Open | SpeedShop™

Understanding Performance of Parallel Codes Using Open|SpeedShop

SEA@UCAR – Boulder, CO - April 3, 2013

Jim Galarowicz : Krell Institute

Martin Schulz : LLNL

Open|SpeedShop Overview

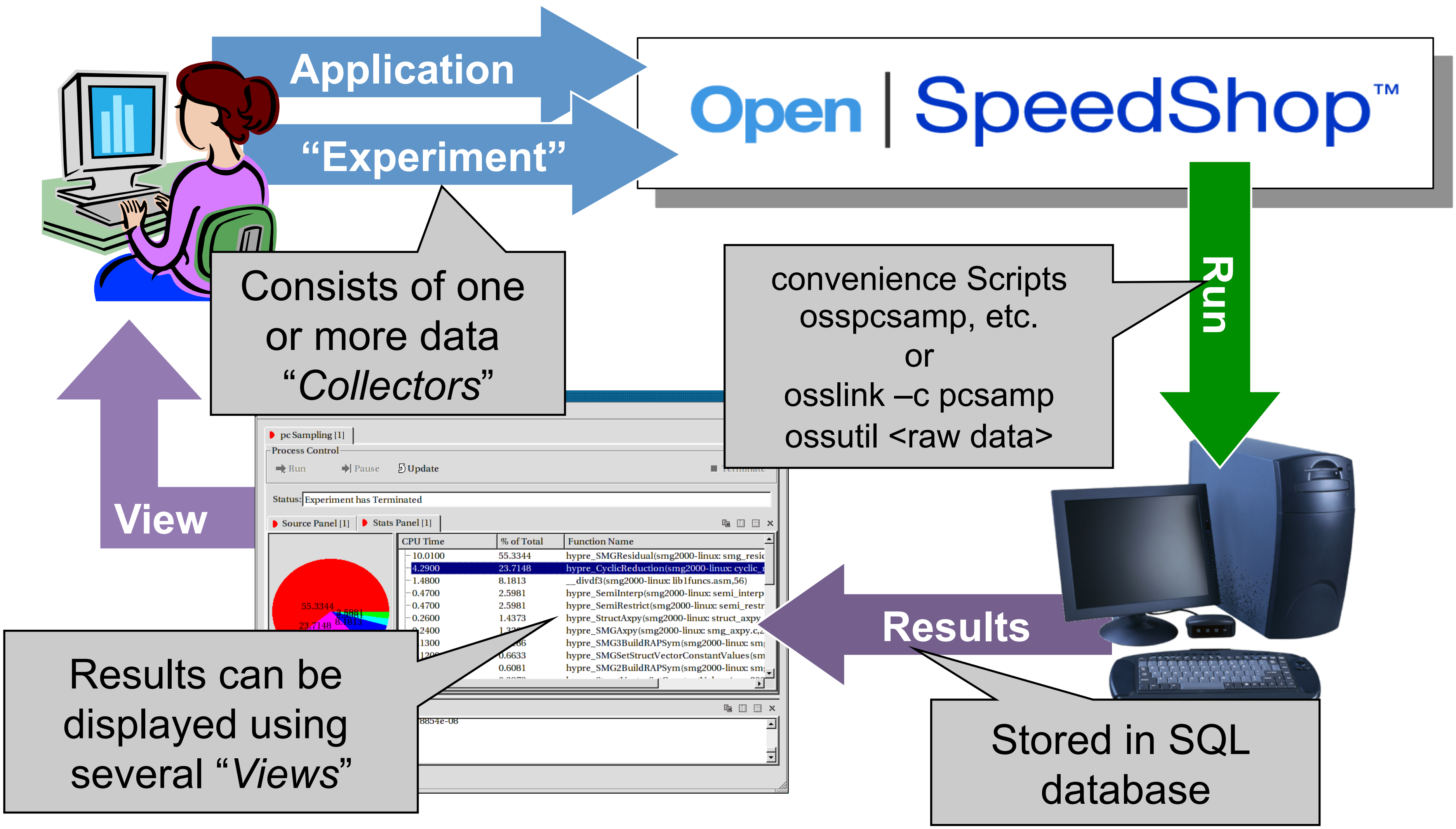
- **Open Source Performance Analysis Tool Framework**
  - Most common performance analysis steps ***all in one tool***
  - Gathers and displays several types of performance information
    - Profiling options to get initial overview
    - Detailed analysis views to drill down to locate bottlenecks
    - Event tracing options to capture performance details
  - Various analysis options for comparisons, load balance, …
- **Flexible and Easy to use**
  - User access through ***GUI***, ***Command Line***, ***Python Scripting, and convenience scripts.***
  - Gather performance data from ***unmodified application binaries***
- **Supports a wide range of systems**
  - Extensively used and tested on a variety of ***Linux clusters***
    - In preparation for SEA adapted to IBM's cluster software
  - ***Cray XT/XE/XK*** and ***Blue Gene P/Q*** support

O|SS Team and History

- **Jim Galarowicz, Krell**
- **Martin Schulz, LLNL**
- **Larger team**
  - Don Maghrak, Krell
  - William Hachfeld,Dave Whitney, Krell
  - Dane Gardner, Argo Navis/Krell
  - Matt Legendre, LLNL
  - Jennifer Green, Philip Romero, LANL
  - David Montoya, David Gunther, Michael Mason, LANL
  - Mahesh Rajan, Anthony Agelastos, SNLs
  - Dyninst group (Bart Miller, UW & Jeff Hollingsworth, UMD)
  - Phil Roth, Michael Brim, ORNL
- **Project history:**
  - Started at SGI in 2004
  - 6-8 developers for two years at SGI
  - Krell and Tri-labs agreement and funding since late 2006
  - Office of Science and DOE STTR and SBIR funding over the time period since 2006
  - Tool available as open source, support through maintenance contracts with Krell

Experiment Workflow

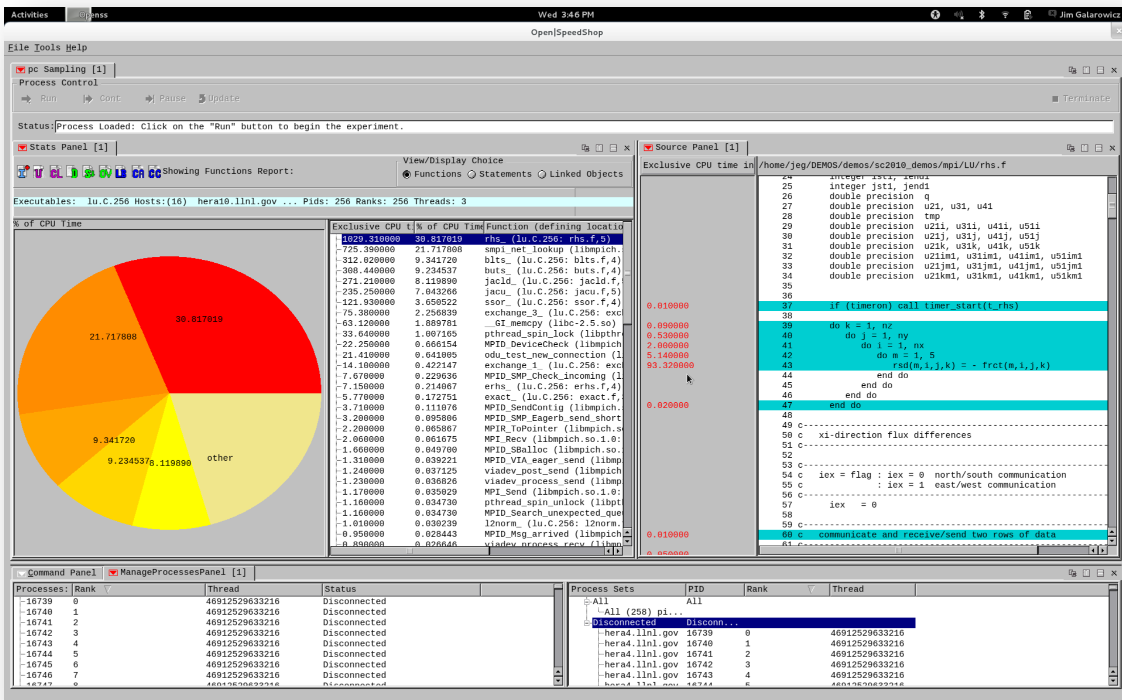
Open|SpeedShop Workflow

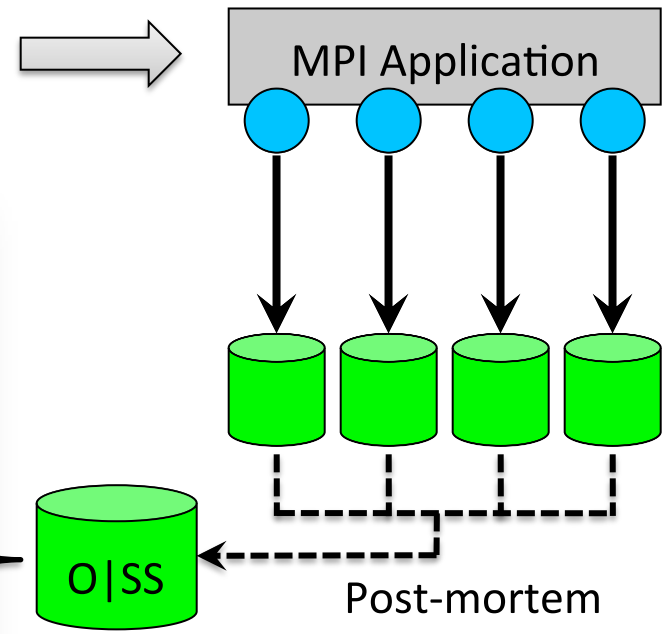


A First Simple Example

- ❖ **Using PC Sampling experiments**
  - ➢ Good initial experiment that shows where time is spent

>> osspcsamp “mpirun –np 2 smg2000 –n 65 65 65”





Installation / Usage on Yellowstone

- **Installed in central location**
  - Use:
    - module use /glade/u/home/galaro/privatemodules
    - module load openss
  - Adds GUI and all scripts into the path
- **Starting an MPI job through LSF**
  - Using mpirun.lsf <binary> from within a batch script
- **A few things to consider**
  - Location of raw data files
    - Environment variable: OPENSS_RAWDATA_DIR
    - On yellowstone by default set to: */glade/scratch/*${USER}
  - Additional environment variables or arguments to convenience scripts can control each experiment
    - Sampling rates, types of counters, subsets of functions to be traced
    - More on environment variables in the tutorial

Example Run with Output (Startup)

❖ osspcsamp "mpirun –np 2 smg2000 –n 65 65 65"

```
osspcsamp "mpirun -np 2 ./smg2000 -n 65 65 65"
[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 /opt/shared/jeg/offline-oss
[openss]: Running offline pcsamp experiment using the command:
"mpirun -np 2 /opt/OSS-mrnet/bin/ossrun "./smg2000 -n 65 65 65" pcsamp"

Running with these driver parameters:
 (nx, ny, nz)    = (65, 65, 65)
 (Px, Py, Pz)    = (2, 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 = 0.049847 seconds
 cpu clock time  = 0.050000 seconds
```

Example Run with Output (App. term.)

- **osspcsamp “mpirun –np 2 smg2000 –n 65 65 65”**

```
============================================
Setup phase times:
============================================
SMG Setup:
 wall clock time = 0.635208 seconds
 cpu clock time  = 0.630000 seconds
============================================
Solve phase times:
============================================
SMG Solve:
 wall clock time = 3.987212 seconds
 cpu clock time  = 3.970000 seconds
Iterations = 7
Final Relative Residual Norm = 1.774415e-07
[openss]: Converting raw data from /opt/shared/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 ...
```

Example Run with Output (Results)

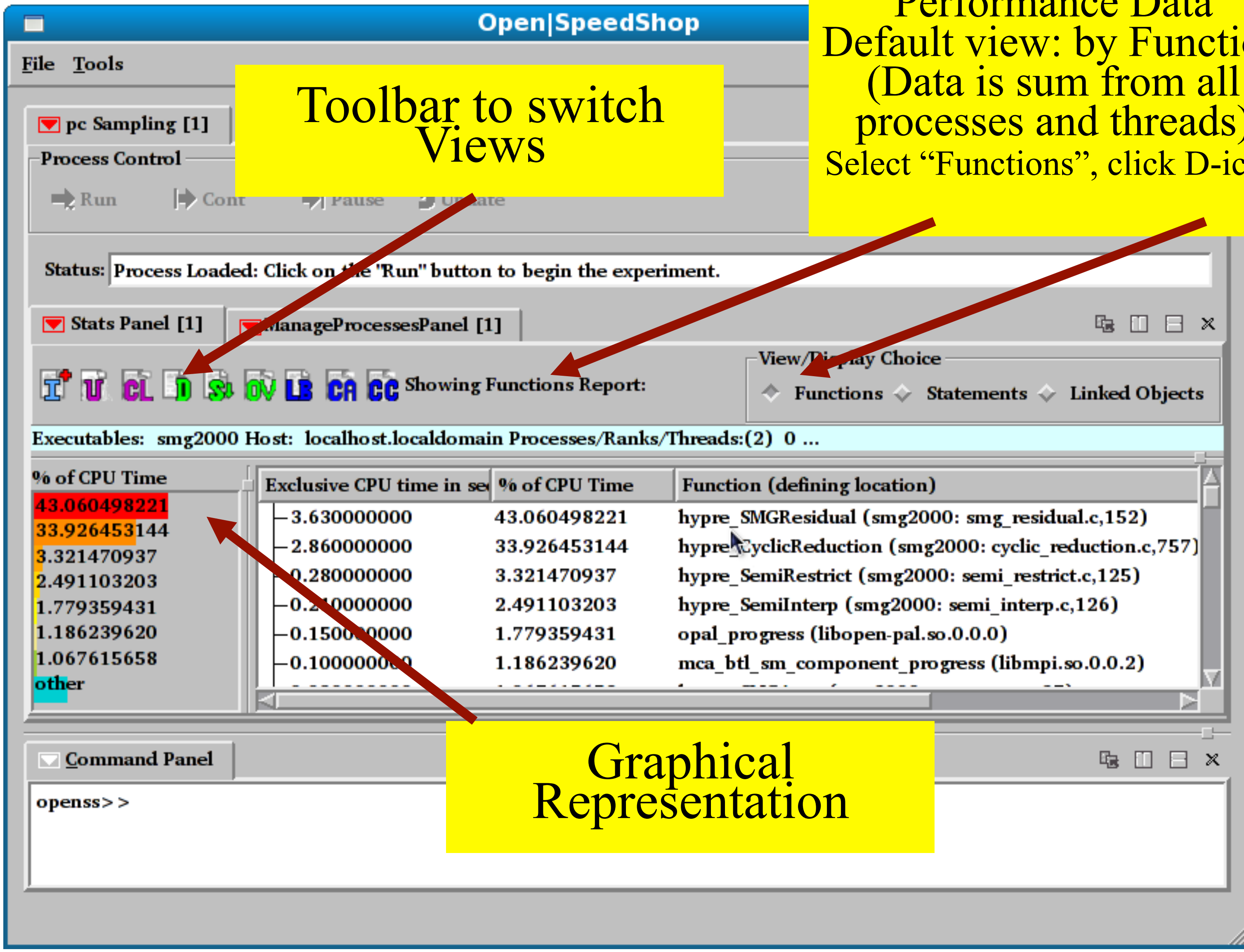
❖ **osspcsamp "mpirun –np 2 smg2000 –n 65 65 65"**

[openss]: Restoring and displaying default view for:
/home/jeg/DEMOS/demos/mpi/openmpi-1.4.2/smg2000/test/smg2000-pcsamp-1.openss
[openss]: The restored experiment identifier is: -x 1

Exclusive CPU time in seconds.	% of CPU Time	Function (defining location)
3.630000000	43.060498221	hypre_SMGResidual (smg2000: smg_residual.c,152)
2.860000000	33.926453144	hypre_CyclicReduction (smg2000: cyclic_reduction.c,757)
0.280000000	3.321470937	hypre_SemiRestrict (smg2000: semi_restrict.c,125)
0.210000000	2.491103203	hypre_SemiInterp (smg2000: semi_interp.c,126)
0.150000000	1.779359431	opal_progress (libopen-pal.so.0.0.0)
0.100000000	1.186239620	mca_btl_sm_component_progress (libmpi.so.0.0.2)
0.090000000	1.067615658	hypre_SMGAxpy (smg2000: smg_axpy.c,27)
0.080000000	0.948991696	ompi_generic_simple_pack (libmpi.so.0.0.2)
0.070000000	0.830367734	__GI_memcpy (libc-2.10.2.so)
0.070000000	0.830367734	hypre_StructVectorSetConstantValues (smg2000: struct_vector.c,537)
0.060000000	0.711743772	hypre_SMG3BuildRAPSym (smg2000: smg3_setup_rap.c,233)

❖ View with GUI: openss –f smg2000-pcsamp-1.openss

Default Output Report View

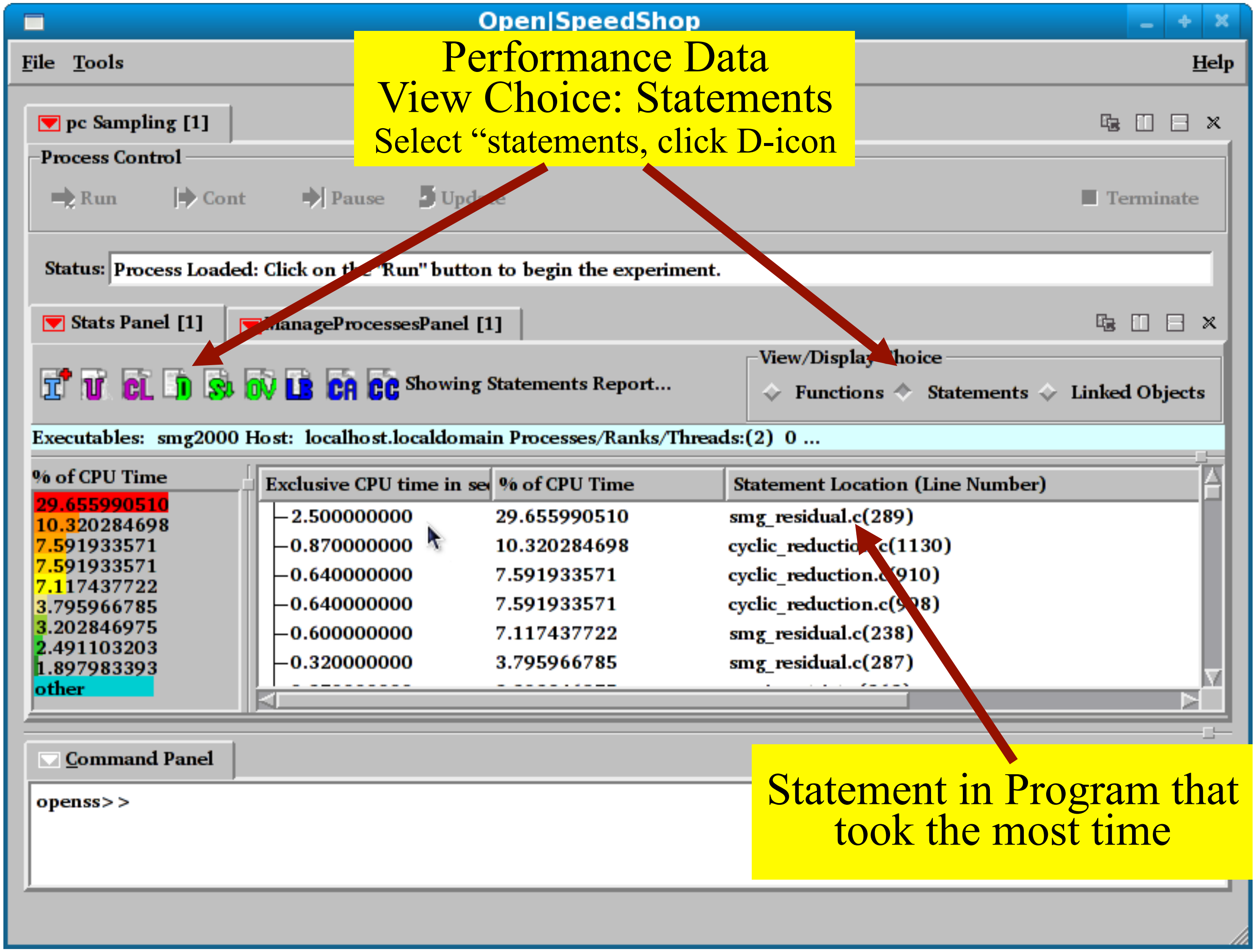


Toolbar to switch Views

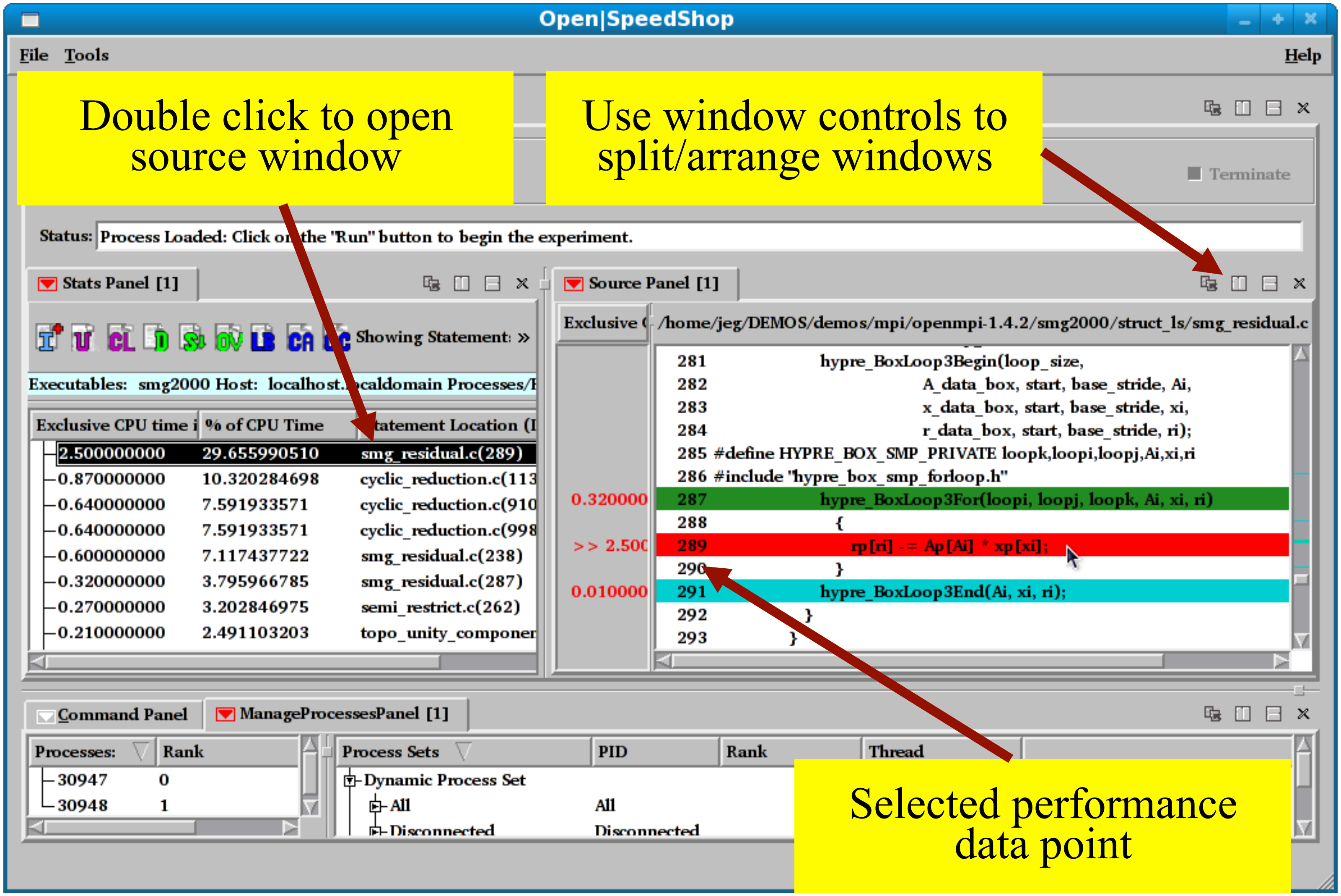
Performance Data
Default view: by Function
(Data is sum from all processes and threads)
Select "Functions", click D-icon

Graphical Representation

Statement Report Output View

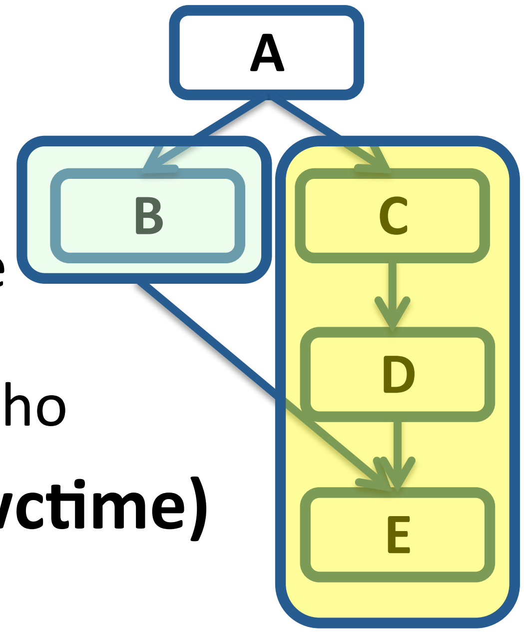


Associate Source & Performance Data



Available Experiments I (Profiling)

- **PC Sampling (pcsamp)**
  - Record PC in user defined time intervals
  - Low overhead overview of time distribution
  - Good first step, lightweight overview
- **Call Path Profiling (usertime)**
  - PC Sampling and Call stacks for each sample
  - Provides inclusive and exclusive timing data
  - Use to find hot call paths, whom is calling who



- **Hardware Counters (hwcsamp, hwc, hwctime)**
  - Access to data like cache and TLB misses
  - hwcsamp:
    - Sample up to six events based on a sample time (hwcsamp)
    - Default events are PAPI_FP_OPS and PAPI_TOT_CYC
    - Good overview of hardware counter event counts
  - hwc, hwctime:
    - Sample a HWC event based on an event threshold
    - Default event is PAPI_TOT_CYC overflows

Available Experiments II (Tracing)

- **Input/Output Tracing (io, iot)**
  - Record invocation of all POSIX I/O events
  - Provides aggregate and individual timings
  - Provide argument information for each call (iot)
- **MPI Tracing (mpi, mpit)**
  - Record invocation of all MPI routines
  - Provides aggregate and individual timings
  - Provide argument information for each call (mpit)
  - Optional experiments to create OTF/Vampir files
- **Floating Point Exception Tracing (fpe)**
  - Triggered by any FPE caused by the application
  - Helps pinpoint numerical problem areas

More details and examples on all experiments during the O|SS tutorial in the coming days

Future Experiments by End of 2013

- **New Open|SpeedShop experiments under construction**
  - **Lightweight I/O experiment (iop)**
    - Profile I/O functions by recording individual call paths
      - Rather than every individual event with the event call path, (**io** and **iot**).
      - More opportunity for aggregation and smaller database files
    - Map performance information back to the application source code.
  - **Memory analysis experiment (mem)**
    - Record and track memory consumption information
      - How much memory was used – high water mark
      - Map performance information back to the application source code
  - **Threading analysis experiment (thread)**
    - Report statistics about pthread wait times
    - Report OpenMP (OMP) blocking times
    - Attribute gathered performance information to proper threads
    - Thread identification improvements
      - Use a simple integer alias for POSIX thread identifier
    - Report synchronization overhead mapped to proper thread
    - Map performance information back to the application source code

Analysis of Parallel Codes

- **By default experiment collectors are run on all tasks**
  - Automatically detect all processes and threads
  - Gathers and stores per thread data
- **Viewing data from parallel codes**
  - By default all values aggregated (summed) across all ranks
  - Manually include/exclude individual ranks/processes/threads
  - Ability to compare ranks/threads
- **Additional analysis options**
  - Load Balance (min, max, average) across parallel executions
    - Across ranks for hybrid openMP/MPI codes
    - Focus on a single rank to see load balance across OpenMP threads
  - Cluster analysis (finding outliers)
    - Automatically creates groups of similar performing ranks or threads
    - Available from the Stats Panel toolbar or context menu

Alternative Interfaces

- **Scripting language**
  - Immediate command interface
  - O|SS interactive command line (CLI)

```
Experiment Commands
    expAttach
    expCreate
    expDetach
    expGo
    expView

List Commands
    list –v exp
```

- **Python module**

```
import openss

my_filename=openss.FileList("myprog.a.out")
my_exptype=openss.ExpTypeList("pcsamp")
my_id=openss.expCreate(my_filename,my_exptype)

openss.expGo()

My_metric_list = openss.MetricList("exclusive")
my_viewtype = openss.ViewTypeList("pcsamp”)
result = openss.expView(my_id,my_viewtype,my_metric_list)
```

CLI Example: sweep3d I/O Experiment

openss –cli –f sweep3d-io.openss

openss>>expview

```
Exclusive      % of       Number  Function (defining location)
I/O Call     Total         of
Time(ms)                  Calls
18.01600  77.255575     36  write (sweep3d-io)
2.364000  10.137221      2  open64 (sweep3d-io)
1.307000   5.604631      2  read (sweep3d-io)
1.040000   4.459691     72  __lseek64 (sweep3d-io)
0.593000   2.542882      2  __close (sweep3d-io)
```

openss>>expview -vcalltrees,fullstack io1

```
Exclusive      % of  Number  Call Stack Function (defining location)
I/O Call     Total     of
Time(ms)             Calls
                            __libc_start_main (sweep3d-io: libc-start.c,118)
                            > @ 226 in generic_start_main (sweep3d-io: libc-start.c,96)
                            >>__wrap_main (sweep3d-io)
                            >>>monitor_main (sweep3d-io)
                            >>>> @ 21 in main (sweep3d-io: fmain.c,11)
                            >>>>> @ 184 in MAIN_ (sweep3d-io: driver.f,1)
                            >>>>>> @ 66 in inner_auto (sweep3d-io: inner_auto.f,2)
                            >>>>>>> @ 164 in inner (sweep3d-io: inner.f,2)
                            >>>>>>>> @ 3339 in _gfortran_st_write_done (sweep3d-io: transfer.c,3333)
                            >>>>>>>>> @ 3228 in finalize_transfer (sweep3d-io: transfer.c,3142)
                            >>>>>>>>>> @ 3132 in _gfortrani_next_record (sweep3d-io: transfer.c,3100)
                            >>>>>>>>>>> @ 70 in _gfortrani_fbuf_flush (sweep3d-io: fbuf.c,143)
                            >>>>>>>>>>>> @ 261 in raw_write (sweep3d-io: unix.c,250)
5.588000  23.962264     1     >>>>>>>>>>>>>write (sweep3d-io)
```

Comparing Performance Data

- **Comparisons: basic operation for performance analysis**
  - Compare performance before/after optimization
  - Track performance during code history
  - Compare ranks to each other
- **Open|SpeedShop enables flexible comparisons**
  - Within databases and across multiple databases
  - Within the same experiment and across experiments
- **Convenience Script: osscompare**
  - Compares Open|SpeedShop databases to each other
  - Compare up to 8 at one time
  - Produces side-by-side comparison listing
  - Optionally create "csv" output for input into spreadsheet (Excel,..)
    - export **OPENSS_CREATE_CSV**=1

Example: Comparison Results

osscompare "smg2000-pcsamp.openss,smg2000-pcsamp-1.openss”

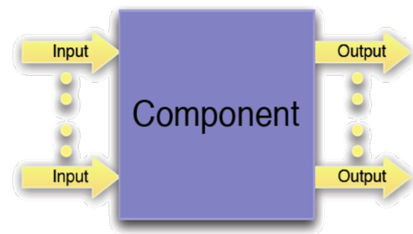
openss]: Legend: -c 2 represents smg2000-pcsamp.openss

[openss]: Legend: -c 4 represents smg2000-pcsamp-1.openss

-c 2, Exclusive CPU time in seconds.	-c 4, Exclusive CPU time in seconds.	Function (defining location)
3.870000000	3.630000000	hypre_SMGResidual (smg2000: smg_residual.c,152)
2.610000000	2.860000000	hypre_CyclicReduction (smg2000: cyclic_reduction.c,757)
2.030000000	0.150000000	opal_progress (libopen-pal.so.0.0.0)
1.330000000	0.100000000	mca_btl_sm_component_progress (libmpi.so.0.0.2: topo_unity_component.c,0)
0.280000000	0.210000000	hypre_SemiInterp (smg2000: semi_interp.c,126)
0.280000000	0.040000000	mca_pml_ob1_progress (libmpi.so.0.0.2: topo_unity_component.c, 0)

Scaling Open|SpeedShop

- ❖ Open|SpeedShop designed for traditional clusters
  - ➢ Tested and works well up to 1,000-10,000 cores
  - ➢ Scalability concerns on machines with 100,000+ cores
  - ➢ Target: ASC capability machines like LLNL's Sequoia (20 Pflop/s BG/Q)
- ❖ Component Based Tool Framework (CBTF)
  - ➢ http://ft.ornl.gov/doku/cbtfw/start
  - ➢ Based on tree based communication infrastructure
  - ➢ Porting O|SS on top of CBTF



- ❖ Improvements:
  - ➢ Direct streaming of performance data to tool without temp. I/O
  - ➢ Data will be filtered (reduced or combined) on the fly
  - ➢ Emphasis on scalable analysis techniques
- ❖ Initial prototype exists, working version: Mid-2013
  - ➢ Little changes for users of Open|SpeedShop
  - ➢ CBTF can be used to quickly create new tools
  - ➢ Additional option: use of CBTF in applications to collect data

Summary

- **Open Source Performance Analysis Tool Framework**
  - Most common performance analysis steps ***all in one tool***
  - Includes a set of experiments:
    - Profiling/Sampling experiments, like pcsamp, to get initial overview
    - Multiple views and analysis options (statement view, comparisons, …)
    - Event tracing options, like I/O tracing, to capture performance details
  - Special parallel analysis options: load balance & clustering
- **Flexible and Easy to use**
  - User access trough multiple interfaces
  - Convenience scripts make it easy to start, just run "oss<exp>"
  - Gather performance data from ***unmodified application binaries***
- **Supports a wide range of systems**
  - Extensively used and tested on a variety of ***Linux clusters***
    - Available on yellowstone
  - ***Cray XT/XE/XK*** and ***Blue Gene P/Q*** support

Open|SpeedShop Tutorial

- **Thursday**
  - Guided tutorial with hands-on exercises
    - Bring your own code or use one of our demos
  - Covers sequential and parallel codes
    - Basic profiling experiments, incl. hardware counters
    - MPI and I/O experiments
    - GUI and command line access
  - Access to yellowstone required (preferably with X connection)
- **Friday**
  - Open discussion and “bring your own code clinic”
    - General Q&A
    - Working on advanced features
    - Experimentation on and analysis of your own codes
  - If people are interested:
    - Help with installing O|SS on other systems beyond yellowstone
    - Introduction into CBTF

Availability and Contact

- **Current version: 2.0.2 update 8**
  - On yellowstone:
    - module use /glade/u/home/galaro/privatemodules
    - module load openss
- **Open|SpeedShop Website**
  - http://www.openspeedshop.org/
- **Open|SpeedShop Quick Start Guide:**
  - http://www.openspeedshop.org/wp/wp-content/uploads/2013/03/OSSQuickStartGuide2012.pdf.
- **Feedback**
  - Bug tracking available from website
  - Contact information on website
  - oss-questions@openspeedshop.org
  - Open|SpeedShop Forum
    - http://www.openspeedshop.org/forums/