From Open|SpeedShop to a Component Based Tool Framework

Jim Galarowicz
Don Maghrak
The Krell Institute

Overall Agenda

- Open|SpeedShop Project Update
- Component Based Tool Framework Project Introduction

Open|SpeedShop Agenda

- Open|SpeedShop Project Overview
- Current Features
- What has changed since last update
- What we are working on now
- Current Release and Status

Project Overview: What is Open|SpeedShop?

- **Comprehensive open source performance analysis framework**
  - Combining *Profiling and Tracing*
  - Common *workflow* for all experiments
  - Flexible *instrumentation* (dynamic and offline)
  - *Extensibility* through plugins
  - *GUI, CLI,* immediate *command* and *Python API* user interfaces
- **Partners**
  - DOE/NNSA Tri-Labs (LLNL, LANL, SNLs)
  - Krell Institute
  - Universities of Wisconsin and Maryland
  - ORNL

Project Overview: What is Open|SpeedShop?

- **What can Open|SpeedShop do for the user?**
  - *Give lightweight overview of where program spends time*
  - *Find hot call paths in user program and libraries*
  - *Give access to hardware counter event information*
  - *Trace calls to POSIX I/O functions, give timing, call paths, and optional info like: bytes read, file names…*
  - *Trace calls to MPI functions. give timing, call paths, and optional info like: source, destination ranks, …..*
  - *Help pinpoint numerical problem areas by tracking FPEs*
- ***Maps the performance information back to the source and displays source annotated with the performance information.***

Project Overview: What is Open|SpeedShop?

- Platforms supported currently:
  - **Linux Clusters** with x86, IA-64, Opteron, and EM64T
  - Ports to Linux: PPC, BlueGene, Cray-XT in progress
- Gather performance data on **unmodified application binaries**
  - **Where no shared library support build statically**
    - **Open|SpeedShop provides "osslink" script to help re-link our collector code into the application**

Terminology

- Concept of an *Experiment*
  - What to measure (metric) and what to analyze (appl.)
  - Experiment chosen by user
- *Experiment* consists of *Collectors* and *Views*
  - Collectors define specific performance data sources
    - Hardware counters
    - Tracing of certain routines
  - Views specify data aggregation and presentation
  - Multiple collectors per experiment possible

Sampling Experiments

- PC Sampling (pcsamp)
  - Record PC in user defined time intervals
  - Low overhead overview of time distribution
  - Good first step to find hot spots in program
- User Time (usertime)
  - PC Sampling + Call stacks for each sample
  - Provides inclusive & exclusive timing data
  - Find hot call paths in application
- Hardware Counters (hwc, hwctime)
  - Sample HWC overflow events
  - Access to data like cache and TLB misses

Tracing Experiments

- I/O Tracing (*io, iot*)
  - Record invocation of all POSIX I/O events
  - Provides I/O aggregate and individual timings
  - iot – Shows bytes read/written, etc. & event by event list
- MPI Tracing (*mpi, mpit, mpiotf*)
  - Record invocation of all MPI routines
  - Provides MPI aggregate and individual timings
  - mpit – Shows bytes transferred, ranks involved, etc. & event by event list
  - mpiotf – Writes open trace format files using vampirtrace under the hood.

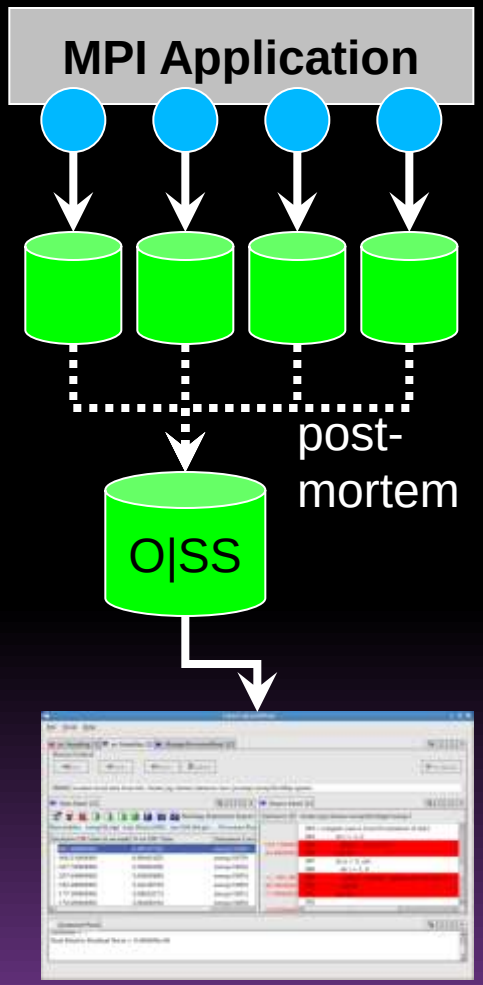
Tracing Experiments (2)

- Floating Point Exception Tracing (fpe)
  - Triggered by any FPE caused by the code
  - Helps pinpoint numerical problem areas
  - Mapped back to source where FPE occurred

Open|SpeedShop Performance Tool Status Update

Instrumentation Choices

Offline



Easy setup

Low overhead

No additional resources

Higher portability

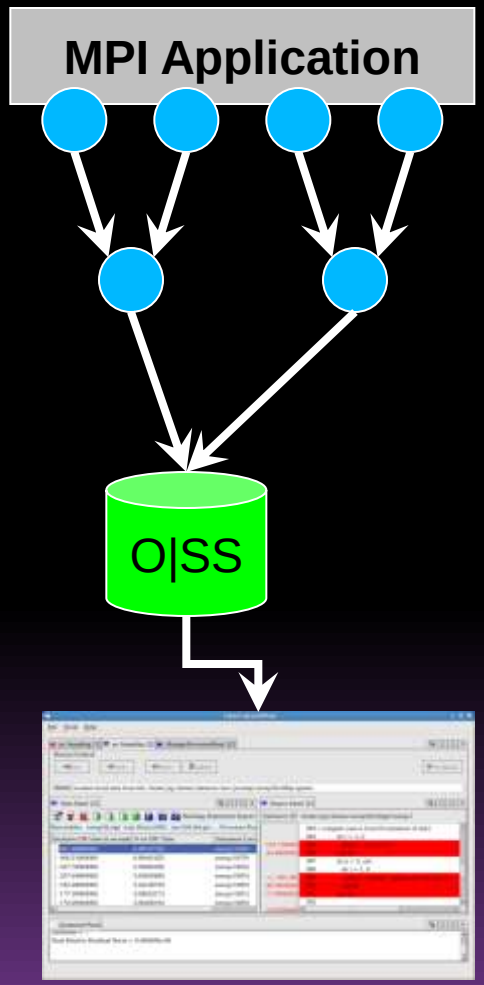
Online analysis

Intermediate updates

Attach to running code

Optional aggregation

MRNet



Open|SpeedShop Performance Tool Status Update

What is new since last Paradyn Week update in 2008?

- **Moved away from DPCL to MRNet as online transport**
- **Developed the offline mode of operation.**
  - *Using libmonitor (Rice) to hook into application, monitor sys calls*
  - *Link our collectors into the application to gather data*
  - *Write raw data files, then create OSS database file*
  - *Transitioned to having offline the default instrumentation mode*
  - *Low start-up overhead and works well in batch environments*
- *Continued to update the open source components we use*
  - *sqlite, libdwarf, libunwind, libmonitor, Dyninst, MRNet, PAPI,…*
- *Improved installation scripts, tools*

What is new since last Paradyn Week update in 2008? (2)

- **Usability Improvements**
  - Optional View window to select which metrics to be used to create the view
  - Ability to quickly switch to function, statement, or library view
  - Improvements (tool bar) for custom comparison view
  - Integrated offline mode support into GUI wizards
  - Created offline convenience scripts to hide the previous syntax
    - osspcsamp, ossusertime, osshwc, osshwctime, ossio, ….
- **In general, tool is more robust. Has been exposed to more applications, compilers, job schedulers, MPI versions.**

What are we working on now?

- **Work on selected modularization of Open|SpeedShop**
  - Ability to build a viewer only version
  - Ability to build only the runtime libraries and collectors
  - **Refactor runtime library component to be more modular**
- **Porting Open|SpeedShop:**
  - Linux PPC
  - BG/L and BG/P
  - CNL: Cray-XT4 and Cray-XT5
- **Supporting current users and assisting new users**
  - Release updates
  - New features and bug fixes to existing code

What are we working on now?

- **Scalability Improvements**
- **Integrate the latest versions of MRNet and Dyninst into Open|SpeedShop (CBTF project)**
  - *Using Dyninst-6.1 and MRNet 2.2 beta for development*
  - *More on this later in the talk*
- **Component Based Tool Framework project**
  - *Subject of next half of this talk*

Current Release and Status

- **Open|SpeedShop 1.9.3.3 available**
  - *Packages and source from sourceforge.net*
  - *Tested on a variety of platforms*
- **Cray-XT, BG, and PPC versions coming soon**
- **Open|SpeedShop website:**

  http://www.openspeedshop.org/
- **Download options:**
  - *Package with Install Script (install.sh or install-oss)*
  - *Source for tool and base libraries*

Building a Community Infrastructure for Scalable On-Line Performance Analysis Tools

Component Based Tool Framework "CBTF"

Jim Galarowicz
Don Maghrak
The Krell Institute

CBTF Agenda

- Project Origin and Team
- Project Rationale
- Project Goals/Objectives
- Research Challenges/Project Requirements
- Performance Tools Pipeline
- Project Results/Outcomes
- Current Status

Project Origin

- *Project Origin*
  - **OASCR Proposal:** *"Building a Community Infrastructure for Scalable On-Line Performance Analysis Tools Around Open SpeedShop"* **for** *Software Development Tools for Improved Ease-of-Use of Petascale Systems*
  - **Jointly funded by OASCR and NNSA**
  - *Three year project*

Project Team

- **Project Team**
  - The Krell Institute
  - University of Maryland
  - University of Wisconsin
  - Oak Ridge National Laboratory
  - Lawrence Livermore National Laboratory
  - Los Alamos National Laboratory
  - Sandia National Laboratories
  - Carnegie Mellon University
  - Others welcome……

Project Rationale

- **Why the need for the project?**
  - *Petascale environments need tool sets that are flexible*
  - *Need to quickly create new and specialized tools*
  - *Better availability of tools across more platforms*
  - *Need to avoid creating stove pipe tools*

Project Goals/Objectives

- **Project Goals/Objectives:**
  - *Create a toolbox of components for building high-level end user tools and/or quickly build tool prototypes.*
  - *Tools should be easily configurable/adjustable w/o rebuilding.*
  - *Able to mix components from several groups and/or vendors. Everyone should be able to contribute and use the new components.*
  - *We would like contributors to define the interfaces with us so that we can share components later in both directions.*

Project Goals/Objectives

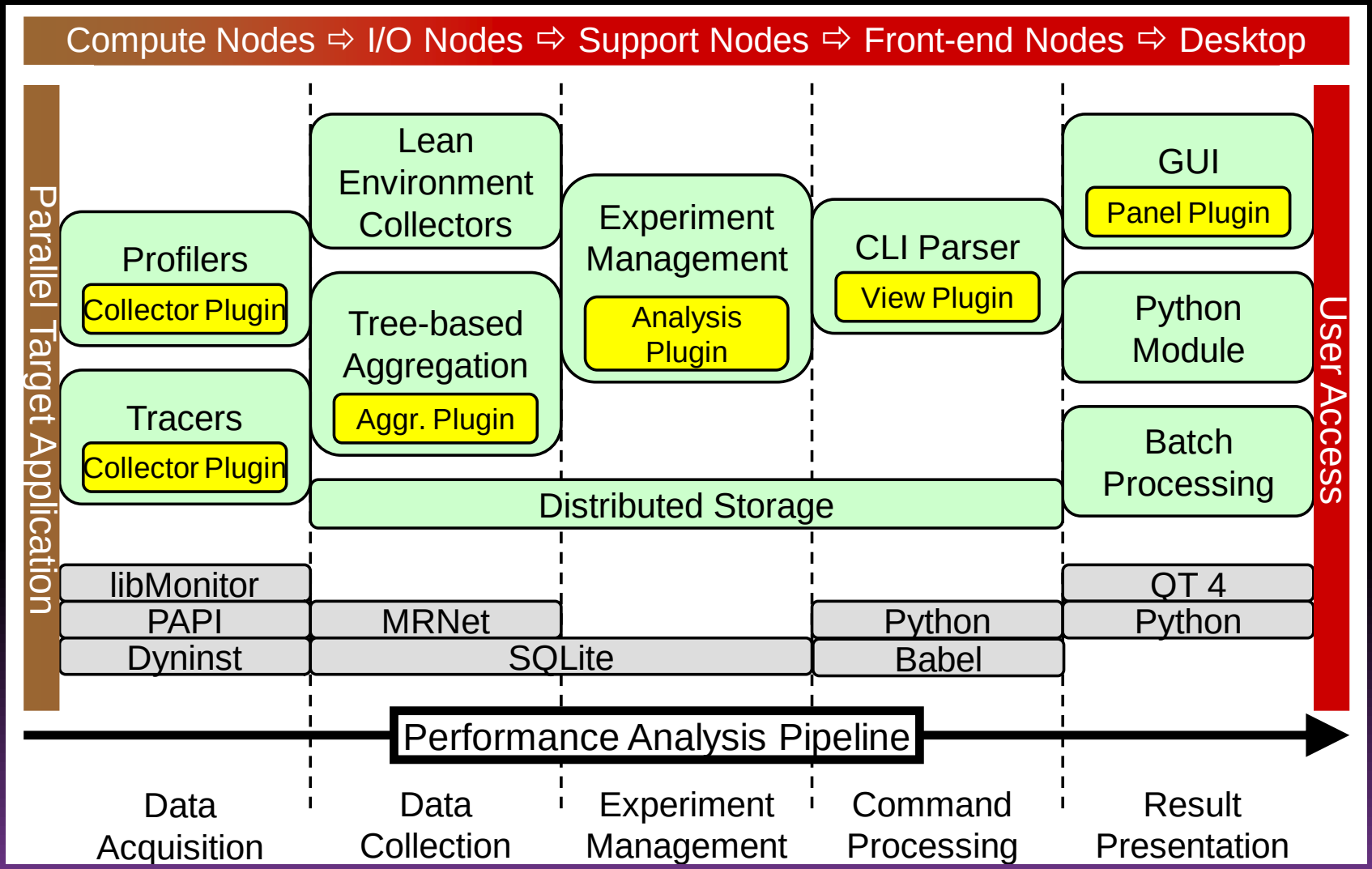
- **Project Goals/Objectives:**
  - *Research into efficient and effective online data aggregation, reduction, filtering, and data transfer*
    - *MRNet*
  - *Research lightweight data acquisition techniques*
    - *Binary rewriting*
  - *Assemble new tool components to create a more modular Open|SpeedShop performance tool*
  - *Support BlueGene and Cray-XT platforms*

Research Challenges Project Requirements

- **Research Challenges/Project Requirements:**
  - *Components must be designed for scale but also have a need for generality.*
  - *Support specialized tool components intended for serial or small scale usage.*
  - *Infrastructure must support online data aggregation because of potentially high data volume at scale.*
  - *Petascale machines are likely to have limited OS capabilities requiring new and light-weight data acquisition techniques.*
  - *Must be able to efficiently store the performance data.*
  - *Must be able to map any combination of tool components to the target architecture.*

Building a Community Infrastructure for Scalable On-Line Performance Analysis Tools

Performance Analysis Pipeline



Performance Tools Pipeline

- **Creating a first Performance Tools Pipeline prototype**
  - *Start with Open|SpeedShop components as one set of examples for such an infrastructure.*
  - *Decompose core components into general building blocks.*
  - *Arrange building blocks into a logical performance analysis pipeline.*
    - *Allows users and tool builders to select individual components for each pipeline stage.*
    - *Supports a flexible mapping onto the target architecture which provides efficient execution and visualization (incl. remote operation) environments.*

Project Results/Deliverables

- **Project Results/Deliverables:**
  - *Set of reusable components for creating performance tools*
  - *Modified version of gprof using reusable components*
  - *Components for online data aggregation, reduction, filtering, and transfer at high scale*
  - *Tool or Open|SpeedShop experiment based on Active Harmony*
  - *A new, more modular Open|SpeedShop performance tool*
  - *Support for BlueGene and Cray-XT platforms*
  - *Special purpose tool, based on need at ORNL*

Project Results/Outcomes

Dyninst/MRNet *Features/Requirements/Desires*

- *Plan to use symtabAPI*
- *Plan to be using the MRNet lightweight library*
- *Plan to use the detach on the fly feature*
- *Plan to use the binary rewriter feature*
- *Would like a floating point register fix up feature*
- *Plan to use the "1st party" stackwalker API*
- *Plan to create an "new" OSS feature based on Active Harmony*
- *Plan to use MRNet transport mechanism*

Current Status

- **Current Status**
  - Open|SpeedShop team design meetings
  - Holding extended CBTF team meetings to discuss ideas for component interfaces
  - Created a CBTF wiki
  - Started prototyping the component interface design
  - Doing a number of improvements and decompositions in Open|SpeedShop in preparation to move to CBTF
  - Plan to focus on transport components first

Questions?

Questions?

jeg@krellinst.org

dpm@krellinst.org

Open|SpeedShop Appendix

Open|SpeedShop Appendix

www.openspeedshop.org

Running in Offline mode pcsamp example

- **osspcsamp "<executable> <arguments>"**
  - One line command to gather PC Sampling results
  - Note: "" around executable line
  - Run command without extra arguments for help or view man page
- **Separate command for each experiment**
  - osspcsamp, ossusertime, osshwc, osshwctime
  - ossio, ossiot, ossmpi, ossmpit, ossfpe
- **Example Sequential run: (example run in following slides)**
  - osspcsamp "./smg2000 –n 80 80 80"
- **Example multi-process run:**
  - ossmpi "mpirun -np 64 sweep3d.mpi"

Example Offline Run With Output

osspcsamp "./smg2000 –n 80 80 80"

jeg@localhost:~

File Edit View Terminal Help

```
[jeg@localhost test]$ osspcsamp "./smg2000 -n 80 80 80"
[openss]: pcsamp experiment using the pcsamp experiment default sampling rate: "100".
[openss]: Using OPENSS_PREFIX installed in /opt/OSS-mrnet
[openss]: Setting up offline raw data directory in /tmp/jeg/offline-oss
[openss]: Running offline pcsamp experiment using the command:
"./smg2000 -n 80 80 80"

Running with these driver parameters:
  (nx, ny, nz)    = (80, 80, 80)
  (Px, Py, Pz)    = (1, 1, 1)
  (bx, by, bz)    = (1, 1, 1)
  (cx, cy, cz)    = (1.000000, 1.000000, 1.000000)
  (n_pre, n_post) = (1, 1)
  dim             = 3
  solver ID       = 0
=============================================
Struct Interface:
=============================================
Struct Interface:
  wall clock time = 0.080000 seconds
  cpu clock time  = 0.060000 seconds
=============================================
Setup phase times:
=============================================
SMG Setup:
  wall clock time = 0.680000 seconds
  cpu clock time  = 0.660000 seconds
=============================================
Solve phase times:
=============================================
SMG Solve:
  wall clock time = 4.810000 seconds
  cpu clock time  = 4.800000 seconds

Iterations = 7
Final Relative Residual Norm = 2.844100e-07


[openss]: Converting raw data from /tmp/jeg/offline-oss into temp file X.0.openss

Processing raw data for smg2000
Processing processes and threads ...
Processing performance data ...
Processing functions and statements ...
[openss]: Restoring and displaying default view for:
        /home/jeg/DEMOS/demos/sequential/smg2000/test/smg2000-pcsamp.openss
[openss]: The restored experiment identifier is:  -x 1
                                                                    46,0-1        Top
```

Application output followed by OSS output

Example Offline Run With Output (2)

osspcsamp "./smg2000 –n 80 80 80"

jeg@localhost:~/DEMOS/demos/sequential/smg2000/test

File Edit View Terminal Help

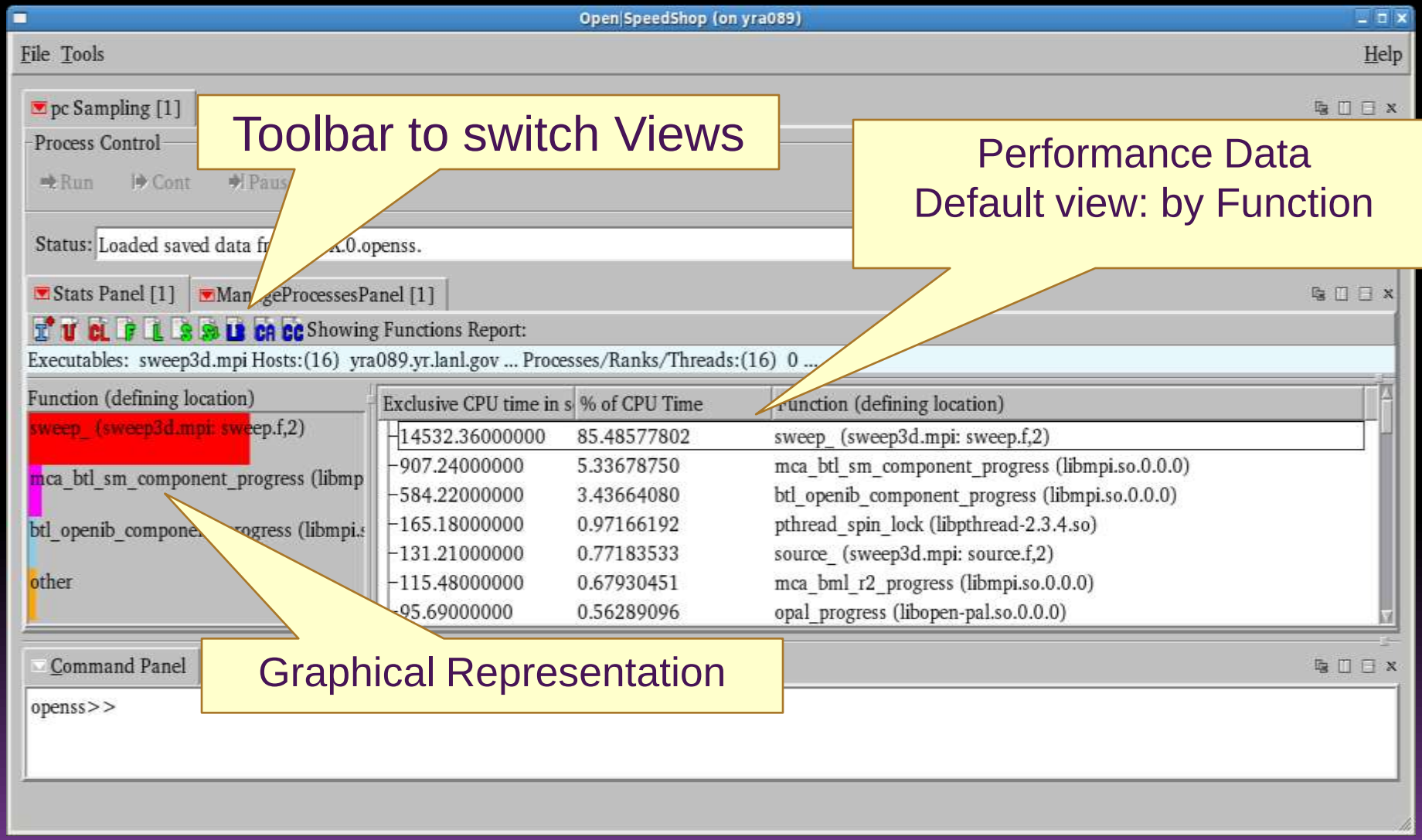
```
Solve phase times:
=================================================
SMG Solve:
  wall clock time = 4.810000 seconds
  cpu clock time  = 4.800000 seconds

Iterations = 7
Final Relative Residual Norm = 2.844100e-07


[openss]: Converting raw data from /tmp/jeg/offline-oss into temp fi

Processing raw data for smg2000
Processing processes and threads ...
Processing performance data ...
Processing functions and statements ...

[openss]: Restoring and displaying default view for:
        /home/jeg/DEMOS/demos/sequential/smg2000/test/smg2000-pcsamp.openss
[openss]: The restored experiment identifier is:  -x 1

  Exclusive CPU time          % of CPU Time  Function (defining location)
         in seconds.
            2.680000              53.174603  hypre_SMGResidual (smg2000: smg_residual.c,152)
            1.660000              32.936508  hypre_CyclicReduction (smg2000: cyclic_reduction.c,757)
            0.110000               2.182540  hypre_SemiInterp (smg2000: semi_interp.c,126)
            0.100000               1.984127  hypre_SemiRestrict (smg2000: semi_restrict.c,125)
            0.080000               1.587302  hypre_SMG2BuildRAPSym (smg2000: smg2_setup_rap.c,156)
            0.050000               0.992063  hypre_SMGAxpy (smg2000: smg_axpy.c,27)
            0.050000               0.992063  hypre_SMG3BuildRAPSym (smg2000: smg3_setup_rap.c,233)
            0.040000               0.793651  hypre_StructVectorClearGhostValues (smg2000: struct_vector.c,592)
            0.040000               0.793651  hypre_StructAxpy (smg2000: struct_axpy.c,25)
            0.030000               0.595238  hypre_StructMatrixInitializeData (smg2000: struct_matrix.c,314)
            0.030000               0.595238  hypre_StructVectorSetConstantValues (smg2000: struct_vector.c,537)
            0.020000               0.396825  hypre_CycRedSetupCoarseOp (smg2000: cyclic_reduction.c,211)
            0.020000               0.396825  hypre_StructInnerProd (smg2000: struct_innerprod.c,32)
            0.020000               0.396825  hypre_StructMatrixSetBoxValues (smg2000: struct_matrix.c,458)
            0.020000               0.396825  main (smg2000: smg2000.c,21)
            0.010000               0.198413  hypre_CreateCommInfoFromStencil (smg2000: communication_info.c,58)
            0.010000               0.198413  _int_malloc (libc-2.10.2.so: malloc.c,0)
            0.010000               0.198413  hypre_CommTypeDestroy (smg2000: communication.c,826)
            0.010000               0.198413  __brk (libc-2.10.2.so)
            0.010000               0.198413  hypre_StructVectorSetBoxValues (smg2000: struct_vector.c,258)
            0.010000               0.198413  hypre_BoxArrayArrayCreate (smg2000: box.c,106)
            0.010000               0.198413  hypre_SMGSetupInterpOp (smg2000: smg_setup_interp.c,88)
            0.010000               0.198413  hypre_SMGRelaxSetup (smg2000: smg_relax.c,357)
            0.010000               0.198413  hypre_SMGSetStructVectorConstantValues (smg2000: smg.c,379)


[jeg@localhost test]$
```

Performance Data
Default view: by Function

Offline Experiment Outputs

- **Outputs from: osspcsamp "executable"**
  - Normal program output while executable is running
  - The sorted list of performance information
    - A list of the functions taking the most time
    - The corresponding sample derived time for each function
  - A performance information database file (.openss file)
    - The database file contains all the information needed to view the data at anytime in the future without the executable(s).
      - Symbol table information from executable(s) and system libraries
      - Performance data openss gathered
      - Time stamps for when dynamic shared libraries were loaded and unloaded

Default GUI View



Open|SpeedShop Performance Tool Status Update

Associate Source and Performance Data