Open|SpeedShop Quick Start Guide

Open|SpeedShop (O|SS) is an open source, multi-platform Linux performance tool for analysis of applications running on both single-node and large-scale IA64, IA32, EM64T, AMD64, IBM Power PC clusters, Cray XT/XE and IBM Blue Gene platforms. O|SS gathers and displays several types of information to aid in solving performance problems, including: program counter sampling for a quick overview of the applications performance, call path profiling to add caller/callee context and locate critical time consuming paths, access to the machine hardware counter information, input/output tracing for finding I/O performance problems, MPI function call tracing for MPI load imbalance detection, and floating point exception tracing. O|SS offers a command-line interface (CLI), a graphical user interface (GUI) and a python scripting API user interface.

ACCESS INFORMATION

The O|SS Website: http://www.openspeedshop.org
O|SS Documentation: http://www.openspeedshop.org/wp/documentation
O|SS Users Guide: http://www.openspeedshop.org/docs/users_guide/

To use O|SS, check with your system administrator to see if a module, dotkit, or softenv file for O|SS exists on your system. O|SS can be installed in user directories as no root access is needed. Visit the O|SS website and click on Build Information for install instructions.

Help email: oss-questions@openspeedshop.org
O|SS Forum: http://www.openspeedshop.org/forums

WHAT OPEN|SPEEDSHOP PRODUCES

O|SS monitors a running application from start to finish and gathers performance data (and symbolic information describing the application), saves it to a SQLite database file and generates a report. The symbolic information allows the performance data to be viewed on another system without needing the application to be present.

PERFORMANCE INFORMATION TYPES

O|SS provides the following options, called experiments, to do specific analyses.

Experiment	*Description*
pcsamp	Periodic sampling the program counters gives a low-overhead view of where the time is being spent in the user application.
usertime	Periodic sampling the call path allows the user to view inclusive and exclusive time spent in application routines. It also allows the user to see which routines called which routines. Several views are available, including the "hot" path.
hwc	Hardware events (including clock cycles, graduated instructions, i- and d-cache and TLB misses, floating-point operations) are counted at the machine instruction, source line and function levels.
hwcsamp	Similar to hwc, except that sampling is based on time, not PAPI event overflows. Also, up to six events may be sampled during the same experiment.
hwctime	Similar to hwc, except that call path sampling is also included.
io	Accumulated wall-clock durations of I/O system calls: read, readv, write, writev, open, close, dup, pipe, creat and others.
iot	Similar to io, except that more information is gathered, such as bytes moved, file names, etc.
mpi	Captures the time spent in and the number of times each MPI function is called. Trace format option displays the data for each call, showing its start and end times.
mpit	Records each MPI function call event with specific data for display using a GUI or a command line interface (CLI).
mpiotf	Write MPI calls trace to Open Trace Format (OTF) files to allow viewing with Vampir or converting to formats of other tools.
fpe	Find where each floating-point exception occurred. A trace collects each with its exception type and the call stack contents. These measurements are exact, not statistical.

SUGGESTED WORKFLOW

We recommend an **O|SS** workflow consisting of two phases. First, gathering the performance data using the convenience scripts. Then using the GUI or CLI to view the data.

CONVENIENCE SCRIPTS

Users are encouraged to use the convenience scripts that hide some of the underlying options for running experiments. The full command syntax can be found in the User's Guide. The script names correspond to the experiment types and are: **osspcsamp, ossusertime, osshwc, osshwcsamp, osshwctime, ossio, ossiot, ossmpi, ossmpit, ossmpiotf, ossfpe** plus an **osscompare** script.
Note: Make sure to set **OPENSS RAWDATA DIR (See KEY ENVIRONMENT VARIABLES** section for info).

When running Open|SpeedShop, use the same syntax that is used to run the application/executable outside of O|SS, but enclosed in quotes; e.g.,
Using an MPI with mpirun: **osspcsamp** "mpirun -np 512 ./smg2000"
Using SLURM/srun: **osspcsamp** "srun -N 64 -n 512 ./smg2000 -n 5 5 5"
Redirection to/from files inside quotes can be problematic, see convenience script "man" pages for more info.

REPORT AND DATABASE CREATION

Running the pcsamp experiment on the sequential program named mexe: **osspcsamp** mexe results in a default report and the creation of a SQLite database file mexe-pcsamp.openss in the current directory; the report:

CPU time	**% CPU Time**	**Function**
11.650	48.990	f3 (mexe: m.c, 24)
7.960	33.478	f2 (mexe: m.c,15)
4.150	17.451	f1 (mexe: m.c,6)
0.020	0.084	work(mexe:m.c,33)

To access alternative views in the GUI: **openss –f** mexe-pcsamp.openss loads the database file. Then use the GUI toolbar to select desired views; or, using the CLI: **openss –cli –f** mexe-pcsamp.openss to load the database file. Then use the **expview** command options for desired views.

CONVENIENCE SCRIPT DESCRIPTION

osscompare: Compare Database Files

osscompare "<db_file1>, < db_file2>[,<db_file>…]" [time | percent | <other metrics>] [rows=nn] [viewtype=functions|statements|linkedobjects]>[oname=<csv filename>]
Example: **osscompare** "smg-run1.openss,smg-run2.openss"
Type "man osscompare" for more details.

osspcsamp: Program Counter Experiment

osspcsamp "<command> < args>" **[high | low | default |** <sampling rate>]
Sequential job example:
osspcsamp "smg2000 –n 50 50 50"
Parallel job example:
osspcsamp "mpirun –np 128 smg2000 –n 50 50 50"
Additional arguments:
high: twice the default sampling rate (samples per second)
low: half the default sampling rate
default: default sampling rate is 100
<sampling rate>: integer value sampling rate

ossusertime: Call Path Experiment

ossusertime "<command> < args>" [**high | low | default** | <sampling rate>]
Sequential job example:
ossusertime "smg2000 –n 50 50 50"
Parallel job example:
ossusertime "mpirun –np 64 smg2000 –n 50 50 50"
Additional arguments:
high: twice the default sampling rate (samples per second)
low: half the default sampling rate
default: default sampling rate is 35
<sampling rate>: integer value sampling rate

osshwc, osshwctime: HWC Experiments

osshwc[time] "<command> < args>" **[default |** <PAPI_event> | <PAPI threshold> | <PAPI_event><PAPI threshold>]
Sequential job example:
osshwc[time] "smg2000 –n 50 50 50"
Parallel job example:
osshwc[time] "mpirun –np 128 smg2000 –n 50 50 50"
Additional arguments:
default: event (PAPI_TOT_CYC), threshold (10000)
<PAPI_event>: PAPI event name
<PAPI threshold>: PAPI integer threshold

osshwcsamp: HWC Experiment

osshwcsamp "<command>< args>" **[default |**<PAPI_event_list>|<sampling_rate>]
Sequential job example: **osshwcsamp** "smg2000"
Parallel job example:
osshwcsamp "mpirun –np 128 smg2000 –n 50 50 50"
Additional arguments:
default: events(PAPI_TOT_CYC and PAPI_FP_OPS), sampling_rate is 100
<PAPI_event_list>: Comma separated PAPI event list
<sampling_rate>:Integer value sampling rate

ossio, ossiot: I/O Experiments

ossio[t] "<command> < args>" [**default** | f_t_list]
Sequential job example:
ossio[t] "smg2000 –n 50 50 50"
Parallel job example:
ossio[t] "mpirun –np 128 smg2000 –n 50 50 50"
Additional arguments:
default: trace all I/O functions
< f_t_list>: Comma-separated list of I/O functions to trace, one or more of the following: **close, creat, creat64, dup, dup2, lseek, lseek64, open, open64, pipe, pread, pread64, pwrite, pwrite64, read, readv, write, and writev**

ossmpi, ossmpit: MPI Experiments

ossmpi[t] "<mpirun><mpiargs><command><args>" [**default** | f_t_list]
Parallel job example: **ossmpi[t]** "mpirun –np 128 smg2000 –n 50 50 50"
Additional arguments: **default:** trace all MPI functions
<f_t_list>: Comma-separated list of MPI functions to trace, consisting of zero or more of: **MPI_Allgather**, …. **MPI_Waitsome** and/or zero or more of the MPI group categories:

MPI Category	*Argument*
All MPI Functions	**all**
Collective Communicators	**collective_com**
Persistent Communicators	**persistent_com**
Synchronous Point to Point	**synchronous_p2p**
Asynchronous Point to Point	**asynchronous_p2p**
Process Topologies	**process_topologies**
Groups Contexts Communicators	**graphs_contexts_comms**
Environment	**environment**
Datatypes	**datatypes**

ossfpe: FP Exception Experiment

ossfpe "<command> < args>" [default | f_t_list]
Sequential job example: **ossfpe** "smg2000 –n 50 50 50"
Parallel job example: **ossfpe** "mpirun –np 128 smg2000 –n 50 50 50"
Additional arguments: **default:** trace all floating-point exceptions
<f_t_list>: Comma-separated list of exceptions to trace, consisting of one or more of: **inexact_result, division_by_zero, underflow, overflow, invalid_operation**

KEY ENVIRONMENT VARIABLES

OPENSS_ENABLE_MPI_PCONTROL
Activates the MPI_Pcontrol function recognition, otherwise MPI_Pcontrol function calls will be ignored by O|SS.

OPENSS_RAWDATA_DIR
Used on cluster systems where a /tmp file system is unique on each node. It specifies the location of a shared file system path which is required for O|SS to save the "raw" data files on distributed systems.
OPENSS_RAWDATA_DIR="shared file system path"
Example: export **OPENSS_RAWDATA_DIR**=/lustre4/fsys/userid

OPENSS_DB_DIR
Specifies the path to where O|SS will build the database file. On a file system without file locking enabled, the SQLite component cannot create the database file. This variable is used to specify a path to a file system with locking enabled for the database file creation. This usually occurs on lustre file systems that don't have locking enabled.
OPENSS_DB_DIR="file system path"
Example: export **OPENSS_DB_DIR**=/opt/filesys/userid

OPENSS_MPI_IMPLEMENTATION
Specifies the MPI implementation in use by the application; only needed for the mpi, mpit, and mpiotf experiments. These are the currently supported MPI implementations: **openmpi, lampi, mpich, mpich2, mpt, lam, mvapich, mvapich2.** For Cray, IBM, Intel MPI implementations, use **mpich2.**
OPENSS_MPI_IMPLEMENTATION="MPI impl. name"
Example: export **OPENSS_MPI_IMPLEMENTATION**=openmpi

In most cases, O|SS can auto-detect the MPI in use.

INTERACTIVE COMMAND LINE USAGE

Simple Usage to Create, Run, View Data
The CLI can be used to run experiments interactively. To invoke O|SS in interactive mode use: **openss –cli**
An experiment can be created, run and viewed with three simple commands, e.g.:
expcreate –f "mexe 2000" pcsamp
expgo
expview

CLI Commands for Other Views
These interactive CLI commands may be used to view the performance data in alternative ways once an experiment has been run and the database file exists. The command: **openss –cli –f** <database-filename> loads the performance experiment. Then, the following commands may be used to view the performance information:

help or **help commands** : display CLI help text
expview : show the default view
expview –v statements : time-consuming statements
expview –v linkedobjects : time spent in libraries
expview –v calltrees,fullstack : all call paths
expview –m loadbalance : see load balance across ranks/threads/processes
expview –r <rank_num> : see data for specific rank(s)
expcompare –r 1 –r 2 –m time : compare rank 1 to rank 2 for metric equal time
list –v metrics : see optional performance data metrics
list –v src : see source files associated with experiment
list –v obj : see object files associated with experiment
list –v ranks : see ranks associated with experiment
list –v hosts : see machines associated with experiment
expview –m <metric from above> : see metric specified
expview –v calltrees,fullstack <experiment type> <number> : see expensive call paths.
For example: **expview –v calltrees,fullstack usertime2**
shows the top two call paths in execution time.
expview <experiment-name><number> shows the top time-consuming functions. For example:
expview pcsamp2 : shows the two functions taking the most time.
expview –v statements <experiment-name><number> shows the top time-consuming statements. For example: **expview –v statements pcsamp2** :shows the two statements taking the most time.

For more information about the Command Line Interface commands please visit:
http://www.openspeedshop.org/docs/cli_doc

GRAPHICAL USER INTERFACE USAGE
The GUI can be used to run experiments or to view and/or compare previously created performance database files. The two main commands used to invoke the GUI are:
openss: Open the GUI in wizard mode.
openss –f database_file.openss: open a previously created file. These commonly used commands are described in the sections below.

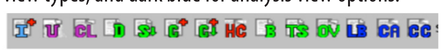
GUI WIZARD PANELS
When **openss** is used with no arguments, the first GUI wizard view shows three options: "Create New Data," "View a Previously Created Database File," or "Compare Two Database Files." If the first option is chosen, the second page of the wizard, "Generate New Performance Data," is shown. The view shows the first page of questions that a novice user can use to guide the creation and execution of an experiment. Subsequent wizard pages further define the experiment to be run, including selection of the application. Answer the wizard questions to create an experiment. On the first wizard page, the second and third options bring up dialog panels to choose the experiments to view or compare.

GUI Source Panel
The Source Panel displays the source used in creating the program that was run during the O|SS experiment. The source is annotated with performance information gathered while the experiment was run. Users can focus the source panel to the point of the performance bottleneck by clicking on the performance information displayed in the Statistics Panel. In order to see per statement statistics, build the application to be monitored with -g enabled.

GUI Statistics Panel
The GUI can also be used to directly view performance data from a previous experiment by opening its database file. For example: **openss –f smg2000.pcsamp.openss**

The GUI Statistics Panel view relates the performance data to the corresponding application source code. Clicking on an entry in the performance data panel focuses the source panel on the function or statement corresponding to the performance item.

The Statistics Panel toolbar icons allow alternative views of the performance data, and also built-in analysis views, e.g., load balance and outlier detection using cluster analysis. To aid in the selection of alternative views, a toolbar with icons corresponding to the views is provided. The icons are colored coded: where light blue icons relate to information about the experiment, purple for general display options, green for optional view types, and dark blue for analysis view options.



I: Information	Show the metadata information such as the experiment type, processes, ranks, threads, hosts and other info.
U: Update	Update the display with performance information from the database file.
CL: Clear Auxiliary Information	If the user has chosen to view a time segment, a specific rank/process/thread, or a specific function's data, then when the CL icon is selected, it will clear those settings so that the next view is reset to show data with the original, initial settings.
D: Default	Show default performance results. First use View and Display Choice buttons to select whether data corresponds to functions, statements, or linked objects then click D-icon.
S, down arrow: Statement results per Function	Show performance results for the source statements for the selected function. Highlight a function before clicking this icon.
C+: Call Path Full Stacks	Show all call paths, including duplicates, in their entirety.
C+, down arrow: Call Path Full Stacks Per Function	Show all call paths for the selected function only. Highlight a function before clicking this icon. All call paths will be shown in their entirety.
HC: Hot Call Path	Show the call path in the application that took the most time.
B: Butterfly View	Show a butterfly view: the callers and callees of the selected function. Highlight a function before clicking this icon.
TS: Time Segment Selection	Show a portion of the performance data results in a selected time segment.
OV: Optional View Selection	Select which performance metrics to show in the new performance data report.
LB: Load Balance View	Show the load balance view: min, max and average performance values. Only used with threaded or multi-process applications.
CA: Comparative Analysis View	Show the result of a cluster analysis algorithm run against the threaded or multi-process performance analysis results. The purpose is to find outlying threads or processes and report groups of like performing threads, processes or ranks.
CC: Custom Comparison View	Allow the user to create custom views of performance analysis results.

GUI Manage Processes Panel
The Manage Processes panel allows focusing on a specific rank, process, or thread or to create process groups and view a group's corresponding data.

GUI General Panel Info
Each view has a set of panel manipulation icons to split the panel vertically or horizontally or remove the panel from the GUI. The icon toolbar found on far right of GUI panels is shown below.

CONDITIONAL DATA GATHERING
Gather performance data for code sections by bracketing your code with MPI. Pcontrol calls. MPI.Pcontrol (1) enables gathering. MPI-Pcontrol (0) disables. **OPENSS_ENABLE_MPI_PCONTROL** must be set.

BLUE GENE AND CRAY XT/XE USAGE
On the Cray-XT/XE platform, support of applications created with -dynamic is through the default workflow. That is, use the convenience scripts to gather the performance data and the GUI and CLI to view it. Please use the target runtime environment (module/dotkit) files while gathering and the host/frontend module/dotkit files to view the data. Dynamic support is not yet available on the Blue Gene platform. On platforms where dynamic shared library support is limited, the O|SS performance information gathering code must be statically linked into the user application. O|SS provides an **osslink** script to add into the application make files to help minimize the impact of the application link step.

Makefile Modification Example
Duplicate and edit this general makefile target:
$(TARGET): $(OBJ)
  $(F90) -o $@ $(FFLAGS) $(OBJ) $(LDFLAGS)
To create a pcsamp experiment: (changes in bold)
oss-pcsamp: $(OBJ)
  osslink -c pcsamp $(F90) -o $(TARGET)-**pcsamp** $(FFLAGS) $(OBJ) $(LDFLAGS)

Running ossutil to Create O|SS Database File
Set **OPENSS_RAWDATA_DIR** prior to application execution.
After the application completes use this command to create the O|SS database file:
ossutil <raw data directory path>.
After the above step, the database file may be viewed like any other O|SS database file.

For more information, please visit
http://www.openspeedshop.org/docs/users_guide/