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

**TAU Architecture**
- **Measurement**: Source code, object code, library, binary, virtual machine
- **Instrumentation**: Event selection, event information
- **Profiling**: Event creation and management, statistics, I/O profiles, trace merging
- **Tracing**: Event control, record creation, trace I/O

**Program Analysis**
- PDT
- Parallel Profile Analysis

**Performance Data Mining**
- GUI
- Analysis Components
- Data Components
- Performance Data
- Metadata
- Data Components
- Analysis Components
- Data Mining
- Expert Knowledge
- Provenance

**TAUoverSupermon**
- Profile Data Mining (PerfExplorer)
- Trace Data Management
- Profile Analysis (ParaProf)
- Trace Visualizers
- Trace Analyzers
- Profile Data Mining (PerfExplorer)

**ParaProf**
- Performance Data
- PerfDFM
- ParaProf

**Graphic Elements**
- TAU Performance System
- Performance Analysis Programs
- Data Mining (ParaProf)
- Data Components
- Analysis Components
- Scripting Interface
- Python

**Other Tools**
- PerfExplorer Component Interfaces
- Data Components
- Performance Data
- Metadata
- Data Components
- Analysis Components
- Data Mining
- Expert Knowledge
- Provenance
TAU Instrumentation / Measurement

**Instrumentation**
- Source code
- Object code
- Library wrapper
- Binary code
- Virtual machine

**Measurement**

**Event creation and management**
- Event identifier
- Entry/exit events
- Atomic events
- Event mapping
- Event control

**Profiling**
- Statistics
- Atomic profiles
- Entry/exit profiles
- Profile sampling
- I/O profiles

**Tracing**
- Trace buffering
- Record creation
- Trace I/O
- Timestamp generation
- Trace filtering
- Trace merging

**Performance data sources**
- Timing
- Hardware counters
- System counters
- Kernel

**OS and runtime system modules**
- Threading
- Interrupts
- Runtime system
- I/O
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 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
• 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
TAU Instrumentation Mechanisms

• Source code
  – Manual (TAU API, TAU component API)
  – Automatic (robust)
    • C, C++, F77/90/95 (Program Database Toolkit (PDT))
    • OpenMP (directive rewriting (Opari), POMP2 spec)
    • Library header wrapping

• Object code
  – Pre-instrumented libraries (e.g., MPI using PMPI)
  – Statically- and dynamically-linked (with LD_PRELOAD)

• Executable code
  – Binary and dynamic instrumentation (Dyninst)
  – Virtual machine instrumentation (e.g., Java using JVMPI)

• TAU_COMPILER to automate instrumentation process
Automatic Source-level Instrumentation

TAU source analyzer

Application source

Parsed program

tau_instrumentor

Instrumented source

Instrumentation specification file
Program Database Toolkit (PDT)

Application / Library

C / C++ parser

Fortran parser F77/90/95

IL analyzer

Fortran IL analyzer

Program Database Files

DUCTAPE

TAU instrumentor

Automatic source instrumentation

6th VI-HPS Tuning Workshop: The TAU Performance System
MPI Wrapper Interposition Library

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

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

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

- Specify a list of events to exclude or include
- # is a wildcard in a routine name
  
  BEGIN_EXCLUDE_LIST
  Foo
  Bar
  D#EMM
  END_EXCLUDE_LIST

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

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

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

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

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

• Portable and scalable parallel profiling solution
  – Multiple profiling types and options
  – Event selection and control (enabling/disabling, throttling)
  – Online profile access and sampling
  – Online performance profile overhead compensation
• Portable and scalable parallel tracing solution
  – Trace translation to OTF, EPILOG, Paraver, and SLOG2
  – Trace streams (OTF) and hierarchical trace merging
• Robust timing and hardware performance support
• Multiple counters (hardware, user-defined, system)
• Performance measurement of I/O and Linux kernel
TAU Measurement Mechanisms

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

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

• Flat profiles
  – Metric (e.g., time) spent in an event (callgraph nodes)
  – Exclusive/inclusive, # of calls, child calls
• Callpath profiles (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
The TAU Performance System

TAU Analysis

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

Profile Analysis (ParaProf)
- Profile Data Mining (PerfExplorer)
- Vampir
- JumpShot
- Paraver
- Expert
- ProfileGen
- Vampir Server

Trace Data Management
- trace translators
- trace storage

TAU Portal
Performance Analysis

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

Parser and Importers
- Basic Analysis + Derived Data
- Internal Representation

Profile Data

ParaProf
- Call Graphs
- Histograms
- Call Trees
- Vis Package
- 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
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...

Performance Knowledge

...to understand this

Context Knowledge

- Source Code
- Build Environment
- Run Environment

Execution

Performance Result

VI-HPS

6th VI-HPS Tuning Workshop: The TAU Performance System
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

6th VI-HPS Tuning Workshop: The TAU Performance System
PerfExplorer v2 Architecture

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

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

Process Control

DBMS (PerfDMF)
### Parallel Profile Analysis – pprof

#### Profiling Output

<table>
<thead>
<tr>
<th>Function</th>
<th>Time (msec)</th>
<th>Inclusive Time (msec)</th>
<th>#Call</th>
<th>#Subs</th>
<th>Inclusive Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>プルーチューム</td>
<td>100.0</td>
<td>3:11:293</td>
<td>1</td>
<td>15</td>
<td>191233289 applu</td>
</tr>
<tr>
<td>プルーチューム</td>
<td>99.6</td>
<td>3:10:463</td>
<td>3</td>
<td>37512</td>
<td>634873925 broadcast</td>
</tr>
<tr>
<td>プルーチューム</td>
<td>67.1</td>
<td>2:08:326</td>
<td>37200</td>
<td>37200</td>
<td>3450 exchange_1</td>
</tr>
<tr>
<td>プルーチューム</td>
<td>44.5</td>
<td>1:25:169</td>
<td>9300</td>
<td>18600</td>
<td>9137 bups</td>
</tr>
<tr>
<td>プルーチューム</td>
<td>41.0</td>
<td>1:18:436</td>
<td>18600</td>
<td>0</td>
<td>4217 MPI_Recv()</td>
</tr>
<tr>
<td>プルーチューム</td>
<td>29.5</td>
<td>56:407</td>
<td>9300</td>
<td>18600</td>
<td>6065 bits</td>
</tr>
<tr>
<td>プルーチューム</td>
<td>26.2</td>
<td>50:142</td>
<td>18204</td>
<td>0</td>
<td>2811 MPI_Send()</td>
</tr>
<tr>
<td>プルーチューム</td>
<td>16.2</td>
<td>31:031</td>
<td>301</td>
<td>602</td>
<td>3103644 cmsg</td>
</tr>
<tr>
<td>プルーチューム</td>
<td>3.9</td>
<td>7:501</td>
<td>9300</td>
<td>0</td>
<td>107 jaccd</td>
</tr>
<tr>
<td>プルーチューム</td>
<td>3.4</td>
<td>6:594</td>
<td>604</td>
<td>1812</td>
<td>10916 exchange_3</td>
</tr>
<tr>
<td>プルーチューム</td>
<td>3.4</td>
<td>6:590</td>
<td>9300</td>
<td>0</td>
<td>709 jacc</td>
</tr>
<tr>
<td>プルーチューム</td>
<td>2.6</td>
<td>4:989</td>
<td>608</td>
<td>0</td>
<td>8205 MPI_Wait()</td>
</tr>
<tr>
<td>プルーチューム</td>
<td>0.2</td>
<td>0.44</td>
<td>1</td>
<td>4</td>
<td>4000800811 init_comm</td>
</tr>
<tr>
<td>プルーチューム</td>
<td>0.2</td>
<td>0.398</td>
<td>1</td>
<td>39</td>
<td>3998349 MPI_Init()</td>
</tr>
<tr>
<td>プルーチューム</td>
<td>0.1</td>
<td>0.247</td>
<td>1</td>
<td>47616</td>
<td>2470885 setiv</td>
</tr>
<tr>
<td>プルーチューム</td>
<td>0.1</td>
<td>0.131</td>
<td>57252</td>
<td>0</td>
<td>2 exact</td>
</tr>
<tr>
<td>プルーチューム</td>
<td>0.1</td>
<td>0.183</td>
<td>103</td>
<td>2</td>
<td>103168 enhs</td>
</tr>
<tr>
<td>プルーチューム</td>
<td>0.1</td>
<td>0.968</td>
<td>96</td>
<td>1</td>
<td>964588 read_input</td>
</tr>
<tr>
<td>プルーチューム</td>
<td>0.0</td>
<td>0.95</td>
<td>9</td>
<td>9</td>
<td>10603 MPI_Bcast()</td>
</tr>
<tr>
<td>プルーチューム</td>
<td>0.0</td>
<td>0.26</td>
<td>44</td>
<td>7937</td>
<td>44878 error</td>
</tr>
<tr>
<td>プルーチューム</td>
<td>0.0</td>
<td>0.234</td>
<td>608</td>
<td>0</td>
<td>156303 MPI_Finalize()</td>
</tr>
<tr>
<td>プルーチューム</td>
<td>0.0</td>
<td>0.151</td>
<td>1</td>
<td>5</td>
<td>12335 setbv</td>
</tr>
<tr>
<td>プルーチューム</td>
<td>0.0</td>
<td>0.4</td>
<td>12</td>
<td>1700</td>
<td>2883 12norm</td>
</tr>
<tr>
<td>プルーチューム</td>
<td>0.0</td>
<td>0.03</td>
<td>8</td>
<td>3</td>
<td>4911 Allreduce()</td>
</tr>
<tr>
<td>プルーチューム</td>
<td>0.0</td>
<td>0.001</td>
<td>1</td>
<td>6</td>
<td>3874 printerr</td>
</tr>
<tr>
<td>プルーチューム</td>
<td>0.0</td>
<td>0.0001</td>
<td>1</td>
<td>0</td>
<td>1007 MPI_Barrier()</td>
</tr>
<tr>
<td>プルーチューム</td>
<td>0.0</td>
<td>0.118</td>
<td>0.837</td>
<td>1</td>
<td>837 exchange_4</td>
</tr>
<tr>
<td>プルーチューム</td>
<td>0.0</td>
<td>0.121</td>
<td>0.512</td>
<td>1</td>
<td>512 MPI_Keyval_create()</td>
</tr>
<tr>
<td>プルーチューム</td>
<td>0.0</td>
<td>0.121</td>
<td>0.353</td>
<td>1</td>
<td>353 exchange_5</td>
</tr>
<tr>
<td>プルーチューム</td>
<td>0.0</td>
<td>0.024</td>
<td>0.191</td>
<td>2</td>
<td>131 exchange_6</td>
</tr>
<tr>
<td>プルーチューム</td>
<td>0.0</td>
<td>0.013</td>
<td>0.103</td>
<td>0</td>
<td>17 MPI_Type_contiguous()</td>
</tr>
</tbody>
</table>
Parallel Profile Analysis – ParaProf

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

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

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

Metric: FAPI_FP_OPS / GET_TIME_OF_DAY

Value: Exclusive
Units: Derived metric shown in microseconds format

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
ParaProf – 3D View (Full Profile)

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

Miranda

16k processors
ParaProf – 3D Scatterplot

• Each point is a “thread” of execution
• A total of four metrics shown in relation
• ParaProf’s visualization library
  – JOGL
• Miranda, 32k cores
Example: Particles distributed on cube surface

```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;
    }
}
```
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?
No Mapping versus Mapping

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

• TAU’s performance mapping can observe performance with respect to scientist’s programming and problem abstractions

![TAU (no mapping)](image1)

![TAU (w/ mapping)](image2)
How is MPI_Wait() distributed relative to solver direction?

Application routine names reflect phase semantics.
NAS BT – Phase Profile

Main phase shows nested phases and immediate events

File Options Windows Help

ParaProf: MPBT/phase/amorris/home/

mean
n,c,t 0,0,0
n,c,t 1,0,0
n,c,t 2,0,0
n,c,t 3,0,0
n,c,t 4,0,0
n,c,t 5,0,0
n,c,t 6,0,0
n,c,t 7,0,0
n,c,t 8,0,0

y_solve phase

z_solve phase

x_solve phase

MPI_Waitall()
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 showing:
- 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)
Profile Snapshot Views

- Only show main loop
- Percentage breakdown

![Profile Snapshot Views](image-url)
Snapshot Replay in ParaProf

All windows dynamically update
PerfExplorer – Runtime Breakdown

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

- WRITE_SAVEFILE
- MPI_Waitall

- 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 [integrate.erl.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: THERMCHM_M::CALC_INV_AVG_MOL_WT [thermchem_m.pp.f90] (127, 5)–(129, 9)
- Loop: THERMCHM_M::CALC_SPECENTH_ALLPTS [thermchem_m.pp.f90] (506, 3)–(512, 8)
- Loop: THERMCHM_M::CALC_TEMP [thermchem_m.pp.f90] (175, 5)–(186, 9)
- Loop: TRANSPORT_M::COMPUTECOEFFICIENTS [mixavg_transport_m.pp.f90] (492, 5)–(520, 9)
- Loop: TRANSPORT_M::COMPUTEHEATFLUX [mixavg_transport_m.pp.f90] (782, 5)–(790, 19)
- Loop: TRANSPORT_M::COMPUTESPECIESDIFFFLUX [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
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
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
• Eclipse CDT and PTP integration
• Performance tools integration (NSF POINT project)
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
   -mpilibrary=‘-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
Stub Makefiles Configuration Parameters

- 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
TAU Measurement Configuration

• % cd /opt/tau-2.19.1/x86_64/lib; ls Makefile.*
  – Makefile.tau-pdt
  – Makefile.tau-mpi-pdt
  – Makefile.tau-mpi-papi-pdt
  – Makefile.tau-mpi-papi-pdt-ptrace
  – 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

• % setenv TAU_MAKEFILE
  /opt/tau-2.19.1/x86_64/lib/Makefile.tau-mpi-pdt
Using TAU: A brief Introduction

- To instrument source code using PDT
  - Choose an appropriate TAU stub makefile in <arch>/lib:
    
    ```
    % setenv TAU_MAKEFILE
    /opt/tau-2.19.1/x86_64/lib/Makefile.tau-mpi-pdt
    % setenv TAU_OPTIONS ‘-optVerbose ...’ (see tau_compiler.sh)
    ```
    And use tau_f90.sh, tau_cxx.sh or tau_cc.sh as Fortran, C++ or C compilers:
    
    ```
    % mpif90 foo.f90
    changes to
    % tau_f90.sh foo.f90
    ```

- Execute application and analyze performance data:
  
  ```
  % pprof  (for text based profile display)
  % paraprof (for GUI)
  ```
% 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
  – % setenv TAU_MAKEFILE
     /usr/local/packages/tau-2.19.1/i386_linux/lib/Makefile.tau-mpi-pdt
-PROFILE Option

- 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 and number of calls
Generating a Flat Profile with MPI

% setenv TAU_MAKEFILE /opt/tau-2.19.1/x86_64
    /lib/Makefile.tau-mpi-pdt
% set path=/opt/tau-2.19.1/x86_64/bin $path
% 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
Generating a Loop-level Profile

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

% set path=/opt/tau-2.19.1/x86_64/bin $path
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% qsub run.job
% paraprof --pack app.ppk
   Move the app.ppk file to your desktop.
% paraprof app.ppk
Compiler-based Instrumentation

% setenv TAU_MAKEFILE /opt/tau-2.19.1/x86_64
    /lib/Makefile.tau-mpi
% setenv TAU_OPTIONS `--optCompInst --optVerbose`
% % set path=/opt/tau-2.19.1/x86_64/bin $path
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)

% qsub run.jjob
% paraprof --pack app.ppk
   Move the app.ppk file to your desktop.
% paraprof app.ppk
-papi Option

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

• When used with tracing (TAU_TRACE=1) option, the first counter must be TIME
  • % setenv 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

```bash
% setenv TAU_MAKEFILE /opt/tau-2.19.1/x86_64/lib/Makefile.tau-papi-mpi-pdt
% setenv TAU_OPTIONS `-opt TauSelectFile=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)
% setenv TAU_METRICS TIME:PAPI_FP_INS

% qsub run.job
% paraprof --pack app.ppk
  Move the app.ppk file to your desktop.
% paraprof app.ppk
  Choose Options -> Show Derived Panel -> Click PAPI_FP_INS, Click / , Click TIME, Apply, choose the metric
```
-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

• TAUDEPTH_LIMIT environment variable specifies depth
  – % setenv TAUDEPTH_LIMIT 1
  – enables instrumentation in only “main”
  – % setenv TAUDEPTH_LIMIT 2
  – enables instrumentation in main and routines that are directly called by main

• Stub makefile has -depthlimit in its name:
  – setenv TAUMAKEFILE <taudir>/<arch>/lib/Makefile.tau-mpi-depthlimit-pdt
Generate a Callpath Profile

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

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

- Generates event-trace logs, rather than summary profiles
  - setenv TAU_TRACE 1
- Traces show when and where an event occurred in terms of location and the process that executed it
- Traces from multiple processes are merged:
  - % tau_treemerge.pl
    - generates tau.trc and tau.edf as merged trace and event definition file
- TAU traces can be converted to Vampir’s OTF/VTF3, Jumpshot SLOG2, Paraver trace formats:
  - % tau2otf tau.trc tau.edf app.otf
  - % tau2vtf tau.trc tau.edf app.vpt.gz
  - % tau2slog2 tau.trc tau.edf -o app.slog2
  - % tau_convert -paraver tau.trc tau.edf app.prv
Generate a Trace File

% setenv TAU_MAKEFILE /opt/tau-2.19.1/x86_64
   /lib/Makefile.tau-mpi-pdt
% set path=/opt/tau-2.19.1/x86_64/bin
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% setenv TAU_TRACE 1
% qsub run.job
% tau_treemerge.pl
(merges binary traces to create tau.trc and tau.edf files)
JUMPSHOT:
% tau2slog2 tau.trc tau.edf -o app.slog2
% jumpshot app.slog2
   OR
VAMPIR:
% tau2otf tau.trc tau.edf app.otf -n 4 -z
(4 streams, compressed output trace)
% vampir app.otf
(or vng client with vngd server)
Instrumentation Specification

% 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
BEGIN_INSTRUMENT_SECTION
loops file="loop_test.cpp" routine="multiply"
# it also understands # as the wildcard in routine name
# and * and ? wildcards in file name.
# You can also specify the full
# name of the routine as is found in profile files.
#loops file="loop_test.cpp" routine="double multiply#"
END_INSTRUMENT_SECTION

% pprof
NODE 0;CONTEXT 0;THREAD 0:

%Time | Exclusive | Inclusive | #Call | #Subrs | Inclusive Name
      | msec      | total msec|       |        | usec/call
---------------------------------------------------------------------------------------
100.0 | 0.12      | 25,162    | 1     | 1      | 25162827 int main(int, char **)
100.0 | 0.175     | 25,162    | 1     | 4      | 25162707 double multiply()
90.5  | 22,778    | 22,778    | 1     | 0      | 22778959 Loop: double multiply()[
    file = <loop_test.cpp> line,col = <23,3> to <30,3> ]
  9.3  | 2,345     | 2,345     | 1     | 0      | 2345823 Loop: double multiply()[
    file = <loop_test.cpp> line,col = <38,3> to <46,7> ]
  0.1  | 33        | 33        | 1     | 0      | 33964 Loop: double multiply()[ file = <loop_test.cpp> line,col = <16,10> to <21,12> ]

6th VI-HPS Tuning Workshop: The TAU Performance System
Using TAU: A brief Introduction

• To instrument source code using PDT
  – Choose an appropriate TAU stub makefile in <arch>/lib:
    % setenv TAU_MAKEFILE
    /opt/tau-2.19.1/x86_64/lib/Makefile.tau-mpi-pdt
    % setenv TAU_OPTIONS ‘-optVerbose ...’ (see tau_compiler.sh)
    And use tau_f90.sh, tau_cxx.sh or tau_cc.sh as Fortran, C++ or C
    compilers:
    % mpif90 foo.f90
    changes to
    % tau_f90.sh foo.f90

• Execute application and analyze performance data:
  % pprof  (for text based profile display)
  % paraprof  (for GUI)
Goal: What routines account for the most time? How much?

Metric: P_VIRTUAL_TIME
Value: Exclusive
Units: seconds

<table>
<thead>
<tr>
<th>Routine</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEQ_IKSWEEPPT</td>
<td>9647.318</td>
</tr>
<tr>
<td>LEQ_BICGS0T</td>
<td>4357.213</td>
</tr>
<tr>
<td>LEQ_MATVECT</td>
<td>2669.887</td>
</tr>
<tr>
<td>SOLVE_SPECIES_EQ</td>
<td>1777.752</td>
</tr>
<tr>
<td>SOLVE_LIN_EQ</td>
<td>1417.986</td>
</tr>
<tr>
<td>PHYSICAL_PROP</td>
<td>1028.448</td>
</tr>
<tr>
<td>RRATES</td>
<td>783.402</td>
</tr>
<tr>
<td>LEQ_MSOLVET</td>
<td>682.376</td>
</tr>
<tr>
<td>INIT_AB_M</td>
<td>530.858</td>
</tr>
<tr>
<td>CALC_MASS_FLUX_SPHR</td>
<td>463.788</td>
</tr>
<tr>
<td>INIT_MU_S</td>
<td>446.025</td>
</tr>
<tr>
<td>CALC_RESID_S</td>
<td>421.747</td>
</tr>
<tr>
<td>SOLVE_ENERGY_EQ</td>
<td>381.363</td>
</tr>
<tr>
<td>SOURCE_PHI</td>
<td>371.199</td>
</tr>
<tr>
<td>DRAG_GS</td>
<td>258.829</td>
</tr>
</tbody>
</table>
Solution: Generating a flat profile with MPI

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

Or
% module load tau
% make F90=tau_f90.sh

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

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

% paraprof app.ppk
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
1729975.333  81095  MAIN
1729975.333  49569  MPI_Bcast()
1729975.333  45669  Loop: MAIN [{matmult.f90} [86,9]-[106,14]]
1729975.333  12412  MPI_Send()
1729975.333  8959   Loop: INITIALIZE [{matmult.f90} [17,9]-[21,14]]
1729975.333  8953   Loop: INITIALIZE [{matmult.f90} [10,9]-[14,14]]
1729975.333  5609.2  MPI_Finalize()
1729975.333  2932.667 MULTIPLE_MATRICES
1729975.333  2577.667 Loop: MAIN [{matmult.f90} [117,9]-[128,14]]
1729975.333  2091.8  MPI_Barrier()
1729975.333  1875.667 Loop: MAIN [{matmult.f90} [112,9]-[115,14]]
1729975.333  1833   Loop: MAIN [{matmult.f90} [71,9]-[74,14]]
1729975.333  107    Loop: MAIN [{matmult.f90} [77,9]-[84,14]]
1729975.333  30     INITIALIZE
1729975.333  14.25  MPI_Comm_rank()
1729975.333  1      MPI_Comm_size()
```
Solution: Generating a loop level profile

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

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

% paraprof app.ppk
Goal: What execution rate do my application loops get in mflops?

Flat profile with PAPI_FP_INS/OPS and time (-papi) with loop instrumentation:

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

Usage Scenarios: MFlops in Loops
Generate a PAPI profile with 2 or more counters

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

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

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

% qsub run.job
% paraprof --pack app.ppk
    Move the app.ppk file to your desktop.
% paraprof app.ppk
```
Generate a Callpath Profile
Callpath Profile

- Generates program callgraph

![Callpath Profile Diagram]
Generate a Callpath Profile

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

% qsub run.job
% paraprof --pack app.ppk
  Move the app.ppk file to your desktop.
% paraprof app.ppk
(Windows -> Thread -> Call Graph)
### Usage Scenario: 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>

---

6th VI-HPS Tuning Workshop: The TAU Performance System
Detect Memory Leaks

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

% qsub run.job
% paraprof --pack app.ppk
  Move the app.ppk file to your desktop.
% paraprof app.ppk
(Windows -> Thread -> Context Event Window -> Select thread -> select... expand tree)
(Windows -> Thread -> User Event Bar Chart -> right click LEAK
 -> Show User Event Bar Chart)

NOTE: setenv TAU_TRACK_HEAP 1 and setenv TAU_TRACK_HEADROOM 1 may be used to track
heap and headroom utilization at the entry and exit of each routine.
TAU_CALLPATH_DEPTH=1 shows just the routine name, and 0 shows just one event for the
entire program.
### Interval Events, Atomic 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 usec/call</th>
<th>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>1105659</td>
<td>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</td>
<td>MPI_Init()</td>
</tr>
<tr>
<td>5.9</td>
<td>0.879</td>
<td>65</td>
<td>40</td>
<td>320</td>
<td>1637</td>
<td>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</td>
<td>MPI_Barrier()</td>
</tr>
<tr>
<td>1.2</td>
<td>13</td>
<td>13</td>
<td>120</td>
<td>0</td>
<td>111</td>
<td>MPI_Recv()</td>
</tr>
<tr>
<td>0.8</td>
<td>9</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>9328</td>
<td>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</td>
<td>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</td>
<td>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</td>
<td>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</td>
<td>MPI_Comm_rank()</td>
</tr>
</tbody>
</table>

#### Atomic Events

- % setenv `TAU_CALLPATH_DEPTH` 0
- % setenv `TAU_TRACK_HEAP` 1

---

**VI-HPS**

6th VI-HPS Tuning Workshop: The TAU Performance System
Atomic Events, Context Events

<table>
<thead>
<tr>
<th>%Time</th>
<th>Exclusive msec</th>
<th>Inclusive total msec</th>
<th>#Call</th>
<th>#Subs</th>
<th>Inclusive Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0</td>
<td>0.253</td>
<td>1.106</td>
<td>1</td>
<td>44</td>
<td>1106701 int main(int, char **) C</td>
</tr>
<tr>
<td>93.2</td>
<td>1.031</td>
<td>1.031</td>
<td>1</td>
<td>0</td>
<td>1031311 MPI_Init()</td>
</tr>
<tr>
<td>6.0</td>
<td>1</td>
<td>66</td>
<td>40</td>
<td>320</td>
<td>1650 void func(int, Int) C</td>
</tr>
<tr>
<td>5.7</td>
<td>63</td>
<td>63</td>
<td>40</td>
<td>0</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>9119 MPI_Finalize()</td>
</tr>
<tr>
<td>0.1</td>
<td>1</td>
<td>120</td>
<td>0</td>
<td>0</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>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>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>1 MPI_Comm_size()</td>
</tr>
<tr>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0 MPI_Comm_rank()</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>NumSamples</th>
<th>MaxValue</th>
<th>MinValue</th>
<th>MeanValue</th>
<th>Std. Dev.</th>
<th>Event Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>5.139E+04</td>
<td>44.39</td>
<td>3.091E+04</td>
<td>1.234E+04</td>
<td>Heap Memory Used (KB) : Entry</td>
</tr>
<tr>
<td>40</td>
<td>5.139E+04</td>
<td>3097</td>
<td>3.114E+04</td>
<td>1.227E+04</td>
<td>Heap Memory Used (KB) : Entry : MPI_BARRIER()</td>
</tr>
<tr>
<td>40</td>
<td>5.139E+04</td>
<td>1.13E+04</td>
<td>3.134E+04</td>
<td>1.187E+04</td>
<td>Heap Memory Used (KB) : Entry : MPI_Bcast()</td>
</tr>
<tr>
<td>1</td>
<td>2067</td>
<td>2067</td>
<td>2067</td>
<td>0</td>
<td>Heap Memory Used (KB) : Entry : MPI_Comm_size()</td>
</tr>
<tr>
<td>1</td>
<td>2066</td>
<td>2066</td>
<td>2066</td>
<td>0</td>
<td>Heap Memory Used (KB) : Entry : MPI_Comm_rank()</td>
</tr>
<tr>
<td>1</td>
<td>5.139E+04</td>
<td>5.139E+04</td>
<td>5.139E+04</td>
<td>5.139E+04</td>
<td>Heap Memory Used (KB) : Entry : MPI_Finalize()</td>
</tr>
<tr>
<td>1</td>
<td>57.56</td>
<td>57.56</td>
<td>57.56</td>
<td>0</td>
<td>Heap Memory Used (KB) : Entry : MPI_Init()</td>
</tr>
<tr>
<td>120</td>
<td>5.139E+04</td>
<td>1.13E+04</td>
<td>3.134E+04</td>
<td>1.187E+04</td>
<td>Heap Memory Used (KB) : Entry : MPI_Recv()</td>
</tr>
<tr>
<td>120</td>
<td>5.139E+04</td>
<td>1.129E+04</td>
<td>3.134E+04</td>
<td>1.187E+04</td>
<td>Heap Memory Used (KB) : Entry : MPI_Send()</td>
</tr>
<tr>
<td>1</td>
<td>44.39</td>
<td>44.39</td>
<td>44.39</td>
<td>0</td>
<td>Heap Memory Used (KB) : Entry : int main(int, char **) C</td>
</tr>
<tr>
<td>40</td>
<td>5.036E+04</td>
<td>2068</td>
<td>3.011E+04</td>
<td>1.227E+04</td>
<td>Heap Memory Used (KB) : Entry : void func(int, int) C</td>
</tr>
</tbody>
</table>

VI-HPS

6th VI-HPS Tuning Workshop: The TAU Performance System
Context Events (default)

% setenv TAU_CALLPATH_DEPTH 2
% setenv TAU_TRACK_HEAP 1

VI-HPS

6th VI-HPS Tuning Workshop: The TAU Performance System
Using tau_exec

```
> cd ~/workshop-point/matmult
> mpi90f 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_MEMORYLeaks 1
> setenv TAU_CALLPATH_DEPTH 100
> mpirun -np 4 tau_exec -io -memory ./matmult
> paraprof
> # Right click on a given rank (e.g. "node 2") and choose "Show Context Event
> # Window" and expand the ".TAU Application" node to see the callpath
> # To use a different configuration (e.g., Makefile.tau-papi-mpi-pdt)
> setenv TAU_METRICS TIME:PAPI_FP_INS:PAPI_L1_DCM
> mpirun -np 4 tau_exec -io -memory -T papi,mpi.pdt ./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
```

---

6th VI-HPS Tuning Workshop: The TAU Performance System
# 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_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_SYNCHRONIZE_CLOCKS</td>
<td>1</td>
<td>Synchronize clocks across nodes to correct timestamps in traces</td>
</tr>
<tr>
<td>TAU_COMM_MATRIX</td>
<td>0</td>
<td>Setting to 1 generates communication matrix display using context events</td>
</tr>
<tr>
<td>TAU_THROTTLE</td>
<td>1</td>
<td>Setting to 0 turns off throttling. Enabled by default to remove instrumentation in lightweight routines that are called frequently</td>
</tr>
<tr>
<td>TAU_THROTTLE_NUMCALLS</td>
<td>100000</td>
<td>Specifies the number of calls before testing for throttling</td>
</tr>
<tr>
<td>TAU_THROTTLE_PERCALL</td>
<td>10</td>
<td>Specifies value in microseconds. Throttle a routine if it is called over 100000 times and takes less than 10 usec of inclusive time per call</td>
</tr>
<tr>
<td>TAU_COMPENSATE</td>
<td>0</td>
<td>Setting to 1 enables runtime compensation of instrumentation overhead</td>
</tr>
<tr>
<td>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 separted list generates other metrics. (e.g., TIME:linuxtimers:PAPI_FP_OPS:PAPI_NATIVE_&lt;event&gt;)</td>
</tr>
</tbody>
</table>
Compile-Time Environment 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 (cparse). Typically
    $(TAU_MPI_INCLUDE) $(TAU_INCLUDE) $(TAU_DEFS)
  - optPdtCxxOpts=""
    Options for C++ parser in PDT (cxxparse). Typically
    $(TAU_MPI_INCLUDE) $(TAU_INCLUDE) $(TAU_DEFS)
Measuring Performance of PGI Accelerator Code

<table>
<thead>
<tr>
<th>Name</th>
<th>Exclusive TIME</th>
<th>Inclusive TIME</th>
<th>Calls</th>
<th>Child Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>__pgi_cu_launch_multiply_matrices (pgi_kernel_7,gx=32,gy=32,gz=1,bx=16,by=16,bz=1) [ymm2.f90][15]</td>
<td>10.901</td>
<td>10.901</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>__pgi_cu_init_multiply_matrices [ymm2.f90][9]</td>
<td>3.912</td>
<td>3.912</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>__pgi_cu_download2_multiply_matrices var=a [ymm2.f90][20]</td>
<td>0.514</td>
<td>0.514</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>__pgi_cu_upload2_multiply_matrices var=b [ymm2.f90][9]</td>
<td>0.252</td>
<td>0.252</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>__pgi_cu_upload2_multiply_matrices var=c [ymm2.f90][9]</td>
<td>0.252</td>
<td>0.252</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>__pgi_cu_free_multiply_matrices [ymm2.f90]</td>
<td>0.125</td>
<td>16.021</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>__pgi_cu_allocate_multiply_matrices [ymm2.f90][5,0]</td>
<td>0.125</td>
<td>16.021</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>__pgi_cu_launch_multiply_matrices (pgi_kernel_2,gx=32,gy=32,gz=1,bx=16,by=16,bz=1) [ymm2.f90][11]</td>
<td>0.023</td>
<td>0.023</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>__pgi_cu_allocate_multiply_matrices [ymm2.f90]</td>
<td>0.019</td>
<td>0.019</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>__pgi_cu_allocate multiply_matrices [ymm2.f90][9]</td>
<td>0.003</td>
<td>15.895</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>__pgi_cu_free_multiply_matrices [ymm2.f90]</td>
<td>0.001</td>
<td>15.893</td>
<td>5</td>
<td>85</td>
</tr>
<tr>
<td>__pgi_cu_module_multiply_matrices [ymm2.f90][9]</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>__pgi_cu_module_function_multiply_matrices [ymm2.f90][11]</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>__pgi_cu_paramset_multiply_matrices [ymm2.f90]</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
</tr>
</tbody>
</table>
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

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

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

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

- Goal: How does my application scale? What bottlenecks at what cpu counts?
- Load profiles in PerfDMF database and examine with PerfExplorer
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()
- FcHdf5Tmpl <DATATYPE>::writeDataSet
- void FcDataAssimilatorUfiles::parseUfiles(const std::vector<... other
- void FcUpdaterComponent::dumpToFile(const std::string & con...
Evaluate Scalability using PerfExplorer Charts

% setenv TAU_MAKEFILE /opt/tau-2.19.1/x86_64
   /lib/Makefile.tau-mpi-pdt
% set path=/opt/tau-2.19.1/x86_64/bin $path
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% qsub run1p.job
% paraprof --pack 1p.ppk
% qsub run2p.job …
% paraprof --pack 2p.ppk … and so on.

On your client:
% perfdfm_configure
(Choose derby, blank user/passwd, yes to save passwd, defaults)
% perfexplorer_configure
(Yes to load schema, defaults)
% paraprof
(load each trial: DB -> Add Trial -> Type (Paraprof Packed Profile) -> OK, OR use perfdfm_loadtrial on the commandline)
% perfexplorer
(Charts -> Speedup)
Goal: What is the volume of inter-process communication? Along which calling path?
Evaluate Scalability using PerfExplorer Charts

% setenv TAU_MAKEFILE
   $TAU/Makefile.tau-mpi-pdt
% set 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)
% setenv TAU_COMM_MATRIX 1

% qsub run.job (setting the environment variables)

% paraprof
(Windows -> Communication Matrix)
(Windows -> 3D Communication Matrix)
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 → Program execution

Performance data → Program output
TAU and Eclipse

PerfDMF

![Image of Eclipse IDE with Fortran code for matrix multiplication]

```
! matmult.f90 - simple matrix multiply implementation

subroutine initialize(a, b, n)
  double precision a(n,n)
  b(n,n)
  integer n

! first initialize the A matrix
  do i = 1, n
    do j = 1, n
      a(i,j) = i
    end do
  end do

! then initialize the B matrix
  do i = 1, n
    do j = 1, n
      b(i,j) = i
    end do
  end do

end subroutine initialize

subroutine multiply_matrices(answer, buffer, b, nsize)
  double precision buffer(nsize)
  answer(nsize)
  b(nsize, nsize)
  integer i, j

! multiply the row with the column
```
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.
For more information

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