TAU PERFORMANCE SYSTEM

Wyatt Spear
Sameer Shende, Alan Morris, Scott Biersdorff
Performance Research Lab

Allen D. Malony, Suzanne Millstein
Department of Computer and Information Science
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, ...)
- http://tau.uoregon.edu
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
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
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)
  – 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
• Events have a type, a group association, and a name
• TAU events names are character strings
  – Powerful way to encode event information
  – Inefficient way to communicate each event occurrence
• TAU maps a new event name to an event ID
  – Done when event is first encountered (get event handle)
  – Event ID is used for subsequent event occurrences
  – Assigning a uniform event ID a priori is problematic
• A new event is identified by a new event name in TAU
  – Can create new event names at runtime
  – Allows for dynamic events (TAU renames events)
  – Allows for context-based, parameter-based, phase events
Automatic Source-level Instrumentation

- TAU source analyzer
- Application source
- Parsed program
- tau_instrumentor
- Instrumentation specification file
- Instrumented source
Program Database Toolkit (PDT)

Application / Library

C / C++ parser

Fortran parser F77/90/95

C / C++ IL analyzer

Fortran IL analyzer

Program Database Files

DUCTAPE

TAU instrumentor

Automatic source instrumentation
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
The Basics

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

- Install TAU
  - `% configure [options]; make clean install`
- Modify application makefile and choose TAU configuration
  - Select TAU’s stub makefile
  - Change name of compiler in makefile
- Set environment variables
  - Directory where profiles/traces are to be stored/counter selection
  - TAU options
- Execute application
  - `% mpirun -np <procs> a.out;`
- Analyze performance data
  - paraprof, vampir, pprof, paraver ...
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
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
TAU Measurement System Configuration

- configure [OPTIONS]
  - {-c++=<CC>, -cc=<cc>} Specify C++ and C compilers
  - -pdt=<dir> Specify location of PDT
  - -opari=<dir> Specify location of Opari OpenMP tool
  - -papi=<dir> Specify location of PAPI
  - -vampirtrace=<dir> Specify location of VampirTrace
  - -mpi[inc/lib]=<dir> Specify MPI library instrumentation
  - -dyninst=<dir> Specify location of DynInst Package
  - -shmem[inc/lib]=<dir> Specify PSHMEM library instrumentation
  - -python[inc/lib]=<dir> Specify Python instrumentation
  - -tag=<name> Specify a unique configuration name
  - -epilog=<dir> Specify location of EPILOG
  - -slog2 Build SLOG2/Jumpshot tracing package
  - -otf=<dir> Specify location of OTF trace package
  - -arch=<architecture> Specify architecture explicitly
    (bgl, xt3,x86_64,x86_64linux…)
  - {-pthread, -sproc} Use pthread or SGI sproc threads
  - -openmp Use OpenMP threads
  - -jdk=<dir> Specify Java instrumentation (JDK)
  - -fortran=[vendor] Specify Fortran compiler
TAU Measurement System Configuration

- configure [OPTIONS]
  - -TRACE Generate binary TAU traces
  - -PROFILE (default) Generate profiles (summary)
  - -PROFILECALLPATH Generate call path profiles
  - -PROFILEPHASE Generate phase based profiles
  - -PROFILEMEMORY Track heap memory for each routine
  - -PROFILEHEADROOM Track memory headroom to grow
  - -USE hardware counters + time
  - -COMPENSATE Compensate timer overhead
  - -CPUTIME Use usertime+system time
  - -PAPIWALLCLOCK Use PAPI’s wallclock time
  - -PAPIVIRTUAL Use PAPI’s process virtual time
  - -SGITIMERS Use fast IRIX timers
  - -LINUXTIMERS Use fast x86 Linux timers
TAU Configuration – Examples

- Configure using PDT and MPI for x86_64 Linux
  
  ```
  ./configure -pdt=/usr/pkgs/pkgs/pdtoolkit-3.15
  -mpiinc=/usr/pkgs/mpich/include -mpilib=/usr/pkgs/mpich/lib
  
  -mpilib.library=‘-lmpich -L/usr/gm/lib64 -lgm -lpthread -ldl’
  ```

- Use PAPI counters (one or more) with C/C++/F90 automatic instrumentation for Cray CNL. Also instrument the MPI library. Use PGI compilers.
  
  ```
  ./configure -arch=craycnl -papi=/opt/xt-tools/papi/3.6.2 -mpi;
  make clean install
  ```

- Stub makefiles
  
  ```
  /usr/pkgs/tau/x86_64/lib/Makefile.tau-mpi-pdt-pgi
  /usr/pkgs/tau/x86_64/lib/Makefile.tau-mpi-papi-pdt-pgi
  ```
• TAU scripts use stub makefiles to select performance measurements
• Variables:
  – TAU_CXX Specify the C++ compiler used by TAU
  – TAU_CC, TAU_F90 Specify the C, F90 compilers
  – TAU_DEFS Defines used by TAU (add to CFLAGS)
  – TAU_LDFLAGS Linker options (add to LDFLAGS)
  – TAU_INCLUDE Header files include path (add to CFLAGS)
  – TAU_LIBS Statically linked TAU library (add to LIBS)
  – TAU_SHLIBS Dynamically linked TAU library
  – TAU_MPI_LIBS TAU’s MPI wrapper library for C/C++
  – TAU_MPI_FLIBS TAU’s MPI wrapper library for F90
  – TAU_FORTRANLIBS Must be linked in with C++ linker for F90
  – TAU_CXXLIBS Must be linked in with F90 linker
  – TAU_INCLUDE_MEMORY Use TAU’s malloc/free wrapper lib
  – TAU_DISABLE TAU’s dummy F90 stub library
  – TAU_COMPILER Instrument using tau_compiler.sh script
% cd /usr/local/packages/tau-2.19.1/i386_linux/lib; ls Makefile.* on LiveDVD
Makefile.tau-pdt
Makefile.tau-mpi-pdt
Makefile.tau-papi-mpi-pdt
Makefile.tau-vampirtrace-papi-mpi-pdt
Makefile.tau-scalasca-papi-mpi-pdt
Makefile.tau-pthread-pdt
Makefile.tau-pthread-mpi-pdt
Makefile.tau-openmp-opari-pdt
Makefile.tau-openmp-opari-mpi-pdt
Makefile.tau-papi-openmp-opari-mpi-pdt
...

- **For an MPI+F90 application, you may want to start with:**
  Makefile.tau-mpi-pdt
    - Supports MPI instrumentation & PDT for automatic source instrumentation
    - % export TAU_MAKEFILE = /usr/local/packages/tau-2.19.1/i386_linux/lib/Makefile.tau-mpi-pdt
• Generates flat profiles
  – One for each MPI process
  – It is the default option.
• Uses wallclock time
  – gettimeofday() sys call
• Calculates exclusive, inclusive time spent in each timer/routine and number of calls
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 (per-iteration) phases
  – Phase profiles may be generated from full callpath profiles in paraprof by choosing events as phases
Interval Events, Atomic Events in TAU

<table>
<thead>
<tr>
<th>Time</th>
<th>Excl. msec</th>
<th>Incl. total msec</th>
<th>#Call</th>
<th>#Subrs</th>
<th>Incl. Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0</td>
<td>0.187</td>
<td>1.105</td>
<td>1</td>
<td>44</td>
<td>1105859 int main(int. char **) C</td>
</tr>
<tr>
<td>93.2</td>
<td>1.030</td>
<td>1.030</td>
<td>1</td>
<td>0</td>
<td>1030654 MPI_Init()</td>
</tr>
<tr>
<td>5.9</td>
<td>0.879</td>
<td>65</td>
<td>40</td>
<td>320</td>
<td>1637 void func(int. int) C</td>
</tr>
<tr>
<td>4.6</td>
<td>51</td>
<td>51</td>
<td>40</td>
<td>0</td>
<td>1277 MPI_Barrier()</td>
</tr>
<tr>
<td>1.2</td>
<td>13</td>
<td>13</td>
<td>120</td>
<td>0</td>
<td>111 MPI_Recv()</td>
</tr>
<tr>
<td>0.8</td>
<td>9</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>9328 MPI_Finalize()</td>
</tr>
<tr>
<td>0.0</td>
<td>0.137</td>
<td>0.137</td>
<td>120</td>
<td>0</td>
<td>1 MPI_Send()</td>
</tr>
<tr>
<td>0.0</td>
<td>0.086</td>
<td>0.086</td>
<td>40</td>
<td>0</td>
<td>2 MPI_Bcast()</td>
</tr>
<tr>
<td>0.0</td>
<td>0.002</td>
<td>0.002</td>
<td>1</td>
<td>0</td>
<td>2 MPI_Comm_size()</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_rank()</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>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>40</td>
<td>Message size for broadcast</td>
</tr>
</tbody>
</table>

% export TAU_CALLPATH_DEPTH = 0
% export TAU_TRACK_HEAP = 1
**Atomic Events, Context Events**

---

### Time Event Table

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

---

### User Events Profile

- **NumSamples**: 40
- **MaxValue**: 5.139E+04
- **MinValue**: 44.39
- **MeanValue**: 3.091E+04
- **Std Dev.**: 1.234E+04
- **Event Name**: Message size for broadcast

---

**% export TAU_CALLPATH DEPTH = 1**

**% export TAU_TRACK HEAP = 1**

---

CScADS 2010 24
Context Events (default)

% export TAU_CALLPATH_DEPTH = 2
% export TAU_TRACK_HEAP = 1
Routine Level Profile

- Goal: What routines account for the most time? How much?

Metric: P_VIRTUAL_TIME
Value: Exclusive
Units: seconds

9647.318 | LEQ_IKSWEEEP
4357.213 | LEQ_BICGS0T
2669.887 | LEQ_MATVECT
1777.752 | SOLVE_SPECIES_EQ
1417.986 | SOLVE_LIN_EQ
1028.448 | PHYSICAL_PROP
  783.402 | RRATES
   682.376 | LEQ_MSOLVET
   530.858 | INIT_AB_M
   463.788 | CALC_MASS_FLUX_SPHR
   446.025 | INIT_MU_S
   421.747 | CALC_RESID_S
   381.363 | SOLVE_ENERGY_EQ
   371.199 | SOURCE_PHI
   258.829 | DRAG_GS
% export TAU_MAKEFILE /opt/tau-2.19.1/x86_64 = /lib/Makefile.tau-mpi
% export TAU_OPTIONS = ‘-optCompInst -optVerbose’
% export 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 run.job
% paraprof --pack app.ppk
  Move the app.ppk file to your desktop.
% paraprof app.ppk
Goal: Easily generate routine level performance data using the compiler instead of PDT for parsing the source code.
% tau_instrumentor
Usage: tau_instrumentor <pdbfile> <sourcefile> [-o <outputfile>] [-noinline]
[-g groupname] [-i headerfile] [-c|-c++|-fortran] [-f <instr_req_file> ]
For selective instrumentation, use -f option
% tau_instrumentor foo.pdb foo.cpp -o foo.inst.cpp -f selective.dat
% cat selective.dat
# Selective instrumentation: Specify an exclude/include list of routines/files.
BEGIN_EXCLUDE_LIST
void quicksort(int *, int, int)
void sort_5elements(int *)
void interchange(int *, int *)
END_EXCLUDE_LIST

BEGIN_FILE_INCLUDE_LIST
Main.cpp
Foo?..c
*.C
END_FILE_INCLUDE_LIST
# Instruments routines in Main.cpp, Foo?..c and *.C files only
# Use BEGIN_[FILE]_INCLUDE_LIST with END_[FILE]_INCLUDE_LIST
Selective Instrumentation File

- Specify a list of events to exclude or include
- Focus on routines of interest/reduce overhead
- Automatic generation with Paraprof
- \# 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
• 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
```
Generating a loop level profile

% export TAU_MAKEFILE /opt/tau-2.19.1/x86_64 = /lib/Makefile.tau-mpi-pdt
% export 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
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

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loop: MULTIPLY_MATRICES</td>
<td>1729975.333</td>
</tr>
<tr>
<td>MPI_Recv()</td>
<td>443194</td>
</tr>
<tr>
<td>MAIN</td>
<td>81095</td>
</tr>
<tr>
<td>MPI_Bcast()</td>
<td>49569</td>
</tr>
<tr>
<td>Loop: MAIN</td>
<td>45669</td>
</tr>
<tr>
<td>MPI_Send()</td>
<td>12412</td>
</tr>
<tr>
<td>Loop: INITIALIZE</td>
<td>8959</td>
</tr>
<tr>
<td>Loop: INITIALIZE</td>
<td>8953</td>
</tr>
<tr>
<td>MPI_Finalize()</td>
<td>5609.2</td>
</tr>
<tr>
<td>MULTIPLY_MATRICES</td>
<td>2932.667</td>
</tr>
<tr>
<td>Loop: MAIN</td>
<td>2577.667</td>
</tr>
<tr>
<td>MPI_Barrier()</td>
<td>2091.8</td>
</tr>
<tr>
<td>Loop: MAIN</td>
<td>1875.667</td>
</tr>
<tr>
<td>Loop: MAIN</td>
<td>1833</td>
</tr>
<tr>
<td>Loop: MAIN</td>
<td>107</td>
</tr>
<tr>
<td>INITIALIZE</td>
<td>30</td>
</tr>
<tr>
<td>MPI_Comm_rank()</td>
<td>14.25</td>
</tr>
<tr>
<td>MPI_Comm_size()</td>
<td>1</td>
</tr>
</tbody>
</table>
-papi Option

• Instead of one metric, profile or trace with more than one metric
  – Set environment variable TAU_METRICS to specify the metric
    • % export TAU_METRICS = TIME:PAPI_FP_INS:PAPI_L1_DCM...
    • % export TAU_METRICS = TIME:PAPI_NATIVE_<native_event>..

• When used with tracing (TAU_TRACE=1) option, the first counter must be TIME
  • % export TAU_METRICS TIME:PAPI_FP_INS...
  • Provides a globally synchronized real time clock for tracing

• -papi appears in the name of the stub Makefile
• papi_avail, papi_event_chooser, and papi_native_avail are useful tools
Generate a PAPI profile

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

% 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)
% export 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 -> Click PAPI_FP_INS, Click /,
    Click TIME, Apply, choose the metric
```
**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

  ![Profile Diagram]

CScADS 2010

37
-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+)
-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
- TAU_DEPTH_LIMIT environment variable specifies depth
  - % export TAU_DEPTH_LIMIT = 1
    - enables instrumentation in only “main”
  - % export TAU_DEPTH_LIMIT = 2
    - enables instrumentation in main and routines that are directly called by main
- Stub makefile has -depthlimit in its name:
  - export TAU_MAKEFILE = <taudir>/<arch>/lib/Makefile.tau-mpi-depthlimit-pdt
Generate a Callpath Profile

```bash
% export TAU_MAKEFILE /opt/tau-2.19.1/x86_64 = /lib/Makefile.tau-mpi-pdt
% export PATH=/opt/tau-2.19.1/x86_64/bin:$PATH
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% export TAU_CALLPATH = 1
% export TAU_CALLPATHDEPTH = 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)
```
Generate a Callpath Profile
Callpath Profile

- Generates program callgraph
Tracing in TAU

• Generates event-trace logs, rather than summary profiles
  – export 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
Generate a Trace File

% export TAU_MAKEFILE = /opt/tau-2.19.1/x86_64/lib/Makefile.tau-mpi-pdt
% export PATH=/opt/tau-2.19.1/x86_64/bin:$PATH
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% export 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
Detect Memory Leaks

% export TAU_MAKEFILE /opt/tau-2.19.1/x86_64 = /lib/Makefile.tau-mpi-pdt
% export TAU_OPTIONS = `-optDetectMemoryLeaks -optVerbose`
(Or: export TAU_TRACK_MEMORY_LEAKS = 1)
% module load tau
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% export 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: export TAU_TRACK_HEAP = 1 and export 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.
Detect Memory Leaks

TAU: ParaProf: Mean Context Events - mem.ppk

<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></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MATRICES::ALLOCATE_MATRICES ([matrix.f90] {10,7}–{13,38})</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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>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></td>
<td></td>
<td></td>
<td></td>
<td></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>Value</th>
<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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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
Generating a flat profile with MPI

% export TAU_MAKEFILE /opt/tau-2.19.1/x86_64 =
   /lib/Makefile.tau-mpi-pdt
% export 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
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_CALLPATHDEPTH 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 ./matmult
> # Using tau_exec with DyninstAPI:
> tau_run matmult -o matmult.i
> mpirun -np 4 tau_exec -io -memory ./matmult.i
>
> tau_run -XrunTAUsh-papi-mpi-pdt matmult -o matmult.i
> mpirun -np 4 tau_exec -io -memory -T papi.mpi.pdt ./matmult.i
> paraprof
```
% export TAU_MAKEFILE /opt/tau-2.19.1/x86_64 =
    /lib/Makefile.tau-python-mpi-pdt
% export PATH=/opt/tau-2.19.1/x86_64/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 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:
% export PYTHONPATH = <taudir>/x86_64/lib/bindings-python-mpi-pdt-pgi
(same options string as TAU_MAKEFILE)
export LD_LIBRARY_PATH = <taudir>/x86_64/lib/bindings-icpc-python-mpi-pdt-pgi":$LD_LIBRARY_PATH
% poe <dir>/pyMPI-2.5b0-<TAU>/bin/pyMPI ./wrapper.py -procs 4
(Instrumented pyMPI with wrapper.py)
Mixed Python+F90+C+pyMPI

• Goal: Generate multi-level instrumentation for Python+MPI+C+F90+C++ ...
# 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_TRACE_HEAP or</td>
<td>0</td>
<td>Setting to 1 turns on tracking heap memory/headroom at routine entry &amp; exit</td>
</tr>
<tr>
<td>TAU_TRACE_HEADROOM</td>
<td></td>
<td>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</td>
</tr>
<tr>
<td></td>
<td></td>
<td>information, setting to 1 generates flat profile and context events have just</td>
</tr>
<tr>
<td></td>
<td></td>
<td>parent information (e.g., Heap Entry: foo)</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</td>
</tr>
<tr>
<td></td>
<td></td>
<td>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</td>
</tr>
<tr>
<td></td>
<td></td>
<td>times and takes less than 10 usec of inclusive time per call</td>
</tr>
<tr>
<td>TAU_COMPENSATE</td>
<td>0</td>
<td>Setting to 1 enables runtime compensation of instrumentation overhead</td>
</tr>
<tr>
<td>TAUPROFILE_FORMAT</td>
<td>Profile</td>
<td>Setting to “merged” generates a single file. “snapshot” generates xml format</td>
</tr>
<tr>
<td>TAU_METRICS</td>
<td>TIME</td>
<td>Setting to a comma separated list generates other metrics. (e.g.,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TIME:linux timers: PAPI_FP_OPS: PAPI_NATIVE &lt;event&gt;)</td>
</tr>
</tbody>
</table>
Compile-Time Variables

- 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)
  - -optPdtCxxOpts="" Options for C++ parser in PDT (cxxparse). Typically
  - $(TAU_MPI_INCLUDE) $(TAU_INCLUDE) $(TAU_DEFS)
  - -optPdtCxxOpts="" Options for C++ parser in PDT (cxxparse). Typically
  - $(TAU_MPI_INCLUDE) $(TAU_INCLUDE) $(TAU_DEFS)

...
TAU Analysis

Profile Data Management (PerfDMF)
- Profile translators
- Metadata (XML)
- Profile database

Profile Analysis (ParaProf)

Profile Data Mining (PerfExplorer)

Trace Data Management
- Trace translators
- Trace storage

Trace Visualizers
- Vampir
- JumpShot
- Paraver

Trace Analyzers
- Expert
- ProfileGen
- Vampir Server

Analysis
- Profiles
- Traces
- Profiles
- Symbol table

Instrumentation
- Event selection
- Event information
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)
Building Bridges to Other Tools
ParaProf Profile Analysis Framework

- Performance Data
  - Profiles
    - TAU, mpiP, ompP, HPMToolkit, Cube, HPCToolkit, Gprof, Dynaprof, PSRun
  - Runtime Data Collection
    - Supermon, MRNet
  - DBMS
    - PostgreSQL, MySQL, Oracle, DB2, Derby

- PerfDMF
  - Parsers and Importers
  - Basic Analysis + Derived Data
  - Internal Representation

- ParaProf
  - Profile Data
  - Call Graphs
  - Histograms
  - Call Trees
  - Bar Charts
  - Comparative Displays
  - Text Displays
  - Vis Package
    - JOGL
    - 3D Displays

- Scripting Interface
  - Jython
Performance Data Management

- 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), Open|SpeedShop, ...
  - Supported DBMS:
    - PostgreSQL, MySQL, Oracle, DB2, Derby/Cloudscape
  - Profile query and analysis API
- Reference implementation for PERI-DB project
PerfDMF Architecture

TAU Performance System
- profile metadata
- raw profiles
- * gprof
- * mpiP
- * psrun
- * HPMtoolkit
- *

Performance Analysis Programs
- scalability analysis
- ParaProf
- cluster analysis

Query and Analysis Toolkit
- Java PerfDMF API
- SQL (PostgreSQL, MySQL, DB2, Oracle)

Data Mining (Weka)
Statistics (R / Omega)
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
Performance Data Mining / Analytics

• 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
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...

Context Knowledge

- Source Code
- Build Environment
- Run Environment

Performance Result

Performance Knowledge

...to understand this

CScADS 2010
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
PerfExplorer v2 Architecture

Data Components:
- Performance Data
- Metadata
- Analysis Results
- Expert Knowledge

Analysis Components:
- Statistical Analysis
- Data Mining
- Inference Engine
- Provenance

DBMS (PerfDMF)

Process Control

Data Persistence
### Parallel Profile Analysis – pprof

![Image of parallel profile analysis output]

<table>
<thead>
<tr>
<th>%Time</th>
<th>Exclusive</th>
<th>Inclusive</th>
<th>#Call</th>
<th>#Subs</th>
<th>Inclusive Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0</td>
<td>3:11:1293</td>
<td>3:11:1293</td>
<td>15</td>
<td>1912393289</td>
<td>applu</td>
</tr>
<tr>
<td>99.6</td>
<td>3:10:463</td>
<td>3:10:463</td>
<td>3</td>
<td>37517</td>
<td>bcast_inputs</td>
</tr>
<tr>
<td>67.1</td>
<td>2:08:326</td>
<td>37200</td>
<td>387</td>
<td>3450</td>
<td>exchange_1</td>
</tr>
<tr>
<td>44.5</td>
<td>1:25:159</td>
<td>9300</td>
<td>18600</td>
<td>9157</td>
<td>bits</td>
</tr>
<tr>
<td>41.6</td>
<td>1:18:436</td>
<td>18600</td>
<td>0</td>
<td>4217</td>
<td>MPI_Recv()</td>
</tr>
<tr>
<td>26.2</td>
<td>50:142</td>
<td>50:142</td>
<td>0</td>
<td>2611</td>
<td>MPI_Send()</td>
</tr>
<tr>
<td>16.2</td>
<td>31:031</td>
<td>301</td>
<td>602</td>
<td>103096</td>
<td>rhsh</td>
</tr>
<tr>
<td>3.9</td>
<td>7:501</td>
<td>9300</td>
<td>0</td>
<td>807</td>
<td>jacid</td>
</tr>
<tr>
<td>3.4</td>
<td>6:594</td>
<td>1812</td>
<td>0</td>
<td>10916</td>
<td>exchange_3</td>
</tr>
<tr>
<td>2.6</td>
<td>4:989</td>
<td>608</td>
<td>0</td>
<td>8205</td>
<td>MPI_Wait()</td>
</tr>
<tr>
<td>0.2</td>
<td>0.44</td>
<td>1</td>
<td>4</td>
<td>400081</td>
<td>init_comm</td>
</tr>
<tr>
<td>0.2</td>
<td>399</td>
<td>399</td>
<td>1</td>
<td>39</td>
<td>399634</td>
</tr>
<tr>
<td>0.1</td>
<td>140</td>
<td>141</td>
<td>1</td>
<td>47626</td>
<td>actlv</td>
</tr>
<tr>
<td>0.1</td>
<td>134</td>
<td>134</td>
<td>2</td>
<td>57252</td>
<td></td>
</tr>
<tr>
<td>0.1</td>
<td>103</td>
<td>1</td>
<td>2</td>
<td>103168</td>
<td>rhsh</td>
</tr>
<tr>
<td>0.1</td>
<td>95</td>
<td>95</td>
<td>1</td>
<td>1</td>
<td>964858</td>
</tr>
<tr>
<td>0.0</td>
<td>25</td>
<td>95</td>
<td>9</td>
<td>1</td>
<td>106303</td>
</tr>
<tr>
<td>0.0</td>
<td>40</td>
<td>44</td>
<td>1</td>
<td>7937</td>
<td>error</td>
</tr>
<tr>
<td>0.0</td>
<td>24</td>
<td>608</td>
<td>0</td>
<td>40</td>
<td>MPI_Irecv()</td>
</tr>
<tr>
<td>0.0</td>
<td>16</td>
<td>15</td>
<td>5</td>
<td>15630</td>
<td>MPI_Finalize()</td>
</tr>
<tr>
<td>0.0</td>
<td>12</td>
<td>1</td>
<td>1700</td>
<td>12335</td>
<td>setbv</td>
</tr>
<tr>
<td>0.0</td>
<td>7</td>
<td>8</td>
<td>3</td>
<td>3</td>
<td>2893</td>
</tr>
<tr>
<td>0.0</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>451</td>
</tr>
<tr>
<td>0.0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3874</td>
</tr>
<tr>
<td>0.0</td>
<td>0.116</td>
<td>0.837</td>
<td>1</td>
<td>4</td>
<td>837</td>
</tr>
<tr>
<td>0.0</td>
<td>0.512</td>
<td>0.512</td>
<td>1</td>
<td>0</td>
<td>512</td>
</tr>
<tr>
<td>0.0</td>
<td>0.121</td>
<td>0.353</td>
<td>1</td>
<td>2</td>
<td>2353</td>
</tr>
<tr>
<td>0.0</td>
<td>0.024</td>
<td>0.191</td>
<td>1</td>
<td>2</td>
<td>131</td>
</tr>
<tr>
<td>0.0</td>
<td>0.103</td>
<td>0.103</td>
<td>0</td>
<td>17</td>
<td>MPI_Type_contiguous()</td>
</tr>
</tbody>
</table>

---

CScADS 2010

67
Parallel Profile Analysis –

- Raw files
- PerfDMF managed (database)
- Application
- Experiment
- Trial
- HPMToolkit
- Metadata
- MpiP
- TAU

CScADS 2010

File Options Help

Applications
- Standard Applications
  - Default App
  - Native Applications
- DB Applications
- CPU
- Cache
- OS
- I/O
- NIC
- HPC

Event Set 1
- Time
- Total time in main mode (seconds)
- CPU, Time (microseconds)
- CPU, Load (L1, L2, L3 cache hit misses)
- CPU, Load (L1, L2, L3 cache miss)
- CPU, Load (L1, L2, L3 cache direct misses)
- CPU, Load (L1, L2, L3 cache indirect misses)
- CPU, Load (L1, L2, L3 cache load references)
- CPU, Load (L1, L2, L3 cache store references)
- CPU, Load (L1, L2, L3 cache store indirect references)
- CPU, Load (L1, L2, L3 cache load references)
- CPU, Load (L1, L2, L3 cache store references)

Experiment
- Trial
- Metric Name: Time
- Value Type: exclusive

Database
- Name
- Application ID
- Experiment ID
- Trial ID
- MpiP
- TAU
Metadata for Each Experiment

TAU: ParaProf Manager

TrialField | Value
---|---
Name | f90/pdt_mpi/examples/tau2/amorris/home/
Application ID | 0
Experiment ID | 0
Trial ID | 0
CPU Cores | 2
CPU MHz | 2992.505
CPU Type | Intel(R) Xeon(R) CPU 5160 @ 3.00GHz
CPU Vendor | GenuineIntel
CWD | /home/amorris/tau2/examples/pdt_mpi/f90
Cache Size | 4096 KB
Executable | /home/amorris/tau2/examples/pdt_mpi/f90
Hostname | demon.nic.uoregon.edu
Local Time | 2007-07-04T04:14:07-07:00
MPI Processor Name | demon.nic.uoregon.edu
Memory Size | 8161240 kB
Node Name | demon.nic.uoregon.edu
OS Machine | x86_64
OS Name | Linux
OS Release | 2.6.9-42.0.3.EL.perfctrsmp
OS Version | #1 SMP Fri Nov 3 07:34:13 PST 2006
Starting Timestamp | 1183548072220996
TAU Architecture | x86_64
TAU Config | -papi=/usr/local/packages/papi-3.5.0-M...n
Timestamp | 1183548074317538
pid | 11395
username | amorris

Multiple PerfDMF DBs
ParaProf – Flat Profile

node, context, thread  8K processors

Miranda
- hydrodynamics
- Fortran + MPI
- LLNL BG/L
Comparing Effects of Multi-Core Processors

**Metric:** PAPI_RES_STL  
**Value:** Exclusive  
**Units:** counts

- **Cray XT3 (4K cores)**
- **Blue is single node**  
- **Red is dual core**  
- **CScADS 2010**

**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 - Stacked View
ParaProf – Callpath Profile

Flash
○ thermonuclear flashes
○ Fortran + MPI
○ Argonne
ParaProf – Scalable Histogram

8k processors

16k processors

CScADS 2010
ParaProf – 3D View (Full Profile)

xbec

128k processors
ParaProf – 3D View (Full Profile)

Miranda

16k processors
• Each point is a “thread” of execution
• A total of four metrics shown in relation
• ParaProf’s visualization library – JOGL
• Miranda, 32k cores
• 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;
    }
}
```
Performance Mapping

```c
int ProcessParticle(Particle *p) {
    /* perform some computation on p */
}
int main() {
    GenerateParticles();
    /* create a list of particles */
    for (int i = 0; i < N; i++)
        /* iterates over the list */
        ProcessParticle(P[i]);
}
```

- How much time (flops) spent processing face i particles?
- What is the distribution of performance among faces?
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

CScADS 2010
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

**Graphs:**

- 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)
- Generate using TAU_PROFILE_SNAPSHOT interface
Profile Snapshot Views

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

All windows dynamically update

TAU: ParaProf: Snapshot Controller:

Snapshot 80
Name: Iteration 80
Time Position: 9 Seconds
[Box]

TAU: ParaProf: Snapshot Controller:

Snapshot 154
Name: final
Time Position: 24 Seconds
[Box]

TAU: ParaProf: n.c.t 0.0.0 - flash4.xml
File Options Windows Help

Metric: Time
Value: Exclusive
Units: seconds

0.666
0.481
0.468
0.428
0.403
0.356
0.329
0.322
0.265
0.258
0.239
0.233
0.217
0.21
0.205
0.202
0.17
0.166
0.156
0.155
0.151
0.147
0.118
0.118
0.113
0.107
0.097
0.095
0.088
0.083

0.621
0.612
0.573
0.483
0.468
0.428
0.384
0.369
0.369
0.365
0.348
0.345
0.322
0.322
0.317
0.291
0.272
0.271
0.271
0.254
0.235

TAU: ParaProf: Snapshot Controller:

Snap 80
Name: Iteration 80
Time Position: 9 Seconds
[Box]

TAU: ParaProf: Snapshot Controller:

Snap 154
Name: final
Time Position: 24 Seconds
[Box]
PerfExplorer – Runtime Breakdown

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

- WRITE_SAVEFILE
- MPI_Waitall

---

Number of Processors

Percentage of Total Runtime

Legend:
- DERIVATIVE_X_COMM [(derivative_x.pp.f90) {53, 14}]
- Loop: CHEMKIN_M::REACTION_RATE_BOUNDS [(chemkin_m.pp.f90) {374, 3}–{386, 7}]
- Loop: DERIVATIVE_X_CALC [(derivative_x.pp.f90) {432, 10}–{441, 15}]
- Loop: DERIVATIVE_X_CALC [(derivative_x.pp.f90) {566, 19}–{589, 24}]
- Loop: DERIVATIVE_Y_CALC [(derivative_y.pp.f90) {431, 10}–{440, 15}]
- Loop: DERIVATIVE_Z_CALC [(derivative_z.pp.f90) {435, 10}–{444, 15}]
- Loop: INTEGRATE [(integrateerk.pp.f90) {73, 3}–{93, 13}]
- Loop: RHSF [(rhsf.pp.f90) {209, 3}–{211, 7}]
- Loop: RHSF [(rhsf.pp.f90) {515, 3}–{535, 16}]
- Loop: RHSF [(rhsf.pp.f90) {537, 3}–{543, 16}]
- Loop: RHSF [(rhsf.pp.f90) {545, 3}–{551, 16}]
- Loop: THERMCHEM_M::CALC_INV_AVG_MOL_WT [(thermchem_m.pp.f90) {127, 5}–{129, 9}]
- Loop: THERMCHEM_M::CALC_SPECENTH_ALLPTS [(thermchem_m.pp.f90) {506, 3}–{512, 8}]
- Loop: THERMCHEM_M::CALC_TEMP [(thermchem_m.pp.f90) {175, 5}–{216, 9}]
- Loop: TRANSPORT_M::COMPUTE_COEFFICIENTS [(mixavg_transport.m.pp.f90) {492, 5}–{520, 9}]
- Loop: TRANSPORT_M::COMPUTE_HEATFLUX [(mixavg_transport.m.pp.f90) {782, 5}–{790, 19}]
- Loop: TRANSPORT_M::COMPUTE_SPECIESDIFFFLUX [(mixavg_transport.m.pp.f90) {630, 5}–{656, 19}]
- Loop: VARIABLES_M::GET_MASS_FRAC [(variables_m.pp.f90) {96, 3}–{99, 7}]
- MPI_Comm_compare0
- MPI_Wait0
- READWRITE_SAVEFILE_DATA [(io.pp.f90) {544, 14}]
- RHSF [(rhsf.pp.f90) {1, 12}]
- WRITE_SAVEFILE [(io.pp.f90) {240, 14}]
- other
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
PerfExplorer – Cluster Analysis

• Four significant events automatically selected
• Clusters and correlations are visible
PerfExplorer – Performance Regression
Performance Regression Testing

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

Date

Exclusive Time (seconds)

- 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<...
- void FcUpdaterComponent::dumpToFile(const std::string &, con...
- other

CScADS 2010
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

[Bar chart showing total LINUX_TIMERS for S3D Jaguar CNL:Scaling with various number of processors and colored bars indicating different processes and loops.]
Evaluate Scalability using PerfExplorer Charts

% export TAU_MAKEFILE /opt/tau-2.19.1/x86_64 =
   /lib/Makefile.tau-mpi-pdt
% export 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 1p.ppk
% qsub run2p.job ...
% paraprof --pack 2p.ppk ... and so on.
On your client:
% perfdmf_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 perfdmf_loadtrial on the commandline)
% perfexplorer
(Charts -> Speedup)
% 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)
Communication Matrix Display

- Goal: What is the volume of inter-process communication? Along which calling path?
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
  – Computational quality of service in CCA
• Unified profile output (-DTAU_EXP_UNIFY)
• Eclipse CDT and PTP integration
• GPU Support (CUDA/OpenCL)
• Performance tools integration (NSF POINT project)
CUDA Linpack Trace
Measuring Performance of PGI Accelerator Code

TAU: ParaProf: n,t,0,0,0 - matlk.ppk

Metric: TIME
Value: Exclusive percent

68.044%

24.417%

3.206%

1.572%

1.572%

0.782%

0.142%

0.122%

0.017%

0.005%

0.002%

2.1E-4%

1.2E-4%

1.2E-4%

__pgi_cu_launch multiply_matrices (pgi_kernel_7,gx=32,gy=32,gz=1,bx=16,by=16,bz=1) [[mm2.f90][15]]
__pgi_cu_init multiply_matrices [[mm2.f90][9]]
__pgi_cu_download2 multiply_matrices var=a [[mm2.f90][20]]
__pgi_cu_upload2 multiply_matrices var=b [[mm2.f90][9]]
__pgi_cu_upload2 multiply_matrices var=c [[mm2.f90][9]]
mymatrixmultiply [[mmdriv.f90][1,0]]
__pgi_cu_launch multiply_matrices (pgi_kernel_2,gx=32,gy=32,gz=1,bx=16,by=16,bz=1) [[mm2.f90][11]]
__pgi_cu_free multiply_matrices [[mm2.f90]]
__pgi_cu_alloc multiply_matrices [[mm2.f90][9]]
multiply_matrices [[mm2.f90][5,0]]
pgi accelerator region
pgi accelerator region
pgi accelerator region
pgi accelerator region
2.1E-4% __pgi_cu_module multiply_matrices [[mm2.f90][9]]
1.2E-4% __pgi_cu_module_function multiply_matrices [[mm2.f90][11]]
1.2E-4% __pgi_cu_module_function multiply_matrices [[mm2.f90]]
1.2E-4% __pgi_cu_module_function multiply_matrices [[mm2.f90]]
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 → TAU instrumented libraries → Compilation/linking with TAU libraries → Temporary copy of instrumented code → Program execution → Program output → Performance data.
TAU and Eclipse
Choosing PAPI Counters with TAU in Eclipse

% /usr/local/packages/eclipse/eclipse
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
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.
• 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