Compiling Fortran Codes with TAU

- If your Fortran code uses free format in .f files (fixed is default for .f):
  \% 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'

- To use an instrumentation specification file:
  \% export TAU_OPTIONS=
     '-optTauSelectFile=select.tau -optVerbose -optPreProcess'

Example select.tau file

BEGIN_INSTRUMENT_SECTION
loops file="*" routine="#"
memory file="foo.f90" routine="#"
io file="abc.f90" routine="FOO"
END_INSTRUMENT_SECTION
% export TAU_MAKEFILE=$TAU/Makefile.tau-bgqtimers-papi-mpi-pdt
% export TAU_OPTIONS='--optTauSelectFile=select.tau --optVerbose'
% cat select.tau
    BEGIN_INSTRUMENT_SECTION
        loops routine="#"
    END_INSTRUMENT_SECTION

% export PATH=$TAU_ROOT/bin:$PATH
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
%
% qsub --env TAU_METRICS=TIME:PAPI_FP_INS:PAPI_L1_DCM -n 4 -t 15 ./a.out
% paraprof --pack app.ppk
    Move the app.ppk file to your desktop.
% paraprof app.ppk
    Choose Options -> Show Derived Metrics Panel -> “PAPI_FP_INS”, click “/”, “TIME”, click “Apply” and choose the derived metric.
% export TAU_MAKEFILE=$TAU/Makefile.tau-bgqtimers-papi-mpi-pdt
% export PATH=$TAU_ROOT/bin:$PATH
% export TAU_OPTIONS=‘-optTrackIO -optVerbose’
% make CC=tau_cc.sh CXX=tau_cxx.sh F90=tau_f90.sh
% mpirun -n 4 ./a.out
% paraprof -pack ioprofile.ppk
% export TAU_TRACK_IO_PARAMS 1
% mpirun -n 4 ./a.out (to track parameters used in POSIX I/O calls as context events)
Installing and Configuring TAU

• Installing PDT:
  – `wget http://tau.uoregon.edu/pdt_lite.tgz`
  – `./configure --prefix=<dir>; make ; make install`

• Installing TAU:
  – `wget http://tau.uoregon.edu/tau.tgz`
  – `./configure --bfd=download --pdt=<dir> --papi=<dir> ...`
  – `make install`

• Using TAU:
  – `export TAU_MAKEFILE=<taudir>/<arch>/lib/Makefile.tau-<TAGS>`
  – `make CC=tau_cc.sh CXX=tau_cxx.sh F90=tau_f90.sh`
% tau_compiler.sh

-optsVerbose
  Turn on verbose debugging messages
-optsCompInst
  Use compiler based instrumentation
-optsNoCompInst
  Do not revert to compiler instrumentation if source instrumentation fails.
-optsTrackIO
  Wrap POSIX I/O call and calculates vol/bw of I/O operations
-optsMemDbg
  Runtime bounds checking (see TAU_MEMDBG_* env vars)
-optsKeepFiles
  Does not remove intermediate .pdb and .inst.* files
-optsPreProcess
  Preprocess sources (OpenMP, Fortran) before instrumentation
-optsTauSelectFile="<file>"
  Specify selective instrumentation file for tau_instrumentor
-optsTauWrapFile="<file>"
  Specify path to link_options.tau generated by tau_gen_wrapper
-optsHeaderInst
  Enable Instrumentation of headers
-optsTrackUPCR
  Track UPC runtime layer routines (used with tau_upc.sh)
-optsPdtF95Opts=""
  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_MEMORYLeaks</td>
<td>0</td>
<td>Setting to 1 turns on leak detection (for use with –optMemDbg or tau_exec)</td>
</tr>
<tr>
<td>TAU_MEMDBG_PROTECT_ABOVE</td>
<td>0</td>
<td>Setting to 1 turns on bounds checking for dynamically allocated arrays. (Use with –optMemDbg or tau_exec –memory_debug).</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>