Open Source Performance Analysis for Large Scale Systems

Open|SpeedShop
Capabilities and Internal Structure:
Current to Petascale

CScADS Workshop, July 16-20, 2007
Jim Galarowicz, Krell Institute
Trademark Acknowledgements

- Intel, Intel Inside (logos) and Itanium are trademarks of Intel Corporation in the United States, other countries, or both.
- Linux is a trademark of Linus Torvalds in the United States, other countries or both.
- Qt and the Qt logo are trademarks of Trolltech in Norway, the United States and other countries.
- SGI SpeedShop, IRIX, SGI and SGI Altix are trademarks of Silicon Graphics Inc.
- IBM is a registered trademarks of International Business Machines Corporation in the United States, other countries, or both.
- All other trademarks mentioned herein are the property of their respective owners.
Talk Outline

- Open|SpeedShop – What is it?
- Capabilities and Feature Overview
- Internal Components and Interaction
- Petascale Computing Support
- Questions
Open|SpeedShop

What is it?

- Comprehensive Parallel Performance Analysis Framework
  - **Goal:** Most common performance analysis steps in one tool
  - **Targets** Users *and* Tool Developers
  - **Set of performance analysis tools built on flexible framework**

- **Funding**
  - DOE/NNSA as part of ASC PathForward
  - Initial phase co-funded by SGI

- **Status**
  - Version 1.0 available as source and RPMs
  - Development version available through cvs
  - Open Source: code is GPL/LGPL
Partners

- Krell Institute
  - Hosts Development
-ASC Tri-Laboratories
  - Lawrence Livermore
  - Los Alamos
  - Sandia
- University of Wisconsin & University of Maryland
  - DynInst & Infrastructure
Acknowledgements

- Open|SpeedShop Team Members
  - Scott Cranford, Sandia National Labs
  - Jim Galarowicz, Krell Institute
  - Bill Hachfeld, Krell Institute
  - Don Maghrak, Krell Institute
  - Dave Montoya, Los Alamos National Labs
  - Martin Schulz, Lawrence Livermore National Labs

- Dyninst Team Members
  - Bart Miller
  - Matt Legendre
  - Drew Bernat
Overview / Highlights

• Open Source Performance Analysis Tool
  ▪ *Extensible* by using plugins for data collection and viewing
  ▪ Emphasis on *usability* from the start - usability studies

• Instrumentation at Runtime
  ▪ Use of *unmodified application binaries*
  ▪ *Attach/Detach to/from* running executables/applications
  ▪ *Load and Start* executables/applications into tool

• Flexible and Easy to use user interfaces
  ▪ *GUI* with wizards to guide users through creation of experiment
  ▪ *Command Line* uses dbx/gdb like commands
  ▪ *Batch* executes commands file or simple create, run view preset
  ▪ *Python Scripting* uses API that feeds into command line interface
Overview / Highlights

- Large Range of Platforms
  - *Linux Clusters* with x86, IA-64, Opteron, and EM64T CPUs
  - *SSI* systems
  - Designed with *portability* in mind

- Availability
  - Used at *all three ASC labs* with lab-size applications
  - Source and RPM versions available
  - [www.openspeedshop.org](http://www.openspeedshop.org)

- Linux versions
  - Tested on typical Linux distributions
    (including *SLES, RHEL, Fedora Core, Suse* ....)
Features:
Performance Experiments

- Available Now:
  - PC sampling (*pcsamp*)
  - User time (*usertime* )
  - Hardware counter (*hwc, hwctime*)
  - MPI call tracing (*mpi, mpit*)
  - I/O call tracing (*io, iot*)
  - Floating Point Exception (FPE) tracing (*fpe*)

- Extensible
  - Plugin concept for collectors and views
  - Well defined/documented APIs – Plugin Guide
Typical Workflow

- Application
  "Experiment"

- Consists of one or more data "Collectors"

- Results can be displayed using several "Views"

- Process Management Panel

- Stored in SQL database

- Open SpeedShop™
Welcome to OpenSpeedShop: Introduction Wizard page 1 of 2

Please select one of the following to begin analyzing your application or your previously saved performance data file for performance issues:

- GENERATE NEW PERFORMANCE DATA: I would like to load or attach to an application/executable and gather new performance information on it. A series of wizard panels will guide you through the process of creating a performance experiment and running it.

- LOAD SAVED PERFORMANCE DATA: I have a saved performance experiment data file that I would like to load and analyze. OpenSpeedShop saved performance experiment filenames have the prefix '.openss'

- COMPARE SAVED PERFORMANCE DATA: I have two saved performance experiment data files that I would like to load and compare. OpenSpeedShop saved performance experiment filenames have the prefix '.openss'

Gather new performance data

Command Line Interface

Analyze existing data from previous runs
Select the type of data to be gathered – choose experiment.

- PCSAMP: I'm trying to find where my program is spending most of its time. Most lightweight impact on application.
- USERTIME: I'd like to see information about which routines are calling other routines in addition to the inclusive/exclusive timing information.
- HWC: I'd like to see what kind of performance information the internal Hardware Counters can show me.
- FPE: I would like to know how many times my program is causing Floating Point Exceptions and where in my program they are occurring.
- I/O: I would like to see which Input/Output calls are being made and where most of that time is being spent.
- MPI: I would like to see what MPI calls are being made and where the MPI calls are being made in my program.
Graphical display with basic charts

Program output

Aggregated Inclusive/Exclusive Time from 64 process MPI job

Features:
- Performance Experiments
- Aggregated Inclusive/Exclusive Time from 64 process MPI job
- Graphical display with basic charts
- Program output

Program output:
```
Experiment 1 has terminated.
openMPI >
Experiment 4 has terminated.
openMPI >
```
Statements with high execution times

16

Per line/statement statistics

Source window

Statements with high execution times

16
Parallel Performance Analysis

- Open|SpeedShop supports MPI and Multithreading
  - MPI Process control using MPIR interface
  - Works with multiple MPI implementations
  - Currently: *mpich, openmpi, lampi, lam, slurm, mpt*
  - Attach to running appl. or create appl. within O|SS

- Parallel Experiments
  - Apply sequential collectors to all nodes
  - Specialized MPI tracing experiments

- Results
  - By default results are aggregated
  - Optional: select individual processes
  - Compare or group ranks
Called MPI routines

<table>
<thead>
<tr>
<th>Exclusive MPI Call Time(ms)</th>
<th>% of Total</th>
<th>Function (defining library)</th>
</tr>
</thead>
<tbody>
<tr>
<td>32131.9560</td>
<td>87.0245</td>
<td>PMPI_Waitall (libmpi.so.1.0:libmpi.so.1.0:...)</td>
</tr>
<tr>
<td>3113.0583</td>
<td>8.4312</td>
<td>PMPI_Isend (libmpi.so.1.0: ...)</td>
</tr>
<tr>
<td>1484.4262</td>
<td>4.0203</td>
<td>PMPI_Irecv (libmpi.so.1.0: /...)</td>
</tr>
<tr>
<td>138.5675</td>
<td>0.3753</td>
<td>PMPI_Allreduce (libmpi.so....)</td>
</tr>
<tr>
<td>54.8887</td>
<td>0.1487</td>
<td>PMPI_Finalize (libmpi.so.1....)</td>
</tr>
</tbody>
</table>

- **Parallel Performance Analysis**

- **Status:** Loaded saved data from file /g/g91/schulz/oss-demo/smg2000/test
Advanced Capabilities

- Stack trace views
  - Included in tracing and user time experiments
  - Visualize as call-tree and trace-back

- Experiment and Rank/Process/Thread Comparisons

- View results by Time segments

- Multi-rank analysis
  - Restrict results to task sets
  - Compare tasks or task sets
  - Cluster Analysis (grouping similar processes)
Open|SpeedShop Architecture
Open|SpeedShop
High Level Interactions
OpenSpeedShop
High Level Interactions

“Tool” Host

Client Machine

EXperiment
DATa Base

Tool Process

User Interface

libopenss-framework

Collector Plugin

“Victim” Host

Victim Process

Collector Runtime

libopenss-runtime

Server Machine

Performance Data Gathered and Buffered Into Data Structures (Blobs) in the Victim Process

Data Blob

Data Blob

BLUE BOXES - Written By Plugin Writer
BLACK BOXES - Part of OpenSpeedShop
OpenSpeedShop
Client Architecture
Open|SpeedShop

Server Architecture
Dyninst Component in Open|SpeedShop

At the node level:

- Obtain and Process Application Symbols
- Attach to a running process
- Insert Code into Application Dynamically
  - Execute at Entry and Exit
  - Execute Now
  - Execute In Place of
- Control the Process/Application (start, stop, ...)
- Offline collectors will use symtabAPI component
DPCL Component

Across nodes:

- Connect to application on each node
- Execute Dyninst functions on each remote node
- Use DPCL daemons to return gathered data to the client
Framework Component

Key Component for Open|SpeedShop

- Multi-threaded to support server/client requests
- Interface with the Instrumentor (DPCL, MRNet, other)
  - Insert instrumentation, start/stop collecting
  - Retrieve and store application symbol table information
- Receive performance data from runtime
- Create and manage Open|SpeedShop database
- Provide User Interface with data for display
Plugin Components

- **Types of Plugins**
  - View, GUI panels, Collector

- **All default experiments use plugin mechanism**

- **Collector Plugins**
  - Client and Runtime plugin for each collector
  - Runtime: what performance data to gather
  - Runtime: inserted into application for gathering
  - Client: how to view the data, start/stop gathering
**Plugin Components**

- **GUI plugins use CLI commands to interface**
  - All commands go through a single interface
    - Including Python Scripting Interface
  - Ensures equal functionality and robustness
  - Enables easier debugging
  - Have GUI history by using command history tracking in the CLI
  - Key functionality that will enable GUI separation, if desired
Other External Components

- **xdr**
  - Encode data for transfer between runtime and client.
  - Takes care of endianness issues.
- **python**
  - Scripting API language
- **SQLite**
  - Performance database storage, queries
- **MPIR interface**
  - Retrieving the list of MPI ranked processes
- **libmonitor for offline collectors**
  - Trap dsos, start gathering, stop gathering, callbacks
Peta-Scale support

Data Collection and Transport

- Replace DPCL with MRNet for distributed communication, control, and monitoring
- Change the existing Instrumentor API to be process group (thread group) centric
- Create MRNet instrumentor
- Define Tool/Daemon Protocol (tool via MRNet to application on nodes)
Peta-Scale support

- Result storage, aggregation, and analysis
  - Use of MRNet to gather and analyze perf data
    - Filter data, use intermediate database based on bandwidth available
    - Use extended cluster analysis techniques, apply to database to reduce amount of data stored
    - Create additional wizards to guide user
    - Use filter plugins to aggregate data

- Offline Collectors
  - Execute experiments without tool backend
  - Target for microkernel architectures
Open|SpeedShop

Petascale Architecture

- Python Module
- Shell
- GUI
- Command Line Interface
- Data Abstraction
- Base Tool Layer
- Data Collector
- Instrumentor Class
- Execution Environment

Plugins
Framework
Instrumentation
Existing
Petascale
Across nodes:

- Execute Dyninst functions on each remote node
- Use tree structure to return gathered data to the client
- Use filters within the tree structure reduce the gathered data on it's way to the client
Alternative method of gathering performance data:

- Targets micro kernel architecture
  - Available in general, but targets platforms where Dyninst support is not available.

- Static application support
  - Requires relinking application with static collector runtime libraries

- Dynamic application support
  - Use LD_PRELOAD to link runtime library to application
  - Leverage libmonitor for dynamic support
Offline Collectors
PetaScale Open|SpeedShop

Alternative method of gathering perf data:

- Offline data written in simple “raw” format
  - Separate tool to convert into native database file format for standard viewing/storage.
  - Eventually Open|SpeedShop client will also do conversion

- Reuse existing collector runtimes where possible.
  - Have run VampirTrace as part of mpiotf offline collector
  - Same collector shared by Open|SpeedShop base tool
Other Future Plans

- Port Open|SpeedShop to other platforms
- Usability improvements from previous usability studies
- New experiments
  - Code coverage plugin – Javelina
  - mpiP
  - Memory tracing
Summary

- Support for wide range of experiments
  - Sampling (timing and hardware counters)
  - Tracing (MPI, I/O, FPE)

- Easy and flexible user access
  - GUI with Wizards
  - Scripting and batch processing

- Plugin infrastructure to extend functionality

- Set of Performance Tools with a flexible framework for additional tool creation
Availability and Contact

Open|SpeedShop website:  

Feedback

- Bug tracking available from website
- Contact information on website
- Email: oss-questions@openspeedshop.org
Questions?

Jim Galarowicz
jeg@krellinst.org

Krell Institute
http://www.krellinst.org