Parallel Performance Tools

Parallel Computing
CIS 410/510
Department of Computer and Information Science
Performance and Debugging Tools

Performance Measurement and Analysis:
- Open|SpeedShop
- 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></td>
<td>X</td>
</tr>
<tr>
<td>mpiP</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Paraver</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>TAU</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
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

```bash
osspcsamp "srun -n4 -N1 smg2000 -n 65 65 65"
```
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
HPCToolkit

- Performance Analysis through callpath sampling
  - Designed for low overhead
  - Hot path analysis
  - Recovery of program structure from binary

![HPCToolkit Workflow](Image by John Mellor-Crummey)
**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

compile & link

app. source

optimized binary

profile execution
[hpcrun]

call stack profile

binary analysis
[hpcstruct]

program structure

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

presentation
[hpcviewer/ hpctraceviewer]

database
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, ...

**HPCToolkit Workflow**

- **compile & link**
- **optimized binary**
- **profile execution** [hpcrun]
- **call stack profile**
- **binary analysis** [hpcstruct]
- **program structure**
- **call stack analysis**
- **interpret profile correlate w/ source** [hpcprof/hpcprof-mpi]
- **presentation** [hpcviewer/hpctraceviewer]
- **database**
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
Analyse 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

**HPC Toolkit Workflow**
HPCToolkit Workflow

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

- Presentation
  - [hpcviewer/hpctraceviewer]

- Interpret profile correlate w/ source
  - [hpcprof/hpcprof-mpi]

Introduction to Parallel Computing, University of Oregon, IPCC
**HPCToolkit Workflow**

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

- **profile execution**
  - [hpcrun]

- **binary analysis**
  - [hpcstruct]

- **call stack profile**

- **program structure**

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

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

- **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

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

- Source code:
  - Source pane
  - View control
  - Metric display
  - Navigation pane

Callpath to hotspot

Image by John Mellor-Crummey

Introduction to Parallel Computing, University of Oregon, IPCC
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 – Visualization Modes (2)**

On local machine with remote VampirServer

```
% vampirserver start -n 12
% vampir
```

![Diagram showing the setup and visualization of performance tools like Vampir and Score-P for Many-Core Program with remote VampirServer and LAN/WAN communication.]
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

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

[Diagram showing a timeline with phases: Initialisation Phase and Computation Phase]
Visualization of the NPB-MZ-MPI / BT trace

Counter Data Timeline

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 → Optimized measurement configuration → Summary report → Report manipulation

Instr. target application

Scalasca trace analysis

Local event traces → Parallel wait-state search → Wait-state report

Optimized measurement configuration

Instrumenter compiler / linker

Instrumented executable

Source modules

Which problem? → Where in the program? → Which process?

Instrumentation library

HWC

Lecture 14 – Parallel Performance Tools

Introduction to Parallel Computing, University of Oregon, IPCC
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

Processes all equally responsible

Point-to-point msgs w/o data

Masses of P2P sync. operations
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]
General Target Computation Model in TAU

- 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 Diagram](image)

**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, MAQAQ)

**Measurement**
- **Events**
  - Static/dynamic
  - Routine, basic block, loop
  - Threading, communication
  - Heterogeneous

- **Profiling**
  - Flat, callpath, phase, parameter, snapshot
  - Probe, sampling, hybrid

**Analysis**
- **Profiles**
  - **ParaProf** parallel profile analyzer / visualizer
  - **TAUdb** parallel profile database
  - **PerfExplorer** parallel profile data mining

- **Tracing**
  - TAU / Scalasca tracing
  - Open Trace Format (OTF)

- **Tracing**
  - TAU trace translation
    - OTF, SLOG-2
  - Trace analysis / visualizer
    - Vampir, Jumpshot

- **Online**
  - Event unification
  - Statistics calculation

---

*Introduction to Parallel Computing, University of Oregon, IPCC*

*Lecture 14 – Parallel Performance Tools*
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 \texttt{(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
- Instrumentation specification file
- Application source
- tau_instrumentor
- tau_wrap
- Instrumented source

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)
    - $MPI_{Send} = PMPI_{Send}$
- Create TAU instrumented MPI library
  - Interpose between MPI and TAU
    - -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
    - uses LD_PRELOAD for Linux
  - Approach will work with other shared libraries (see later slide)
  - Use TAU tau_exec (see later slide)
    - `% mpirun -np 4 tau_exec a.out`
Binary Instrumentation

- TAU has been a long-time user of DyninstAPI
- Using DyninstAPI’s binary re-writing capabilities, created a binary re-writer tool for TAU (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 tau_run
  - Runtime (pre-execution) and binary re-writing supported
- Simplifies code instrumentation and usage greatly!
  \% tau_run a.out -o a.inst
  \% mpirun -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, shapshot) 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

![Diagram of Profile Analysis Framework]

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

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

**ParaProf**
- **Call Graphs**
- **Histograms**
- **Call Trees**
- **Bar Charts**
- **Comparative Displays**
- **Text Displays**

**Vis Package**
- **JOGL**
- **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
TAUdb Database Schema

- 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

node, context, thread

8K processors

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 Problems
- Application
- Machine

...to understand this

Performance Result

Context Metadata

- Source Code
- Build Environment
- Run Environment

Execution

Introduction to Parallel Computing, University of Oregon, IPCC
Score-P Architecture

**Score-P Architecture**

- **TAU**: TAU instrumentor + measurement support
- **Vampir**: Call-path profiles (CUBE4)
- **Scalasca**: Online interface
- **TAU**: Event traces (OTF2)
- **Periscope**: Hardware counter (PAPI)
- **Score-P measurement infrastructure**: TAU adaptor
- **Application (MPI, OpenMP, hybrid)**
- **Instrumentation**
- **MPI wrapper**
- **Compiler**
- **TAU instrumentor**
- **OPARI 2**
- **COBI**
For More Information …

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

- HPC Linux
  - 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