Supercomputing for Everyone Series: Performance Tuning Summer School

L6: Visual tools can be fun

Holger Brunst
Technische Universität Dresden, Germany

August 17–21, 2015
Organization of this class

- course runs August 17-21, from 11 a.m. until 4.30 p.m. (EDT)
- 20 sites, each site should have an instructor/TA/helper
- please put your microphones on mute, if you are not asking questions
- set up in lectures (60 min) and exercises (75 min)
- Q&A time at end of each lecture
- chat for discussions
- class material and links: http://go.iu.edu/CtB
### Lunch breaks are important

**Monday**
- 11.00-12.30  L1_Performance is ambiguous
- 12.30-13.15  X1_First steps on Blue Waters
- 13.15-14.15  **Lunch break**
- 14.15-15.15  L2_Evaluation first!
- 15.15-16.30  X2_Pen, paper, and performance

**Tuesday**
- 11.00-12.00  L3_Use simple tools for simple questions
- 12.00-13.15  X3_The command line is your friend
- 13.15-14.15  **Lunch break**
- 14.15-15.15  L4_Tuning needs persistence
- 15.15-16.30  X4_Benchmarks provide the baseline

**Wednesday**
- 11.00-12.00  L5_Core tuning pays off n-times
- 12.00-13.15  X5_How to access data efficiently
- 13.15-14.15  **Lunch break**

**Thursday**
- 11.00-12.00  L6_Visual tools can be fun, sometimes
- 12.00-13.15  X6_Zoom and scroll
- 13.15-14.15  **Lunch break**
- 14.15-15.15  L7_Sharing may double the sorrow (OpenMP)
- 15.15-16.30  X7_OpenMP enables quick 'n easy gains

**Friday**
- 11.00-12.00  L8_Hand made parallelization hurts (MPI)
- 12.00-13.15  X8_From bad MPI to good MPI
- 13.15-14.15  **Lunch break**
- 14.15-15.15  L9_Climb the mount Olympus with GPUs
- 15.15-16.30  X9_Hybrid tuning in practice

**14.15-15.15**  L6_Visual tools can be fun, sometimes  
**15.15-16.30**  X6_Zoom and scroll

*All times are EDT*
Goal

- introduce
  - visual performance analysis
  - the different analysis approaches
  - common visualization techniques
Collecting data for multiple tools in one step
Analyzing profiles visually
Browsing automatically detected bottlenecks
Visually detecting anomalies in traces

OUTLINE
Credits

• these slides include material from
  – Virtual Institute – High Productivity Supercomputing

• with the kind permission of
  – Brian Wylie, Jülich Supercomputing Centre
  – Markus Geimer, Jülich Supercomputing Centre
  – Sameer Shende, University of Oregon
Virtual Institute – High Productivity Supercomputing

- goal: improve simulation codes running on highly-parallel computer systems
- start-up funding (2006–2011) by Helmholtz Association of German Research Centres
  - development and integration of HPC performance and correctness tools
  - workshops
  - service
    - support email lists
    - application engagement
- http://www.vi-hps.org
The founding institutions

Forschungszentrum Jülich
- Jülich Supercomputing Centre

RWTH Aachen University
- Centre for Computing & Communication

Technische Universität Dresden
- Centre for Information Services & HPC

University of Tennessee (Knoxville)
- Innovative Computing Laboratory
Barcelona Supercomputing Center
  ■ Centro Nacional de Supercomputación
German Research School
  ■ Laboratory of Parallel Programming
Lawrence Livermore National Lab.
  ■ Centre for Applied Scientific Computing
Technical University of Munich
  ■ Chair for Computer Architecture
University of Oregon
  ■ Performance Research Laboratory
University of Stuttgart
  ■ HPC Centre
University of Versailles St-Quentin
  ■ LRC ITACA
Allinea Software Ltd
Productivity tools

- MUST - MPI usage correctness checking
- PAPI - Interfacing to hardware performance counters
- Periscope - Automatic analysis via an on-line distributed search
- Scalasca - Large-scale parallel performance analysis
- TAU - Integrated parallel performance system
- Vampir - Interactive graphical trace visualization & analysis
- Score-P - Community instrumentation & measurement infrastructure
Productivity tools (cont.)

- DDT/MAP/PR: Parallel debugging & profiling
- KcacheGrind: Callgraph-based cache analysis [x86 only]
- MAQAO: Assembly instrumentation & optimization [x86-64 only]
- mpiP/mpiPview: MPI profiling tool and analysis viewer
- Open MPI: Integrated memory checking
- Open Speedshop: Integrated parallel performance analysis environment
- Paraver/Dimemas/Extrae: Event tracing and graphical trace visualization & analysis
- Rubik: Process mapping generation & optimization [BG only]
- SIONlib/Spindle: Optimized native parallel file I/O & library loading
- STAT: Stack trace analysis tools
Collecting data for multiple tools in one step
Analyzing profiles visually
Browsing automatically detected bottlenecks
Visually detecting anomalies in traces

OUTLINE
Score-P: A Joint Performance Measurement Run-Time Infrastructure

Markus Geimer\textsuperscript{2)}, Bert Wesarg\textsuperscript{1)}, Brian Wylie\textsuperscript{2)}

With contributions from Andreas Knüpfer\textsuperscript{1)} and Christian Rössel\textsuperscript{2)}
\textsuperscript{1)}ZIH TU Dresden, \textsuperscript{2)}FZ Jülich
Fragmentation of tools landscape

- Several performance tools co-exist
  - Separate measurement systems and output formats
- Complementary features and overlapping functionality
- Redundant effort for development and maintenance
  - Limited or expensive interoperability
- Complications for user experience, support, training
Score-P functionality

- Provide typical functionality for HPC performance tools
- Support all fundamental concepts of partners’ tools

- Instrumentation (various methods)
- Flexible measurement without re-compilation:
  - Basic and advanced profile generation
  - Event trace recording
  - Online access to profiling data

- MPI/SHMEM, OpenMP/Pthreads, and hybrid parallelism (and serial)
- Enhanced functionality (OpenMP 3.0, CUDA, highly scalable I/O)
Design goals: functional requirements

- Generation of call-path profiles and event traces
- Using direct instrumentation, later also sampling
- Recording time, visits, communication data, hardware counters
- Access and reconfiguration also at runtime
- Support for MPI, OpenMP, basic CUDA, and all combinations
  - Later also OpenCL/OpenACC/…
Design goals: non-functional requirements

- Portability: all major HPC platforms
- Scalability: petascale
- Low measurement overhead
- Easy and uniform installation through UNITE framework
- Robustness
- Open Source: New BSD License
Future features and commitment

- Scalability to maximum available CPU core count
- Support for OpenCL, OpenACC, Intel MIC
- Support for sampling, binary instrumentation
- Support for new programming models, e.g., PGAS
- Support for new architectures

- Ensure a single official release version at all times which will always work with the tools
- Allow experimental versions for new features or research

- Commitment to joint long-term cooperation
Instrument Application

- ...  
- # The Fortran compiler used for MPI programs  
- #MPIF77 = mpif77  
- # Alternative variants to perform instrumentation  
- ...  
- MPIF77 = scorep mpif77  
- # This links MPI Fortran programs; usually the same as ${MPIF77}  
- FLINK = $(MPIF77)  
- ...  
  
prefix compiler with scorep wrapper
Re-build executable using Score-P instrumenter

- `% make bt-mz CLASS=W NPROCS=4`
- `cd BT-MZ; make CLASS=W NPROCS=4 VERSION=
- make: Entering directory 'BT-MZ'
- cd ../sys; cc -o setparams setparams.c -lm
- ../sys/setparams bt-mz 4 W
- `scorep mpif77 -c -O3 -fopenmp bt.f`
- `make bt-mz CLASS=W NPROCS=4`
- cd ../common; `scorep mpif77 -c -O3 -fopenmp timers.f`
- `scorep mpif77 -O3 -fopenmp -o ../bin.scorep/bt-mz_W.4` 
  `bt.o initialize.o exact_solution.o exact_rhs.o set_constants.o`
- `adi.o rhs.o zone_setup.o x_solve.o y_solve.o exch_qbc.o`
- `solve_subs.o z_solve.o add.o error.o verify.o mpi_setup.o`
- `../common/print_results.o ../common/timers.o`
- `Built executable ../bin.scorep/bt-mz_W.4`
- `make: Leaving directory 'BT-MZ'`
Configure Score-P measurements via environment variables

```
% scorep-info config-vars --full
SCOREP_ENABLE_PROFILING
   Description: Enable profiling
SCOREP_ENABLE_TRACING
   Description: Enable tracing
SCOREP_TOTAL_MEMORY
   Description: Total memory in bytes for the measurement system
SCOREP_EXPERIMENT_DIRECTORY
   Description: Name of the experiment directory
SCOREP_FILTERING_FILE
   Description: A file name which contain the filter rules
   [... More configuration variables ...]
```
Run application

% cd bin.scorep
% export SCOREP_EXPERIMENT_DIRECTORY=scorep_bt-mz_W_4x4_sum
% OMP_NUM_THREADS=4  mpiexec -np 4 ./bt-mz_W.4

NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP Benchmark
Number of zones:   4 x   4
Iterations:  200   dt:   0.000800
Number of active processes:     4
Use the default load factors with threads
Total number of threads:   16  (  4.0 threads/process)
Use the default load factors with threads
[...]
BT-MZ Benchmark Completed.
Time in seconds = 54.39
Examine Summary Analysis Report

- Creates experiment directory ./scorep_bt-mz_W_4x4_sum containing
  - a record of the measurement configuration (scorep.cfg)
  - the analysis report that was collated after measurement (profile.cubex)

```bash
% ls
... scorep_bt-mz_W_4x4_sum
% ls scorep_bt-mz_W_4x4_sum
profile.cubex scorep.cfg
```
Explore experiment graphically with CUBE and ParaProf

% cube scorep.bt-mz_W_4x4_sum/profile.cubex

[CUBE GUI showing summary analysis report]

% paraprof scorep.bt-mz_W_4x4_sum/profile.cubex

[TAU ParaProf GUI showing summary analysis report]
Collecting data for multiple tools in one step
Analyzing profiles visually
Browsing automatically detected bottlenecks
Visually detecting anomalies in traces

OUTLINE
Profile Examination with TAU ParaProf

Sameer Shende
Performance Research Lab, University of Oregon
http://TAU.uoregon.edu
TAU Performance System®

- Measurement and analysis support
  - Parallel profiling and tracing
  - Use of Score-P for native OTF2 and CUBEX generation
  - Efficient callpath profiles and trace generation using Score-P

- Analysis
  - Parallel profile analysis (ParaProf), data mining (PerfExplorer)
  - Performance database technology (TAUdb)
  - 3D profile browser
Binary Rewriting Instrumentation

- Support for both static and dynamic executables
- Specify a list of routines to instrument
- Specify the TAU measurement library to be injected
- MAQAO:
  - % tau_rewrite -T [tags] a.out –o a.inst
- Dyninst:
  - % tau_run -T [tags] a.out –o a.inst
- Pebil:
  - % tau_pebil_rewrite -T [tags] a.out \ 
    -o a.inst
- Execute the application to get measurement data:
  - % mpirun –np 256 ./a.inst
Event Based Sampling in TAU

% export TAU_MAKEFILE=$TAU/Makefile.tau-icpc-papi-mpi-pdt
% make CC=tau_cc.sh CXX=tau_cxx.sh
% export TAU_SAMPLING=1
% mpirun –np 256 ./a.out
% paraprof
Opari OpenMP Instrumentation with Sampling

% export TAU_MAKEFILE=$TAU/Makefile.tau-icpc-papi-mpi-pdt-opari-openmp
% make CC=tau_cc.sh CXX=tau_cxx.sh
% export TAU_SAMPLING=1; export OMP_NUM_THREADS=16
% mpirun –np 256 ./a.out
% paraprof
Profiling Power Using TAU with PAPI and RAPL

```
#include <TAU.h>

TAU_TRACKPOWER(); // In Fortran: call TAU_TRACK_POWER()

% sudo chmod –R go+r /dev/cpu/*/msr
% sudo /sbin/setcap cap_sys_rawio=ep ./a.out
% unset LD_LIBRARY_PATH
% ldd ./a.out
should have no “not found” entries, Use –Wl,-rpath,/path while linking
% ./a.out
% paraprof
```
ParaProf Profile
Analysis Framework
ParaProf: Main Window

Metric: Time
Value: Exclusive
Mixed MPI and OpenMP Instrumentation
ParaProf Comparison Window
## ParaProf: Thread Statistics Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Exclusive Time</th>
<th>Inclusive Time</th>
<th>Calls</th>
<th>Child Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>!$omp do @y_solve.f:52</td>
<td>5.817</td>
<td>5.817</td>
<td>3,216</td>
<td>0</td>
</tr>
<tr>
<td>!$omp do @z_solve.f:52</td>
<td>5.657</td>
<td>5.657</td>
<td>3,216</td>
<td>0</td>
</tr>
<tr>
<td>!$omp do @x_solve.f:54</td>
<td>5.609</td>
<td>5.609</td>
<td>3,216</td>
<td>0</td>
</tr>
<tr>
<td>!$omp do @rhs.f:191</td>
<td>0.609</td>
<td>0.609</td>
<td>3,232</td>
<td>0</td>
</tr>
<tr>
<td>!$omp do @rhs.f:80</td>
<td>0.583</td>
<td>0.583</td>
<td>3,232</td>
<td>0</td>
</tr>
<tr>
<td>MPI_Waitall</td>
<td>0.402</td>
<td>0.402</td>
<td>603</td>
<td>0</td>
</tr>
<tr>
<td>!$omp implicit barrier</td>
<td>0.402</td>
<td>51,680</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>!$omp do @rhs.f:301</td>
<td>0.36</td>
<td>0.366</td>
<td>3,232</td>
<td>3,232</td>
</tr>
<tr>
<td>!$omp implicit barrier</td>
<td>0.026</td>
<td>0.026</td>
<td>3,216</td>
<td>0</td>
</tr>
<tr>
<td>!$omp implicit barrier</td>
<td>0</td>
<td>0</td>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>!$omp do @rhs.f:37</td>
<td>0.343</td>
<td>0.343</td>
<td>3,232</td>
<td>0</td>
</tr>
<tr>
<td>!$omp do @rhs.f:62</td>
<td>0.225</td>
<td>0.228</td>
<td>3,232</td>
<td>3,232</td>
</tr>
<tr>
<td>!$omp implicit barrier</td>
<td>0.004</td>
<td>0.004</td>
<td>3,216</td>
<td>0</td>
</tr>
<tr>
<td>!$omp implicit barrier</td>
<td>0</td>
<td>0</td>
<td>16</td>
<td>0</td>
</tr>
</tbody>
</table>
Parallel Profile Visualization: ParaProf
ParaProf: Windows -> 3D Visualization -> Bar Plot
ParaProf: Scatter Plot
ParaProf: Topology 3D View (IBM BG/P)
ParaProf: Topology View Torus (IBM BG/Q)
Collecting data for multiple tools in one step
Analyzing profiles visually
**Browsing automatically detected bottlenecks**
Visually detecting anomalies in traces

**OUTLINE**
Automatic trace analysis
with Scalasca

Markus Geimer
Jülich Supercomputing Centre
Automatic search for patterns of inefficient behaviour

- Classification of behavior & quantification of significance
- Guaranteed to cover the entire event trace
- Quicker than manual/visual trace analysis
- Parallel replay analysis exploits available memory & processors to deliver scalability
The Scalasca project

- Project started in 2006
  - Follow-up to pioneering KOJAK project (started 1998)
- Joint development of
  - Jülich Supercomputing Centre
  - German Research School for Simulation Sciences
- Development of a scalable performance analysis toolset for most popular parallel programming paradigms
- Specifically targeting large-scale parallel applications
  - such as those running on IBM BlueGene or Cray XT systems with one million or more processes/threads
Scalasca 2.1 features

• Open source, BSD 3-clause license
• Fairly portable
  – IBM Blue Gene, IBM SP & blade clusters, Cray XT/XE/XK/XC, SGI Altix, Solaris & Linux clusters, Fujitsu FX10 & K computer, ...
• Uses Score-P instrumenter & measurement libraries
  – Scalasca 2.1 core package focuses on trace-based analyses
  – Supports common data formats
    • Reads event traces in OTF2 format
    • Writes analysis reports in CUBE4 format
• Current limitations:
  – No support for nested OpenMP parallelism and tasking
  – Unable to handle OTF2 traces containing CUDA events
Scalasca trace analysis

Measurement library
Instr. target application

Optimized measurement configuration

Instr. target application

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

Source modules
Instrumenter compiler/linker

Instrumented executable

Which problem?
Where in the program?
Which process?

Instrumenter compiler/linker

Optimized measurement configuration

Summary report
Report manipulation

RESEARCH TECHNOLOGIES
INDIANA UNIVERSITY
University Information Technology Services

PERVASIVE TECHNOLOGY INSTITUTE
INDIANA UNIVERSITY
Example: Late Sender

- Waiting time caused by a blocking receive operation posted earlier than the corresponding send
- Applies to blocking as well as non-blocking communication
Example
Late Broadcast

- Waiting times if the destination processes of a collective 1-to-N operation enter the operation earlier than the source process (root)
- Applies to: MPI_Bcast, MPI_Scatter, MPI_Scatterv
scalasca: the command for (almost) everything...

- `scalasca`
- Scalasca 2.1
- Toolset for scalable performance analysis of large-scale applications
- usage: scalasca [OPTION]... ACTION <argument>...
  1. prepare application objects and executable for measurement:
     scalasca -instrument <compile-or-link-command> # skin (using scorep)
  2. run application under control of measurement system:
     scalasca -analyze <application-launch-command> # scan
  3. interactively explore measurement analysis report:
     scalasca -examine <experiment-archive|report> # square

- -c, --show-config  show configuration and exit
- -h, --help         show this help and exit
- -n, --dry-run      show actions without taking them
- --quickref         show quick reference guide and exit
- -v, --verbose      enable verbose commentary
- -V, --version      show version information and exit
BT-MZ summary measurement

- `% export SCOREP_EXPERIMENT_DIRECTORY=scorep_bt-mz_W_4x4_sum`
- `% OMP_NUM_THREADS=4 scan mpiexec -np 4 ./bt-mz_W.4`
- S=C=A=N: Scalasca 2.1 runtime summarization
- S=C=A=N: ./scorep_bt-mz_W_4x4_sum experiment archive
- S=C=A=N: Thu Jun 12 18:05:17 2014: Collect start
- mpiexec -np 4 ./bt-mz_W.4

NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP Benchmark

- Number of zones: 8 x 8
- Iterations: 200 dt: 0.000300
- Number of active processes: 4

[... More application output ...]

- S=C=A=N: Thu Jun 12 18:05:39 2014: Collect done (status=0) 22s
- S=C=A=N: ./scorep_bt-mz_W_4x4_sum complete.
BT-MZ summary analysis report examination

- Score summary analysis report

```bash
% square -s scorep_bt-mz_W_4x4_sum
INFO: Post-processing runtime summarization result...
INFO: Score report written to ./scorep_bt-mz_W_4x4_sum/scorep.score
```

- Post-processing and interactive exploration with CUBE

```bash
% square scorep_bt-mz_W_4x4_sum
INFO: Displaying ./scorep_bt-mz_W_4x4_sum/summary.cubex...
```

- The post-processing derives additional metrics and generates a structured metric hierarchy
Post-processed summary analysis report

Split base metrics into more specific metrics
BT-MZ trace measurement collection...

% export SCOREP_EXPERIMENT_DIRECTORY=scorep_bt-mz_W_4x4_trace
% OMP_NUM_THREADS=4 scan -t mpiexec -np 4 ./bt-mz_W.4
S=C=A=N: Scalasca 2.1 trace collection and analysis
S=C=A=N: ./scorep_bt-mz_W_4x4_trace experiment archive
S=C=A=N: Thu Jun 12 18:05:39 2014: Collect start
mpiexec -np 4 ./bt-mz_B.4
NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP Benchmark

Number of zones:  8 x  8
Iterations:  200    dt:  0.000300
Number of active processes:  4

[... More application output ...]

S=C=A=N: Thu Jun 12 18:05:58 2014: Collect done (status=0) 19s
[... continued ...]
BT-MZ trace measurement ... analysis

- S=C=A=N: Thu Jun 12 18:05:58 2014: Analyze start
- mpiexec -np 4 scout.hyb ./scorep_bt-mz_W_4x4_trace/traces.otf2
- SCOUT Copyright (c) 1998-2012 Forschungszentrum Juelich GmbH
  Copyright (c) 2009-2012 German Research School for Simulation Sciences GmbH

- Analyzing experiment archive ./scorep_bt-mz_W_4x4_trace/traces.otf2
- Opening experiment archive ... done (0.002s).
- Reading definition data ... done (0.004s).
- Reading event trace data ... done (0.130s).
- Preprocessing ... done (0.259s).
- Analyzing trace data ...
  - Wait-state detection (fwd) (1/4) ... done (0.575s).
  - Wait-state detection (bwd) (2/4) ... done (0.138s).
  - Synchpoint exchange (3/4) ... done (0.358s).
  - Critical-path analysis (4/4) ... done (0.288s).
- done (1.360s).
- Writing analysis report ... done (0.121s).
- Total processing time : 1.924s
- S=C=A=N: Thu Jun 12 18:06:00 2014: Analyze done (status=0) 2s
Post-processed trace analysis report

Additional trace-based metrics in metric hierarchy
Online metric description

Access online metric description via context menu
Online metric description

Late Sender Time

Description:
Refers to the time lost waiting caused by a blocking receive operation (e.g., MPI_Recv or MPI_Wait) that is posted earlier than the corresponding send operation.

If the receiving process is waiting for multiple messages to arrive (e.g., in an call to MPI_Waitall), the maximum waiting time is accounted, i.e., the waiting time due to the latest sender.

Unit:
Seconds

Diagnosis:
Try to replace MPI_Recv with a non-blocking receive MPI_Irecv that can be posted earlier, proceed concurrently with computation, and complete with a wait operation after the message is expected to have been sent. Try to post sends earlier, such that they are available when receivers need them. Note that outstanding messages (i.e., sent before the receiver is ready) will occupy internal message buffers, and that large numbers of posted receive buffers will also introduce message management overhead, therefore moderation is advisable.

Parent:
MPI Point-to-point Communication Time

Children:
Critical-path analysis

Critical-path profile shows wall-clock time impact.
Critical-path analysis

Critical-path imbalance highlights inefficient parallelism
To investigate most severe pattern instances, connect to a trace browser...

...and select trace file from the experiment directory
Show most severe pattern instances

Select “Max severity in trace browser” from context menu of call paths marked with a red frame
Investigate most severe instance in Vampir

Vampir will automatically zoom to the worst instance in multiple steps (i.e., undo zoom provides more context)
Collecting data for multiple tools in one step
Analyzing profiles visually
Browsing automatically detected bottlenecks
Visually detecting anomalies in traces

OUTLINE
Performance Visualization with Vampir

Holger Brunst, Bert Wesarg, Andreas Knüpfer
ZIH, Technische Universität Dresden

http://www.vampir.eu
vampirsupport@zih.tu-dresden.de
Mission

- Visualization of dynamics of complex parallel processes
- Full details for arbitrary temporal and spatial levels
- Supplement to automatic analysis

- 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?
Product overview

- Vampir & VampirServer
  - Interactive trace visualization and analysis
  - Intuitive browsing and zooming
  - Scalable to large trace data sizes (20 TByte)
  - Scalable to high parallelism (200,000 processes)

- Vampir is available for Linux, Windows and Mac OS X
The 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
How Vampir opens up for the first time
Master Timeline

Detailed information about functions, communication and synchronization events for collection of processes.
Process Timeline

Detailed information about different levels of function calls in a stacked bar chart for an individual process.
Typical program phases

- Initialization Phase
- Computation Phase
Counter Data Timeline

Detailed counter information over time for an individual process.
Performance Radar

Detailed counter information over time for a collection of processes.
Zoom in: Initialization Phase

Context View: Detailed information about function “initialize ”.
Feature: Find Function

Execution of function “initialize_” results in higher page fault rates.
Computation Phase

Computation phase results in higher floating point operations.
Zoom in: Computation Phase

MPI communication results in lower floating point operations.
Zoom in: Finalization Phase

“Early reduce” bottleneck.
Function Summary: Overview of the accumulated information across all functions and for a collection of processes.

Process Summary: Overview of the accumulated information across all functions and for every process independently.
Process Summary

Find groups of similar processes and threads by using summarized function information.
Collecting data for multiple tools in one step
Analyzing profiles visually
Browsing automatically detected bottlenecks
Visually detecting anomalies in traces
Class evaluation

- we appreciate your opinion and feedback
- please consider filling-in our class evaluation questionnaire

https://www.surveymonkey.com/r/pptune15