TAU PERFORMANCE SYSTEM

Wyatt Spear
Sameer Shende, Allen D. Malony
University of Oregon
What Is Tau?

• Tuning and Analysis Utilities (15+ year project)
• Performance problem solving framework for HPC
  - Integrated, scalable, flexible, portable
  - Target all parallel programming / execution paradigms
• Integrated performance toolkit (open source)
  - Instrumentation, measurement, analysis, visualization
  - Widely-ported performance profiling / tracing system
  - Performance data management and data mining
• Broad application use (NSF, DOE, DOD, …)
TAU Instrumentation / Measurement

Instrumentation

- source code
- object code
- library wrapper
- binary code
- virtual machine

Measurement

Event creation and management
- event identifier
- entry/exit events
- atomic events
- event mapping
- event control

Profiling
- statistics
- atomic profiles
- entry/exit profiles
- I/O profiles
- profile sampling

Tracing
- trace buffering
- record creation
- trace I/O
- timestamp generation
- trace filtering
- trace merging

Performance data sources
- timing
- hardware counters
- system counters
- kernel

OS and runtime system modules
- threading
- interrupts
- runtime system
- I/O
TAU Measurement Approach

• Portable and scalable parallel profiling solution
  - Multiple profiling types and options
  - Event selection and control (enabling/disabling, throttling)
  - Online profile access and sampling
  - Online performance profile overhead compensation

• Portable and scalable parallel tracing solution
  - Trace translation to OTF, EPILOG, Paraver, and SLOG2
  - Trace streams (OTF) and hierarchical trace merging

• Robust timing and hardware performance support
• Multiple counters (hardware, user-defined, system)
• Performance measurement of I/O and Linux kernel
TAU Measurement Mechanisms

- Parallel profiling
  - Function-level, block-level, statement-level
  - Supports user-defined events and mapping events
  - Support for flat, callgraph/callpath, phase profiling
  - Support for parameter and context profiling
  - Support for tracking I/O and memory (library wrappers)
  - Parallel profile stored (dumped, snapshot) during execution

- Tracing
  - All profile-level events
  - Inter-process communication events
  - Inclusion of multiple counter data in traced events
Types of Parallel Performance Profiling

- **Flat profiles**
  - Metric (e.g., time) spent in an event (callgraph nodes)
  - Exclusive/inclusive, # of calls, child calls

- **Callpath profiles (Calldesent profiles)**
  - Time spent along a calling path (edges in callgraph)
  - "main=> f1 => f2 => MPI_Send" (event name)
  - TAU_CALLPATH_DEPTH environment variable

- **Phase profiles**
  - Flat profiles under a phase (nested phases are allowed)
  - Default "main" phase
  - Supports static or dynamic (per-iteration) phases
  - Phase profiles may be generated from full callpath profiles in paraprof by choosing events as phases
Direct Performance Observation

- Execution actions of interest 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 performance system to observe
- Observation is direct
  - Direct instrumentation of program (system) code (probes)
  - Instrumentation invokes performance measurement
  - Event measurement: performance data, meta-data, context
- Performance experiment
  - Actual events + performance measurements
- Contrast with (indirect) event-based sampling
• Support for standard program events
  - Routines, classes and templates
  - Statement-level blocks
  - Begin/End events (Interval events)
• Support for user-defined events
  - Begin/End events specified by user
  - Atomic events (e.g., size of memory allocated/freed)
  - Flexible selection of event statistics
• Provides static events and dynamic events
• Enables “semantic” mapping
• Specification of event groups (aggregation, selection)
• Instrumentation optimization
TAU Instrumentation Mechanisms

- **Source code**
  - Manual (TAU API, TAU component API)
  - Automatic (robust)
    - C, C++, F77/90/95 (Program Database Toolkit (PDT))
    - OpenMP (directive rewriting (Opari), POMP2 spec)
    - Library header wrapping
- **Object code**
  - Pre-instrumented libraries (e.g., MPI using PMPI)
  - Statically- and dynamically-linked (with LD_PRELOAD)
- **Executable code**
  - Binary and dynamic instrumentation (Dyninst)
  - Virtual machine instrumentation (e.g., Java using JVMPI)
- TAU_COMPILER to automate instrumentation process
Automatic Source-level Instrumentation

TAU source analyzer

Parsed program

tau_instrumentor

Instrumentation specification file

Application source

Instrumented source

TAU 2011
Program Database Toolkit (PDT)

- Application / Library
- C / C++ parser
- Fortran parser F77/90/95
- IL analyzer
- Fortran IL analyzer
- Program Database Files
- DUCTAPE
- TAU instrumentor
- Automatic source instrumentation
Selective Instrumentation File

- Specify a list of events to exclude or include
- # is a wildcard in a routine name

BEGIN_EXCLUDE_LIST
Foo
Bar
D#EMM
END_EXCLUDE_LIST

BEGIN_INCLUDE_LIST
int main(int, char **)
F1
F3
END_INCLUDE_LIST
Selective Instrumentation File

- Optionally specify a list of files
- * and ? may be used as wildcard characters

BEGIN_FILE_EXCLUDE_LIST
f*.f90
Foo?.cpp
END_FILE_EXCLUDE_LIST
BEGIN_FILE_INCLUDE_LIST
main.cpp
foo.f90
END_FILE_INCLUDE_LIST
Selective Instrumentation File

- User instrumentation commands
  - Placed in INSTRUMENT section
  - Routine entry/exit
  - Arbitrary code insertion
  - Outer-loop level instrumentation

- 

  BEGIN_INSTRUMENT_SECTION
  loops file="foo.f90" routine="matrix#"
  memory file="foo.f90" routine="#"
  io routine="matrix#"
  [static/dynamic] phase routine="MULTIPLY"
  dynamic [phase/timer] name="foo" file="foo.cpp" line=22
  to line=35
  file="foo.f90" line = 123 code = " print *, \" Inside foo\"
  exit routine = "int foo()" code = "cout <<\"exiting foo\"<<endl;"
END_INSTRUMENT_SECTION
MPI Wrapper Interposition Library

- Uses standard MPI Profiling Interface
  - Provides name shifted interface
  - MPI_Send = PMPI_Send
  - Weak bindings

- Create TAU instrumented MPI library
  - Interpose between MPI and TAU
  - Done during program link
  - -lmpi replaced by -lTauMpi -lpmpi -lmpi
  - No change to the source code!
  - Just re-link application to generate performance data
MPI Shared Library Instrumentation

- Interpose the MPI wrapper library for applications that have already been compiled
  - Avoid re-compilation or re-linking
- Requires shared library MPI
  - Uses LD_PRELOAD for Linux
  - On AIX use MPI_EUILIB / MPI_EUILIBPATH
  - Does not work on XT3
- Approach will work with other shared libraries
- Use TAU tauex
  - % mpirun -np 4 tauex a.out
Performance Analysis

- Analysis of parallel profile and trace measurement
- Parallel profile analysis (ParaProf)
  - Java-based analysis and visualization tool
  - Support for large-scale parallel profiles
- Performance data management framework (PerfDMF)
- Parallel trace analysis
  - Translation to VTF (V3.0), EPILOG, OTF formats
  - Integration with Vampir / Vampir Server (TU Dresden)
  - Profile generation from trace data
- Online parallel analysis and visualization
- Integration with CUBE browser (Scalasca, UTK / FZJ)
ParaProf – Flat Profile

Metric Name: Time
Value Type: exclusive

node, context, thread

8K processors

Miranda
- hydrodynamics
- Fortran + MPI
- LLNL BG/L
### Interval Events, Atomic Events in TAU

**Interval event**
- e.g., routines (start/stop)

```plaintext
NODE 0:CONTEXT 0:THREAD 0:

<table>
<thead>
<tr>
<th>%Time</th>
<th>Exclusive msec</th>
<th>Inclusive total msec</th>
<th>#Call</th>
<th>#Subrs</th>
<th>Inclusive Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0</td>
<td>0.187</td>
<td>1.105</td>
<td>1</td>
<td>44</td>
<td>int main(int char **) C</td>
<td></td>
</tr>
<tr>
<td>93.2</td>
<td>1.030</td>
<td>1.030</td>
<td>1</td>
<td>0</td>
<td>MPI_Init()</td>
<td></td>
</tr>
<tr>
<td>5.9</td>
<td>0.879</td>
<td>65</td>
<td>40</td>
<td>320</td>
<td>void func(int, int) C</td>
<td></td>
</tr>
<tr>
<td>4.6</td>
<td>51</td>
<td>51</td>
<td>40</td>
<td>0</td>
<td>MPI_Barrier()</td>
<td></td>
</tr>
<tr>
<td>1.2</td>
<td>13</td>
<td>13</td>
<td>120</td>
<td>0</td>
<td>MPI_Recv()</td>
<td></td>
</tr>
<tr>
<td>0.8</td>
<td>9</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>MPI_Finalize()</td>
<td></td>
</tr>
<tr>
<td>0.0</td>
<td>0.137</td>
<td>0.137</td>
<td>120</td>
<td>0</td>
<td>MPI_Send()</td>
<td></td>
</tr>
<tr>
<td>0.0</td>
<td>0.086</td>
<td>0.086</td>
<td>40</td>
<td>0</td>
<td>MPI_Bcast()</td>
<td></td>
</tr>
<tr>
<td>0.0</td>
<td>0.002</td>
<td>0.002</td>
<td>1</td>
<td>0</td>
<td>MPI_Comm_size()</td>
<td></td>
</tr>
<tr>
<td>0.0</td>
<td>0.001</td>
<td>0.001</td>
<td>1</td>
<td>0</td>
<td>MPI_Comm_rank()</td>
<td></td>
</tr>
</tbody>
</table>
```

**Atomic events**
- (trigger with value)

```plaintext
USER EVENTS Profile :NODE 0, CONTEXT 0, THREAD 0

<table>
<thead>
<tr>
<th>NumSamples</th>
<th>MaxValue</th>
<th>MinValue</th>
<th>MeanValue</th>
<th>Std. Dev.</th>
<th>Event Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>365</td>
<td>5.138E+04</td>
<td>44.39</td>
<td>3.09E+04</td>
<td>1.234E+04</td>
<td>Heap Memory Used (KB) : Entry</td>
</tr>
<tr>
<td>365</td>
<td>5.138E+04</td>
<td>2064</td>
<td>3.115E+04</td>
<td>1.21E+04</td>
<td>Heap Memory Used (KB) : Exit</td>
</tr>
<tr>
<td>40</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>0</td>
<td>Message size for broadcast</td>
</tr>
</tbody>
</table>
```

% setenv TAU_CALLPATH_DEPTH 0
% setenv TAU_TRACK_HEAP 1
Atomic Events, Context Events

<table>
<thead>
<tr>
<th>%Time</th>
<th>Exclusive msec</th>
<th>Inclusive total msec</th>
<th>#Call</th>
<th>#Subs</th>
<th>Inclusive usec/call</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0</td>
<td>0.253</td>
<td>1.106</td>
<td>1</td>
<td>44</td>
<td>1106701 int main(int , char **) C</td>
<td></td>
</tr>
<tr>
<td>93.2</td>
<td>1.031</td>
<td>1.031</td>
<td>1</td>
<td>0</td>
<td>1031311 MPI_Init()</td>
<td></td>
</tr>
<tr>
<td>6.0</td>
<td>1</td>
<td>66</td>
<td>40</td>
<td>320</td>
<td>1650 void func(int , int) C</td>
<td></td>
</tr>
<tr>
<td>5.7</td>
<td>63</td>
<td>63</td>
<td>40</td>
<td>0</td>
<td>1588 MPI_Barrier()</td>
<td></td>
</tr>
<tr>
<td>0.8</td>
<td>9</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>9119 MPI_Finalize()</td>
<td></td>
</tr>
<tr>
<td>0.1</td>
<td>1</td>
<td>120</td>
<td>0</td>
<td>10</td>
<td>MPI_Recv()</td>
<td></td>
</tr>
<tr>
<td>0.0</td>
<td>0.141</td>
<td>0.141</td>
<td>120</td>
<td>0</td>
<td>1 MPI_Send()</td>
<td></td>
</tr>
<tr>
<td>0.0</td>
<td>0.085</td>
<td>0.085</td>
<td>40</td>
<td>0</td>
<td>2 MPI_Bcast()</td>
<td></td>
</tr>
<tr>
<td>0.0</td>
<td>0.001</td>
<td>0.001</td>
<td>1</td>
<td>0</td>
<td>1 MPI_Comm_size()</td>
<td></td>
</tr>
<tr>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0 MPI_Comm_rank()</td>
<td></td>
</tr>
</tbody>
</table>

USER EVENTS Profile : NODE 0. CONTEXT 0. THREAD 0

<table>
<thead>
<tr>
<th>NumSamples</th>
<th>MaxValue</th>
<th>MinValue</th>
<th>MeanValue</th>
<th>Std. Dev.</th>
<th>Event Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>5.139E+04</td>
<td>44.39</td>
<td>3.091E+04</td>
<td>1.234E+04</td>
<td>Message size for broadcast</td>
</tr>
<tr>
<td>365</td>
<td>1.13E+04</td>
<td>1.13E+04</td>
<td>3.134E+04</td>
<td>1.187E+04</td>
<td>Heap Memory Used (KB) : Entry</td>
</tr>
<tr>
<td>40</td>
<td>5.139E+04</td>
<td>5.139E+04</td>
<td>5.139E+04</td>
<td>0.0008905</td>
<td>MPI_Barrier()</td>
</tr>
<tr>
<td>1</td>
<td>2067</td>
<td>2067</td>
<td>2067</td>
<td>0</td>
<td>Heap Memory Used (KB) : Entry</td>
</tr>
<tr>
<td>1</td>
<td>2066</td>
<td>2066</td>
<td>2066</td>
<td>0</td>
<td>MPI_Finalize()</td>
</tr>
<tr>
<td>1</td>
<td>5.139E+04</td>
<td>5.139E+04</td>
<td>5.139E+04</td>
<td>0.0008905</td>
<td>Heap Memory Used (KB) : Entry</td>
</tr>
<tr>
<td>1</td>
<td>57.56</td>
<td>57.56</td>
<td>57.56</td>
<td>0</td>
<td>MPI_Init()</td>
</tr>
<tr>
<td>120</td>
<td>1.13E+04</td>
<td>1.13E+04</td>
<td>3.134E+04</td>
<td>1.187E+04</td>
<td>Heap Memory Used (KB) : Entry</td>
</tr>
<tr>
<td>120</td>
<td>1.129E+04</td>
<td>1.129E+04</td>
<td>1.187E+04</td>
<td>0</td>
<td>Heap Memory Used (KB) : Entry</td>
</tr>
<tr>
<td>1</td>
<td>44.39</td>
<td>44.39</td>
<td>44.39</td>
<td>0</td>
<td>Heap Memory Used (KB) : Entry</td>
</tr>
<tr>
<td>40</td>
<td>5.036E+04</td>
<td>2068</td>
<td>3.011E+04</td>
<td>1.227E+04</td>
<td>Heap Memory Used (KB) : Entry</td>
</tr>
</tbody>
</table>

% setenv TAU_CALLPATH_DEPTH 1
% setenv TAU_TRACK_HEAP 1
Configuring TAU

- TAU can measure several metrics with profiling and tracing approaches
- Different tools can also be invoked to instrument programs for TAU measurement
- Each configuration of TAU produces a measurement library for an architecture
- Each measurement configuration of TAU also creates a corresponding stub makefile that can be used to compile programs
- Typically configure multiple measurement libraries
cd /soft/apps/tau/tau_latest/bgp/lib/; ls Makefile.tau-*
  •  Makefile.tau-bgptimers-gnu-mpi-python-pdt
  •  Makefile.tau-bgptimers-gnu-papi-mpi-pdt
  •  Makefile.tau-bgptimers-gnu-papi-mpi-pdt-openmp-opari
  •  Makefile.tau-bgptimers-mpi-pdt...
  •  For an MPI+F90 application, you may want to start with:
     - Makefile.tau-bgptimers-mpi-pdt
     - Supports MPI instrumentation & PDT for automatic source instrumentation
  •  export
     TAU_MAKEFILE=/soft/apps/tau/tau_latest/bgp/lib/Makefile.tau-bgptimers-mpi-pdt
Using TAU: A brief Introduction

• To instrument source code using PDT
  - Choose an appropriate TAU stub makefile in <arch>/lib:
    % setenv TAU_MAKEFILE
    /opt/tau-2.19.1/x86_64/lib/Makefile.tau-mpi-pdt

    % setenv TAU_OPTIONS ‘-optVerbose ...’ (see tau_compiler.sh)

    And use tau_f90.sh, tau_cxx.sh or tau_cc.sh as Fortran, C++ or C
    compilers:

    % mpif90 foo.f90

    changes to

    % tau_f90.sh foo.f90

• Execute application and analyze performance data:
  % pprof (for text based profile display)
  % paraprof (for GUI)
Application Build Environment

- Minimize impact on user’s application build procedures
- Handle parsing, instrumentation, compilation, linking
- Dealing with Makefiles
  - Minimal change to application Makefile
  - Avoid changing compilation rules in application Makefile
  - No explicit inclusion of rules for process stages
- Some applications do not use Makefiles
  - Facilitate integration in whatever procedures used
- Two techniques:
  - TAU shell scripts (tau <compiler>.sh)
  - Invokes all PDT parser, TAU instrumenter, and compiler
  - TAU_COMPILER
Optional parameters for TAU_OPTIONS: [tau_compiler.sh -help]

- **-optVerbose**
  Turn on verbose debugging messages

- **-optCompInst**
  Use compiler based instrumentation

- **-optDetectMemoryLeaks**
  Turn on debugging memory allocations/de-allocations to track leaks

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

- **-optPreProcess**
  Preprocess Fortran sources before instrumentation

- **-optTauSelectFile=""**
  Specify selective instrumentation file for tau_instrumentor

- **-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)

- **-optTauSelectFile=""**
  Specify selective instrumentation file for tau_instrumentor

- **-optNoCompInst**
  Do not revert to compiler-based instrumentation if source instrumentation fails

- **-optPdtF95Opts=""**
  Add options for Fortran parser in PDT (f95parse/gfparse)

- **-optPdtF95Reset=""**
  Reset options for Fortran parser in PDT (f95parse/gfparse)

- **-optPdtCOpts=""**
  Options for C parser in PDT (cparse). Typically
  $(TAU_MPI_INCLUDE) $(TAU_INCLUDE) $(TAU_DEFS)

- **-optPdtCxxOpts=""**
  Options for C++ parser in PDT (cxxparse). Typically
  $(TAU_MPI_INCLUDE) $(TAU_INCLUDE) $(TAU_DEFS)
Goal: What routines account for the most time?

Metric: P_VIRTUAL_TIME
Value: Exclusive
Units: seconds

- LEQ_IKSWEEP: 9647.318
- LEQ_BICGS0T: 4357.213
- LEQ_MATVECT: 2669.887
- SOLVE_SPECIES_EQ: 1777.752
- SOLVE_LIN_EQ: 1417.986
- PHYSICAL_PROP: 1028.448
- RRATES: 783.402
- LEQ_MSOLVET: 682.376
- INIT_AB_M: 530.858
- CALC_MASS_FLUX_SPHR: 463.788
- INIT_MU_S: 446.025
- CALC_RESID_S: 421.747
- SOLVE_ENERGY_EQ: 381.363
- SOURCE_PHI: 371.199
- DRAG_GS: 258.829
Solution: Generating a flat profile with MPI

% setenv TAU_MAKEFILE /opt/tau-2.19.1/x86_64
    /lib/Makefile.tau-mpi-pdt
% set path=('./opt/tau-2.19.1/x86_64/bin $path)

Or

% module load tau
% make F90=tau_f90.sh

Or

% tau_f90.sh matmult.f90 -o matmult
(Or edit Makefile and change F90=tau_f90.sh)

% qsub run.job
% paraprof

To view. To view the data locally on the workstation,
% paraprof --pack app.ppk
    Move the app.ppk file to your desktop.

% paraprof app.ppk
Goal: What loops account for the most time? How much?

Flat profile with wallclock time with loop instrumentation:

Metric: GET_TIME_OF_DAY
Value: Exclusive
Units: microseconds

<table>
<thead>
<tr>
<th>Time</th>
<th>Function</th>
<th>Loop Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>1729975.333</td>
<td>Loop: MULTIPLY_MATRICES [{matmult.f90} {31,9}-{36,14}]</td>
<td></td>
</tr>
<tr>
<td>443194</td>
<td>MPIRecv()</td>
<td>Loop: MULTIPLY_MATRICES [{matmult.f90} {31,9}-{36,14}]</td>
</tr>
<tr>
<td>81095</td>
<td>MAIN</td>
<td>Loop: MULTIPLY_MATRICES [{matmult.f90} {31,9}-{36,14}]</td>
</tr>
<tr>
<td>49569</td>
<td>MPI_Bcast()</td>
<td>Loop: MAIN [{matmult.f90} {86,9}-{106,14}]</td>
</tr>
<tr>
<td>45669</td>
<td>Loop: MAIN [{matmult.f90} {86,9}-{106,14}]</td>
<td>MPI_Send()</td>
</tr>
<tr>
<td>12412</td>
<td>MPI_Send()</td>
<td>Loop: INITIALIZE [{matmult.f90} {17,9}-{21,14}]</td>
</tr>
<tr>
<td>8959</td>
<td>Loop: INITIALIZE [{matmult.f90} {17,9}-{21,14}]</td>
<td>Loop: INITIALIZE [{matmult.f90} {10,9}-{14,14}]</td>
</tr>
<tr>
<td>8953</td>
<td>Loop: INITIALIZE [{matmult.f90} {10,9}-{14,14}]</td>
<td>Loop: INITIALIZE [{matmult.f90} {10,9}-{14,14}]</td>
</tr>
<tr>
<td>5609.2</td>
<td>MPI_Finalize()</td>
<td>Loop: MULTIPLY_MATRICES</td>
</tr>
<tr>
<td>2932.667</td>
<td>MULTIPLY_MATRICES</td>
<td>Loop: MULTIPLY_MATRICES</td>
</tr>
<tr>
<td>2577.667</td>
<td>Loop: MAIN [{matmult.f90} {117,9}-{128,14}]</td>
<td>Loop: MAIN [{matmult.f90} {117,9}-{128,14}]</td>
</tr>
<tr>
<td>2091.8</td>
<td>MPI_BARRIER()</td>
<td>Loop: MAIN [{matmult.f90} {112,9}-{115,14}]</td>
</tr>
<tr>
<td>1875.667</td>
<td>Loop: MAIN [{matmult.f90} {112,9}-{115,14}]</td>
<td>Loop: MAIN [{matmult.f90} {112,9}-{115,14}]</td>
</tr>
<tr>
<td>1833</td>
<td>Loop: MAIN [{matmult.f90} {71,9}-{74,14}]</td>
<td>Loop: MAIN [{matmult.f90} {71,9}-{74,14}]</td>
</tr>
<tr>
<td>107</td>
<td>Loop: MAIN [{matmult.f90} {77,9}-{84,14}]</td>
<td>Loop: MAIN [{matmult.f90} {77,9}-{84,14}]</td>
</tr>
<tr>
<td>30</td>
<td>INITIALIZE</td>
<td>Loop: MAIN [{matmult.f90} {77,9}-{84,14}]</td>
</tr>
<tr>
<td>14.25</td>
<td>MPI_Comm_rank()</td>
<td>Loop: MAIN [{matmult.f90} {77,9}-{84,14}]</td>
</tr>
<tr>
<td>1</td>
<td>MPI_Comm_size()</td>
<td>Loop: MAIN [{matmult.f90} {77,9}-{84,14}]</td>
</tr>
</tbody>
</table>
Solution: Generating a loop level profile

% setenv TAU_MAKEFILE /opt/tau-2.19.1/x86_64
    /lib/Makefile.tau-mpi-pdt
% setenv TAU_OPTIONS `-optTauSelectFile=select.tau –optVerbose’
% cat select.tau
    BEGIN_INSTRUMENT_SECTION
    loops routine="#"
    END_INSTRUMENT_SECTION

% module load tau
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% qsub run.job
% paraprof --pack app.ppk
    Move the app.ppk file to your desktop.
% paraprof app.ppk
Using tau_exec

```bash
> cd ~/workshop-point/matmult
> mpif90 matmult.f90 -o matmult
> mpirun -np 4 ./matmult
>
> # To use tau_exec to measure the I/O and memory usage:
> mpirun -np 4 tau_exec -io -memory ./matmult
>
> # To measure memory leaks and get complete callpaths
> setenv TAU_TRACK_MEMORY_LEAKS 1
> setenv TAU_CALLPATH_DEPTH 100
> mpirun -np 4 tau_exec -io -memory ./matmult
> paraprof
> # Right click on a given rank (e.g. "node 2") and choose "Show Context Event
> # Window" and expand the ".TAU Application" node to see the callpath
> # To use a different configuration (e.g., Makefile.tau-papi-mpi-pdt)
> setenv TAU_METRICS TIME:PAPI_FP_INS:PAPI_L1_DCM
> mpirun -np 4 tau_exec -io -memory -T papi.mpi.pdt ./matmul
> # Using tau_exec with DyninstAPI:
> tau_run matmul -o matmul.i
> mpirun -np 4 tau_exec -io -memory ./matmul.i
>
> tau_run -XrunTAUsh-papi-mpi-pdt matmul -o matmul.i
> mpirun -np 4 tau_exec -io -memory -T papi.mpi.pdt ./matmul.i
> paraprof
```
Measuring Performance of PGI Accelerator Code

![Screenshot of TAU Profiler output](image)

### Table: TAU: ParaProf: n,c,t 0,0,0 - matlkk.ppk

<table>
<thead>
<tr>
<th>Name</th>
<th>Exclusive TIME</th>
<th>Inclusive TIME</th>
<th>Calls</th>
<th>Child Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>__pgi_cu_launch multiply_matrices (pgi_kernel_7, gx=32, gy=32, gz=1, bx=16, by=16, bz=1) [[mm2.f90][15]]</td>
<td>10.901</td>
<td>10.901</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>__pgi_cu_init multiply_matrices [[mm2.f90][9]]</td>
<td>3.912</td>
<td>3.912</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>__pgi_cu_download2 multiply_matrices var=a [[mm2.f90][20]]</td>
<td>0.514</td>
<td>0.514</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>__pgi_cu_upload2 multiply_matrices var=b [[mm2.f90][9]]</td>
<td>0.252</td>
<td>0.252</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>__pgi_cu_upload2 multiply_matrices var=c [[mm2.f90][9]]</td>
<td>0.252</td>
<td>0.252</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>__pgi_cu_launch multiply_matrices (pgi_kernel_2, gx=32, gy=32, gz=1, bx=16, by=16, bz=1) [[mm2.f90][11]]</td>
<td>0.125</td>
<td>16.021</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>__pgi_cu_free multiply_matrices [[mm2.f90][1]]</td>
<td>0.023</td>
<td>0.023</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>__pgi_cu_free multiply_matrices [[mm2.f90][9]]</td>
<td>0.02</td>
<td>0.02</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>__pgi_cu_alloc multiply_matrices [[mm2.f90][9]]</td>
<td>0.019</td>
<td>0.019</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>__pgi_cu_alloc multiply matrices [[mm2.f90][5,0]]</td>
<td>0.003</td>
<td>15.895</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>__pgi_accelerator region</td>
<td>0.001</td>
<td>15.893</td>
<td>5</td>
<td>85</td>
</tr>
<tr>
<td>__pgi_cu_module multiply_matrices [[mm2.f90][9]]</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>__pgi_cu_module_function multiply_matrices [[mm2.f90][11]]</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>__pgi_cu_paramset multiply_matrices [[mm2.f90]]</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
</tr>
</tbody>
</table>
Usage Scenarios: MFlops in Loops

• Goal: What execution rate do my application loops get in mflops?

• Flat profile with PAPI_FP_INS/OPS and time (-papi) with

Metric: PAPI_FP_INS / GET_TIME_OF_DAY
Value: Exclusive
Units: Derived metric shown in microseconds format

<table>
<thead>
<tr>
<th>Time (microseconds)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>770.699</td>
<td>Loop: MULTIPLY_MATRICES [{matmult.f90} {31.9}-{36.14}]</td>
</tr>
<tr>
<td>223.39</td>
<td>Loop: INITIALIZE [{matmult.f90} {10.9}-{14.14}]</td>
</tr>
<tr>
<td>223.24</td>
<td>Loop: INITIALIZE [{matmult.f90} {17.9}-{21.14}]</td>
</tr>
<tr>
<td>171.855</td>
<td>Loop: MAIN [{matmult.f90} {71.9}-{74.14}]</td>
</tr>
<tr>
<td>170.862</td>
<td>Loop: MAIN [{matmult.f90} {112.9}-{115.14}]</td>
</tr>
<tr>
<td>122.96</td>
<td>Loop: MAIN [{matmult.f90} {117.9}-{128.14}]</td>
</tr>
<tr>
<td>37.549</td>
<td>MULTIPLY_MATRICES</td>
</tr>
<tr>
<td>21.367</td>
<td>INITIALIZE</td>
</tr>
<tr>
<td>13.796</td>
<td>Loop: MAIN [{matmult.f90} {86.9}-{106.14}]</td>
</tr>
<tr>
<td>11</td>
<td>MPI_Comm_size()</td>
</tr>
<tr>
<td>8.935</td>
<td>Loop: MAIN [{matmult.f90} {77.9}-{84.14}]</td>
</tr>
<tr>
<td>1.131</td>
<td>MPI_Send()</td>
</tr>
<tr>
<td>0.794</td>
<td>MPI_Comm_rank()</td>
</tr>
<tr>
<td>0.647</td>
<td>MPI_Bcast()</td>
</tr>
<tr>
<td>0.355</td>
<td>MPI_Recv()</td>
</tr>
<tr>
<td>0.171</td>
<td>MPI_BARRIER()</td>
</tr>
<tr>
<td>0.115</td>
<td>MPI_Finalize()</td>
</tr>
<tr>
<td>0.023</td>
<td>MAIN</td>
</tr>
</tbody>
</table>
Generate a PAPI profile with 2 or more counters

```bash
% setenv TAU_MAKEFILE /opt/tau-2.19.1/x86_64
   /lib/Makefile.tau-papi-mpi-pdt
% setenv TAU_OPTIONS '-optTauSelectFile=select.tau -optVerbose'
% cat select.tau
   BEGIN_INSTRUMENT_SECTION
   loops routine="#"
   END_INSTRUMENT_SECTION

% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% setenv TAU_METRICS TIME:PAPI_FP_INS
% qsub run.job
% paraprof --pack app.ppk
   Move the app.ppk file to your desktop.
% paraprof app.ppk
   Choose Options -> Show Derived Panel -> Arg 1 = PAPI_FP_INS,
   Arg 2 = GET_TIME_OF_DAY, Operation = Divide -> Apply, choose.
```
Goal: Easily generate routine level performance data using the compiler instead of PDT for parsing the source code.
Use Compiler-Based Instrumentation

```bash
% setenv TAU_MAKEFILE /opt/tau-2.19.1/x86_64
   /lib/Makefile.tau-mpi-pdt
% setenv TAU_OPTIONS '-optCompInst -optVerbose'
% module load tau
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)

% qsub run.job
% paraprof --pack app.ppk
  Move the app.ppk file to your desktop.
% paraprof app.ppk
```
-PROFILECALLPATH Option

- Generates profiles that show the calling order (edges and nodes in callgraph)
  - A=>B=>C shows the time spent in C when it was called by B and B was called by A
  - Control the depth of callpath using TAU_CALLPATH_DEPTH environment variable
  - -callpath in the name of the stub Makefile name or setting TAU_CALLPATH= 1 at runtime (TAU v2.18.1+)

TAU 2011
Callpath Profile

- Generates program callgraph
Generate a Callpath Profile

% setenv TAU_MAKEFILE /opt/tau-2.19.1/x86_64
   /lib/Makefile.tau-mpi-pdt
% set path=($opt/tau-2.19.1/x86_64/bin $path)
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% setenv TAU_CALLPATH 1
% setenv TAU_CALLPATH_DEPTH 100

to generate the callpath profiles without any recompilation.
% qsub run.job
% paraprof --pack app.ppk
Move the app.ppk file to your desktop.
% paraprof app.ppk
(Windows -> Thread -> Call Graph)
-DEPTHLIMIT Option

- Allows users to enable instrumentation at runtime based on the depth of a calling routine on a callstack
  - Disables instrumentation in all routines a certain depth away from the root in a callgraph
- TAUDEPTH_LIMIT environment variable specifies depth
  - % setenv TAUDEPTH_LIMIT 1
    - enables instrumentation in only “main”
  - % setenv TAUDEPTH_LIMIT 2
    - enables instrumentation in main and routines that are directly called by main

- Stub makefile has -depthlimit in its name:
  - setenv TAU_MAKEFILE <taudir>/<arch>/lib/Makefile.tau-mpi-depthlimit-pdt
Goal: What is the volume of inter-process communication? Along which calling path?
Communication Matrix Profiles

% export TAU_MAKEFILE = $TAU/Makefile.tau-mpi-pdt
% export PATH=/usr/local/packages/tau-2.19.1/x86_64/bin=$PATH
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% export TAU_COMM_MATRIX = 1

% qsub run.job (setting the environment variables)

% paraprof
(Windows -> Communication Matrix)
(Windows -> 3D Communication Matrix)
Usage Scenario: Detect Memory Leaks
Detect Memory Leaks

% setenv TAU_MAKEFILE /opt/tau-2.19.1/x86_64 /lib/Makefile.tau-mpi-pdt
% setenv TAU_OPTIONS '-optDetectMemoryLeaks -optVerbose'
% module load tau
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% setenv TAU_CALLPATH_DEPTH 100

% qsub run.job
% paraprof --pack app.ppk
  Move the app.ppk file to your desktop.
% paraprof app.ppk
(Windows -> Thread -> Context Event Window -> Select thread -> select...
  expand tree)
(Windows -> Thread -> User Event Bar Chart -> right click LEAK
  -> Show User Event Bar Chart)
NOTE: setenv TAU_TRACK_HEAP 1 and setenv TAU_TRACK_HEADROOM 1 may be used to track
heap and headroom utilization at the entry and exit of each routine.
TAU_CALLPATH_DEPTH=1 shows just the routine name, and 0 shows just one event for the
entire program.
Usage Scenarios: Mixed Python+F90+C+pyMPI

• Goal: Generate multi-level instrumentation for Python+MPI+C+F90+C++ ...
Generate a Multi-Language Profile w/ Python

% setenv TAU_MAKEFILE /opt/tau-2.19.1/x86_64
   /lib/Makefile.tau-python-mpi-pdt
% set path=/opt/tau-2.19.1/x86_64/bin $path)
% setenv TAU_OPTIONS `--optShared --optVerbose...`
(Python needs shared object based TAU library)
% make F90=tau_f90.sh CXX=tau_cxx.sh CC=tau_cc.sh (build pyMPI w/TAU)
% cat wrapper.py
   import tau
   def OurMain():
       import App
       tau.run('OurMain()')

Uninstrumented:
% poe <dir>/pyMPI-2.4b4/bin/pyMPI ./App.py --procs 4

Instrumented:
% setenv PYTHONPATH $taudir/x86_64/lib/bindings-python-mpi-pdt-pgi
(same options string as TAU_MAKEFILE)
setenv LD_LIBRARY_PATH $taudir/x86_64/lib/bindings-icpc-python-mpi-pdt-pgi:
% poe <dir>/pyMPI-2.5b0-TAU/bin/pyMPI ./wrapper.py --procs 4
(Instrumented pyMPI with wrapper.py)
Tracing in TAU

- Generates event-trace logs, rather than summary profiles
  - `setenv TAU_TRACE 1`
- Traces show when and where an event occurred in terms of location and the process that executed it
- Traces from multiple processes are merged:
  - `% tau_treemerge.pl`
  - generates `tau.trc` and `tau.edf` as merged trace and event definition file
- TAU traces can be converted to Vampir’s OTF/VTF3, Jumpshot SLOG2, Paraver trace formats:
  - `% tau2otf tau.trc tau.edf app.otf`
  - `% tau2vtf tau.trc tau.edf app.vpt.gz`
  - `% tau2slog2 tau.trc tau.edf -o app.slog2`
  - `% tau_convert -paraver tau.trc tau.edf app.prv`
% setenv TAU_MAKEFILE /opt/tau-2.19.1/x86_64
            /lib/Makefile.tau-mpi-pdt
% set path=/opt/tau-2.19.1/x86_64/bin $path
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% setenv TAU_TRACE 1
% qsub run.job
% tau_treemerge.pl
(merges binary traces to create tau.trc and tau.edf files)
JUMPSHOT:
% tau2slog2 tau.trc tau.edf -o app.slog2
% jumpshot app.slog2
   OR
VAMPIR:
% tau2otf tau.trc tau.edf app.otf -n 4 -z
(4 streams, compressed output trace)
% vampir app.otf
(or vng client with vngd server)
Trace Visualization
Jumpshot

- Developed at Argonne National Laboratory as part of the MPICH project
  - Also works with other MPI implementations
  - Installed on IBM BG/P
  - Jumpshot is bundled with the TAU package
- Java-based tracefile visualization tool for postmortem performance analysis of MPI programs
- Latest version is Jumpshot-4 for SLOG-2 format
  - Scalable level of detail support
  - Timeline and histogram views
  - Scrolling and zooming
  - Search/scan facility
Jumpshot
<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_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.</td>
</tr>
<tr>
<td>TAU_SYNCHRONIZE_CLOCKS</td>
<td>1</td>
<td>Synchronize clocks across nodes to correct timestamps in traces</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 separted list generates other metrics. (e.g., TIME:linuxtimers:PAPI_FP_OPS:PAPI_NATIVE_&lt;event&gt;)</td>
</tr>
</tbody>
</table>
How to explain performance?

• Should not just redescribe the performance results
• Should explain performance phenomena
  - What are the causes for performance observed?
  - What are the factors and how do they interrelate?
  - Performance analytics, forensics, and decision support
• Need to add knowledge to do more intelligent things
  - Automated analysis needs good informed feedback
  - Iterative tuning, performance regression testing
  - Performance model generation requires interpretation
• We need better methods and tools for
  - Integrating meta-information
  - Knowledge-based performance problem solving
Role of Metadata and Knowledge Role

You have to capture these...

Performance Knowledge

...to understand this

Execution

Performance Result

Context Knowledge

Source Code
Build Environment
Run Environment

Application
Machine

Performance Problems

TAU 2011
• Provide an open, flexible framework to support common data management tasks
  - Foster multi-experiment performance evaluation
• Extensible toolkit to promote integration and reuse across available performance tools (PerfDMF)
  - Originally designed to address critical TAU requirements
  - Supported profile formats:
    TAU, CUBE (Scalasca), HPC Toolkit (Rice), HPM Toolkit (IBM), gprof, mpiP, psrun (PerfSuite), OpenSpeedShop, ...
  - Supported DBMS:
    PostgreSQL, MySQL, Oracle, DB2, Derby/Cloudscape
  - Profile query and analysis API
• Reference implementation for PERI-DB project
PerfDMF Architecture

TAU Performance System

Performance Analysis Programs

- scalability analysis
- ParaProf
- cluster analysis

Query and Analysis Toolkit

- Data Mining (Weka)
- Statistics (R / Omega)

Java PerfDMF API

SQL (PostgreSQL, MySQL, DB2, Oracle)

Profile metadata

Raw profiles

* gprof
* mpiP
* psrun
* HPMtoolkit
* ...

XML document

Formatted profile data
Metadata Collection

- Integration of XML metadata for each parallel profile
- Three ways to incorporate metadata
  - Measured hardware/system information (TAU, PERI-DB)
    - CPU speed, memory in GB, MPI node IDs, ...
  - Application instrumentation (application-specific)
    - TAU_METADATA() used to insert any name/value pair
    - Application parameters, input data, domain decomposition
  - PerfDMF data management tools can incorporate an XML file of additional metadata
    - Compiler flags, submission scripts, input files, ...

- Metadata can be imported from / exported to PERI-DB
## Parallel Profile Analysis - pprof

<table>
<thead>
<tr>
<th></th>
<th>Exclusive</th>
<th>Inclusive</th>
<th>#Call</th>
<th>#Subrs</th>
<th>Inclusive Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0%</td>
<td>1</td>
<td>3:11.293</td>
<td>1</td>
<td>15</td>
<td>apcpu</td>
</tr>
<tr>
<td>99.6%</td>
<td>3,667</td>
<td>3:10.463</td>
<td>3</td>
<td>37517</td>
<td>bcast_inputs</td>
</tr>
<tr>
<td>67.1%</td>
<td>491</td>
<td>2:08.326</td>
<td>37200</td>
<td>37200</td>
<td>exchange_1</td>
</tr>
<tr>
<td>44.5%</td>
<td>6,481</td>
<td>1:25.169</td>
<td>9300</td>
<td>9300</td>
<td>bufs</td>
</tr>
<tr>
<td>41.0%</td>
<td>1:18.436</td>
<td>1:18.436</td>
<td>18000</td>
<td>0</td>
<td>4217 MPI_Recv()</td>
</tr>
<tr>
<td>29.5%</td>
<td>6,778</td>
<td>56,407</td>
<td>9300</td>
<td>9300</td>
<td>6065 bts</td>
</tr>
<tr>
<td>26.2%</td>
<td>50,142</td>
<td>50,142</td>
<td>19204</td>
<td>0</td>
<td>2611 MPI_Send()</td>
</tr>
<tr>
<td>16.2%</td>
<td>24,451</td>
<td>31,031</td>
<td>301</td>
<td>602</td>
<td>103096 rhs</td>
</tr>
<tr>
<td>3.9%</td>
<td>7,501</td>
<td>7,501</td>
<td>9300</td>
<td>0</td>
<td>807 jacid</td>
</tr>
<tr>
<td>3.4%</td>
<td>638</td>
<td>6,594</td>
<td>604</td>
<td>1812</td>
<td>10918 exchange_3</td>
</tr>
<tr>
<td>3.4%</td>
<td>6,590</td>
<td>6,590</td>
<td>9300</td>
<td>0</td>
<td>709 jacu</td>
</tr>
<tr>
<td>2.6%</td>
<td>4,989</td>
<td>4,989</td>
<td>608</td>
<td>0</td>
<td>8205 MPI_Wait()</td>
</tr>
<tr>
<td>0.2%</td>
<td>0.44</td>
<td>400</td>
<td>1</td>
<td>4</td>
<td>400081 init_comm</td>
</tr>
<tr>
<td>0.2%</td>
<td>398</td>
<td>399</td>
<td>1</td>
<td>39</td>
<td>399634 MPI_Init()</td>
</tr>
<tr>
<td>0.1%</td>
<td>140</td>
<td>247</td>
<td>1</td>
<td>47616</td>
<td>247986 setiv</td>
</tr>
<tr>
<td>0.1%</td>
<td>131</td>
<td>131</td>
<td>57252</td>
<td>0</td>
<td>2 exact</td>
</tr>
<tr>
<td>0.1%</td>
<td>89</td>
<td>103</td>
<td>1</td>
<td>2</td>
<td>103168 erhs</td>
</tr>
<tr>
<td>0.1%</td>
<td>0.966</td>
<td>96</td>
<td>1</td>
<td>2</td>
<td>96458 read_input</td>
</tr>
<tr>
<td>0.0%</td>
<td>95</td>
<td>95</td>
<td>9</td>
<td>0</td>
<td>10603 MPI_Bcast()</td>
</tr>
<tr>
<td>0.0%</td>
<td>26</td>
<td>44</td>
<td>1</td>
<td>7937</td>
<td>error</td>
</tr>
<tr>
<td>0.0%</td>
<td>24</td>
<td>24</td>
<td>608</td>
<td>0</td>
<td>40 MPI_Irecv()</td>
</tr>
<tr>
<td>0.0%</td>
<td>15</td>
<td>15</td>
<td>1</td>
<td>5</td>
<td>15630 MPI_Finalize()</td>
</tr>
<tr>
<td>0.0%</td>
<td>4</td>
<td>12</td>
<td>1</td>
<td>1700</td>
<td>12335 setbv</td>
</tr>
<tr>
<td>0.0%</td>
<td>7</td>
<td>8</td>
<td>3</td>
<td>3</td>
<td>2893 init</td>
</tr>
<tr>
<td>0.0%</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>491 MPI_Allreduce()</td>
</tr>
<tr>
<td>0.0%</td>
<td>1</td>
<td>3</td>
<td>6</td>
<td>1</td>
<td>3874 pinte</td>
</tr>
<tr>
<td>0.0%</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1007 MPI_Barrier()</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.116</td>
<td>0.837</td>
<td>1</td>
<td>4</td>
<td>837 exchange_4</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.512</td>
<td>0.512</td>
<td>1</td>
<td>0</td>
<td>512 MPI_Keyval_create()</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.121</td>
<td>0.353</td>
<td>1</td>
<td>2</td>
<td>353 exchange_5</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.024</td>
<td>0.191</td>
<td>1</td>
<td>2</td>
<td>191 exchange_6</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.103</td>
<td>0.103</td>
<td>0</td>
<td>1</td>
<td>47 MPI_Type_contiguous()</td>
</tr>
</tbody>
</table>

TAU 2011
Metadata for Each Experiment

Multiple PerfDMF DBs
Comparing Effects of Multi-Core Processors

AORSA2D

- magnetized plasma simulation
- Blue is single node
- Red is dual core
- Cray XT3 (4K cores)
Comparing FLOPS (AORSA2D, Cray XT3)

AORSA2D
- Blue is dual core
- Red is single node
- Cray XT3 (4K cores)
- Data generated by Richard Barrett, ORNL
ParaProf - Scalable Histogram

8k processors

16k processors
ParaProf – 3D View (Full Profile)

xbec

128k processors
ParaProf – 3D Scatterplot

- Each point is a “thread” of execution
- A total of four metrics shown in relation
- ParaProf’s visualization library
  - JOGL
- Miranda, 32k cores
ParaProf – 3D Topology View

- Thread/Node points from metidata or user defined
- Visibility selection by value and position

- Sweep3D 16k BGL Run
- Default Cartesian and User Defined Sphere Mappings
Performance Mapping

• Example: Particles distributed on

```c
Particle* P[MAX]; /* Array of particles */
int GenerateParticles() {
    /* distribute particles over all faces of the cube */
    for (int face=0, last=0; face < 6; face++) {
        /* particles on this face */
        int particles_on_this_face = num(face);
        for (int i=last; i < particles_on_this_face; i++) {
            /* particle properties are a function of face */
            P[i] = ... f(face);
            ...
        }
        last+= particles_on_this_face;
    }
}
```
No Mapping versus Mapping

- Typical performance tools report performance with respect to routines.
- Does not provide support for mapping.

- TAU’s performance mapping can observe performance with respect to scientist’s programming and problem abstractions.
How is MPI_Wait() distributed relative to solver direction?

Application routine names reflect phase semantics.

How is MPI_Wait() distributed relative to solver direction?
NAS BT – Phase Profile

Main phase shows nested phases and immediate events
Phase Profiling of HW Counters

- GTC particle-in-cell simulation of fusion turbulence
- Phases assigned to iterations
- Poor temporal locality for one important data
- Automatically generated by PE2 python script

Increasing phase execution time
Decreasing flops rate
Declining cache performance
Profile Snapshots in ParaProf

- Profile snapshots are parallel profiles recorded at runtime
- Shows performance profile dynamics (all types allowed)

![Diagram showing TAU measurement, application run on parallel system, and parallel profile snapshots with timeline and breakdown of snapshot data.]

- Initialization
- Checkpointing
- Finalization
Profile Snapshot Views

- Only show main loop
- Percentage breakdown
Snapshot Replay in ParaProf

All windows dynamically update
• Conduct systematic and scalable analysis process
  - Multi-experiment performance analysis
  - Support automation, collaboration, and reuse
• Performance knowledge discovery framework
  - Data mining analysis applied to parallel performance data
  - comparative, clustering, correlation, dimension reduction, ...
  - Use the existing TAU infrastructure
• PerfExplorer v1 performance data mining framework
  - Multiple experiments and parametric studies
  - Integrate available statistics and data mining packages
  - Weka, R, Matlab / Octave
  - Apply data mining operations in interactive environment
PerfExplorer v2 – Requirements

- Component-based analysis process
  - Analysis operations implemented as modules
  - Linked together in analysis process and workflow
- Scripting
  - Provides process/workflow development and automation
- Metadata input, management, and access
- Inference engine
  - Reasoning about causes of performance phenomena
  - Analysis knowledge captured in expert rules
- Persistence of intermediate analysis results
- Provenance
  - Provides historical record of analysis results
Usage Scenarios: Evaluate Scalability

- Goal: How does my application scale? What bottlenecks at what cpu counts?
- Load profiles in PerfDMF database and examine with PerfExplorer
Evaluate Scalability using PerfExplorer Charts

% setenv TAU_MAKEFILE /opt/tau-2.19.1/x86_64
  /lib/Makefile.tau-mpi-pdt
% set path=/opt/tau-2.19.1/x86_64/bin $path
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% qsub run1p.job
% paraprof --pack lp.ppk
% qsub run2p.job ...
% paraprof --pack 2p.ppk ... and so on.
On your client:
% perfdfm_configure
(Choose derby, blank user/passwd, yes to save passwd, defaults)
% perfexplorer_configure
(Yes to load schema, defaults)
% paraprof
(load each trial: DB -> Add Trial -> Type (Paraprof Packed Profile) -> OK, OR use perfdfm_loadtrial on the commandline)
% perfexplorer
(Charts -> Speedup)
PerfExplorer - Runtime Breakdown

Total Runtime Breakdown for S3D (Jaguar, ORNL): Harness Scaling Study:
GET_TIME_OF_DAY

MPI_Waitall
WRITE_SAVEFILE
PerfExplorer – Relative Comparisons

- Total execution time
- Timesteps per second
- Relative efficiency
- Relative efficiency per event
- Relative speedup
- Relative speedup per event
- Group fraction of total
- Runtime breakdown
- Correlate events with total runtime
- Relative efficiency per phase
- Relative speedup per phase
- Distribution visualizations
PerfExplorer – Correlation Analysis

Strong negative linear correlation between CALC_CUT_BLOCK_CONTRIBUTIONS and MPI_Barrier.
- 0.995 indicates strong, negative relationship

As `CALC_CUT_BLOCK_CONTRIBUTIONS()` increases in execution time, `MPI_Barrier()` decreases
PerfExplorer - Cluster Analysis
• Four significant events automatically selected
• Clusters and correlations are visible
PerfExplorer – Performance Regression
Other Projects in TAU

- TAU Portal
  - Support collaborative performance study
- Kernel-level system measurements (KTAU)
  - Application to OS noise analysis and I/O system analysis
- TAU performance monitoring
  - TAUoverSupermon and TAUoverMRNet
- PerfExplorer integration and expert-based analysis
  - OpenUH compiler optimizations
- Performance tools integration (NSF POINT project)
- Eclipse CDT and PTP integration
TAU Integration with IDEs

- High performance software development environments
  - Tools may be complicated to use
  - Interfaces and mechanisms differ between platforms / OS
- Integrated development environments
  - Consistent development environment
  - Numerous enhancements to development process
  - Standard in industrial software development
- Integrated performance analysis
  - Tools limited to single platform or programming language
  - Rarely compatible with 3rd party analysis tools
  - Little or no support for parallel projects
TAU and Eclipse
Choosing PAPI Counters with TAU in Eclipse

% /usr/local/packages/eclipse/eclipse
Support Acknowledgements

- Department of Energy (DOE)
  - Office of Science
    - MICS, Argonne National Lab
  - ASC/NNSA
    - University of Utah ASC/NNSA Level 1
    - ASC/NNSA, Lawrence Livermore National Lab
- Department of Defense (DoD)
  - HPC Modernization Office (HPCMO)
- NSF Software Development for Cyberinfrastructure (SDCI)
- Research Centre Juelich
- ANL, NASA Ames, LANL, SNL
- TU Dresden
- ParaTools, Inc.
- University of Oregon Performance Research Lab
For more information

• TAU Website:
  http://tau.uoregon.edu
  
  • Software
  
  • Release notes
  
  • Documentation

• To use Paraprof on your local system without installing TAU:
  http://tau.uoregon.edu/paraprof
  
• Paratools: http://www.paratools.com