Logistics

- Problems with the AXIS GPUs
  - BIOS is not recognizing them
  - 64-bit base address register (BAR) issue
  - Still investigating …

- Will use Talapas instead
  - Slurm partition “cis631” containing a RACS dev node with 2 NVIDIA A100 GPUs
    - available 24x7
    - CIS 631 jobs will have dedicated access
  - Slurm reservations that reserve 2 general GPU nodes
    - total of 8 NVIDIA K80 GPUs
    - 25 hours each week dedicated for CIS 631 users
  - CIS 631 users may also access the general GPU node partitions
    - approximately 96 NVIDIA K80 GPUs
    - shared with research users
Parallel Performance, Engineering, and Tools

- I am interested in the nature of parallel performance
- *Performance* has been the fundamental driving concern for parallel computing and high-performance computing (HPC)
- Achieving performance is an *engineering process*
  - **Observation**: measure and characterize performance behavior
  - **Diagnosis**: identify and understand performance problems
  - **Tuning**: modify to run optimally on high-end machines
- How to make the process more effective and productive?
  - What is the nature of the performance problem solving?
  - What is the performance technology to be applied?
- Compelling reasons to build integrate performance tools
- Parallel systems evolution will drive changes in the technology and process and how they are applied in practice
Parallel Performance Engineering Process

- Follow an empirically-based approach
  observation ⇔ experimentation ⇔ diagnosis ⇔ tuning
- Performance technology developed for each level
Evolving Performance Challenges

- High-performance computing (HPC) has been driven by ever increasing powerful parallel hardware and sophisticated software capabilities.
- Performance technology developed for HPC systems must evolve to address the new requirements introduced for measurement and analysis.
- However, inherently greater machine and software complexity introduces new challenges to accurately observe, understand, and optimize performance.
- Integration of big data and AI/ML workloads poses new performance problems as well.
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>☒</td>
<td>☒</td>
<td>☒</td>
<td>☒</td>
</tr>
<tr>
<td>HPCToolkit</td>
<td>☒</td>
<td>☒</td>
<td></td>
<td>☒</td>
</tr>
<tr>
<td>Vampir</td>
<td></td>
<td>☒</td>
<td>☒</td>
<td></td>
</tr>
<tr>
<td>Open</td>
<td>SpeedShop</td>
<td>☒</td>
<td>☒</td>
<td>☒</td>
</tr>
<tr>
<td>Periscope</td>
<td>☒</td>
<td></td>
<td>☒</td>
<td></td>
</tr>
<tr>
<td>mpiP</td>
<td>☒</td>
<td>☒</td>
<td>☒</td>
<td></td>
</tr>
<tr>
<td>Paraver</td>
<td></td>
<td>☒</td>
<td>☒</td>
<td>☒</td>
</tr>
<tr>
<td>TAU</td>
<td>☒</td>
<td>☒</td>
<td>☒</td>
<td>☒</td>
</tr>
</tbody>
</table>
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

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**

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

- **database**
Measure execution unobtrusively

- launch optimized application binaries
  - dynamically-linked applications: launch with \texttt{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
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]
database

compile & link
call stack profile
profile execution [hpcrun]
optimized binary
binary analysis [hpcstruct]
program structure

Lecture 9 – Parallel Performance Tools
CIS 631: Advanced Parallel Computing
**HPCToolkit Workflow**

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

Diagram:

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

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

**Associated source code**

**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 – Visualization Modes (2)

On local machine with remote VampirServer

% vampirserver start -n 12

% vampir

VampirServer

Vampir 8

LAN/WAN

Score-P

Many-Core Program

Trace File (OTF2)

Large Trace File
(stays on remote machine)

MPI parallel application

Many-Core Program

Large Trace File
(stays on remote machine)

MPI parallel application
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
```

- **Master Timeline**
- **Navigation Toolbar**
- **Function Summary**
- **Function Legend**
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

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

- 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

Measurements library

Instr. target application

HWC

Measurement library

Local event traces

Parallel wait-state search

Wait-state report

Optimized measurement configuration

Summary report

Report manipulation

Instrumenter compiler / linker

Instrumented executable

Source modules

Scalasca trace analysis

Which problem?

Where in the program?

Which process?

Instrumentation library

Measurement library

Intr. target application

HWC

Optimized measurement configuration

Summary report

Report manipulation

Instrumenter compiler / linker

Instrumented executable

Source modules

Scalasca trace analysis

Which problem?

Where in the program?

Which process?
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

Simulation Profile:
- 66% of simulation time is spent on P2P communication
- Primarily in scatter and gather operations
Callpath Profile: P2P Synchronization

- Point-to-point msgs w/o data
- Masses of P2P sync. operations
- Processes all equally responsible

Processes such as **14.2 evoscaler2** contribute significantly to the performance profile.
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
TAU Performance System
Performance Technology Eras

- HPC performance technology has evolved to serve the dominant architectures and programming models
  - **Observability** era (1991 – 1998)
    - instrumentation, measurement, analysis
  - **Diagnosis** era (1998 – 2007)
    - identifying performance inefficiencies
  - **Complexity** era (2008 – 2012)
    - scale, memory hierarchy, network, multicore, GPU
  - **Productivity** (2013 – future)
    - heterogeneity, richer applications, big data, ML, exascale

- 20+ years of stable parallel execution models
  - “1st person” application focus for tools worked well
  - Productivity era involves performance concerns across system

- Applications are more complex and diverse as HPC attracted workflows, big data analytics, and AI computing
TAU Performance System®

- Tuning and Analysis Utilities (28+ 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-port ed 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 – Present)**

*parallel profiling, tracing, performance extrapolation*

*multiple languages, source analysis, automatic instrumentation*

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

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

2005-2007: More emphasis on tool integration, usability, and data presentation. TAU  
v2.0 released.  
*performance visualization, binary instrumentation, integration,  
performance diagnosis and modeling*

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, heterogeneous  
support, runtime system integration, dynamic adaptation. 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]


**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
TCAU Components

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

- Measurement
  - Probe-based and sample-based supported
  - Internode: MPI, OpenSHMEM, ARMCI, PGAS, DMAPP, ...
  - Intranode: Pthreads, OpenMP, OpenACC, hybrid, ...
  - Heterogeneous: GPU, MIC, CUDA, OpenCL, OpenACC, HIP, ...
  - 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 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**
Source-level / Wrapper Instrumentation

PDT source analyzer

Parsed program

Instrumentation specification file

tau_instrumentor

tau_wrap

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

Instrumented source
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)
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/OTF2, Paraver, SLOG2
  - Trace streams (OTF) and hierarchical trace merging

- Robust timing and hardware performance support
- Multiple counters (hardware, user-defined, system)
- Metadata (hardware/system, application, …)
Configurable Measurement

- TAU offers configurable measurement capabilities
- Decide how to make events observable
  - *Probes*: direct instrumentation in code
  - *Samples*: indirect observation via interrupts
- Decide on how much data you want to obtain
  - *Profiling*: runtime statistics of varying detail
  - *Tracing*: timestamped events with values
- Control perspective and context
  - Levels of abstraction
  - Capture of metadata
  - Heterogeneous components
Direct Performance Observation

- Execution actions 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 observation

- Observation is direct
  - Direct instrumentation of program code (probes)
  - Instrumentation invokes performance measurement
  - Event measurement = performance data + context

- Performance experiment
  - Actual events + performance measurements
Indirect Performance Observation

- Program code instrumentation is not used
- Performance is observed indirectly
  - Execution is interrupted
    - can be triggered by different events
  - Execution state is queried (sampled)
    - different performance data measured
  - Event-based sampling (EBS)
- Performance attribution is inferred
  - Determined by execution context (state)
  - Observation resolution determined by interrupt period
  - Performance data associated with context for period
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
TAU Profile Analysis Framework

Performance Data
- Profiles
  - TAU, mpiP, ompP, HPMToolkit, Cube, HPCToolkit, Gprof, Dynaprof, PSRun
  - Supermon, MRNet
- Runtime Data Collection
- 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

GUI
Parallel Profile Analysis (ParaProf)
ParaProf – Single Thread of Execution View

Miranda
- hydrodynamics
- Fortran + MPI
- LLNL BG/L
ParaProf 3D Visualization
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
PerfExplorer Performance Data Mining

- Programmable, extensible framework to support workflow automation
- Rule-based inference for expert system analysis
S3D Scalability Analysis

Jaguar

BG/P

\( r = 1 \) implies direct correlation.
S3D on Intrepid
Metadata Collection

- 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

<table>
<thead>
<tr>
<th>TrialField</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>190/pdt_mpi/examples/tau2/amorris/home/</td>
</tr>
<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>2992.505</td>
</tr>
<tr>
<td>CPU Type</td>
<td>Intel(R) Xeon(R) CPU S160 3.00GHZ</td>
</tr>
<tr>
<td>CPU Vendor</td>
<td>GenuineIntel</td>
</tr>
<tr>
<td>CWD</td>
<td>/home/amorris/tau2/examples/pdt_mpi/f90</td>
</tr>
<tr>
<td>Cache Size</td>
<td>4096 KB</td>
</tr>
<tr>
<td>Executable</td>
<td>/home/amorris/tau2/examples/pdt_mpi/f90</td>
</tr>
<tr>
<td>Hostname</td>
<td>demon.nic.uoregon.edu</td>
</tr>
<tr>
<td>Memory Size</td>
<td>8161240 kB</td>
</tr>
<tr>
<td>Node Name</td>
<td>demon.nic.uoregon.edu</td>
</tr>
<tr>
<td>OS Machine</td>
<td>x86_64</td>
</tr>
<tr>
<td>OS Name</td>
<td>Linux</td>
</tr>
<tr>
<td>OS Release</td>
<td>2.6.9-42.0.3 EL perfctrsmp</td>
</tr>
<tr>
<td>OS Version</td>
<td>#1 SMP Fri Nov 3 07:34:13 PST 2006</td>
</tr>
<tr>
<td>Starting Timestamp</td>
<td>118354807220996</td>
</tr>
<tr>
<td>TAU Architecture</td>
<td>x86_64</td>
</tr>
<tr>
<td>TAU Config</td>
<td>-papli=/usr/local/packages/papli-3.5.0 -M...</td>
</tr>
<tr>
<td>Timestamp</td>
<td>1183548074317538</td>
</tr>
<tr>
<td>pid</td>
<td>11395</td>
</tr>
<tr>
<td>username</td>
<td>amorris</td>
</tr>
</tbody>
</table>

Multiple TAUdb databases
Tracing: Jumpshot (ships with TAU)
Tracing: Vampir
**TAU Execution Command – tau_exec**

- **Uninstrumented execution**
  
  \[
  \% \text{mpirun} -np 256 ./a.out
  \]

- **Track MPI performance**
  
  \[
  \% \text{mpirun} -np 256 \text{ tau}_\text{exec} ./a.out
  \]

- **Track I/O and MPI performance (MPI default)**
  
  \[
  \% \text{mpirun} -np 256 \text{ tau}_\text{exec} -io ./a.out
  \]

- **Track memory operations**
  
  \[
  \% \text{setenv} \text{ TAU}\_\text{TRACK}\_\text{MEMORY}\_\text{LEAKS} 1
  \% \text{mpirun} -np 256 \text{ tau}_\text{exec} -\text{memory} ./a.out
  \]

- **Track I/O performance and memory operations**
  
  \[
  \% \text{mpirun} -np 256 \text{ tau}_\text{exec} -io -\text{memory} ./a.out
  \]
TAU’s Support for Runtime Systems (1)

- **MPI**
  - PMPI profiling interface
  - MPI_T tools interface (performance and control variables)

- **Pthread**
  - Captures performance per thread of execution

- **OpenMP**
  - OMPT tools interface to track salient OpenMP runtime events
  - Opapi source rewriter
  - Preloading wrapper OpenMP runtime library

- **OpenACC**
  - OpenACC instrumentation API
  - Track data transfers between host and device (per-variable)
  - Track time spent in kernels

- **OpenCL**
  - OpenCL profiling interface
  - Track timings of kernels
TAU’s Support for Runtime Systems (2)

- CUDA (NVIDIA)
  - Cuda Profiling Tools Interface (CUPTI)
  - Track data transfers between host and GPU
  - Track access to uniform shared memory between host and GPU

- ROCm (AMD)
  - Rocprofiler and Roctracer instrumentation interfaces
  - Track data transfers and kernel execution between host and GPU

- Kokkos
  - Kokkos profiling API
  - Push/pop interface for region, kernel execution interface

- Python
  - Python interpreter instrumentation API
  - Tracks Python routine transitions as well as Python to C transitions

- Intel® oneAPI DPC++/SYCL
  - Level Zero
  - Track time spent in kernels executing on GPU
  - Track time spent in OneAPI runtime calls
Multi-Level Measurement

- MPI + OpenMP
  - MPI_T + PMPI + OMPT may be used to track MPI and OpenMP
- MPI + CUDA
  - PMPI + CUPTI interfaces
- OpenCL + ROCm
  - Rocprofiler + OpenCL instrumentation interfaces
- MPI + Intel® oneAPI DPC++/SYCL
  - PMPI + Level Zero interfaces
- Kokkos + OpenMP
  - Kokkos profiling API + OMPT to transparently track events
- Kokkos + pthread + MPI
  - Kokkos + pthread wrapper interposition library + PMPI layer
- Python + CUDA
  - Python + CUPTI + pthread profiling interfaces (e.g., Tensorflow, PyTorch)
- MPI + OpenCL
  - PMPI + OpenCL profiling interfaces
What does TAU support?

C/C++  CUDA  UPC  OpenCL  Python
Fortran  OpenACC  GPU  GPI  MPI
OpenMP
Intel MIC  Java  Sun
GNU  LLVM  PGI  Cray  AIX
MPC  Linux  Windows  BlueGene
Linux  Fujitsu  ARM64
NVIDIA  Power 9  OS X

Insert yours here
**TAU Availability**

- Ported to all major HPC platforms
  - Intel, AMD, Arm, IBM
  - CPU and GPU
- Available at all DOE national laboratories
  - Targeting exascale platforms
- Works with all HPC programming systems
- Works with HPC communication libraries
- Active constant development
- Supported IBM systems, compilers, AIX/Linux, and POWER processors for many years
Tools / Technology Integration

Tools / Technologies
- TAU
- mpiP
- PAPI
- GPTL
- RCRToolkit
- PBound
- Roofline
- PEBIL
- PSiNtracer
- ROSE
- CHiLL
- Active
- Harmony
- Orio

Research Areas
- Performance
- Modeling
- Reliability
- Code analysis
- Autotuning

Integration End-to-end
Performance Optimization
SciDAC applications
Resilience
Energy

SUPER was a 5-year SciDAC Institute (2011-2016)
Working with the RAPIDS / RAPIDS-2 SciDAC Institute (2017-now)
End-to-End Productivity in Future HPC

Increased complexity in exascale applications

Application Software Productivity
- Metrics/attributes
- Ease of use
- Heterogeneous portability
- Reusability
- Interoperability/maintainability
- Hardware capabilities discovery
- Software performance engineering

Execution-time Productivity
- Metrics/attributes-execution models
- Observability/controllability/monitoring
- Instrumentation/measurement/traceability
- Productivity modeling/simulation: prediction, diagnosis
- Performance engineering
- Dynamic performance adaptation

 Courtesy of Thomas Ndousse-Fetter, DOE
Performance Portability

- Performance portability objective
  - Write once and run everywhere
  - Minimize how much special code needs to be written for each target platform
  - Can be done by using portability layers, language features, and libraries that provide functionality across many systems

- Kokkos programming for performance portability
- PETSc library for portable PDE solving
Kokkos

- Productive, portable, and performant programming model
  - Shared-memory programming model
  - Provides abstractions for node level parallelism (X in MPI+X)
- C++ API for expressing parallelism in your program
  - Provides data abstractions
  - Aggressive compiler transformations using C++ templates
  - Low-level code targets different backends: OpenMP, Pthread, CUDA
- Helps you create single source performance portable codes
- Creates a problem for performance evaluation tools
  - Gap between performance data and higher-level abstractions
- Kokkos profiling API for mapping performance data

https://kokkos.org
CabanaMD on IBM AC922 with V100 GPUs

- Demonstrates TAU’s support for integrated probe-based and sample-based measurement
- Utilizes Kokkos profiling interface
TAU runs on all these pre-exascale platforms

TAU will run on all these exascale platforms
Open source projects lay the foundation for DOE simulations

Parallel Programming Models
- OpenMPI
- Kokkos
- UPC
- MPICH
- RAJA
- GASNet-EX

Filesystems & I/O
- ZFS
- Lustre
- ADIOS

Packaging/Build
- Spack
- BLT
- SHIFTER
- Charliecloud

Meshing / Finite Elements
- SAMRAI
- CHOMBO
- MFEM
- AMReX

Resource Managers
- SLURM
- flux

Scientific Visualization
- ParaView
- visit
- Ascent
- VTK

Parallel Solvers
- hypre
- TRILINOS
- PETSc

TAU works with all of these ECP software projects
Further Information

- TAU Performance System
  http://tau.oregon.edu

- ParaTools, Inc.
  http://www.paratools.com
  - TAU Commander
    http://taucommander.com
  - HPC Linux
    http://www.hpclinux.com

- Extreme-Scale Scientific Software Stack (E4S)
  http://e4s.io
Support Acknowledgements

- US Department of Energy (DOE)
  - ANL
  - Office of Science contracts, ECP
  - SciDAC, LBL contracts
  - LLNL-LANL-SNL ASC/NNSA contract
  - Battelle, PNNL and ORNL contracts
- Department of Defense (DoD)
  - PETTT, HPCMP
- National Science Foundation (NSF)
  - SI2-SSI, Glassbox
- NASA
- CEA, France
- Partners:
  - University of Oregon
  - The Ohio State University
  - ParaTools, Inc.
  - University of Tennessee, Knoxville
  - T.U. Dresden, GWT
  - Jülich Supercomputing Center