Intuitive Performance Engineering at the Exascale with TAU and TAU Commander

John C. Linford
ParaTools, Inc.

Argonne Extreme Scale Computing Training Program
10 August 2016, Pheasant Run “Resort”
A SILLY INTRODUCTION
This is faster than an abacus

- 10 petaflops (2.08 gigaflops/Watt)
- 1.25 quadrillion abacuses
- ~172,000 abacuses per living person
This is even faster

- 180 petaflops (5.62 gigaflops/Watt)
- 22.5 quadrillion abacuses
- ~3.1M abacuses per living person
Unfreeze your science fish!

• Performance engineering tools are essential to success.
• Increase your capability to answer the hard questions.
• Reduce time-to-solution (i.e. publish first).
• Lower costs.
• Credibility, provenance, etc.
Intuitive Performance Engineering

PERFORMANCE ENGINEERING
Define your project

• What are the **target** hardware and software?
  – Shared memory via OpenMP, pthreads, etc?
  – Distributed memory via MPI, SHMEM, etc?
  – Intel compilers? IBM compilers?

• What are the **application** features?
  – What language(s)?
  – MPI? OpenMP? CUDA?
  – Compute-intense? Data intense? I/O intense?

• What do I want to **measure**? And how?
  – Wall clock time?
  – Hardware performance counters?
  – Probes? Library wrapping? Sampling?
  – Profiles? Traces? Both?
T-A-M Model for Performance Engineering

• Target
  – Installed software
  – Available compilers
  – Host architecture/OS

• Application
  – MPI, OpenMP, CUDA, OpenACC, etc.

• Measurement
  – Profile, trace, or both
  – Sample, source inst...

Experiment =
(Target, Application, Measurement)
Which platform is best for my application?

- Many targets
  - Possibly same hardware, different software
- One measurement
- One application

![Diagram showing Measurement, Application, and Targets 0 to N]
What are the performance characteristics of my application?

- One target
- Many measurements
- One application
How well does my target perform various tasks?

- **One** target
- **One** measurement
- **Many** applications (e.g. benchmarks)
### TAU Commander

#### Alpha release!

<table>
<thead>
<tr>
<th>Name</th>
<th>Profile</th>
<th>Trace</th>
<th>Sample</th>
<th>Source Inst.</th>
<th>Compiler Inst.</th>
<th>MPI</th>
<th>OpenMP</th>
<th>Callpath Depth</th>
<th>Mem. Usage</th>
<th>Mem. Alloc</th>
<th>In Projects</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex-profile</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>automatic</td>
<td>fallback</td>
<td>No</td>
<td>compiler_default</td>
<td>0</td>
<td>No</td>
<td>No</td>
<td>ex-mm</td>
</tr>
<tr>
<td>ex-trace</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>automatic</td>
<td>fallback</td>
<td>No</td>
<td>compiler_default</td>
<td>0</td>
<td>No</td>
<td>No</td>
<td>ex-mm</td>
</tr>
<tr>
<td>ex-sample</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>never</td>
<td>never</td>
<td>No</td>
<td>compiler_default</td>
<td>0</td>
<td>No</td>
<td>No</td>
<td>ex-mm</td>
</tr>
</tbody>
</table>

#### Projects (/home/jlinford/.tau)

<table>
<thead>
<tr>
<th>Name</th>
<th>Targets</th>
<th>Applications</th>
<th>Measurements</th>
<th>Home</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex-mm</td>
<td>localhost</td>
<td>ex-mm-serial</td>
<td>ex-profile</td>
<td>/home/jlinford/.tau</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ex-mm-openmp</td>
<td>ex-trace</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ex-sample</td>
<td></td>
</tr>
</tbody>
</table>

No trials. Use 'tau <command>' or 'tau trial create <command>' to create a new trial

```
jlinford@east03 ~/workspace/taucmdr/examples/mm $ taudash
```

---

**ParaTools**

ATPSEC'16, Copyright © ParaTools, Inc.
Intuitive Performance Engineering

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/](http://tau.uoregon.edu/)
• 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
**Performance Data Measurement**

**Direct via Probes**

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

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

**Indirect via Sampling**

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

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

app: $(OBJJS)
  $(CXX) $(CXXFLAGS) $(OBJJS) -o @
  $(OBJJS)

.cpp.o:
  $(CXX) $(CXXFLAGS) -c $<
```

**Makefile with TAU**

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

app: $(OBJJS)
  $(CXX) $(CXXFLAGS) $( OBJJS) -o @
  $(OBJJS)

.cpp.o:
  $(CXX) $(CXXFLAGS) -c $<
```
# 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 (Dyninst)
- Binary (Dininst, 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
Using TAU: A Brief Introduction

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

% soft add +tau-latest
% ls $TAU/Makefile.*
Makefile.tau-bgqtimers-papi-gnu-ompt-pdt-openmp
Makefile.tau-bgqtimers-papi-mpi-pdt
Makefile.tau-bgqtimers-papi-mpi-pdt-openmp-opari
Makefile.tau-bgqtimers-papi-mpi-pdt-scorep
Makefile.tau-bgqtimers-papi-mpi-pthread-pdt
Makefile.tau-bgqtimers-papi-pdt
Makefile.tau-gnu-bgqtimers-papi-mpi-pdt
Makefile.tau-gnu-bgqtimers-papi-mpi-pdt-openmp-opari
Makefile.tau-gnu-bgqtimers-papi-mpi-pthread-pdt
% export TAU_MAKEFILE=$TAU/Makefile.tau-bgqtimers-papi-mpi-pdt
Using TAU: A Brief Introduction

1. Choose an appropriate TAU_MAKEFILE:

   % soft add +tau-latest
   % export TAU_MAKEFILE=/soft/perftools/tau/tau_latest/
     bgq/lib/Makefile.tau-bgqtimers-papi-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)
Hands-on (18:30 – 21:15)

```
% ssh mira.alcf.anl.gov
% tar xvfz /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?

% 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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<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>&lt;module&gt;</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>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) | 849,270.344 | 192,825.168 | 0.078 | 147,832.141 | 62,621.576 |
| Message size for all-gather | 4,096 | 1 | 4,096 | 4,096 | 4,096 | 0 |
| Message size for all-reduce | 23,340 | 843 | 320 | 4 | 27.687 | 64.653 |
| Message size for all-to-all | 104 | 26 | 4 | 4 | 4 | 0 |
| Message size for broadcast | 24,923 | 206 | 8,788 | 4 | 120.985 | 860.992 |
| Message size for reduce | 8,912 | 8 | 8,788 | 4 | 1,114 | 2,900.511 |
| free size (bytes) | 27,417,881,391.51 | 413,600.719 | 24,025,667 | 1 | 66,290.701 | 199,538.234 |
| malloc size (bytes) | 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 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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<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>OurMain</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>&lt;module&gt;</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>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>

#### Total allocated/deallocated:

\[
\text{Total allocated/deallocated} \Rightarrow \text{Show Context Event Window}
\]
### Allocation / Deallocation Events

**TAU: ParaProf: Mean Context Events – sphere_np32_nsteps5_mem.ppk**

<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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<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>MPI_Finalize()</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>OurMain()</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>&lt;module&gt;</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>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)**

<table>
<thead>
<tr>
<th></th>
<th>849,270.344</th>
<th>192,825.168</th>
<th>0.078</th>
<th>147,832.141</th>
<th>62,621.576</th>
</tr>
</thead>
<tbody>
<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>
</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>
</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>
</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>
</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>
</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>
</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>
</tr>
</tbody>
</table>
What are the I/O Characteristics?

<table>
<thead>
<tr>
<th>Name</th>
<th>Total</th>
<th>MeanValue</th>
<th>NumSamples</th>
<th>MinValue</th>
<th>MaxValue</th>
<th>Std. Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>.TAU application</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>read()</td>
<td>25,235</td>
<td>1,097.174</td>
<td>23</td>
<td>11</td>
<td>12,032</td>
<td>2,851.143</td>
</tr>
<tr>
<td>fopen64()</td>
<td>22,707</td>
<td>1,746.592</td>
<td>13</td>
<td>11</td>
<td>12,032</td>
<td>3,660.642</td>
</tr>
<tr>
<td>fclose()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OurMain {{wrapper.py}[3]}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>read()</td>
<td>3,877</td>
<td>323.083</td>
<td>12</td>
<td>32</td>
<td>981</td>
<td>252.72</td>
</tr>
<tr>
<td>free size</td>
<td>1,536</td>
<td>194.429</td>
<td>7</td>
<td>32</td>
<td>464</td>
<td>148.122</td>
</tr>
<tr>
<td>fopen64()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fclose()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;module&gt; [[obe.py][8]]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>writeRestartData</td>
<td>4,195,104</td>
<td>17,927.795</td>
<td>234</td>
<td>1</td>
<td>1,048,576</td>
<td>133,362.946</td>
</tr>
<tr>
<td>samarcWriteRestartData</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>write()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WRITE Bandwidth</td>
<td>74.565</td>
<td>117</td>
<td>0</td>
<td>2,156,889</td>
<td>246,386</td>
<td></td>
</tr>
<tr>
<td>WRITE Bandwidth</td>
<td>77.594</td>
<td>117</td>
<td>0</td>
<td>1,941.2</td>
<td>228,366</td>
<td></td>
</tr>
<tr>
<td>WRITE Bandwidth</td>
<td>76.08</td>
<td>234</td>
<td>0</td>
<td>2,156,889</td>
<td>237,551</td>
<td></td>
</tr>
<tr>
<td>Bytes Written</td>
<td>2,097,552</td>
<td>17,927.795</td>
<td>117</td>
<td>1</td>
<td>1,048,576</td>
<td>133,362.946</td>
</tr>
<tr>
<td>Bytes Written</td>
<td>2,097,552</td>
<td>17,927.795</td>
<td>117</td>
<td>1</td>
<td>1,048,576</td>
<td>133,362.946</td>
</tr>
<tr>
<td>Bytes Written</td>
<td>4,195,104</td>
<td>17,927.795</td>
<td>234</td>
<td>1</td>
<td>1,048,576</td>
<td>133,362.946</td>
</tr>
<tr>
<td>open64()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

% 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>

Metric: TIME  
Value: Exclusive  
Units: seconds  

- **Message sizes**
- **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)
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

**ParaTools**

ATPSEC'16, Copyright © ParaTools, Inc.
What Caused My Application to Crash?

Right-click to see source code
What Caused My Application to Crash?

```c
/*
 * Take a timestep - advance solution from "time" to "time + dt"
 */
void SAMINT::timestep(const double time,  
    const double dt)
{
    cout << "SAMINT::timestep()" << endl;  
    timestep_(time, dt);  
    int x = 4 / (4-4);  
    cout <<" x = "<<x<<endl;
}
/*
 * Write data to output
 * (visit, fieldview, or overgrid - set in samarc input file)
 */
void SAMINT::writePlotData(const double time,  
    const int step)
{
    cout << "SAMINT::writePlotData()" << endl;
}
```
Intuitive Performance Engineering

TAU COMMANDER
(ALPHA RELEASE!)
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.

```
43° 74' 35" N
69° 39' 15" W
```
# TAU Commander CLI Dashboard

```plaintext
ejlinford@east03 ~/workspace/taucmdr/examples/mm $ tau dash
-- Targets (/home/jlinford/tau)

Name | Host OS | Host Arch | C | C++ | Fortran | In Projects
-----------------------------------------------
localhost | Linux | x86_64 | /usr/bin/gcc | /usr/bin/g++ | /usr/bin/gfortran | ex-mm

-- Applications (/home/jlinford/tau)

Name | OpenMP | Pthreads | MPI | CUDA | MIC | SHMEM | MPC | In Projects
-------------------|----------|----------|-----|------|-----|--------|-----|-----------------
ex-mm-serial | ex-mm |
ex-mm-openmp | ex-mm |
ex-mm-openmp mpi | Yes | Yes | ex-mm |

-- Measurements (/home/jlinford/tau)

Name | Profile | Trace | Sample | Source Inst. | Compiler Inst. | MPI | OpenMP | Callpath Depth | Mem. Usage | Mem. Alloc | In Projects
-----------------|---------|-------|--------|--------------|----------------|-----|--------|---------------|-----------|-----------|-----------------
ex-profile | Yes | No | No automatic | fallback | No | compiler_default | 0 | No | No | ex-mm
ex-trace | No | Yes | No automatic | fallback | No | compiler_default | 0 | No | No | ex-mm
ex-sample | Yes | No | Yes | never | never | No | compiler_default | 0 | No | No | ex-mm

-- Projects (/home/jlinford/tau)

Name | Targets | Applications | Measurements | Home
---------------------|----------|-------------|--------------|------
ex-mm | localhost | ex-mm-serial | ex-profile | /home/jlinford/tau
| | ex-mm-openmp | ex-trace | ex-sample
| | ex-mm-openmp mpi | ex-sample |

**ex-mm (localhost, ex-mm-openmp, ex-profile) Trials**

No trials. Use `tau <command>` or `tau trial create <command>` to create a new trial.
```
```
TAU Commander CLI

This command’s usage

Subcommand usage

Shortcuts

This command's usage:

```
tau [arguments] <subcommand> [options]
```

Optional arguments:
- `--help` show this help message and exit
- `--verbose` Set logging level to DEBUG
  - default: INFO

Configuration subcommands:
- `application` Create and manage application configurations.
- `measurement` Create and manage measurement configurations.
- `project` Create and manage project configurations.
- `target` Create and manage target configurations.

Subcommands:
- `build` Instrument programs during compilation and/or linking.
- `dashboard` Show all projects and their components.
- `help` Show help for a command or suggest actions for a file.
- `make` Instrument programs during compilation and/or linking with `./make`.
- `trial` Create and manage experiment trials.

Shortcuts:
- `tau <compiler>` Execute a compiler command
  - Example: `tau gcc *.c -o a.out`
  - Alias for `'tau build <compiler>'`
- `tau <program>` Gather data from a program
  - Example: `tau ./a.out`
  - Alias for `'tau trial create <program>'`
- `tau run <program>` Gather data from a program
  - Example: `tau ./a.out`
  - Alias for `'tau trial create <program>'`
- `tau show` Show data from the most recent trial
  - An alias for `'tau trial show'`

See `tau help <subcommand>` for more information on `<subcommand>`.
First use on a “vanilla” system

Put \texttt{tau} in front of every command

Detects, downloads, and installs required dependencies

Configures environment, wraps compiler
Executions create experiment trials

---

Put `tau` in front of every command

```
`tau show` to see data from last trial
```
Executions create experiment trials

Each execution is a new trial
Changing from serial to MPI+OpenMP

```
== Projects (/home/jlinford/tau) ==

<table>
<thead>
<tr>
<th>Name</th>
<th>Targets</th>
<th>Applications</th>
<th>Measurements</th>
<th>Home</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex-mm</td>
<td>localhost</td>
<td>ex-mm-serial</td>
<td>ex-profile</td>
<td>/home/jlinford/tau</td>
</tr>
<tr>
<td></td>
<td>localhost-openmp</td>
<td>ex-mm-openmp</td>
<td>ex-trace</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ex-mm-openmp-mpi</td>
<td>ex-sample</td>
<td></td>
</tr>
</tbody>
</table>

== ex-mm (localhost-openmp, ex-mm-openmp-mpi, ex-profile) Trials ==

No trials. Use 'tau <command>' or 'tau trial create <command>' to create a new trial

jlinford@east03 ~/workspace/taucmdr/examples/mm $ tau mpicc *.c -fopenmp -o mm
Installing TAU at '/home/jlinford/.tau/TAU/' from 'http://tau.uoregon.edu/tau.tgz' with --platform 64 and MPI compilers
Using TAU source archive at '/home/jlinford/.tau/src/tau.tgz'
Reusing TAU source files found at '/home/jlinford/.tau/src/.tau-2.24.1'
Configuring TAU with -iowrapper...
Compiling and installing TAU...
TAU installation complete
tau_cc.sh matmult.c matmult_initialize.c -fopenmp -o mm
jlinford@east03 ~/workspace/taucmdr/examples/mm $
```

Automatically reconfigures TAU for MPI+OpenMP

Put tau in front of every command
Workflow is unchanged

`tau show` to see data from last trial
Intuitive Performance Engineering

CASE STUDIES
Strand Technology

Technology Drivers

- **Timeliness** (automation of mesh generation)
- **Timeliness** (automation and scalability of domain connectivity)
- **Timeliness/Physical accuracy** (computational efficiency and scalability of aerodynamic solvers)
- **Processor architecture** (small memory footprint maps well to hierarchical memory architectures, e.g., multi-core, GPU)

CREATE-AV Example

This is a new meshing paradigm introduced in 2007 by current members of the CREATE-AV technical staff. The technology is being matured in the Helios product and will be deployed through both Helios and Kestrel.
Target Platforms

Armstrong [XC30]

Haise [iDataPlex]

Lightning [XC30]

Kilrain [iDataPlex]

Cori
Initial Profile on Babbage

MPI_Barrier

MPI_Send

File I/O

Useful Work!
Hot Spot Optimization

Useful work!

MPI_Waitall

Useful work!

File I/O
65% Runtime Reduction (~2x faster)
Cray XC30

Slower! What happened???
No worries, I fix it
• INCITE magnetohydrodynamcis 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
• Oversubscribe nodes: 32k ranks vs. 16k per node
• Overall time improvement: 71.23% of original
CONCLUSION
Downloads

http://tau.uoregon.edu

http://github.com/ParaToolsInc/taucmdr

http://www.hpclinux.com

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
REFERENCE
Online References

- **PAPI:**
  - PAPI documentation is available from the PAPI website:

- **TAU:**
  - TAU Users Guide and papers available from the TAU website:
    [http://tau.uoregon.edu/](http://tau.uoregon.edu/)

- **VAMPIR:**
  - VAMPIR website:

- **Scalasca:**
  - Scalasca documentation page:

- **Eclipse PTP:**
  - Documentation available from the Eclipse PTP website:
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=$TAU/Makefile.tau-bqgtimers-papi-mpi-pdt
% export TAU_OPTIONS=‘-optTauSelectFile=select.tau -optVerbose’
% cat select.tau
   BEGIN_INSTRUMENT_SECTION
   loops routine="#"
   END_INSTRUMENT_SECTION
%
% export PATH=$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.
Tracking I/O in static binaries

% 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.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

- **-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

- **-optMemDbg**
  Runtime bounds checking (see TAU_MEMDBG_* env vars)

- **-optKeepFiles**
  Does not remove intermediate .pdb and .inst.* files

- **-optPreProcess**
  Preprocess sources (OpenMP, Fortran) before instrumentation

- **-optTauSelectFile”<file>”**
  Specify selective instrumentation file for tau_instrumentor

- **-optTauWrapFile”<file>”**
  Specify path to link_options.tau generated by tau_gen_wrapper

- **-optHeaderInst**
  Enable Instrumentation of headers

- **-optTrackUPCR**
  Track UPC runtime layer routines (used with tau_upc.sh)

- **-optPdtF95Opts””**
  Add options for Fortran parser in PDT (f95parse/gfparse) ...
## Runtime Environment Variables

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAU_TRACE</td>
<td>0</td>
<td>Setting to 1 turns on tracing</td>
</tr>
<tr>
<td>TAU_CALLPATH</td>
<td>0</td>
<td>Setting to 1 turns on callpath profiling</td>
</tr>
<tr>
<td>TAU_TRACK_MEMORY_LEAKS</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>TAUPROFILE_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>