Introduction to the TAU Performance System®

ESP Code for Q Workshop 2012 at Argonne National Laboratory, ALCF, Bldg. 240,# 1416, Apr 30-May 2, 2012, Argonne, IL

Wyatt Spear, U. Oregon
wspear@cs.uoregon.edu
http://tau.uoregon.edu
Acknowledgements: U. Oregon, ParaTools, Inc.

- Dr. Allen D. Malony, Professor, CIS Dept, and Director, NeuroInformatics Center, and CEO, ParaTools, Inc.
- Dr. Kevin Huck, Computer Scientist, ParaTools, Inc.
- Dr. John Linford, Computer Scientist, ParaTools, Inc.
- Dr. Chee Wai Lee, Senior software engineer, UO
- Wyatt Spear, Software engineer, UO, ParaTools, Inc.
- Suzanne Millstein, Software engineer, UO
- Scott Biersdorff, Software engineer, UO
- Nick Chaimov, Ph.D. student, UO
- William Voorhees, Ph.D. student, UO
- Dr. Robert Yelle, Research faculty, UO
What is TAU?

• TAU is a performance evaluation tool
• It supports parallel profiling and tracing toolkit
• Profiling shows you how much (total) time was spent in each routine
• Tracing shows you when the events take place in each process along a timeline
• Profiling and tracing can measure time as well as hardware performance counters from your CPU
• TAU can automatically instrument your source code (routines, loops, I/O, memory, phases, etc.)
• It supports C++, C, Chapel, UPC, Fortran, Python and Java
• TAU runs on all HPC platforms and it is free (BSD style license)
• TAU has instrumentation, measurement and analysis tools
• To use TAU, you need to set a couple of environment variables and substitute the name of the compiler with a TAU shell script
Performance Evaluation

• Profiling
  – Presents summary statistics of performance metrics
    – number of times a routine was invoked
    – exclusive, inclusive time/hpm counts spent executing it
    – number of instrumented child routines invoked, etc.
    – structure of invocations (calltrees/callgraphs)
    – memory, message communication sizes also tracked

• Tracing
  – Presents when and where events took place along a global timeline
    – timestamped log of events
    – message communication events (sends/receives) are tracked
      – shows when and where messages were sent
    – large volume of performance data generated leads to more perturbation in the program
TAU Performance Profiling

- Performance with respect to nested event regions
  - Program execution event stack (begin/end events)

- Profiling measures inclusive and exclusive data

- Exclusive measurements for region only performance

- Inclusive measurements includes nested “child” regions

- Support multiple profiling types
  - Flat, callpath, and phase profiling
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 (**Calldepth** 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 (e.g., per-iteration) phases
Routine Level Flat Profile

- **Goal:** What routines account for the most time? How much?
- **Flat profile with wallclock time:**
  - **Metric:** P_VIRTUAL_TIME
  - **Value:** Exclusive
  - **Units:** seconds

![Flat profile diagram]

**Top routines (wallclock time):**
- LEQ_IKSWEEP
- LEQ_BICGS0T
- LEQ_MATVECT
- SOLVE_SPECIES_EQ
- SOLVE_LIN_EQ
- PHYSICAL_PROP
- RRATES
- LEQ_MSOLVET
- INIT_AB_M
- CALC_MASS_FLUX_SPHR
- INIT_MU_S
- CALC_RESID_S
- SOLVE_ENERGY_EQ
- SOURCE_PHI
- DRAG_GS
Setting TAU_CALLPATH=1 Generates program callgraph
TAU Performance System Architecture

Instrumentation

Event selection

Event information

Measurement

Profiles

Traces

Profile Data Management (PerfDMF)

Profile Analysis (ParaProf)

Profile Data Mining (PerfExplorer)

Trace Data Management

Trace Visualizers

Trace Analyzers

TAU Portal

ParaTools

TAU Performance System Architecture

Instrumentation

Event selection

Event information

Measurement

Profiles

Traces

Profile Data Management (PerfDMF)

Profile Analysis (ParaProf)

Profile Data Mining (PerfExplorer)

Trace Data Management

Trace Visualizers

Trace Analyzers

TAU Portal

ParaTools
TAU Instrumentation Approach

• **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)
  – Selection of event statistics

• Support definition of “semantic” entities for mapping

• Support for event groups (aggregation, selection)

• Instrumentation optimization
  – Eliminate instrumentation in lightweight routines
Automatic Source-Level Instrumentation in TAU

TAU source analyzer

Parsed program

tau_instrumentor

Instrumented source

Instrumentation specification file
Using TAU: A brief Introduction

- TAU supports several measurement options (profiling, tracing, profiling with hardware counters, etc.)
- Each measurement configuration of TAU corresponds to a unique stub makefile that is generated when you configure it
- To instrument source code using PDT
  - Choose an appropriate TAU stub makefile in <arch>/lib:
    % soft add +tau-latest (on BG/Q)
    % export TAU_MAKEFILE=/soft/perftools/tau/tau_latest/bgq/lib/Makefile.tau-bgqtimers-mpi-pdt
    % export TAU_OPTIONS="-optVerbose …" (see tau_compiler.sh -help)
  And use tau_f90.sh, tau_cxx.sh or tau_cc.sh as Fortran, C++ or C compilers:
    % mpixlf90_r foo.f90
    changes to
    % tau_f90.sh foo.f90
    % qsub -A <...> ./a.out (to submit the job)
- Execute application and analyze performance data:
  % pprof (for text based profile display)
  % paraprof (for GUI)
  Install TAU or go to tau.uoregon.edu/paraprof to load paraprof on your local machine
Interval, Atomic and Context Events in TAU

**Interval Event**

<table>
<thead>
<tr>
<th>%Time</th>
<th>Exclusive msec</th>
<th>Inclusive total msec</th>
<th>#Call</th>
<th>#Subrs</th>
<th>Inclusive Name usec/call</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0</td>
<td>0.007</td>
<td>0.256</td>
<td>1</td>
<td>5</td>
<td>256 MAIN</td>
</tr>
<tr>
<td>97.3</td>
<td>0.132</td>
<td>0.249</td>
<td>5</td>
<td>5</td>
<td>50 FOO</td>
</tr>
<tr>
<td>40.6</td>
<td>0.104</td>
<td>0.104</td>
<td>5</td>
<td>0</td>
<td>21 BAR</td>
</tr>
<tr>
<td>36.3</td>
<td>0.013</td>
<td>0.093</td>
<td>3</td>
<td>3</td>
<td>31 G</td>
</tr>
</tbody>
</table>

**Context Event**

**Atomic Event**

% `pprof`
Parallel Profile Visualization: ParaProf

% soft add +tau-latest
% paraprof (Windows -> 3D Visualization)
ParaProf: 3D Communication Matrix

% qsub –env TAU_COMM_MATRIX=1 ...
% paraprof (Windows -> 3D Communication Matrix)
ParaProf: Scatter Plot
ParaProf: Topology View: MPI_Send on BG/P
ParaProf: Topology View: BG/Q Core Layout
Jumpshot [ANL]: Trace Visualization

% qsub -env TAU TRACE=1 ...
% tau_treemerge.pl
% tau2slog2 tau.trc tau.edf –o app.slog2

ParaTools % jumpshot app.slog2
Job and System Display for BGQ Resource Manager
Quick Reference on Cetus at /tmp/tau.txt

To use TAU on Cetus:
soft add +tau-latest
cd /veas-fs0/<your working directory>

cp /soft/perftools/tau/tau_latest/examples/matmult/matmult.f90 .
export TAU_MAKEFILE=$TAU/Makefile.tau-bqtimers-mpi-pdt
tau_f90.sh matmult.f90 -o matmult
qsub -A <YOUR_ACCOUNT> -n 2 --mode c8 -t 10 ./matmult
paraprof
Or use tracing:
qsub -A <YOUR_ACCOUNT> -n 2 --mode c8 -t 10 --env TAU_TRACE=1 ./matmult
tau_treemerge.pl
tau2slog2 tau.trc tau.edf -o app.slog2
jumpshot app.slog2
To profile an application on a large number of nodes:
qsub -A <YOUR_ACCOUNT> -n 64 --mode c16 -t 10 --env TAU_PROFILE_FORMAT="merged" ./matmult
See slides on Cetus: /soft/perftools/tau/ppt/*.ppt

Download TAU from http://tau.uoregon.edu for your desktop.
tau.uoregon.edu/tau.dmg or tau.uoregon.edu/tau.exe or tau.uoregon.edu/tau.tgz
Want more examples? Download workshop examples from:
/soft/perftools/tau/workshop.tar.gz
To load paraprof with Java webstart: tau.uoregon.edu/paraprof
TAU: A Quick Reference
TAU Performance System®

- Integrated toolkit for performance problem solving
  - Instrumentation, measurement, analysis, visualization
  - Portable performance profiling and tracing facility
  - Performance data management and data mining
- Based on direct performance measurement approach
- Open source
- Available on all HPC platforms
- http://tau.uoregon.edu

ParaTools
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 memory profiling (headroom, malloc/leaks)
  - Support for tracking I/O (wrappers, read/write/print calls)
  - Parallel profiles written at end of execution
  - Parallel profile snapshots can be taken during execution

- **Tracing**
  - All profile-level events + inter-process communication
  - Inclusion of multiple counter data in traced events
Performance Evaluation Alternatives

Each alternative has:
- one metric/counter
- multiple counters

Volume of performance data
Building Bridges to Other Tools
% cd /soft/perftools/tau/tau_latest/bgq/lib; ls Makefile.*

Makefile.tau-pdt
Makefile.tau-mpi-pdt
Makefile.tau-bgqttimers-mpi-pdt
Makefile.tau-bgqttimers-gnu-mpi-pdt
Makefile.tau-mpi-papi-pdt
Makefile.tau-papi-mpi-openmp-opari-pdt
Makefile.tau-pthread-pdt...

• For an MPI+F90 application, you may want to start with:

  Makefile.tau-mpi-pdt
  - Supports MPI instrumentation & PDT for automatic source instrumentation
  - % soft add +tau-latest
  - % export TAU_MAKEFILE=$TAU/Makefile.tau-bgqttimers-mpi-pdt
  - % make CC=tau_cc.sh CXX=tau_cxx.sh F90=tau_f90.sh
  - % qsub -n 2 -mode c16 -t 10 -A <account> ./a.out
  - % paraprof
## Runtime Environment Variables in TAU

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAU_TRACE</td>
<td>0</td>
<td>Setting to 1 turns on tracing</td>
</tr>
<tr>
<td>TAU_CALLPATH</td>
<td>0</td>
<td>Setting to 1 turns on callpath profiling</td>
</tr>
<tr>
<td>TAU_TRACK_MEMORY_LEAKS</td>
<td>0</td>
<td>Setting to 1 turns on leak detection</td>
</tr>
<tr>
<td>TAU_TRACK_HEAP or TAU_TRACK_HEADROOM</td>
<td>0</td>
<td>Setting to 1 turns on tracking heap memory/headroom at routine entry &amp; exit using context events (e.g., Heap at Entry: main=&gt;foo=&gt;bar)</td>
</tr>
<tr>
<td>TAU_CALLPATH_DEPTH</td>
<td>2</td>
<td>Specifies depth of callpath. Setting to 0 generates no callpath or routine information, setting to 1 generates flat profile and context events have just parent information (e.g., Heap Entry: foo)</td>
</tr>
<tr>
<td>TAU_TRACK_SIGNALS</td>
<td>0</td>
<td>Setting to 1 generate debugging callstack info when a program crashes</td>
</tr>
<tr>
<td>TAU_SAMPLING</td>
<td>0</td>
<td>Setting to 1 generates sample based profiles</td>
</tr>
<tr>
<td>TAU_COMM_MATRIX</td>
<td>0</td>
<td>Setting to 1 generates communication matrix display using context events</td>
</tr>
<tr>
<td>TAU_THROTTLE</td>
<td>1</td>
<td>Setting to 0 turns off throttling. Enabled by default to remove instrumentation in lightweight routines that are called frequently</td>
</tr>
<tr>
<td>TAU_THROTTLE_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>
</tbody>
</table>
Automatic Source-Level Instrumentation in TAU using Program Database Toolkit (PDT)
Automatic Instrumentation

• We now provide compiler wrapper scripts
  – Simply replace `CC` with `tau_cxx.sh`
  – Automatically instruments C++ and C source code, links with TAU MPI Wrapper libraries.

• Use `tau_cc.sh` and `tau_f90.sh` for C and Fortran

Before

```
CXX = mpixlcxx_r
F90 = mpixlf90_r
CFLAGS =
LIBS = -lm
OBJJS = f1.o f2.o f3.o ... fn.o

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

After

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

app: $(OBJJS)
    $(CXX) $(LDFLAGS) $(OBJJS) -o $@
    $(LIBS)
.cpp.o:
    $(CC) $(CFLAGS) -c $<
```
TAU_COMPILER Commandline Options

• See `<taudir>/<arch>/bin/tau_compiler.sh -help`

• Compilation:

  % mpixlf90_r -c foo.f90

  Changes to

  % gfparse foo.f90 $(OPT1)
  % tau_instrumentor foo.pdb foo.f90 -o foo.inst.f90 $(OPT2)
  % mpixlf90_r -c foo.inst.f90 $(OPT3)

• Linking:

  % mpixlf90_r foo.o bar.o -o app

  Changes to

  % mpixlf90_r foo.o bar.o -o app $(OPT4)

• Where options OPT[1-4] default values may be overridden by the user:

  F90 = tau_f90.sh
Compile-Time Environment Variables

- **Optional parameters for TAU_OPTIONS**: `[tau_compiler.sh –help]
  - `optVerbose` Turn on verbose debugging messages
  - `optCompInst` Use compiler based instrumentation
  - `optNoCompInst` Do not revert to compiler instrumentation if source instrumentation fails.
  - `optDetectMemoryLeaks` Turn on debugging memory allocations/de-allocations to track leaks
  - `optTrackIO` Wrap POSIX I/O call and calculates vol/bw of I/O operations (Requires TAU to be configured with –iowrapper)
  - `optKeepFiles` Does not remove intermediate .pdb and .inst.* files
  - `optPreProcess` Preprocess Fortran sources before instrumentation
  - `optTauSelectFile=""` Specify selective instrumentation file for tau_instrumentor
  - `optTauWrapFile=""` Specify link_options.tau generated by tau_gen_wrapper
  - `optLinking=""` Options passed to the linker. Typically
    ```
    $(TAU_MPI_FLIBS) $(TAU_LIBS) $(TAU_CXXLIBS)
    ```
  - `optCompile=""` Options passed to the compiler. Typically
    ```
    $(TAU_MPI_INCLUDE) $(TAU_INCLUDE) $(TAU_DEFS)
    ```
  - `optPdtF95Opts=""` Add options for Fortran parser in PDT (f95parse/gfparse)
  - `optPdtF95Reset=""` Reset options for Fortran parser in PDT (f95parse/gfparse)
  - `optPdtCxxOpts=""` Options for C parser in PDT (cpparse). Typically
    ```
    $(TAU_MPI_INCLUDE) $(TAU_INCLUDE) $(TAU_DEFS)
    ```
  - `optPdtCxxOpts=""` Options for C++ parser in PDT (cxxparse). Typically
    ```
    $(TAU_MPI_INCLUDE) $(TAU_INCLUDE) $(TAU_DEFS)
    ```
Compiling Fortran Codes with TAU

• If your Fortran code uses free format in .f files (fixed is default for .f), you may use:
  % 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 -optDetectMemoryLeaks"'

• To use an instrumentation specification file:
  % export TAU_OPTIONS='"-optTauSelectFile=mycmd.tau -optVerbose -optPreProcess"'
  % cat mycmd.tau
BEGIN_INSTRUMENT_SECTION
memory file="foo.f90" routine="#"
# instruments all allocate/deallocate statements in all routines in foo.f90
loops file="*" routine="#"
io file="abc.f90" routine="FOO"
END_INSTRUMENT_SECTION
Steps of Performance Evaluation

• Collect basic routine-level timing profile to determine where most time is being spent

• Collect routine-level hardware counter data to determine types of performance problems

• Collect callpath profiles to determine sequence of events causing performance problems

• Conduct finer-grained profiling and/or tracing to pinpoint performance bottlenecks
  – Loop-level profiling with hardware counters
  – Tracing of communication operations
Usage Scenarios: Debugging (beta)

• Goal: Where does my code crash? Compile with –g and TAU compiler scripts.

• qsub -t 20 -n 64 -q prod-devel -A TAU --env TAU_TRACK_SIGNALS=1 ./a.out
Usage Scenarios: Routine Level Profile

- Goal: What routines account for the most time? How much?
- Flat profile with wallclock time:

  Metric: PVIRTUAL_TIME
  Value: Exclusive
  Units: seconds

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

```
% export TAU_MAKEFILE=/soft/perftools/tau/tau_latest/
   /lib/Makefile.tau-bgqtimers-mpi-pdt
% export PATH=/soft/perftools/tau/tau_latest/bgq/bin:$PATH
OR
% soft add +tau_latest
% tau_f90.sh matmult.f90 -o matmult
(Or edit Makefile and change F90=tau_f90.sh)

% mpirun -np 4 ./matmult
% paraprof --pack app.ppk
   Move the app.ppk file to your desktop.

% paraprof app.ppk
```
Usage Scenarios: Loop Level Instrumentation

- 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

1729975.333

443194

81095

MAIN

49569

MPI_Bcast()

45669

Loop: MAIN [{matmult.f90} {86.9}-{106.14}]

12412

MPI_Send()

8959

Loop: INITIALIZE [{matmult.f90} {17.9}-{21.14}]

8953

Loop: INITIALIZE [{matmult.f90} {10.9}-{14.14}]

5609.2

MPI_Finalize()

2932.667

MULTIPLY_MATRICES

2577.667

Loop: MAIN [{matmult.f90} {117.9}-{128.14}]

2091.8

MPI_Barrier()

1875.667

Loop: MAIN [{matmult.f90} {112.9}-{115.14}]

1833

Loop: MAIN [{matmult.f90} {71.9}-{74.14}]

107

Loop: MAIN [{matmult.f90} {77.9}-{84.14}]

30

INITIALIZE

14.25

MPI_Comm_rank()

1

MPI_Comm_size()
Solution: Generating a loop level profile

```bash
% export TAU_MAKEFILE=/soft/perftools/tau/tau_latest/bgq
                   /lib/Makefile.tau-mpi-pdt
% export TAU_OPTIONS=’-optTauSelectFile=select.tau -optVerbose’
% cat select.tau
BEGIN_INSTRUMENT_SECTION
  loops routine=“#”
END_INSTRUMENT_SECTION

% export PATH=/soft/perftools/tau/tau_latest/ppc64/bin:$PATH
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% mpirun -np 4 ./a.out
% paraprof --pack app.ppk
  Move the app.ppk file to your desktop.

% paraprof app.ppk
```
Usage Scenarios: Compiler-based Instrumentation

- Goal: Easily generate routine level performance data using the compiler instead of PDT for parsing the source code
Use Compiler-Based Instrumentation

% export TAU_MAKEFILE=/soft/perftools/tau/tau_latest/bgq/lib/Makefile.tau-mpi
% export TAU_OPTIONS='--optCompInst --optVerbose'
% export PATH=/soft/perftools/tau/tau_latest/ppc64/bin:$PATH
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)

% qsub --mode c8 ./a.out
% paraprof --pack app.ppk
  Move the app.ppk file to your desktop.
% paraprof app.ppk
Usage Scenarios: Generating Callpath Profile

• Goal: Who calls my MPI_Barrier()? Where?
• Callpath profile for a given callpath depth:
Callpath Profile

• Generates program callgraph
Generate a Callpath Profile

% export TAU_MAKEFILE=/soft/perftools/tau/tau_latest/bgq/lib/Makefile.tau-mpi-pdt
% export PATH=/soft/perftools/tau/tau_latest/ppc64/bin:$PATH
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% export TAU_CALLPATH=1
% export TAU_CALLPATH_DEPTH=100

% qsub -mode c8
% paraprof --pack app.ppk
   Move the app.ppk file to your desktop.
% paraprof app.ppk
(Windows -> Thread -> Call Graph)

NOTE: In TAU v2.18.1+, you may choose to just set:
% export TAU_CALLPATH=1
instead of recompiling your code with the above stub makefile.
Any TAU instrumented executable can generate callpath profiles.
## Usage Scenario: Detect Memory Leaks

<table>
<thead>
<tr>
<th>Name</th>
<th>NumSamples</th>
<th>MaxValue</th>
<th>MinValue</th>
<th>MeanValue</th>
<th>Std. Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAIN ([matrix.f90] {141,7}–{146,22})</td>
<td>1</td>
<td>8,000,000</td>
<td>8,000,000</td>
<td>8,000,000</td>
<td>0</td>
</tr>
<tr>
<td>MEMORY LEAK! malloc size &lt;file=matrix.f90, variable=C, line=11&gt;</td>
<td>1</td>
<td>8,000,000</td>
<td>8,000,000</td>
<td>8,000,000</td>
<td>0</td>
</tr>
<tr>
<td>MATRICES::ALLOCATE_MATRICES ([matrix.f90] {10,7}–{13,38})</td>
<td>1</td>
<td>8,000,000</td>
<td>8,000,000</td>
<td>8,000,000</td>
<td>0</td>
</tr>
<tr>
<td>malloc size &lt;file=matrix.f90, variable=A, line=11&gt;</td>
<td>1</td>
<td>8,000,000</td>
<td>8,000,000</td>
<td>8,000,000</td>
<td>0</td>
</tr>
<tr>
<td>malloc size &lt;file=matrix.f90, variable=B, line=11&gt;</td>
<td>1</td>
<td>8,000,000</td>
<td>8,000,000</td>
<td>8,000,000</td>
<td>0</td>
</tr>
<tr>
<td>malloc size &lt;file=matrix.f90, variable=C, line=11&gt;</td>
<td>1</td>
<td>8,000,000</td>
<td>8,000,000</td>
<td>8,000,000</td>
<td>0</td>
</tr>
<tr>
<td>MATRICES::DEALLOCATE_MATRICES ([matrix.f90] {14,7}–{17,40})</td>
<td>1</td>
<td>8,000,000</td>
<td>8,000,000</td>
<td>8,000,000</td>
<td>0</td>
</tr>
<tr>
<td>free size &lt;file=matrix.f90, variable=A, line=15&gt;</td>
<td>1</td>
<td>8,000,000</td>
<td>8,000,000</td>
<td>8,000,000</td>
<td>0</td>
</tr>
<tr>
<td>free size &lt;file=matrix.f90, variable=B, line=15&gt;</td>
<td>1</td>
<td>8,000,000</td>
<td>8,000,000</td>
<td>8,000,000</td>
<td>0</td>
</tr>
</tbody>
</table>

**User Event Window: mem.ppk**

Name: MEMORY LEAK! malloc size <file=matrix.f90, variable=C, line=11> : MAIN ([matrix.f90] {141,7}–{146,22}) => MATRICES::ALLOCATE_MATRICES ([matrix.f90] {10,7}–{13,38})

Value Type: Max Value

<table>
<thead>
<tr>
<th>Mean</th>
<th>n,c,t 0,0</th>
<th>n,c,t 1,0</th>
<th>n,c,t 2,0</th>
<th>n,c,t 3,0</th>
<th>Std. Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>8000000</td>
<td>8000000</td>
<td>8000000</td>
<td>8000000</td>
<td>8000000</td>
<td>0</td>
</tr>
</tbody>
</table>

**Paratools**

47

**University of Oregon**
Detect Memory Leaks

% export TAU_MAKEFILE=/soft/perftools/tau/tau_latest/bgq
          /lib/Makefile.tau-mpi-pdt
% export TAU_OPTIONS='`-optDetectMemoryLeaks -optVerbose'`
% export PATH=/soft/perftools/tau/tau_latest/ppc64/bin:$PATH
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% export TAU_CALLPATH_DEPTH=100

% qsub --mode c8
% 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)
Usage Scenarios: Instrument a Python program

• Goal: Generate a flat profile for a Python program
Usage Scenarios: Instrument a Python program

Original code:

```
% cat foo.py
#!/usr/bin/env python
import numpy
ra=numpy.random
la=numpy.linalg

size=2000
a=ra.standard_normal((size,size))
b=ra.standard_normal((size,size))
c=la.linalg.dot(a,b)
print c
```

Create a wrapper:

```
% cat wrapper.py
#!/usr/bin/env python

# setenv PYTHONPATH $PET_HOME/pkgs/tau-2.17.3/ppc64/lib/bindings-gnu-python-pdt

import tau

def OurMain():
    import foo

tau.run('OurMain()')
```
Generate a Python Profile

% export TAU_MAKEFILE=/soft/perftools/tau/tau_latest/bgq
    /lib/Makefile.tau-python-pdt
% export PATH=/soft/perftools/tau/tau_latest/ppc64/bin:$PATH
% cat wrapper.py
    import tau
    def OurMain():
        import foo
        tau.run('OurMain()')

Uninstrumented:
% ./foo.py
Instrumented:
% export PYTHONPATH= <taudir>/bgq/<lib>/bindings-python-pdt
(same options string as TAU_MAKEFILE)
% export LD_LIBRARY_PATH=<taudir>/bgq/lib/bindings-python-pdt:
    $LD_LIBRARY_PATH
% ./wrapper.py

Wrapper invokes foo and generates performance data
% pprof/paraprof
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

% export TAU_MAKEFILE=/soft/perftools/tau/tau_latest/bgq
   /lib/Makefile.tau-python-mpi-pdt
% export PATH=/soft/perftools/tau/tau_latest/ppc64/bin:$PATH
% export 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 libs, pyMPI w/TAU)
% cat wrapper.py
   import tau
   def OurMain():
       import App
       tau.run('OurMain()')

Uninstrumented:
% qsub -n 4 <dir>/pyMPI-2.5b0/bin/pyMPI ./App.py

Instrumented:
% export PYTHONPATH=<taudir>/bgq/<lib>/bindings-python-mpi-pdt
(same options string as TAU_MAKEFILE)
% export LD_LIBRARY_PATH=<taudir>/bgq/lib/bindings-python-mpi-pdt:
   $LD_LIBRARY_PATH
% qsub -a -n 4 <dir>/pyMPI-2.5b0-TAU/bin/pyMPI ./wrapper.py
(Instrumented pyMPI with wrapper.py)
Usage Scenarios: Generating a Trace File

- **Goal**: Identify the temporal aspect of performance. What happens in my code at a given time? When?
- **Event trace visualized in Vampir/Jumpshot**
VNG Process Timeline with PAPI Counters
Vampir Counter Timeline Showing I/O BW
Generate a Trace File

% export TAU_MAKEFILE=/soft/perftools/tau/tau_latest/bgq
    /lib/Makefile.tau-mpi-pdt
% export PATH=/soft/perftools/tau/tau_latest/ppc64/bin:
$PATH
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% export TAU_TRACE=1
% mpirun -np 4 ./a.out
% 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).
Usage Scenarios: Evaluate Scalability

- **Goal:** How does my application scale? What bottlenecks occur at what core counts?
- Load profiles in PerfDMF database and examine with PerfExplorer

---

*Relative Speedup - S3D (Jaguar, ORNL): Harness Scaling Study: GET_TIME_OF_DAY*

*Total Runtime Breakdown for S3D (Jaguar, ORNL): Harness Scaling Study: GET_TIME_OF_DAY*
Usage Scenarios: Evaluate Scalability
Performance Regression Testing

FACETS Bassi Regression: 32 Procs (events above 2%)

- int main(int, char **)
- std::vector<double, std::allocator<double>> FcCoreCellUpdate...
- void FcTmCoreFluxCalc::computeFluxes()  MPI_Recv()
- double FcDataAssimilator::getValue(const std::string &, const... MPI_Init()
- FcHdf5Tmp<DATATYPE>::writeDataSet
- void FcDataAssimilatorUfiles::parseUfiles(const std::vector<... other
- void FcUpdaterComponent::dumpToFile(const std::string &... con...
Evaluate Scalability using PerfExplorer Charts

% export TAU_MAKEFILE=/soft/perftools/tau/tau_latest/bgq
    /lib/Makefile.tau-mpi-pdt
% export PATH=/soft/perftools/tau/tau_latest/ppc64/bin:$PATH
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% mpirun -np 1 .a.out
% paraprof --pack 1p.ppk
% mpirun -np 2 .a.out ...
% paraprof --pack 2p.ppk ... and so on.
On your client:
% perfdfm_configure --create-default
(Chooses 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
Then,
% perfexplorer
(Select experiment, Menu: Charts -> Speedup)
Communication Matrix Display

- Goal: What is the volume of inter-process communication? Along which calling path?
Evaluate Scalability using PerfExplorer Charts

% export TAU_MAKEFILE=/soft/perftools/tau/tau_latest/bgq
    /lib/Makefile.tau-mpi-pdt
% export PATH=/soft/perftools/tau/tau_latest/ppc64/bin:$PATH
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% export TAU_COMM_MATRIX=1

% qsub --mode c8 ./a.out (setting the environment variables)

% paraprof
(Windows -> Communication Matrix, Windows -> 3D Communication Matrix)
Communication Matrix Display

- Goal: What is the volume of inter-process communication? Along which calling path?
Interval Events, Atomic Events in TAU

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

**Atomic events**
(trigger with value)

% setenv TAU_CALLPATH_DEPTH 0
% setenv TAU_TRACK_HEAP 1
Context Events (default)

% setenv TAU_CALLPATH_DEPTH 2
% setenv TAU_TRACK_HEAP 1

ParaTools
Binary Rewriting: DyninstAPI [U.Wisc] and TAU

/home/livetau/tutorial
/home/livetau/tutorial% # Build an uninstrumented bt NAS Parallel Benchmark
/home/livetau/tutorial% make bt CLASS=bt NPROCS=4
/home/livetau/tutorial% cd bin
/home/livetau/tutorial/bin% # Run the instrumented code
/home/livetau/tutorial/bin% mpiexec -np 4 ./bt.W.4
/home/livetau/tutorial/bin% # Instrument the executable using TAU with DyninstAPI
/home/livetau/tutorial/bin% taurun ./bt.W.4 -o ./bt.i
/home/livetau/tutorial/bin% rm -rf profile.* MULTIX
/home/livetau/tutorial/bin% mpiexec -np 4 ./bt.i
/home/livetau/tutorial/bin% paraprof
/home/livetau/tutorial/bin% # Choose a different TAU configuration
/home/livetau/tutorial/bin% ls $TAU/libTAUsh
libTAUsh-depthlimit-mpi-pdt.so* libTAUsh-papi-pdt.so*
libTAUsh-mpi-pdt.so*
libTAUsh-papi-pthread-pdt.so*
libTAUsh-mpi-pdt-upc.so*
libTAUsh-param-mpi-pdt.so*
libTAUsh-pdt.so*
libTAUsh-papi-mpi-pdt.so*
libTAUsh-papi-pthread_pdt.so*
libTAUsh-papi-mpi-pdt-upc.so*
libTAUsh-param-papi-mpi-pdt.so*
libTAUsh-pdt-trace.so*
libTAUsh-papi-mpi-pdt-upc-udp.so*
libTAUsh-phase-papi-mpi-pdt.so*
libTAUsh-pthread-pdt.so*
libTAUsh-papi-mpi-pdt-vampirtrace-trace.so* libTAUsh-python-pdt.so*
libTAUsh-papi-mpi-python-pdt.so*
/home/livetau/tutorial/bin% ls $TAU/libTAUsh-
/home/livetau/tutorial/bin% taurun -XrunTAUsh-papi-mpi-pdt-vampirtrace-trace bt.W.4 -o bt.vpt
/home/livetau/tutorial/bin% setenv VT_METRICS_PAPI_FP_INS:PAPI_L1_DCM
/home/livetau/tutorial/bin% mpiexec -np 4 ./bt.vpt
/home/livetau/tutorial/bin% vampir bt.vpt.otf &
/home/livetau/tutorial/bin%
Using PAPI and TAU
Hardware Counters

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

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

Hardware counter information can be obtained with:

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

• Open Source software from U. Tennessee, Knoxville
• http://icl.cs.utk.edu/papi
• Middleware to provide a consistent programming interface for the performance counter hardware found in most major micro-processors.
• Countable events are defined in two ways:
  – Platform-neutral preset events
  – Platform-dependent native events
• Presets can be derived from multiple native events
• All events are referenced by name and collected in EventSets
$ utils/papi_avail -h
Usage: utils/papi_avail [options]

Options:

General command options:
  -a, --avail     Display only available preset events
  -d, --detail    Display detailed information about all preset events
  -e EVENTNAME    Display detail information about specified preset or native event
  -h, --help      Print this help message

Event filtering options:
  --br           Display branch related PAPI preset events
  --cache        Display cache related PAPI preset events
  --cnd          Display conditional PAPI preset events
  --fp           Display Floating Point related PAPI preset events
  --ins          Display instruction related PAPI preset events
  --idl          Display Stalled or Idle PAPI preset events
  --l1           Display level 1 cache related PAPI preset events
  --l2           Display level 2 cache related PAPI preset events
  --l3           Display level 3 cache related PAPI preset events
  --mem          Display memory related PAPI preset events
  --msc          Display miscellaneous PAPI preset events
  --tlb          Display Translation Lookaside Buffer PAPI preset events

This program provides information about PAPI preset and native events.
PAPI preset event filters can be combined in a logical OR.
PAPI Utilities: papi_avail

$ utils/papi_avail
Available events and hardware information.
--------------------------------------------------------------------------------
PAPI Version : 4.0.0.0
Vendor string and code : GenuineIntel (1)
Model string and code : Intel Core i7 (21)
CPU Revision : 5.000000
CPUID Info : Family: 6 Model: 26 Stepping: 5
CPU Megahertz : 2926.000000
CPU Clock Megahertz : 2926
Hdw Threads per core : 1
Cores per Socket : 4
NUMA Nodes : 2
CPU's per Node : 4
Total CPU's : 8
Number Hardware Counters : 7
Max Multiplex Counters : 32
--------------------------------------------------------------------------------
The following correspond to fields in the PAPI_event_info_t structure.

[MORE... ]
The following correspond to fields in the PAPI_event_info_t structure.

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

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

avail.c  PASSED
PAPI Utilities: \textit{papi\_avail}

\begin{verbatim}
$ utils/papi_avail -e PAPI_FP_OPS
[...]

The following correspond to fields in the PAPI\_event\_info\_t structure.

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

Native Code[1]: 0x4000081b |FP\_COMP\_OPS\_EXE:SSE\_DOUBLE\_PRECISION|
Number of Register Values: 2
Register[ 0]: 0x0000000f |Event Selector|
Register[ 1]: 0x00008010 |Event Code|
Native Event Description: |Floating point computational micro-ops, masks:SSE* FP double precision Uops|
\end{verbatim}
PAPI Utilities: `papi_native_avail`

```
UNIX> utils/papi_native_avail
Available native events and hardware information.
--------------------------------------------------------------------------------

[...]

<table>
<thead>
<tr>
<th>Event Code</th>
<th>Symbol</th>
<th>Long Description</th>
</tr>
</thead>
</table>

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

<p>| 0x40000011        | BR_INST_RETIRED | Retired branch instructions                           |
| 40000411          | :ALL_BRANCHES  | Retired branch instructions (Precise Event)          |
| 40000811          | :CONDITIONAL   | Retired conditional branch instructions (Precise Event) |
|                  |               |                                                       |</p>
<table>
<thead>
<tr>
<th>40001011</th>
<th>:NEAR_CALL</th>
<th>Retired near call instructions (Precise Event)</th>
</tr>
</thead>
</table>

[...]
```
PAPI Utilities: \texttt{papi\_native\_avail}

UNIX> \texttt{utils/papi\_native\_avail -e DATA\_CACHE\_REFILLS}
Available native events and hardware information.
--------------------------------------------------------------------------------
 [...]
--------------------------------------------------------------------------------
The following correspond to fields in the PAPI\_event\_info\_t structure.

Event name: \texttt{DATA\_CACHE\_REFILLS}
Event Code: \texttt{0x4000000b}
Number of Register Values: 2
Description: |Data Cache Refills from L2 or System|
Register[ 0]: \texttt{0x0000000f} |Event Selector|
Register[ 1]: \texttt{0x00000042} |Event Code|

Unit Masks:

Mask Info: |:SYSTEM|Refill from System|
Register[ 0]: \texttt{0x0000000f} |Event Selector|
Register[ 1]: \texttt{0x000000142} |Event Code|

Mask Info: |:L2\_SHARED|Shared-state line from L2|
Register[ 0]: \texttt{0x0000000f} |Event Selector|
Register[ 1]: \texttt{0x000000242} |Event Code|

Mask Info: |:L2\_EXCLUSIVE|Exclusive-state line from L2|
Register[ 0]: \texttt{0x0000000f} |Event Selector|
Register[ 1]: \texttt{0x000000442} |Event Code|
$ utils/papi_event_chooser PRESET PAPI_FP_OPS
Event Chooser: Available events which can be added with given events.

<table>
<thead>
<tr>
<th>Name</th>
<th>Code</th>
<th>Deriv</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAPI_L1_DCM</td>
<td>0x80000000</td>
<td>No</td>
<td>Level 1 data cache misses</td>
</tr>
<tr>
<td>PAPI_L1_ICM</td>
<td>0x80000001</td>
<td>No</td>
<td>Level 1 instruction cache misses</td>
</tr>
<tr>
<td>PAPI_L2_ICM</td>
<td>0x80000003</td>
<td>No</td>
<td>Level 2 instruction cache misses</td>
</tr>
<tr>
<td>PAPI_L1_DCA</td>
<td>0x80000040</td>
<td>No</td>
<td>Level 1 data cache accesses</td>
</tr>
<tr>
<td>PAPI_L2_DCR</td>
<td>0x80000044</td>
<td>No</td>
<td>Level 2 data cache reads</td>
</tr>
<tr>
<td>PAPI_L2_DCW</td>
<td>0x80000047</td>
<td>No</td>
<td>Level 2 data cache writes</td>
</tr>
<tr>
<td>PAPI_L1_ICA</td>
<td>0x8000004c</td>
<td>No</td>
<td>Level 1 instruction cache accesses</td>
</tr>
<tr>
<td>PAPI_L2_ICA</td>
<td>0x8000004d</td>
<td>No</td>
<td>Level 2 instruction cache accesses</td>
</tr>
<tr>
<td>PAPI_L2_TCA</td>
<td>0x80000059</td>
<td>No</td>
<td>Level 2 total cache accesses</td>
</tr>
<tr>
<td>PAPI_L2_TCW</td>
<td>0x8000005f</td>
<td>No</td>
<td>Level 2 total cache writes</td>
</tr>
<tr>
<td>PAPI_FML_INS</td>
<td>0x80000061</td>
<td>No</td>
<td>Floating point multiply instructions</td>
</tr>
<tr>
<td>PAPI_FDV_INS</td>
<td>0x80000063</td>
<td>No</td>
<td>Floating point divide instructions</td>
</tr>
</tbody>
</table>

Total events reported: 34

event_chooser.c PASSED
PAPI Utilities: papi_event_chooser

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

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

Total events reported: 2

event_chooser.c                          PASSED
PAPI Utilities: `papi_event_chooser`

```bash
$ utils/papi_event_chooser NATIVE RESOURCESTALLS:LD_ST X87_OPS_RETIRED INSTRUCTIONS_RETIRED

[...]
```

---

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

---

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

---

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

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

---

Total events reported: 3
event_chooser.c  PASSED
Usage Scenarios: Calculate mflops in Loops

- Goal: What MFlops am I getting in all loops?
- Flat profile with PAPI_FP_INS/OPS and time with loop instrumentation:

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

![Profile diagram showing loop metrics]

```
Loop: MULTIPLY_MATRICES ([matmult.f90] {31.9}-{36.14])
Loop: INITIALIZE ([matmult.f90] {10.9}-{14.14})
Loop: INITIALIZE ([matmult.f90] {17.9}-{21.14})
Loop: MAIN ([matmult.f90] {71.9}-{74.14})
Loop: MAIN ([matmult.f90] {112.9}-{115.14})
Loop: MAIN ([matmult.f90] {117.9}-{128.14})
```

```
770.699
223.39
223.24
171.855
170.862
122.96
37.549
21.367
13.796
8.935
1.131
0.794
0.647
0.355
0.171
0.115
0.023
```
ParaProf: Mflops Sorted by Exclusive Time

low mflops?
Generate a PAPI profile with 2 or more counters

% export TAU_MAKEFILE=$TAU_ROOT/lib/Makefile.tau-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.
Estimation of tool intrusiveness
PAPI Utilities: *papi_cost*

```
$ utils/papi_cost -h
This is the PAPI cost program.
It computes min / max / mean / std. deviation for PAPI start/stop pairs;
for PAPI reads, and for PAPI_accums.

Usage:

    cost [options] [parameters]
    cost TESTS QUIET

Options:

- -b BINS     set the number of bins for the graphical
distribution of costs. Default: 100
- -d           show a graphical distribution of costs
- -h           print this help message
- -s           show number of iterations above the first
  10 std deviations
- -t THRESHOLD set the threshold for the number of
  iterations. Default: 100,000
```
$ utils/papi_cost
Cost of execution for PAPI start/stop and PAPI read.
This test takes a while. Please be patient...
Performing start/stop test...

Total cost for PAPI_start/stop(2 counters) over 1000000 iterations
min cycles : 63
max cycles : 17991
mean cycles : 69.000000
std deviation: 34.035263
   Performing start/stop test...

Performing read test...

Total cost for PAPI_read(2 counters) over 1000000 iterations
min cycles : 288
max cycles : 102429
mean cycles : 301.000000
std deviation: 144.694053
   cost.c                                   PASSED
Cost distribution profile

63:****************************** 999969 counts ******************************
153:
243:
[...]
1863:
1953:**********************
2043:
2133:**********************
2223:
2313:
2403:**********************
2493:**********************
2583:**********************
2673:**********************
2763:**********************
2853:**********************
2943:
3033:**********************
3123:**********************
3213:**********************
3303:
3393:
3483:
3573:
3663:**********************
Profile Measurement – Three Flavors

• Flat profiles
  – Time (or counts) spent in each routine (nodes in callgraph).
  – Exclusive/inclusive time, no. of calls, child calls
  – E.g.: MPI_Send, foo, …

• Callpath Profiles
  – Flat profiles, **plus**
  – Sequence of actions that led to poor performance
  – Time spent along a calling path (edges in callgraph)
  – E.g., “main=> f1 => f2 => MPI_Send” shows the time spent in MPI_Send when called by f2, when f2 is called by f1, when it is called by main. Depth of this callpath = 4 (TAU_CALLPATH_DEPTH environment variable)

• Phase based profiles
  – Flat profiles, **plus**
  – Flat profiles under a phase (nested phases are allowed)
  – Default “main” phase has all phases and routines invoked outside phases
  – Supports static or dynamic (per-iteration) phases
  – E.g., “IO => MPI_Send” is time spent in MPI_Send in IO phase
Phase Profiling (NAS BT, Flat Profile)

How is MPI_Wait() distributed relative to solver direction?

Application routine names reflect phase semantics.
NAS BT – Phase Profile (Main and X, Y, Z)

Main phase shows nested phases and immediate events
TAU Timers and Phases

- **Static timer**
  - Shows time spent in all invocations of a routine (foo)
  - E.g., “foo()” 100 secs, 100 calls

- **Dynamic timer**
  - Shows time spent in each invocation of a routine
  - E.g., “foo() 3” 4.5 secs, “foo 10” 2 secs (invocations 3 and 10 respectively)

- **Static phase**
  - Shows time spent in all routines called (directly/indirectly) by a given routine (foo)
  - E.g., “foo() => MPI_Send()” 100 secs, 10 calls shows that a total of 100 secs were spent in MPI_Send() when it was called by foo.

- **Dynamic phase**
  - Shows time spent in all routines called by a given invocation of a routine.
  - E.g., “foo() 4 => MPI_Send()” 12 secs, shows that 12 secs were spent in MPI_Send when it was called by the 4th invocation of foo.
Performance Dynamics: Phase-Based Profiling

- Profile phases capture performance with respect to application-defined ‘phases’ of execution
  - Separate full profile produce for each phase
- GTC particle-in-cell simulation of fusion turbulence
- Phases assigned to iterations
- Data change affects cache
Memory and I/O evaluation
• TAU provides a wealth of options to measure the performance of an application
• Need to simplify TAU usage to easily evaluate performance properties, including I/O, memory, and communication
• Designed a new tool (*tau_exec*) that leverages runtime instrumentation by pre-loading measurement libraries
• Works on dynamic executables (default under Linux, not on IBM Blue Gene where we must compile with -dynamic)
• Substitutes I/O, MPI, and memory allocation/deallocation routines with instrumented calls
  – Interval events (e.g., time spent in write())
  – Atomic events (e.g., how much memory was allocated)
• Measure I/O and memory usage
TAU Execution Command (tau_exec)

- Uninstrumented execution (compiled with –Wl,–Bdynamic on BG/P)
  - % qsub –n 256 –t 10 ./a.out

- Track MPI performance (-T <options>)
  - % tau_exec –qsub -T bgqtimers,mpi,pdt -- qsub –n 256 –t 10 ./a.out

- Track I/O and MPI performance (MPI by default, use –T serial for serial)
  - % tau_exec –io qsub -T bgqtimers,mpi,pdt -- qsub –n 256 –t 10 ./a.out

- Track memory operations
  - % tau_exec –memory –env TAU_TRACK_MEMORY_LEAKS=1 qsub -T bgqtimers,mpi,pdt -- qsub –n 256 –t 10 ./a.out
Library wrapping: tau_gen_wrapper

• How to instrument an external library without source?
  – Source may not be available
  – Library may be too cumbersome to build (with instrumentation)

• Build a library wrapper tools
  – Used PDT to parse header files
  – Generate new header files with instrumentation files
  – Three methods to instrument: runtime preloading, linking, redirecting headers to re-define functions

• Application is instrumented

• Add the –optTauWrapFile=<wrapperdir>/link_options.tau file to TAU_OPTIONS env var while compiling with tau_cc.sh, etc.

• Wrapped library
  – Redirects references at routine callsite to a wrapper call
  – Wrapper internally calls the original
  – Wrapper has TAU measurement code
HDF5 Library Wrapping

$ tau_gen_wrapper hdf5.h /usr/lib/libhdf5.a -f select.tau

Usage: tau_gen_wrapper <header> <library> [-r|-d|-w (default)] [-g groupname] [-i headerfile] [-c|-c++|-fortran] [-f <instr_req_file> ]
- instruments using runtime preloading (-r), or -Wl,-wrap linker (-w), redirection of header file to redefine the wrapped routine (-d)
- instrumentation specification file (select.tau)
- -g group may be specified (hdf5)
- tau_exec loads libhdf5_wrap.so shared library using -loadlib=<libwrap_pkg.so>
- creates the wrappers/ directory with linkoptions.tau passed to the TAU_OPTIONS environment variable using -optTauWrapFile=<file>

NODE 0;CONTEXT 0;THREAD 0:

----------------------------------------
%Time   Exclusive  Inclusive  #Call   #Subrs  Inclusive Name  
      msec    total msec usec/call
----------------------------------------
100.0   0.057       1           13       1236 .TAU Application
70.8    0.875       0.875       1         0  hid_t H5Fcreate()
9.7     0.12        0.12        1         0  herr_t H5Fclose()
6.0     0.074       0.074       1         0  herr_t H5Dcreate()
3.1     0.038       0.038       1         0  herr_t H5Dwrite()
2.6     0.032       0.032       1         0  herr_t H5Dclose()
2.1     0.026       0.026       1         0  herr_t H5check_version()
0.6     0.008       0.008       1         0  herr_t H5Screate_simple()
0.2     0.002       0.002       1         0  herr_t H5Tset_order()
0.2     0.002       0.002       1         0  herr_t H5Tcopy()
0.1     0.001       0.001       1         0  herr_t H5Sclose()
A New Approach: tau_exec

- Runtime instrumentation by pre-loading the measurement library
- Works on dynamic executables (default under Linux)
- Substitutes I/O, MPI and memory allocation/deallocation routines with instrumented calls
- Track interval events (e.g., time spent in write()) as well as atomic events (e.g., how much memory was allocated) in wrappers
- Accurately measure I/O and memory usage
Tracking I/O in static binaries (IBM Blue Gene)

• The linker can substitute TAU’s I/O wrapper and intercept POSIX I/O Calls
• We can track parameters that flow through the I/O calls
• Configure TAU with –iowrappers
• Use –optTrackIO in TAU_OPTIONS
Tracking I/O in static binaries

% export TAU_MAKEFILE=$TAU_ROOT/lib/Makefile.tau-bggtimers-mpi-pdt
% export PATH=$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)
Issues

• Heap memory usage reported by the mallinfo() call is not 64-bit clean.
  – 32 bit counters in Linux roll over when > 4GB memory is used
  – We keep track of heap memory usage in 64 bit counters inside TAU

• Compensation of perturbation introduced by tool
  – Only show what application uses
  – Create guards for TAU calls to not track I/O and memory allocations/de-allocations performed inside TAU

• Provide broad POSIX I/O and memory coverage
## I/O Calls Supported

<table>
<thead>
<tr>
<th>Unbuffered I/O</th>
<th>Buffered I/O</th>
<th>Communication</th>
<th>Control</th>
<th>Asynchronous I/O</th>
</tr>
</thead>
<tbody>
<tr>
<td>open</td>
<td>fopen</td>
<td>socket</td>
<td>fcntl</td>
<td>aio_read</td>
</tr>
<tr>
<td>open64</td>
<td>fopen64</td>
<td>pipe</td>
<td>rewind</td>
<td>aio_write</td>
</tr>
<tr>
<td>close</td>
<td>fdopen</td>
<td>socketpair</td>
<td>lseek</td>
<td>aio_suspend</td>
</tr>
<tr>
<td>read</td>
<td>freopen</td>
<td>bind</td>
<td>lseek64</td>
<td>aio_cancel</td>
</tr>
<tr>
<td>write</td>
<td>fclose</td>
<td>accept</td>
<td>fseek</td>
<td>aio_return</td>
</tr>
<tr>
<td>readv</td>
<td>fprintf</td>
<td>connect</td>
<td>dup</td>
<td>lio_listio</td>
</tr>
<tr>
<td>writev</td>
<td>fscanf</td>
<td>recv</td>
<td>dup2</td>
<td></td>
</tr>
<tr>
<td>creat</td>
<td>fwrite</td>
<td>send</td>
<td>mkstep</td>
<td></td>
</tr>
<tr>
<td>creat64</td>
<td>fread</td>
<td>sendto</td>
<td>tmpfile</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>recvfrom</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>pclose</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Tracking I/O in Each File

![Screenshot of TAU Profiler](image.png)

<table>
<thead>
<tr>
<th>Bytes Read</th>
<th>Total</th>
<th>NumSamples</th>
<th>MinVal</th>
<th>MaxVal</th>
<th>MeanVal</th>
<th>Std Dev</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="">file:/opt/openmpi/intel/1.4/etc/openmpi-mca-params.conf</a></td>
<td>20,024</td>
<td>32</td>
<td>8,192</td>
<td>4</td>
<td>255.75</td>
<td>2,914.699</td>
</tr>
<tr>
<td><a href="">file:/opt/openmpi/intel/1.4/share/openmpi/help-mpi-btl-openib.txt</a></td>
<td>2,812</td>
<td>1</td>
<td>2,812</td>
<td>2,812</td>
<td>2,812</td>
<td>0</td>
</tr>
<tr>
<td><a href="">file:/opt/openmpi/intel/1.4/share/openmpi/mca-btl-openib-device-params.ini</a></td>
<td>8,192</td>
<td>2</td>
<td>8,192</td>
<td>8,192</td>
<td>8,192</td>
<td>0</td>
</tr>
<tr>
<td><a href="">file:/sys/class/infiniband/mthca0/node_type</a></td>
<td>8,727</td>
<td>2</td>
<td>8,192</td>
<td>8,727</td>
<td>8,192</td>
<td>0</td>
</tr>
<tr>
<td><a href="">file:/sys/class/infiniband/mthca0/ports/1/gids/0</a></td>
<td>8</td>
<td>1</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td><a href="">file:/sys/class/infiniband/verbs/abi_version</a></td>
<td>41</td>
<td>1</td>
<td>41</td>
<td>41</td>
<td>41</td>
<td>0</td>
</tr>
<tr>
<td><a href="">file:/sys/class/infiniband/verbs/overbs0/abi_version</a></td>
<td>8</td>
<td>1</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td><a href="">file:/sys/class/infiniband/verbs/overbs0/device/device</a></td>
<td>24</td>
<td>3</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td><a href="">file:/sys/class/infiniband/verbs/overbs0/vendor</a></td>
<td>24</td>
<td>3</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td><a href="">file:/sys/class/infiniband/verbs/overbs0/ibdev</a></td>
<td>64</td>
<td>1</td>
<td>64</td>
<td>64</td>
<td>64</td>
<td>0</td>
</tr>
<tr>
<td><a href="">file:/sys/devices/system/cpu/cpu0/topology/core_id</a></td>
<td>7</td>
<td>1</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td><a href="">file:/sys/devices/system/cpu/cpu0/topology/physical_package_id</a></td>
<td>7</td>
<td>1</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td><a href="">file:/sys/devices/system/cpu/cpu1/topology/core_id</a></td>
<td>7</td>
<td>1</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td><a href="">file:/sys/devices/system/cpu/cpu1/topology/physical_package_id</a></td>
<td>7</td>
<td>1</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td><a href="">file:/sys/devices/system/cpu/cpu2/topology/core_id</a></td>
<td>7</td>
<td>1</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td><a href="">file:/sys/devices/system/cpu/cpu2/topology/physical_package_id</a></td>
<td>7</td>
<td>1</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td><a href="">file:/sys/devices/system/cpu/cpu3/topology/core_id</a></td>
<td>7</td>
<td>1</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td><a href="">file:/sys/devices/system/cpu/cpu3/topology/physical_package_id</a></td>
<td>7</td>
<td>1</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td><a href="">file:/sys/devices/system/cpu/cpu4/topology/core_id</a></td>
<td>7</td>
<td>1</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td><a href="">file:/sys/devices/system/cpu/cpu4/topology/physical_package_id</a></td>
<td>7</td>
<td>1</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td><a href="">file:/sys/devices/system/cpu/cpu5/topology/core_id</a></td>
<td>7</td>
<td>1</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td><a href="">file:/sys/devices/system/cpu/cpu5/topology/physical_package_id</a></td>
<td>7</td>
<td>1</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td><a href="">file:/sys/devices/system/cpu/cpu6/topology/core_id</a></td>
<td>7</td>
<td>1</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td><a href="">file:/sys/devices/system/cpu/cpu6/topology/physical_package_id</a></td>
<td>7</td>
<td>1</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td><a href="">file:/sys/devices/system/cpu/cpu7/topology/core_id</a></td>
<td>7</td>
<td>1</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td><a href="">file:/sys/devices/system/cpu/cpu7/topology/physical_package_id</a></td>
<td>7</td>
<td>1</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td><a href="">file:/sys/devices/system/cpu/cpu8/topology/core_id</a></td>
<td>7</td>
<td>1</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td><a href="">file:/sys/devices/system/cpu/cpu8/topology/physical_package_id</a></td>
<td>7</td>
<td>1</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td><a href="">file:/sys/devices/system/cpu/cpu9/topology/core_id</a></td>
<td>7</td>
<td>1</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td><a href="">file:/sys/devices/system/cpu/cpu9/topology/physical_package_id</a></td>
<td>7</td>
<td>1</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td><a href="">file:/sys/devices/system/cpu/cpu10/topology/core_id</a></td>
<td>7</td>
<td>1</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td><a href="">file:/sys/devices/system/cpu/cpu10/topology/physical_package_id</a></td>
<td>7</td>
<td>1</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td><a href="">file:/pipe/</a></td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

**READ Bandwidth (MB/s):**

- <file:/opt/openmpi/intel/1.4/etc/openmpi-mca-params.conf> 2,932.118 32 1,170,286 0.001 91.629 265.282
- <file:/opt/openmpi/intel/1.4/share/openmpi/help-mpi-btl-openib.txt> 512,444 1 512,444 512,444 512,444 0
- <file:/opt/openmpi/intel/1.4/share/openmpi/mca-btl-openib-device-params.ini> 1,170,286 2 1,170,286 1,170,286 1,170,286 0
- <file:/sys/class/infiniband/mthca0/node_type> 1,291,5 2 1,024 267,5 645,75 378,285
- <file:/sys/class/infiniband/mthca0/ports/1/gids/0> 4 1 4 4 4 0
- <file:/sys/class/infiniband/verbs/abi_version> 4 1 4 4 4 0
- <file:/sys/class/infiniband/verbs/overbs0/abi_version> 4 1 4 4 4 0
- ...
Time Spent in POSIX I/O write()
### Volume of I/O by File, Memory

<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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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>malloc size</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>free size</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>OurMain [twrapper.py[3]]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>read()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>malloc size</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>219.429</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 [samarcinterface.py[145]]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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 (MB/s) &lt;file=&quot;samarc/restore.00002/nodes.00004/proc.00001&quot;&gt;</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 (MB/s) &lt;file=&quot;samarc/restore.00001/nodes.00004/proc.00001&quot;&gt;</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 (MB/s)</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 &lt;file=&quot;samarc/restore.00002/nodes.00004/proc.00001&quot;&gt;</td>
<td>2,097,552</td>
<td>17,927.795</td>
<td>117</td>
<td>1,048,576</td>
<td>133,362.946</td>
<td></td>
</tr>
<tr>
<td>Bytes Written &lt;file=&quot;samarc/restore.00001/nodes.00004/proc.00001&quot;&gt;</td>
<td>2,097,552</td>
<td>17,927.795</td>
<td>117</td>
<td>1,048,576</td>
<td>133,362.946</td>
<td></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>
Bytes Written
Memory Leaks in MPI

<table>
<thead>
<tr>
<th>Name</th>
<th>Total</th>
<th>Mean Value</th>
<th>NumSamples</th>
<th>MaxValue</th>
<th>MinValue</th>
<th>Std. Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>malloc size</td>
<td>5,013,902</td>
<td>65,972.395</td>
<td>76</td>
<td>5,000,000</td>
<td>2</td>
<td>569,732.815</td>
</tr>
<tr>
<td>MEMORY LEAK!</td>
<td>5,000,264</td>
<td>500,026.4</td>
<td>10</td>
<td>5,000,000</td>
<td>3</td>
<td>1,499,991.2</td>
</tr>
<tr>
<td>write()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WRITE Bandwidth (MB/s)</td>
<td>0.635</td>
<td>102</td>
<td>12</td>
<td>0</td>
<td>1.472</td>
<td></td>
</tr>
<tr>
<td>Bytes Written</td>
<td>24</td>
<td>24</td>
<td>1</td>
<td>24</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Bytes Written</td>
<td>1,456</td>
<td>14.275</td>
<td>102</td>
<td>28</td>
<td>0.089</td>
<td>5.149</td>
</tr>
<tr>
<td>WRITE Bandwidth (MB/s)</td>
<td>0.528</td>
<td>97</td>
<td>12</td>
<td>0.089</td>
<td>1.32</td>
<td></td>
</tr>
<tr>
<td>Bytes Written</td>
<td>54</td>
<td>16</td>
<td>4</td>
<td>28</td>
<td>0</td>
<td>1.22</td>
</tr>
<tr>
<td>WRITE Bandwidth (MB/s)</td>
<td>1.714</td>
<td>1</td>
<td>1.714</td>
<td>1.714</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Bytes Written</td>
<td>1,368</td>
<td>14.103</td>
<td>97</td>
<td>24</td>
<td>12</td>
<td>4.562</td>
</tr>
<tr>
<td>WRITE Bandwidth (MB/s)</td>
<td>2.967</td>
<td>4</td>
<td>5.6</td>
<td>0</td>
<td>2.644</td>
<td></td>
</tr>
<tr>
<td>writev()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WRITE Bandwidth (MB/s)</td>
<td>4.108</td>
<td>2</td>
<td>7.667</td>
<td>0.549</td>
<td>3.559</td>
<td></td>
</tr>
<tr>
<td>Bytes Written</td>
<td>297</td>
<td>148.5</td>
<td>2</td>
<td>230</td>
<td>67</td>
<td>81.5</td>
</tr>
<tr>
<td>WRITE Bandwidth (MB/s)</td>
<td>4.108</td>
<td>2</td>
<td>7.667</td>
<td>0.549</td>
<td>3.559</td>
<td></td>
</tr>
<tr>
<td>Bytes Written</td>
<td>297</td>
<td>148.5</td>
<td>2</td>
<td>230</td>
<td>67</td>
<td>81.5</td>
</tr>
<tr>
<td>readv()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bytes Read</td>
<td>112</td>
<td>28</td>
<td>4</td>
<td>36</td>
<td>20</td>
<td>8</td>
</tr>
<tr>
<td>READ Bandwidth (MB/s)</td>
<td>25.5</td>
<td>4</td>
<td>36</td>
<td>10</td>
<td>11.079</td>
<td></td>
</tr>
<tr>
<td>Bytes Read</td>
<td>112</td>
<td>28</td>
<td>4</td>
<td>36</td>
<td>20</td>
<td>8</td>
</tr>
<tr>
<td>READ Bandwidth (MB/s)</td>
<td>25.5</td>
<td>4</td>
<td>36</td>
<td>10</td>
<td>11.079</td>
<td></td>
</tr>
<tr>
<td>MPL_Comm_free()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>free size</td>
<td>10,952</td>
<td>195.571</td>
<td>56</td>
<td>1,024</td>
<td>48</td>
<td>255.353</td>
</tr>
</tbody>
</table>
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

- Provide an interface for configuring TAU’s automatic instrumentation within Eclipse’s build system
- Manage runtime configuration settings and environment variables for execution of TAU instrumented programs

C/C++/Fortran Project in Eclipse → Add or modify an Eclipse build configuration w/ TAU → Temporary copy of instrumented code

TAU instrumented libraries → Compilation/linking with TAU libraries → Performance data

Program execution → Program output
TAU and Eclipse

PerfDMF
Choosing PAPI Counters with TAU in Eclipse

% /soft/perftools/tau/eclipse/eclipse
Configuring Job Submission for Remote Execution on BGQ

![Image of Paratools configuration interface for BGQ](image_url)
Job and System Display for BGQ Resource Manager
Labs!
Lab Instructions

Get workshop.tar.gz using:

```
% wget http://www.paratools.com/anl12/workshop.tar.gz
```

Or

```
% cp /soft/perftools/tau/workshop.tar.gz .
% tar zxf workshop.tar.gz
% soft add +tau-latest
```
Lab Instructions

To profile a code using TAU:

1. Change the compiler name to tau_cxx.sh, tau_f90.sh, tau_cc.sh:
   F90 = tau_f90.sh

2. Choose TAU stub makefile
   % export TAU_MAKEFILE=/soft/perftools/tau/tau_latest/bgq/lib/Makefile.tau-[options]
   % soft add +tau_latest
   % make F90=tau_f90.sh

3. If stub makefile has \texttt{-papi} in its name, set the TAU_METRICS environment variable:
   % qsub --env
   TAU_METRICS=TIME:PAPI\_L2\_DCM:PAPI\_TOT\_CYC...

4. Build and run workshop examples, then run \texttt{pprof/paraprof}

Paratools
More Information

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

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

• VAMPIR References
  – VAMPIR-NG website
    http://www.vampir-ng.de/

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

• Eclipse PTP References
  – Documentation available from the Eclipse PTP website:
    http://www.eclipse.org/ptp/
Acknowledgements

- Department of Energy
  - Office of Science
  - Argonne National Laboratory
  - ORNL
  - NNSA/ASC Trilabs (SNL, LLNL, LANL)

- HPCMP DoD PET Program

- National Science Foundation

- University of Tennessee
  - Shirley Moore
  - Daniel Terpstra

- University of Oregon
  - Allen D. Malony, Chee Wai Lee
    W. Spear, S. Biersdorff

- TU Dresden
  - Holger Brunst, Andreas Knupfer
    Wolfgang Nagel

- Research Centre Juelich, Germany
  - Bernd Mohr
  - Felix Wolf