Performance and Debugging Tools

Performance Measurement and Analysis:
- OpenSpeedShop
- HPCToolkit
- Vampir
- Scalasca
- Periscope
- mpiP
- Paraver
- PerfExpert
- TAU

Modeling and prediction
- Prophesy
- MuMMI

Debugging
- Stat

Autotuning Frameworks
- Active Harmony
### Performance Tools Matrix

<table>
<thead>
<tr>
<th>TOOL</th>
<th>Profiling</th>
<th>Tracing</th>
<th>Instrumentation</th>
<th>Sampling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalasca</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>HPCToolkit</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Vampir</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Open</td>
<td>SpeedShop</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Periscope</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>mpiP</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Paraver</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>TAU</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

CIS 410/510: Parallel Computing, University of Oregon, Spring 2016

Lecture 13 – Parallel Performance Tools
Open|SpeedShop

Krell Institute (USA)

http://www.openspeedshop.org

Open|SpeedShop
Open|SpeedShop Tool Set

- Open Source Performance Analysis Tool Framework
  - Most common performance analysis steps *all in one tool*
  - Combines *tracing* and *sampling* techniques
  - *Extensible* by plugins for data collection and representation
  - Gathers and displays several types of performance information

- Flexible and Easy to use
  - User access through: *GUI, Command Line, Python Scripting, convenience scripts*

- Scalable Data Collection
  - Instrumentation of *unmodified application binaries*
  - New option for *hierarchical online data aggregation*

- Supports a wide range of systems
  - Extensively used and tested on a variety of *Linux clusters*
  - *Cray XT/XE/XK* and *Blue Gene L/P/Q* support
OpenSpeedShop Workflow

osspcsamp "srun -n4 -N1 smg2000 -n 65 65 65"

MPI Application

O|SS

Post-mortem
Central Concept: Experiments

- Users pick experiments:
  - What to measure and from which sources?
  - How to select, view, and analyze the resulting data?

- Two main classes:
  - Statistical Sampling
    - periodically interrupt execution and record location
    - useful to get an overview
    - low and uniform overhead
  - Event Tracing (DyninstAPI)
    - gather and store individual application events
    - provides detailed per event information
    - can lead to huge data volumes

- O|SS can be extended with additional experiments
Performance Analysis in Parallel

- How to deal with concurrency?
  - Any experiment can be applied to parallel application
    - Important step: aggregation or selection of data
  - Special experiments targeting parallelism/synchronization

- O|SS supports MPI and threaded codes
  - Automatically applied to all tasks/threads
  - Default views aggregate across all tasks/threads
  - Data from individual tasks/threads available
  - Thread support (incl. OpenMP) based on POSIX threads

- Specific parallel experiments (e.g., MPI)
  - Wraps MPI calls and reports
    - MPI routine time
    - MPI routine parameter information
  - The mpit experiment also store function arguments and return code for each call
HPCToolkit

John Mellor-Crummey
Rice University (USA)

http://hpctoolkit.org
HPCToolkit

- Integrated suite of tools for measurement and analysis of program performance
- Works with multilingual, fully optimized applications that are statically or dynamically linked
- Sampling-based measurement methodology
- Serial, multiprocess, multithread applications
• Performance Analysis through callpath sampling
  – Designed for low overhead
  – Hot path analysis
  – Recovery of program structure from binary
HPCToolkit DESIGN PRINCIPLES

- Employ binary-level measurement and analysis
  - observe *fully optimized*, dynamically linked executions
  - support *multi-lingual codes* with external binary-only libraries
- Use sampling-based measurement (avoid instrumentation)
  - controllable overhead
  - minimize systematic error and avoid blind spots
  - enable data collection for large-scale parallelism
- Collect and correlate multiple derived performance metrics
  - diagnosis typically requires more than one species of metric
- Associate metrics with both static and dynamic context
  - loop nests, procedures, inlined code, calling context
- Support top-down performance analysis
  - natural approach that minimizes burden on developers
**HPCToolkit Workflow**

1. **app. source** → **optimized binary**
   - **compile & link**

2. **profile execution**
   - **[hpcrun]**

3. **binary analysis**
   - **[hpcstruct]**

4. **call stack profile**

5. **program structure**

6. **interpret profile correlate w/ source**
   - **[hpcprof/hpcprof-mpi]**

7. **presentation**
   - **[hpcviewer/hpctraceviewer]**

- **database**
HPCToolkit Workflow

- For dynamically-linked executables on stock Linux
  - compile and link as you usually do: nothing special needed
- For statically-linked executables (e.g. for Blue Gene, Cray)
  - add monitoring by using `hpclink` as prefix to your link line
    - uses “linker wrapping” to catch “control” operations
      - process and thread creation, finalization, signals, ...

For dynamically-linked executables on stock Linux:
- compile and link as you usually do: nothing special needed

For statically-linked executables (e.g. for Blue Gene, Cray):
- add monitoring by using `hpclink` as prefix to your link line
  - uses “linker wrapping” to catch “control” operations
    - process and thread creation, finalization, signals, ...

- presentation
  - `hpcviewer/hpctraceviewer`

- interpret profile correlate w/ source
  - `hpcprof/hpcprof-mpi`

- call stack profile
- program structure
- optimized binary
- compiled & link
- app. source
- binary analysis
  - `hpcstruct`
- profile execution
  - `[hpcrun]`
- call stack profile
- program structure
- interpreted profile
- correlate w/ source
  - `hpcprof/hpcprof-mpi`
- database
- presentation
  - `hpcviewer/hpctraceviewer`
Measure execution unobtrusively

- launch optimized application binaries
  - dynamically-linked applications: launch with `hpcrun` to measure
  - statically-linked applications: measurement library added at link time
    - control with environment variable settings
- collect statistical call path profiles of events of interest

- interpret profile correlate w/ source
  - database
  - presentation [hpcviewer/hpctraceviewer]

- compile & link
  - app. source
  - optimized binary
  - binary analysis [hpcstruct]
  - profile execution [hpcrun]
  - call stack profile
  - program structure
HPCToolkit Workflow

- Analyze binary with **hpcstruct**: recover program structure
  - analyze machine code, line map, debugging information
  - extract loop nesting & identify inlined procedures
  - map transformed loops and procedures to source

presentation
[hpcviewer/hpctraceviewer]

interpret profile correlate w/ source
[hpcprof/hpcprof-mpi]
HPCToolkit Workflow

- Combine multiple profiles
  - multiple threads; multiple processes; multiple executions
- Correlate metrics to static & dynamic program structure

Presentation:
- [hpcviewer/hpctraceviewer]

Database:
- [hpcprof/hpcprof-mpi]

Correlate profile w/ source
Presentation

- explore performance data from multiple perspectives
  - rank order by metrics to focus on what’s important
  - compute derived metrics to help gain insight
    - e.g. scalability losses, waste, CPI, bandwidth
- graph thread-level metrics for contexts
- explore evolution of behavior over time
Analyzing results with hpcviewer

- **Callpath to hotspot**
- **Source pane**
- **View control**
- **Metric display**
- **Navigation pane**
- **Metric pane**

**Costs for**
- inlined procedures
- loops
- function calls in full context

**Image by John Mellor-Crummey**
Vampir

Wolfgang Nagel
ZIH, Technische Universität Dresden (Germany)

http://www.vampir.eu
Mission

- Visualization of dynamics of complex parallel processes
- Requires two components
  - Monitor/Collector (Score-P)
  - Charts/Browser (Vampir)

Typical questions that Vampir helps to answer:

- What happens in my application execution during a given time in a given process or thread?
- How do the communication patterns of my application execute on a real system?
- Are there any imbalances in computation, I/O or memory usage and how do they affect the parallel execution of my application?
Event Trace Visualization with Vampir

- Alternative and supplement to automatic analysis
- Show dynamic run-time behavior graphically at any level of detail
- Provide statistics and performance metrics

Timeline charts
- Show application activities and communication along a time axis

Summary charts
- Provide quantitative results for the currently selected time interval
Vampir – Visualization Modes (1)

Directly on front end or local machine

```vampir
% vampir
```
Vampir – Visualization Modes (2)

On local machine with remote VampirServer

% vampirserver start -n 12

% vampir

VampirServer

Many-Core Program

Score-P

MPI parallel application

LAN/WAN

Large Trace File (stays on remote machine)

Trace File (OTF2)

Many-Core Program

LAN/WAN

Large Trace File (stays on remote machine)

MPI parallel application

CIS 410/510: Parallel Computing, University of Oregon, Spring 2016

Lecture 13 – Parallel Performance Tools
Main Displays of Vampir

- Timeline Charts:
  - Master Timeline
  - Process Timeline
  - Counter Data Timeline
  - Performance Radar

- Summary Charts:
  - Function Summary
  - Message Summary
  - Process Summary
  - Communication Matrix View
Visualization of the NPB-MZ-MPI / BT trace

```
vampir scorep_bt-mz_B_4x4_trace
```
Visualization of the NPB-MZ-MPI / BT trace

Master Timeline

Detailed information about functions, communication and synchronization events for collection of processes.
Visualization of the NPB-MZ-MPI / BT trace

Process Timeline

Detailed information about different levels of function calls in a stacked bar chart for an individual process.
Visualization of the NPB-MZ-MPI / BT trace

Typical program phases

- Initialisation Phase
- Computation Phase
Visualization of the NPB-MZ-MPI / BT trace

Counter Data Timeline

Details:
- Detailed counter information over time for an individual process.
Visualization of the NPB-MZ-MPI / BT trace

Performance Radar

Detailed counter information over time for a collection of processes.
Visualization of the NPB-MZ-MPI / BT trace

Zoom in: Computation Phase

MPI communication results in lower floating point operations.
Vampir Summary

- Vampir & VampirServer
  - Interactive trace visualization and analysis
  - Intuitive browsing and zooming
  - Scalable to large trace data sizes (20 TByte)
  - Scalable to high parallelism (200000 processes)
- Vampir for Linux, Windows and Mac OS X
- Vampir does neither solve your problems automatically nor point you directly at them
- Rather it gives you FULL insight into the execution of your application
Scalasca

Bernd Mohr and Felix Wolf

Jülich Supercomputing Centre (Germany)
German Research School for Simulation Sciences

http://www.scalasca.org
Scalable parallel performance-analysis toolset
- Focus on communication and synchronization

Integrated performance analysis process
- Callpath profiling
  - performance overview on callpath level
- Event tracing
  - in-depth study of application behavior

Supported programming models
- MPI-1, MPI-2 one-sided communication
- OpenMP (basic features)

Available for all major HPC platforms
Scalasca Project: Objective

- Development of a scalable performance analysis toolset for most popular parallel programming paradigms
- Specifically targeting large-scale parallel applications
  - 100,000 – 1,000,000 processes / thread
  - IBM BlueGene or Cray XT systems
- Latest release:
  - Scalasca v2.0 with Score-P support (August 2013)
Scalasca: Automatic Trace Analysis

- **Idea**
  - Automatic search for patterns of inefficient behavior
  - Classification of behavior and quantification of significance

- Guaranteed to cover the entire event trace
- Quicker than manual/visual trace analysis
- Parallel replay analysis online
Scalasca Workflow

Measurement library

Instr. target application

Scalasca trace analysis

Optimized measurement configuration

Summary report

Report manipulation

Local event traces

Parallel wait-state search

Wait-state report

Which problem?

Where in the program?

Which process?

Instrumented executable

Instrumenter compiler / linker

Source modules

HWC

CIS 410/510: Parallel Computing, University of Oregon, Spring 2016 Lecture 13 – Parallel Performance Tools
Callpath Profile: Computation

- Execution time excl. MPI comm
- Just 30% of simulation
- Widely spread in code
Callpath Profile: P2P Messaging

MPI point-to-point communication time

P2P comm 66% of simulation

Primarily in scatter & gather
Callpath Profile: P2P Synchronization

Masses of P2P sync. operations

Processes all equally responsible

Point-to-point msgs w/o data
Scalasca Approach to Performance Dynamics

Overview

• Capture overview of performance dynamics via time-series profiling
  • Time and count-based metrics

Focus

• Identify pivotal iterations - if reproducible

In-depth analysis

• In-depth analysis of these iterations via tracing
  • Analysis of wait-state formation
  • Critical-path analysis
  • Tracing restricted to iterations of interest

New
TAU Performance System®

- Tuning and Analysis Utilities (20+ year project)
- Performance problem solving framework for HPC
  - Integrated, scalable, flexible, portable
  - Target all parallel programming / execution paradigms
- Integrated performance toolkit
  - Multi-level performance instrumentation
  - Flexible and configurable performance measurement
  - Widely-ported performance profiling / tracing system
  - Performance data management and data mining
  - Open source (BSD-style license)
- Broad use in complex software, systems, applications

http://tau.uoregon.edu
TAU History

1992-1995: Malony and Mohr work with Gannon on DARPA pC++ project work. TAU is born. [parallel profiling, tracing, performance extrapolation]

1995-1998: Shende works on Ph.D. research on performance mapping. TAU v1.0 released. [multiple languages, source analysis, automatic instrumentation]

1998-2001: Significant effort in Fortran analysis and instrumentation, work with Mohr on POMP, Kojak tracing integration, focus on automated performance analysis. [performance diagnosis, source analysis, instrumentation]

2002-2005: Focus on profiling analysis tools, measurement scalability, and perturbation compensation. [analysis, scalability, perturbation analysis, applications]


2008-2011: Add performance database support, data mining, and rule-based analysis. Develop measurement/analysis for heterogeneous systems. Core measurement infrastructure integration (Score-P). [database, data mining, expert system, heterogeneous measurement, infrastructure integration]

2012-present: Focus on exascale systems. Improve scalability. Add hybrid measurement support, extend heterogeneous and mixed-mode, develop user-level threading. Apply to petascale / exascale applications. [scale, autotuning, user-level]
Lecture 13 – Parallel Performance Tools

- Node: physically distinct shared memory machine
  - Message passing node interconnection network
- Context: distinct virtual memory space within node
- Thread: execution threads (user/system) in context

General Target Computation Model in TAU

Interconnection Network

* Inter-node message communication

Node: physically distinct shared memory machine

- Message passing node interconnection network

Context: distinct virtual memory space within node

Thread: execution threads (user/system) in context
TAU Architecture

- TAU is a parallel performance framework and toolkit
- Software architecture provides separation of concerns
  - Instrumentation | Measurement | Analysis

**TAU Architecture**

- **Instrumentation**
  - Source
    - C, C++, Fortran
    - Python, UPC, Java
    - Robust parsers (PDT)
  - Wrapping
    - Interposition (MPI)
    - Wrapper generation
  - Linking
    - Static, dynamic
    - Preloading
  - Executable
    - Dynamic (Dyninst)
    - Binary (Dyninst, MAGAO)

- **Measurement**
  - Events
    - static/dynamic
    - routine, basic block, loop
    - threading, communication
    - heterogeneous
  - Profiling
    - flat, callpath, phase, parameter, snapshot
    - probe, sampling, hybrid
  - Metadata
    - system, user-defined

- **Analysis**
  - Profiles
    - ParaProf parallel profile analyzer / visualizer
    - TAU/db parallel profile database
    - PerfExplorer parallel profile data mining
  - Tracing
    - TAU trace translation
      - OTF, SLCG-2
    - Trace analysis / visualizer
      - Vampir, Jumpshot
  - Online
    - event unification
    - statistics calculation
TAU Observation Methodology and Workflow

- TAU’s (primary) methodology for parallel performance observation is based on the insertion of measurement probes into application, library, and runtime system
  - Code is instrumented to make visible certain events
  - Performance measurements occur when events are triggered
  - Known as probe-based (direct) measurement

- Performance experimentation workflow
  - Instrument application and other code components
  - Link / load TAU measurement library
  - Execute program to gather performance data
  - Analysis performance data with respect to events
  - Analyze multiple performance experiments

- Extended TAU’s methodology and workflow to support sampling-based techniques
TAU Components

- Instrumentation
  - Fortran, C, C++, OpenMP, Python, Java, UPC, Chapel
  - Source, compiler, library wrapping, binary rewriting
  - Automatic instrumentation

- Measurement
  - Internode: MPI, OpenSHMEM, ARMCI, PGAS, DMAPP
  - Intranode: Pthreads, OpenMP, hybrid, …
  - Heterogeneous: GPU, MIC, CUDA, OpenCL, OpenACC, …
  - Performance data (timing, counters) and metadata
  - Parallel profiling and tracing (with Score-P integration)

- Analysis
  - Parallel profile analysis and visualization (ParaProf)
  - Performance data mining / machine learning (PerfExplorer)
  - Performance database technology (TAUdb)
  - Empirical autotuning
TAU Instrumentation Approach

- Direct and indirect performance instrumentation
  - Direct instrumentation of program (system) code (probes)
  - Indirect support via sampling or interrupts
- Support for standard program code events
  - Routines, classes and templates
  - Statement-level blocks, loops
  - Interval events (start/stop)
- Support for user-defined events
  - Interval events specified by user
  - Atomic events (statistical measurement at a single point)
  - Context events (atomic events with calling path context)
- Provides static events and dynamic events
- Instrumentation optimization
TAU Instrumentation Mechanisms

- Source code
  - Manual (TAU API, TAU component API)
  - Automatic (robust)
    - C, C++, F77/90/95, OpenMP (POMP/OPARI), UPC
  - Compiler (GNU, IBM, NAG, Intel, PGI, Pathscale, Cray, …)

- Object code (library-level)
  - Statically- and dynamically-linked wrapper libraries
    - MPI, I/O, memory, …
  - Powerful library wrapping of external libraries without source

- Executable code / runtime
  - Runtime preloading and interception of library calls
  - Binary instrumentation (Dyninst, MAQAO, PEBIL)
  - Dynamic instrumentation (Dyninst)
  - OpenMP (runtime API, CollectorAPI, GOMP, OMPT)

- Virtual machine, interpreter, and OS instrumentation
Instrumentation for Wrapping External Libraries

- **Preprocessor substitution**
  - Header redefines a routine with macros (only C and C++)
  - Tool-defined header file with same name takes precedence
  - Original routine substituted by preprocessor callsite

- **Preloading a library at runtime**
  - Library preloaded in the address space of executing application intercepts calls from a given library
  - Tool wrapper library defines routine, gets address of global symbol (*dlsym*), internally calls measured routine

- **Linker-based substitution**
  - Wrapper library defines wrapper interface
    - wrapper interface then which calls routine
  - Linker is passed option to substitute all references from applications object code with tool wrappers
**Automatic Source-level / Wrapper Instrumentation**

- PDT source analyzer
- Parsed program
- Application source
- tau_instrumentor
- Instrumented source
- tau_wrap

**Instrumentation specification file**

```
BEGIN_EXCLUDE_LIST
Foo
Bar
D#EMM
END EXCLUDE LIST
```

```
BEGIN_FILE_EXCLUDE_LIST
f*.f90
Foo?.cpp
END_FILE_EXCLUDE_LIST
BEGIN_FILE_INCLUDE_LIST
main.cpp
foo.f90
END_FILE_INCLUDE_LIST
```
MPI Wrapper Interposition Library

- Uses standard MPI Profiling Interface
  - Provides name shifted interface (weak bindings)
    - \texttt{MPI\_Send = PMPI\_Send}
- Create TAU instrumented MPI library
  - Interpose between MPI and TAU
    - \texttt{-lmpi replaced by -lTauMpi -lpmpi -lmpi}
  - No change to the source code, just re-link application!
- Can we interpose MPI for compiled applications?
  - Avoid re-compilation or re-linking
  - Requires shared library MPI
    - \texttt{uses LD\_PRELOAD for Linux}
  - Approach will work with other shared libraries (see later slide)
  - Use TAU \texttt{tau\_exec} (see later slide)
    - \% mpirun -np 4 tau\_exec a.out
TAU has been a long-time user of DyninstAPI

Using DyninstAPI’s binary re-writing capabilities, created a binary re-writer tool for TAU (\texttt{tau\_run})

- Supports TAU's performance instrumentation
- Works with TAU instrumentation selection
  - files and routines based on exclude/include lists
- TAU’s measurement library (DSO) is loaded by \texttt{tau\_run}
- Runtime (pre-execution) and binary re-writing supported

Simplifies code instrumentation and usage greatly!

\%
\texttt{tau\_run a.out \textendash o a.inst}
\%
\texttt{mpirun \textendash np 4 ./a.inst}

Support PEBIL and MAQAO binary instrumentation
Library Interposition

- Simplify TAU usage to assess performance properties
  - Application, I/O, memory, communication
- Designed a new tool that leverages runtime instrumentation by pre-loading measurement libraries
- Works on dynamic executables (default under Linux)
- Substitutes routines (e.g., I/O, MPI, memory allocation/deallocation) with instrumented calls
  - Interval events (e.g., time spent in write())
  - Atomic events (e.g., how much memory was allocated)
Library Wrapping – tau_gen_wrapper

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

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

- Add to TAU_OPTIONS environment variable:
  
  -optTauWrapFile=<wrapperdir>/link_options.tau

- Wrapped library
  - Redirects references at routine callsite to a wrapper call
  - Wrapper internally calls the original
  - Wrapper has TAU measurement code
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)
- Metadata (hardware/system, application, …)
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
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

- Parameter and context profiling
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 (TAUdb)
- Performance data mining (PerfExplorer)
- Parallel trace analysis
  - Translation to VTF (V3.0), EPILOG, OTF formats
  - Integration with Vampir / Vampir Server (TU Dresden)
- Integration with CUBE browser (Scalasca, UTK / FZJ)
- Scalable runtime fault isolation with callstack debugging
- Efficient parallel runtime bounds checking
Profile Analysis Framework

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

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

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

Scripting Interface
- Jython
Performance Data Management (TAUdb)

- 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
  - Supported multiple profile formats: TAU, CUBE, gprof, mpiP, psrun, …
  - Supported DBMS: PostgreSQL, MySQL, Oracle, DB2, Derby, H2
Parallel performance profiles

- Timer and counter measurements with 5 dimensions
  - Physical location: process / thread
  - Static code location: function / loop / block / line
  - Dynamic location: current callpath and context (parameters)
  - Time context: iteration / snapshot / phase
  - Metric: time, HW counters, derived values

Measurement metadata

- Properties of the experiment
- Anything from name:value pairs to nested, structured data
- Single value for whole experiment or full context (tuple of thread, timer, iteration, timestamp)
TAUdb Tool Support

- ParaProf
  - Parallel profile analyzer
    - visual pprof
  - 2, 3+D visualizations
  - Single and comparative experiment analysis

- PerfExplorer
  - Data mining framework
    - Clustering, correlation
  - Multi-experiment analysis
  - Scripting engine
  - Expert system
ParaProf – Single Thread of Execution View

**Metrics: node, context, thread**

**8K processors**

- **MPI_Alltoall()**
- **MPI_Group_translate_ranks()**
- **RCFTY**
- **RCFTX**
- **MPI_BARRIER()**
- **BANBKS**
- **DENSITY**
- **MPI_Init()**
- **ADVDFACC**
- **TRANZ**
- **MPI_Comm_group()**
- **FOURIER**
- **INCOMPRESSIBLE**
- **DERIV**
- **FILTERZ**
- **MPI_Comm_size()**
- **SGS**
- **CCFTY**

**Software: Miranda**
- **hydrodynamics**
- **Fortran + MPI**
- **LLNL BG/L**
ParaProf – Full Profile / Comparative Views

Full profile
- threads
- events

bargraph view

landscape view

Comparative
- three event axes
- one event color
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
  - Performance model generation requires interpretation
- Build these capabilities into performance tools
  - Support broader experimentation methods and refinement
  - Access and correlate data from several sources
  - Automate performance data analysis / mining / learning
  - Include predictive features and experiment refinement
Role of Knowledge and Context in Analysis

You have to capture these...

Performance Knowledge

...to understand this

Performance Result
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
    - parametric, comparative, clustering, correlation, …
  - Integrate available statistics and data mining packages
    - Weka, R, Matlab / Octave
  - Apply data mining operations in interactive environment
  - Meta-analysis based on metadata collection in TAU
    - hardware/system, application, user, …
PerfExplorer Performance Data Mining

- Programmable, extensible framework to support workflow automation
- Rule-based inference for expert system analysis
Integration of XML metadata with parallel profile

Three ways to incorporate metadata

- Measured hardware/system information
  - 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

- TAUdb can load an XML file of additional metadata
  - Compiler flags, submission scripts, input files, …

Metadata can be imported from several sources
# Performance Experiment Metadata

## TAU: ParaProf Manager

- **Applications**
  - Default App
    - `papi_fp_ops`
    - `get_time_of_day`

- **Trial Field**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application ID</td>
<td>0</td>
</tr>
<tr>
<td>Experiment ID</td>
<td>0</td>
</tr>
<tr>
<td>Trial ID</td>
<td>0</td>
</tr>
<tr>
<td>CPU Cores</td>
<td>2</td>
</tr>
<tr>
<td>CPU MHz</td>
<td>2.992505</td>
</tr>
<tr>
<td>CPU Type</td>
<td>Intel(R) Xeon(R) CPU 5160 @ 3.00GHz</td>
</tr>
<tr>
<td>CPU Vendor</td>
<td>GenuineIntel</td>
</tr>
<tr>
<td>CWD</td>
<td><code>/home/amorris/tau2/examples/pdt_mpi/f90</code></td>
</tr>
<tr>
<td>Executable</td>
<td><code>/home/amorris/tau2/examples/pdt_mpi/f90</code></td>
</tr>
<tr>
<td>Hostname</td>
<td><code>demon.nic.uoregon.edu</code></td>
</tr>
<tr>
<td>Local Time</td>
<td><code>2007-07-04T04:21:14-07:00</code></td>
</tr>
<tr>
<td>MPI Processor Name</td>
<td><code>demon.nic.uoregon.edu</code></td>
</tr>
<tr>
<td>Memory Size</td>
<td>8161240 kB</td>
</tr>
<tr>
<td>Node Name</td>
<td><code>demon.nic.uoregon.edu</code></td>
</tr>
<tr>
<td>OS Machine</td>
<td><code>x86_64</code></td>
</tr>
<tr>
<td>OS Name</td>
<td>Linux</td>
</tr>
<tr>
<td>OS Release</td>
<td><code>2.6.9-42.0.3.EL perfctrmsmp</code></td>
</tr>
<tr>
<td>OS Version</td>
<td><code>#1 SMP Fri Nov 3 07:34:13 PST 2006</code></td>
</tr>
<tr>
<td>Starting Timestamp</td>
<td><code>118354807220996</code></td>
</tr>
<tr>
<td>TAU Architecture</td>
<td><code>x86_64</code></td>
</tr>
<tr>
<td>TAU Config</td>
<td><code>-papi=/usr/local/packages/papi-3.5.0-....</code></td>
</tr>
<tr>
<td>Timestamp</td>
<td><code>1183548074317538</code></td>
</tr>
<tr>
<td>uid</td>
<td>11395</td>
</tr>
<tr>
<td>username</td>
<td>amorris</td>
</tr>
</tbody>
</table>

---

*Multiple TAUdb databases*
Performance Visualization

- Large performance data presents interpretation challenges
- Visualization aids in data exploration and pattern analysis
  - 3D visualization can help in identifying relations between events/metrics
- Existing tools provide “canned” views
  - TAU provides a few
    - 2D: bargraph, histogram
    - 3D: full profile, correlation
- Developing new visualizations is a challenge
  - Strategy 1: Create new view for each problem
  - Strategy 2: Use external visualization environment
- Provide high-level support to use within existing framework
User defines visualization
- Based on performance data model

Specifies layout based on events, metrics, and metadata

UI provides control of data binding and visualization
Using Process Topology Metadata

- Inspired by the Scalasca CUBE topology display
- Each point represents a thread of execution (MPI process)
  - Positioned according to the Cartesian \((x,y,z,t)\) coordinates
- Color is determined by selected event/metric value
- Topology information can be recorded in TAU metadata
- ParaProf reads metadata to determine topology and create layout
- Sweep3D is a 3D neutron transport application
  - 16K run on BG/P
  - Color is exclusive time in the “sweep” function
Topology Control UI

- **Layout** tab allows customization of the position and visibility of data points
- Performance event/metric data used to define color and position is selected in the **Event** tab
- Additional rendering options, such as color scale and point size are available
- 4k-core S3D run on BG/P
Viewing Internal Structure

- Dense topologies can hide internal structure
- Restrict visibility by color value to expose performance patterns
- ParaProf visualization UI now allows for range filtering
  - Mid-level values can be excluded
  - Remaining points are:
    - high outliers (hotspots)
    - low outliers (underutilized nodes)
Heterogeneous Parallel Performance

- Heterogeneous parallel systems are very relevant today
  - Multi-CPU, multicore shared memory nodes
  - Manycore (throughput) accelerators
  - Cluster interconnection network

- Performance is the main driving concern
- Heterogeneity is an important path to extreme scale
- Heterogeneous software technology to get performance
  - More sophisticated parallel programming frameworks
  - Integrated parallel performance tools
    - support heterogeneous performance model and perspectives
Implications for Performance Tools

- Current status quo is somewhat comfortable
  - Mostly homogeneous parallel systems and software
  - Shared-memory multithreading – OpenMP
  - Distributed-memory message passing – MPI

- Parallel computational models are relatively stable
  - Corresponding performance models are tractable
  - Parallel performance tools can keep up and evolve

- Heterogeneity creates richer computational potential
  - Greater performance diversity and complexity

- Heterogeneous system environments

- Performance tools have to support richer computation models and more versatile performance perspectives

- Will utilize more sophisticated programming and runtime
Heterogeneous Performance Views

- Want to create performance views that capture heterogeneous concurrency and execution behavior
  - Reflect interactions between components
  - Capture performance semantics of computation
  - Assimilate performance for all execution paths

- Existing parallel performance tools are CPU (host)-centric
  - Event-based sampling
  - Direct measurement (instrumentation of events)

- What perspective does the host have of other components?
  - Determines semantics of the measurement data
  - Determines assumptions about behavior and interactions

- Performance views may have to work with reduced data
TAU for Heterogeneous Performance Analysis

- Extend TAU for heterogeneous performance analysis
- Integrate Host-GPU support in TAU measurement
  - Enable host-GPU measurement approach
    - target CUDA and NVIDIA GPUs
    - utilize CUPTI and PAPI CUDA
    - interface with compilers (PGI)
  - Provide both heterogeneous profiling and tracing
    - contextualization of asynchronous kernel invocation
- Additional support
  - TAU wrapping of libraries (tau_gen_wrapper)
  - Work with library preloading (tau_exec)

How TAU GPU Support Works

- Approach 1: CUDA library wrapping + CUDA events
  - CUDA driver or runtime API are intercepted

- Approach 2: CUDA Profiling Tools Interface (CUPTI)
  - Callback API – a callback is registered for every API call
  - Counter API -- GPU device counters (device-level)
  - Activity API – kernel and memory copy timing information

- With both of these methods, TAU records the activity that occurred on the GPU at a synchronization point

- Performance data is merged with TAU profile/trace
  - Divided between CPU and GPU
  - Each GPU stream/context presented as a separate thread
GPU Performance Tool Interoperability
Score-P Architecture

**Score-P Measurement Infrastructure**

1. Application (MPI, OpenMP, hybrid)
2. Instrumentation
3. Call-path profiles (CUBE4)
4. Event traces (OTF2)
5. Online interface

**Supplemental Instrumentation + Measurement Support**

- TAU
- Vampir
- Scalasca
- TAU
- Periscope

**Hardware Counter (PAPI)**

**TAU Adaptor**

**MPI Wrapper**

**Compilers**

- TAU instrumentor
- OPARI 2
- COBI

CIS 410/510: Parallel Computing, University of Oregon, Spring 2016
For More Information …

- **TAU Website**
  - [http://tau.uoregon.edu](http://tau.uoregon.edu)
    - Software
    - Release notes
    - Documentation

- **HPC Linux**
  - [http://www.hpclinux.com](http://www.hpclinux.com)
    - Parallel Tools “LiveDVD”
    - Boot up on your laptop or desktop
    - Includes TAU and variety of other packages
    - Include documentation and tutorial slides