DOD Open|SpeedShop (O|SS) Quick Start Guide

O|SS is an open source application performance analysis tool. It gathers performance data while your application runs and creates a sqlite3 database with performance data and symbol information from your application. O|SS has a graphical user interface (GUI) tool to view the data, as well as a "gdb-like" command line interface (CLI) tool. O|SS works on the application binary, so no recompilation is necessary. If your application is compiled with the –g compiler option, O|SS can give per statement and per loop performance information, otherwise O|SS reports per function information.

The older version of O|SS, called the offline version, writes raw performance data files to a shared file system and then reads those files at the point the application is finished running. OS|S is transitioning away from this approach to make the tool more scalable. The new version of O|SS, sends the raw performance data up a multi-cast network to tool daemons, where the performance data is reduced/filtered on its way to the tool client.

Note that all statistics/metrics gathered and displayed by O|SS can be mapped back to the application source code where they occurred.

Platforms running DOD O|SS

- Armstrong.navydsrc.hpc.mil (Cray XC30)
- Conrad.navydsrc.hpc.mil (Cray XC40)
- Copper.ors.hpc.mil (Cray XE6)
- Excalibur.arl.hpc.mil (Cray XC40)
- Garnet.erdc.hpc.mil (Cray XE6)
- Gordon.navydsrc.hpc.mil (Cray XC40)
- Haise.navydsrc.hpc.mil (IBM iDataPlex)
- Kilrain.navydsrc.hpc.mil (IBM iDataPlex)
- Lightning.afrl.hpc.mil (Cray XC30)
- Riptide.mhpcc.hpc.mil (IBM iDataPlex)
- Shepard.navydsrc.hpc.mil (Cray XC30)
- Spirit.afrl.hpc.mil (SGI ICE X)
- Thunder.afrl.hpc.mil (SGI ICE X)
- Topaz.erdc.hpc.mil (SGI ICE X)

DOD O|SS module file locations

The module file for the current version of Open|SpeedShop can be found here:

- **$PET_HOME/modules/openpeedshop** # new version, has new lighter weight I/O, MPI, and memory experiments
- **$PET_HOME/modules/openspeedshop-offline** # older version, but has base experiments

To see all module files in $PET_HOME/modules upon module avail, please use this command:
module use $PET_HOME/modules
After executing the "module use" command above, to load the O|SS module file, please use the following command:
module load openspeedshop

DOD Unique Requirements to use O|SS

- No need to recompile, though –g allows O|SS to show performance information at the statement and loop level. Profiling optimized code is not a problem.
- Cray specific requirements (not needed if using offline version)
  - PBS option: **-l ccm=1** is needed to make the PBS node and processor value environment variables available to O|SS. Examples of the PBS variables used by O|SS are:
    - BC_NODE_ALLOC=2
    - BC_CORES_PER_NODE=32
    - BC_MPI_TASKS_ALLOC=64
    - PBS_NODEFILE=/var/spool/PBS/aux/2880009.pbssrv1-wlm
  - The default version of O|SS on the Cray requires extra nodes for tool daemons, which are used to reduce/filter the performance data coming from the application on the way to the O|SS client tools. For this to work, **O|SS requires extra nodes (that are not part of your aprun count of nodes)** for these daemons at a rate of one extra node for each 1-100 nodes allocated. This is not required on the IBM iDataplex or SGI platforms, it is due to the aprun usage on the Cray platform.

Examples for Submitting Batch Jobs Using O|SS on DOD Platforms

To use O|SS in a batch job, please consider the following:

- Load the O|SS module file:
  - **module use $PET_HOME/modules**
  - **module load openspeedshop**
- However, you run your application normally, put that in quotes and prefix it with the O|SS convenience script corresponding to the performance data you would like to gather and view. For example:
  - osspcsamp: Program counter sampling (where my program is spending time)
  - ossusertime: Call path profiling (Show call paths through program, butterfly view, who calls who)
  - Please refer to the PERFORMANCE INFORMATION TYPES and CONVENIENCE SCRIPT DESCRIPTION sections below for more experiment definition and convenience script information.

Program counter experiment (pcsamp) usage examples could look like this:

- IBM iDataPlex: osspcsamp "mpirun –np 32 <full path to application>"
- Cray: osspcsamp "aprun –n 32 <full path to application>"
- SGI: osspcsamp "mpiexec_mpt –np 32 <full path to application>"

A full batch script example from a Cray platform (from shepard) follows:

```
#!/bin/bash
#PBS -q debug
#PBS -l select=3:ncpus=16:mpiprocs=16
#PBS -l walltime=00:12:00
#PBS -j oe
#PBS -l ccm=1
#PBS -N osshwcsamp
#PBS -A <account>
cd /p/home/galarowi/application_test_demos/bin
source ${MODULESHOME}/init/bash
module unload perftools
module use $PET_HOME/modules
module load openspeedshop
osshwcsamp "aprun -n 36 ./mpi-nbody-mpich-cray"
```

- **If on a Cray platform:**
  - **Use this PBS command: #PBS -l ccm=1**
  - **Allocate an extra node if your job uses from 1 to 100 nodes, and an additional 1 node for every additional 100 node grouping.**

Static Application Usage Information

The **cbtflink** command links the O|SS collectors and runtime libraries into the static executable and manages the setting the appropriate libraries based on the collector value that is one of the inputs to cbtflink. Here is a section of a makefile where nbody is linked normally and where cbtflink is used to link the program sampling collectors and runtimes of O|SS into the static nbody application. Please run the cbtflink --help for more details.

```
SHELL = /bin/sh
.SUFFIXES: .c .o
MPIcc = cc -DUSE_MPI=1
CC = $(MPIcc)
SOURCES = nbody-mpi.c
OBJECTS = $(SOURCES:.c=.o)
CFLAGS = -g -O3 -I. -static
LDFLAGS = -g -O3 -L /opt/cray/dmapp/7.0.1-1.0502.11080.8.74.gem/lib64 -ldmapp
.c.o: nbody-mpi.c
	@echo "Building $<"
	$(CC) -c $(CFLAGS) -o $@ $<
all: nbody-static nbody-pcsamp nbody-usertime
nbody-static: $(OBJECTS)
	@echo "Linking"
	$(CC) $(OBJECTS) $(LDFLAGS) -lm -o nbody-static
nbody-pcsamp: $(OBJECTS)
	@echo "Linking"
	cbtflink --mode mpi --mpitype mpich -v -c pcsamp $(CC) $(OBJECTS) $(LDFLAGS) -lm -o nbody-pcsamp
nbody-usertime: $(OBJECTS)
	@echo "Linking"
	cbtflink --mode mpi --mpitype mpich -v -c usertime $(CC) $(OBJECTS) $(LDFLAGS) -lm -o nbody-usertime
```

GENERAL O|SS INFORMATION

ACCESS INFORMATION

The O|SS Website: http://www.openspeedshop.org
O|SS Documentation, including the O|SS Users Guide: http://www.openspeedshop.org/documentation
CBTF Information: http://sourceforge.net/projects/cbtf

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-contact@openspeedshop.org. To register for access to forum questions and answers: oss-questions@openspeedshop.org

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 and butterfly view.
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.
iop*	Same functions as io are profiled in a light weight manner. Less overhead than io, iot.
iot	Similar to io, except that per event information is gathered, such as bytes moved, file names, etc.
mem*	Captures the time spent in and the number of times each memory function was called.
mpi	Captures the time spent in and the number of times each MPI function is called.
mpip*	Same functions as mpi are profiled in a light weight manner. Less overhead than mpi, mpit.
mpit	Like MPI but also 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.
pthreads*	Reports POSIX thread related performance information.
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.
cuda*	Traces all NVIDIA CUDA kernel executions and the data transfers between main memory and the GPU. Records the call sites, time spent, and data transfer sizes.

***CBTF Version only**

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 (for dynamically linked applications) 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. The CBTF version of O|SS adds these additional convenience scripts for the CBTF specific experiments: **ossiop, ossmem, osspthreads, ossmpip, and osscuda.** Note: If using offline version, 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

Running a convenience script with no arguments lists the accepted arguments. For the hwc scripts the accepted PAPI counters available are listed.
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"
Additional arguments for comparison metric:
Produces side-by-side comparison. 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_TOT_INS), sampling_rate is 100
<PAPI_event_list>: Comma separated PAPI event list
<sampling_rate>:Integer value sampling rate

ossio, ossiop*, ossiot: I/O Experiments

ossio "<command>< args>" [**default** | f_t_list]
Sequential job example: ossio "bonnie++"
Parallel job example:
ossio "mpirun –np 128 IOR"
Additional arguments:
default:trace all I/O functions
ossiop "<command>< args>" [**default** | f_t_list]
Sequential job example:
ossiop "bonnie++"
Parallel job example:
ossiop "mpirun –np 128 IOR"
Additional arguments:
default:trace all I/O functions
ossiot "<command>< args>" [**default** | f_t_list]
Sequential job example:
ossiot "bonnie++"
Parallel job example:
ossiot "mpirun –np 128 IOR"
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**

ossmem*: Memory Analysis Experiments

ossmem "<command><args>" [**default** | f_t_list]
Sequential job example:
ossmem "smg2000 –n 50 50 50"
Parallel job example:
ossmem "mpirun –np 128 smg2000 –n 50 50 50"
Additional arguments:
default: trace all memory functions
< f_t_list>: Comma-separated list of memory functions to trace, one or more of the following: **malloc, free, memalign, posix_mem align, calloc and realloc**

osspthreads*: POSIX Thread Analysis Experiments

osspthreads "<command><args>" [**default** | f_t_list]
Sequential job example:
osspthreads "smg2000 –n 50 50 50"
Parallel job example:
osspthreads "mpirun –np 128 smg2000 –n 50 50 50"
Additional arguments:
default: trace all POSIX thread functions
< f_t_list>: Comma-separated list of POSIX thread functions to trace, one or more of the following: **pthreads_create, pthreads_mutex_init, pthreads_mutex_destroy, pthreads_mutex_lock, pthreads_mutex_trylock, pthreads_mutex_unlock, pthreads_cond_init, pthreads_cond_destroy, pthreads_cond_signal, pthreads_cond_broadcast, pthreads_cond_wait, and pthreads_cond_timedwait**

ossmpi, ossmpip*, ossmpit, ossmpiotf: MPI Experiments

ossmpi "<mpirun><mpiargs><command><args>" [**default**|f_t__list]
Parrallel job expample: ossmpi "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:
ossmpip "<mpirun><mpiargs><command><args>" [**default**|f_t__list]
Parrallel job expample: ossmpip "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:
ossmpit "<mpirun><mpiargs><command><args>" [**default**|f_t__list]
Parrallel job expample: **ossmpit** "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:
ossmpiotf "<mpirun><mpiargs><command><args>" [**default**|f_t__list]
Parrallel job expample: **ossmpiotf** "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**
File I/O	**file_io**

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**

osscuda*: NVIDIA CUDA Experiment

osscuda "<command> < args>"
Sequential job example: **osscuda** "eigenvalues --matrix-size=4096"
Parallel job example: **osscuda** "mpirun -np 64 -npernode 1 lmp_linux -sf gpu < in.lj"
***CBTF Version only**

KEY ENVIRONMENT VARIABLES

OPENSS_RAWDATA_DIR (offline version only)

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_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.** For SGI MPT, use **mpich.**
OPENSS_MPI_IMPLEMENTATION="MPI impl. name"
Example: export **OPENSS_MPI_IMPLEMENTATION**=openmpi
In most cases, O|SS can auto-detect the MPI in use.

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_ENABLE_MPI_PCONTROL

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

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
list –v savedviews : list the views that have been saved for immediate redisplay
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**
expview –v statements <experiment-name><number> :shows the top time-consuming 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.

Show a chronological list of function calls for tracing experiments: iot,mpit,mem
expview -v trace

iot experiment: Display number of bytes transferred, showing function call start and stop time and return value (dependent on function call)
expview -vtrace -m start_time,stop_time,retval

mpit experiment: Show origin rank, source rank, and destination rank of all MPI functions (default) or for a list of MPI functions (-f function1, function2,...)
expview -vtrace -m start_time,stop_time,rankid,source,dest [-f comma separated function list]
For hybrid applications, rankid can be replaced with id which shows rank:thread

mem experiment: Find the call path for the largest allocation by using metric max_bytes in the calltree view:
expview -vcalltrees,fullstack -m max_bytes
Show only the top nn call paths showing the largest allocation, mem is the experiment name and nn is the number of paths
expview -vcalltrees,fullstack -m max_bytes mem[nn]

Tracing output can create extensive output. Directing the output in this fashion: **expview -v trace** [optional args] > output_file.txt may be useful.

For more information about the Command Line Interface commands please consult the O|SS Users Guide:
http://www.openspeedshop.org/documentation

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 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.

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